SCALABLE TECHNIQUES FOR ANOMALY DETECTION

A Dissertation

by

SANDEEP YADAV

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2012

Major Subject: Computer Engineering

ABSTRACT

Computer networks are constantly being attacked by malicious entities for various reasons. Network based attacks include but are not limited to, Distributed Denial of Service (DDoS), DNS based attacks, Cross-site Scripting (XSS) etc. Such attacks have exploited either the network protocol or the end-host software vulnerabilities for perpetration. Current network traffic analysis techniques employed for detection and/or prevention of these anomalies suffer from significant delay or have only limited scalability because of their huge resource requirements. This dissertation proposes more scalable techniques for network anomaly detection.

We propose using DNS analysis for detecting a wide variety of network anomalies. The use of DNS is motivated by the fact that DNS traffic comprises only 2-3% of total network traffic reducing the burden on anomaly detection resources. Our motivation additionally follows from the observation that almost any Internet activity (legitimate or otherwise) is marked by the use of DNS. We propose several techniques for DNS traffic analysis to distinguish anomalous DNS traffic patterns which in turn identify different categories of network attacks.

First, we present MiND, a system to detect misdirected DNS packets arising due to poisoned name server records or due to local infections such as caused by worms like DNSChanger. MiND validates misdirected DNS packets using an externally collected database of authoritative name servers for second or third-level domains. We deploy this tool at the edge of a university campus network for evaluation.

Secondly, we focus on domain-fluxing botnet detection by exploiting the high entropy inherent in the set of domains used for locating the Command and Control (C&C) server. We apply three metrics namely the Kullback-Leibler divergence, the Jaccard Index, and the Edit distance, to different groups of domain names present in Tier-1 ISP DNS traces obtained from South Asia and South America. Our evaluation successfully detects existing domain-fluxing botnets such as Conficker and also recognizes new botnets. We extend this approach by utilizing DNS failures to improve the latency of detection. Alternatively, we propose a system which uses temporal and entropy-based correlation between successful and failed DNS queries, for fluxing botnet detection.

We also present an approach which computes the reputation of domains in a bipartite graph of hosts within a network, and the domains accessed by them. The inference technique utilizes *belief propagation*, an approximation algorithm for marginal probability estimation. The computation of reputation scores is seeded through a small fraction of domains found in black and white lists. An application of this technique, on an HTTP-proxy dataset from a large enterprise, shows a high detection rate with low false positive rates.

To

my mother,

my father,

and my little sister.

ACKNOWLEDGMENTS

This is to thank Dr. Narasimha Reddy for his unwavering support during this stint. It is only through his inexhaustible patience, brilliant intellectual support, and careful advising that I have been able to reach this step.

I would also express my thanks to my parents for their support, and my sister who has been an inspiration as always.

I am grateful to Soups Ranjan who has been an incredible mentor and someone from whom I have learned a lot, and to Ashwath Reddy for being a fantastic friend and peer.

My committee has also been pivotal in steering my work and this is to thank Dr. Bettati, Dr. Loguinov, and Dr. Stoleru. My mentors and friends at Hewlett-Packard labs have also been greatly helpful in shaping my work and thoughts and inputs are really appreciated. This is to thank Pratyusa, Bill, Prasad, Stuart, and Raj.

I have also had the pleasure of sharing my office and thoughts with guys who are more like friends than officemates. Thank you Hang Su, Navid, and Sanghwan. I would also like to thank Carolyn for her support at all times.

Last but not the least, I would like to thank all my colleagues, teachers, mentors, roommates, and friends, who have in some way touched my life and shaped this dissertation. I am indebted to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A. Motivation

Networks around the world are subject to constant barrage of attacks that exploit either the end host (victim's) application or network protocol vulnerabilities. While attacks exploiting host software vulnerabilities include DNS misdirection, XSS (Cross Site Scripting) based attacks, malware infections for click-jacking, botnet expansion, etc., network attacks such as Denial of Service (DoS) or Distributed Denial of Service (DDoS), Domain Name System (DNS) cache poisoning, DNS amplification, etc., belong to the latter category. In particular, the botnet based attacks have seen a surge in activity in the recent past, where several thousand bots coordinate from different parts of the world to achieve a considerable scale for malicious activities such as for spamming, phishing, or DDoS attacks.

Identifying infected hosts or malicious network traffic is imperative for protecting user information such as passwords, credit card details, other confidential information, or for prevention of previously mentioned attacks. Many of the attacks require communication with external rogue entities. For instance, bots (machines infected with malware) communicate with a Command and Control (C&C) server either for retrieving instructions or for transferring the data stolen from victims. These communication patterns, if distinguishable from legitimate network traffic, can potentially lead to the detection of infected bots and the C&C servers coordinating the bots. In this work, we focus on developing several mechanisms which exploit or mine such differences for anomaly detection.

Previous approaches for such anomaly detection have deployed blacklisting mechanisms, honeypot/honeynet systems, or perform complete network traffic analysis. In the context of botnet detection, blacklist based solutions offer information about IP addresses acting as bots spewing spam, or the addresses of C&C servers herding the botnets. In addition, domain names which help locate the commonly observed rogue hosts (servers in most cases) may also be present in the blacklist. Using blacklists, however, incurs significant delay in terms of identifying the corresponding rogue domain name or IP address, and for establishing ample community confidence for blacklisting.

A honeypot system refers to the deliberate infection of a pristine user system such that the installed malware assumes its presence on a regular system. The infected system is generally a virtual machine with controlled network access to avoid the spread of malware within the network. Honeypots help understand the mode of operation of malware so that signatures expressing this behavior can be developed for future detection. Once the analysis using a honeypot is complete, the infected system image is erased or reset. While effective in the recent past, honeypot systems require significant manual work for setting up virtual machines. Such a solution additionally suffers from the drawback of malware being able to recognize its presence within a virtual machine (VM) rendering the honeypot system ineffective. The virtual machines are identified when malware searches for memory footprints or configuration specific registry entries at pre-determined locations. Malware can stop propagating itself to such VM-based honeypots in order to evade detection. At present, security researchers discover at least 1000 malware samples equipped with VM-detection capability every week [1].

Approaches which look at the complete network traffic include techniques which utilize machine learning or correlation techniques for anomaly detection [2, 3]. Such an approach suffers from the problem of scalability as a reliable analysis requires considerable resources even for small-sized networks (such as a university campus). Counter strategies to complete network traffic analysis have explored network sampling where only a fraction of the traffic is analyzed. Sampling based methods, however, result in many false negatives due to missed communication traffic which translates to a low reliability [4].

To ensure network protection while overcoming drawbacks of previous approaches, we propose analyzing DNS traffic for anomaly detection. The analysis of DNS traffic is motivated by the following. DNS comprises of only about 2-3% of the total network traffic. While there are several communication protocols available for use, malware uses the DNS for initiating any type of communication with an external malicious actor. Note that the alternative use of hard-coded rogue server IP addresses is antiquated and easily detectable through reverse engineering or controlled network analysis. The use of DNS, thus, provides footprints of anomalous communication which may then be identified through statistical techniques as proposed in this work. In addition, the protocol specific information that needs to be analyzed is relatively less compared to other application layer protocols such as HTTP, SMTP, etc. Therefore, we develop techniques utilizing DNS for detection and subsequent prevention of a wide variety of anomalies.

First, we present MiND, a tool to detect DNS packet indirection attacks within an autonomous system (AS) which arise due to infections caused by malware such as DNSChanger [5]. MiND uses a name server database to detect misdirected DNS queries by examining only the network layer information. The name server database uses publicly available DNS PTR and NS records to populate itself. The validity and authenticity of name server information is ensured through continuous updates. Using our tool, we detect the presence of malicious misdirections within our autonomous system with a performance that results in a false positive rate of less than 0.8%, with improved query verification latency when compared to prior solutions. We deploy MiND as an online analysis tool without requiring significant infrastructure upgrade or coordination from different entities.

Our second work focusses on the detection of domain fluxing botnets. A botnet refers to a network of compromised hosts (or bots) which communicate with a C&C server for instructions. The C&C server instructs the bot to either spam a list of email addresses, steal victim's private data, or assist in launching a DDoS attack. The botnets are of mainly two types viz. Peer-to-Peer based botnets or Central Command and Control based botnets. The difference lies in that the server (disemminating instructions) does not have a fixed location in the P2P architecture. Centralized C&C servers allow for an easier control but several approaches have been proposed to detect them. As a consequence, recent botnets have employed evasion methodologies that include frequently changing the IP address of the C&C server (also called *IP fluxing*) and more recently, frequently changing the domain names which resolve to the C&C server (called *Domain fluxing*).

Recent Botnets such as Conficker, Kraken and Torpig have used DNS based “domain fluxing” for command-and-control, where each bot queries for existence of a series of domain names and the owner has to register only one such domain name. From the point of view of a botnet owner, the economics work out quite well. They only have to register one or a few domains out of the several domains that each bot would query every day. Whereas, security vendors would have to pre-register *all* the domains that a bot queries every day, even before the botnet owner registers them, to thwart the communication between the bot and the C&C server. In all the cases above, the security vendors had to reverse engineer the bot executable to derive the exact algorithm being used for generating domain names. In some cases, their algorithm would predict domains successfully until the botnet owner would patch all the bots with a re-purposed executable with a different domain generation algorithm [6].

In the second part of this work, we develop a methodology to detect such “domain fluxes” in DNS traffic by looking for patterns inherent to domain names that are generated algorithmically, in contrast to those generated by humans. In particular, we look at distribution of alphanumeric characters as well as bigrams in the domain names at different levels. We present and compare the performance of several metrics, including K-L distance, Edit distance and Jaccard measure, for distinguishing algorithmic generated names from human generated names. We train by using a good data set of domains obtained via a crawl of domains mapped to all IPv4 address space and modeling bad data sets based on behaviors seen so far and expected. We apply our methodology to packet traces collected at a Tier-1 ISP and show that we can automatically detect domain fluxing as used by Conficker and other botnets with minimal false positives.

Botnets such as Conficker and Torpig utilize domain names with high alphanumeric entropy for fluxing and evasion. Bots may query a large number of domains, some of which may fail. In this extension of our fluxing botnet detection approach, we present techniques where the failed domain queries (NXDOMAIN) may be utilized for: (i) Speeding up the detection strategies which rely only on successful DNS queries. (ii) Detecting Command and Control (C&C) server addresses through features such as temporal correlation and information entropy of domain names of both successful and failed DNS queries. We apply our technique to a Tier-1 ISP dataset obtained from South Asia, and a campus DNS trace to validate our methods, detecting Conficker botnet IPs and other anomalies with false positive rates as low as 0.02%. Our technique can be applied at the edge of an autonomous system for real-time detection of fluxing botnets.

In the last part of this dissertation, we attempt to identify a wider variety of anomalies through graph inference techniques. Malware infections within a network are spawned and spread primarily through domains accessed by network clients. Thus, containment of infected hosts and prevention of access to malicious domains is critical for every network. Reliably identifying infected hosts and malicious domains, however, is challenging. Modern malware such as rootkits may evade anti-virus products and other host-based analysis techniques. Similarly, identification via network and communication traffic analysis is difficult as botnet communication by design tries to mimic legitimate applications [3]. Many enterprises use blacklists to identify and prevent malicious domain accesses; such lists, however, incur a significant delay in adding new domains as they rely on several manual sources. Machine learning techniques, on the other hand, require large feature data sets and labeled training sets [7, 8]. There is a need to develop techniques that can function with limited or small data sets in characterizing the observed network behavior to detect anomalies.

We propose an approach to infer the reputation of a domain or a host as being malicious (or benign) using a limited amount of already known information about hosts within the network, and about the domains that the hosts access. To apply our approach, we first construct bipartite graphs of hosts and domains from the HTTP-proxy dataset of a huge enterprise network, and also from a campus DNS recursive resolver dataset. We apply the belief propagation algorithm for marginal probability estimation, on the bipartite graph seeded with the small ground truth. We evaluate our approach using K-fold cross validation and also by verification against multiple external sources and validation parameters. Our results on the two datasets show that application of our technique gives high detection rate with low false positive rates and low false discovery rates. Consequently, our approach can be applied passively at the edge of an autonomous system for real-time detection.

B. Domain Naming System

Figure 1 shows how the naming system works. The DNS hierarchy consists of a tree of domains or zones. The root node (denoted by ".") is at the top. The next level has the well known domain names called top-level domains. For instance, in a domain name such as *www.google.com.*, *"com"* is a top-level domain (TLD). Different levels of domain names are separated by a period. In the DNS hierarchy, subsequent domain labels (such as *google*) are the children of the TLD node, and so on.

The local recursive resolver traverses the DNS hierarchy to resolve a domain name query. For instance, if a user wishes to find the IP address for *www.google.com*, the client machine (also known as the *stub* resolver) will pass this query to the recursive name server

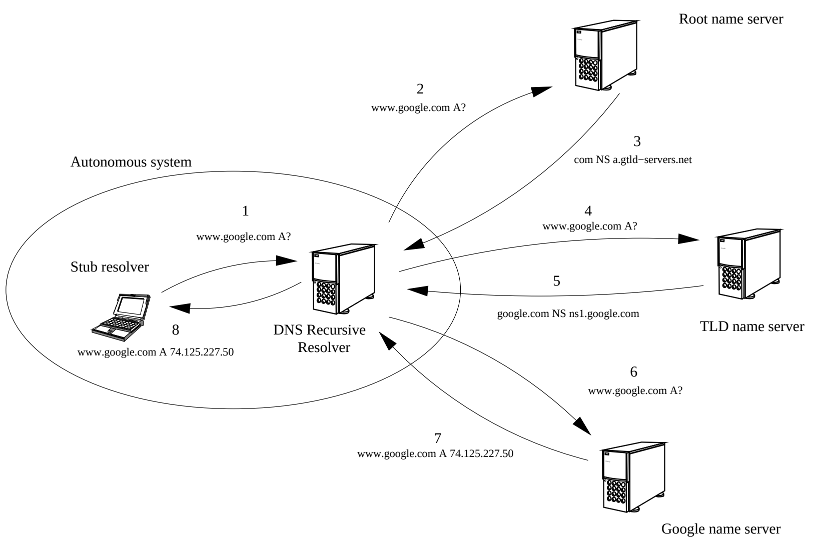


Fig. 1. How DNS works

configured to provide answers to DNS queries (step 1 in the figure). Assuming that no information is cached already, the recursive name server contacts the root name server asking for the address of *www.google.com*. The root name server refers the query to *com* name servers which in turn refers the query to *google.com* name servers, which reply with the required answer. Note that the stub resolver issued a *recursive* DNS query to the local recursive DNS server, which retrieves the answer from the web *iteratively* by reaching out to the necessary name servers on the Internet.

The name servers responsible for answering the all DNS queries belonging to a domain, are considered to be *authoritative* for that domain name. For instance, queries such as *mail.google.com*, *plus.google.com*, etc., are answered with fresh information from the authoritative name servers such as *ns1.google.com*. This information may have already been cached and it is up to the end user to use cached data or authoritative data. The DNS response indicates whether the reply was sent from an authoritative name server, by setting an AA bit in the DNS response packet.

The recursive name server outlined above may also act as a DNS *aggregator* which serves a small subnetwork. The stub resolvers within the subnet are configured to issue their DNS queries to the aggregator. The aggregator on receiving queries, does not *iteratively* resolve the queries, as a normal recursive server would. Instead, the aggregator acts like a stub resolver and forwards the query recursively to another DNS recursive server which resolves the query, and provides the answer back to the aggregator which then forwards it to the respective stub resolver.

DNS information propagates in units called DNS Resource Records (RRs). The RRs may indicate IP address of the host name asked (*A* record), the host name for an IP address (reverse mapping using a *PTR* record), the name servers for the queried domain (*NS* records), and more. Various techniques proposed in the chapters ahead look either at the domain names of various DNS RRs or look at the corresponding value for the type of RR. For instance, for type A records, we look at the IP address, for PTR record, we use the corresponding domain name again. It is important to note the terminology we use in the chapters ahead. For a host-name such as *abc.examplesite.org*, we consider *org* as the TLD or a first-level domain, *examplesite* as the second-level domain label, *examplesite.org* as second-level domain. Similarly, *abc* denotes third-level domain label and *abc.examplesite.org* is the third-level domain and a sub-domain of *examplesite.org*.

In context of various DNS based attacks, cache poisoning attempts to pollute the huge cache of DNS recursive resolvers which serve stub resolvers within the autonomous system. Poisoning of DNS records results in modification of the RRs such that the information a RR represents is rogue. For instance, attackers can poison DNS type A record for *www.citibank.com* where the true IP address of the web server is now replaced by the IP address of a rogue server with a website resembling the original website. As a result, when a stub resolver queries for the affected domain, the user will be tricked into accessing the rogue website with possibly losing their confidential information to attackers. Our work focuses on identifying poisoned NS records and also DNS packets transmitted to rogue DNS servers rather than the local recursive server. In addition, we focus on characteristics of the domain name queried by the stub resolver for detecting botnets or malicious domain/URL access.

CHAPTER II

MIND: MISDIRECTED DNS PACKET DETECTOR

A. Introduction

Domain Naming System (DNS) forms one of the most critical network infrastructure components in the Internet. Recently it has become increasingly vulnerable to exploits and attacks. Various types of attacks have been launched on or using the local DNS recursive resolvers of victim networks as well as authoritative name servers of various domains [9]. These include reflection attacks, amplification attacks, Denial of Service (DoS), Distributed Denial of Service (DDoS), cache poisoning, indirection and many more. Cache poisoning and indirection attacks, in particular, have seen a surge in the recent past. These attacks expose users to malicious entities eager to mine confidential user information. In this chapter, we present a detection and prevention mechanism for attacks based on indirection and poisoned NS records.

DNS cache poisoning refers to the replacement of valid answer/authority/additional resource records (RRs) with malformed content. Cache poisoning attack requires the attacker to guess the 16-bit TXID, the transaction ID used to identify a DNS packet and match the response to the query issued by a recursive server. Guessing the correct ID makes the recursive resolver accept a corrupted packet, thereby serving malicious information to its clients. However, randomization of DNS and IP layer features of the network packet makes poisoning somewhat difficult.

Indirection attack, however, refers to a stealthy change in the DNS resolver address on a stub resolver. The change in the DNS resolver address directs all recursive DNS queries to a malicious resolver. The malicious resolver then provides the victim with false domain information, subsequently leading to the victim's confidential data being compromised.

Indirection can be achieved by malware installation, as done by the notorious DNSChanger worm [10].

Many solutions are proposed to counter DNS cache poisoning and indirection. However, most have not been widely adopted. For instance, DNSSEC [11–13] provides authenticity of DNS data using cryptographic techniques. However, it still awaits deployment owing to requirements of large infrastructural changes. Similarly, DJBDNS overcomes many of BIND's shortcomings but is yet to see a notable deployment. Changes adaptable to existing systems, however, enjoy ready adoption. For instance, DNS Black Lists(DNSBLs) have been used to counter email spam. Therefore, it becomes essential that we design techniques that can co-exist with present systems. Hence, we propose *MiND* (Misdirected dNs packet Detector), an easily deployable technique that provides counter-threat measures without any changes to the existing DNS infrastructure. Our technique provides domain specific security without requiring cooperation from other network entities.

MiND is used to counter the threat posed by DNS packet indirection arising due to poisoned NS records present in the recursive resolver cache, or due to malware induced modifications of system's resolver settings. We implicitly verify the validity of all types of DNS records by ensuring the validity of their destination address. Our tool uses a self accumulated database of name servers, which continuously updates itself. To obtain a DNS record, the local recursive resolver should be able to resolve such a query by sending iterative queries to root name servers, the Top Level Domain (TLD) name servers and ultimately the domain's name servers respectively. If however, we find a DNS query directed towards IP addresses other than the valid name servers (with respect to the queried domain), the query is considered as an anomaly. We obtain the valid name server information using the publicly available authoritative PTR/NS records. To counter the dynamic nature of the valid set of name servers for a domain, we also consider Time-to-Live(TTL) of DNS name server records when validating DNS queries. Expired records are updated and hence used for query verification. We show that such a simple validation is effective in determining misdirected DNS packets originating from within the network. To the best of our knowledge, we are the first to collect the name server information by executing a light-weight crawl of the IPv4 address space, and using it for DNS packet validation.

The main contributions of this chapter are:

- We propose MiND, a counter-measure to DNS indirection and cache poisoning attacks, that is readily deployable.
- We demonstrate the quick building of a name server database, which helps us associate a given domain name to a specific set of domain authoritative name servers.
- Demonstrate that MiND indeed makes it difficult for the attacker to compromise various recursive DNS resolvers as well as stub resolvers (ordinary hosts).

With a low false positive rate and moderate hardware and software requirements, we show that MiND can be used to effectively protect a network from DNS based attacks.

B. Related Work

In this section, we review prior solutions proposed to counter DNS cache poisoning and indirection. We also review known attack techniques and briefly cover the literature that has helped build our system.

Dagon et al. [9] describe and define the problem of DNS cache poisoning and indirection. The authors in [14] use 0x20-bit encoding scheme to encode the question section within the DNS packet and hence counter DNS cache poisoning. The 0x20 encoding scheme changes cases of letters in a domain name based on the key unique to the resolver. The resolution software is expected to reply with the same cased letters as originally requested, increasing the difficulty for the attacker as she has to guess more entities (case letters, TXID) with respect to the packet, within the round trip time of a DNS reply. WSEC-DNS [15] utilizes the wild card capability of CNAME records to increase cache poisoning difficulty significantly. Yuan et al. [16] propose using a peer-to-peer system to counter DNS cache poisoning. Our work makes cache poisoning of NS records highly difficult, with reduced latency. Additionally, we are able to handle DNS packet indirection arising from stub resolvers within our network.

Delays between various hosts and DNS servers are measured in [17]. These delay measurements help in outlining timeout requirements of various dependent applications. These results guide our work as explained later in this chapter. DNS TXID and port randomization [18] are two of the most prevalent techniques for preventing cache poisoning. However, with the advent of Kaminsky attack, the said measures have become obsolete [19].

The Kaminsky attack coaxes the stub resolver to query for sub-domains of the victim domain name. To achieve poisoning, the attacker attempts to guess the randomized packet parameters. The stub resolver is forced to query other sub-domains of the victim domain without waiting for TTL expiry. In most cases, Kaminsky attack has been shown to poison DNS resolver caches in a matter of seconds. Our method is independent of the technique used to poison, and hence can immediately detect and isolate suspicious poisoned records.

The employment of a secondary cache of expired DNS records is proposed to mitigate the effects of DoS attacks on DNS infrastructure [20]. Our approach also employs a secondary cache or a database for DNS records, but our approach is focused on detecting anomalous DNS resolutions in real-time and the information cached by our tool is only a subset of that proposed in [20].

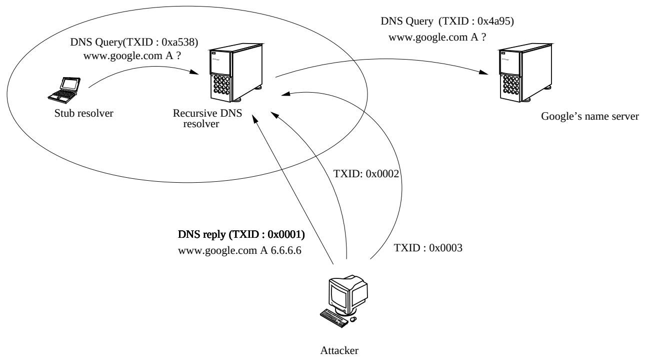


Fig. 2. DNS cache poisoning scenario.

C. Threat Model

Cache poisoning attacks can be initiated from outside or from within an autonomous system. Figure 2 shows cache poisoning attempts where the attacker tries to guess the correct TXID (16-bit transaction ID) of the DNS packet. It is assumed that the DNS traffic is invisible to entities outside the autonomous system, and hence it is impossible to observe the domain query packets. If assumed otherwise, poisoning DNS resolver cache becomes trivial, as the correct TXID can now be placed in the spoofed reply. Mechanisms such as [14, 15] make it much more difficult to spoof a reply to a DNS query.

Attackers who do not possess control over entities within a network, utilize hit-and-trial methods to meet their objectives. For instance, the attackers can spoof replies for most commonly accessed domain names (such as *www.google.com*) when attempting to poison a resolver. The chances that their spoofed packets get accepted increase, depending on the popularity of such domain names.

Another form of attack can be initiated from within the local network (or autonomous system) using ordinary stub resolvers. A stub resolver forced to query for particular host names can reduce the time required to poison resolver's cache. These client machines can stage a Kaminsky style attack by coordinating with external hosts.

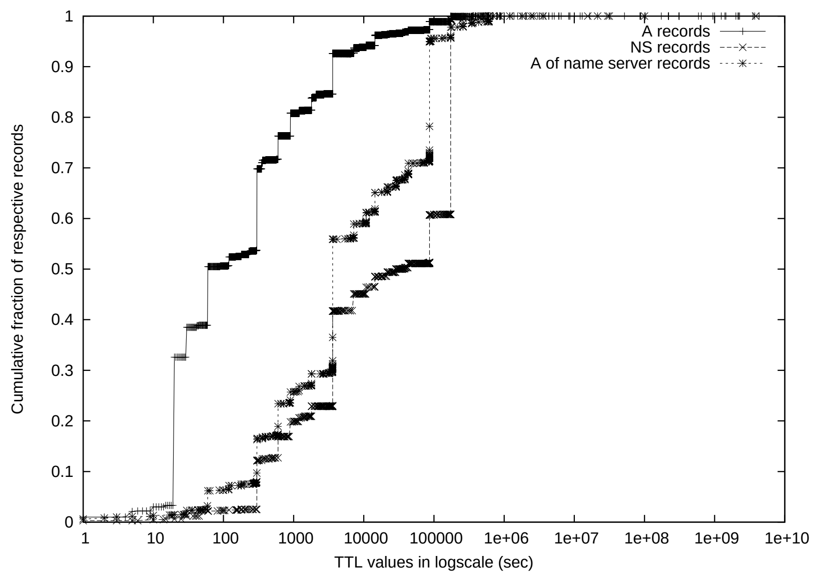


Fig. 3. Cumulative Distribution Function (CDF) of TTL values for A, NS, and A of name server records.

Our threat model considers both the insider and outsider threats as described above.

D. The MiND System

In this section, we describe MiND, the tool used for detecting DNS indirection of all types of DNS records, and cache poisoning of NS records, within a local autonomous system. Our approach employs a database of authoritative name servers for all domain names. In sub-section 3, we explain how the name server database is created and maintained. The destination name server in the DNS query is validated against the set of authentic name servers stored in the database, and an anomaly alert is raised if there is no match. The name server check against the database is done in parallel to the query resolution and thus the results in the response packet can be questioned if it is resolved by an unknown name server.

Several domains employ rapidly changing name server identities for load balancing or other purposes. Such a system uses fast changing NS records, and/or name server's A

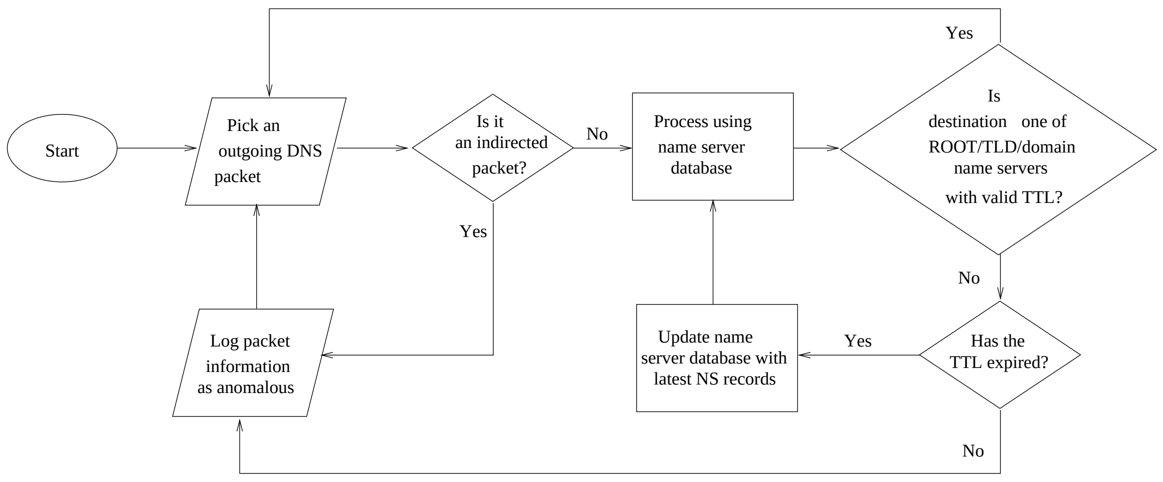

Start
Pick an outgoing DNS packet
Is it an indirected packet?
No
Yes
Process using name server database
Is destination one of ROOT/TLD/domain name servers with valid TTL?
Yes
No
Has the TTL expired?
Yes
No
Update name server database with latest NS records
Log packet information as anomalous

Fig. 4. MiND flowchart.

records. It is vital that our system recognizes such configurations appropriately and does not raise false alarms. To decrease the possibility of false alarms, we continuously update the authoritative name server information. The freshness and validity of the database is ensured using the TTL parameters present in the NS and A records. The following subsections discuss each component of the MiND system in detail. We also outline the benefits and potential issues of our technique.

1. Name Server Database Feasibility

Figure 3 represents the CDF of the TTL values for general A records, NS records, and A records of name servers respectively. From the graph, we observe that the TTL values of NS records are typically higher than the A records. While 90% of A records have TTL values less than 10000 sec, the same TTL value holds true only for about 45% of NS records. It is noteworthy that A records of the name servers also exhibit large TTL values, with 55% of A records of name servers having TTLs smaller than 10000 seconds. This suggests that maintaining the name server database is more convenient as the name server information is stable over a longer period of time. Alternatively, it allows us to use the same database for

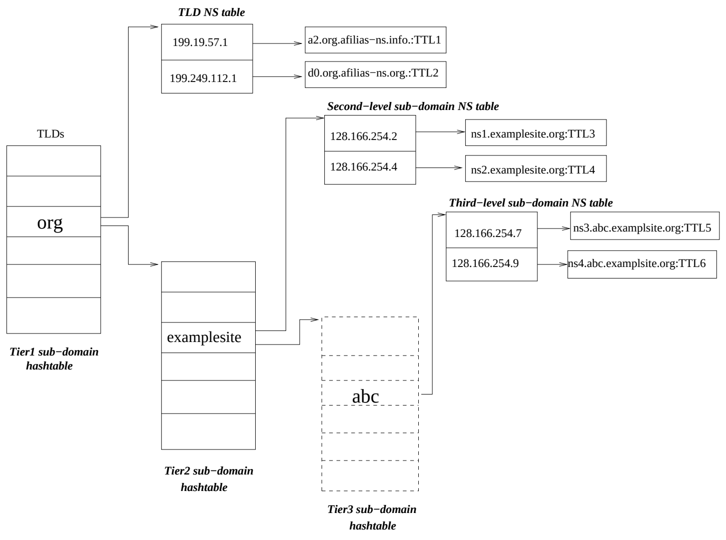

TLD NS table
199.19.57.1
199.249.112.1
a2.org.afilias−ns.info.:TTL1
d0.org.afilias−ns.org.:TTL2
Second−level sub−domain NS table
128.166.254.2
128.166.254.4
ns1.examplesite.org:TTL3
ns2.examplesite.org:TTL4
Third−level sub−domain NS table
128.166.254.7
128.166.254.9
ns3.abc.examplsite.org:TTL5
ns4.abc.examplsite.org:TTL6
TLDs
org
Tier1 sub−domain hashtable
examplesite
Tier2 sub−domain hashtable
abc
Tier3 sub−domain hashtable

Fig. 5. MiND query verifying data structure.

DNS query verification over longer intervals, making our approach scalable.

2. MiND Query Verification

The MiND anomaly alert system sits at the edge of the administrative domain observing all DNS packets. Our primary objective is to determine anomalies present within an autonomous system. Therefore, we present our analysis for outgoing DNS packets. As described later, incoming packets may also be validated using our tool.

The packet validation algorithm is presented in Figure 4. As the flowchart demonstrates, a packet is first checked for being a DNS indirection instance. Commonly, a stub resolver directs *recursive* DNS requests to local name servers only. However, if we encounter packets whose source IP address is different from the local recursive name servers, it is flagged as an indirection instance. On the other hand, independently installed resolvers within the local network can make iterative DNS requests to name servers outside too. With MiND in place, such resolvers need to be white-listed and this helps the network administrators to be aware of other resolvers in the network. Also, known free recursive DNS services like GoogleDNS [21] are white-listed.

Following the check for DNS indirection, we validate the query packet for being misdirected due to a poisoned resolver cache. If the cache of the local recursive DNS resolver contains corrupted NS records, then DNS packets are routed to name servers different from authoritative name servers for the queried domain. Using the name server database, we build a data structure in memory which quickly identifies the authoritative name servers for a given domain name. As shown in Figure 5, the data structure is organized into different tiers, with each tier containing name server information relevant to the corresponding subdomain. For query verification, using the name server information for up to the third-level domains, improves the results considerably. We justify the rationale of using such a design in the results section.

To verify DNS query packets, we also use TTL values associated with the name servers, obtained while collecting the database. There are two types of TTLs associated with a name server. One type specifies the TTL value of name servers for a particular domain. The second type specifies the TTL of the IP address associated with the name server. To ensure correctness of our analysis, we consider the minimum of the two TTL values.

The use of TTL is motivated by the fact that many networks change their configuration rapidly, which includes altering service and/or name server IP addresses. For example, Content Delivery Networks (CDNs) such as Akamai and LimeLight Networks exhibit such behavior. Therefore, the name server database might not always be able to determine the exact network state. However, by considering TTLs, we are able to determine the currently valid set of authoritative records, which if found expired, may be fetched again.

We check an outgoing DNS query against the name server information of multiple subdomains corresponding to the query. If the outgoing address matches the valid name server information obtained from any level of the query verifying data structure, we consider the query as directed to a safe name server. Else, it is considered misdirected.

Fast changing name servers pose problems during packet verification. We could falsely designate a packet as misdirected because the query does not seem to be bound for the most recent set of name servers publicized by the domain, although the name servers belong to the authoritative AS. This may happen because of caching of NS records with TTL values modified at the clients or any of the intermediate resolvers. In such a scenario, we use an additional *expired cache* of name servers. Such a cache is stored in the same way as the list of current (and valid) name servers associated with a domain name. However, the expired cache consists of authoritative name servers seen previously for the domain, and thus, we may see DNS packets destined for such name servers. We associate a learning time for such an expired cache, allowing us to supplement the valid set of name servers appropriately during packet verification. Each name server in the expired set still represents a trusted entity. We note that the use of expired records makes MiND vulnerable, only in the event where any of the expired name server IP address' authority is transferred to (or bought by) a rogue identity. However, such an event would take a considerable time to happen. To prevent such abuse, we frequently flush the expired name server set. Section E highlights the benefits obtained by using such an expired cache.

If a packet matches any of the validation criteria as outlined in Figure 4, it is deemed harmless. However, on failing every test mentioned above, we flag such a packet as anomalous. The network administrator may thus choose to take appropriate action in the form of either patching necessary machines, correcting mis-configuration at source hosts, blacklisting domains and/or hosts, etc.

In addition to detecting the compromised machines (as determined above), MiND may be used to *prevent* cache poisoning of NS records too. Prevention is achievable through an analysis of all incoming DNS packets' NS records present in either answer, authoritative, or the additional sections of the records. The MiND system thus provides counter-mechanism for Kaminsky attacks [22]. With MiND in place, malformed name server address(es) as advertised in a rogue DNS response packet, can be immediately identified and the attack thwarted. In addition, on observing multiple DNS replies for a query within a short window of time, our tool may be configured to raise a poisoning attempt alert.

3. Collecting the Name Server Database

The name server database is simply a collection of name servers corresponding to different domains present in a host name. The name server database consists only of NS records (with A records obtained only for the name servers) relevant to domains crawled. We use the publicly available PTR records corresponding to IPv4 addresses and thereby determine the list of valid name servers. We argue that PTR record specification determines the authoritative host name corresponding to an IP address. The name servers obtained by querying for multiple domain levels present in a PTR record listed host name, correspond to an authoritative set. It is important to note that PTR record pollution requires hijacking the name server for the victim domain's address space. For attackers, the benefits obtained from such an attack are very limited and thus their focus instead is to pollute the recursive resolver serving an autonomous system.

The name server database also stores TTL value associated with every name server/IP address combination. Such a collection of records is only a small subset of what a regular resolver is expected to store in its cache. The small set that we use can be quickly read into the main memory and has a small memory footprint. This sub-section discusses how we build this database within a small period of time.

PTR records contain IP address to host name mapping. Therefore, building an exhaustive name server database, at first, requires sending out a DNS PTR query for every IP address in the IPv4 (32-bit) space and hence obtaining the host name for that IP address. The name servers are then obtained by sending out an NS query (and if not cached already, an A query to determine the name server's IP address). The results are logged into the database. We query the name server records for second and third-level sub-domains of the host name returned as a response to a PTR request. To elaborate, if the PTR record corresponding to an arbitrary IP address gives *random.abc.examplesite.org* as the answer, we determine the name servers for *abc.examplesite.org* as well as *examplesite.org*. For PTR responses which contain multiple host names for an IP address (in the form of CNAME or more PTR records), we process each returned answer individually.

Determining the host name and the name servers for the corresponding domains requires sending out at least two queries for every address in the IPv4 space which may thus require sending a large number of queries cumulatively. However, we apply numerous heuristics to drastically reduce the required resources. We note that a considerable part of the Internet address space is unallocated/reserved which reduces the number of queries that need to be sent [23]. We also avoid generating name server queries for domains already seen, thus saving a large bandwidth and reducing latency of database collection.

To further reduce the volume of queries generated for aggregating a name server database, we analyze flags in DNS packets obtained in response to PTR requests for class A, B, C network prefixes. Thus, crawling large networks may altogether be avoided if the response packet indicates so. Note that we crawl the IP address space randomly to avoid appearing as a malicious scanning host.

```
random.abc.examplesite.org

ns1.examplesite.org:128.166.254.2:300|ns1.examplesite.org:128.166.254.4:300

ns3.abc.examplesite.org:128:166.254.7:1600

...
```

Fig. 6. Sample line from a name server database file.

Figure 6 shows a sample line from one of our name server database files. The first line corresponds to the host name of an IP address, returned as a reply for the PTR request. The following line denotes the name servers authoritative for *examplesite.org*. Name server information for *abc.examplesite.org* is present immediately below. Note that we also log each name server's IPv4 address and TTL associated with the name server. We also enable accumulating name servers with multiple IP addresses. Apart from host names corresponding to the 32-bit IPv4 space, the MiND query verifier requires information about name servers corresponding to the top level domains (TLDs). We use the list provided by [24, 25] to compile all TLDs and hence determine the name servers for each TLD.

The MiND system can also be deployed without complete authoritative name server information. However, the latency of query verification will be higher in such a scenario as the corresponding authoritative name server information will now have to be retrieved explicitly. Instead, once a database is built, only a small fraction of it needs to be updated frequently thereafter.

a. Database Authenticity and Completeness

With reference to utilizing a personal DNS resolver for database collection, a natural question arises as to why can't our own recursive resolver be poisoned? The database collection resolver, unlike a recursive resolver serving network clients, is configured to refuse any recursive queries from unknown hosts. All queries are made on the loopback interface. The database may be updated only when we receive the same answer from two resolvers located outside our domain/AS. Compromising two resolvers makes it harder for the attacker to poison their caches. Utilizing external resolvers, however, increases the latency. Therefore, deploying such a system for online query verification may delay applications requiring quick responses. DoX [16] utilizes such a configuration. However, fetching a small number of NS records, as we do, makes the solution viable.

The latency cost of our verification is significantly reduced by the employment of a name server database (as highlighted in section E). We focus on verifying all records by caching NS records only. In a simple scenario for DoX, which has a minimum of two peers, every query requires a query resolution from both of the peers. With MiND however, an additional query is issued to populate the name server database only if the required information is not present. MiND is also view independent, unlike DoX. This implies that inconsistencies arising with peers in DoX due to geographic separation, does not affect MiND as the database is collected within the autonomous system to be protected.

Another security enhancement measure is by executing a TCP based database collection. Using the TCP protocol ensures that the packet cannot be spoofed and that the DNS record is authentic, unless poisoned prior to collection.

As a consequence of all such measures, the database resolver becomes less prone to cache poisoning attacks and the integrity and authenticity of the database can be maintained.

We note that although an optional part of DNS configuration, PTR record specification by domain administrators can improve the quality of our database. We also note that with the absence of name server information in our database, we fetch the required name servers complementing the verification procedure. For instance, many web domains may be hosted by an IP address. However, only one PTR record pointing to one of the hostnames, may be present. In such a case, for all unseen domains, we find the name servers with a new DNS lookup.

4. Security Implications

MiND has been developed with the goal of providing an additional layer of security over the existing mechanisms. In a normal scenario, DNS resolver poisoning may be achieved by focusing attack resources only on the recursive resolver. With MiND deployed, attackers must now additionally compromise the name server database to achieve success.

With our tool, we focus on preventing corruption of NS records. However, by verifying the destination address of every type of DNS record, we implicitly ensure each record's correctness. We note that incoming DNS replies may also be spoofed and thus the DNS records may serve malicious data. With MiND, we also deploy a DNS response anomaly detector which looks for a burst of incoming responses within a short time. Thus, attackers need to be successful in matching TXID and random port number with the first packet, as subsequent packets will result in a response anomaly alert.

E. Results

1. System Requirements

The name server database collection engine is a multi-threaded application written using C and C++, and the commonly available resolver (*libresolv*) and packet capture libraries [26–28]. We use an Intel Core 2 Duo (@2.33GHz)Linux box running kernel 2.6.31, with 2GB RAM and plenty of hard drive space. The actual name server persistent database size is less than 1GB. The box uses a BIND9 DNS resolver and our application makes all DNS requests to this box. The private DNS resolver is configured against misuse and accepts recursive queries from trusted hosts only. The query checker is also developed in C and C++ too. The MiND query verifier system uses the same hardware configuration as the name server database collection engine.

DNS query verification uses the campus DNS trace (UDP port 53 traffic) for analysis. We use a fresh database for every analysis. We also use several campus traces from December 2009, for offline analysis. The subsections below highlight whether the results are based on online or offline DNS traffic.

Table I. MiND performance

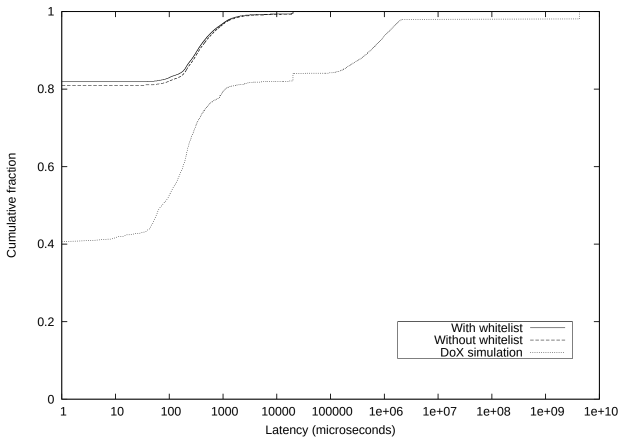
Exp. #	**Approx. hours of training**	**Total outgoing packets analyzed**	**Anomalies after forward checks (%)**
1	2	111306	921 (0.83)
2	5	173962	1342 (0.77)
3	8	286114	2704 (0.94)
4	15	288441	2423 (0.84)
5	30	1152469	8793 (0.76)

2. Performance Analysis

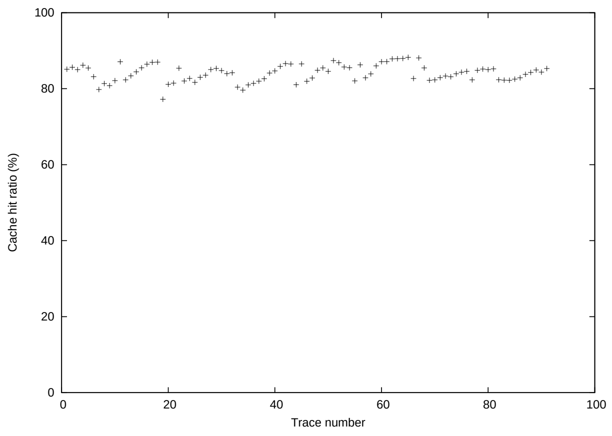
Table I shows the results from the analysis of live DNS traces. Each experiment is performed for a different duration of time (or the total number of packets). For each experiment, we consider both outgoing and incoming DNS packets (which use the UDP protocol) flowing between our recursive resolver and name servers outside our autonomous system. However, we validate only the outgoing packets.

The second column in table I represents the time for which our tool is trained before it starts packet verification. By training, we imply that the expired name servers for observed domains were not discarded. Rather, they were moved to a separate cache and thus used in conjunction with the current set of name servers for validating outgoing DNS packets. Column 3 represents the total outgoing packets which were verified with our tool, after the initial training.

On subjecting the outgoing packets to MiND query verification and forward checks, we find an average of 0.84% anomalies, as highlighted in column 4. By forward checks, we imply validating the destination address of the DNS query against the name servers of sub-domains of the third-level domain name. False positive rate can be further reduced by checking if the name server and the destination IP address of the DNS packet, belong to the same AS. This check reduces the false positive rate to an average of 0.046%. However, such a check increases the scope of an attack as the rogue resolver may belong to the same



(a)



(b)

Fig. 7. (a) Latency of verification. (b) Cache hit ratios for a week long dataset.

AS. With time, we expect the fraction of false positives to decrease as more name servers get aggregated in the expired cache.

We further analyze the packets remaining from above, specifically for experiment 3. From our analysis we find that 15.38% of the anomalous packets, belong to malicious domains. The malicious nature of the suspect domains is confirmed with a domain reputation verification service [29]. The bulk of the malicious packets belong to *exitguide.ru*. We further confirm the malfeasance of this domain using McAfee's domain reputation checker [30] which associates this domain name with highest risk. We also find that another 24.17% of the anomalous packets from experiment 3, belong to the category where either the NS records could not be retrieved or the A records for corresponding NS records were absent. Such a case may represent botnet hosted domains which publish name servers at specific times. In addition to above, we find 38.47% packets for which the name servers were present, but were seen to be in a different AS than checked before. This commonly occurs for CDNs or domains with low TTLs, which switch ASes rapidly. The remaining 22.17% of the packets occur due to the CNAME replies observed for NS queries. However, for forward and AS check, we do not consider CNAME referrals.

3. Latency of Query Verification

Figure 7(a) shows the cumulative distribution function of latency observed with MiND deployed under different scenarios. The figure shows the latency for three cases. The top most plot of latency refers to using MiND when the DNS packets destined for white-listed domains, are unconditionally verified. The middle plot refers to the latency observed when no white-listing is considered. For this particular analysis, we consider five frequently used domains for white-listing, namely *akamai.net*, *gslb.com*, *weather.com*, *facebook.com*, and *adnxs.com*.

The last plot represents latency observed with a simplistic setup for DoX [16]. With this setup, every outgoing DNS query is validated with another peer. In a more powerful setup for DoX, multiple resolvers resolve the queries and consult with other *peers* before providing clients with an answer. Our setup for DoX should bear minimum latency.

As the plot shows, the difference between latency observed with MiND and DoX is quite large. We owe this to the fact that many outgoing queries are verified quickly using our database of records. On the other hand, with DoX, every query needs to be fetched and then compared with peers for validation. We also observe that consideration of a white-list of only 5 domain names, results in a slight improvement in latency. We believe that with a white-list consisting of hundreds of domains, the latency gain would be much higher.

4. Interesting Observations

Here we present the interesting events observed as a consequence of DNS query verification. All the instances reported below represent different anomalies that may interest network administrators.

Absent name servers: From our analysis, we discover several irregularities, owing to no name servers being found for the domain names present in the query. Rather, such queries were resolved by simply following referrals from iterative replies (that is, using the AUTHORITY section data in DNS responses). Our implementation does not consider AUTHORITY section data. We observe that blacklisting services like *spamcop.net*, and *surbl.org* fall into this category and attribute for almost 20% of such packets.

Mistyped domain names: During our analysis, we observe several mistyped domain names. The frequently misspelled DNS domain name packets include MX queries for *g-mail.com*, A queries for *google.co,* MX queries for *hotmal.com, hotmai.com*, *hotmaill.com*, and more. While in all of the cases mentioned above, the mistyped domain names have been pre-registered by the responsible authorities, we note that the clients within our autonomous system generating such domain names, may represent irregularities. Such domains show up in our analysis because of the absence of NS records corresponding to these domains. With the frequency of these queried domains being of the order of several hundreds within a short time, MiND helps detect them for examination by system administrators.

Poisoning attacks: Our findings indicate that clients within our autonomous system repeatedly query for domains such as *i.metarss.com*, *computerfinance.net* and a few others within a short time. Such an analysis also highlights the local resolver configuration. It seems that the caching of NXDOMAIN/SERVFAIL records is disabled. Therefore, the client applications keep querying for the same hostname even after previous failures. Network administrators may patch such anomalous clients thereby saving a lot of bandwidth.

Simulated attack: We also simulate a poisoning attack on our private recursive resolver. The attack is accomplished by staging a Denial of Service attack from one of our host machines, to our experimental recursive resolver. We use a custom program to generate DNS packet bursts with random IDs and use *iptables* to redirect malformed packets, to our victim resolver. With the detection mechanism in place, we are indeed able to generate instant alerts.

5. Highly Dynamic Name Servers

We use each experiment (as in table I) to determine the frequently changing name server configurations. To determine the fast changing name servers, we simply determine the frequency of every second-level domain name, for which we fetch the name servers repeatedly. The frequent update is forced by the use of small TTL values for the NS or A records. Based on our analysis, the top five domains for which we update our database repeatedly, includes *akamai.net*, *gslb.com*, *weather.com*, *facebook.com*, and *adnxs.com*. The domain names also highlight their popularity within our autonomous system, as these domain queries would form a good proportion of all outgoing packets.

The data presented above identifies frequently requested domain names. Using such domain names helps reduce the latency of query verification. By white-listing IP addresses of the name servers for known domains, we can quickly verify several queries without the need for frequent database update. However, such an analysis can also reveal rogue domains which exhibit domain fast fluxing behavior.

6. Determining False Negatives

To assess the ability of the MiND system to identify indirection anomalies, we use simulated malicious DNS queries. To generate misdirected packets from our network, we infect five machines with a custom program designed to periodically generate DNS queries for well-known domain names. For instance, an infected host generates DNS type A requests with the queried host name as *qNum.google.com* where *qNum* is the query number sent out by the particular stub resolver. The destination name server IP address for each stub resolver was assigned arbitrarily (e.g. “6.6.6.6”) taking care that the destination IP address did not belong to the queried domain’s autonomous system. The time period for querying was set to one minute. The queried domain names assigned to other hosts include *face-*

book.com, *citibank.com*, *yahoo.com* and *boa.com*. All queries were recursive in nature.

Using our tool, we log *all* simulated malicious queries generated by stub resolvers in our network. We confirm the source of malformed queries by examining the domain query names and IP addresses assigned to our test hosts.

7. Effect of Resolver Cache

We use the offline traces to determine the cache hit ratio for the campus recursive resolver. Cache hit ratio refers to the percentage of records, for which the answer could be found in the local resolver cache and hence immediately returned to the stub resolver. The cache hit ratio at the recursive resolver outlines the efficiency of the MiND system. A higher cache hit ratio would imply less burden on the MiND system, and thereby lower latencies for query validation. We determine the cache hit ratio by observing the number of queries directed by local stub resolvers towards the recursive resolvers and thus computing the fraction of DNS requests generated by the recursive resolver, to the name servers outside our autonomous system. As observed from figure 7(b), the average cache hit ratio is about 80%. This implies that the MiND system, which sits at the campus edge, verifies the direct correctness of the remaining 20% of the queries.

CHAPTER III

DETECTING ALGORITHMICALLY GENERATED MALICIOUS DOMAIN NAMES

A. Introduction

Recent botnets such as Conficker, Kraken and Torpig have brought in vogue a new method for botnet operators to control their bots: DNS “domain fluxing”. In this method, each bot algorithmically generates a large set of domain names and queries each of them until one of them is resolved and then the bot contacts the corresponding IP-address obtained that is typically used to host the command-and-control (C&C) server. Besides for command-and-control, spammers also routinely generate random domain names in order to avoid detection. For instance, spammers advertise randomly generated domain names in their spam emails to avoid detection by regular expression based domain blacklists that maintain signatures for recently ‘spamvertised’ domain names.

The botnets that have used random domain name generation vary widely in the random word generation algorithm as well as the way it is seeded. For instance, Conficker-A [31] bots generate 250 domains every three hours while using the current date and time at UTC (in seconds) as the seed, which in turn is obtained by sending empty HTTP GET queries to a few legitimate sites such as *google.com, baidu.com, answers.com etc.* This way, all bots would generate the same domain names every day. In order to make it harder for a security vendor to pre-register the domain names, the next version, Conficker-C [32] increased the number of randomly generated domain names per bot to 50K. Torpig [6,33] bots employ an interesting trick where the seed for the random string generator is based on one of the most popular trending topics in Twitter. Kraken employs a much more sophisticated random word generator and constructs English-language alike words with properly matched vowels and consonants. Moreover, the randomly generated word is combined with a suffix chosen randomly from a pool of common English nouns, verbs, adjective and adverb suffixes, such as -able, -dom, -hood, -ment, -ship, or -ly.

From the point of view of botnet owner, the economics work out quite well. They only have to register one or a few domains out of the several domains that each bot would query every day. Whereas, security vendors would have to pre-register *all* the domains that a bot queries every day, even before the botnet owner registers them. In all the cases above, the security vendors had to reverse engineer the bot executable to derive the exact algorithm being used for generating domain names. In some cases, their algorithm would predict domains successfully until the botnet owner would patch all his bots with a re-purposed executable with a different domain generation algorithm [6].

We argue that reverse engineering of botnet executables is resource- and time-intensive and precious time may be lost before the domain generation algorithm is cracked and consequently before such domain name queries generated by bots are detected. In this regards, we raise the following question: *can we detect algorithmically generated domain names while monitoring DNS traffic even when a reverse engineered domain generation algorithm may not be available?*

Hence, we propose a methodology that analyzes DNS traffic to detect *if* and *when* domain names are being generated algorithmically as a line of first defense. In this regards, our proposed methodology can point to the presence of bots within a network and the network administrator can disconnect bots from their C&C server by filtering out DNS queries to such algorithmically generated domain names.

Our proposed methodology is based on the following observation: current botnets do not use well formed and pronounceable language words since the likelihood that such a word is already registered at a domain registrar is very high; which could be self-defeating as the botnet owner would then not be able to control his bots. In turn this means that such algorithmically generated domain names can be expected to exhibit characteristics vastly different from legitimate domain names. Hence, we develop metrics using techniques from signal detection theory and statistical learning which can detect algorithmically generated domain names that may be generated via a myriad of techniques: (i) those generated via pseudo-random string generation algorithms as well as (ii) dictionary-based generators, for instance the one used by Kraken [34, 35] as well as a publicly available tool, Kwyjibo [36] which can generate words that are pronounceable yet not in the english dictionary.

Our method of detection comprises of two parts. First, we propose several ways to group together DNS queries: (i) either by the Top Level Domain (TLD) they all correspond to or; (ii) the IP-address that they are mapped to or; (iii) the connected component that they belong to, as determined via connected component analysis of the IP-domain bipartite graph. Second, for each such group, we compute metrics that characterize the distribution of the alphanumeric characters or bigrams (two consecutive alphanumeric characters) within the set of domain names. Specifically, we propose the following metrics to quickly differentiate a set of legitimate domain names from malicious ones: (i) Information entropy of the distribution of alphanumerics (unigrams and bigrams) within a group of domains; (ii) Jaccard index to compare the set of bigrams between a malicious domain name with good domains and; (iii) Edit-distance which measures the number of character changes needed to convert one domain name to another.

We apply our methodology to a variety of data sets. First, we obtain a set of legitimate domain names via reverse DNS crawl of the entire IPv4 address space. Next, we obtain a set of malicious domain names as generated by Conficker, Kraken and Torpig as well as model a much more sophisticated domain name generation algorithm: Kwyjibo [36]. Finally, we apply our methodology to one day of network traffic from one of the largest Tier-1 ISPs in Asia and South America and show how we can detect Conficker as well as a botnet hitherto unknown, which we call *Mjuyh* (details in Section E).

Our extensive experiments allow us to characterize the effectiveness of each metric in detecting algorithmically generated domain names in different attack scenarios. We model different attack intensities as number of domain names that an algorithm generates. For instance, in the extreme scenario that a botnet generates 50 domains mapped to the same TLD, we show that K-L divergence over unigrams achieves 100% detection accuracy albeit at 15% false positive rate (legitimate domain groups classified as algorithmic). We show how our detection improves significantly with much lower false positives as the number of words generated per TLD increases, *e.g.,* when 200 domains are generated per TLD, then Edit distance achieves 100% detection accuracy with 8% false positives and when 500 domains are generated per TLD, Jaccard Index achieves 100% detection with 0% false positives.

Finally, our methodology of grouping together domains via connected components allows us to detect not only "domain fluxing" but also if it was used in combination with "IP fluxing". Moreover, computing the metrics over components yields better and faster detection than other grouping methods. Intuitively, even if botnets were to generate random words and combine them with multiple TLDs in order to spread the domain names thus generated (potentially to evade detection), as long as they map these domains such that at least one IP-address is shared in common, then they reveal a group structure that can be exploited by our methodology for quick detection. We show that per-component analysis detects 26.32% more IP addresses than using per-IP analysis and 16.13% more hostnames than using per-domain analysis when we applied our methodology to detect Conficker in a Tier-1 ISP trace.

B. Related Work

Characteristics, such as IP addresses, whois records and lexical features of phishing and non-phishing URLs have been analyzed by McGrath and Gupta [37]. They observed that the different URLs exhibited different alphabet distributions. Our work builds on this earlier work and develops techniques for identifying domains employing algorithmically generated names, potentially for “domain fluxing”. Ma, et al [38], employ statistical learning techniques based on lexical features (length of domain names, host names, number of dots in the URL *etc.*) and other features of URLs to automatically determine if a URL is malicious, *i.e.*, used for phishing or advertising spam. While they classify each URL independently, our work is focused on classifying a group of URLs as algorithmically generated or not, solely by making use of the set of alphanumeric characters used. In addition, we experimentally compare against their lexical features in Section E and show that our alphanumeric distribution based features can detect algorithmically generated domain names with lower false positives than lexical features. Overall, we consider our work as complimentary and synergistic to the approach in [38].

With reference to the practice of “IP fast fluxing”, *e.g.*, where the botnet owner constantly keeps changing the IP-addresses mapped to a C&C server, [39] implements a detection mechanism based on passive DNS traffic analysis. In our work, we present a methodology to detect cases where botnet owners may use a combination of both domain fluxing with IP fluxing, by having bots query a series of domain names and at the same time map a few of those domain names to an evolving set of IP-addresses. Also earlier papers [40, 41] have analyzed the inner-working of IP fast flux networks for hiding spam and scam infrastructure. With regards to botnet detection, [2, 3] perform correlation of network activity in time and space at campus network edges, and Xie et al in [42] focus on detecting spamming botnets by developing regular expression based signatures from a dataset of spam URLs .

We find that graph analysis of IP addresses and domain names embedded in DNS queries and replies reveal interesting macro relationships between different entities and enable identification of bot networks (Conficker) that seemed to span many domains and TLDs. With reference to graph based analysis, [43] utilizes rapid changes in user-bot graphs structure to detect botnet accounts.

Statistical and learning techniques have been employed by various studies for prediction [44–46]. We employed results from detection theory in designing our strategies for classification [47, 48].

Several studies have looked at understanding and reverse-engineering the inner workings of botnets [6, 34, 35, 49–51]. Botlab has carried out an extensive analysis of several bot networks through active participation [52] and provided us with many example datasets for malicious domains.

C. Detection Metrics

In this section, we present our detection methodology that is based on computing the distribution of alphanumeric characters for groups of domains. First, we motivate our metrics by showing how algorithmically generated domain names differ from legitimate ones in terms of distribution of alphanumeric characters. Next, we present our three metrics, namely Kullback-Leibler (K-L) distance, Jaccard Index (JI) measure and Edit distance. Finally, in Section D we present the methodology to group domain names.

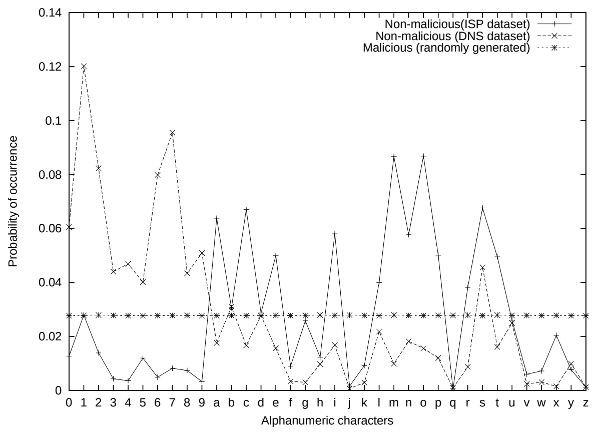
1. Data Sets

We first describe the data sets and how we obtained them: (i) **Non-malicious ISP Dataset:** We use network traffic trace collected from across 100+ router links at a Tier-1 ISP in Asia. The trace is one day long and provides details of DNS requests and corresponding replies. There are about 270,000 DNS name server replies. (ii) **Non-malicious DNS Dataset:** We performed a reverse DNS crawl of the entire IPv4 address space to obtain a list of domain names and their corresponding IP-addresses. We further divided this data set in to several parts, each comprising of domains which had 500, 200, 100 and 50 domain labels. The

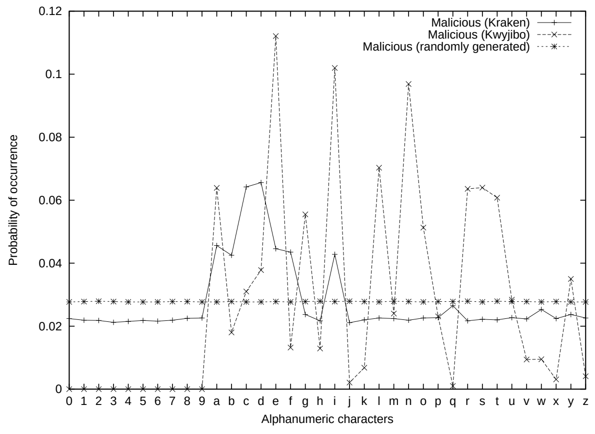
DNS Dataset is considered as non-malicious for the following reasons. Botnets may own only a limited number of IP addresses. Based on our study, we find that a DNS PTR request maps an IP address to only one domain name. The dataset thus obtained will contain very few malicious domain names per analyzed group. In the event that the bots exhibit IP fluxing, it is noteworthy that the botnet owners cannot change the PTR DNS mapping for IP addresses not owned. Although, the malicious name servers may point to any IP address. (iii) **Malicious datasets:** We obtained the list of domain names that were known to have been generated by recent Botnets: Conficker [31, 32], Torpig [6] and Kraken [34, 35]. As described earlier in the Introduction, Kraken exhibits the most sophisticated domain generator by carefully matching the frequency of occurrence of vowels and consonants as well as concatenating the resulting word with common suffixes in the end such as -able, -dom, *etc.* (iv) **Kwyjibo**: We model a much more sophisticated algorithmic domain name generation algorithm by using a publicly available tool, Kwyjibo [36] which generates domain names that are pronounceable yet not in the English language dictionary and hence much more likely to be available for registration at a domain registrar. The algorithm uses a syllable generator, where they first learn the frequency of one syllable following another in words in English dictionary and then automatically generate pronounceable words by modeling it as a Markov process.

2. Motivation

Our detection methodology is based on the observation that algorithmically generated domains differ significantly from legitimate (human) generated ones in terms of the distribution of alphanumeric characters. Figure 8 shows the distribution of alphanumeric characters, defined as the set of English alphabets (a-z) and digits (0-9) for both legitimate as well



(a) Non-malicious and malicious domains.



(b) Only malicious entities

Fig. 8. Probability distributions of malicious and non-malicious domains

as malicious domains [1]. We derive the following points: (i) First, note that both the non-malicious data sets exhibit a non-uniform frequency distribution, *e.g.*, letters ‘m’ and ‘o’ appear most frequently in the non-malicious ISP data set whereas the letter ‘s’ appears most frequently in the non-malicious DNS data set. (ii) Even the most sophisticated algorithmic domain generator seen in the wild for Kraken botnet has a fairly uniform distribution, albeit with higher frequencies at the vowels: ‘a’, ‘e’ and ‘i’. (iii) If botnets of future were to evolve and construct words that are pronounceable yet not in the dictionary, then they would not exhibit a uniform distribution as expected. For instance, Kwyjibo exhibits higher frequencies at alphabets, ‘e’, ‘g’, ‘i’, ‘l’, ‘n’, *etc.* In this regards, techniques that are based on only the distribution of unigrams (single alphanumeric characters) may not be sufficient, as we will show through the rest of this section.

[1]Even though domain names may contain characters such as ‘-’, we currently limit our study to alphanumeric characters only.

3. Metrics for Anomaly Detection

The K-L(Kullback-Leibler) divergence metric is a non-symmetric measure of ”distance“ between two probability distributions. The divergence (or distance) between two discretized distributions P and Q is given by: $D_{KL}(P||Q) = \sum_{i=1}^{n} P(i)log\frac{P(i)}{Q(i)}$,

where n is the number of possible values for a discrete random variable. The probability distribution *P* represents the test distribution and the distribution *Q* represents the base distribution from which the metric is computed.

Since the K-L measure is asymmetric, we use a symmetric form of the metric, which helps us deal with the possibility of singular probabilities in either distribution. The modified K-L metric is computed using the formula: $D_{sym}(PQ) = \frac{1}{2}(D_{KL}(P||Q)+D_{KL}(Q||P))$.

Given a test distribution $\underline{q}$ computed for the domain to be tested, and non-malicious and malicious probability distribution over the alphanumerics as $\underline{g}$ and $\underline{b}$ respectively, we characterize the distribution as malicious or not via the following optimal classifier (for proof see section H):

$$D_{sym}(\underline{qb}) - D_{sym}(\underline{qg}) \underset{b}{\overset{g}{\gtrless}} 0 \tag{3.1}$$

For the test distribution $\underline{q}$ to be classified as non-malicious, we expect $D_{sym}(\underline{qg})$ to be less than $D_{sym}(\underline{qb})$. However, if $D_{sym}(\underline{qg})$ is greater than $D_{sym}(\underline{qb})$, the distribution is classified as malicious.

a. Measuring K-L Divergence with Unigrams

The first metric we design measures the K-L divergence of unigrams by considering all domain names that belong to the same group, *e.g.* all domains that map to the same IP-address or those that belong to the same top-level domain. We postpone discussion of groups to Section D. Given a group of domains for which we want to establish whether they were generated algorithmically or not, we first compute the distribution of alphanumeric characters to obtain the test distribution. Next, we compute the K-L divergence with a good distribution obtained from the non-malicious data sets (ISP or DNS crawl) and a malicious distribution obtained by modeling a botnet that generates alphanumerics uniformly. As expected, a simple unigram based technique may not suffice, especially to detect Kraken or Kwyjibo generated domains. Hence, we consider bigrams in our next metric.

b. Measuring K-L Divergence with Bigrams

A simple obfuscation technique that can be employed by algorithmically generated malicious domain names could be to generate domain names by using the same distribution of alphanumerics as commonly seen for legitimate domains. Hence, in our next metric, we consider distribution of bigrams, *i.e.*, two consecutive characters. Using such an additional measure in collaboration with the previously proposed metric, limits the flexibility for the attacker to compose domain names.

Analogous to the case above, given a group of domains, we extract the set of bigrams present in it to form a bigram distribution. Note that for the set of alphanumeric characters that we consider $[a\text{-}z, 0\text{-}9]$, the total number of bigrams possible are 36×36, *i.e.*, 1,296. Our improved hypothesis now involves validating a given test bigram distribution against the bigram distribution of non-malicious and malicious domain labels. We use the database of non-malicious words to determine a non-malicious probability distribution. For a sample malicious distribution, we generate bigrams randomly. Here as well, we use K-L divergence over the bigram distribution to determine if a test distribution is malicious or legitimate.

c. Using Jaccard Index between Bigrams

We present the second metric to measure the similarity between a known set of components and a test distribution, namely the *Jaccard index* measure. The metric is defined as

$$JI = \frac{A \cap B}{A \cup B}$$

where, *A* and *B* each represent the set of random variables. For our particular case, the set comprises of bigrams that compose a domain label or a hostname. Note that Jaccard index (JI) measure based on bigrams is a commonly used technique for web search engine spell-checking [53].

The core motivation behind using the JI measure is same as that for K-L divergence. We expect that bigrams occurring in randomized (or malicious) hostnames to be mostly different when compared with the set of non-malicious bigrams. To elaborate, we construct a database of bigrams which point to lists of non-malicious words, domain labels or hostnames, as the case may be. Now for each sub-domain present in a test set, we determine all non-malicious words that contain at least 75% of the bigrams present in the test word. Such a threshold helps us discard words with less similarity. However, longer test words may implicitly satisfy this criteria and may yield ambiguous JI value. As observed in section E, the word sizes for 95% of non-malicious words do not exceed 24 characters, and hence we divide all test words into units of 24 character strings. Figure 9 presents the CDF of domain label sizes as observed in our DNS PTR dataset (described in section E).

Calculating the JI measure is best explained with an example. Considering a randomized hostname such as *ickoxjsov.botnet.com*, we determine the JI value of the domain label *ickoxjsov* by first computing all bigrams (eight, in this case). Next, we examine each bigram's queue of non-malicious domain labels, and short list words with at least 75% of bigrams, *i.e.*, six of the eight bigrams. Words satisfying this criteria may include thequ*ick*brownf*oxj*ump*sov*erthelazydog (35 bigrams). However, such a word still has a low JI value owing to the large number of bigrams in it. Therefore, the JI value is thus computed as 6/(8 + 35 - 6) = 0.16. The low value indicates that the randomized test word does not match too well with the word from the non-malicious bigram database.

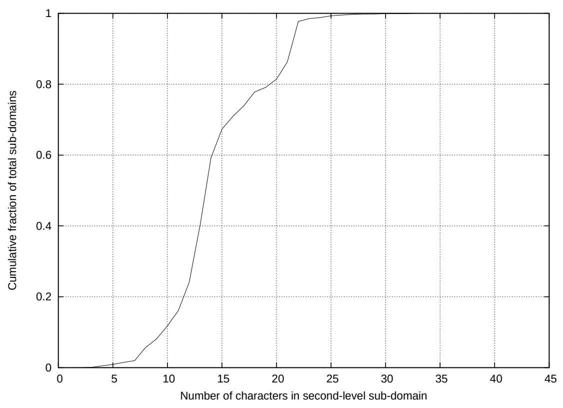


Fig. 9. CDF of domain label sizes for DNS PTR dataset.

The JI measure is thus computed for the remaining words. The test words might comprise of a large number of bigrams and therefore do not always ensure a high JI value. We compute the JI measure using the equation described above and average it for all test words belonging to a particular group being analyzed. The averaged JI value for a non-malicious domain is expected to be higher than those for malicious groups.

As observed via our experiments in Section E, the JI measure is better at determining domain based anomalies. However, it is also computationally expensive as the database of non-malicious bigrams needs to be maintained in the memory. Also, classifying a non-malicious hosts will take more CPU cycles as we would obtain and compare a large set of words consisting of test word's bigrams. Section G examines the computational complexity of various metrics that we use.

d. Edit Distance

Note that the two metrics described earlier, rely on definition of a "good" distribution (K-L divergence) or database (JI measure). Hence, we define a third metric, Edit distance, which classifies a group of domains as malicious or legitimate by only looking at the domains within the group, and is hence not reliant on definition of a database or distribution. The Edit distance between two strings represents an integral value identifying the number of transformations required to transform one string to another. It is a symmetric measure and provides a measure of intra-domain entropy. The type of eligible transformations are addition, deletion, and modification. For instance, to convert the word *cat* to *dog*, the edit distance is three as it requires all three characters to be replaced. With reference to determining anomalous domains, we expect that all domain labels (or hostnames) which are randomized, will, on an average, have higher edit distance value. We use the Levenshtein edit distance dynamic algorithm for determining anomalies. The algorithm for computing the Levenshtein edit distance has been shown in Algorithm 1 [53].

Algorithm 1 Dynamic programming algorithm for finding the edit distance

```
EditDist($s_1$,$s_2$)
 1. int m[i,j] = 0
 2. for i $\leftarrow$ 1 to $|s_1|$
 3. do m[i,0] = i
 4. for j $\leftarrow$ 1 to $|s_2|$
 5. do m[0,j] = j
 6. for i $\leftarrow$ 1 to $|s_1|$
 7. do for j $\leftarrow$ 1 to $|s_2|$
 8.    do m[i,j] = min{m[[i-1,j-1] + if($s_1$[i] = $s_2$[j]) then 0 else 1 fi,
 9.            m[i-1,j] + 1
10.            m[i,j-1] + 1}
11. return m[$|s_1|$,$|s_2|$]
```

D. Grouping Domain Names

In this section, we present ways by which we group together domain names in order to compute metrics that were defined in Section C earlier.

1. Per-domain Analysis

Note that several botnets use several second-level domain names to generate algorithmic sub-domains. Hence, one way by which we group together domain names is via the second-level domain name. The intention is that if we begin seeing several algorithmically generated domain names being queried such that all of them correspond to the same second-level domain, then this may be reflective of a few favorite domains being exploited. Hence for all sub-domains, *e.g.*, *abc.examplesite.org*, *def.examplesite.org*, *etc.*, that have the same second-level domain name *examplesite.org*, we compute all the metrics over the alphanumeric characters and bigrams of the corresponding domain labels. Since domain fluxing involves a botnet generating a large number of domain names, we consider only domains which contain a sufficient number of third-level domain labels, *e.g.*, 50, 100, 200 and 500 sub-domains.

2. Per-IP Analysis

As a second method of grouping, we consider all domains that are mapped to the same IP-address. This would be reflective of a scenario where a botnet has registered several of the algorithmic domain names to the same IP-address of a command-and-control server. Determining if an IP address is mapped to several such malicious domains is useful as such an IP-address or its corresponding prefix can be quickly blacklisted in order to sever the traffic between a command-and-control server and its bots. We use the dataset from a Tier-1 ISP to determine all IP-addresses which have multiple hostnames mapped to it. For a large number of hostnames representing one IP address, we explore the above described metrics, and thus identify whether the IP address is malicious or not.

A few botnets have taken the idea of domain fluxing further and generate names that span multiple TLDs, *e.g.*, Conficker-C generates domain names in 110 TLDs. At the same time domain fluxing can be combined with another technique, namely "IP fluxing" [39] where each domain name is mapped to an ever changing set of IP-addresses in an attempt to evade IP blacklists. Indeed, a combination of the two is even harder to detect. Hence, we propose the third method for grouping domain names into connected components.

We first construct a bipartite graph G with IP-addresses on one side and domain names on the other. An edge is constructed between a domain name and an IP-address if that IP-address was ever returned as one of the responses in a DNS query. When multiple IP addresses are returned, we draw edges between all the returned IP addresses and the queried host name.

First, we determine the connected components of the bipartite graph G, where a connected component is defined as one which does not have any edges with any other components. Next, we compute the various metrics (K-L divergence for unigrams and bigrams, JI measure for bigrams, Edit distance) for each component by considering all the domain names within a component.

Component extraction separates the IP-domain graph into components which can be classified in to the following classes: (i) **IP fan**: these have one IP-address which is mapped to several domain names. Besides the case where one IP-address is mapped to several algorithmic domains, there are several legitimate scenarios possible. First, this class could include domain hosting services where one IP-address is used to provide hosting to several domains, *e.g.* Google Sites, *etc.* Other examples could be mail relay service where one mail server is used to provide mail relay for several MX domains. Another example could be when domain registrars provide domain parking services, *i.e.*, someone can purchase a domain name while asking the registrar to host it temporarily. (ii) **Domain fan**: these consist of one domain name connected to multiple IPs. An IP-fluxing domain would be belong to this category. This class will contain components belonging to the legitimate content providers such as Google, Yahoo!, *etc.* (iii)**Many-to-many component**: these are components that have multiple IP addresses and multiple domain names, *e.g.*, Content Distribution Networks (CDNs) such as Akamai. Components of this type include the Conficker-C botnet group where a domain name maps to multiple IP addresses and the corresponding C&C server is shared by the fluxing domains.

In section F, we briefly explain the classification algorithm that we use to classify test components as malicious or not.

E. Results

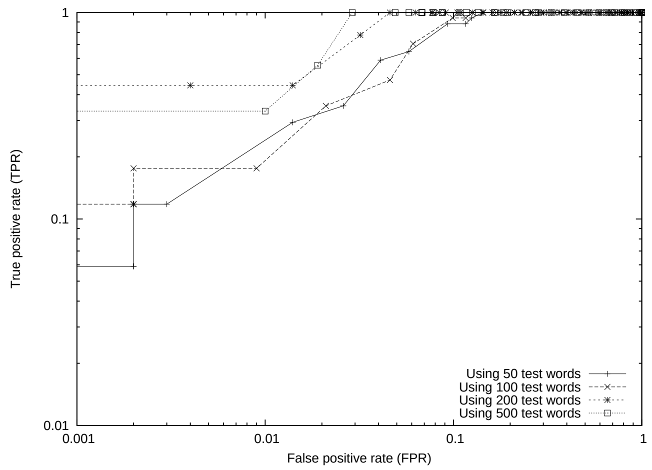
In this section, we present results of employing various metrics across different groups, as described in section C and D. We briefly describe the data set used for each experiment.

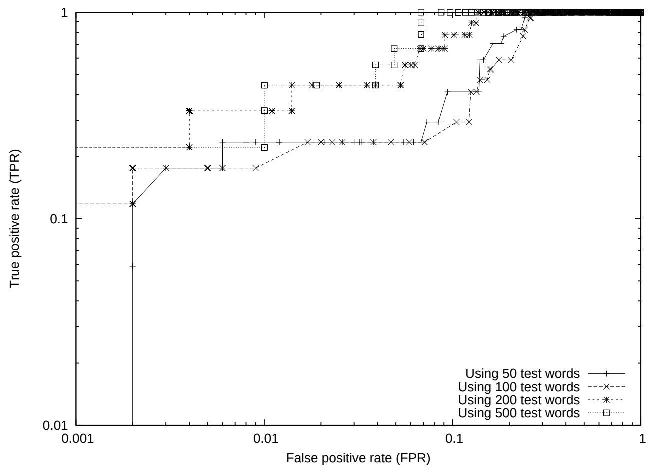
With all our experiments, we present the results based on the consideration of increasing number of domain labels. In general, we observe that using a larger test data set yields better results.

1. Per-domain Analysis

a. Data Set

The analysis in this sub-section is based only on the domain labels belonging to a domain. The non-malicious distribution $\underline{g}$ may be obtained from various sources. For our analysis, we use a database of DNS PTR records corresponding to all IPv4 addresses. The database contains 659 second-level domains with at least 50 third-level sub-domains, while there are 103 second-level domains with at least 500 third-level sub-domains. From the database, we





(a) K-L metric with unigram distribution (Per-domain). (b) K-L metric with bigram distribution (Per-domain).

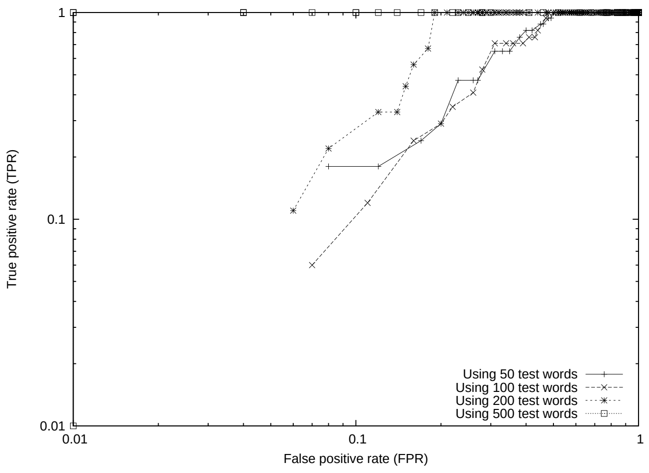
Fig. 10. ROC curves for Per-domain analysis (K-L metric's unigram and bigram evaluation)

extract all second-level domains which have at least 50 third-level sub-domains. All third-level domain labels corresponding to such domains are used to generate the distribution $\underline{g}$. For instance, a second-level domain such as *university.edu* may have many third-level domain labels such as *physics*, *cse*, *humanities* etc. We use all such labels that belong to trusted domains, for determining $\underline{g}$.

To generate a malicious base distribution $\underline{b}$, we randomly generate as many characters as present in the non-malicious distribution. We use domain labels belonging to well-known malware based domains identified by Botlab, and also a publicly available webspam database, as malicious domains [54, 55] for verification using our metrics. Botlab provides us with various domains used by Kraken, Pushdo, Storm, MegaD, and Srizbi [54]. For *per-domain* analysis, the test words used are the third-level domain labels.

Figure 8 shows how malicious/non-malicious distributions appear for the DNS PTR dataset as well as the ISP dataset described in the following sections.

We will present the results for all the four measures described earlier, for domain-based analysis. In later sections, we will only present data from one of the measures for brevity.


(a) Jaccard measure for bigrams (Per-domain).

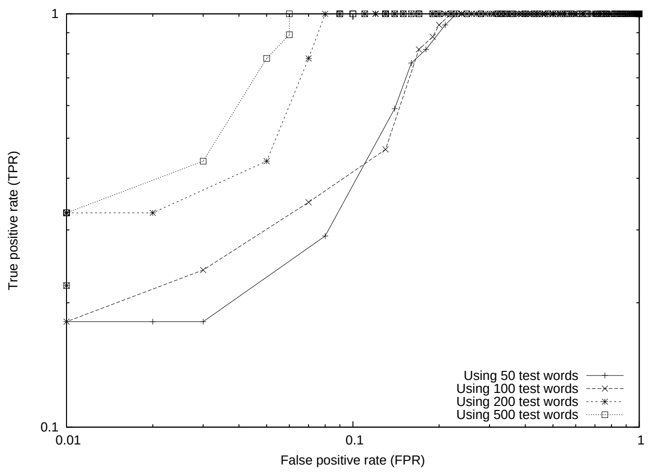

(b) Edit distance (Per-domain).

Fig. 11. ROC curves for Per-domain analysis (Jaccard Index and Edit Distance evaluation)

b. K-L Divergence with Unigram Distribution

We measure the symmetric K-L distance metric from the test domain to the malicious/non-malicious alphabet distributions. We classify the test domain as malicious or non-malicious based on equation (10) in section H. Figure 10(a) shows the results from our experiment presented as an ROC (Receiver Operating Characteristic) curve which evaluates detection rates against the false positive rates.

The figure shows that the different sizes of test data sets produce relatively different results. The area under the ROC is a measure of the goodness of the metric. We observe that with 200 or 500 domain labels, we cover a relatively greater area, implying that using many domain labels helps obtain accurate results. For example, using 500 labels, we obtain 100% detection rate with only 2.5% false positive rate. Note that with a larger data set, we indeed expect higher true positive rates for small false positive rates, as larger samples will stabilize the evaluated metrics.

The number of domain labels required for accurate detection corresponds to the latency of accurately classifying a previously unseen domain. The results suggest that a domain-fluxing domain can be accurately characterized by the time it generates around 500 names.

c. K-L Divergence with Bigram Distribution

Figure 10(b) presents the results of employing K-L distance metric over bigram distributions. We observe again that using 200 or 500 domain labels does better than using smaller number of labels, with 500 labels doing the best. Experiments with 50/100 domain labels yield similar results.

We note that the performance with unigram distributions is slightly better than using bigram distributions. However, when botnets employ counter measures to our techniques, the bigram distributions may provide better defense compared to unigram distributions as they require more effort to match the good distribution ($\underline{g}$).

d. Jaccard Measure of Bigrams

The Jaccard Index measure does significantly better in comparison to the previous metrics. From figure 11(a), it is evident that using 500 domain labels gives us a clear separation for classification of test domains (and hence an area of 1). Using 50 or 100 labels is fairly equivalent with 200 labels doing comparatively better. The JI measure produces higher false positives for smaller number of domains (50/100/200) than K-L distance measures.

e. Edit Distance of Domain Labels

Figure 11(b) shows the performance using edit distance as the evaluation metric. The detection rate for 50/100 test words reaches 1 only for high false positive rates, indicating that a larger test word set should be used. For 200/500 domain labels, 100% detection rate is achieved at false positive rates of 5-7%.

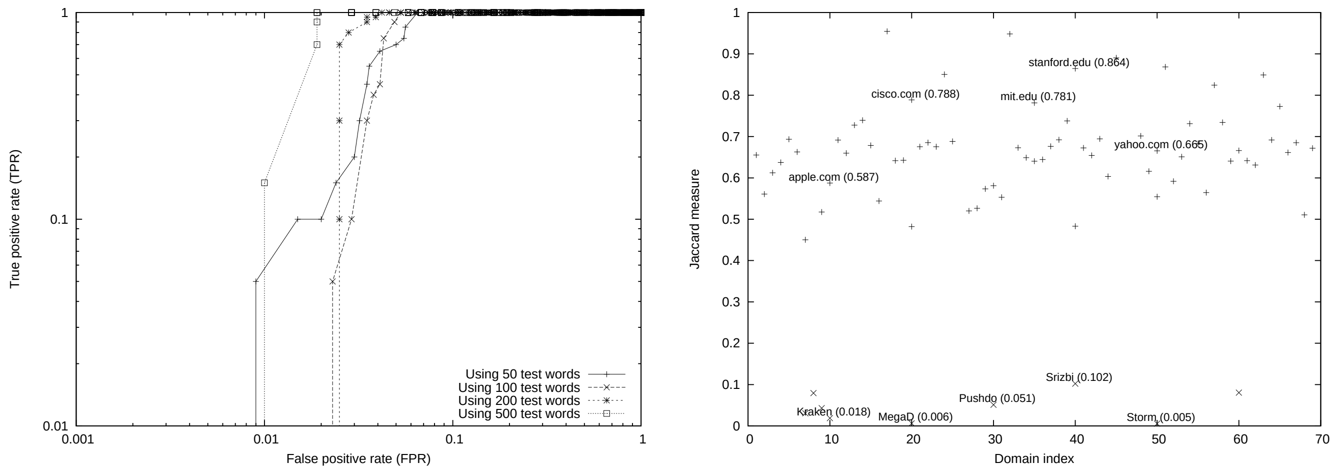


Fig. 12. (a) ROC curve : K-L metric with unigram distribution (Kwyjibo). (b) Scatter plot with Jaccard Index for bigrams (500 test words).

f. Kwyjibo Domain Label Analysis

Kwyjibo is a tool to generate random words which can be used as domain labels [36]. The generated words are seemingly closer to pronounceable words of the english language, in addition to being random. Thus many such words can be created in a short time. We anticipate that such a tool can be used by attackers to generate domain labels or domain names quickly with the aim of defeating our scheme. Therefore, we analyze Kwyjibo based words, considering them as domain labels belonging to a particular domain.

The names generated by Kwyjibo tool could be accurately characterized by our measures given sufficient names. Example results are presented in Fig. 12(a) with K-L distances over unigram distributions. From figure 12(a), we observe that verification with unigram frequency can lead to a high detection rate with very low false positive rate. Again, the performance using 500 labels is the best. We also observe a very steep rise in detection rates for all the cases. The Kwyjibo domains could be accurately characterized with false positive rates of 6% or less.

The initial detection rate for Kwyjibo is low as compared to the per-domain analysis.

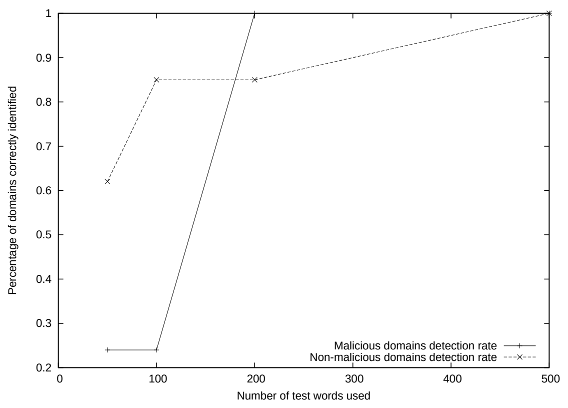


Fig. 13. Illustrating benefits of progressive demarcation with JI measure.

This is because the presence of highly probable non-malicious unigrams in Kwyjibo based domains makes detection difficult at lower false positive rates. The results with other measures (K-L distance over bigram distributions, JI and edit distances) were similar: kwyjibo domains could be accurately characterized at false positive rates in the range of 10-12%, but detection rates were nearly zero at false positive rates of 10% or less.

The scatter plot presented in Fig. 12(b) indicates the clear separation obtained between non-malicious and malicious domains. The plot represents the Jaccard measure using 500 test words. We highlight the detection of botnet based malicious domains such as *Kraken*, *MegaD*, *Pushdo*, *Srizbi*, and *Storm*. A few well-known non-malicious domains such as *apple.com*, *cisco.com*, *stanford.edu*, *mit.edu*, and *yahoo.com* have also been indicated for comparison purposes.

g. Progressive Demarcation

The earlier results have showed that good detection rates can be obtained at low false positive rates once we have 500 or more hostnames of a test domain. As discussed earlier, the number of hostnames required for our analysis corresponds to latency of accurately characterizing a previously unseen domain. During our experiments, not all the test domains required 500 hostnames for accurate characterization since the distributions were either very close to the good distribution $\underline{g}$ or bad distribution $\underline{b}$. These test domains could be characterized with a smaller latency (or smaller number of hostnames).

In order to reduce the latency for such domains, we tried an experiment at progressive demarcation or characterization of the test domains. Intuitively, the idea is to draw two thresholds above one there are clearly good domains, below the second threshold there are clearly bad domains and the domains between the two thresholds require more data (or hostnames) for accurate characterization. These thresholds are progressively brought closer (or made tighter) as more hostnames become available, allowing more domains to be accurately characterized until we get 500 or more hostnames for each domain. The results of such an experiment using the JI measure are shown in Fig. 13.

We establish the lower bound using the formula $\mu_b + \sigma_b$ where μ_b is the mean of JI values observed for bad or malicious domains and σ_b is the standard deviation. Similarly, the upper bound is obtained using the expression $\mu_g - \sigma_g$ where the subscript g implies good domains. Figure 13 shows the detection rate for the considered domains. We see a monotonically increasing detection rate for both good and bad domains. It is observed that 85% of bad domains could be so characterized accurately with only 100 hostnames while only about 23% of good domains can be so characterized with 100 hostnames. In addition, our experiments indicate that only a small percentage of domains require 200 or more hostnames for their characterization.

2. Per-IP Analysis

a. Data Set

Here, we present the evaluation of domain names that map to an IP address. For analyzing the per-IP group, for all hostnames mapping to an IP-address, we use the domain labels except the top-level domain TLD as the test word. For instance, for hostnames *physics.university.edu* and *cse.university.edu* mapping to an IP address, say *6.6.6.6*, we use *physicsuniversity* and *cseuniversity* as test words. However, we only consider IP addresses with at least 50 hostnames mapping to it. We found 341 such IP addresses, of which 53 were found to be malicious, and 288 were considered non-malicious. The data is obtained from DNS traces of a Tier-1 ISP in Asia.

Many hostnames may map to the same IP address. Such a mapping holds for botnets or other malicious entities utilizing a large set of hostnames mapping to fewer C&C(Command and Control) servers. It may also be valid for legitimate internet service such as for Content Delivery Networks (CDNs). We first classify the IPs obtained into two classes of malicious and non-malicious IPs. The classification is done based on manual checking, using blacklists, or publicly available Web of Trust information [56]. We manually confirm the presence of *Conficker* based IP addresses and domain names [32]. The ground truth thus obtained may be used to verify the accuracy of classification. Figure 8 shows the distribution of non-malicious test words and the randomized distribution is generated as described previously.

We discuss the results of per-IP analysis below. For the sake of brevity, we present results based on K-L distances of bigram distributions only. Summary of results from other metrics is also provided.

The ROC curve for K-L metric shows that bigram distribution can be effective in accurately characterizing the domain names belonging to different IP addresses. We observe a very clear separation between malicious and non-malicious IPs with 500, and even with 200 test words. With a low false positive rate of 1%, high detection rates of 90% or more are obtained with 100 or greater number of test words.

The bigram analysis was found to perform better than unigram distributions. The per-IP bigram analysis performed better than per-domain bigram analysis. We believe that the bigrams obtained from the ISP dataset provide a comprehensive non-malicious distribution.

The second-level domain labels also assist in discarding false anomalies, and therefore provide better accuracy.

The JI measure performed very well, even for small set of test words. The area covered under the ROC curve was 1 for 200/500 test words. For the experiment with 100 test words, we achieved the detection rates of 100% with false positive rate of only 2%.

Edit distance with domains mapping to an IP, results in a good performance in general. The experiments with 100 test words results in a low false positive rate of about 10% for a 100% detection rate. However for using only 50 test words, the detection rate reaches about 80% for a high false positive rate of 20%. Thus, we conclude that for per-IP based analysis, the JI measure performs relatively better than previous measures applied to this group. However, as highlighted in section G, the time complexity for computing jaccard index is higher.

3. Summary

For a larger set of test words, the relative order of efficacy of different measures decreases from JI, to edit distance to K-L distances over bigrams and unigrams. However, interestingly, we observe the exact opposite order when using a small set of test words. For instance, with 50 test words used for the per-domain analysis, the false positive rates at which we obtain 100% detection rates, are approximately 50% (JI), 20% (ED), 25% (K-L with bigram distribution), and 15% (K-L with unigram distribution). Even though the proof in section H indicates that K-L divergence is an optimal metric for classification, in practice, it does not hold as the proof is based on the assumption that it is equally likely to draw a test distribution from a good or a bad distribution.

Table II. Different types of component classes

Type of class	# of components	# of IP addresses	# of domain names	Types of components found
Many-to-many	440	11K	35K	Legitimate services (Google, Yahoo), CDNs, Cookie tracking, Mail service, Conficker botnet
IP fans	1.6K	1.6K	44K	Domain Parking, Adult content, Blogs, small websites
Domain fans	930	8.9K	9.3K	CDNs (Akamai), Ebay, Yahoo, Mjuyh botnet

Table III. Summary of interesting networks discovered through component analysis

Comp. type	#Comps.	#domains	#IPs
Conficker botnet	1	1.9K	19
Helldark botnet	1	28	5
Mjuyh botnet	1	121	1.2K
Misspelt Domains	5	215	17
Domain Parking	15	630	15
Adult content	4	349	13

F. Detection via Supervised Learning

As discussed in Section 3 immediately above, the relative merits of each measure vary depending, for instance, on the number of sub-domains present in a domain being tested. In this section, we formulate detection of malicious domains (algorithmically generated) as a supervised learning problem such that we can combine the benefits afforded by each measure while learning the relative weights of each measure during a training phase. We divide the one-day long trace from the South Asian Tier-1 ISP in to two halves such that the first one of 10 hours duration is used for training. We test the learnt model on the remainder of the trace from South Asian ISP as well as over a different trace from a Tier-1 ISP in South America. In this section, we use the grouping methodology of connected components, where all "domain name, response IP-address" pairs present during a time

Table IV. Domain names used by bots

Type of group	Domain names
Conficker botnet	vddxnvzqjks.ws gcvwknnxz.biz joftvvtvmx.org
Mjuyh bot	935c4fe[0-9a-z]+.6.mjuyh.com c2d026e[0-9a-z]+.6.mjuyh.com
Helldark Trojan	may.helldark.biz X0R.ircdevils.net www.BALDMANPOWER.ORG

window (either during training or test phases) are grouped in to connected components.

1. L1-regularized Linear Regression

We formulate the problem of classifying a component as malicious (algorithmically generated) or legitimate in a supervised learning setting as a linear regression or classification problem. We first label all domains within the components found in the training data set by querying against domain reputation sites such as McAfee Site Advisor [57] and Web of Trust [56] as well as by searching for the URLs on search-engines [58]. Next, we label a component as good or bad depending on a simple majority count, *i.e.*, if more than 50% of domains in a component are classified as malicious (adware, malware, spyware, *etc.*) by any of the reputation engines, then we label that component as malicious.

Define the set of features as F which includes the following metrics computed for each component: K-L distance on unigrams, JI measure on bigrams and Edit distance. Also define the set of Training examples as T and its size in terms of number of components as $|T|$. Further, define the output value for each component $y_i = 1$ if it was labeled malicious or $= 0$ if legitimate. We model the output value y_i for any component $i \in T$ as a linear weighted sum of the values attained by each feature where the weights are given by β_j for each feature $j \in F$: $y_i = \sum_{j \in F} \beta_j x_j + \beta_0$

In particular, we use the LASSO, also known as L1-regularized Linear Regression

[59], where an additional constraint on each feature allows us to obtain a model with lower test prediction errors than the non-regularized linear regression since some variables can be adaptively shrunk towards lower values. We use 10-fold cross validation to choose the value of the regularization parameter $\lambda \in [0\text{-}1]$ that provides the minimum training error (equation below) and then use that λ value in our tests:

$$arg \min_{\beta} \sum_{i=1}^{|T|} (y_i - \beta_0 - \sum_{j \in F} \beta_j x_j)^2 + \lambda \sum_{j \in F} |\beta_j|. \tag{3.2}$$

2. Results

First, note the various connected components present in the South Asian trace as classified in to three classes: IP fans, Domain fans, and Many-to-many components in Table II. During the training phase, while learning the LASSO model, we mark 128 components as good (these consist of CDNs, mail service providers, large networks such as Google) and one component belonging to the Conficker botnet as malicious. For each component, we compute the features of K-L divergence, Jaccard Index measure, and Edit distance. We train the regression model using glmnet tool [59] in statistical package R, and obtain the value for the regularization parameter λ as $1e-4$, that minimizes training error during the training phase. We then test the model on the remaining portion of the one day long trace. In this regard, our goal is to check if our regression model can not only detect Conficker botnet but whether it can also detect other malicious domain groups during the testing phase over the trace. During the testing stage, if a particular component is flagged as suspicious then we check against Web of Trust [56], McAfee Site Advisor [57] as well as via Whois queries, search engines, to ascertain the exact behavior of the component. Next, we explain the results of each of the classes individually.

On applying our model to the rest of the trace, 29 components (out of a total of 3K components) are classified as malicious, and we find 27 of them to be malicious after cross checking with external sources (Web of Trust, McAfee, *etc.*) while two components (99 domains) are false positives and comprise of Google and domains belonging to news blogs. Note that here we use a broad definition of malicious domains as those that could be used for any nefarious purposes on the web, *i.e.*, we do not necessarily restrict the definition to only include botnet domain generation algorithm. Out of the 27 components that were classified as malicious, one of them corresponds to the Conficker botnet, which is as expected since our training incorporated features learnt from Conficker. We next provide details on the remaining 26 components that were determined as malicious (see Table III).

Mjuyh Botnet: The most interesting discovery from our component analysis is that of another Botnet, which we call Mjuyh, since they use the domain name *mjuyh.com* (see Table IV). The fourth-level domain label is generated randomly and is 57 characters long. Each of the 121 domain names belonging to this bot network return 10 different IP addresses on a DNS query for a total of 1.2K IP-addresses. Also, in some replies, there are invalid IP addresses like 0.116.157.148. All the 10 IP addresses returned for a given domain name, belong to different network prefixes. Furthermore, there is no intersection in the network prefixes between the different domain names of the mjuyh bot. We strongly suspect that this is a case of "domain fluxing" along with "IP fast fluxing", where each bot generated a different randomized query which was resolved to a different set of IP-addresses.

Helldark Trojan: We discovered a component containing five different third-level domains (a few sample domain names are as shown in Table IV). The component comprises of 28 different domain names which were all found to be spreading multiple trojans. One such trojan spread by these domains is Win32/Hamweq.CW that spreads via removable drives, such as USB memory sticks. They also have an IRC-based backdoor, which may be used by a remote attacker directing the affected machine to participate in Distributed Denial of Service attacks, or to download and execute arbitrary files [60].

Mis-spelt component: There are about five components (comprising 220 domain names) which used tricked (mis-spelt or slightly different spelling) names of reputed domain names. For example, these components use domain names such as uahoo.co.uk to trick users trying to visit yahoo.co.uk (since the alphabet ‘u’ is next to the alphabet ‘y’, they expect users to enter this domain name by mistake). Dizneyland.com is used to misdirect users trying to visit Disneyland.com (which replaces the alphabet ‘s’ with alphabet ‘z’). We still consider these components as malicious since they comprise of domains that exhibit unusual alphanumeric features.

Domain Parking: We found 15 components (630 domain names) that were being used for domain parking, *i.e.*, a practice where users register for a domain name without actually using it, in which case the registrar’s IP-address is returned as the DNS response. In these 15 components, one belongs to GoDaddy (66 domain names), 13 of them belong to Sedo domain parking (510 domain names) and one component belongs to OpenDNS (57 domain names). Clearly these components represent something abnormal as there are many domains with widely disparate algorithmic features clustered together on account of the same IP-address they are mapped to.

Adult Content: We find four components that comprise of 349 domains primarily used for hosting adult content sites. Clearly this matches the well known fact, that in the world of adult site hosting, the same set of IP-addresses are used to host a vast number of domains, each of which in turn may use very different words in an attempt to drive traffic.

In addition, for comparison purposes, we used the lexical features of the domain names such as the length of the domain names, number of dots and the length of the second-level domain name (for example, *xyz.com*) for training on the same ISP trace, instead of using the K-L divergence, JI measure and Edit distance measures used in our study. These lexical features were found to be useful in an earlier study in identifying malicious URLs [38]. The model trained on these lexical features correctly labeled four components as malicious

(Conficker bot network, three adult content components and one component containing mis-spelt domain names) during the testing phase, but it also resulted in 30 components which were legitimate as being labeled incorrectly; compare this against 27 components that were correctly classified as malicious and two that were false positives on using our alphanumeric features.

We also test our model on a trace obtained from a South America based Tier-1 ISP. This trace is about 20 hours long and is collected on a smaller scale as compared to the ISP trace from Asia. The time lag between the capture of S. American Tier-1 ISP trace and the previously used ISP trace from Asia, is about 15 days. We use the same training set for the prediction model as we use for the ISP trace from Asia. In the prediction stage, we successfully detect the Conficker component with no false positives. The Conficker component has 185 domain names and 10 IP addresses. Of the 10 IP addresses determined for the Conficker component of the South American trace, nine are common with the Asia ISP trace's Conficker component. We conclude that Conficker based C&C servers have relatively large TTLs. However, out of the 185 domain names only five domains are common from this component and the component from the ISP trace from Asia. Clearly, the Conficker botnet exhibits rapid domain fluxing. Overall, this experiment shows that a training model learnt in one network can be applied to a completely different network and still successfully detect malicious domain groups.

G. Discussion

1. Usefulness of Component Analysis

Conficker botnet, present in our ISP trace, employs domain fluxing across TLDs, that became directly visible after IP-domain components were extracted and analyzed from the trace. The component analysis allowed the application of our detection methods across several different domains, which otherwise would have been separated from each other. In addition, component analysis allowed us to detect Conficker domains that would not have been detectable with our approach when applied to domain names alone since some of these domains contained fewer than 50 names needed for accurate analysis. Similarly, some of the IP addresses in the component hosted fewer than 50 names and would not have been detected with the IP address based analysis either. However, these domains will be included in the component analysis as long as the component has altogether more than 50 names.

Let D_c be the number of hostnames and I_c be the number of IP addresses in the component. If D_d, I_d are the number of hostnames and corresponding IP addresses detected through domain level analysis, we define domain level completeness ratios as D_d/D_c and I_d/I_c. Similarly, we can define the completeness ratios for IP-based analysis as D_i/D_c and I_i/I_c, where D_i and I_i correspond to the total number of hostnames and IP addresses of the Conficker botnet detected by the IP-based analysis.

For the Conficker botnet, these completeness ratios for IP-based analysis were 73.68% for IP addresses and 98.56% for hostnames. This implies that we are able to detect an additional 26.32% of IP addresses and a relatively small fraction of 1.44% of hostnames for those IP addresses. The completeness ratios for domain based analysis were found to be 100% for IP addresses and 83.87% for the hostnames. Therefore, we do 16.13% better in terms of determining the hostnames using the per-domain analysis. This shows that the component level analysis provides additional value in analyzing the trace for malicious domains.

2. Complexity of Various Measures

Table V identifies the computational complexity for every metric, and for all groups that we use. We observe that K-L metrics analyzing unigram and bigram distributions can be

Notation	
A	Alphabet size
W	Maximum word size
K	Number of test words
K'	Number of test words in a component
$S_{\underline{g}}$	Number of words in non-malicious database

Table V. Computational complexity of detection metrics

Grp.	**K-L unigram**	**K-L bigram**	**JI**	**ED**
dom.	$O(KW +A)$	$O(KW+ A^2)$	$O(KW^2S_{\underline{g}})$	$O(K^2W^2)$
IP	$O(KW +A)$	$O(KW+ A^2)$	$O(KW^2S_{\underline{g}})$	$O(K^2W^2)$
Com.	$O(K'W +A)$	$O(K'W +A^2)$	$O(K'W^2S_{\underline{g}})$	$O(K'^2W^2)$

computed fairly efficiently. However, for the JI measure, the size of the non-malicious database largely influences the time taken to compute the measure. A good database size results in a higher accuracy, at the cost of increased time taken for analysis. Similarly, edit distance takes longer for large word lengths, and the number of test words. However, it is independent of any database, hence the space requirements are smaller.

We briefly describe how we determine the bounds as expressed in Table V for the per-domain group. For the K-L unigram analysis since we examine every character of every test word, the complexity is bounded by KW. We then compute, for every character in the alphabet *A*, the divergence values. Therefore, we obtain the complexity as *O(K*W *+ A)*. Bigram distribution based K-L divergence is calculated similarly except that the new alphabet size is A^2. While calculating the Jaccard index, note that the number of bigrams obtained is *O(*$W-1$*)*. For each bigram, we examine the queues pointing to words from the non-malicious database. Thus, for each bigram, we examine *O(*$WS_{\underline{g}}$*)* bigrams. Since we do it for *K* test words, we obtain *O(*$KW^2S_{\underline{g}}$*)*. For every test word used while obtaining the edit distance, we examine it against the *K - 1* test words. Therefore, the total complexity is simply *O(K^2W^2)*. The expressions for *per-IP* and *per-component* groups are obtained analogously.

It is interesting to note that *A* is of the size 36 (0-9, a-z characters). *K* used in our analysis varies as 50/100/200/500. However, the average value for *K'* is higher in comparison. The DNS PTR dataset considered for *per-domain* analysis has approximately 469,000 words used for training purposes. This helps us estimate $S_{\underline{g}}$. For the ISP dataset, $S_{\underline{g}}$ is of the order of 11522 words. An estimate of *W* for the DNS PTR dataset is obtained from figure 9.

H. Proof of K-L divergence as being the Optimal Classifier

Let $\mathcal{A} = \{a_1, a_2, \ldots, a_M\}$ denote M the letters of the alphabet from which the domain names are chosen (in our case, this is English alphabet with spaces and special characters). Let $\underline{g} = [g_1, g_2, \ldots, g_M]$ and $\underline{b} = [b_1, b_2, \ldots, b_M]$ be the distribution of the letters in the good and bad domains, respectively. Let $\underline{x}$ be the actual domain name of length N, that has to be classified as being good or bad. Let the letter a_i appear n_i times in $\underline{x}$ such that $\sum_i n_i = N$. Let $\underline{q} = [q_1, q_2, \ldots, q_M]$ be the distribution of the different letters in $\underline{x}$, i.e., $q_i = n_i/N$.

Under the assumption that *a priori*, $\underline{x}$ can belong to a good or bad domain with equal probability, the classifier that minimizes the probability of error (wrong classification) is given by the maximum-likelihood classifier which classifies $\underline{x}$ according to

$$P(\underline{x}|\underline{g}) \underset{b}{\overset{g}{\gtrless}} P(\underline{x}|\underline{b}) \tag{3.3}$$

Intuitively, $\underline{x}$ is classified as good, if it is more likely to have resulted from the good distribution than from the bad distribution. The above classifier can be specified in terms of the likelihood ratio given by

$$\lambda(\underline{x}) = \frac{P(\underline{x}|\underline{g})}{P(\underline{x}|\underline{b})} \underset{b}{\overset{g}{\gtrless}} 1 \tag{3.4}$$

As we will see later, it is easier to work with an equivalent quantity $\frac{1}{N}\log\lambda(x)$. The classifier is then given according to

$$\frac{1}{N}\log\lambda(x) = \frac{1}{N}\log\frac{P(\underline{x}|\underline{g})}{P(\underline{x}|\underline{b})} \underset{b}{\overset{g}{\gtrless}} 0 \tag{3.5}$$

Under the assumption that the letters in $\underline{x}$ have been generated independently from the same distribution, $P(\underline{x}|\underline{g})$ is given by

$$P(\underline{x}|\underline{g}) = \prod_{k=1}^{N} P(x_k|\underline{g}) = \prod_{i=1}^{M} P(a_i|\underline{g})^{n_i} = \prod_{i=1}^{M} g_i^{n_i} = \prod_{i=1}^{M} g_i^{q_i N}. \tag{3.6}$$

The second equality follows by grouping all the occurrences of the letters a_i together and recall that there are n_i such occurrences. Similarly,

$$P(\underline{x}|\underline{b}) = \prod_{k=1}^{N} P(x_k|\underline{b}) = \prod_{i=1}^{M} P(a_i|\underline{b})^{n_i} = \prod_{i=1}^{M} b_i^{n_i} = \prod_{i=1}^{M} b_i^{q_i N}. \tag{3.7}$$

Using (3.6) and (3.7) in (3.5), the log-likelihood ratio can be seen to be

$$\frac{1}{N}\log\lambda(x) = \frac{1}{N}\log\frac{P(\underline{x}|\underline{g})}{P(\underline{x}|\underline{b})} = \log\frac{\prod_{i=1}^{M} g_i^{q_i}}{\prod_{i=1}^{M} b_i^{q_i}} \tag{3.8}$$

Dividing the numerator and the denominator by $\prod_i {q_i}^{q_i}$, we get

$$\begin{aligned} \frac{1}{N}\log\lambda(x) &= \log\frac{\prod_{i=1}^{M}\left(\frac{g_i}{q_i}\right)^{q_i}}{\prod_{i=1}^{M}\left(\frac{b_i}{q_i}\right)^{q_i}} && (3.9)\\ &= \sum_i q_i \log\frac{g_i}{q_i} - \sum_i q_i \log\frac{b_i}{q_i} && (3.10)\\ &= D(\underline{q}|\underline{b}) - D(\underline{q}|\underline{g}) && (3.11) \end{aligned}$$

where $D(\underline{q}|\underline{b})$ is the Kullback-Leibler (KL) distance between the two distributions.

Thus, the optimal classifier given in (3.5) is equivalent to

$$D(\underline{q}|\underline{b}) - D(\underline{q}|\underline{g}) \underset{b}{\overset{g}{\gtrless}} 0 \tag{3.12}$$

This result is intuitively pleasing since the classifier essentially computes the KL "distance" between $\underline{q}$ and the two distributions and chooses the one that is 'closer'.

CHAPTER IV

WINNING WITH DNS FAILURES: STRATEGIES FOR FASTER BOTNET DETECTION

A. Introduction

To automate the domain name generation for fluxing, botnet owners rely on generating domain names algorithmically which exhibit high *information entropy*. Reverse engineering of bot executables may yield the domain name generation algorithm and subsequently the domain names that a bot may query in the future. These domain names may be blacklisted or pre-registered in advance by security researchers. Domain fluxing botnets overcome this vulnerability by choosing to generate a large number of names, where only a few of them may host the C&C server. The large number of domain names is expected to overwhelm the pre-registration by others and potentially provide a cover for the actual name of the C&C server used by the botnet.

Botnets that employ domain fluxing can be characterized by the following two important features: (a) The alphanumeric distribution or entropy of the domain names for C&C servers is considerably different from human generated names. (b) The bots generate many failed DNS queries as many of the algorithmically generated domain names may not be registered or not available as C&C servers. Figure 14 highlights the behavior of a bot belong to a domain fluxing botnet. We observe multiple DNS queries issued by the bot, to obtain the location (IP address) of the C&C server. The bot may continue to issue the queries until one of them would resolve successfully, after which the bot may choose to sleep or proceed with an attack. We exploit the above described two important properties to detect botnets with very low latency, where we define latency as the number of domain names required for successful anomaly detection (or the time taken to collect those domains).

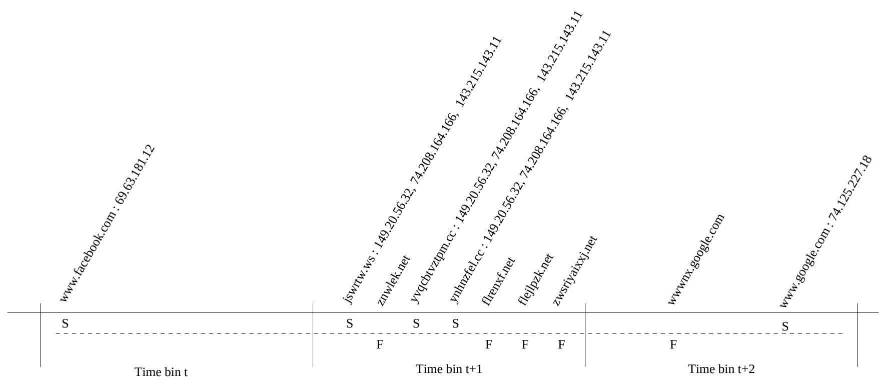

www.facebook.com : 69.63.181.12
jswrtw.ws : 149.20.56.32, 74.208.164.166, 143.215.143.11
znwlek.net
yvqcbtvztpm.cc : 149.20.56.32, 74.208.164.166, 143.215.143.11
ynhnzfel.cc : 149.20.56.32, 74.208.164.166, 143.215.143.11
flrenxf.net
flejlpzk.net
zwsriyaixxj.net
wwwnx.google.com
www.google.com : 74.125.227.18
S
S
F
S
S
F
F
F
F
S
Time bin t
Time bin t+1
Time bin t+2

Fig. 14. Sample set of queries seen from an infected host within the AS.

With our approach, we analyze successful DNS queries, and the failed DNS queries within the vicinity of the successful queries, thus exploiting their features to not only detect the C&C servers of those botnets faster, but also simultaneously detect bots within the network. While our detection mechanism is designed specifically to detect domain fluxing botnets by utilizing DNS failures, previous approaches relying only on domain entropy analysis can still be used in the event that there are no DNS failures. By analyzing the failed queries along with the successful queries, we increase the data available for analysis and hence speed up the detection process. While our technique can be used online (or in real-time), we focus on a trace-driven evaluation methodology here to keep the explanation simpler. Additionally, our analysis is based only on DNS network traffic, thereby reducing resource requirements, in comparison to techniques relying on general network traffic analysis.

When individual clients/hosts query a resolver, the failed queries can potentially be attributed to the presence of bots on that client and the successful queries close to the failures can be assumed to be related with high confidence. However, when queries are forwarded to a resolver from another local resolver or a DNS query aggregator, the queries from many clients can be grouped together and relating failed queries to other successful queries in the query stream becomes problematic. Our approach is cognizant of this difficulty and is capable of producing accurate results in the presence of aggregated query streams.

The main contributions of this work are:

- We utilize the failures around successful DNS queries and the entropy of the domains belonging to such queries, for *detecting* botnets with lower latency compared to previous techniques.

- We propose and evaluate a *speeding* technique which correlates DNS domain query failures for faster detection of domain fluxing botnets' C&C server IPs. We utilize temporal correlation between DNS queries and entropy-based correlation between domain names, for speedier detection.

- We show through a trace driven analysis that the proposed techniques can considerably speed up the detection of botnets that generate many DNS query failures. This in turn will constrain the domain name generation algorithms further if they want to evade detection by techniques such as proposed here.

We apply our techniques to two datasets. The first is a Tier-1 ISP dataset obtained from South Asia, captured for a period of approximately one day. Additionally, we analyze a university campus DNS trace captured over a month. The datasets consist of botnets validated through previous techniques applied to the trace [61]. Based on our analysis, we detect the presence of the recently discovered *Conficker* botnet. Our experiments indicate a false positive rate as low as 0.02% with a high detection rate. Our evaluation also yields how different features characterizing botnets can be varied, to assist a network administrator in tuning these parameters for their network(s).

B. Related Work

Alphabet entropy measures to detect algorithmically generated botnets by using successful domain name queries mapping to IP addresses, are proposed in [61]. Jiang et al. [62] use DNS failures to determine suspicious activity within the local autonomous system. Their technique analyzes bipartite graphs between failed DNS domain names and querying clients, to determine connected components with anomalous activity. Our approach, additionally, analyzes related successful queries and detects the botnet C&C servers along with the bots. The authors in [63] analyze unproductive network traffic of multiple protocols to classify malicious hosts based on features such as rate of failed traffic generation, entropy of ports used etc. In our work, we do not require training of data, and only rely on DNS based features for botnet detection.

Botnet identification using DNS has been explored in [64] where the authors utilize query rates based features of successful and failed DNS queries, to identify botnet anomalies. [65] detects new bots based on the similarity in querying behavior for known malicious hosts. Our technique does not rely on query rates and can detect botnets even if each bot queries for an independent botnet C&C server's domain names. Previous work on botnet detection has also examined the correlation of network activity between time and space as exhibited by users within a network [66]. We, however, use only the DNS traffic for detecting botnet activity, drastically reducing the resource requirements. Also, DNS security has been investigated in a number of recent studies [67] which have focused on DNS indirection and cache poisoning prevention.

C. Methodology

In this section, we describe our technique for detecting botnets through DNS traffic analysis using successful and failed DNS queries. We also highlight correlating failed domains, for

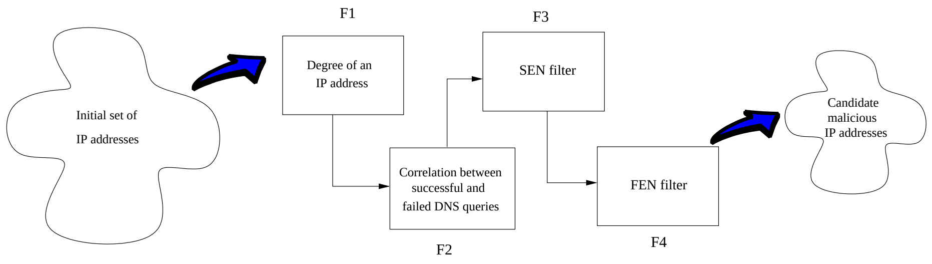

F1
Degree of an IP address
F3
SEN filter
Initial set of IP addresses
Candidate malicious IP addresses
Correlation between successful and failed DNS queries
FEN filter
F2
F4

Fig. 15. Filtering steps

faster detection of malicious IP addresses. In section D, we highlight the improvement in latency obtained by using failed DNS queries. Our primary goal is to detect the IP address of a domain fluxing botnet's C&C server. As a consequence of our analysis, we obtain the bots within the local network, and the domain names belonging to the botnet, thus exposing the botnet altogether. To discover anomalous IP address(es), we exploit multiple features such as the botnet structure, related domain name feature, and the temporal correlation between DNS query patterns of participating bots.

Our technique applies multiple filters to the DNS dataset used for evaluation (see Figure 15). At each stage (identified by a filter), we discard the set of IP addresses which do not exhibit malicious behavior as expected by the filter. Our filters are motivated by the structure and functioning of domain fluxing botnets. Since our primary goal is to identify botnet's C&C server address, we analyze information *with respect to an IP address* returned as the response in a DNS packet. Every candidate IP address (denoted by $cncip$) may be classified as malicious or benign when subject to these filters. Our approach can be suitably modified to consider candidate malicious domains on grouping them by sub-domains or the connected-components of bipartite graphs (between the set of domains and IP addresses they resolve to), thus exposing IP-fluxing botnets as well.

As an early indicator for domain fluxing, we use the number of domain names mapping to an IP address (D_{cncip}) as an initial filter. We call this as the $degree$ of the $cncip$. Since a C&C server IP address of interest to us, will have multiple domain names mapped to it owing to domain fluxing, an IP address with a high D_{cncip} insinuates greater maliciousness. Content Distribution Networks (CDNs) behave similarly and thus our additional filters help exclude such non-rogue entities.

Based on the behavior observed within the DNS datasets (as also illustrated in Figure 15), we note that a number of DNS queries are likely to fail around a successful resolution of a C&C server address. Hence, we look to see if a potential C&C IP address returned as a DNS response, has a high temporal correlation with failed queries. For every $cncip$, we analyze the time series or the sequence of DNS packets for all querying DNS clients (where a $cncip$ occurs) and thus compute the correlation measure ($Corr_{cncip}$). A higher correlation (larger $Corr_{cncip}$) is used as an indicator of botnet-like DNS querying pattern.

When domain-fluxing botnets employ random name generation algorithms, the domain names exhibit high information entropy. We measure the entropy present in both the successful DNS queries for the IP address being analyzed (the entropy being denoted by SEN_{cncip}) and that of the temporally correlated failed queries (denoted by FEN_{cncip}). We use normalized edit distance to measure entropy. The computation of this metric *does not require a reference or training* with external sources, and thus can be applied independently. The normalized edit distance metric measures the transformation required to convert one string to another, normalized over the length of the longer string. Thus, for a given set of strings, normalized edit distance computed over all pairs belonging to the set, measures the *intra-domain* entropy for the set. For SEN_{cncip} calculation, we apply this metric for all pairs of successful domain names (which map to $cncip$). Thus, the analyzed IP address is considered to be suspicious if the observed SEN_{cncip} value is close to 1.0. Similarly, we calculate FEN_{cncip} by computing the normalized edit distance between pairs of failed domain name strings, present within the vicinity of the successful DNS queries (related to

$cncip$). Using the same Domain Generation Algorithm (DGA), the entropy observed for a botnet's failed queries is expected to be *similar* to that of succeeding queries. As outlined in section 2, this hypothesis may be used for correlating relevant domain names for faster detection.

The temporal correlation between the set of DNS failures (FQ_{cncip}) and successes (SQ_{cncip}), and the subsequent requirement of proximity between entropy computed for FQ_{cncip} and SQ_{cncip}, extracts relevant information from the network traffic, in spite of the presence of DNS aggregators within the autonomous system (AS). It is noteworthy that for query streams seen from aggregators, legitimate IP addresses may be accompanied by failures more often, and thus $cncip$s which have high entropy for SQ_{cncip} and FQ_{cncip}, can be labelled malicious by satisfying the criterion set by our detection filters. However, as we shall note in section D, we obtain reasonable false positive rates.

Prior to subjecting the set of IP addresses extracted from DNS responses, to the filters used for detection, we also white-list packets for a set of well-known domain names and IP addresses. The 31 domain names present in our white-list comprise of known benign second-level domains (such as *facebook.com, apple.com*) and several DNS blacklist (DNSBL) service domains such as *spamhaus.org, uribl.com*. Inclusion of DNSBLs discards failures that may otherwise produce noise during analysis. We also discard RFC 1918 (private) addresses from analysis [68]. The details of the filtering steps are described below in more detail.

1. Filtering Steps

Figure 15 demonstrates the steps involved in narrowing down the set of IP addresses that are returned in DNS response packets (post white-listing), to a relatively smaller list of anomalous IP addresses. With each filter, we select a fraction of the input supplied by the previous filter, reducing the subsequent burden. In the following subsections, we describe each filter applied for a candidate C&C IP address (denoted by $cncip$) resulting in $cncip$ being discarded as legitimate or subject to additional filters. The measures employed by each filter, are either computed using select or all the time windows in which the candidate IP address occurs. For instance, we may discard those bins from analysis where we do not observe enough failed DNS queries for analysis. The typical time bin/window length used in our trace may vary depending upon the dataset in consideration. For evaluation presented in section D, we typically use a 128 sec window (64 sec symmetric about $cncip$). The following subsections detail how each measure is computed.

a. Degree of an IP address (D_{cncip}):

Domain fluxing is characterized by multiple domain names mapping to an IP address. We define the degree of an IP address as the number of domain names that map to a $cncip$. As a first filter, we use the degree (F_1 in Figure 15) to separate a set of IP addresses more likely to exhibit botnet like domain fluxing. For a given IP address, this number may vary based on the length of the trace analyzed. For instance, an IP address analyzed for an hour may have five domain names mapping to it. However, if analyzed for two hours, eight domain names may map to it, which includes previously expired domain names. While we consider the IPs which have a degree of at least two, we vary D_{cncip} to evaluate how quickly we detect anomalies. For a typical analysis, we use a degree threshold of eight, independent of the time for which a candidate IP address is analyzed. Thus, the filter F_1 can be bypassed if an IP has less than eight domain names mapping to it. However, this puts a constraint on the fluxing that a botnet server can exhibit. It should be noted that, Content Distribution Networks (CDNs) also have a high degree. However, CDNs get separated through additional filters as described ahead.

b. Correlation Metric ($Corr_{cncip}$):

As introduced earlier, bots generate burst of DNS queries, a fraction of which may fail. Thus, we exploit the temporal correlation between DNS successes and failures to identify malicious behavior. On observing a time window of DNS queries for a bot, we may observe the presence of failures, more frequently, than for legitimate clients. It is represented by filter F_2 in Figure 15.

The correlation metric ($Corr_{cncip}$) for a candidate IP address is computed as the probability of observing at least one failed DNS query in a time bin, given that $cncip$ was returned as an answer to a successful DNS query in the same bin. For detection, we heuristically choose the threshold as 0.5 implying that majority of windows in which a $cncip$ appears, should also have failures for it to be considered a meaningful anomaly. In section D, we study how the false positives decrease on increasing this threshold or when the correlation metric changes upon restricting our analysis to windows with more failures.

We use the following equation to compute this metric:

$$Corr_{cncip} = \frac{\sum \text{Time bins with } (S_{cncip} \cap F_{client})}{\text{Time bins with } S_{cncip}} \tag{4.1}$$

where S_{cncip} denotes the boolean condition of whether $cncip$ occurs in a time bin. F_{client} refers to the boolean variable indicating the presence of at least one failure in the corresponding time bin for the *client*. The correlation metric is computed with the time series of all clients which receive $cncip$ as the DNS response address. This metric may not be sufficient in topologies comprising of DNS aggregators where the temporal co-occurrence of DNS failures and successes is more frequent. Further developed measures limit the errors produced due to DNS aggregators.

c. Succeeding Domain Set Entropy (SEN_{cncip}):

We use *edit distance* as a metric for determining the similarity between a pair of domain names. Algorithmically generated domains exhibit a high value for this metric, owing to limited similarity between a given pair of domains. However, domain names observed for a legitimate entity, frequently have repeated occurrence of certain characters, which lower the computed normalized edit distance (or the entropy associated with the entity), as substantiated by [61]. For instance, a pair of domain names such as *www.google.com, ns.google.com* have a lower normalized edit distance than a pair like *jswrts.ws, yvqcbtvztpm.cc*, as observed for Conficker.

Edit distance is defined as an integral value indicating the number of transformations required to convert a given string to the other. The type of eligible transformations include addition, deletion, and modification of a character. We use the normalized edit distance measure computed as the Levenshtein edit distance [53] between a pair of strings normalized by the length of the longer string. The entropy of domains mapping to an IP address (and hence successful DNS queries), SEN_{cncip}, is determined by computing the normalized edit distance between every pair of domains that map to $cncip$ (taken from set with cardinality $|D_{cncip}|$), and averaged over all such pairs. Therefore, the complexity of entropy calculation is $O(n^2)$ where n is the number of domain names successfully mapping to an IP address over the duration of analysis. This duration is defined either in terms of a predetermined time, or the first few successful domains encountered for a given $cncip$. Once SEN_{cncip} is computed, if it exceeds a threshold (reserved for highly domain fluxing entities), we consider it for further analysis. Our evaluation shows that while high SEN_{cncip} IPs may be detected easily, even entities with relatively low entropy are detected with small false positive rates, making it difficult for botnet owners to improve their domain generation algorithm (DGA).

d. Failing Domain Set Entropy (FEN_{cncip}):

For a botnet, the domain name generation algorithm for failed domain names is no different than the domain names successfully resolved. The features expressed through alphanumeric characters composing the failed and successful DNS queries generated by a botnet, are therefore very similar. Thus, the failed domain names can help reduce the latency of analysis and improve detection since many more names can be analyzed in a shorter period of time, when associated with the succeeding queries.

To compute the entropy of failed domain names (denoted as FEN_{cncip}), we analyze the failing queries that occur in the vicinity of a successful DNS query. Our hypothesis is as follows. *For a bot issuing a burst of DNS queries to determine the C&C server address, the entropy of failed DNS queries present in the burst, is of the same order as the entropy of the successful queries*. We again use the normalized edit distance for determining the entropy of failed domain names present in a time bin containing successful query resolution. It is symmetric about the time instant where $cncip$ was observed. It is noteworthy that all failed DNS queries present in the time bin, may not be related to the successful DNS query. Such queries deviate the output. The noise is especially amplified at DNS *aggregators* which query on behalf of several individual local clients. Thus, choosing an appropriate time window length is critical for accurate analysis. During evaluation, we show how changing window size affects the performance.

To compute the failed domain entropy (FEN) for a candidate C&C IP address, we use the following equation:

$$FEN_{cncip} = \frac{\sum(FEN_{client})}{\text{Number of clients}} \tag{4.2}$$

where FEN_{client} is the FEN value computed by examining *client's* time series of query generation, with respect to $cncip$. To elaborate, the failed query entropy for a client is computed between pairs of strings (failed domain names) present within every time window in which $cncip$ occurs. Subsequently, all such FEN values are averaged thus giving FEN_{cncip}. The computation of this entropy requires at least two failed domain names within the window of consideration. A higher number of failures increase the confidence in the computed FEN value for that window implying that botnets are detected more accurately. Alternately, individual failed queries can be directly compared with the candidate successful domain names to compute the edit distance relevant to each failed domain name. We have evaluated both approaches and obtained similar results.

To further filter the candidate set of anomalous IP addresses, we consider only those addresses with FEN_{cncip} greater than a threshold. We choose a conservative threshold to avoid ignoring genuine anomalies. To apply our hypothesis of the proximity of SEN_{cncip} and FEN_{cncip}, we use the following inequality to further eliminate false positives:

$$(SEN_{cncip} - \delta) \leq FEN_{cncip} \leq (SEN_{cncip} + \delta) \tag{4.3}$$

where δ represents a small bound or the *proximity* within which FEN_{cncip} and SEN_{cncip} are expected to lie. To choose an appropriate δ, we compute the standard deviation σ of entropy for domains belonging to the known botnet IPs present in our dataset. Thereby, we choose δ as 3σ.

From the above description, the temporal correlation and entropy related parameters are analyzed and IPs satisfying the outlined malicious criteria, help in identifying bots within the network as well. In an autonomous system, where the DNS queries are observed from local clients and DNS aggregators, our technique may be applied recursively at the aggregator, yielding the bots which use the aggregator as their DNS recursive resolver.

2. Correlating Failures for Improved Latency

Here, we present an alternate strategy for *speeding* up the detection technique which relies only on successful queries. The work in the previous chapter [61] emphasizes upon applying statistical techniques such as K-L divergence, Jaccard Index, and Edit distance, to the set of successful domains for a $cncip$ (SQ_{cncip}). Through evaluation, it is shown that a large set improves the accuracy of anomaly detection. However, accumulating a larger set requires a considerable amount of time. Therefore, we propose supplementing the set SQ_{cncip} with failed queries that occur within the vicinity of successful DNS queries. The resulting set is accumulated faster, decreasing the latency of analysis by an order of magnitude.

The detection technique proposed in this chapter provides the basis for temporally associating the failed queries with successfully resolving DNS queries. This implies that only those failed DNS queries may be considered which occur around the time at which the successful DNS query, containing the IP address under analysis, occurs. In addition to temporal characteristics, to improve the quality of failed domain name set, we propose considering only those failing domains within the time window, whose entropy characteristics are similar to the successful domain set. Such similarity parameters for entropy can help identify a DGA by empirically noting the entropy expressed by domains generated using the DGA. Here, we explore supplementing SQ_{cncip} with the temporal and entropy-based features. Additional features as described for detection can also be used.

To realize the faster accumulation of relevant domains, we compute the entropy (normalized edit distance) between a failed domain name and each of the successful DNS domain names discovered under analysis. A domain yielding an entropy value close to the average SEN (or the entropy of successful domains) is considered relevant. The measure of closeness is defined using eqn. 4.3 as described above. For this particular experiment,

Table VI. Trace description.

	ISP trace	**Campus trace**
Trace collection period	Nov 03-04, 2009	Aug 22 - Sep 22, 2010
Total number of DNS sessions	1.61 M	112.7 M
White-listed sessions	770.23 K	54.4 M
Total number of failed DNS packets after white-listing	57.72 K	1.28 M
IP addresses analyzed for maliciousness	9948	74.7 K (per segment avg.)
Number of clients (or aggregators)	8472	1735 (per segment avg.)

we choose $\delta = \sigma$. Such computation with little additional overhead for identifying relevant domains may improve latency *and* accuracy. We evaluate this correlation strategy for Conficker's C&C server addresses in section D. Note that the manual analysis of discarded failed domain names identifies irrelevant queries belonging to services like *qq.com* and *ask.com*, which do not appear to exhibit the malicious domain fluxing. We also note that only a few domains belonging to Conficker are discarded, strengthening our confidence in the new set.

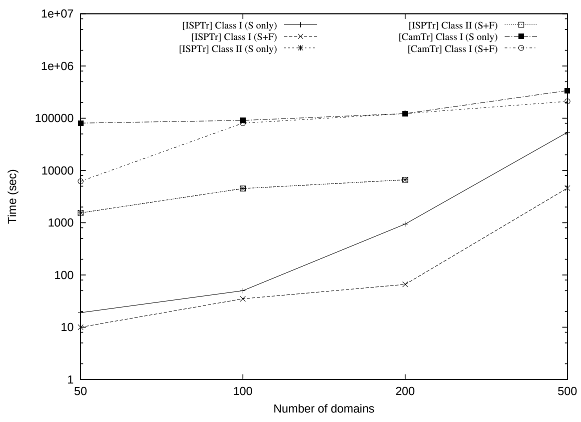
D. Results

We validate our technique using the DNS datasets described below. For our analysis, we consider only DNS type A records. Several DNS blacklist based services utilize the A record to verify whether an IP address, domain name, or an executable (a feature used by McAfee) is present in the blacklists. To exclude these queries from analysis, we white-list a total of 31 trusted second-level domain names including several blacklist services, Content Distribution Network services (such as *akamai.net, cloudfront.net*) and popular domains (such as *google.com, facebook.com*). The white-list helps us focus on other potentially malicious domains, in addition to refining the failed domains set used for analysis. Additionally, we avoid processing answers with RFC 1918 (private) addresses [68].

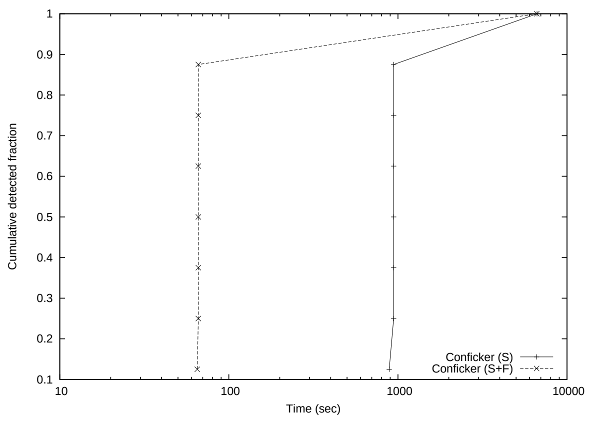
1. Data Sets

Table VI details the traces used for analysis. The 20-hour long ISP trace contains known malicious IPs belonging to the Conficker botnet. Using a blacklist, we obtain a set of 100 odd IPs labelled malicious, which we further verify manually by checking against exhaustive databases such as *robtex.com* and *mywot.com*. We believe that these two sources provide us with the most recent information concerning the queried domains or IP addresses. The 19 IPs obtained post verification with the above sources, contain two IP addresses hosting adult websites. We disregard these as benign due to their non-fluxing behavior. One C&C address apparently belongs to the domain-fluxing Kraken/Bobax botnet (based on the domain names we see). However, the Kraken C&C address has a degree of only two. The 16 remaining IPs belong to Conficker, some of which are sinkhole servers [69]. Nonetheless, we consider them as anomalous as they help keep the botnet alive. As a result, we consider the remaining 9931 IPs as legitimate. Note that all IPs considered for ground truth evaluation, have a degree of at least two.

We also use a DNS trace captured at a primary recursive resolver of a university network. We divide the month-long trace into approximately week-long segments and present our results on randomly chosen segments. Each segment contains an average of 295K IP addresses returned as DNS responses (after white-listing). As table VI shows, approximately 75K IP addresses have degree $D_{cncip} > 2$. For the campus trace, we use the C&C server information from the ISP trace to obtain 29 IP addresses (out of 75K), labelled as malicious. Since the ground truth information for the ISP trace is relatively old, we again verify this set manually. As a result, we are left with *four* Conficker C&C addresses which are common with those present in the ISP trace.



(a)



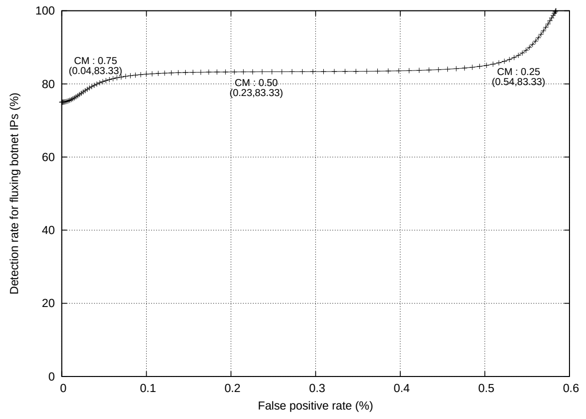
(b)

Fig. 16. Latency comparison (a) for different number of domain names. (b) for 200 domain names.

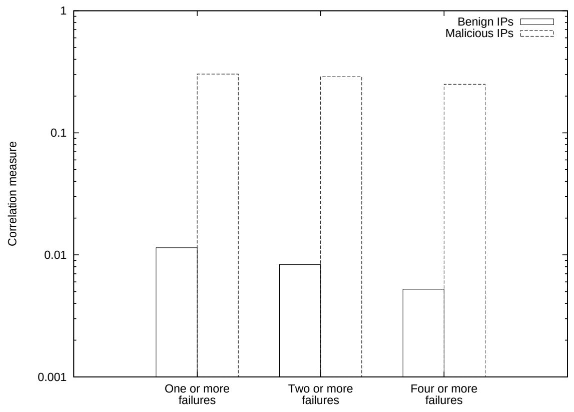
2. Latency Comparison

The latency of detection is expressed in terms of the number of successful domain names required to analyze and detect a rogue server accurately. Figure 16(a) shows the gain obtained in terms of time taken to collect a set of botnet domain names. The figure shows two classes of botnet IPs that we observe. Class I represents those C&C server addresses where domain fluxing yielded both successful and failed queries. However, for the C&C server in Class II, the bots issued none or very few failed DNS queries. In our ISP trace with more than 50 domain names mapping to it, we find eight C&C server addresses belonging to Class I and two belonging to Class II. Also, we observe that all four C&C addresses in the analyzed campus trace segment belong to Class I.

Figure 16(a) shows the improved latency for the average time taken by Class I or II addresses. From the figure, we observe that when failed domain names are correlated with successful DNS domains, the time taken to collect 50, 100, 200 or 500 domain names is considerably reduced. Especially, for 500 domain names, we see a gain of an order of magnitude when the time of collection reduces from approximately 54000 secs to only



(a)



(b)

Fig. 17. (a) ROC curve for changing correlation thresholds. (b) Correlation comparison for changing flux behavior.

4600 secs. With Class II, however, we do not observe any gain since no failures help in supplementing the set of successful botnet domains. Thus, we infer that in context of applying statistical techniques for anomaly detection, the proposed correlation mechanism can significantly reduce the time to collect input required for analysis of Class I C&C addresses. Note that we do not obtain 500 successful domains for the Class II addresses. The analysis with the campus trace follows analogous behavior. However, we plot the latency observed when correlating failures with successes, using the criterion highlighted in sub-section 2. We also note a higher initial latency for domain name collection, owing to slower traffic seen for a campus (Tier-4) as compared to a Tier-1 network trace. Although the traffic is slower, the time of collection is reduced considerably even with the campus trace. For instance, we observe 100 domains are collected 10K seconds faster than when using only successful DNS traffic.

While figure 16(a) shows average time taken for Class I and II anomalous entities, figure 16(b) shows the pace at which those anomalies are detected, for the specific case of 200 domain names. From the figure, we see that using the both successful and failed DNS domains, we can detect more than 80% of the total IP addresses, an order of magnitude faster than when using only the successful ones. We note that the cumulative detection reaches 100% because of the presence of Class II addresses.

From the figures, we conclude that considering failed domain names assists in speeding up detection of a domain-fluxing botnet. While speeding up the detection through methods presented in [61], the worst case detection latency is same as the original latency where only domains from SQ_{cncip} are used. We also transform botnet detection to a real-time detection approach through the speeding mechanism presented above, as well as through the detection strategy.

3. Effect of the Correlation Parameter

We evaluate the significance of using the correlation as a feature for anomaly detection. Figure 17(a) represents the ROC curve for changing thresholds for correlation between DNS successes and failures. The ROC curve shows a decrease in false positive rate with a decreasing detection rate, when increasing $Corr_{cncip}$ thresholds. A higher threshold requirement would imply that failures coincide with the successful queries more frequently. We would expect benign IP addresses to have a low correlation value. Hence, increasing the threshold results in decreasing false positives. For this particular experiment, we note detecting a maximum of 12 (out of 17 C&C IPs) as the remaining IPs do not have enough domain names for analysis (at least eight domains). We also note a maximum false positive rate of only 0.6% which primarily comprises of legitimate IPs with relatively lower entropy than seen for fluxing botnets. The false positives additionally contain failed DNS queries within the window of analysis (resulting in a correlation value just above the threshold). In contrast, fluxing botnets usually have a high number of failed domain names within the corresponding bin. We observed that the false positives include ISP's intra-AS DNS resolution queries where the hosts have been assigned random-appearing domain names.

4. Correlation vs Number of DNS Failures

Through this experiment, we aim to study the behavior seen for malicious IPs, in terms of the number of failed domain names generated. Figure 17(b) shows the correlation observed for malicious and benign IPs. We compute the correlation for three cases where we expect at least one, two or four failures to occur within the same window, as the $cncip$. The figure shows a decrease in correlation, for both benign and legitimate IPs, though the correlation reduces only slightly for the malicious set. Such a study implies that the correlation criteria may be adapted towards highly fluxing botnets while reducing false positives.

5. Variation in Entropy

We evaluate the impact of information entropy expressed by the domains which map to a candidate $cncip$ (that is, SEN_{cncip}). The effect of changing the entropy thresholds for considering a $cncip$ is shown through an ROC curve as in Figure 18. The figure shows an increase in the detection rate and the false positive rate as the threshold for entropy is decreased. This is in line with the observation that for a low threshold, several sets of domain names, and in particular those for CDNs satisfy the filters used for detection. Analogous to the performance with varying correlation threshold, the maximum false positive rate observed is 0.52%. Analysis of false positives reveals DNSBL services (*redcondor*), popular websites (e.g. *sina.com.cn* which offer multiple services), DNS and HTTP servers providing service to multiple entities, blogging services (*blogspot*) and CDN addresses. For instance, we observe *redcondor* IPs labelled malicious when the entropy thresholds are 0.35 or lower, indicating that even though correlation may be high for this DNSBL service, the entropy helps distinguish it from actual anomalies, when using higher thresholds. In section E, we discuss how botnets may attempt to fool our detection approach into generating domains with low entropy. Note that we determine the detection rate over the 12

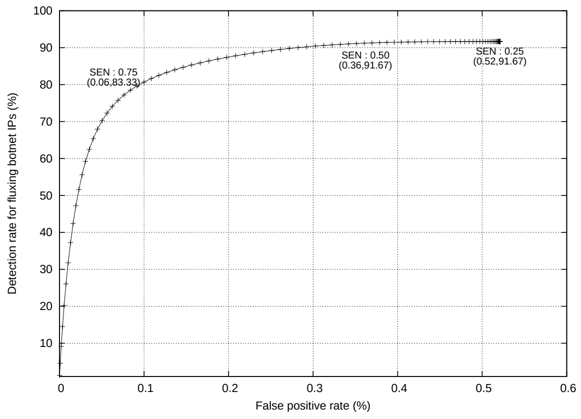


Fig. 18. Performance with changing entropy thresholds

Table VII. Impact of changing time bin size on botnet detection

Window size (sec)	**4**	**8**	**16**	**32**	**64**	**128**	**256**
	ISP trace						
FPR (%)	0.022	0.043	0.043	0.097	0.173	0.259	0.302
TPR (%)	75.0	75.0	75.0	75.0	75.0	83.33	83.33
	Campus trace						
FPR (%)	0.021	0.039	0.084	0.120	0.209	0.434	0.752
TPR (%)	100.0	100.0	100.0	100.0	100.0	100.0	100.0

detectable botnet IPs, as discussed in section 3.

6. Size of Time Bin

The impact of varying the size of time bins is shown in Table VII. In all experiments, we observe that false positives increase with larger time bins. The wider bins allow a higher possibility of inclusion of failures within the corresponding $cncip$ bin, resulting in an increased $Corr_{cncip}$ value. Thus, more candidate IP addresses meet the criteria set by filter F_2 (in figure 15) and the ones with high entropy may be incorrectly labelled malicious. From the table, we also observe that smaller window sizes result in fewer IP addresses being detected within the ISP trace. An inference of this observation is bots can distribute their DNS queries making temporal correlation between failures and successes difficult. The false positive rate, however, is less than 0.3% for ISP trace (and 0.75% for campus trace) for all experiments making the choice of larger windows possible. However, the marginal or no increase in detection rate and false positive rate, for the change in window size from 4 secs to 128 secs implies that a window as small as 4 secs may be sufficient for detecting a domain-fluxing botnet like Conficker. Note that the false positives for the campus trace are higher when compared to the ISP trace, as we frequently observe DNSBL (used by email filtering systems) based NXDOMAIN failures within the campus dataset. Such DNSBL failures affect the correlation/entropy parameters resulting in more benign IPs classified as malicious.

Dyndns is a service for generating customizable/random temporary domain names. In the ISP trace, we observe several random sub-domains for *dyndns* which exhibit high entropy characteristics, with several failures belonging to this service. Our detection mechanism designates these IP addresses as benign owing to presence of temporally distant failures, reinforcing our hypothesis of utilizing temporal correlation of failed domains for anomaly detection through the use of small time bins. We also limit our study to a maximum bin size of 256 seconds as fast-fluxing botnets have low DNS Time-To-Live (TTL) values.

E. Discussion

In this work, we detect Conficker C&C addresses which exhibit high entropy owing to randomized distribution of alphanumeric characters composing the domain names. However, to evade our detection mechanism, botnet owners may alter the way domain names are composed. For instance, the work in the earlier chapter has observed combination of dictionary words being used as an alternate way of domain-fluxing. Some example domain names that we observe for a botnet are *haireconomy.ru, greedycake.ru* and *empirekey.ru* [70]. The information entropy computed for a set of such domain names indicates that owing to high edit distance values, such domain names can still be distinguished through entropy analysis.

To validate the robustness of our approach, we artificially inject domain queries as observed above, with some of them failing and some domain names successfully mapping to a reserved address. We randomly choose clients to insert such DNS queries. Based on our study, we still detect the simulated anomalies with similar experiment parameters as used previously for evaluation. Also, in future if botnet owners formulate a DGA where the observed entropy is lower than that observed for fluxing botnet detected in this work, our detection mechanism can detect them with low false positives, as hinted by figure 18.

A direct weakness of our detection strategy is reliance on failed domain names. Our experiments are based on analyzing the first few successful domain names and correlating failures that are present in their vicinity. In the event that no failures are present, or failures that occur right after the window of analysis, our detection strategy may fail in which case switching to the algorithm for correlating failures to *supplement* the set SQ_{cncip} would help. Thus, a combination of both the strategies presented in this work may be useful for fast anomaly detection. It is possible to generate DNS queries slowly such that the failed queries are outside the time window considered in our scheme. Such an approach, however, will slow down the bot in identifying its C&C server and hence constraining the botnet writer again.

Our technique for detection can be mapped back to detect individual bots that issued the queries for a malicious $cncip$. For instance, with our campus trace analysis, we observe 12 hosts within our AS querying for three of the four C&C addresses, and 10 hosts (subset of the 12 above) querying for the remaining C&C address. While the Tier-1 ISP trace may not have individual clients (we would mostly observe aggregators), the mechanism applied for a smaller-sized network or when applied recursively, may result in more accurate detection of anomalies and bots, owing to a better DNS failure signal.

CHAPTER V

INFERENCE MALFEASANCE THROUGH BIG DATA ANALYSIS

A. Introduction

Malware infections spread via many vectors such as drive-by downloads, removable drives, and social engineering. Malicious domain or URL accesses, however, cause a majority of infections these days [71]. For example, a user may be tricked into clicking malicious links hidden in iFrames, sent via email, and embedded in advertisements [10, 72]. If infected, malware installed on hosts may be involved in pilfering sensitive data, spreading infections, DDoS attacks, and spamming. The stakes are high for enterprise networks due to legal issues and loss of intellectual property, money, and reputation. Hence to contain malware, enterprises must identify and isolate infected hosts in their network and prevent the other hosts from accessing malicious domains.

Reliably identifying infected hosts and malicious domains, however, is challenging. Modern malware such as rootkits may evade anti-virus products and other host-based analysis techniques. Similarly, identification via network and communication traffic analysis is difficult as botnet communication by design tries to mimic that of legitimate applications [3]. Many enterprises use DNS blacklists to identify and prevent malicious domain accesses; such lists, however, incur a significant delay in adding new domains as they rely on many manual and automated sources. Malicious domain inference using DNS and network properties such as a domain's IP addresses and BGP prefixes require large data collection. Hence these techniques may not scale well to large enterprise settings.

In this chapter, we propose a scalable and near-real time approach to detect infected hosts in an enterprise and malicious domains accessed by the enterprise's hosts. We first construct a *host-domain access graph* by adding edges between every host in the enterprise and the domains visited by the host. The graph is built using only the DNS based information, which comprises of only a small subset of the total network traffic. Such a choice reduces resource requirement. We *seed* the graph with ground truth information about a small fraction of nodes from external blacklists and white-lists, i.e., we label a few nodes as malicious and benign, and label the rest of the nodes as unknown. We then use *belief propagation* to estimate an unknown node's *marginal probability* of being malicious [73, 74]. A node's marginal probability is an estimate of the node being malicious given the state of the graph's labeled nodes; belief propagation is a fast and approximate estimation algorithm used in diverse fields such as computer vision and error correcting codes.

We applied our approach to host-domain graphs generated from two data sets: 760GB of HTTP proxy logs and DHCP logs collected in a global enterprise's worldwide locations and DNS query logs collected from a university. Our results show that with minimal ground truth information, e.g., 1.42% nodes in a graph, we achieve high true positive rates (TPR) with low false positive rates (FPR) and low false discovery rates (FDR). A benign node labeled as malicious by our approach is a false positive. Similarly, FDR is the fraction of benign domains among the domains labeled malicious by our approach. Low FPR and FDR are essential in enterprise settings. We also validated our results using well-known external sources to demonstrate our approach's robustness.

Our approach scales to large enterprise settings and *simultaneously* identifies malicious domains without any active interference inside the hosts. Our passive approach relies on data already collected by enterprises for regulatory compliance and forensic reasons. Also, unlike "global" DNS blacklists, our approach can be customized to an enterprise by utilizing the enterprise's data sets.

B. The Belief Propagation Algorithm

Marginal probability estimation in context of a graph $G = (V, E)$ with V nodes and E edges, implies estimating the state of a node based on the probabilistic model of the states assumed by other graph nodes. The marginal probability estimation problem is NP-Complete [74]. The *belief propagation* (BP) algorithm has been proposed as a fast and approximate solution for computing the marginal probability and has been successfully applied to various domains such as error-correction, image restoration, social-network based spam detection, and more.

Given a set of random variables with their respective probability distributions, the belief propagation technique is used for estimating the marginal probability for a particular variable [74]. In all of the previously described models, the goal is to *infer* or *estimate* a graph node's state. To elaborate, given a graph with random variables as nodes, each with an initial state distribution, and the relationship between nodes defined by a pre-determined criteria, the BP algorithm helps infer the probability (or the *belief*) of a node to be in a particular state. The belief propagation technique computes the marginal probabilities based on *messages* passed between nodes over multiple iterations. The iterations repeat until the system converges to a stable probability distribution of the states for each node.

The probability of a node i being in a state x_i, is referred to as the *belief*, denoted by $b_i(x_i)$. The computation of $b_i(x_i)$ depends on two factors: (a) the initial probability estimate for each node to be in a state, and (b) the mutual relationship between states of the two nodes. The initial probability of a node i to be in a state x_i, is called the $prior$, denoted by $\phi_i(x_i)$. In context of the belief computation for a bipartite graph with hosts and domains as nodes, the state assumed by every node in the graph can either be *malicious* or *benign*. The initial state distribution, therefore, estimates the node as being in either of the above two classes. The second key factor required to apply belief propagation is expressed by the relationship between two nodes i and j, also called the *edge potential*. The *edge potential* between nodes i and j, represents the probability of j being in a state x_j, given that the state of i is x_i, and vice versa. It is denoted by $\psi_{ij}(x_i, x_j)$.

In a graphical model with nodes connected to each other through edges, the computation of a belief is dependent upon the messages passed from one node to the other. A message from i to j which estimates node i's perception of node j being in a particular state (x_j), depends on the *prior* for node i, the *edge potential* for i and j, and the message inputs that i obtains from all its neighbors (excluding j). Mathematically, the message is defined as:

$$m_{ij}(x_j) = \sum_{x_i \in S_i} \phi_i(x_i)\psi_{ij}(x_i, x_j) \prod_{k \in N(i) \setminus j} m_{ki}(x_i) \tag{5.1}$$

where $N(i)$ represents the neighbors of node i, and S_i represents the set of all states that node i can be in (that is, $S_i \in \{benign, malicious\}$). Each message m_{ij} gets updated over every iteration and the algorithm stops when all messages converge, that is, the messages do not change significantly over iterations. The messages may be normalized such that $\sum_{x_i \in S_i} m_{ki}(x_i) = 1$. Normalization of messages prevents underflow. In later sections, we show that additional measures are required to prevent underflow in a well-connected graph with several nodes having degrees of the order of 10K. Finally, the messages from the converged (final) iteration are used for computing beliefs through the equation:

$$b_i(x_i) = C\phi(x_i) \prod_{k \in N(i)} m_{ki}(x_i) \tag{5.2}$$

where C denotes the normalization constant (that is, ensuring $\sum_{x_i \in S_i} b_i(x_i) = 1$). In our context of malfeasance detection, the belief value such as $b_i(x_i) = 0.6$ means that the node i is malicious with a 60% chance and benign with a 40% chance.

C. Methodology

In this section, we present our approach used to identify malicious (or benign) hosts and domains by applying the belief propagation technique on a bipartite graph. We also describe the bipartite graph generation process and justify the choice of parameters used.

1. Data Sets and Ground Truth

To generate the bipartite graph, we use a 26-hour long dataset of HTTP logs collected on September 29, 2010. The logs are collected from over 300 HTTP proxy servers catering to more than 300K clients within an enterprise network, with the proxy servers are spread all across the world. Each HTTP query in the log gives information about which client IP asked for a URL, the corresponding URL's domain, the time at which the query was made, the HTTP header etc. Every query in an HTTP log also gives us information about the status of the HTTP request made by a client. For instance, the requested URL could be served from the proxy's cache, fetched from the Internet, or the URL retrieval could also fail. In addition to the HTTP logs, we collect DHCP logs from six DHCP servers responsible for the 300K+ hosts. The DHCP logs were collected over five days before and including Sep 29, 2010. Each DHCP log identifies the duration for which an IP address is assigned to a physical host, which we correlate for an accurate host $\rightleftharpoons$ domain association. We also use a week long (August 22-29, 2010) DNS trace captured at a recursive DNS resolver of a university campus. The recursive resolver serves approximately 7.9K hosts within the university. Due to absence of the corresponding DHCP logs, the bipartite graph for the campus dataset thus contains client IP addresses as host nodes and the domains they query, as the domain nodes.

To establish the ground truth before applying the algorithm, we use a proprietary blacklist of domains, another proprietary blacklist of IPs, and the publicly available Alexa

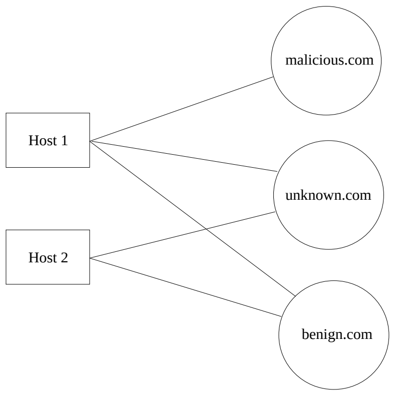


Fig. 19. Sample Host $\rightleftharpoons$ Domain bipartite graph containing nodes with a blacklisted (*malicious.com*), a white-listed (*benign.com*), and an unknown (*unknown.com*) domain.

white-list [75]. The domain blacklist contains approximately 658K fully qualified domain names (FQDNs, that is, not just sub-domains). The IP blacklist contains about 1.32M IP addresses. Both blacklists have been accumulated at around the same time as the logs. From the Alexa white-list, we use the top 5000 of the one million domains sorted by popularity. A conservative choice of only 5000 domains allows a greater confidence in assigning priors to domains in the dataset.

2. Overview

We apply the belief propagation algorithm to a bipartite graph consisting of a unique set of hosts and a unique set of domains accessed by those hosts. One such sample graph is shown in Figure 19 where a client within the network under analysis, identified as $Host_1$, accesses three domains viz. *malicious.com, unknown.com,* and *benign.com*. The order in which the domains are accessed is irrelevant. Similarly, client $Host_2$ accesses two domains, both of which are common with those accessed by $Host_1$. Thus, our problem may be posed as determining the reputation of the unknown domain *unknown.com* or that of the hosts, based on the known information about domains.

The actual graph generated from the HTTP-proxy and DHCP dataset is much larger, with approximately 2M nodes and 12M edges, and with the ground truth available only for approximately 20K nodes. We describe the enterprise network bipartite graph generation procedure with greater detail in section 3. Once the bipartite graph is generated, we assign *priors* and choose appropriate *edge potentials* before we apply belief propagation to compute the reputation of each node (hosts or domain) in the graph. The reputation is inferred through message passing as described in section B. In section 4, we describe the choice of *priors* and *edge potentials* specific to our problem. We also highlight the non-trivial assumptions for application of BP to a large graph as encountered for our analysis. The reputation of each node, which is the marginal probability estimated through BP, is subsequently used for labelling the node as either *malicious* or *benign*. System administrators may thus choose the domain designations to supplement intra-AS blacklists, white-lists, or access control lists (ACLs), thereby curbing the spread of infection within the AS, or utilize host designations for patching infected machines.

3. Composing the Bipartite Graph

The bipartite graph of hosts and domains for an autonomous system represents the domains accessed by hosts. This relationship is expressed through an edge from a host node to a domain node in the graph. Such a graph may have overlaps between different hosts or domains. That is, multiple hosts may access the same domain, in addition to a host accessing multiple domains. The reader may note that with regard to BP, the belief computation is dependent on the messages traversing graph edges, and therefore the beliefs are influenced by the graph structure.

Each HTTP log from our dataset identifies the URLs accessed by a client (represented through an IP address). For our problem, we analyze only the domain in the URL. Recognizing IP address churns due to dynamic addressing, we also use the DHCP logs for accurate association of a host with a domain. The DHCP logs help us identify all IP address(es) bound to each client over the duration of analysis, through unique records associating the 48-bit hardware (MAC) address of the network interface, with an IP address, at a particular time instant. Thus, all host nodes in a bipartite graph represent unique network interfaces.

To accomplish graph generation, we determine the temporal intersection of two types of bindings: (a) From HTTP-proxy logs: what domains does an IP address access?, and (b) From DHCP logs: what IP address(es) got associated with a MAC address? The resulting bipartite graph yields 141K hosts (or MAC addresses) and approximately 1.3M domains. The graph also contains approximately 12M undirected edges where each edge joins a host to a domain. The degree distribution gives us an indication of the feasibility for a message to traverse the length of the graph. In context of applying BP for anomaly detection, the *base* message which carries the known (or ground truth) information, should propagate across the whole graph. A well-connected bipartite graph thus ensures that such *base* messages travel farther helping in an accurate analysis. In section e, we discuss the possibility of *excessive* information propagation leading to noise.

Note that the nodes representing the unknown domains in the bipartite graph represent unique second-level domains. Considering only the second-level domains improves the degree distribution of domain nodes, increasing the probability of existence of a path between two nodes in the graph. Such a choice is also motivated by the assumption that usually second-level domains are responsible for (mis)happenings within their domain or sub-domains. Thus, our approach may designate a domain *example.com* as malicious, even though only a sub-domain *badsite.sub.example.com* is actually involved in malicious activities. It is also worth noting that a domain node may also represent a web server IPv4 address that we obtain from the HTTP-proxy logs. In this case, we represent the complete IPv4 address as a node in the bipartite graph. We ignore illegitimate domain names (with no TLDs) and retrieve the second-level domains from complete domain names using Google's Guava library for Java [76].

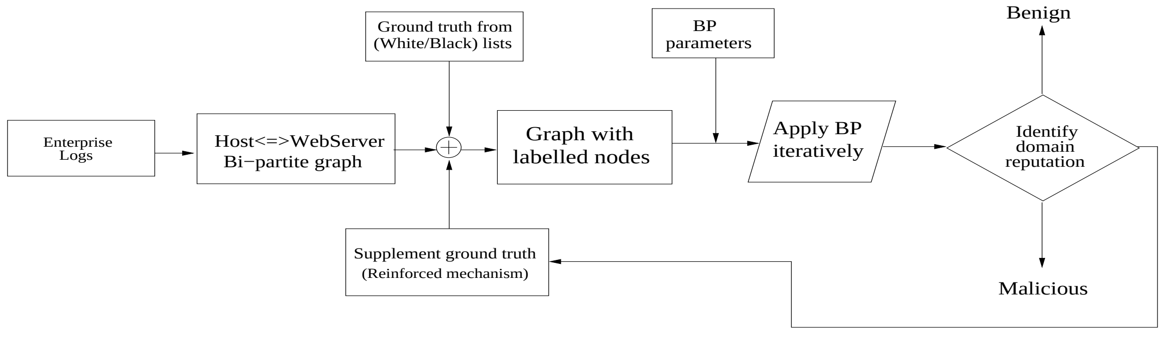

Enterprise Logs
Host<=>WebServer Bi−partite graph
Ground truth from (White/Black) lists
Supplement ground truth (Reinforced mechanism)
Graph with labelled nodes
BP parameters
Apply BP iteratively
Identify domain reputation
Benign
Malicious

Fig. 20. Flow chart for reputation inference.

4. Anomaly Detection with BP

Figure 20 illustrates the components of our algorithm. After generating the bipartite graph as described in section 3, each node is assigned a prior (probability of being malicious). We use a proprietary domain blacklist to assign a prior of 0.99 (most used value) to a node with its corresponding domain (or IP) in the blacklist. We use a publicly available white-list [75] to assign a prior of 0.01 to domain nodes representing domains in the white-list. Note that we use only the top 5000 domains from the white-list. From the lists, we find that approximately 16K domains are in the blacklists and about 4.5K domains are in the white-list. Thus, the a priori information for our algorithm is very small and comprises of only 1.42% of the total graph nodes. Note that our dataset does not contain any information about host nodes being infected. Therefore, we assign a prior of 0.5 to all host nodes implying that the probability of a host node being malicious is 0.5 (which is same as that of being benign).

After prior assignment to graph nodes, we apply the iterative belief propagation algorithm. In every iteration, the algorithm passes messages over all the directed edges, where a message is computed from equation (5.1). The message computation also depends on the choice of an *edge potential* matrix. One such sample matrix is present in Table VIII.

From the table we infer the following about the relationship between two graph nodes.

Table VIII. A sample edge potential matrix (ψ_1)

x_i \ x_j	**Benign**	**Malicious**
Benign	0.51	0.49
Malicious	0.49	0.51

Table IX. Another edge potential matrix (ψ_2)

x_i \ x_j	**Benign**	**Malicious**
Benign	0.75	0.25
Malicious	0.49	0.51

Given a host, if it is assumed to be benign, we consider the domain it accesses (represented as a graph edge) to be benign with a probability of 0.51. This follows the general intuition of associating goodness together, akin to a social network. Similarly, if a host is assumed to be malicious, then there is a greater (51%) chance that the domain it accesses is malicious. The relationship holds even in the alternate case where the inference over a host is based on the state of the domain. Thus, the edge potential matrix symmetrically estimates the state of a domain, given a host, and vice versa. For the curious reader, we note that using asymmetric edge potential results in final beliefs oscillating between two equilibrium values. The asymmetric edge potential here implies, for instance, that given a malicious host, the probability of domain being malicious is different from the probability of the host being malicious when it accesses a malicious domain. As a result, the beliefs obtained with an asymmetric ψ, are inconclusive as the messages obtained in odd iterations are distinctly different from the messages in even iterations, and therefore not considered further in this work.

The belief propagation technique has been applied to graphical models where nodes are usually associated with a small degree, such as for image restoration or error correction. However, for a model like ours where domain nodes may be contacted by thousands of hosts or a host node can access several domains, the computation of messages becomes non-trivial. Specifically, since an outgoing message from a node is dependent on messages sent from its neighbors, equation (5.1) for a large number of neighbors leads to underflow. Thus, corresponding beliefs computed also suffer from underflow during computation. To counter the problem of underflow, we use Java's *BigDecimal* data type for storing extremely small decimal values, in addition to normalizing messages. Note that the conventional data types such as *float* and *double* are not sufficient.

The algorithm iterates over all edges until the messages converge. A converged message $m_{ij}(x_j)$ outlines node i's estimate of node j being in a state x_j, and that estimate reaches an equilibrium which does not change considerably over time. Once the messages have converged, the beliefs are computed using equation (5.2). The beliefs, when normalized, indicate the relative reputation of nodes. Note that the belief values vary from 0.0 to 1.0. Thus, based on an appropriate choice of a threshold, we designate the node with belief value greater than or equal to the threshold as malicious, and benign otherwise.

D. Evaluation

We now present our evaluation over the HTTP-proxy logs and the campus DNS dataset described earlier. To apply the belief propagation algorithm on our huge dataset, we use a 12-core 2.8 GHz machine with 48 GB of RAM. In another setup, we utilize a 48-core 2.3 GHz and 128 GB RAM, assisted with multi-threaded message computation. A typical experiment with 24M directed edges as in our graph, consumes approximately 25 GB of memory. Our implementation is in Java.

1. Analyzing HTTP Dataset

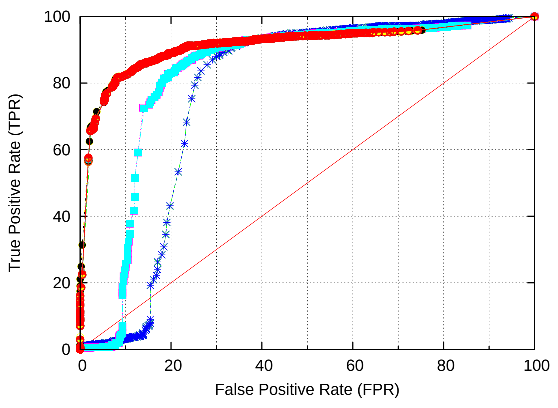
a. Performance

We evaluate the performance of our approach by examining the ROC curves for detection rate and false positive rates which are computed using the K-fold cross validation technique with K as 10. For K-fold cross validation, we divide the ground truth into K distinct parts. We then use K - 1 parts as seeds for applying belief propagation, and validate the labels assigned by our approach to the remaining part. Unless otherwise stated, we use a 10-fold cross validation for our analysis. Figure 21(a) shows the quality of identifying the malicious and benign domains correctly, for one (of 10) segment(s). The initial iteration is in blue while the final iterations are in red color. The curve shows detection improving with every iteration with the increasing area under ROC curve. Note that the bigger the area under an ROC curve, the better the performance.

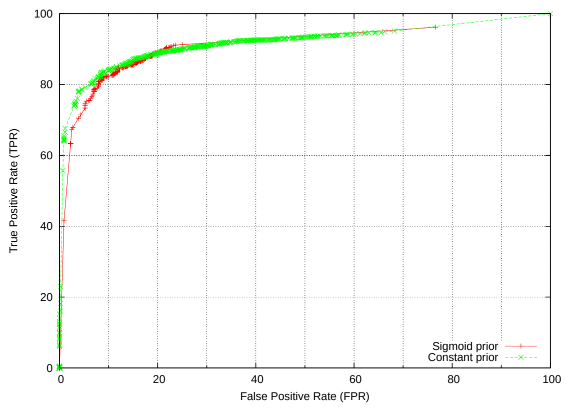
We observe that for the converging iteration, we obtain a detection rate (TPR) of approximately 80% with a 7% false positive rate (FPR). We note that for large networks, our tool may result in raising too many false alarms. This can be countered by delegating the analysis of the tool to subnets within the larger network, making analysis feasible. In later section, we show that the performance can be further improved using zone-based evaluation where we obtain the same detection rate for an FPR less than 1%. From the figure, we also observe the false discovery rate (FDR) to be between 2-5% for a false positive rate less than 10%. Note that the false discovery rate measures the expected proportion of false positives among all significant hypotheses. The relatively poor performance for initial iterations can be attributed to messages which do not traverse the path long enough, thus prohibiting propagation of correct state estimates embedded in *base* messages (as defined previously). This affects the computation of beliefs and hence the performance.

b. Sensitivity to Priors

For Unknown Entities: In previous experiments, we use 0.5 as the prior for nodes for which there is no initial evidence of being malicious or benign. The prior of 0.5, however, does not influence the outgoing messages based on incoming information (equation (5.1)). Therefore, for unknown domain or host nodes, we experiment with priors derived from the



(a)



(b)

Fig. 21. (a) ROC curve for a typical experiment with priors for unknown nodes as 0.5, and the edge potential as in Table VIII. Initial iterations are in blue while the converging iteration is in red. The straight line is the 0th iteration (no reputations assigned). (b) ROC comparing the performance of using constant priors vs. sigmoid priors. Performance with constant prior is similar to sigmoid, but takes twice as many iterations.

corresponding node's characteristics. One such function that we use for assigning priors is the sigmoid function, as given by the following equation:

$$f(x) = \frac{1}{1 + e^{\frac{-(x-k)}{W}}} \qquad \text{where, } 0 \leq f(x) \leq 1 \tag{5.3}$$

where $f(x) = 0.5$ when $x = k$. W, the sigmoid width, is a parameter which defines the slope or the rate of increase or decrease of $f(x)$. Note that the sigmoid function can be monotonically increasing or decreasing depending upon the coefficient for $(x - k)$ in equation (5.3). If the coefficient is negative (as in equation (5.3)), $f(x)$ is monotonically increasing and decreasing otherwise.

To assign priors for unknown host nodes, we choose an increasing sigmoid function, where the prior is a function of the number of HTTP requests made by a host. We assume that a large number of HTTP requests indicates higher maliciousness as malware may repeatedly communicate with the rogue master for various activities (section A). For this experiment, the parameter k for the host prior function is 6219, which is the average number of HTTP requests made by host nodes in the bipartite graph.

Similarly, we choose a decreasing sigmoid function for unknown domain nodes where the function varies with degree of the node. A node with large degree indicates high popularity (e.g. *facebook.com*), and thus the function assigns a low prior to the node. The average domain degree (k in equation (5.1)) comes out to be nine. Note that we compute W such that $f(x)$ is in the range [0.01, 0.99].

Figure 21(b) presents the performance with sigmoid priors. We observe that the original choice of constant priors performs marginally better for lower false positive rates. For instance, for a false positive rate of 6%, we note the true positive rate increasing from 73% to 80%. However, the number of iterations (or the time taken) to converge to final belief values is almost *half* for sigmoid based priors. The lower latency is attributed to speedier transition of messages across the edges due to a relatively biased ($\neq$ 0.5) prior.

c. Sensitivity to Edge Potentials

For graphical models with high degree, the message value is sensitive to the choice of edge potential matrix. For high degree nodes, the outgoing message converges to specific elements of the edge potential matrix. This behavior is recognizable from equation (5.1). Therefore, an appropriate choice of the edge potentials is critical, especially for large graphs.

The performance with the edge potential from Table VIII has been shown in Figure 21(a). We experiment with another matrix as shown in Table IX. In this table, we assume a prevalence of beneficence and thus assign a relatively lower probability to spread of malware. While the performance (in terms of detection rates and corresponding false positive rates) with the new edge potential matrix is almost the same as that for the edge potentials of Table VIII, we observe two significant differences. First, we observe that the convergence rate for the new edge potentials, is *three times* faster than that for the edge potential from Table VIII. The second observation is that the performance even for initial iterations, is very close to that for converging iterations (influencing faster convergence). We attribute these advantages to the edge potential values which reflect the probability estimates between nodes better than Table VIII. Consequently, the beliefs computed from the first few iterations can be utilized for assessing a host or a domain, giving us a 10-20 times latency reduction by an order of magnitude compared to our original setup.

d. Analysis of Labelled Domains

We now evaluate the labels assigned by our technique to the unknown domain nodes in the bipartite graph. To accomplish unknown domain node verification, we use three public sources for determining domain (or IP address) reputation viz. Web of Trust (WoT), McAfee TrustedSource, and *Robtex*. The WoT assigns numerical reputation and confidence scores (0-100) for four categories, namely, *Trustworthiness, Vendor reliability, Privacy,* and *Child Safety*. We use scores only from the *Trustworthiness* category. TrustedSource's evaluation of a queried domain or IP address, is relatively more generic as it only assigns categorical labels such as High/Medium/Unverified/Minimal risk. Robtex is a multipurpose web-based tool which checks the queried subject against hundreds of DNSBL services.

Automated validation: First, we use the WoT API to check 2000 domain nodes with the highest and the lowest belief values each. The domain nodes with top 2000 belief values (denoted by *maldom-list*) represent what our technique considers the most malicious, while the bottom 2000 are considered to be most benign (the *bendom-list*). The beliefs are computed with the *edge potential* as in Table VIII, the known node priors being 0.99 or 0.01, and the unknown node prior assigned using a sigmoid function. When checked against the WoT API we find that only 12.3% of the domains in the *maldom-list* are present in the WoT database. On the contrary, 65.1% domains from *bendom-list* have a score from WoT.

Table X. Distribution of domains from designated malicious and benign lists into various categories. We consider only the ones with high confidence (≥ 23).

Category (Score range)	Maldom-list fraction (%)	Bendom-list fraction (%)
Excellent (80-100)	0.4	25.0
Good (60-79)	0.05	3.0
Unsatisfactory (40-59)	0	1.7
Poor (20-39)	0.05	1.3
Very Poor (0-19)	0	1.95
Total	0.3	21.98

Note that the lag between the actual dataset and external validation as described here, is approximately 11 months. Thus, we infer that the *maldom-list* primarily consists of volatile domains, a prominent feature of malware or domain-fluxing botnets [61] [62] [77]. Of the domains in the two lists with scores present, we look for the ones with good *reputation* (≥ 60) and high *confidence* (≥ 23). We find that only 0.45% domains from *maldom-list* and 28.0% domains from the *bendom-list* satisfy the criteria above. Table X categorizes the domains in the two lists based on their reputation scores. Note that we present the distribution only for the scores which have a corresponding high confidence which is a conservative estimate. Further sections reveal our evaluation of both benign and malicious labelled domains using another source and different levels of malfeasance (or beneficence) tests.

Manual validation: From the domains which do not have scores in the WoT database, we choose 100 domains randomly from each list and validate them by checking against TrustedSource and *Robtex*. We consider domains as malicious only if they belong to High risk category (TrustedSource) or are present in at least two[1] of the blacklists (checked by *Robtex*). Similarly, to identify benignness, we look for domains categorized as "Minimal Risk" and not present in any of the blacklists checked by *Robtex*. Tables XI and XII summarize our analysis.

[1]We choose this threshold heuristically. Later, we highlight 8 as a reliable threshold.

Table XI. Manual validation of 100 random domains (with absent WoT scores) from the **maldom** list.

TrustedSource \ Robtex	≥ 0 BL	≥ 1 BL	≥ 2 BLs	≥ 4 BLs	≥ 8 BLs
High risk	51%	47%	47%	42%	17%
Unverified risk	25%	11%	10%	7%	0%
Minimal risk	22%	1%	1%	1%	0%

Table XII. Manual validation of 100 random domains (with absent WoT scores) from the **bendom** list.

TrustedSource \ Robtex	≥ 0 BL	≥ 1 BL	≥ 2 BLs	≥ 4 BLs	≥ 8 BLs
High risk	2%	1%	1%	0%	0%
Unverified risk	11%	2%	1%	1%	0%
Minimal risk	87%	6%	0%	0%	0%

From the table, we observe that 60%(15) of the malicious-labelled domains are found in at least two of the DNSBLs checked by *Robtex* while for *bendom-list*, this fraction is 20% (five) of the 25 random domains. We later show how we improve the labelling by qualitatively using labels as ground truth for further prediction.

e. Impact of Graph Modification

In previous sections, we note the impact of parameters directly affecting the belief propagation algorithm. Here we present another aspect which is to study the effect of modifying the graph structure, on the performance of our approach. The belief messages containing probabilistic reputation scores, traverse the edges in graph. The convergence parameters identify the spread of such messages, in essence limiting the number of hops the messages can make. Thus, a highly connected graph allows noise to travel farther than it should. To better explain the scenario, let us take an example. Assuming a host is infected with malware, visits a malicious domain in addition to several benign and unknown domains. In an attempt to identify another malicious domain, the propagation of malicious messages will get overwhelmed by the benign messages leading to a false negative and therefore hurting performance. To constrain the scope of such noise, we modify the graph reducing

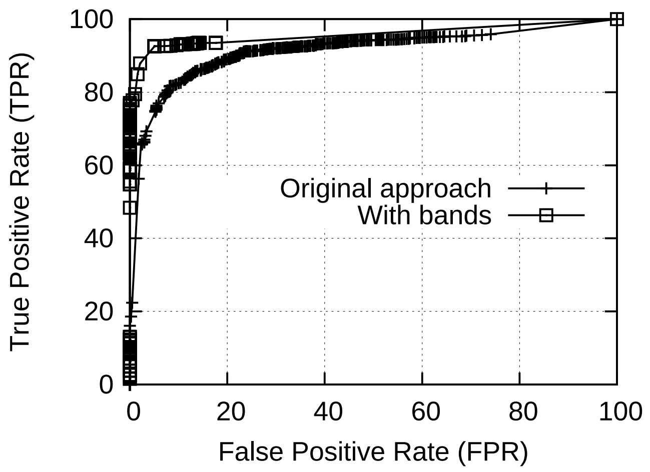


Fig. 22. Analyzing the separation of labelled domains.

the number of propagation paths.

Path removal in our context translates to removing edges. The maximum impact is thus realized by removing nodes from the graph with the highest degrees. Therefore, to understand the impact of curbed noise, we remove 500 unknown domain nodes with the highest degrees. As a result, there is a 52.13% reduction in the number of edges, from 11.85M to 5.67M undirected edges. As a result, using default values for remaining parameters, we observe a minor improvement in performance. More specifically, we observe that for false positive rates ranging from 10-20%, the detection rates improve to 2-3% confirming our hypothesis of presence of noise and the subsequent alleviation. For larger networks, the gains obtained due to such a modification can be significant, especially when our tool is used at the edge of the large network.

f. Qualitative Analysis of Assigned Labels

The reputation score or the final belief value represents the confidence of our algorithm in designating a node as benign or malicious. For instance, a benign test domain is expected to have a lower belief value (tending to zero) while a malicious test domain's reputation score would be close to one. We now study the quality of this score assignment.

To assess the designated labels, we split the final belief values into three regions or bands, symmetric about the mid-point of range of final beliefs. The uppermost band consists of domains with highest beliefs and therefore labelled malicious by our algorithm (*malRegion*). Similarly, the lower most region represents domains labelled benign (*benRegion*). We do not, however, assign labels to nodes in the central region. Thus, the central region represents a low confidence zone which may not represent the true state of test nodes and through demarcation of such a zone, we intend to make fewer classification mistakes. Figure 22 highlights such an evaluation. A false positive in this scenario is a benign node present in only the *malRegion* while malicious nodes falling in the *benRegion* are false negatives.

Upon linearly expanding the central region about the mid-point, we obtain a performance as shown in figure 22. We note a significant improvement when avoiding to classify domains with beliefs in the low-confidence zone. For instance, at a false positive rate of approximately 5%, we note the detection rate improved by 18% (from 70% to 88%), while at the approximately 10% false positive rate, this increase is 11%. Hence, given a wide central region, a system administrator could conservatively choose to populate the local domain blacklist and white-list with domains only from the narrow *malRegion* and *benRegion* respectively tolerating malware infections while avoiding to unnecessarily patch (thus, disrupt) clean machines (false positives). An aggressive approach, however, would use narrow central bands in an attempt to find future infections.

2. Analyzing Campus DNS Dataset

To understand the implications of our approach in a different environment, we also apply our technique to a university campus DNS dataset. Thus, we observe the DNS queries made by the hosts and the resolver's replies. The DNS replies may either be satisfied from the

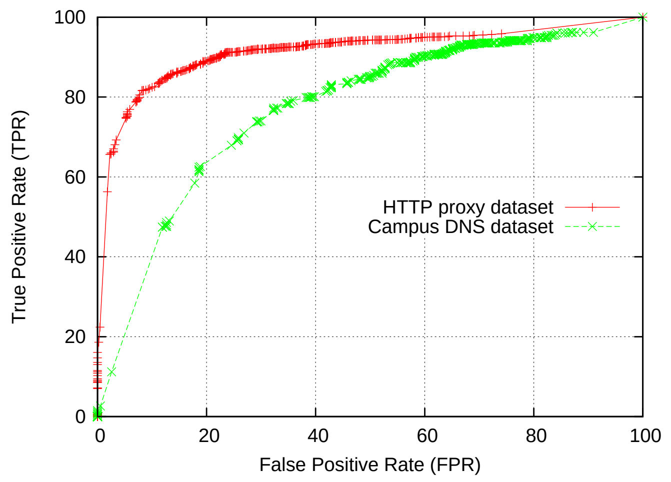


Fig. 23. Performance comparison of our approach on different datasets.

resolver's cache or obtained iteratively by querying the respective name servers. To build the new bipartite graph, we consider only the queries observed within the autonomous system, that is, those that flow between the hosts and the recursive DNS resolver.

The bipartite graph thus generated has 281K nodes with 1.27M undirected edges. Of the 281K nodes, 7.7K represent hosts with our network making DNS queries to one specific recursive resolver. The remaining nodes represent unique domains queried over the week long period. The ground truth utilizing the same blacklist and white-list (as for HTTP-proxy dataset) results in 5.2K domains labelled malicious and 3.8K domains labelled benign. Figure 23 compares the results from the BP algorithm on the campus data set, using the default parameters, against the HTTP dataset. The inferior performance for the campus dataset is attributed to lack of information due to limited time of capture and relatively small network size. We believe that the week long capture for the small network, results in a graph with many edges absent as compared to a dataset captured over a longer time period which expresses a wider variety of domains a user accesses. Therefore, the correct information in the form of beliefs does not propagate to the analyzed nodes, resulting in an inferior performance. In comparison, the HTTP-proxy dataset is

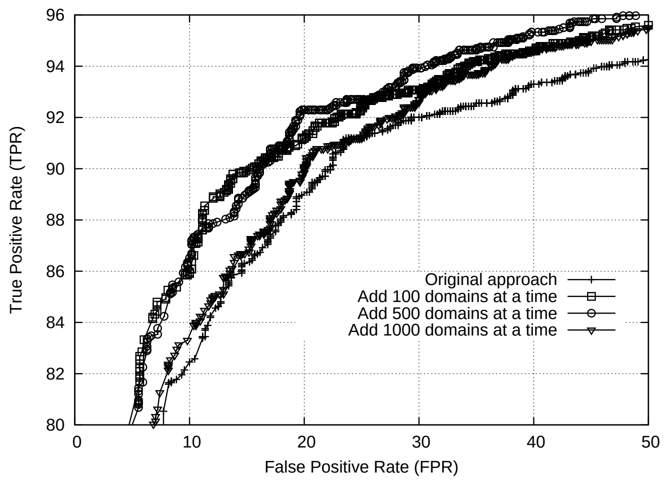
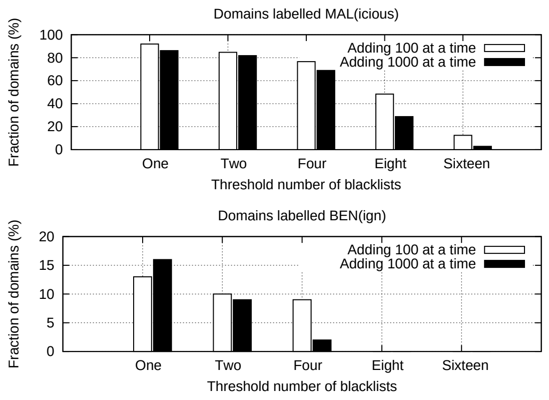


Fig. 24. Reinforcing ground truth with labelled domains. This figure measures the performance when adding domains qualitatively.

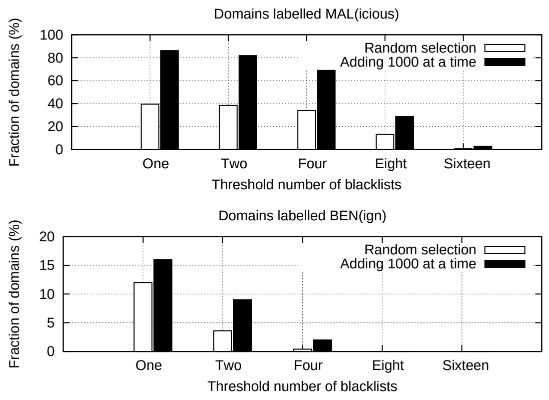
only a day long, but represents the behavior of more than 144K hosts. Therefore, a sound approach would evaluate the algorithm against a dataset which represents a variety of hosts in addition to being collected over a longer period of time. Such a dataset may be obtained at a Tier-1 ISP. The computational complexity, however, increases with increasing graph connections.

E. Reinforcing Detection via Qualitative Learning

We modify our technique to improve detection by adding to the ground truth, those domains labelled benign or malicious by our algorithm. Figure 20, the flowchart for reputation inference, shows the feedback mechanism incorporated to use labelled domains for future classification. An iteratively improving ground truth is useful for many reasons. For instance, domain fluxing botnets regularly change domain names that map to the Command and Control (C&C) server. An iterative learning technique such as ours would help detect the new set of malicious domains by virtue of bots contacting the same server (or IP address) over time. Therefore, the knowledge of malicious domains from the previous set



(a)



(b)

Fig. 25. (a) Qualitative comparison of labelled domains for ground truth updates with 100 and 1000 domains. (b) Randomly chosen domains with high/low degrees are compared with domains classified by our technique.

could now be used for identification of the new set of fluxing domains and possibly new bots.

To obtain the benefits of repeated ground truth updates, it is essential to ensure that the newly added knowledge is qualitatively superior. The reputation scores help us decide the qualitative aspect. We, therefore, use X number of most malicious and benign domains for addition to ground truth. An excessively large X may create noise while an extremely small X may not offer enough information. To understand these implications, we experiment with X as 100, 500, and 1000. That is, for $X = 500$, we add 500 of the most malicious and benign domains (ordered by belief values) to the ground truth and evaluate our mechanism again with a 10-fold cross validation.

The ROC plot in Figure 24 shows the advantages of iterative addition. First, we observe that adding labelled domains from ground truth considerably improves the performance. For instance, at a false positive rate of 10%, we note that for $X = 100$, there is about 4% improvement in detection rate when compared to a static ground truth. Also, this improvement over $X = 1000$ is about 3%. We note that the gain when adding 500 of the most benign/malicious domains is not too different from $X = 100$. For even smaller values of X, we expect the performance to degrade, converging towards using only a static ground truth.

In addition to the ROC based evaluation, we analyze the newly labelled domains by validating their reputation against external source such as *Robtex*. Note that the newly labelled domains are not part of the ground truth, but are obtained as a result of applying BP on the graph and retrieving most benign/malicious domain. Using a Java API [78], we automatically collect information about domains or IPs classified by our technique, for each of the two experiments where X is 100 and 1000. From the two experiments, we obtain 209 (out of 1000) domains labelled malicious, which are unique to each scenario. Similarly, the count for benign-classified domains is 100. Therefore, we collect *Robtex* information for unique domains only.

Figure 25(a) summarizes the qualitative distinction. The top histogram plot represents fraction of domains (out of 209) which are present in at least 1, 2, 4, 8, or 16 blacklists. As the figure shows, the malicious domains obtained for $X = 100$ experiment are consistently greater in different blacklists than for $X = 1000$. This is especially true for a blacklist threshold of 8 and 16 which we consider reliable cutoffs for validation. Specifically, for 8 as the blacklist threshold, there are two times as many malicious domains for $X = 100$ than for $X = 1000$. This ratio is almost four times for 16 as the minimum number of blacklists. For benign labelled domains, we note no domains present in at least 8 or 16 blacklists, unlike for a blacklist threshold of 4. On analyzing the exceptions in the benign list of domains for the blacklist threshold of 4, we note that there are 9 IPs for $X = 100$, all present in a /20 subnet.

Interestingly, our evaluation of malicious and benign domains reveals a distinct pattern based on the degree of the classified domain. To elaborate, we observe that the benign domains have high degrees while the malicious domains mostly have lower degrees. To validate that our approach is not a manifestation of a simple degree based classification, we analyze two sets of randomly chosen high degree and low degree domains present in the bipartite graph. The degree range for the two sets is same as that observed for the domains classified as benign/malicious with our technique. For every domain in the two sets, each with 250 randomly chosen domains, we again use *Robtex* to determine the number of blacklists that the domain is present in. Figure 25(b) highlights that the belief propagation algorithm identifies a greater fraction of domains correctly. Specifically, with a blacklist threshold of eight, we observe that our technique helps find *twice* as many domains when compared to a degree-based classifier. Similarly for benign domains (that is, high degree domains from the random set), both techniques do not have any false positives with 8 as the blacklist threshold.

Thus, reinforcing the propagation of beliefs through qualitative addition of new information, amounts to speeding up the detection mechanism as the classified domains can now be used as an additional source of information. The reinforced mechanism can therefore be adapted for online detection or integrated into early-warning systems where new domains or IPs can be used as hints for future infection vectors, enabling pre-emptive action. The reinforced learning is likely to offer significant benefit when the ground truth is small, as gained information from the BP algorithm can be a significant boost to the ground truth.

F. Discussion

Our bipartite graph is composed of domain nodes which represent second-level domains. Thus using belief propagation, we label the second-level domain and consequently all its sub-domains as malicious or benign. However, such an approach can mis-classify legitimate websites hosted by Content Distribution Networks (CDNs) or blogger hosts, in a scenario where a few hosted websites are actually malicious. For instance, a blogging host server such as *blogexample.com* can get labelled as malicious due to a rogue sub-domain such as *maliciousblog.blogexample.com*, although the CDN also hosts *benignblog1.blogexample.com, benignblog2.blogexample.com* etc. First, we note that CDNs are usually careful of what content is being hosted on their servers. However, instead of delegating responsibility to the CDN authority, we may instead use white-list or alternatively apply metrics which identify CDNs, deferring the computation of reputation for such groups. Alternatively, we can employ a demarcation technique such that only "high enough" beliefs are labelled as malicious, or benign if the computed belief is considerably small. The ones in between need not be labelled (as done with zone-based analysis). Another alternative is to use a mix of second-level and third-level domain nodes in the bipartite graph (assuming the availability of corresponding ground truth).

From our analysis we note that our graph is composed of nodes, some of which have a high degree. Also, due to relative prevalence of benign nodes compared to malicious nodes, the belief for a high degree node can get swayed such that the node is labelled as benign. The key here is to choose the correct parameters which best represents the present context. Also, the parameters (*prior* and the *edge potential*) applicable to our dataset's bipartite graph, may not be applicable to another graph or the same graph over time. Therefore, in addition to using general parameters for the whole graph, one can also split a bipartite graph into connected components (smaller bipartite graphs) and choose parameters corresponding to each connected component, increasing performance robustness.

The computation of beliefs using belief propagation and message passing over the bipartite graph with a large number of edges consumes a considerable amount of time. However, as messages may not traverse from one connected component to the other (due to absence of a connecting edge), we may apply belief propagation to individual connected components only to reduce the latency of label inference for a graph node. We also note that in an attempt to defeat our inference approach, an attacker can segregate its bots (infected slave hosts) so that there is no path, limiting the inference due to restricted flow of information. However, this seriously curbs the functionality and flexibility of the master (or botnet owner).

In the previous sections, we highlight that excessive paths can result in undesired propagation of noise through the bipartite graph. While removing edges is a possible solution to curbing noise, alternatively we tried to slow the propagation by modifying messages. Specifically, we use square and cube rooted substitutes for the original message in our analysis. Such messages are less influenced by node degrees (high degree nodes have very small messages) and reduce the impact of noise. This experiment, however, improves performance only for high false positive rates rendering this alteration useless. We believe that the degradation occurs due to slowing down of the correct informational messages.

G. Related Work

Domain analysis: Determination of a domain's reputation has been studied extensively. The authors in [8] predict a domain's reputation based on network-based, zone-based and evidence-related features and thus requires a considerable effort in establishing the ground truth. In addition, they use both supervised and unsupervised heuristics to designate domains with reputation score. Feature extraction from DNS traffic and classification of URLs has been used in [79] which again uses machine learning techniques for malicious domain identification. The attributes require external collection of a test domain's IP address and TTL information, thereby increasing resource requirement. [61] explores the entropy present in a group of domains where domain fluxing botnets are characterized by higher entropy. Their analysis, however, requires a larger set of domains (more than 100) for reliable detection. Also, the authors in [61] focus on second-level domain, an IP or connected component (of IP and domain) detection, while our approach is used to identify individual domains and can be extended for characterizing URLs. More recently, authors in [80] highlight how cached malicious domains do not disrupt botnet functioning in spite of the malicious domains removed from their TLD hosts.

URL analysis: Ma et al. [81] classify websites based on URL characteristics. Their approach, however, relies on aggregating additional information such as host IP and geographical properties (as noted for [8]. Similarly, [82] classifies spam URLs again using supervised learning algorithms derived from features which use web browser behavior, DNS, and the corresponding IP information. Alphanumeric character distributions for distinguishing legitimate from phishing URLs, has been studied in [37]. Another approach explores generating spamming botnet signatures by analyzing a corpus of spam URLs [42]. The technique generates regular expressions identifying a particular campaign and therefore correlating fluxing botnets is difficult for this technique.

Host analysis: Malware and bot detection within an autonomous system has seen a significant interest. For instance, authors in [2] analyze the network traffic looking for abnormal *crowd density* (in a set of network flows) and *crow homogeneity* (similarity in response messages) for identifying the C&C server and the bots within the network. Similarly, [66] looks at the correlation between malign activities that result in infection. Specifically, the series of events such as egg download, contact with the C&C, and the subsequent attack are used as indicators of anomaly. Note that these approaches require raw network data analysis affecting scalability. The technique in [83], however, looks only at a specific malicious seed set, identifying patterns and consequently the infected hosts which match those patterns. The success of this approach relies on a comprehensive seed pool set. Behavior of known infected hosts has also been studied in [84] where a malware is subjected to alternate network responses in an attempt to understand its behavior and thus use new information for predicting future attacks.

Graph analysis: Graph analysis for anomaly detection has been studied in [62]. The authors in [62] analyze the graph structure for hosts and failed DNS queries and highlight tight clusters as indicators of an anomaly. [85] produces blacklists of attack sources by performing a link analysis. The relationship defined by links provides relevance of attack sources to networks. Coupled with the attack severity as inputs, the algorithm outputs the rank assigned to an attack source in a blacklist. Correlation of temporal and entropy-based features of DNS failures for botnet based C&C (domain/IP) detection, has been studied in [77]. Although we do not utilize DNS failures, our approach may be leveraged by such failures leading to a bigger set of *a priori* information.

Belief propagation has previously been used for anomaly detection. [86] uses the BP technique for detecting only the infected hosts within the network by applying the inference over a bipartite graph of hosts and resident files. Inference for multi-class problems using belief propagation such as that for Twitter, has been studied in [87]. Inference over hyper links between web pages for determining the “importance”, is done using the PageRank algorithm [88]. TrustRank, on the other hand, determines spam pages by propagating trust over the web-graph. The belief propagation algorithm is similar to the PageRank as both look for quality of the inbound links, to identify the reputation of a given page. The belief propagation algorithm, however, computes the marginal probability in contrast to PageRank which only propagates a single state information along the graph edges. Note that TrustRank follows a procedure similar to that of PageRank. [89] detects P2P based bots by analyzing *mixing rates* for random walks over the graph topology. With respect to applying belief propagation, the authors in [90] provide useful insights for choosing the key parameters for the algorithm.

CHAPTER VI

CONCLUSION AND FUTURE WORK

In this work, we present several scalable techniques for anomaly detection. First, we propose using the DNS traffic for identifying a wide variety of anomalies. The use of DNS is motivated by DNS traffic comprising only about 2% of the total network traffic. We then demonstrate MiND, a tool for detecting misdirected DNS packets occurring within an autonomous system, due to a poisoned resolver cache or due to malware induced host modification. The MiND system comprises of the DNS query verifier, which identifies anomalies utilizing a continuously updating database containing authoritative name server information. Our results indicate that we are able to detect all misdirected packets. Our analysis results in a false positive rate of less than 0.8%, which can be further reduced to 0.046% by autonomous system checks. As a future work, we plan to analyze the incoming DNS response packets to detect and thwart poisoning attempts directed towards a recursive DNS resolver. We also plan to the evaluate the effect of using BGP prefix information, with different prefixes for DNS packet validation.

We also propose a methodology for detecting algorithmically generated domain names as used for "domain fluxing" by several recent Botnets. We propose statistical measures such as Kullback-Leibler divergence, Jaccard index, and Levenshtein edit distance for classifying a group of domains as malicious (algorithmically generated) or not. We perform a comprehensive analysis on several data sets including a set of legitimate domain names obtained via a crawl of IPv4 address space as well as DNS traffic from a Tier-1 ISP in Asia. One of our key contributions is the relative performance characterization of each metric in different scenarios. In general, the Jaccard measure performs the best, followed by the Edit distance measure, and finally the K-L divergence. Furthermore, we show how our methodology when applied to the Tier-1 ISP's trace was able to detect Conficker as well as a botnet yet unknown and unclassified, which we call as *Mjuyh*. In this regards, our methodology can be used as a first alarm to indicate the presence of domain fluxing in a network, and thereafter a network security analyst can perform additional forensics to infer the exact algorithm being used to generate the domain names. As future work, we plan to generalize our metrics to work on n-gram for values of $n > 2$.

We extend the previous technique utilizing failed domain names in the quest for rapid detection of a fluxing botnet's C&C server, the bots within the local network, and the related domain names, and thus revealing the botnet infrastructure. Utilizing only DNS traffic, we reduce the resource requirement for botnet detection. We also considerably reduce the latency of detection when compared to previous techniques. For faster detection, we utilize not only the entropy of the domain names successfully mapping to an IP address, but also that of the correlated failed DNS queries occurring within the vicinity of the succeeding DNS query. With our technique, we achieve a false positive rate as low as 0.02% with a high detection rate. As a future work, we plan to utilize SERVER FAILURE based DNS failures, or failures related to the name servers, as a means for detecting botnets which exhibit double fast flux.

We propose another technique to scale network analysis further. We present a technique to check malfeasance by identification of malicious domains that infected host access (or may access). We infer a host or domain to be malicious (or benign), with the belief propagation algorithm applied to a bipartite graph generated from a huge dataset of HTTP and DHCP logs and to a campus DNS dataset. We justify the parameters chosen for inducing correct beliefs and present tradeoffs for variants of those parameters. With a small ground truth for domains, our results verify labelling of graph nodes with high detection rates for low false positive rates and low false discovery rates. We also present a modification where the classified domains may be used as a source of information resulting in an improved performance. For future work, we plan to experiment with several variations of our approach.

We plan to focus on problem of identifying *sub-categories* of malicious domains or hosts such as whether an infected host is part of DDoS botnet, spews only spam, or is involved in other malicious activity. We also plan to evaluate a real-time approach by choosing parameters for faster convergence along with splitting huge bipartite graph into smaller components and thus distributing belief propagation analysis to independent components. Another interesting application of our technique is using belief propagation to determine advertisement networks through inference from white-listed or popular domains. In addition, we plan to use several (non-) linear prior assignment functions to evaluate the effect on performance.

REFERENCES

[1] "VM Detection by In-The-Wild Malware," http://www.networkforensics.com/2010/12/13/vm-detection-by-in-the-wild-malware/.

[2] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic," *Proc. of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[3] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection," *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[4] L. Braun, G. Munz, and G. Carle, "Packet Sampling for Worm and Botnet Detection in TCP Connections," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Osaka, Japan, 2010.

[5] "DNS Changer Malware," www.fbi.gov/DNS-changer-malware.pdf.

[6] 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 *ACM Conference on Computer and Communications Security (CCS)*, November 2009.

[7] M. Antonakakis, R. Perdisci, W. Lee, N. V. II, and D. Dagon, "Detecting Malware Domains at the Upper DNS Hierarchy," *USENIX Security Symposium*, 2011.

[8] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a Dynamic Reputation System for DNS," *USENIX Security Symposium*, 2010.

[9] D. Dagon, N. Provos, C. P. Lee, and W. Lee, "Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority," *Networks and Distributed Security Symposium*, 2008.

[10] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose, "All Your iFRAMEs Point to Us," *ACM Security Symposium*, pp. 1–15, 2008.

[11] "DNSSEC," http://www.ripe.net/training/dnssec/material/dnssec.pdf.

[12] R. Aitchison, "A Case against DNSSEC," 2007, http://www.circleid.com/posts/070814_case_against_dnssec/.

[13] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," 2005, http://tools.ietf.org/html/rfc4033.

[14] D. Dagon, M. Antonakakis, and P. Vixie, "Increased DNS Forgery Resistance Through 0x20-Bit Encoding," *Computer and Communications Security*, pp. 211–222, 2008.

[15] R. Perdisci, M. Antonakakis, X. Luo, and W. Lee, "WSEC DNS: Protecting Recursive DNS Resolvers from Poisoning Attacks," *Dependable Systems and Networks*, pp. 3–12, 2009.

[16] L. Yuan, K. Kant, P. Mohapatra, and C.-N. Chuah, "DoX: A Peer-to-Peer Antidote for DNS Cache Poisoning Attacks," *IEEE Communications*, pp. 2345–2350, June 2006.

[17] D. Leonard and D. Loguinov, "Turbo King: Framework for Large-Scale Internet Delay Measurements," *INFOCOM*, pp. 31–35, 2008.

[18] "DNS Best Practices, Network Protections, and Attack Identification," http://www.cisco.com/web/about/security/intelligence/dns-bcp.html.

[19] S. Friedl, “An Illustrated Guide to the Kaminsky DNS Vulnerability,” 2008, http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html.

[20] H. Ballani and P. Francis, “Mitigating DNS DoS attacks,” *Computer and Communications Security*, pp. 189–198, 2008.

[21] “Google DNS,” http://code.google.com/speed/public-dns/.

[22] D. Barr, “Common DNS Operational and Configuration Errors,” 1996, http://http://tools.ietf.org/html/rfc1912.

[23] “IANA IPv4 Address Space Registry,” http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml.

[24] “IANA Root Zone Database,” http://www.iana.org/domains/root/db/.

[25] “TLD List,” https://wiki.mozilla.org/TLD_List.

[26] B. Barney, “POSIX Threads Programming,” https://computing.llnl.gov/tutorials/pthreads/.

[27] “Programming with pcap,” http://www.tcpdump.org/pcap.htm.

[28] C. Liu and P. Albitz, *DNS and BIND*. O’Reilly Media, Inc., 2006.

[29] “SURBL,” http://www.surbl.org.

[30] “McAfee TrustedSource,” http://www.trustedsource.org.

[31] P. Porras, H. Saidi, and V. Yegneswaran, “An Analysis of Conficker’s Logic and Rendezvous Points,” Tech. Rep., March 2009.

[32] “Conficker C Analysis,” http://mtc.sri.com/Conficker/addendumC/.

[33] "Twitter API Still Attracts Hackers," http://blog.unmaskparasites.com/2009/12/09/twitter-api-still-attracts-hackers/.

[34] "PC Tools Experts Crack new Kraken," http://www.pctools.com/news/view/id/202/.

[35] "On Kraken and Bobax Botnets," http://www.damballa.com/downloads/r_pubs/Kraken_Response.pdf.

[36] H. Crawford and J. Aycock, "Kwyjibo: Automatic Domain Name Generation," in *Software Practice and Experience, John Wiley & Sons, Ltd.*, 2008.

[37] D. K. McGrath and M. Gupta, "Behind Phishing: An Examination of Phisher Modi Operandi," *Proc. of USENIX workshop on Large-scale Exploits and Emergent Threats (LEET)*, April 2008.

[38] S. S. J. Ma, L.K. Saul and G. Voelker, "Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs," *Proc. of ACM KDD*, July 2009.

[39] R. Perdisci, I. Corona, D. Dagon, and W. Lee, "Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces," in *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, December 2009, pp. 311 –320.

[40] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi, "FluXOR: Detecting and Monitoring Fast-Flux Service Networks," in *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 186–206. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-70542-0_10

[41] M. Konte, N. Feamster, and J. Jung, "Dynamics of Online Scam Hosting Infrastructure," *Passive and Active Measurement Conference*, 2009.

[42] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov, “Spamming Botnets: Signatures and Characteristics,” *ACM SIGCOMM Computer Communication Review*, 2008.

[43] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum, “BotGraph: Large Scale Spamming Botnet Detection,” *USENIX Symposium on Networked Systems and Design Implementation (NSDI ’09)*, 2009.

[44] A. Bratko, G. V. Cormack, D. R, B. Filipi, P. Chan, T. R. Lynam, and T. R. Lynam, “Spam Filtering using Statistical Data Compression Models,” *Journal of Machine Learning Research*, vol. 7, pp. 2673–2698, 2006.

[45] R. Perdisci, G. Gu, and W. Lee, “Using an Ensemble of One-Class SVM Classifiers to Harden Payload-based Anomaly Detection Systems,” *In Proceedings of the IEEE International Conference on Data Mining (ICDM ’06)*, 2006.

[46] S. Gianvecchio, M. Xie, Z. Wu, and H. Wang, “Measurement and Classification of Humans and Bots in Internet Chat,” *In Proceedings of the 17th USENIX Security Symposium (Security ’08)*, 2008.

[47] H. L. V. Trees, “Detection, Estimation and Modulation Theory,” *Wiley*, 2001.

[48] T. Cover and J. Thomas, “Elements of Information Theory,” *Wiley*, 2006.

[49] T. Holz, M. Steiner, F. Dahl, E. W. Biersack, and F. Freiling, “Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm,” *In First Usenix Workshop on Large-scale Exploits and Emergent Threats (LEET)*, April 2008.

[50] P. Porras, H.Saidi, and V. Yegneswaran, “Conficker C P2P protocol and implementation,” *SRI International Tech. Report*, September 2009.

[51] J. Stewart, “Inside the Storm: Protocols and Encryption of the Storm Botnet,” *Black Hat Technical Security Conference, USA*, 2008.

[52] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy, “Studying Spamming Botnets Using Botlab,” in *NSDI: Proceedings of the 6th USENIX symposium on Networked Systems Design and Implementation*, 2009.

[53] C. D. Manning, P. Raghavan, and H. Schutze, “An Information to Information Retrieval,” *Cambridge University Press*, 2009.

[54] “Botlab,” http://botlab.org/.

[55] “Yahoo Webspam Database,” http://barcelona.research.yahoo.net/webspam/datasets/uk2007/.

[56] “Web of Trust,” http://mywot.com.

[57] “McAfee Site Advisor,” http://www.siteadvisor.com.

[58] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci, “Unconstrained Endpoint Profiling: Googling the Internet,” in *ACM SIGCOMM*, August 2008.

[59] R. T. Jerome Friedman, Trevor Hastie, “glmnet: Lasso and elastic-net Regularized Generalized Linear Models,” Tech. Rep.

[60] “Win32/hamewq,” http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Win32/Hamweq.

[61] S. Yadav, A. K. K. Reddy, A. L. N. Reddy, and S. Ranjan, “Detecting Algorithmically Generated Malicious Domain Names,” *Internet Measurement Conference*, 2010.

[62] N. Jiang, J. Cao, Y. Jin, L. E. Li, and Z.-L. Zhang, "Identifying Suspicious Activities Through DNS Failure Graph Analysis," *IEEE Conference on Network Protocols*, 2010.

[63] Z. Zhu, V. Yegneswaran, and Y. Chen, "Using Failure Information Analysis to Detect Enterprise Zombies," *Security and Privacy in Communication Networks*, 2009.

[64] R. Villamarín-Salomón and J. C. Brustoloni, "Identifying Botnets Using Anomaly Detection Techniques Applied to DNS Traffic," *Consumer Communications and Networking Conference*, 2008.

[65] R. Villamarin-Salomon and J. C. Brustoloni, "Bayesian Bot Detection Based on DNS Traffic Similarity," in *Proceedings of the 2009 ACM symposium on Applied Computing*, ser. SAC '09. New York, NY, USA: ACM, 2009, pp. 2035–2041. [Online]. Available: http://doi.acm.org/10.1145/1529282.1529734

[66] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation," *Proc. of the 16th USENIX Security Symposium (Security'07)*, August 2007.

[67] S. Yadav and A. L. N. Reddy, "MiND : Misdirected dNs packet Detector," *IASTED Computer and Information Security*, 2010.

[68] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. de Groot, and E. Lear, "Address Allocation for Private Internets," 1996, http://www.ietf.org/rfc/rfc1918.txt.

[69] "Conficker Working Group," http://www.confickerworkinggroup.org/wiki/pmwiki.php/ANY/FAQ#toc5.

[70] "New Technique Spots Sneaky Botnets," http://mobile.darkreading.com/9292/show/4711c9403b772e7281ae08cee69758cc\&t=461a4a89abc0a0c761234d11086f5003.

[71] "Symantec Internet Security Threat Report, vol 17, 2011." http:/www.symantec.com/content/en/us/enterprise/other_resources/b-istr/_main_report_2011_21239364.en-us.pdf.

[72] "Malware Delivered by Yahoo, Fox, Google ads," http://news.cnet.com/8301-27080_3-20000898-245.html.

[73] J. Pearl, "Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach," in *Proceedings of the American Association of Artificial Intelligence National Conference on AI*, Pittsburgh, PA, 1982, pp. 133–136.

[74] J. Yedida, W. Freeman, and Y. Weiss, "Understanding Belief Propagation and its Generalizations," *Exploring Aritificial Intelligence in the New Millennium*, pp. 236–239, 2003.

[75] "Alexa: Top Sites," http://www.alexa.com/topsites.

[76] "Guava: Google Core Libraries for Java 1.5+," http://code.google.com/p/guava-libraries/.

[77] S. Yadav and A. L. N. Reddy, "Winning with DNS Failures: Strategies for Faster Botnet Detection," *Security and Privacy in Communication Networks (SecureComm)*, 2011.

[78] "HtmlUnit: GUI-Less Browser for Java Programs," http://htmlunit.sourceforge.net/.

[79] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "EXPOSURE: Finding Malicious Domain Using Passive DNS Analysis," *Network and Distributed System Security Symposium*, 2011.

[80] J. Jiang, J. Liang, K. Li, J. Li, H. Duan, and J. Wu, “Ghost Domain Names: Revoked Yet Still Resolvable,” in *Proceedings of the 19th Annual Network & Distributed System Security Symposium*. Internet Society, 2012.

[81] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs,” in *ACM SIGKDD*, June 2009.

[82] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, “Design and Evaluation of a Real-Time URL Spam Filtering Service,” *IEEE Security and Privacy*, 2011.

[83] G. Stringhini, T. Holz, B. Stone-Gross, C. Kruegel, and G. Vigna, “BOTMAGNIFIER: Locating Spambots on the Internet,” *USENIX Security Symposium*, 2011.

[84] Y. Nadji, M. Antonakakis, R. Perdisci, and W. Lee, “Understanding the Prevalence and Use of Alternative Plans in Malware with Network Games,” in *Proceedings of the 27th Annual Computer Security Applications Conference*, ser. ACSAC ’11. New York, NY, USA: ACM, 2011, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/2076732.2076734

[85] J. Zhang, P. Porras, and J. Ullrich, “Highly Predictive Blacklisting,” in *Proceedings of the 17th conference on Security symposium*, ser. SS’08. Berkeley, CA, USA: USENIX Association, 2008, pp. 107–122. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496711.1496719

[86] D. H. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, “Polonium: Tera-Scale Graph Mining for Malware Detection,” *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2010.

[87] U. Kang, D. H. Chau, and C. Faloutsos, “Mining Large Graphs: Algorithms, Inference, and Discoveries,” *International Conference of Data Engineering*, 2011.

[88] S. Brin and L. Page, “The Anatomy of a Large-scale Hypertextual Web Search Engine,” *Comput. Netw. ISDN Syst.*, vol. 30, pp. 107–117, April 1998. [Online]. Available: http://dx.doi.org/10.1016/S0169-7552(98)00110-X

[89] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov, “BotGrep: Detecting P2P Botnets Using Structured Graph Analysis,” *USENIX Security Symposium*, 2010.

[90] K. Murphy, Y. Weiss, and M. Jordan, “Loopy Belief Propagation for Approximate Inference: An Empirical Study,” *Uncertainity in Artificial Intelligence*, 1999.