DNS for Massive-Scale Command and Control

Kui Xu *Member, IEEE*, Patrick Butler, Sudip Saha, Danfeng (Daphne) Yao *Member, IEEE*

Abstract: Attackers in particular botnet controllers use stealthy messaging systems to set up large-scale command and control. In order to systematically understand the potential capability of attackers, we investigate the feasibility of using domain name service (DNS) as a stealthy botnet command-and-control channel. We describe and quantitatively analyze several techniques that can be used to effectively hide malicious DNS activities both at the host and network levels.

Our experimental evaluation makes use of a two-month-long 4.6GB campus network dataset and 1 million domain names obtained from `alexa.com`. We conclude that the DNS-based stealthy command-and-control channel (in particular the codeword mode) can be very powerful for attackers, showing the need for further research by defenders in this direction. The statistical analysis of DNS payload as a countermeasure has practical limitations inhibiting its large-scale deployment.

Keywords: DNS tunneling, command and control, information theory.

I. Introduction

Botnet command and control (C&C) channel refers to the protocol used by bots and botmaster (i.e., botnet controller) to communicate to each other, e.g., for bots to receive new attack commands and updates from botmaster, or to submit stolen data. A C&C channel for a botnet needs to be reliable, redundant, non-centralized, and easily disguised as legitimate traffic. Many botnet operators used the Internet Relay Chat protocol (IRC) or HTTP servers to pass information. Botnet operators constantly explore new stealthy communication mechanisms to evade detection. HTTP-based command and control is difficult to distinguish from legitimate Web traffic. The feasibility of email as a stealthy botnet command and control protocol was studied by researchers in [29]. In this paper, we systematically investigate the feasibility of solely using Domain Name System (DNS) queries for botnet command and control. DNS provides a distributed infrastructure for storing, updating, and disseminating data that conveniently fits the need for a large-scale command and control system. The HTTP protocol is for the end-to-end communication between a client and a server. In comparison, DNS provides not only a means of communication between computers, but also systematic mechanisms for naming, locating, distributing, and caching resources with fault tolerance. These features of DNS may be utilized to fulfill a more effective command-and-control system than what HTTP servers may provide.

The decentralized nature of domain name systems (DNS) with a series of redundant servers potentially provides an effective channel for covert communication of a large distributed system, including botnets. To play the devil's advocate, we focus on systematically analyzing the feasibility of a pure DNS-based command-and-control [1]. Such a study has never been reported in the literature. Our C&C system is compatible with existing DNS infrastructure without enlisting any Web or special-purpose servers. The DNS channel is aided by being a high traffic channel such that data can be easily hidden. As virtually anyone can create and register their own domain names (if available) and DNS servers, it is a system that can be easily infiltrated by hackers and botnet operators.

DNS tunneling is a technique known for transmitting arbitrary data via DNS protocol, e.g., DNScat and DeNiSe. One application of DNS tunneling is to bypass firewalls, as both inbound and outbound DNS connections are usually allowed by organizational firewall rules. Because DNS is often overlooked in current security measures, it offers a command-and-control channel that is unimpeded. Because nearly all traffic requires DNS to translate domain names to IP addresses and back, simple firewall rules cannot easily be created without harming legitimate traffic. Recently, Dietrich and colleagues reported `Feederbot` that used DNS as a communication channel for C&C traffic [5]. However, `Feederbot` fails

The preliminary version of this work appeared in the 9th International Conference on Applied Cryptography and Network Security (ACNS '11), Lecture Notes in Computer Science 6715, pages 238-254 [2]. This work has been supported in part by National Science Foundation grants CNS-0831186, CAREER CNS-0953638, ARO grant STIR-450080, and NSF S2ERC.

Yao is the corresponding author. Her email address is danfeng@cs.vt.edu and mailing address is 2202 Kraft Dr. KWII, Virginia Tech, Blacksburg VA 24060. Xu, Butler, and Saha are Ph.D. candidates at VT Department of Computer Science.

[1]Other C&C protocols (e.g., HTTP) also involve DNS queries for name translation, but they do not use DNS for command and control.

to utilize any distributed storage and query mechanisms offered by DNS. This botnet simply tunnels its command and control traffic by sending it in DNS format for the end-to-end communication between bots and the bot master. The domains used by them are not registered and cannot be resolved.

While using DNS tunneling for command and control has been observed [13], it was still unclear how effective and feasible to use DNS to maintain stealthy large botnets. Specifically, three items to consider in order to evade detection for attackers are

1) *Query activities.* When and how frequent do bots issue DNS queries to pull updates from or submit data to the bot master? How to modify the victim's operating system to implement automatic query strategies?
2) *Domain names.* What domain names to use for communication and how to synchronize the generation of new domain names between bots and the bot master?
3) *DNS payload.* How to evade the detection through deep packet inspection by defenders on DNS payload?

Our work systematic addresses these questions using system engineering, networking, and data mining techniques. Our technical contributions are summarized as follows.

- We describe techniques for hiding query activities, including *i) piggybacking query strategy* – a bot blends its (outbound) DNS queries with legitimate DNS queries and *ii) exponentially distributed query strategy* – a bot probabilistically distributes DNS queries so that inter-arrival times follow an exponential distribution. We demonstrate the ability for a bot to send piggybacking DNS traffic through traffic sniffing in Linux.
- For automatic domain flux, where the domain names used for communications in the botnet are changed frequently and in a synchronized fashion across all bots and their controllers, we describe a practical automatic domain flux method with Markov chain, and experimentally evaluate it with 1 million domain names from `alexa.com`.
- Statistical methods can be used by defenders to detect anomalies in the content of DNS packets, through comparing the probability distributions of normal DNS traffic and tunneling traffic. We evaluate these methods as countermeasures and point out the practical limitations that hinder the large-scale deployment by defenders.

We perform comprehensive experiments to evaluate the behaviors of proposed query strategies in terms of how quickly new commands are disseminated to a large number of bots. Our analysis utilizes a 4.6GB two-month-long wireless network trace obtained from an organization. We conclude that the DNS-based botnet command-and-control channel is feasible, powerful, and difficult to detect and block.

Organization We describe the basic DNS tunneling mechanisms in Section II. We present new strategies for improving the stealth of DNS-based command and control in Section III. Automatic domain flux is discussed in Section IV. We describe a countermeasure that requires examining the content of DNS packets and performing statistical analysis in Section V. Related work is given in Section VI. Conclusions and open problems are given in Section VII.

II. Communication Modes

In this section, we describe protocols that pass messages over the DNS between distributed entities, and illustrate the ease of setting up large-scale command-and-control via DNS. We describe two forms of communication modes: *codeword* mode and *tunneled* mode. *Codeword communication* allows one-way communication from botmaster to a bot client, which is suitable for issuing attack commands. *Tunneled communication* allows for the transmitting of arbitrary data in both directions between bot and botmaster, which may be used for both issuing commands and collecting stolen data. The former only requires the ability to set a particular domain name response, this could be done via any free DNS service, while the latter requires setting up an authoritative domain server.

The controller of the botnet first needs to create a domain or subdomain, which is administered from a special DNS server. This DNS server waits for special name lookups, which it then translates into incoming data. The DNS server then responds with the appropriate data using the agreed-upon semantics. We assume that the botnet controller (i.e., botmaster) has access to the authoritative domain name server for some domains or sub-domains. Bots across the Internet frequently receive commands and updates from a botmaster and launch attacks accordingly, as well as submit stolen data to the botmaster. We give brief background information on DNS records.

DNS Resources Records The DNS system allows a name server administrator to associate different types of data with either a fully qualified domain name or an IP address. To send a message to a bot, an adversary can store data in any one of these types of records.

- `A` record specifies an IP address for a given host name.
- `CNAME` and `MX` records can point to textual data representing the alias or mailing host of a particular host name.
- `TXT` records are designed to store arbitrary textual data up to 255 characters.
- `EDNS0` record allows storing up to a 1280 byte payload [24]. EDNS0 was introduced in RFC261 in order to extend the DNS protocol. When a capable server or client encounters this field, it can decode the packets, allowing several improvements to the basic DNS protocol. These features include larger UDP packet size, a list of attribute value pairs, and several extra bytes for commonly used flags.

A. Codeword Mode

The *codeword mode* is a stealthy communication mechanism. It requires a botnet operator to decide upon a set of agreed upon codewords *a priori*. Each codeword represents a specific type of commands or attacks. The codeword appears in the DNS query as an innocent hostname, for example `codeword.domain.com`. This hostname may be stored as any type of record (e.g. `A`, `MX`, `CNAME`). A request for an `A` or `CNAME` record tends to be the most common and therefore a preference should be given to these records types so that queries would appear most like legitimate traffic. The client queries `codeword.domain.com`, and waits for a particular value in the server's response. Upon receiving the query, the DNS server (controlled by the botnet operator) returns the pre-set response that contains command information. If the codeword corresponds to denial-of-service (DoS) attacks, then the response may represent a target of DoS attacks. If the codeword corresponds to update, the client may contact the IP address returned for updated code or other instructions.

It is important to note that the codeword can be chosen *arbitrarily* and does not need to correspond to a specific host or service. The codeword method allows a stealthy *one-way* commanding system. It can effectively evade detection approaches based on non-conforming packet sizes [13], i.e., DNS packets whose sizes are outside the range of [28, 300] bytes. Codewords may be arbitrarily generated, or may be common service names such as `www`, `mail`, or `ftp`. In the latter case, packet statistics cannot be performed to find anomalies.

B. Tunneled Mode

The purpose of tunneled mode is to allow the *two-way* transfer of arbitrary binary data between a server and a client. This mode is referred to as tunneled mode, as one can tunnel streaming data over this DNS communication method.

- Upstream: Ask CNAME for:
  `NBSWY3DPFQQHO33SNRSA000.domain.com`
- Downstream: CNAME points to:
  `NBUSYIDCN5ZXG000.domain.com`
  `3600`
  `CNAME`
  `NBSWY3DPFQQHO33SNRSA000.domain.com`

Fig. 1. Example data packets sent to and from a server in tunneled mode: To server: "hello, world". From server: "hi, boss". In this example, the domain server for domain.com is the malicious server and the response has one hour TTL.

- *Upstream communication* is for a client to submit data to a (malicious) domain server. The client submits the data as a `CNAME` query by *i)* encoding the data using a base32 encoding, *ii)* using the encoded string to construct a host name, and *iii)* send a `CNAME` DNS query. An example is shown in Figure 1.
- *Downstream communication* is for the server to issue commands to clients. Upon receiving the above query from the client on a hostname h, the server *i)* encodes the response as base32 data, and *ii)* constructs and returns a `CNAME` record for h. An example is shown in Figure 1.

To prevent DNS caching from disrupting the communications, the server may set a short *time-to-live* (TTL). This tunneling method gives an operator the most options after implementation as the data stream can be arbitrary. Because of the arbitrary payload, the distribution of packet bytes may differ significantly from conventionally DNS payload. We perform more analysis in Section V.

DNS protocol does not allow the server to initiate a connection with the client, the client needs to continually *pull* updates from the server. Both the tunneled mode and codeword mode require clients to frequently pull updates from name servers by querying the corresponding botnet's domain. Straightforward querying patterns are easy to detect (e.g., periodically sending DNS queries) and susceptible to simple aggregate analysis, such as counting DNS queries for each unique domains and identifying domains with abnormally large query volume at the host, local area network, or internet service provider levels. We analyze several simple-yet-effective methods for bots to hide their DNS traffic in the next section.

Were DNSSEC to become widespread it would provide both an advantage to, as well as a disadvantage to, a potential attacker. In the attacker's favor is the increased

usage of DNS over TCP and the extra packet size and reliability provided. In return, the attacker would lose easy access to many protected name servers that might have otherwise been compromised. However, given that an attacker can legally purchase their own domains and that some DNS operators place their signing keys on the DNS itself, it is unclear how much protection DNSSEC would offer for stopping DNS based C&C channels.

III. Query Strategies and Quantitative Evaluation

In this section, we play the devil's advocate and describe and experimentally evaluate new techniques for hiding DNS query activities on a host, in order to defeat anomaly detection that targets abnormal temporal patterns. The proposed strategies are useful for both the tunneling and codeword modes. We quantitatively analyze the detection countermeasures in Section V.

A. Exponentially Distributed Query and Piggybacking Query

We describe an exponentially distributed query strategy and a piggybacking query strategy, both can be used to hide bot activities while communicating with a botmaster in a timely fashion. In our experiments in Section III-B, we provide an experimental evaluation on both query methods.

Exponentially distributed query strategy The Poisson process is previously believed to be a suitable model for representing stochastic processes where arrivals are independent on each other, i.e., memoryless. In [23], client-side DNS request arrivals are modeled by Poisson processes with exponential random variables with different rates λ (e.g., 2.63 queries/hour for `www.google.com` and 0.78 queries/hour for `www.cnn.com`). In our exponentially distributed query strategy, a bot probabilistically distributes DNS queries so that their intervals follow an exponential distribution with a parameterized arrival rate λ_b. Because of the memoryless feature of the model, the bot does not need to store the previous communication history. One simple way to implement this query strategy is as follows.

1) The bot sends a DNS query;
2) It computes an interval t by drawing from an exponential distribution with parameter λ_b (hardcoded or dynamically generated);
3) The bot sleeps for t, and repeats from Step 1.

There is a trade-off between being stealthy and communication efficiency. We study a bot's strategy in finding an optimal λ_b in Section III-B, given the host-wide DNS query rates.

Piggybacking query strategy Many (legitimate) websites contain content from multiple independent domains due to third-party content delivery, advertisements, or content mashup. Therefore, multiple DNS queries are usually issued by a host with temporal proximity. The composition of domains is usually dynamic. The piggybacking query strategy leverages this fact. A bot passively listens on the host's DNS traffic or name-translation related function calls and sends DNS queries when legitimate DNS queries are being made. Thus, the bot's query is blended among a group of legitimate DNS queries.

In the piggybacking query strategy, a bot's communication with the controller is constrained by the host's activities. Therefore, we focus on analyzing its timeliness, in terms of the dissemination efficiency of new command and data. We define *time-to-communicate* (TTC), *minimum TTC* , and *maximum TTC*. Minimum TTC is a threshold aiming to prevent a bot from sending queries too frequently, whereas maximum TTC is a threshold for keeping the liveliness of the communication between the bot and the bot master in case of an inactive host.

Definition 1: Time-to-communicate (TTC) is defined as the time interval between two network connections (DNS queries in our setting) of a bot for retrieving information from or submitting data to the botmaster server.

Definition 2: Minimum TTC (e.g., *min-TTC*) is the lower bound of time-to-communicate, whereas maximum TTC (e.g., *max-TTC*) is the upper bound of time-to-communicate. Let t and t' be the timestamps of two adjacent DNS queries that the bot sends. Then, t and t' need to satisfy the following constraints.

$$\textit{min-TTC} \leq \mid t - t' \mid \leq \textit{max-TTC}$$

In other words, a bot does not send any DNS query, if the bot's previous DNS query was sent within the minimum TTC. At time t, a bot needs to send a DNS query to check for update from the bot master, if the interval between t and the time when the bot sends the previous DNS query equals the maximum TTC. These two parameters put constraints on the bot's query frequency.

In this piggybacking mode, bots need to know when a legitimate DNS query is made. There are different approaches for obtaining this information.

1) *Traffic sniffing.* Because DNS server runs on port 53, an outgoing packet from the host (client) to a destination IP on port 53 is an indication of an outbound DNS request. The bot may sniff the network traffic to identify the right moment to

issue its DNS queries. The bot may either directly program with pcap library or call existing tools such as `iftop` (in Linux), which uses pcap. This approach gives a cross-platform and system-wide monitoring solution, which covers the DNS queries from any application.

We have implemented this approach in Linux. With the pcap library, the traffic sniffing is relatively straightforward to realize with a small piece of code. We attach the code to host machine's network devices, which usually requires root privilege. In the code, function `pcap_findalldevs()` returns all available network devices, `pcap_open_live()` opens a network device, and `pcap_loop()` performs the traffic sniffing. The program needs root privilege to run. It filters all TCP and UDP packets, both of which are possible protocols for carrying DNS requests, and reports the timestamps whenever a DNS request is identified. It also distinguishes the bot's own DNS query from legitimate ones, so that piggybacking is only performed on legitimate DNS queries. This prototype demonstrates a feasible way for our proposed piggybacking query strategy.

2) *Process hooking.* Another approach is to hook DNS-related function. In Linux, bots can watch the calls for DNS-related APIs such as `gethostbyname()` function in `libbind` library. `gethostbyname` looks up all IP addresses associated with a host name and is implemented in the resolver library. One way of hooking into the API function is for bots to register a `.so` file (shared library) to the `LD_PRELOAD` environmental variable, which may or may not require root. In the registered `.so` file, the target API function is replaced by the attacker's version which can notify the bot whenever this function is called. Similarly, in Windows [11] gives a solution, Detour, which has the ability to perform function interception and rewriting. Compared with traffic sniffing approach above, this method is less desirable, as it is process-specific. The hooking is done in the process memory. The original shared library is not changed, which is different from approach 3 below.
3) *Rogue library.* Alternatively, attackers may replace the network related shared libraries in the OS with an instrumented version (requiring root), so that the shared library is changed permanently on the hard disk. The rogue library is loaded by any application. It reports every DNS request to the bot. This approach is system wide, so the change is once-and-for-all.

B. Experimental Evaluation

The goal of this evaluation is to understand how effective the aforementioned stealthy query strategies are. Specifically, how soon botmaster disseminates commands to all or most bots; and how soon stolen data is harvested bots by botmaster? We do not allow bots to submit DNS queries at will, in order to avoid detection. We only allow bots to either piggyback their queries with legitimate DNS queries from the victim host, or follow a special inter-query distribution.

Our implementation uses the Python Modular DNS Server (`pymds`) and a specially designed plugin to respond to DNS requests. PyMDS implements the full DNS protocol while allowing the user to implement a programmatic and dynamic backend to generate the DNS records returned. Instead of returning records from a static file, PyMDS allowed for the decoding of codewords and the creation of appropriate responses.

To evaluate the *piggyback query strategy*, our dataset is a two-month-long network trace obtained from a university and collected with the IPAudit tool. The trace covered users from three departments and several research and education centers. (All machines were connected to the Internet wirelessly, i.e., there was no wired connection.) The raw dataset is 4.6GB. We identify and analyze the DNS traffic on port 53 of remote destinations. For data preprocessing, we select the most active 200 users from the our dataset by partitioning users by their (static) MAC address and sorting users by their traffic volume. We simulate the piggyback DNS-query strategy by having a bot send outbound communication whenever a host issues a UDP datagram on remote host port 53. Figure 2 shows the percentage of packets whose TTC is above the given minimum TTC in a 10-hour-span. Three minimum TTC values are analyzed: 1, 30, and 60 minutes.

Results in Figure 2 show that the piggybacking query strategy is quite effective – at least 80% of bots are able to communicate with the botmaster within 2 hours. Clearly, there is a trade-off between minimum TTC and how soon bots communicate with the headquarter. For an active botnet where commands may change every day, minimum TTC may be set to 60 minutes.

Piggybacking case studies We select four hosts from our dataset to simulate the piggybacking behaviors on them and evaluate the mean time-to-communicate. The four hosts are the first, 50-th, 100-th, and 200-th most active hosts according to their total traffic volume during

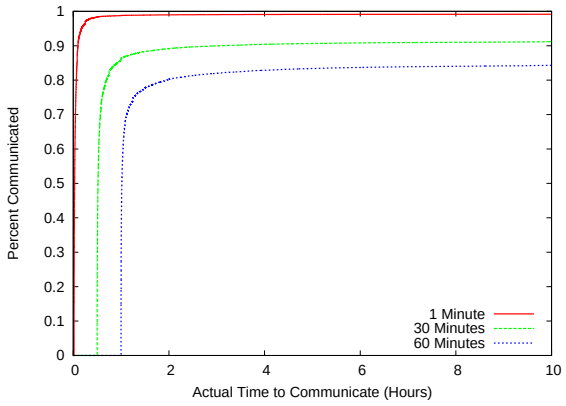


Fig. 2. Cumulative density function on the percentage of bots that have successfully sent at least one DNS query by piggybacking after a time period (X-axis). Each line corresponds to a different minimum TTC (1, 30, and 60 minutes). The figure shows a 10-hour span.

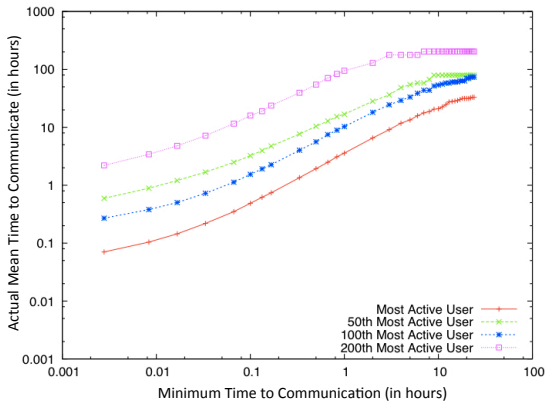


Fig. 3. Case studies on the time-to-communicate for four hosts with varying active traffic volume, given minimum TTC values shown on X-axis.

the 2-month period. Figure 3 plots how the mean TTC changes with the minimum TTC in a piggybacking query strategy, with the maximum TTC set to infinity. The results show that bot's communication efficiency is higher on more active hosts as expected. A maximum TTC (e.g., 48 hours) may be set to ensure periodic communication of the bot. Mean time-to-communicate grows with minimum TTC and is almost always greater than minimum TTC. The relationships for the studied are shown in Figure 4.

For the *exponentially distributed query strategy*, our goal is to identify an optimal range for λ_b – bot's query arrival rate on a host. We analyze the difference between two distributions: *i)* host-wide inter-arrival time for regular DNS queries with arrival rate λ, and *ii)* inter-arrival time for the bot-mixed DNS queries, i.e., new arrival rate $\lambda + \lambda_b$, where λ_b is the bot's query rate.

We use Kolmogorov-Smirnov (KS) test, which is

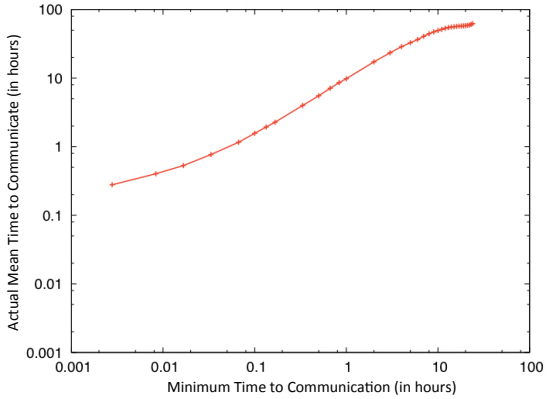


Fig. 4. Mean TTC vs. minimum TTC for 200 hosts.

suitable for comparing unbinned distributions that are functions of a *single* independent variable as in our case [8]. In our KS test, a higher p value ([0, 1]) represents a higher resemblance between the normal and the bot-mixed distributions. To simulate the Poisson process, we use two estimated λ values – high arrival rate of 131.5 queries/hour and low arrival rate of 39 queries/hour – based on results from [23].

Intuitively, a higher legitimate DNS query rate makes it easier for a bot to blend in its traffic. Our results in Figure 5 and Figure 6 confirm the intuition. High rate $\lambda = 131.5$ is shown in Figure 5, and low rate $\lambda = 39$ in Figure 6, where each line represents a different amount of data collected: 10, 24, 48, and 100 hours. X-axis is the varying λ_b value. The horizontal line represents a 5% cut-off threshold that may be used for detecting anomalies.

Our results show that longer traces make it easier for defenders to discern data. Higher λ tolerates higher λ_b, allowing bots to communicate more often. Given a p value threshold, the KS test can be used to find a suitable λ_b. The experiments show that even when data is collected for long periods of time, such as 100 hours, it can be difficult to detect bots using a small λ_b. In the case of less active hosts, λ_b can be come undetectable at 4 requests per hour, and with more active hosts λ_b can be as high as 10 requests per hour.

For the defender to run the KS test on DNS logs for anomaly detection as shown above, one may need to collect logs from the suspicious host for a substantial amount of time (e.g., 100 hours). This procedure may be performed periodically or as needed. The logs may be discarded after the test. To save space, the information to be logged by the defender can be represented as a vector of timestamps when DNS queries are observed.

Summary The experiments suggest that both the pig-

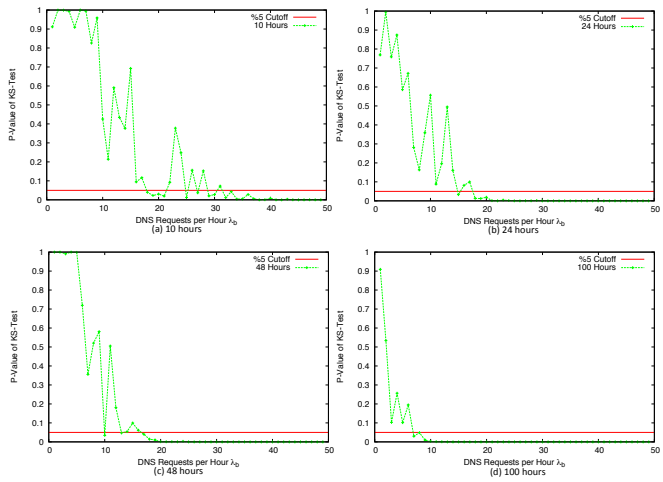


Fig. 5. KS test results between queries with the arrival rate of $\lambda = 131.5$ queries/hour and bot-mixed queries of $\lambda + \lambda_b$ (X-axis). Four runs of simulation lasting for 10, 24, 48, and 100 hours are shown in (a), (b), (c), and (d), respectively.

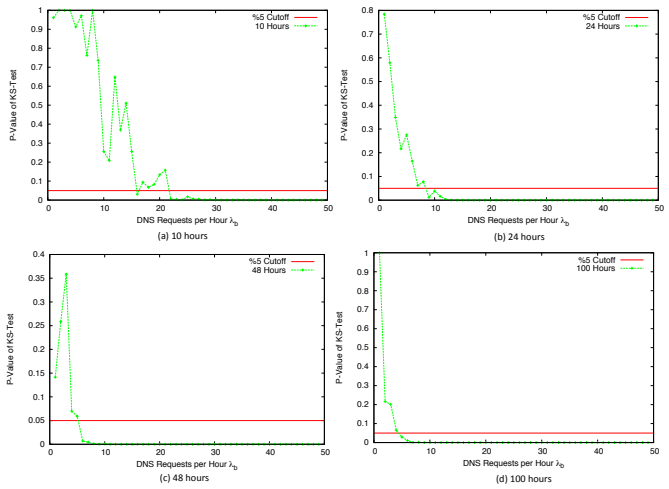


Fig. 6. KS test results between queries with the arrival rate of $\lambda = 39$ queries/hour and bot-mixed queries with $\lambda + \lambda_b$ (X-axis). Four runs of simulation lasting for 10, 24, 48, and 100 hours are shown in (a), (b), (c), and (d), respectively.

gybacking and exponentially-distributed query strategies can be effective in allowing the majority of bots to communicate in a reasonable time frame without being detected. The exponentially-distributed query strategy gives the bot slightly more control over when to query. On the other hand, the optimal query rate λ_b depends on the host-wide query rate, which may change.

IV. Domain Generation With Markov Model

Long-lived domain names are easy to manage and cheaper to maintain, however, they are susceptible to aggregate analysis. Domain flux refers to using short-lived domain names in botnet C&C [20]. Domain flux typically requires bots and botnet controller to independently derive new domain names periodically. In this section, we play the devil's advocate and describe methods that a botnet controller may use to generate domain names for domain flux.

To have short-lived domains, a static approach is to have a botmaster generate an ordered list of domain names and pack the list in malware code for bot to look up, which is similar to the use of a one-time password pad for authentication. However, there are two disadvantages for this method: large storage and high code-homogeneity – long lists of domain names shared by all bot code making the code susceptible to signature-based malware detection. Having the botmaster sends to all bots the next domain name during the current epoch is not an acceptable solution, because a communication failure may prevent the bots from learning the correct name for the next epoch.

One simple approach is for bots and their controller to independently compute the hash value of an incremental counter and a shared secret at each epoch, i.e., $H(counter \| secret)$, where H is a one-way collision-resistant hash function. For example, the bot and botmaster may share a secret, which is concatenated with the current epoch, e.g., current date, to derive a hash value for short-lived domain names, e.g., $H(s$, *current-date)*. The clocks on all bots and the bot master have to be synchronized, especially when the epoch is short. This timestamp-based domain-generation algorithm was observed in several botnets such as Torpig [31] and Conficker worm [21], [22]. An alternative domain-generation mechanism is based on the Lamport one-time password authentication scheme [15]. Technical issues associated with hash-based domain names such as domain length, collusion and unavailability of domain names can be easily dealt with and are omitted.

We describe a useful technique that a botnet controller may use for automatically generating realistic-looking domain names, which is more sophisticated than the previous hash-chain based method. We utilize techniques for automatic text generation, such as the ones used by SCIgen [25].

In automatic text generation or in applications such as finding genes similar to a pool of genes, Markov Chain models are often successfully used. Markov chain model makes use of the probabilities with which one entity follows another. We use the same Markov chain model to model letter transitions in domain names. We use 1 million top visited domain names from `alexa.com` collected on May 25, 2010 to generate the transition

probabilities and thereby the Markov chain of 37 characters (26 alphabets, 10 digits, 1 hyphen - that occur in domain names). To make the length of the generated domain names to be consistent with legitimate ones (in terms of their probability distributions), we take the length from the legitimate length distribution. The length distribution of 1 million domain name is shown in Figure 7.

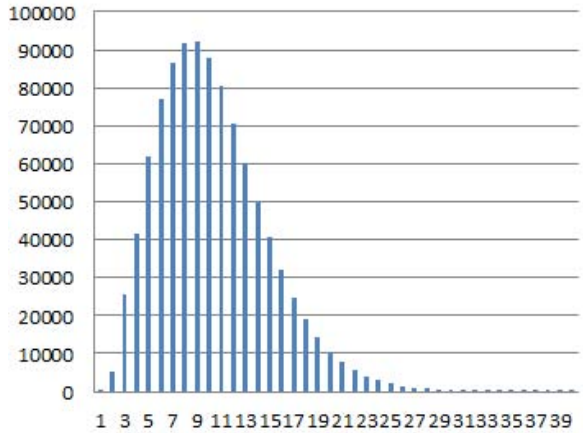


Fig. 7. Length distribution of 1 million legitimate domain names, domain index on x axis, number of domains at y axis.

Given the Markov chain (MC) models and the length distribution, new domain names can be generated. We show some of them along with some hash based domain names in Table I. Hash based names look random, and have digits. MC-based names have English word patterns in the names.

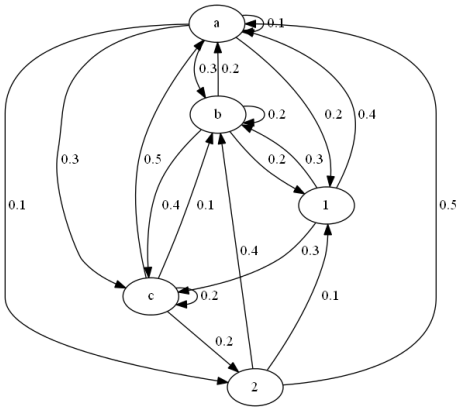


Fig. 8. A Markov chain example for a simple model with five characters in the alphabet - a,b,c,1,2.

Figure IV shows a simple Markov chain(MC) for only five characters – a, b, c, 1, and 2. Any generated domain name of a given length corresponds to a random walk of that length in this chain.

The domain generation has to be independent, consistent, and synchronized. These can be achieved in this

#	Markov chain based	Hash based
1	x-airnarienghy	93mp3com
2	islegawhosh	qqdgdcom
3	cnewallonderc	58qqcomcn
4	yasesaug	juegos666com
5	veadgeupot	c8048com
6	iga-ngakhodo	yishu666info
7	yarchma	xinwen666com
8	ialeparteb	z6zzcom
9	beatersashinehu	nr33com
11	iyerishopes	2107665188
10	nollanguilcw9xb	wwwli

TABLE I

COMPARISON OF DOMAIN NAMES GENERATED BY HASH FUNCTION (RIGHT) AND MARKOV CHAIN (LEFT).

MC-based approach by using the same random number generator and the same initial seed. Generators like linear congruential generator may be used in this respect. The botmaster has to communicate with the bots the following items:

- the Markov probabilities for state transitions,
- parameters for random number generator,
- parameters for the initial seed and timing interval.

We compute the cosine similarity between MC-based domains and 5,000 legitimate domains. We also apply the same metric to new legitimate domains with respect to the same 5,000 domains, in order to compare the quality of MC-based domain names. The cosine similarity is defined in the following manner. Each domain name is represented as a vector $X = [X_1 X_2 ... X_{37}]^T$ of 37 features; the feature set includes 26 letters in the alphabet, 10 digits and the character '-'. The cosine similarity S between two domain names A and B, represented as vectors, is defined as,

$$S(A,B) = \frac{\sum_{i=1}^{37} A_i \times B_i}{\sqrt{\sum_{i=1}^{37} A_i^2} \times \sqrt{\sum_{i=1}^{37} B_i^2}} \qquad (1)$$

To compute the similarity of a particular domain name, $DomX$ to the available 5000 legitimate domain names $DomL_1, DomL_2, ..., DomL_{5000}$, we take the average of the individual cosine similarities between $DomX$ and $DomL_i$. We compare the quality of a MC based domain name and a legitimate domain name with respect to their similarity to the list of 5000 most visited domain names. The results are shown in Figure 9. We observe that the MC-based domains and legitimate domains have comparable similarity values.

We also measure the cosine similarity for hash-based domain names. Figure 10 shows the distinction between Markov chain generated names and hash generated

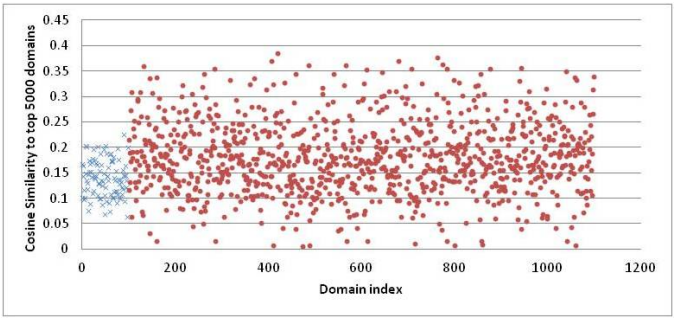


Fig. 9. Cosine similarities between the 5,000 most visited domains and 100 Markov chain generated domains (blue checks) vs. 1,000 legitimate domains (red dots), respectively.

names. Clearly, MC generated names have a higher similarity to legitimate domain names than their hash alternatives.

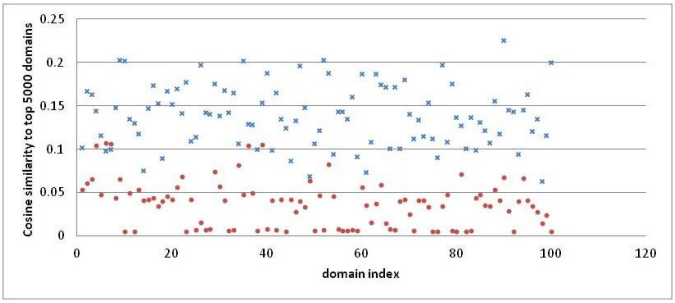


Fig. 10. Cosine Similarity to 5,000 most visited domains for 100 Markov chain generated domains (blue checks) vs. 100 hash generated domains (red dots), respectively.

Botnets have been observed to use subdirectories for communication, e.g., `www.example.com/products`, `www.example.com/home`. However, for a DNS-tunneling based channel, subdirectory approach does not apply, as the botmaster does not run a Web server and the communication is based solely on domain name systems. Consider that botnets often use third-level domains instead of subdirectories, Dagon proposed to use the ratio between second-level domains (SLD) and third-level domains (3LD) to identify botnet traffic [3].

When using short-lived domain names, the botmaster needs to consider whether to use the generated string for second-level or third-level domain name. Using as 3LD makes it easy for botmaster to update the DNS records of a domain, which is static and long-lived. However, it may be detected by the ratio-based method mentioned above. Using short-lived SLDs (e.g., `1a735009.com`, `635b5790.net`, `9e30f817.org`) would likely render the ratio-based detection ineffective. Its downside might be that the SLD may not always be available, although automatically generated domains may be less likely to collide with existing domains.

From the attacker's point view, while generating realistic looking domain names, it is reasonable to generate only the SLD's and pick some TLD suitably or randomly. Compared to millions of SLD's, the list of TLD's is very small. However, the SLD and TLD names are not entirely independent; the character-pair distribution of the SLD's may vary, particularly when the TLD includes country codes (ccTLD). For example, there are many words (in pinyin [2]) from Chinese language in SLD's with `.cn` TLD; the character-pair probabilistic distribution of these SLD's should be different from the general character-pair distribution of SLD's. The attacker can make use of this dependency to generate more realistic looking domain names. E.g., the attacker can construct an MC based on Chinese character-pair distribution only to generate Chinese word based SLD's for `.cn` TLD, or an MC based on Arabic character-pair distribution to generate SLD's for `.om` TLD (for Oman).

Domain name registration. We expect botnet controllers to create and register the domains to be used, which allows them to take advantage of the existing and decentralized DNS infrastructure for scalability, fault tolerance, and storage [3]. In order to register a new second level domain associated with a top level domain (such as `.com`), a registrant (e.g., a botnet controller) needs to apply to a domain name registrar and pay the (annual) fees. The botnet controllers may specify their own IP addresses of authoritative name servers to host the domain's resource records. This approach gives them the flexibility in managing the command and control operations and customizing the DNS traffic. There is monetary cost associated with registering a new second level domain name. A botnet controller may create new third level domains (3LD) with a fixed SLD. Sites like `dyndns.org` make it easy to create lots of domains of the form `x.dyndns.org`. For defenders, it is difficult to block SLD domains such as `dyndns.org` because legitimate sites use them as well.

Summary on domain flux Markov-chain based domain names achieve an average cosine similarity of 0.138 which is very close to the similarity of legitimate names evaluated. This value is much higher than the similarity of 0.036 achieved by hash generated domains under the same experimental condition. This observation indicates that it is possible to generate near-realistic looking domain names that closely follow legitimate byte

[2]Pinyin is for entering Chinese characters into computers.

[3]In contrast, domains used by Feederbot cannot be resolved by public resolvers [5], as they were not registered.

distribution.

V. Deep Packet Inspection

In this section, we describe and experimentally evaluate a countermeasure against DNS-based stealthy messaging systems that requires deep packet inspection and statistical analysis. Deep packet inspection examines packet payload beyond the packet header. Specifically, we quantitatively analyze the probability distributions of (bot's) DNS-packet content.

We describe and evaluate a concrete countermeasure against stealthy DNS channels through statistically analyzing traffic content. To compute the byte distribution in normal and tunneling traces, we use the Jensen-Shannon (JS) Divergence D_{JS}, which is a common metric for quantifying the difference between two probability distributions P and Q, and is a commutative version of Kullback-Leibler divergence of Q from P. A lower D_{JS} value means a higher similarity in two probability distributions. The JS Divergence is particularly suited in situations where the random variable is discretized.

$$M = \frac{1}{2}(P+Q) \quad (2)$$

$$D_{KL}(P,Q) = \sum_{i=0}^{n} p_i \log \frac{p_i}{q_i} \quad (3)$$

$$D_{JS} = \frac{1}{2}(D_{KL}(P,M) + D_{KL}(Q,M)) \quad (4)$$

We experimentally compare DNS packet traces recorded on a host, specifically, on how different tunneling packets are from legitimate ones in terms of the probability distribution of content. Such probability measures may be taken on a per-host or per subnet basis, however since a filter based on these methods must only keep an probability distribution of the bytes in a packet, no identifying information can be inferred. In this way privacy concerns can be kept at a minimum.

In the following tests, three normal DNS traces were recorded and one tunneling DNS trace via tunneled mode was recorded. Each trace corresponds to an hour-long network activities on a host. Sizes of our traces are 862KB for the tunneling trace, 823KB for normal trace 1, 699KB for normal trace 2, and 153KB for normal trace 3. In addition, the tunnel trace contained 191 `A` queries and 1433 `TXT` queries, while the normal trace 1 contained 1750 `A` queries and no `TXT` queries, and normal trace 2 contained 2417 `A` queries and no `TXT` queries. Tunneling trace contains encrypted Secure Shell (SSH) activities, i.e., SSH traffic through DNS tunneling. DNS or DNSSEC does not provide query confidentiality, i.e., the DNS query payload is not required to be encrypted.

When the entire packet including header is analyzed, we find that the divergence of normal traces (normal 1 and normal 3) is large (not shown). To get a more stable comparison, we drop the UDP headers and only observe the DNS payload. Figure 11 shows how the Jensen-Shannon divergence changes as more tunneling message carrying packets are mixed in. The X-axis is the ratio of tunnel trace to normal trace 1.

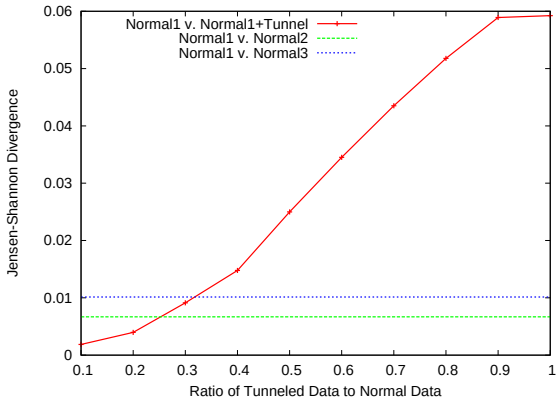


Fig. 11. Divergence computed from the payload of UDP datagrams. Horizontal lines represent the divergence of normal streams. The red line is the divergence of mixed traces.

Our results show that in our experiments a divergence threshold of 0.015 can sufficiently distinguish normal traces from mixed traces containing more than 30% bot queries. These results indicate that analyzing DNS payload as a countermeasure is more effective than analyzing the entire DNS datagram with JS divergence. While DNSSEC does not specifically require DNS over TCP, it is commonplace because DNSSEC adds to the length of DNS packets and many deployed systems cannot handle these larger UDP packets. Therefore the data collection must be done with consideration to the TCP protocol. The TCP protocol has a number of bytes that iterate and change over each packet, so the defender must take extra care to only analyze the DNS payload and not the TCP headers as well as the key authentication headers which may skew the distribution of the data.

However, there are practical issues and constraints when executing the statistical detection by defenders in large scale, besides the obvious storage and computation overheads. For example, many legitimate applications use DNS for storing non-IP data, such as public keys in the DomainKeys protocols [6], [7]. The anomaly detection analysis may result in false alarms.

Furthermore, traffic in our codeword mode described

in Section II-A is statistically indistinguishable from legitimate DNS traffic. Thus, we conclude that DNS-based botnet command-and-control is both feasible and practical.

VI. Related Work

Despite the fact that DNS tunneling is known for bypassing firewalls and encapsulating arbitrary data such as SSL traffic [9], [4], Exploring DNS protocol as a practical command-and-control channel and identifying its limitations have not been scientifically studied. Various proof-of-concept botnet command and control systems via unconventional media exist, such as via bluetooth [28] and social networks [14]. In comparison, our work is useful beyond the specific DNS-based communication channel studied in two aspects.

- We present new quantitative techniques and evaluation regarding the detection and construction of general-purpose distributed stealthy communication systems, including temporal strategies for making stealthy communication and statistical content analysis.
- We give a practical technique that is useful in domain flux from the attacker's perspective, namely Markov chain based domain name generation.

For DNS-based anomaly detection, Karasaridis *et al* described the use of the Kullback-Leibler distance to measure byte distribution in DNS datagrams [13]. Dagon [3] proposed to quantify how anomalous the number of queries for each domain name during an hour in a day with Chebyshev's inequality and distance measures previously used for examining anomalous payloads. DNS-based anomaly detection approaches are presented in [32] for detecting botnet C&C activities. One method is to detect dynamic domain names whose query rates are abnormally high or temporally concentrated using outlier detection metrics such as Chebyshev's inequality. Our work describes stealthy DNS behaviors whose querying patterns are hard to distinguish with legitimate domains, which make the counting based detection less effective.

Stone-Gross *et al* observed the use of domain flux in Torpig botnet [31], where new communication domains are generated periodically and registered by the C&C server. Torpig bots communicated with the server over HTTP, after resolving the domain name. Patterns of fast-flux botnets are measured and analyzed in [10]. In comparison, we investigate the feasibility of solely DNS-based command and control, without requiring any additional Web servers. The work in [35] utilizes machine learning techniques to identify domain names that are algorithmically generated. Though it remains unclear whether our MC-generated domain names can be experimentally distinguished from legitimate domain names by the techniques in [35], we conjecture that the MC-generated domains would be difficult to distinguish from legitimate ones. The work in [1] describes 15 features that can be used to detect anomalous DNS traffic in wide area networks, including IPs (e.g., a domain mapped to multiple IPs across different countries), TTL values (e.g., short TTL), temporal features (e.g., regularities in aggregated query patterns), and domain name features (similar to [35]). The stealth techniques on query pattern and domain name generation described in this work may help evade the machine learning based detection, showing the need for further research in this direction.

Our piggybacking DNS queries should not be confused with previously reported piggybacking methods for reducing DNS traffic. Those techniques usually take advantage of empty payload space in UDP datagrams. For example, renewal using piggyback method was proposed to piggyback cached DNS records to DNS queries to refresh expired cached records [12]. Related domains may also be piggybacked in DNS queries [26], e.g., to include `i.cnn.net` in the DNS packet for `www.cnn.com` as they are likely to be requested together by the browser.

Millen did pioneering work on covert-channel analysis [16], [17], in particular in a system (host) environment. Covert channel has been heavily analyzed in the context of traffic-analysis prevention [19] and routing anonymity [18]. Our work differs from them in that we focus on designing practical covert channels across the Internet.

Our work is complementary to host-based malware detection and prevention solutions, such as the cryptographic provenance verification technique [30], [33].

VII. Conclusions and Open Problems

We conducted a systematic study on the feasibility of solely using DNS queries for massive-scale stealthy communications among entities on the Internet. Our work shows that DNS – in particular the codeword mode combined with advanced querying strategies – can be used as an extremely effective stealthy command-and-control channel. To address the open problem raised in [2] on how to algorithmically generate short-lived and realistic-looking domain names, we found that using Markov chain produces realistic-looking domain names.

Our work points out the severity of DNS abuse for massive-scale communications and the challenges associated with its detection. Understanding the capacity of botnets communication power helps identify and

eliminate nefarious attacks launched from them. DNS-based botnet command-and-control is more stealthy than application-based command-and-control (e.g., email [29] or social network [14]), and such a C&C system also benefits from the decentralization of DNS. Some of our anomaly detection analysis is useful beyond the specific DNS tunneling problem studied.

We would like to point out the open research problems related to DNS-based stealthy communication. Besides command and control, DNS tunneling may also be used for exfiltrating sensitive data by attackers including rogue insiders. Payload inspection has been proposed for detecting data leaks (e.g., privacy-preserving data-leak detection in large TCP segments [27]). How effective these solutions are against leaks via small DNS queries remains unclear. From defenders' perspective, the approach of user-intention based anomaly detection has been demonstrated effective in detecting abnormal system events such as unauthorized file creation [34] and malware-triggered outbound traffic [36]. Because DNS queries are usually automatically issued by applications or the OS, the causal relations between user actions and DNS traffic may not be obvious. How to extend the user-intention based anomaly detection approach to identify anomalous DNS traffic on a host is an open problem.

REFERENCES

[1] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. Exposure: Finding malicious domains using passive DNS analysis. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, February 2011.

[2] P. Butler, K. Xu, and D. Yao. Quantitatively analyzing stealthy communication channels. In *Proceedings of the 9th International Conference on Applied Cryptography and Network Security (ACNS '11)*, number 6715 in Lecture Notes in Computer Science (LNCS), pages 238–254, 2011.

[3] D. Dagon. Botnet detection and response, the network is the infection, 2005. Domain Name System Operations Analysis and Research Center Workshop.

[4] DeNiSe. `http://c0re.23.nu/c0de/snap/DeNiSe-snap-20021026.tar.gz`.

[5] C. J. Dietrich, C. Rossow, F. C. Freiling, H. Bos, M. van Steen, and N. Pohlmann. On botnets that use dns for command and control. In *Proceedings of European Conference on Computer Network Defense*, September 2011.

[6] Yahoo! Anti-Spam Resource Center - DomainKeys. http://antispam.yahoo.com/domainkeys, 2008. This is an electronic document. Date retrieved: February 1, 2007.

[7] M. T. Goodrich, R. Tamassia, and D. Yao. Accredited DomainKeys: a service architecture for improved email validation. In *Proceedings of the Conference on Email and Anti-Spam (CEAS '05)*, July 2005.

[8] M. Hollander and D. A. Wolfe, editors. *Nonparametric Statistical Methods*. Wiley-Interscience, second edition, 1999.

[9] M. V. Horenbeeck. DNS tunneling. `http://www.daemon.be/maarten/dnstunnel.html`.

[10] X. Hu, M. Knysz, and K. G. Shin. Measurement and analysis of global IP-usage patterns of fast-flux botnets. In *Proceedings of International Conference on Computer Communications (INFOCOM)*, 2011.

[11] G. Hunt and D. Brubacher. Detours: Binary interception of Win32 functions. In *Proceedings of the Third USENIX Windows NT Symposium*, 1999.

[12] B. Jang, D. Lee, K. Chon, and H. chul Kim. DNS resolution with renewal using piggyback. *Journal of Communications and Networks*, 11(4), August 2009.

[13] A. Karasaridis, K. S. Meier-Hellstern, and D. A. Hoeflin. Detection of DNS anomalies using flow data analysis. In *GLOBECOM*. IEEE, 2006.

[14] E. Kartaltepe, J. Morales, S. Xu, and R. Sandhu. Social network-based botnet command-and-control: Emerging threats and countermeasures. In *Proceedings of Applied Cryptography and Network Security (ACNS)*, volume 6123 of *Lecture Notes in Computer Science*, pages 511–528. Springer, 2010.

[15] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[16] J. K. Millen. Covert channel capacity. In *IEEE Symposium on Security and Privacy*, pages 60–66, 1987.

[17] J. K. Millen. 20 years of covert channel modeling and analysis. In *IEEE Symposium on Security and Privacy*, pages 113–114, 1999.

[18] I. Moskowitz, R. E. Newman, D. P. Crepeau, and A. R. Miller. Covert channels and anonymizing networks. In *In Workshop on Privacy in the Electronic Society (WPES 2003*, pages 79–88. ACM, 2003.

[19] R. E. Newman, I. S. Moskowitz, P. Syverson, and A. Serjantov. Metrics for traffic analysis prevention. In *in Proceedings of Privacy Enhancing Technologies Workshop (PET 2003*, pages 48–65. Springer-Verlag, LNCS, 2003.

[20] G. Ollmann. Botnet communication topologies – understanding the intricacies of botnet command-and-control. Available at `http://www.damballa.com/`.

[21] C. P. Pfleeger. Crypto: Not just for the defensive team. *IEEE Security & Privacy*, 8(2):63–66, 2010.

[22] N. Provos and P. Mavrommatis. All your iframes point to us. In *Proceedings of USENIX Security Symposium*, 2008.

[23] M. A. Rajab, F. Monrose, A. Terzis, and N. Provos. Peeking through the cloud: DNS-based estimation and its applications. In S. M. Bellovin, R. Gennaro, A. D. Keromytis, and M. Yung, editors, *ACNS*, volume 5037 of *Lecture Notes in Computer Science*, pages 21–38, 2008.

[24] Extension Mechanisms for DNS (EDNS0). RFC 2671. The Internet Society. August, 1999. `http://tools.ietf.org/html/rfc2671`.

[25] SCIgen - an automatic CS paper generator. `http://pdos.csail.mit.edu/scigen/`.

[26] H. Shang and C. E. Wills. Piggybacking related domain names to improve DNS performance. *Comput. Netw.*, 50(11):1733–1748, 2006.

[27] X. Shu and D. Yao. Data-leak detection as a service. In *Proceedings of the 8th International Conference on Security and Privacy in Communication Networks (SECURECOMM)*, September 2012.

[28] K. Singh, S. Sangal, N. Jain, P. Traynor, and W. Lee. Evaluating bluetooth as a medium for botnet command and control. In *Proceedings of International Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2010.

[29] K. Singh, A. Srivastava, J. T. Giffin, and W. Lee. Evaluating email's feasibility for botnet command and control. In *DSN*, pages 376–385. IEEE Computer Society, 2008.

[30] D. Stefan, C. Wu, D. Yao, and G. Xu. Cryptographic provenance verification for the integrity of keystrokes and outbound network traffic. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security (ACNS)*, June 2010.

[31] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, November 2009.

[32] R. Villamarín-Salomón and J. C. Brustoloni. Identifying botnets using anomaly detection techniques applied to DNS traffic. In *Proceedings of the 5th IEEE Consumer Communications and Networking Conference (CCNC)*, 2008.

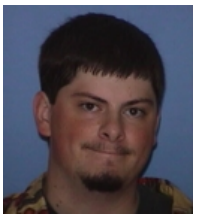
[33] K. Xu, H. Xiong, D. Stefan, C. Wu, and D. Yao. Data-provenance verification for secure hosts. *IEEE Transaction on Dependable and Secure Computing (TDSC)*, 9(2):173 – 183, March/April 2012.

[34] K. Xu, D. Yao, Q. Ma, and A. Crowell. Detecting infection onset with behavior-based policies. In *Proceedings of the Fifth International Conference on Network and System Security (NSS)*, September 2011.

[35] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 48–61, New York, NY, USA, 2010. ACM.

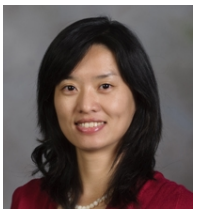
[36] H. Zhang, W. Banick, D. Yao, and N. Ramakrishnan. User intention-based traffic dependence analysis for anomaly detection. In *Proceedings of Workshop on Semantics and Security (WSCS)*, May 2012. in conjunction with the IEEE Symposium on Security and Privacy.

Kui Xu received the bachelor degree in computer science from the University of Science and Technology of China. He is a PhD candidate in the Department of Computer Science at Virginia Tech, Blacksburg. He is interested in cyber security research. In particular, he focuses on utilizing user behavior information in strengthening security. Major research topics cover drive-by-download detection, user-activity-based authentication, DNS tunneling analysis, and personalized anomaly detection. He is a student member of the IEEE.



Patrick Butler is a PhD candidate in the Department of Computer Science at Virginia Tech, Blacksburg. He received dual BS degrees in Computer Science and Physics from Virginia Tech in 2005. He is currently working in the realm of data mining, specifically temporal data mining. His other research interests include distributed data storage, Linux kernel network stack, and network security.

Sudip Saha received BS in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET) in 2006. He received MS in computer science from University of Memphis in 2010. Currently he is pursuing Ph.D. in computer science at Virginia Tech. His research interest lies in social and complex networks, game theory, and computer security.



Danfeng (Daphne) Yao is an assistant professor in the Department of Computer Science at Virginia Tech, Blacksburg. She received the PhD degree in computer science from Brown University. Before joining Virginia Tech, she was a tenure-track assistant professor in the Computer Science Department at Rutgers University for two years. Her research interests are in network and information security, in particular user-centric security and privacy, social- and human-behavior pattern recognition, insider threats, data privacy, and applied cryptography. She received the US National Science Foundation CAREER Award in 2010 for her work on human-behavior-driven malware detection. She won the Best Student Paper Award at ICICS 2006 and the Award for Technological Innovation from Brown in 2006, both for her privacy-preserving identity management work, and the Best Paper Award at CollaborateCom 2010 for keystroke security. In February 2012, she received the Outstanding New Assistant Professor Award from VT. She is a member of the IEEE and the IEEE Computer Society.