Increased DNS Forgery Resistance Through 0x20-Bit Encoding

SecURItY viA LeET QueRieS

David Dagon
Georgia Institute of Technology
dagon@cc.gatech.edu

Manos Antonakakis
Georgia Institute of Technology
manos@cc.gatech.edu

Paul Vixie
Internet Systems Consortium
Paul_Vixie@isc.org

Tatuya Jinmei
Internet Systems Consortium
Jinmei_Tatuya@isc.org

Wenke Lee
Georgia Institute of Technology
wenke@cc.gatech.edu

ABSTRACT

We describe a novel, practical and simple technique to make DNS queries more resistant to poisoning attacks: mix the upper and lower case spelling of the domain name in the query. Fortuitously, almost all DNS authority servers preserve the mixed case encoding of the query in answer messages. Attackers hoping to poison a DNS cache must therefore guess the mixed-case encoding of the query, in addition to all other fields required in a DNS poisoning attack. This increases the difficulty of the attack.

We describe and measure the additional protections realized by this technique. Our analysis includes a basic model of DNS poisoning, measurement of the benefits that come from case-sensitive query encoding, implementation of the system for recursive DNS servers, and large-scale real-world experimental evaluation. Since the benefits of our technique can be significant, we have simultaneously made this DNS encoding system a proposed IETF standard. Our approach is practical enough that, just weeks after its disclosure, it is being implemented by numerous DNS vendors.

General Terms

DNS, DNS poisoning, DNS transaction security, DNS forgery resistance, protocol security, network security, DNS security

1. INTRODUCTION

CCS '08, Virginia USA

DNS poisoning attacks present a persistent, ongoing threat to server operations. While there are a variety of DNS poisoning techniques, those directed at large cache servers often use two steps: (a) they force the recursive server to perform a lookup; and then (b) spoof misleading DNS answers, using the source address of the authority server. A successful attacker can change a DNS cache entry, and redirect all users of the victim DNS server to arbitrary proxy locations. When done to obtain transactional information (e.g., banking), this technique is called pharming (or large-scale phishing) [27].

Numerous solutions have been proposed to prevent DNS poisoning, e.g., whitelisting [14], cryptographic systems [33], and many client-based systems have been suggested. Solutions requiring changes to the DNS infrastructure, however, face larger hurdles in deployment. For example, DNSSEC [5] and DLV [34] use cryptography to provide strong DNS messaging integrity. However, these approaches require significant changes to the world's DNS infrastructure: the signing of zones, the creation of policies to manage those keys, and the deployment of DNSSEC-aware clients and servers.

Other DNS security solutions contemplate even larger changes to the network infrastructure, e.g., the creation of DHT-based naming or cooperative naming systems that replace DNS [24, 37]. Even if these systems prevent poisoning, they are more likely to find adoption in new, developing network architectures, such as P2P systems, compared to existing network systems. DNS is so widely used, deployed in tens of millions of systems, and so central to every other protocol, that one must expect it will survive the creation of novel replacement solutions.

The goal of our work is to devise *practical* security solutions for DNS that make resolvers more resistant to poisoning. Specifically, we desire the creation of DNS light-weight forgery-resistance technology that has several properties:

1. *No Radical Changes.* DNS improvements should ideally require no large-scale replacement or modification of existing DNS infrastructure. (If large changes were needed, one could

argue that zone owners should instead just deploy DNSSEC.)

2. *Protocol Stability*. Improvements should require no alteration of the DNS protocol, which would in turn require reimplementation of DNS server and client code. (Surveys have shown there are tens of millions of DNS servers deployed world-wide, many on embedded devices [8, 25]. Amending them to handle a new protocol is likely cost-prohibitive.)

3. *Backward Compatible*. Any improvements should be optional, and not disrupt other technologies that rely on existing DNS standards.

We present a defense technique against poisoning that satisfies these requirements. We propose the mixed-case encoding of query and reply messages between recursive and authority servers. For example, instead of querying for `www.example.com`, recursive DNS servers would instead query for `wwW.eXamPLe.cOM`, or some other pattern of case variations.

Since *almost all* authority DNS servers preserve the case encoding of DNS queries, bit-for-bit, as presented by the recursive server, only the recursive servers need to change how they format questions.

The pattern of mixed-case encoding of domain names, unique to each transaction between DNS initiators and responders, provides an additional means to track messages between servers. We call our encoding system "DNS-0x20" after the bit position used to manipulate case.

The main contributions of this paper include:

- We propose DNS-0x20, a simple change to the formatting of DNS queries. We have implemented DNS-0x20, and have offered the technology as an IETF standards proposal [32]. At this writing, the proposal has progressed to working group status. As further proof that our scheme is practical, workable, and useful, numerous DNS vendors (at this writing) are now incorporating DNS-0x20 encoding into their servers and products–just weeks after the idea was first proposed.

- We present an in-depth analysis of the cache poisoning attack and the ID field vulnerability. We use an basic model of DNS poisoning, but extend it to consider parameters (e.g., server diversity) commonly used in DNS operations. We show that DNS-0x20 encoding increases message integrity far more than authority and recursive diversification.

- To show how DNS-0x20 encoding improves resolver security, we study the number of additional bits available, based on a large-scale DNS traffic trace. For short domains, of course, the benefits are less. Nonetheless, since each additional bit doubles the search space of the attacker, even small improvements obtained through DNS-0x20 results in a query stream that is exponentially harder to successfully attack. While not offering complete security, our system significantly raises the bar.

Section 2 presents a succinct overview of DNS, and essential background on DNS poisoning. Readers already familiar with DNS may skip to Section 3, where we offer a model of DNS poisoning. Our encoding system is presented in Section 4, and is evaluated in Section 5.

2. BACKGROUND

A critically-important component of the Internet infrastructure, the Domain Name System (DNS) [21, 22], maps between names and addresses. DNS is a complex protocol with numerous controlling RFCs. We therefore focus on only those details relevant to DNS forgery attacks. Readers requiring a more general overview may consult [31].

2.1 DNS Overview

In DNS, domain names are composed of labels, separated by periods, which correspond to namespaces in a hierarchical tree structure. Each domain is a node, and the bottom-up concatenation of nodes creates a fully qualified domain name. A zone is collection of such nodes, constituting a separate tree structure, with the zone's *start of authority*, or `SOA`, at the apex. The contents of the `SOA` (either mappings of labels to hosts, or further downward delegation), is available from "DNS authority servers". In DNS nomenclature, these authority servers are sometimes called the SOA.

There are two other DNS resolvers typically involved in poisoning attacks: recursive resolvers, and (less frequently) stub resolvers. A recursive resolver is what one normally thinks of as a "DNS server". Such resolvers accept queries from users, understand the zone hierarchy system, and properly implement the various rules and RFCs to obtain and cache answers.

DNS initiators on host machines are called stub resolvers. They typically don't interact with the zone hierarchy, and with a few exceptions, don't cache answers. Instead, they implement enough DNS logic to pose basic queries to recursive servers.

A short example illustrates how these three classes of DNS systems interact. Assuming no intermediate caching, resolving a domain name like `www.example.com` potentially requires numerous steps:

- First, the stub resolver sends the query to the recursive server. In our example, we assume no previous resolutions whatsoever remain cached.

- Next, the recursive resolver consults with the root servers, which are the authority for the empty label (the dot, `"."`, implicit at the end of all fully qualified domain names). In this example, the root servers would indicate a downward delegation of the "`com.`" zone to other authority servers. (For example, the client might be told to visit the DNS server at `a.gtld-servers.net.`, run by VeriSign, and further be given the IP address of that DNS server as "glue" to avoid additional lookups).

- Next, the recursive server will consult with the "`com.`" zone authority servers, which again will indicate further downward delegation to the `example.com.` zone. (For example, instead of being given an answer, the client might be told next to visit `ns1.example.com.`, or the appropriate authority server for the zone.[1])

- Next, the recursive server consults the `example.com.` zone, which would reveal the host address record (or "`A` record") for `www.example.com.`

- Finally, the answer is returned to the stub resolver, and cached by the recursive resolver to assist in future resolutions.

Each one of these consultations involves the recursive resolver expecting an answer from a remote authority server–either an indication of further delegation or a terminating RRset. A DNS poisoner could anticipate or induce this chain of resolutions and, before the authority responds, inject false answers with spoofed source

[1]At this writing, the `NS` for the `example.com` are the hosts `a` and `b` in the zone `iana-servers.net`; however, we've simplified this sample to presume an authority at `ns1.example.com`.

addresses. This form of DNS poisoning is a packet race. The recursive servers accept whichever answer arrives first–so long as the arriving message matches a few simple transactional requirements.

2.2 DNS Poisoning

To better understand the transactional issues in DNS poisoning, we can reduce the complexity of DNS lookups into a simplified model. Figure 1 shows a basic conceptual model of these three DNS actors critical to a DNS poisoning attack. In Figure 1, the stub resolver first queries a caching server (labeled `A?` in the diagram). Since in our example, the recursive lacks a cache entry for the query, it contacts the authority server (labeled `SOA` in the diagram). The answer (labeled `IN A` in Figure 1) is returned to the recursive server, which caches and sends the answer to the stub. Note that we have omitted any reference to the zone hierarchies.

For purposes of our analysis, DNS has but a single messaging format, whether used to ask or answer a query. The protocol format for DNS messages includes a 16-bit ID field, and a query field holding a wire representation of the domain name. Figure 1 shows how the ID field is used to establish the uniqueness of each message.

A DNS poisoner's task, in the simple case, is to guess the 16-bit query ID field. Figure 1 shows a DNS poisoner offering several (spoofed source) DNS answers to a recursive server (indicated as the "crafted `IN A`" answers in the diagram). If the attacker guesses the ID field, and her packet arrives before the authority server's answer, the recursive server will accept and cache her malicious answer.

Clearly, DNS poisoners are most effective when they can guess the ID field. Early versions of DNS servers deterministically incremented the ID field (until OpenBSD developer Theo de Raadt suggested they be randomized). In [19, 16], Klein demonstrated that if the ID field is not securely randomized, it can be attacked successfully after a few interactions with the server.

Because there are only 65,536 possible ID field values, previous work has noted the use of birthday attacks, and techniques to exploit weak random number generation [15, 16, 17, 18, 19, 28], see also [37].

Accordingly, some DNS implementers have sought additional sources of entropy to protect server messaging. D.J. Bernstein [9] first suggested using the UDP source port fields, to show additional correspondence between queries and answers. In this approach, recursive DNS servers would send a query, using a random 16-bit source port, and (conceptually) listen over some 65K open sockets for the appropriate reply. Not all source ports might be used (for example, one might want to avoid well known ports < 1024 typically used by other protocols) [12], and of course pools of sockets could be used instead. But regardless of the implementation, DNS servers that use both the ID field and source port have $\approx 2^{30}$ to $\approx 2^{32}$ possible combinations that an attacker must guess (depending on how the server handles reserved ports).

Recently, Dan Kaminsky announced a technique to replace `NS` records by performing a series of nonce queries [13]. In this technique, the attacker merely induces a random `A?` query, and spoofs answers with appropriate `IN A` answers *as well as* an `NS` update. If the spoofed attack fails to match the ID field, another random `A?` query is generated, and another round of spoofed answers is sent. Eventually (within ≈ 6 seconds on most networks), a matching ID field is generated by the attacker, and the attacker uses the authority section of the winning packet to evict the previous cached `NS` record. This innovative approach reduces the attack time from weeks to seconds, allowing trivial control of DNS cache lines. An unprecedented multi-vendor response followed [30]. For the most part, DNS vendors defended against the Kaminsky-class attack by implementing port randomization to "grow the key space". DNS vendors also changed their glue-handling policies to better validate or reject the rogue `NS` update.

In the DNS attacker/defender cat-and-mouse game, DNS operators continually look for additional opportunities to improve transaction integrity, and attackers search for weaknesses in implementations, and other methods to predict transaction tokens.

3. BASIC DNS POISONING MODEL

While others have shown that DNS stub resolvers can be subverted [8], our concern is in protecting the recursive resolver in its transactions with the authority servers. To do this, we first need to characterize the risk of poisoning to any server.

Many of the basic mechanical steps in DNS poisoning are well-known to attackers. For example, anticipating when a recursive server will initiate a DNS query is straightforward. Attackers can iteratively observe cache values over time (and initiate attacks when previously valid cached entries time out and are queried again). Similarly, open recursive servers can be forced to do lookups, e.g., [8]. Additionally, secured DNS servers might be obligated to initiate lookups for domains that the attacker sends to the attention of protected networks (e.g., by interacting with mail servers, firewalls, or logging webservers, which in turn resolve domains associated with the sessions).

Without loss of generality, we use the scenario where an attacker identifies and queries open recursive (OR) servers. Without a cached record, recursive servers need to start "upstream" iterative queries in order to locate the authoritative servers. As noted in Section 2, portions of the iterative SOA discovery may be cached (e.g., the authority server for the TLD may be cached). Figure 2 shows all of the various stages of this iterative process, assuming no caching takes place.

The x-axis of Figure 2 indicates time. The period between steps t_5 and t_6 in the diagram constitutes the *vulnerable window* for a DNS poisoning attack. During this $\triangle t$ period, the OR waits for an answer from the SOA. Attackers can send malicious answers to the OR, and repeat the process until they guess the appropriate ID field, or the authority finally responds.

Figure 2 shows (in densely packed arrows) numerous packets sent by the attacker to the recursive server. The diagram shows a progression that finally matches the ID field. Each constitutes a single guess of the required ID field and port values. In our model, we also assume the answer's TTL (or caching period) is such that, if the attack fails, the attacker must wait a lengthy period of time before trying again. (The poisoner is free to try again, of course, but must wait TTL seconds–assumed to be a very long time.)

Definition 1. *We say a DNS server is* forgery resistant *where $TTL \gg \triangle t$, and the chance of an attack being successful within $\triangle t$ time is low.*

We realize that terms such as "low" are unclear. After all, determined attackers may try an attack, regardless of the chance of success. We clarify Definition 1 with the following assumption:

Assumption 1. *If attack is not 10% likely to succeed within T_{max}, we deem the DNS server is forgery resistant.*

We pick 1 day for T_{max}, a time that matches a very commonly used TTL period (86400 seconds). Further, one day is a reasonable

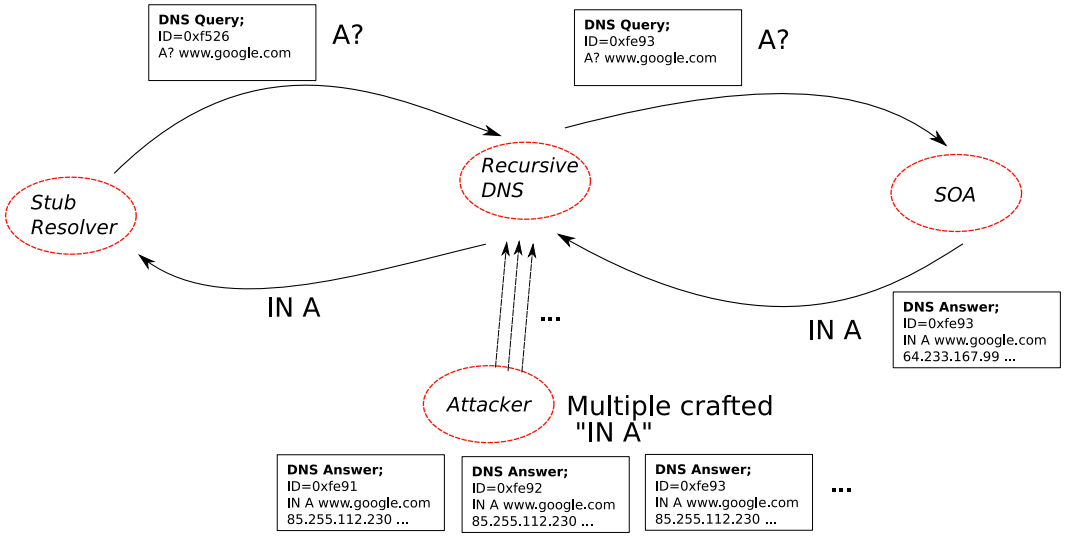


Figure 1: Simplified model of DNS resolution, and poisoning.

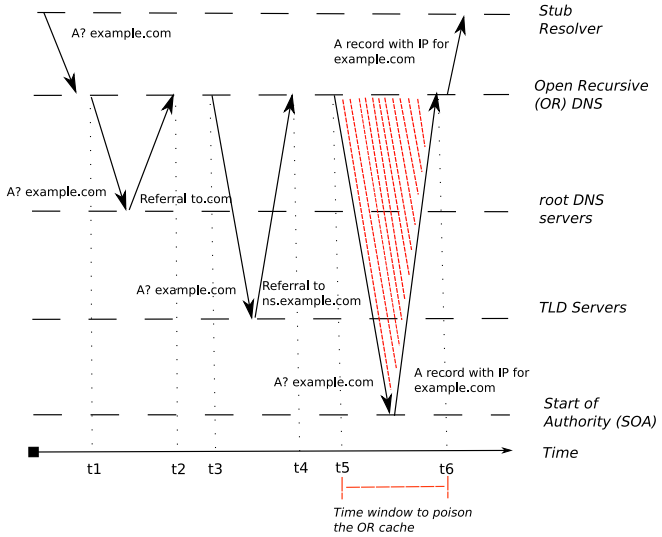


Figure 2: The attacker's time window for a cache poisoning attack on a DNS server during a iterative query.

period of time during which DNS logs could (should) be read by an administrator, or the poisoning attack otherwise noticed by IDS equipment.

We also note that, while 10% is clearly arbitrary, it provides us with a simple means of assessing DNS poison resistance. Absent protocols such as DNSSEC, all DNS servers are vulnerable to some level of poisoning attack. The goal is to make the chance of success as low as possible.

For a particular context, depending on the value of the target, one can adjust this value, and determine the resistance of a DNS deployment to poisoning. Note that the purpose of our work is to demonstrate an improvement in security. Using this assumption lets us show the *relative* improvement in forgery resistance, as discussed in Section 5. Thus, a threshold of 10% is suitable for our purposes.

Clearly, the RTT (or delay between the OR and SOA), plays an important role in the attacker's chances of success. If $\triangle t$ is large, the poisoner can send more spoofed packets, one of which might match the required transactional ID field and port numbers. Even in a Kaminsky-class attack (which is largely bandwidth limited), the RTT determines the number of spoofed packets one can send. As noted in Section 2, many DNS vendors have also changed their glue handling policies so that rogue `NS` updates are inspected and re-validated. This means RTT remains one the most important variables for an attacker.

In practice, the RTT for DNS messaging varies, since recursive and authority DNS servers could be located anywhere. Fortunately, there are known techniques to measure the RTT between any tuple of open recursive and authority servers. In [11], Gummadi, et al., described "King", a measurement technique that uses repeated probes of open recursive servers. In general, King uses two queries to measure the RTT between a recursive and authority server: one for a nonce record that is not in cache, and a second, duplicate query that gets answered from the recursive's newly populated cache. The time difference between the two is the RTT between the recursive and authority servers.

To observe variability in RTT, and suggest reasonable bounds for estimating $\triangle t$ (which in turn determines the number of attack packets that can be sent) we implemented a larger, expanded version of King. We followed several steps.

1. First, we obtained lists of open recursive servers, from both the Measurement Factory's study, [36], and by contacting the authors of [8], who measured tens of millions of such servers. We mapped each open recursive to an Autonomous System (AS), and randomly selected 5,000 resolvers from ripencc, arin, apnic and 500 from afrinic servers. We further verified the hosts were still open recursive.

2. We then created a domain, created an NS for the domain, and made sure its NS propagated to the parent zone.

3. We next "primed" each open recursive to make sure they had cached the root servers, TLDs, and required intermediate zones. (This avoided measuring the time needed by the recursive to locate the authority server.)

4. We then used the following probe technique, for hundreds of random labels within our domain. For each random label, R_i, we asked from several locations:

   - *Iteratively* asked the OR for the label R_i. I.e., we made sure it had not somehow cached the answer already. We recorded the response time as t_A.
   - *Recursively* asked the OR for R_i again. I.e., we forced it to consult the authority server. We knew from previous steps that the parent zone information and NS for our zone were already cached. We measured this time as t_B
   - *Iteratively* asked the OR for R_i again. We noted the time it took as t_C.
   - *Calculate:* $RTT_i = t_C - t_B$. As a sanity check, we also verified that $t_C - t_B \approx t_C - t_A$. I.e., the difference between the recursive and *any* iterative probe should be the same. (The observed variance, due to inherent variability in network delays, is reported in Figure 4, and discussed below.)

5. After noting the RTT_i for each $1 \ldots n$ round of queries, we calculated the average RTT between our SOA and the recursive server.

The distribution of RTT times, from the stub resolver's perspective, appears in Figure 4(a). A CDF plot of these RTT times is shown in Figure 4(b). There are several observations one can make. First, these measurements generally fit the prevailing wisdom of DNS operators that all DNS messages take anywhere 100 to 400 milliseconds to complete, with a long tail taking much longer due to drops, timeouts, and other problems (e.g., server failure).

Second, if the domain is cached, then the average query/answer response time is less than 100 millisecond. On the other hand if the query was not cached it can take close to 400 milliseconds for the recursive server to present an answer back to the resolver.

Our small study helps us understand the dimensions of $\triangle t$. For a given percentage of queries (say, the RTT for 90% of all lookups between an OR and SOA tuple), and can estimate the RTT, and from there determine the number of "guesses" an attacker can make before the correct authority answer arrives.

Definition 2. We can therefore state the chance of successfully poisoning a DNS server, for a single packet:

We assume:

α = Number of Different DNS IDs (universally 2^{16} or 65,535 values)

β = Number of Source Ports (conceptually 2^{16}).

γ = Number of Ports excluded (often 1024, or depending on kernel resources [12].)

θ = Number of authority servers and recursive IPs. Many authority clusters include multiple DNS servers with independent public IP addresses (to provide power and geographic diversity). A recursive server normally RTT sorts the servers, and then queries the closest host. No RFC mandates this, however, and recursive can also randomly select SOA server θ_i. Additionally, some recursive servers are multi-homed, and could select any routable source address for its query. θ is the sum of all public facing addresses used by the recursive and authority servers. In addition to the port and query ID, an attacker has to spoof the correct authority source address, and send this to the correct recursive address.

$$P_{success(1st)} = \frac{1}{\alpha * (\beta - \gamma) * \theta}$$

In the common case of a server employing both ID and source port randomization, with 3 authority servers, this amounts to:

$$P_{success(1st)} = \frac{1}{2^{16} * (2^{16} - 1024) * 3} \approx \frac{1}{12.7B}$$

In sending n packets, an attacker may succeed with:

$$P_{success(n)} = \frac{n}{\alpha * (\beta - \gamma) * \theta}$$

Other parameters of course affect the actual chance of success. Bandwidth and traffic loss are also critical variables we've not included in our model. However, with the pervasive availability of botnets, compromised machines, and proxies, we assume an attacker would not be constrained. A more complex model would introduce this as a separate constraint. Since network measurement studies have generally observed that bandwidth correlates positively to RTT [7], we omitted this parameter in our simplified model.

Figure 3 shows the chance an attack will be successful against a variety of defenses, and illustrates the properties of the simplified model. The logscale x-axis depicts the number of attack packets. Based on the RTT study (and assuming a window of 100ms, with the attacker using a 100Mb/s connection), some 13,000 packets can be sent. The linear y-axis shows a range of θ, the combined IPs of the authority and recursive servers. The logscale z-axis shows the probability of a successful attack. RFC 1912 recommends a small number of authority DNS servers, and no more than ≈ 7 [6]. While not a standard, this advice is general wisdom, and even large enterprise zones (e.g., search engines, Fortune 500 companies) have just three or four public IPs for their authority server farms.

The upper mesh drawn in Figure 3 shows the rate of success against a DNS server using just ID field randomization and a fixed port. Unless significant numbers of additional authority servers are brought online (in excess of those normally used, and more than those recommended by RFC 1912), the chance of an attacker successfully poisoning a DNS server rises with the number of attack packets. In general, Figure 3 shows that IP diversity provides a linear increase in security, while port randomization provides an exponential improvement.

The lower mesh in Figure 3 shows a much better resistance to forgery attempts. One might be tempted to think that port randomization has solved DNS poisoning completely. While clearly useful, [28, 29], port randomization can be overcome by determined attackers able to send large amounts of traffic [20]. We desire additional means of security for several reasons:

- Not every recursive DNS server can implement port random-

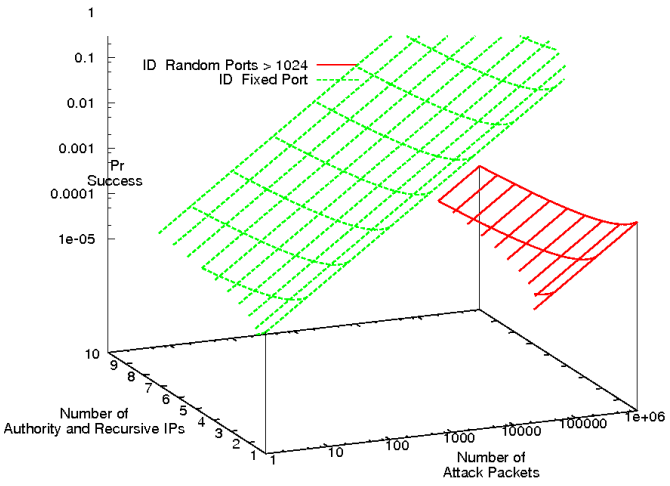


Figure 3: Probability of DNS poisoning attack success, for fixed and randomized ports.

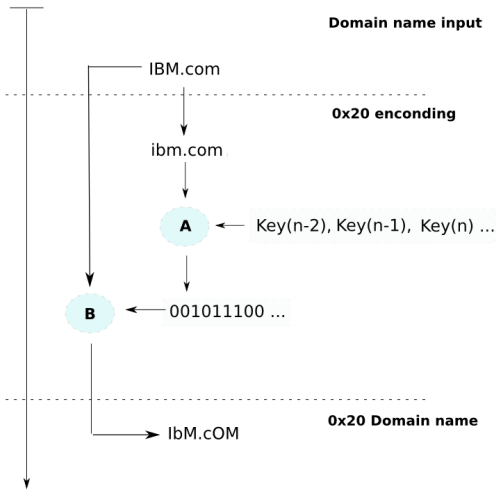


Figure 5: A proposed algorithm for encoding DNS-0x20 bits into queries. While other techniques are possible, this approach is stateless, and allows for simple verification of the answers with constant memory overhead.

ization, since it poses unique engineering challenges. Potentially, a server using source port randomization might have to `select(2)` over thousands of open sockets, opening and closing them as they are used. For embedded systems, implementers may be left with expensive poll techniques. As noted in [8], there are likely millions of recursive servers in embedded systems.

- Some DNS servers are more important targets, (e.g., ISP DNS servers that could potentially yield millions of victims). Even if a DNS server used both the ID field and port randomization, it may still present a tempting target for persistent, ongoing, low-grade attacks.

We therefore need additional DNS protection measures, not merely to increase forgery resistance, but also to provide a diversity of defense options for a variety of platforms.

4. DNS-0X20 BIT ENCODING QUERIES

As noted in Section 2, DNS labels are case insensitive, and in fact no DNS message assigns any meaning to case differences of letters. Further, even if a zone configuration file contains a particular case pattern, e.g., `WWW.EXAMPLE.COM`, queries using any case pattern, e.g., `www.example.com` will be answered. Case formatting may be preserved in cache lines, in service of trademarks; however matching and resolution is entirely case insensitive.

It turns out that, with minor exceptions, all queries are copied from the initiator's packet, exactly into the response. Based on the available open source implementations that exhibit this behavior, it appears this behavior comes as a side-effect of efficient programming. Instead of copying the DNS query in memory, it is rewritten, in place, just as it arrived over the wire. I.e., the authority servers flip source port and IP fields, change flags, checksums, and adjust a few parameters (e.g., authority and answer sections) *in place*. Thus, answer messages contain the query field *in the same case pattern* as originally offered by the DNS initiator.

This provides an opportunity to use the 0x20 bit of any ASCII letter (in the ranges `0x41 ... 0x5A` and `0x61 ... 0x7A`, e.g., `A ... Z` and `a ... z`) in the question name, to encode transactional state information. The mixed pattern of upper and lower case letters constitutes a channel–one that can be used to improve DNS security.

An example shows how this encoding can trivially correspond to a unique query. The following question names will be treated as equal by a responder (for purposes of cache matching), but can be treated as unique by a DNS initiator:

Domain Name	*Field Value*
`www.example.com`	1111111111111
`WWW.EXAMPLE.COM`	0000000000000
`WwW.eXaMpLe.CoM`	0101010101010
`wWw.ExAmPlE.cOm`	1010101010101

In the second column, we can indicate a numerical value that represents the encoding, where lowercase == 1, and uppercase == 0. The DNS initiator can use this encoding as an additional means of verifying message integrity.

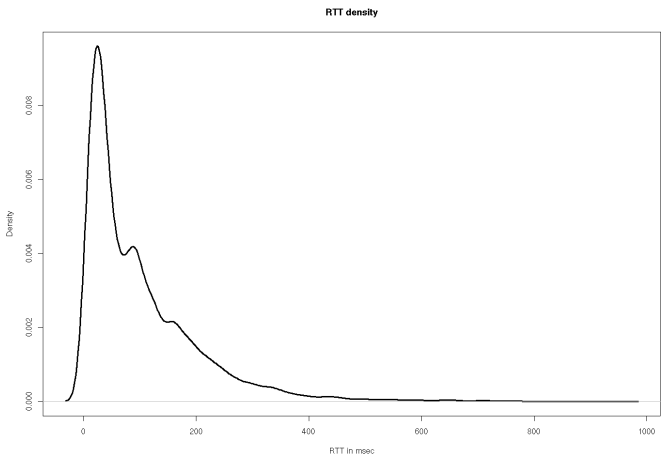
To efficiently encode a query, we propose a simple algorithm. Figure 5 illustrates the following steps:

1. As an input, a domain name input arrives: either an answer from a server, or a query from a stub resolver. Figure 5 shows the arrival of `IBM.com` as a query string.

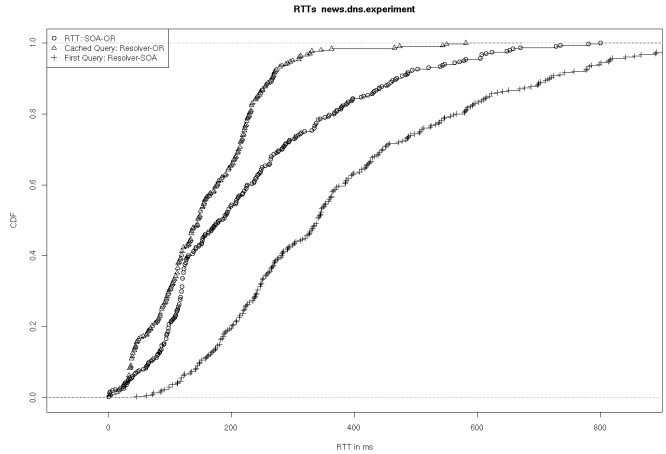
2. First, one transforms the query field into a canonical format, e.g., all lowercase.

3. Second, one uses a chosen encryption scheme to encrypt the canonical query, e.g., perhaps with AES [23], and a key shared by all queries on the recursive server. This is illustrated as step A in Figure 5. This step could equivalently use a small number of keys, one for a given time epoch. (Key management is beyond the scope of this algorithm, but briefly noted below.)

4. Since the resulting cipher block is longer than the original query in terms of bytes, bits are read in sequential fashion from the cipher block. The query field, called `buff` is read



(a) RTT Density



(b) ECDF of RTT

Figure 4: (a) Distribution of RTT times in OR-SOA experiment. (b) Cumulative density of RTT times.

one byte at a time. Step B in Figure 5 shows the encoding of all "0x20 capable" characters (i.e., `A-Za-z`.) In such a case, one reads the next bit j from the ciphered block, and:

(a) if the jth bit is 0, make the i query character upper case (i.e., `buff[i] |= 0x20`).

(b) if the jth bit is 1, make the i query character lower case (i.e., `buff[i] &= 0x20`).

5. This produces a 0x20-encoded domain name, as shown in the final segment of Figure 5. This can be sent to an authority server. Likewise, it can be used to verify the query field returned by an authority server.

The mathematical operations used to change case ($\wedge = 0x20$ and $\vee = 0x20$, above) suggested the name for the "DNS-0x20 encoding" scheme. I.e., upper and lower case characters are 0x20 bits apart in the ASCII table, and the 0x20th bit in a query becomes a channel.

Since the encoding bits are derived from the domain name, the system is stateless. That is, the DNS server does not have to remember that a query has been sent, and how it encoded the 0x20-capable characters. If one were to include such state in a DNS server, it would likely be a DDoS target (at worst), or introduce performance overheads in accessing main memory (at best). Obviously, other implementations are possible, and we suggest this merely as an engineering efficiency, not as a requirement.

A secure encoding scheme, such as AES, can be used to make sure that attackers do not guess the encoding key. We do not consider issues of key management in our proposal. However, we note that, if a weak encoding system is used, attackers may interact with an 0x20-encoding DNS server repeatedly, asking for labels in a zone the attacker controls, in an attempt to mount a plain text attack.

We see this attack as orthogonal. To prevent such attacks on an 0x20-enabled server, the key can be changed out frequently, based on use or time. Thus, figure 5 shows one of several keys being selected to encode a query. Keys can be retired after repeated use to minimize the risk of such attacks. Other implementations are also possible.

5. ANALYSIS

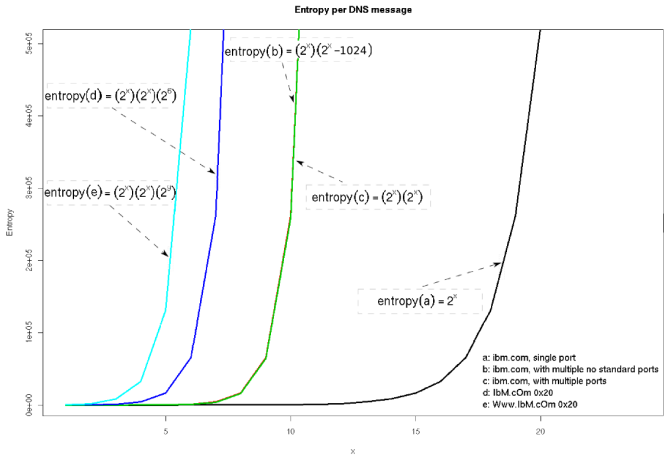
Our proposed criteria in Section 1 requires that DNS-based anti-poisoning measures result in improved security. DNS-0x20 encoding improves the forgery resistance of DNS messages only in proportion to the number of upper or lower case characters in a given query. For example, the domain `cia.gov` has only 2^6 additional combinations for the attacker to guess in a poisoning attack, while `licensing.disney.com` has 2^{18}. In the pathological case, queries for a ccTLD (country code top-level domains, e.g., `".cx"`), would enjoy just two additional bits.

To see if DNS-0x20 improves the average case, we gathered DNS traces (using passive DNS [35]) from a university network for several months, and examined the query fields extracted answer packets. We selected only packets that had `AA-bit` flags enabled, indicating they contained authority responses. In total, the traffic amounted to 5.6 million packets.

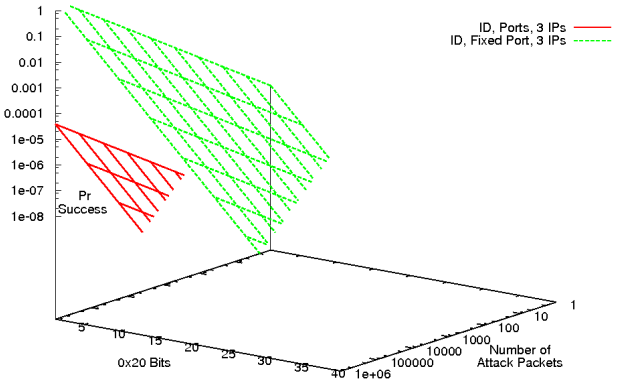
Figure 7(a) shows a correlation between the number of 0x20-capable characters, and the overall length of the query (excluding the `"."` characters between labels). The vast majority of domains were under 50 characters. For this grouping, over 2/3 of the characters were 0x20 capable. Some clusters of longer packets occur at 100, 150 and about 200 character intervals, and have decidedly fewer 0x20-capable characters. An inspection of these packets shows them to be DNSBL and sensor-related traffic. For example, some mail servers encode state information in lengthy alphanumeric labels, which are then checked against centrally run DNSBLs.

Figure 7(b) also illustrates how domain depth relates to the number of available DNS-0x20 characters. In the far corner of Figure 7(b), when one encounters domains with ≈ 34 labels (i.e. separated by nearly many periods), the number of usable DNS-x20 characters is small. Domains with such a depth correspond to reverse IPv6 lookups, where only the `A ...F` hex characters (or dot-separated nibble bits) in IPv6 address can be case flipped.

For the most part, however, Figure 7(b) shows that with increased domain depth, the number of DNS-0x20 capable characters increases slightly. This is confirmed in Figure 7(d) which compares domain depth to all *non*-0x20 characters. Figure 7(c) gives some further insights into the variance of DNS-0x20 characters. This plots the number of digits, in proportion to the length of the domain



(a) Comparison of DNS Transaction Protection Techniques



(b) Improved Resistance

Figure 6: (a) A comparison of various DNS anti-forgery techniques shows the improvements due to DNS-0x20 encoding. (b) Effect of 0x20-Encoding on attack success probabilities, for various character counts. The 0x20 encoding particularly helps DNS servers that cannot implement port randomization schemes, because of platform resource limitations.

name. There is an obvious linear correlation, where some domain names are nearly entirely composed of digits. The diagram thus shows "stair cases" of clusters, with approximately 50, 70, and 90 digits.) This group corresponds to reverse DNS lookups, and other customized DNSBL formats that use numerical encodings. The bulk of the observations made in Figure 7(c), however, appear in the lower corner of the plot, below 50 characters in length. Since, on average, domain names with $\leq$ 50 characters total have only $\leq$ 10 characters devoted to numbers, there are many characters available for DNS-0x20 encoding.

As a whole, Figure 7 shows there is variation in the number of DNS-0x20 characters in DNS lookups. The Figure also illustrates interesting types of lookups (e.g., reverse DNS) that tend to be poor in DNS-0x20 lookups. While such queries could be poisoned, we suspect that attackers are more likely to target "high value" domains, such as banks, social networking sites, and auction sites. These domains are composed almost of entirely of 0x20-capable characters, and would benefit even more from mixed-case encoding. Figure 8(a)-(b) presents a CDF and histogram of the 0x20 characters in all domain queries. It demonstrates that overall, 25% of domain queries provide approximately 20 0x20-capable characters; about 80% had at least 12 available 0x20 characters.

To express the average security improvements of DNS-0x20, we therefore define a convenience function ℓ, which returns the number of 0x20 characters in a domain name. A DNS server that performs both ID field and port-encoding will have, on average, $\bar{\ell}$ additional bits of entropy, or $2^{32+\bar{\ell}}$ possible values. As shown above, for many types of queries, $\bar{\ell} \approx 12$. Note that each additional bit doubles the number of combinations that an attacker must guess correctly. Exponential growth is punishing, particularly for larger exponents. Figure 6(a) shows the search space an attacker must guess against, for a simple encoding of `ibm.com`. The x-axis is the total number of bits available to encode transaction identities. The y-axis indicates the number of possible combinations (or the denominator in any probability model for successful guessing). If the DNS initiator merely used the ID field, and a single (non-variable) source port, the additional benefits of 0x20-encoding are shown in the line labeled "a" in Figure 6(a). Note that by adding port randomization, the DNS server enjoys the growth curve found in lines "b" and "c". This plot also shows that excluding well known ports (e.g., ≤ 1024) is just a linear reduction of an exponential term, does not significantly affect outcomes. (I.e., $2^{16} \gg 1024$).

Using DNS-0x20, we can restate our simple model of DNS poisoning. The chance that the nth packet would successfully poison a DNS server, for the domains, d, usually handled by the DNS server:

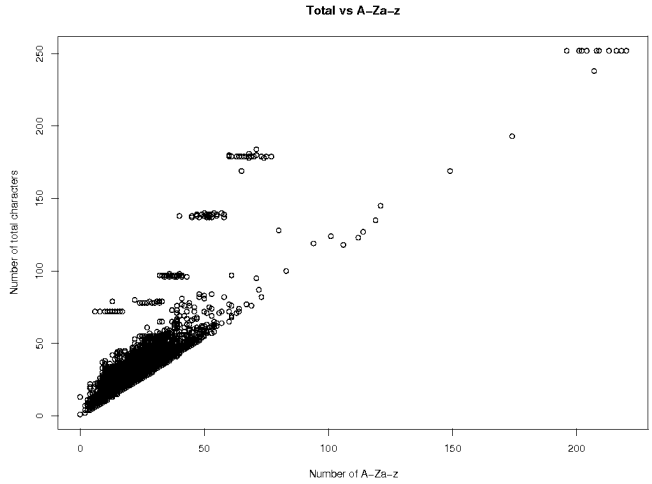
$$P_{CumulativeSuc(n)} = 1 - \prod_{i=0}^{n-1} \left(1 - \frac{1}{2^{\ell(d)} * \alpha * \theta * (\beta - \gamma) - i}\right)$$

Figure 6(b) plots the resulting probability of success for an attacker. Unlike the plot in Figure 3, we fix the number of additional authority servers to 3 (a conservatively high number usually seen in enterprise networks; most networks tend to have just two). The average number of 0x20 characters handled by the server, $\bar{\ell(d)}$, is represented on the y-axis. Figure 6(b) shows how DNS-0x20 has the most improvement for DNS servers using only the randomized ID field and a single port. The chance of success dips with more 0x20 characters in each query. (As noted, the average number of such characters was 12 in our sample study, with a median of 16.) While not as dramatic a reduction as the use of randomized ports (which provide at least 14 bits on average), 0x20 encoding reduces the attacker's chance of success. Recall that above a certain threshold, exponential growth becomes quite punishing. Each bit of DNS-0x20 encoding doubles the work an attacker must perform to achieve similar poisoning results.

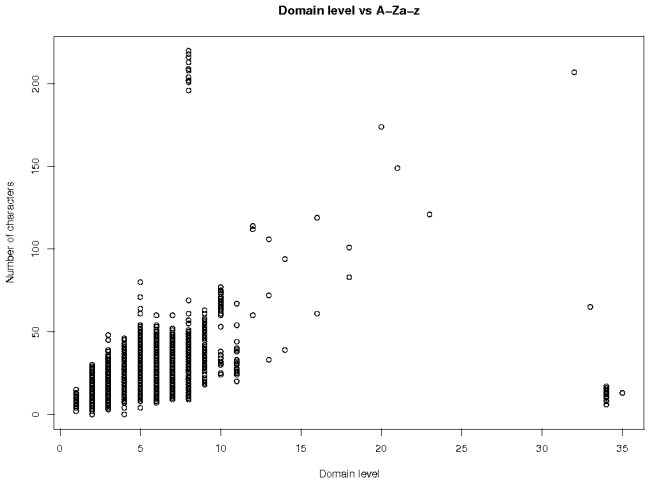
5.1 0x20 probing

Our criteria for a practical DNS-based protection system also requires that it be widely deployable. To evaluate this, we checked which authority servers supported and preserved DNS-0x20 encodings. Conceptually, this can be done by posing a mixed-case query to authority servers regarding labels within their delegation zone. For example, one might ask `ns1.google.com` (one of the listed authorities for the `google.com` zone) the following:

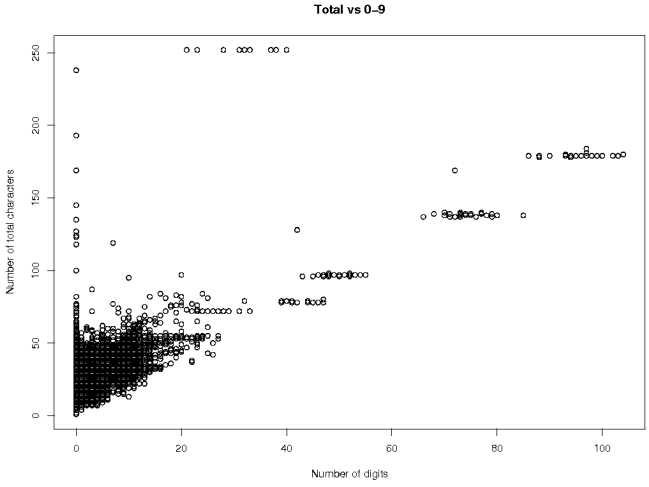
```
dig  @ns1.google.com  wWw.GooGle.COm
```



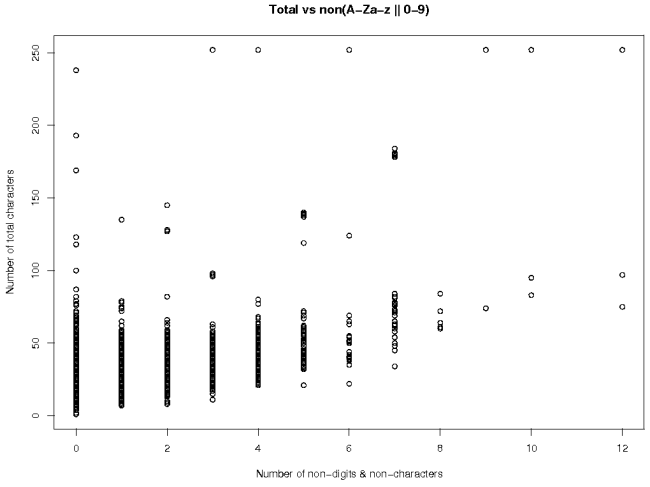
(a) Query Length vs. 0x20 Chars



(b) Domain Depths vs. 0x20 Chars

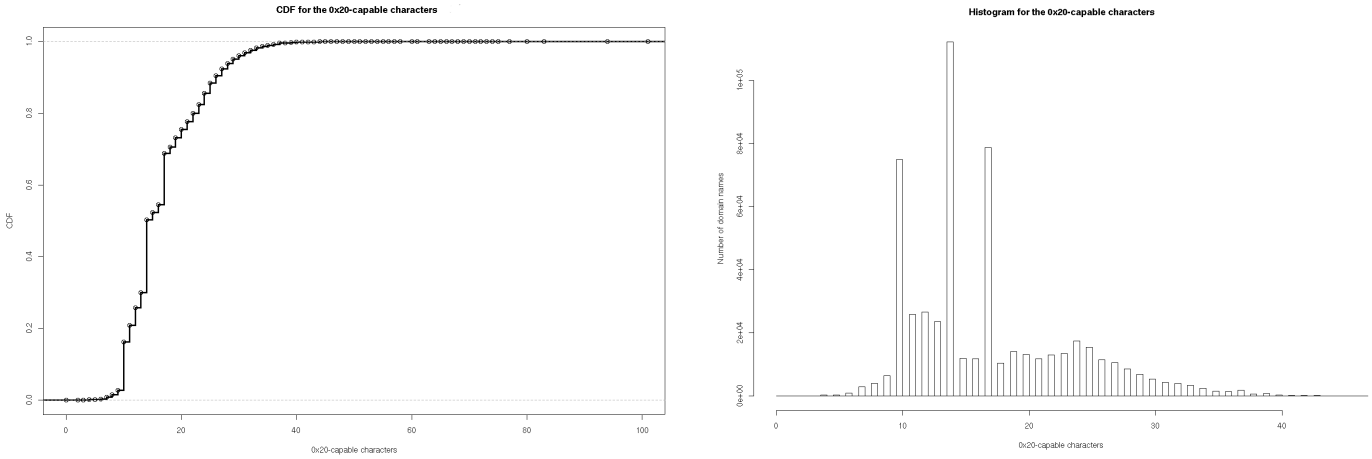


(c) Query Length vs. Digits



(d) Domain Depths vs. Other

Figure 7: (a) Correlation plots of query lengths against the number of 0x20-available characters. (b) Domain depth vs 0x20 characters. Since most high-value user sights (e.g., banks) are only 3LDs, the decline in 0x20-characters in deeper domain depths may not be as significant. (c) Query length and digits. (d) Domain depth vs other other characters.



(a) CDF of 0x20 Characters in Trace

(b) Histogram of 0x20 Character Counts

Figure 8: (a) CDF of number of 0x20 characters in domain names, observed in the passive DNS trace. (b) Histogram of the number of 0x20 characters.

NS Vendor	Pct. Population
JHSOFT simple DNS plus	39%
incognito DNS commander v2.3.1.1 – 4.0.5.1	1.9%
DJ Bernstein TinyDNS 1.05	0.5%
ISC BIND 8.3.0-RC1 - 9.4.0a0	7%
menandmice QuickDNS	1.5%
Sourceforge JDNSS	0.1%
Timeout and no matches	50%

Table 1: DNS Servers Reporting 0x20 mismatches.

The returned answer should repeat the query, bit for bit, including the chosen case variation. One must also check this behavior under relatively high volumes, over time, and from different locations.

Unfortunately, there is no available academic testbed of all known DNS authority servers. So, to evaluate if DNS servers could handle our encoding schema gracefully, we scanned the Internet non-stop for 3 weeks, targeting the authority servers listed in the `.com` and `.net` zone files. These zone files list some 75 million name servers (in aggregate), on average; our probes amounted to some 7 million queries, spread across every DNS server listed in these TLD zones.

The results of our scans are shown in two matrices, in Table 2. There appear to be just a few DNS servers that do not perform proper DNS-0x20 encoding, under certain circumstances. Altogether, they amount to $\approx$ 0.3% of the servers we contacted. We tended to observe a failure to preserve DNS-0x20 encodings under very high query volumes, e.g., dozens of identical queries per second, for the same domain.

Table 1 shows the results of DNS fingerprinting scans of these servers. A few of these authority servers, e.g., BIND, are known (because of source code) to DNS-0x20 compliant. Although DNS fingerprinting is approximate, we surmise that some networks (and not the DNS servers) have server load balancers or hardware accelerators for their DNS farm. We are continuing our efforts to identify and contact the operators of these networks. Notably, google recently changed the behavior of its `l.google.com` host, to be DNS-0x20 compliant. It appears, however, that less than **0.28**% of the servers behave this way.

Type	Mismatch	Mismatch pct.	Domain scanned
`.com` TLD	15451	0.327%	4786993
`.net` TLD	4437	0.204%	2168352

Table 2: Authority servers preserving 0x20 encoding, by TLD

Thus, over 99.7% of all DNS servers we studied could support our DNS-0x20 encoding scheme *without changing their code base*. Those that don't support it appear inconsistent in their "flattening" of queries. We therefore deem that 0x20 is not a radical departure from existing protocols, and very likely to be adopted. We will of course test this view in our IETF standards submission, which seeks to codify what authority servers appear to already do.

6. RELATED WORK

Our proposal fits into the larger debate about how to better secure DNS systems. In [26], the authors consider how transitive trust (via insecure secondaries) provides another potential avenue for attacking DNS servers. Our work, in contrast, proposes a precise model for characterizing the risk to a DNS server, and is restricted to poisoning attacks, rather than attacks on secondaries.

Some proposed standards RFCs have considered improving DNS security. For example, TSIG [33] or SIG(0) [2], and TKEY [3] all seek to improve message integrity. TSIG and SIG(0) use keys between servers to verify messages. These techniques, while effective against forgery attacks, have proved difficult to deploy, because of the need for key pairing between servers, and their strict time synchronization requirements. TKEY solves the key distribution problem, but has considerable computational costs that may be leveraged in a DDoS attack on the DNS server. DNS-0x20, by contrast, is *extremely* light weight, and requires no coordination between pairs of DNS communicators. But unlike TSIG, SIG(0) and TKEY, DNS-0x20 does not provide strong support against DNS forgery. Instead, DNS-0x20 raises the bar.

A recent proposed IETF standard called "Domain Name System (DNS) Cookies" is related to our approach [1]. Like our ap-

proach, DNS Cookies attempt to provide weak, yet practical DNS transactional protection, but creating an `OPT RR` option. The DNS cookie is essentially an HMAC of the requestor's IP, and transaction. While still lightweight compared to other DNS transaction protection systems, e.g., TSIG, DNS Cookies do require substantially more implementation. Specifically, it requires DNS initiators and responders make code changes to handle the DNS cookies. In comparison, DNS-0x20 is even lighter weight, and requires only implementation on a single recursive resolver to work.

A recent IETF draft on DNS forgery resilience discusses many aspects of DNS poisoning [4]. We recommend the IETF draft as an excellent overview of DNS poisoning, and practical countermeasures.

DNS poisoning motivated the work in [37], where the authors proposed DoX, a peer-to-peer DNS replacement. Their approach requires the creation of verification channels, using a P2P system. In contrast, our system uses an existing channel in the working DNS system. Similarly DoX requires a peer system to improve DNS security. Our approach can be implemented by a single recursive server today, and immediately improves the integrity of messages to authority servers.

We believe that the work most related to ours is found outside of the DNS field. TCP SYN Cookies were first proposed by DJ Bernstein and Eric Schenk in 1996, as a means to stop resource exhaustion DDoS attacks on TCP stacks [10]. The idea behind SYN Cookies is superficially similar to our DNS encoding scheme. Both save server state to efficiently associate two packet events in time. Both add this state by overloading the meaning of a protocol field. In the case of SYN Cookies, a selected TCP sequence number has two meanings: that from the protocol, and also an HMAC. Randomized DNS ports, also proposed by DJ Bernstein, uses a similar field-overloading logic. We believe DNS-0x20 is in that same spirit: field overloading yields additional state, and can be done by only one party in a transaction to improve security.

7. CONCLUSION

DNS poisoning attacks present a persistent, ongoing threat to the Internet's critical infrastructure. There have been many proposed solutions, both from the operator and academic communities. The lack of adoption and delays in deployment suggest the need for very-light weight, practical improvements to DNS security. We therefore considered solutions that provide incomplete security, but nonetheless offer measured improvements.

To be successful, we argued that such a protocol must: (a) require no radical changes to the DNS infrastructure; (b) make no major changes to the existing protocol; and (c) be backwards compatible, so that even just a few DNS servers can elect to adopt it. We believe these elements will speed the adoption of the security measure.

DNS-0x20 encoding meets these requirements, but necessarily at the cost of complete protection. It does not require a radical restructure of the DNS infrastructure, and can be adopted unilaterally by recursive servers. With small exceptions ($\approx 0.3\%$) the world's authority servers appear to already preserve the encoding scheme. Indeed, DNS vendors are now incorporating the system into their code bases.

But unlike complete, heavy-weight solutions to DNS poisoning, DNS-0x20 encoding does not provide strong guarantees for transaction integrity. Using large trace files, we found that on average, DNS messages can have an additional 12-bits of state. The slow adoption of other, more complete DNS transaction protection systems suggests the immediate need for this light-weight solution.

7.1 Future Works

We endeavored to create *practical* DNS-based security enhancements that can be rapidly adopted. No doubt, there will be many issues that arise in DNS-0x20 implementation that we have not considered. For example, as alluded to in Section 1, there may be key management issues to consider.

Our future work will address other efficient, stateless encoding schemes for domain names, using the 0x20 bitset of queries. We will also consider modifications and implementation strategies for resource-limited systems, such as embedded devices and home DSL systems. Although our system does not penalize recursive DNS servers that refuse to implement DNS-0x20, our future work will also consider techniques to update deployed embedded DNS systems. We will also consider policy options for DNS-0x20 recursive servers, so they can identify and work around the few ($\approx 0.3\%$) DNS servers that may not support DNS-0x20 encoding.

We also note that DNS-0x20 does not create, but rather exploits for beneficial purposes, a covert channel within DNS. Future work will measure the capacity of such a channel, and note how DNS-0x20 encoding indirectly contributes to a reduction in the capacity of a malicious (if somewhat obvious) covert channel.

Acknowledgements

This material is based upon work supported in part by the National Science Foundation under Grant No. 0627477 and the Department of Homeland Security under Contract No. FA8750-08-2-0141. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation and the Department of Homeland Security.

8. REFERENCES

[1] Donald E. Eastlake 3d. Domain name system (dns) cookies. `http://tools.ietf.org/html/draft-eastlake-dnsext-cookies-03`, 2008.

[2] D. Eastlake 3rd. Dns request and transaction signatures (SIG(0)s). `http://tools.ietf.org/html/rfc2931`, September 2000.

[3] D. Eastlake 3rd. Secret key establishment for DNS (TKEY RR). `http://tools.ietf.org/html/rfc2930`, September 2000.

[4] A. Hubert and R. van Mook. Measures for making dns more resilient against forged answers. `http://tools.ietf.org/html/draft-ietf-dnsext-forgery-resilience-06`, July 2008.

[5] M. Andrews. The dnssec lookaside validation (dlv) dns resource record, rfc 4431. `http://tools.ietf.org/html/rfc4431`, 2006.

[6] D. Barr. Common dns operational and configuration errors. `http://tools.ietf.org/html/rfc2845`, 1996.

[7] Saad Biaz and Nitin H. Vaidya. Is the round-trip time correlated with the number of packets in flight? In *Proceedings of the ACM SIGCOMM Internet Measurement Conference (IMC'03)*, 2003.

[8] David Dagon, Niels Provos, Christopher P. Lee, and Wenke Lee. Corrupted dns resolution paths: The rise of a malicious resolution authority. In *Proceedings of Network and Distributed Security Symposium (NDSS '08)*, 2008.

[9] DJ Bernstein. The dns_random library interface. http://cr.yp.to/djbdns/dns_random.html, 2008.

[10] DJ Bernstein. SYN cookies. http://cr.yp.to/syncookies.html, 2008.

[11] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: estimating latency between arbitrary internet end hosts. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 5–18, 2002.

[12] Internet Assigned Numbers Authority. Port numbers. http://www.iana.org/assignments/port-numbers, 2008.

[13] Dan Kaminsky. Its the end of the cache as we know it. http://www.doxpara.com/DMK_BO2K8.ppt, 2008.

[14] JungMin Kang and DoHoon Lee. Advanced white list approach for preventing access to phishing sites. In *International Conference on Convergence Information Technology*, 2007.

[15] Amit Klein. BIND 8 DNS cache poisoning. http://www.trusteer.com/docs/bind8dns.html, 2007.

[16] Amit Klein. BIND 9 DNS cache poisoning. http://www.trusteer.com/docs/bind9dns.html, 2007.

[17] Amit Klein. OpenBSD DNS cache poisoning and multiple OS predictable IP ID vulnerability. http://www.trusteer.com/docs/dnsopenbsd.html, 2007.

[18] Amit Klein. Windows DNS cache poisoning. http://www.trusteer.com/docs/microsoftdns.html, 2007.

[19] Amit Klein. PowerDNS recursor DNS cache poisoning. http://www.trusteer.com/docs/powerdnsrecursor.html, 2008.

[20] John Markoff. Leaks in patch for web security hole. http://www.nytimes.com/2008/08/09/technology/09flaw.html, August 2008.

[21] P. Mockapetris. Domain names - concepts and facilities. http://www.faqs.org/rfcs/rfc1034, November 1987.

[22] P. Mockapetris. Domain names - implementation and specification. http://www.faqs.org/rfcs/rfc1035, November 1987.

[23] NIST. Announcing the advanced encryption standard (aes). ttp://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.

[24] KyoungSoo Park, Vivek S. Pai, Larry Peterson, and Zhe Wang. Codns: Improving dns performance and reliability via cooperative lookups. In *In Proceedings of the Sixth Symposium on Operating Systems Design and Implementation(OSDI '04)*, 2004.

[25] V. Ramasubramanian and E.G. Sirer. The design and implementation of a next generation name service for the internet. *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 331–342, 2004.

[26] Venugopalan Ramasubramanian and Emin Gun Sirer. Perils of transitiive trust in the domain system. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference (IMC'05)*, 2005.

[27] Sid Stamm, Zulfikar Ramzan, and Markus Jakobsson. Drive-by pharming. http://www.cs.indiana.edu/~sstamm/papers/drive-by-pharming-router-dns-stamm-ramzan-jakobsson.pdf, 2006.

[28] Joe Stewart. DNS cache poisoning – the next generation. http://www.secureworks.com/research/articles/dns-cache-poisoning/, 2003.

[29] US Cert. Vulnerability note vu#457875. http://www.kb.cert.org/vuls/id/457875, 2002.

[30] US-CERT. Multiple dns implementations vulnerable to cache poisoning. http://www.kb.cert.org/vuls/id/800113, 2008.

[31] Paul Vixie. DNS complexity. *ACM Queue*, 5(3), April 2007.

[32] Paul Vixie and David Dagon. Use of bit 0x20 in DNS labels to improve transaction identity. http://tools.ietf.org/html/draft-vixie-dnsext-dns0x20-00, 2008.

[33] Paul Vixie, O. Gudmundsson, D. Eastlake 3rd, and B. Wellington. Secret key transaction authentication for DNS (TSIG). http://tools.ietf.org/html/rfc2845, May 2000.

[34] S. Weiler. Dnssec lookaside validation (dlv), rfc 5074. http://tools.ietf.org/html/rfc5074, November 2007.

[35] Florian Weimer. Passive dns replication. http://www.enyo.de/fw/software/dnslogger/first2005-paper.pdf, April 2005.

[36] Duane Wessels. The measurement factory open recursive dns reports. http://dns.measurement-factory.com/surveys/openresolvers/ASN-reports/, 2007.

[37] Lihua Yuan, Krishna Kant, Prasant Mohapatra, and Chen-Nee Chuah. DoX: A peer-to-peer antidote for DNS cache poisoning attacks. In *Proceedings of the IEEE International Conference on Communications (ICC'06)*, volume 5, pages 8164–9547, June 2006.