MiND : Misdirected dNs packet Detector

Sandeep Yadav and A.L. Narasimha Reddy
Department of Electrical and Computer Engineering
Texas A&M University
College Station, TX 77843, USA
Email: sandeepy@tamu.edu, reddy@ece.tamu.edu

ABSTRACT
In this paper, we present MiND, a tool to detect DNS packet indirection attacks within an autonomous system (AS). 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 domains within our autonomous system. Our analysis using MiND 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.

KEY WORDS
Database, DNS, Indirection, MiND, Poisoning

1 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 [16]. 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 paper, 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 in the DNS packet. Guessing the correct ID makes the recursive resolver accept the 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. The recently discovered Kaminsky attack [17], on the contrary, demonstrates quick poisoning of a resolver cache.

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 [21].

Many solutions are proposed to counter DNS cache poisoning and indirection. However, most have not been widely adopted. For instance, DNSSEC [2, 10, 11] 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 very 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 paper 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.

The remaining sections of the paper are organized as follows. Section 2 covers the related work where we review already implemented solutions and research in context of DNS based cache poisoning and indirection. The threat model is described in section 3. Section 4 details our approach. Results are detailed in section 5 and finally conclusion and future work is presented in section 6.

2 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. [16] describe and define the problem of DNS cache poisoning and indirection. The authors in [15] use 0x20-bit encoding scheme to encode the question section within the DNS packet and hence counter DNS cache poisoning. Their method increases the difficulty for the attacker as she has to guess more entities with respect to the packet, within the round trip time of a DNS reply. WSEC-DNS [20] utilizes the wild card capability of CNAME records to increase cache poisoning difficulty significantly. Yuan et al. [22] 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 [18]. These delay measurements help in outlining timeout requirements of various dependent applications. These results guide our work as explained later in the paper. DNS TXID and port randomization [1] are two of the most prevalent techniques for preventing cache poisoning. However, with the advent of Kaminsky attack, the said measures have become obsolete [17].

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 poisoning attempts.

The employment of a secondary cache of expired DNS records is proposed to mitigate the affects of DoS attacks on DNS infrastructure [12]. 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 [12].

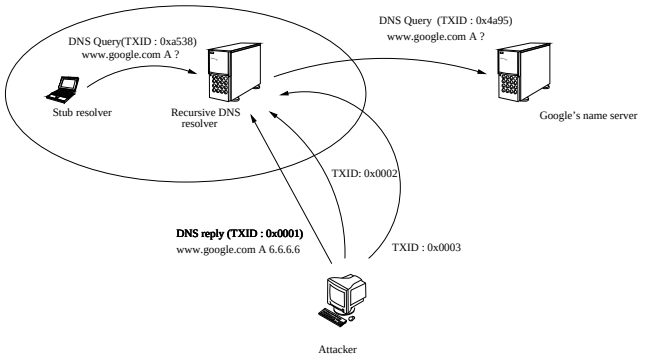


Figure 1. DNS cache poisoning scenario.

3 Threat model

Cache poisoning attacks can be initiated from outside or from within an autonomous system. Figure 1 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 [15, 20] 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.

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

4 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 section 4.3, 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

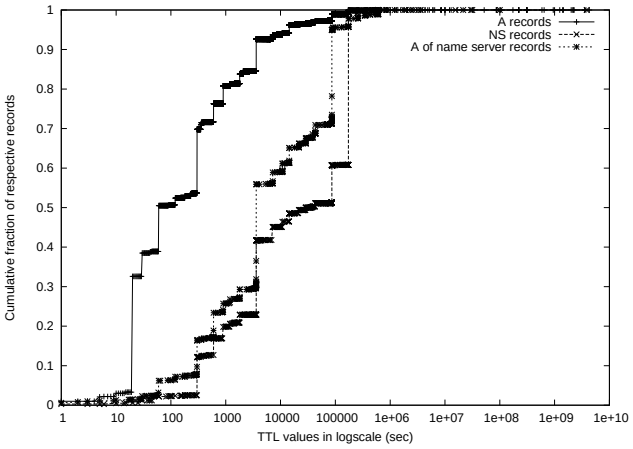


Figure 2. Cumulative Distribution Function (CDF) of TTL values for A, NS, and A of name server records.

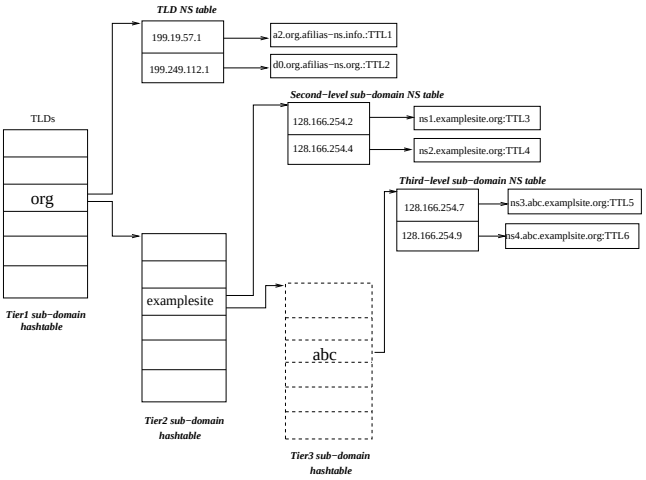


Figure 4. MiND query verifying data structure.

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 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 sub-sections discuss each component of the MiND system in detail. We also outline the benefits and potential issues of our technique. It is important to note the terminology we use in the sections ahead. For a hostname 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*.

4.1 Name server database feasibility

Figure 2 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 DNS query verification over longer intervals, making our approach scalable.

4.2 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 3. 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 [3] 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 4, the data structure is organized into different tiers, with each tier containing name server information relevant to the corresponding sub-domain. For query verification, using the name server information for upto 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 net-

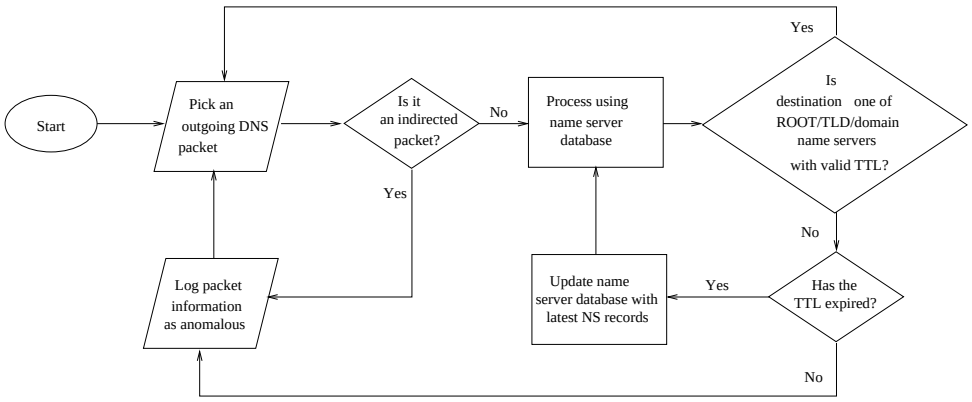


Figure 3. MiND flowchart.

works 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 sub-domains 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 5 highlights the benefits obtained by using such an expired cache.

If a packet matches any of the validation criteria as outlined in Figure 3, 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 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 countermechanism for Kaminsky style attacks[14]. 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.

4.3 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 [4]. 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

...

Figure 5. Sample line from a name server database file.

Figure 5 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 [5, 9] 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. Instead, once a database is built, only a small fraction of it needs to be updated frequently thereafter.

4.3.1 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 external 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 [22] 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 5). 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, 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.

4.4 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. However, we note that incoming DNS replies may also be spoofed and thus the DNS records may serve

Table 1. 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)

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.

5 Results

5.1 System requirements

The name server database collection engine is a multi-threaded application written using C and C++, and the commonly available resolver libraries (*libresolv*) and packet capture libraries [13, 7, 19]. 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 much 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.

5.2 Performance analysis

Table 1 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 1 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 AS. With time, we expect the fraction of false positives to decrease as more name servers get aggregated in the expired cache.

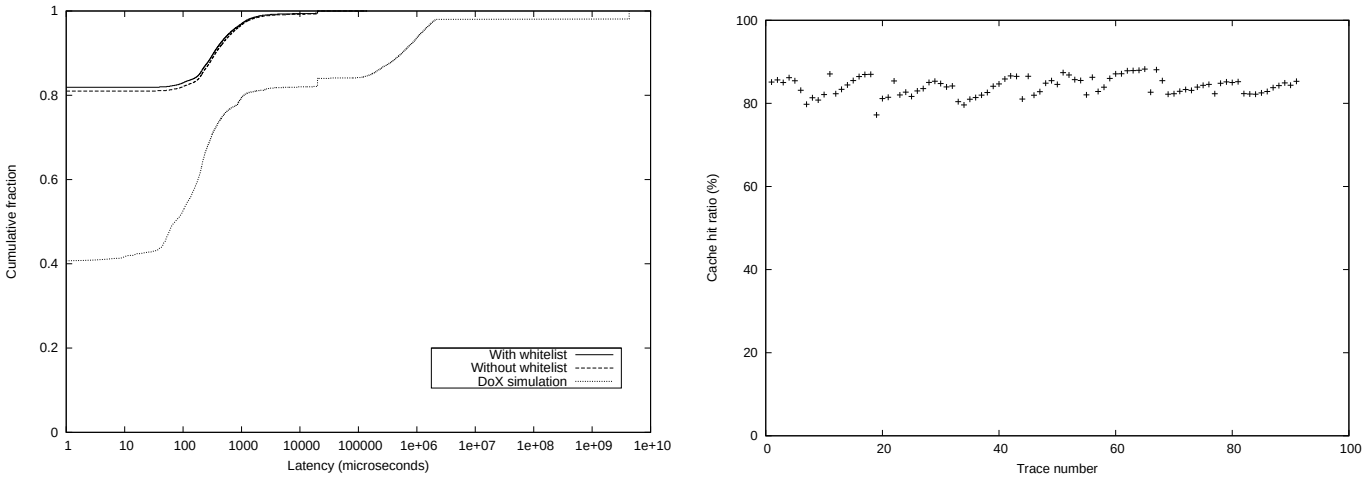
We further analyze the packets remaining from above, specifically for experiment 3. From our analysis we find that 15.38% of the anomalous packets, belong to malicious domains. The malicious nature of the suspect domains is confirmed with a domain reputation verification service [8]. The bulk of the malicious packets belong to *exitguide.ru*. We further confirm the malfeasance of this domain using McAfee's domain reputation checker [6] 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.

5.3 Latency of query verification

Figure 6(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 [22]. 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 hun-



(a) (b)

Figure 6. (a) Latency of verification. (b) Cache hit ratios for a week long dataset.

dreds of domains, the latency gain would be much higher.

5.4 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.

5.5 Highly dynamic name servers

We use each experiment (as in table 1) 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 5 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 asked domain names. However, such an analysis can also reveal rogue domains which exhibit domain fast fluxing behavior. 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.

5.6 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 5 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 *facebook.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.

5.7 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 6(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.

6 Conclusions and Future Work

In this paper, we have demonstrated 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. Our tool also helps in discovering malicious domains prevalent within our local network. We also show that the latency of maintaining a database and packet verification is lower than previously proposed solutions.

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.

7 Acknowledgements

We thank the Computing and Information Systems, Texas A&M University, for their help in providing the DNS traces. This research is supported in part by a Qatar National Research Foundation grant, Qatar Telecom and NSF grants 0702012 and 0621410.

References

[1] *DNS best practices, network protections, and attack identification*, `http://www.cisco.com/web/about/security/intelligence/dns-bcp.html`.

[2] *DNSSEC*, `http://www.ripe.net/training/dnssec/material/dnssec.pdf`.

[3] *Google DNS*, `http://code.google.com/speed/public-dns/`.

[4] *IANA IPv4 Address Space Registry*, `http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml`.

[5] *IANA Root Zone Database*, `http://www.iana.org/domains/root/db/`.

[6] *McAfee TrustedSource*, `http://www.trustedsource.org`.

[7] *Programming with pcap*, `http://www.tcpdump.org/pcap.htm`.

[8] *SURBL*, `http://www.surbl.org`.

[9] *TLD List*, `https://wiki.mozilla.org/TLD_List`.

[10] Ron Aitchison, *A Case against DNSSEC*, 2007, `http://www.circleid.com/posts/070814_case_against_dnssec/`.

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

[12] Hitesh Ballani and Paul Francis, *Mitigating DNS DoS attacks*, Computer and Communications Security (2008), 189–198.

[13] Blaise Barney, *POSIX Threads Programming*, `https://computing.llnl.gov/tutorials/pthreads/`.

[14] D. Barr, *Common DNS Operational and Configuration Errors*, 1996, `http://http://tools.ietf.org/html/rfc1912`.

[15] David Dagon, Manos Antonakakis, and Paul Vixie, *Increased DNS Forgery Resistance Through 0x20-Bit Encoding*, Computer and Communications Security (2008), 211–222.

[16] David Dagon, Niels Provos, Christopher P. Lee, and Wenkee Lee, *Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority*, Networks and Distributed Security Symposium (2008).

[17] Steve Friedl, *An Illustrated Guide to the Kaminsky DNS vulnerability*, 2008, `http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html`.

[18] Derek Leonard and Dmitri Loguinov, *Turbo King: Framework for Large-Scale Internet Delay Measurements*, INFOCOM (2008), 31–35.

[19] Cricket Liu and Paul Albitz, *DNS and BIND*, O'Reilly Media, Inc., 2006.

[20] Roberto Perdisci, Manos Antonakakis, Xiapu Luo, and Wenkee Lee, *WSEC DNS: Protecting Recursive DNS Resolvers from Poisoning Attacks*, Dependable Systems and Networks (2009), 3–12.

[21] Niels Provos, Panayiotis Mavrommatis, Mohbeeb Abu Rajab, and Fabian Monrose, *All your iFRAMEs point to Us*, ACM Security Symposium (2008), 1–15.

[22] Lihua Yuan, Krishna Kant, Prasant Mohapatra, and Chen-Nee Chuah, *DoX: A Peer-to-Peer Antidote for DNS Cache Poisoning Attacks*, IEEE Communications (2006), 2345–2350.