DoX: A Peer-to-Peer Antidote for DNS Cache Poisoning Attacks

Lihua Yuan
ECE, UC Davis
lyuan@ece.ucdavis.edu

Krishna Kant
Intel Corporation, OR
krishna.kant@intel.com

Prasant Mohapatra
CS, UC Davis
prasant@cs.ucdavis.edu

Chen-Nee Chuah
ECE, UC Davis
chuah@cs.ucdavis.edu

***Abstract*— The mapping service provided by the Domain Name System (DNS) is fundamental not only to the health of the Internet but also to the protection and integrity of the data. Recently, the DNS infrastructure has suffered several malicious attacks including *DNS cache poisoning*, which causes the DNS to return false name-to-IP mappings and can be used as a foothold for more insidious attacks.**

In this paper, we propose a peer-to-peer based scheme to quickly detect and correct *inaccurate* DNS records caused by cache poisoning attacks. This scheme also helps DNS servers to improve cache consistency by detecting and removing stale cached records. This scheme does not require modifications to the current infrastructure and can be deployed quickly. In addition, it does not use cryptographic techniques and thus does not suffer from the key management and processing overhead issues of those techniques.

I. Introduction

The Domain Name Service (DNS) provides name resolution (or mapping) between human-friendly machine and domain names and machine-friendly IP addresses. DNS is one of the most valuable parts in the Internet infrastructure. Almost all applications, including http, email and ftp, need to resolve a given domain name to its corresponding IP address prior to establishing connections. DNS is a distributed database that uses intensive replication and caching to achieve high scalability and resiliency to server failures. However, it was not designed to counter malicious attacks like cache poisoning. Being probably the most valuable infrastructure in the Internet, it warrants thorough investigation to ensure its security.

DNS cache poisoning[1] refers to cases where the cache of a DNS server is injected with false information that affects the accuracy of DNS lookups. Consequently, when queries arrive at the server, they get inaccurate, and probably malicious, replies. There are numerous ways to inject false information into a DNS server by exploiting protocol or software vulnerabilities. After one poisoned record is injected into the cache, it can spread to other parts of the cache or other servers through query/response between DNS servers.

DNS cache poisoning is often used as a foothold for escalating to more harmful sub-attacks. The attacker can redirect the querier to an IP address which (1) is nonexistent, thus causing Denial-of-Service (DOS) to the querier, (2) is a malicious site that drops Malware/Spyware [1, 2], or (3) is a masquerade server for man-in-the-middle (MITM) attacks [3], which includes large scale *phishing* attacks known as *pharming* [4] and hijacking of emails or SSL sessions. All these attacks were observed in the March 2005 incident [2] reported by the SANS Internet Storm Center.

Although cache poisoning attacks are extremely dangerous — some refer to them as "the Achilles' heel of the Internet" [5] — there are few defense mechanisms to ensure the accuracy of DNS lookups. In addition to piecemeal patches of specific vulnerabilities, past work has focused on applying cryptography techniques to authenticate DNS records. Secret key transaction authentication for DNS (TSIG) [6] uses symmetric key cryptography to authenticate client-server communications. However, the key distribution is not automatic and it is impractical to establish trust relationships with every other DNS server. The proposed DNS security extensions (DNSSEC) uses a public-key cryptography to authenticate DNS zone data. It relies on the DNS itself to serve as a public-key infrastructure (PKI) to distribute the public key.

There are a few potential problems with DNSSEC that warrant investigation for alternative solutions. First, DNSSEC requires a major overhaul to the current DNS system and global cooperation from all parties involved. Although it has been in development for more than 10 years, its wide deployment still remains to be seen. Second, the success of DNSSEC requires modifications to the DNS servers at every level, the availability of security-aware resolver for every operating system, and complex key management. Considering the fact that many DNS servers are currently running with *known* vulnerabilities, one probably cannot expect the complete deployment of DNSSEC in a short time frame. Third, DNSSEC incurs a significant performance penalty because PKI operations are known to be very computationally intensive [7] and the signed DNS packets could be substantially larger than original.

A similar poison to cache poisoning is a stale cache, which is caused by record updates at authoritative servers. DNS caches maintain weak consistency using a time-to-live (TTL) timer before data is re-fetched from the origin server. A stale record in the DNS cache provides an incorrect name to IP mapping and subsequently causes application failures. DNSSEC does not solve the problem of stale cache since it only guarantees the authenticity of the data but not its freshness. If attackers use authentic but obsolete records as the poison, they can cause Denial-of-Service (DOS)-like failures.

[1] Also referred to as domain name spoofing, hijacking, or cache pollution.

Both a poisoned cache and stale cache cause *inaccurate* DNS records, which lead to further attacks or failures, to be returned to the clients. This paper proposes a peer-to-peer (P2P) solution to the problem based on the observation that DNS records are expected to be consistent with what is dictated by the authoritative server. If all peers resolve a given name correctly, their results should also be consistent, except for a few cases which we will discuss in Section IV. In contrast, cache poisoning is a local attack which affects individual servers. It is difficult for attackers to compromise several DNS servers simultaneously with the same false data for the following reasons: First, the vulnerabilities are specific to the architecture and software of the DNS server. Second, compromising multiple servers is intrinsically more difficult than compromising a single one, and especially so if this must be accomplished within a short time frame. For man-in-the-middle (MITM) attacks, attackers can bet on the chance to win the race, but the probability of winning at multiple locations during the same short time frame would be very low.

Based on these observations, we propose a P2P domain name cross-referencing (DoX) technique, which groups peers together to compare their DNS results and detect inconsistencies. Upon the detection of any inconsistency — be it caused by a poisoned or stale records in the cache — DoX peers cooperate to resolve the inconsistency and converge to the current, correct record. DoX is a client-side solution that does not require any modification to the current DNS system.

The major contributions of this paper are

1) We perform an in-depth study on DNS cache poisoning attacks, the evolution, propagation and consequences of a poisoned cache.
2) We propose DoX, a novel P2P solution for detecting and removing cache poisoning.
3) We briefly discuss a hierarchical model that we have developed to characterize poisoning and update propagation in the DNS.

The rest of this paper is organized as follows. Section II presents some related background about the DNS architecture and Section III details various cache poisoning attacks. Section IV presents the DoX algorithm in detail and we evaluate its performance in Section V. We conclude in Section VII

II. Background

DNS is commonly regarded as a distributed database in the form of an inverted hierarchical tree (the right side of Figure 1). Each DNS server owns a domain (subtree) for which it is the authority. For scalability, it can delegate a sub-domain to another server and only maintain a referral to that server.

When an application needs to resolve a domain name, it uses standard APIs, e.g. *gethostbyname()*, to access the resolver offered by the operating system. The resolver, in turn, queries its DNS servers. Administrators often configure the end-hosts to point to a few cache-only name servers, which then forwards the queries to the central recursion name server. Since the address of the root DNS servers are well-known, any DNS

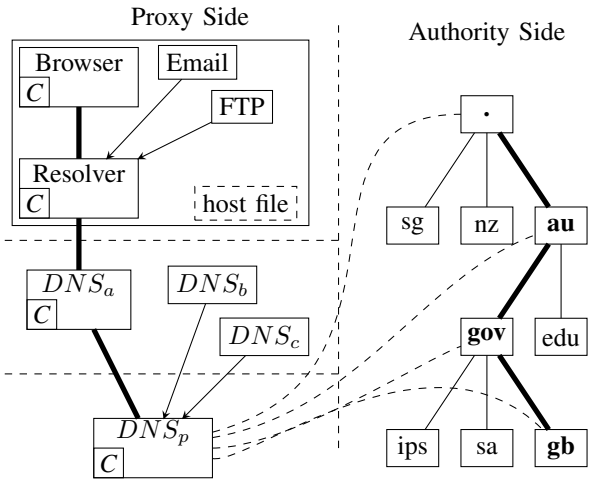


Fig. 1. DNS Architecture

servers capable of recursive lookups can follow the referrals to the authoritative DNS server and get the desired response.

Caching is used at almost every stage of lookup process and is instrumental to the scalability of DNS. If a query can be answered based on the cached contents, the query will not be forwarded and a response is returned. Otherwise, a *best-matching* algorithm is used. A DNS server caches not only the answer to the original query but also everything else it learned so that it does not need to start from the root server every time. The resolver, and even some applications, e.g. web browsers, maintain its own cache. The extensive use of caching significantly reduces the lookup time and improves scalability. However, if any cache on the entire path is poisoned, the end user will get a malicious reply.

III. Cache Poisoning Attacks

Although software vendors release patches soon after vulnerabilities are discovered, new ones emerge over time and a large percentage of DNS servers are running with well-known vulnerabilities [8]. A recent survey by Ramasubramanian and Sier [9] shows that the correct lookup of a domain name depends on a surprisingly numerous 46 servers on average. In this section, we look at how a cache poisoning attack evolves.

A. Poisoning Individual Records

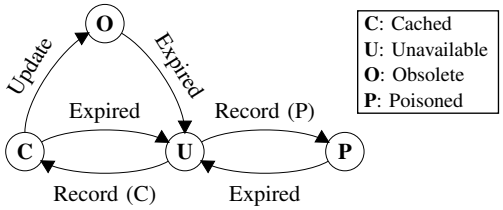


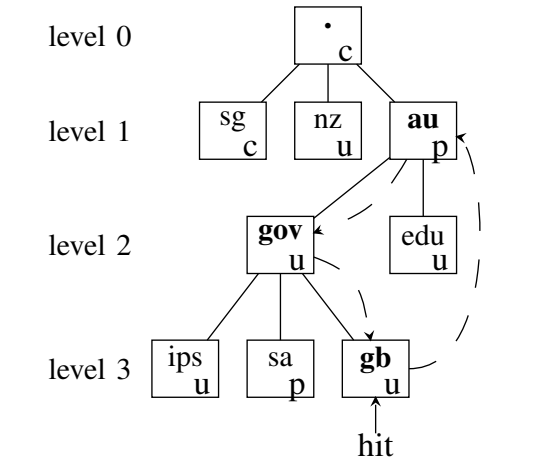
Fig. 2. Individual Record

There are a few vulnerabilities in the design of DNS that allow the injection of poisoned records without compromising the entire system. One could attach malicious records as additional information to a legitimate reply and some server

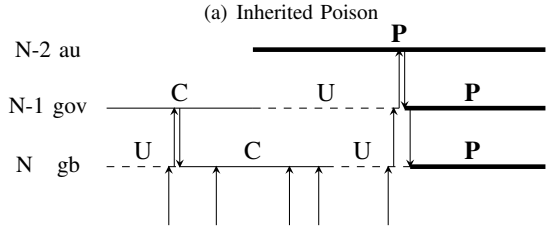
implementations will cache the additional records [10, 11]. DNS is UDP-based and the transaction ID (and sometimes source port number) is the sole form of authentication for a DNS reply. Unfortunately, guessing the right transaction ID is possible for many implementations [12]. The probability of guessing the right transaction ID could be much higher due to weaknesses in the random number generators, use of multiple queries and the consequent birthday paradox [12].

As illustrated in Figure 2, a DNS record can move from *unavailable (U)* to *cached (C)* (*poisoned (P)*) if a correct (poisoned) response is received. A cached record may become *obsolete (O)* if the authoritative server modifies its local version. DNS maintains its cache with a TTL-based consistency mechanism, in which a record is cached for its TTL duration. After the TTL expires, the record is discarded and becomes *unavailable (U)*.

B. Poison Inheritance



(a) Inherited Poison



(b) Hit-Through Model

Fig. 3. DNS Cache Evolution

Once some poison is injected, it can spread within the cache or to its clients through query-response. In Figure 3, we model the DNS name space as an inverted tree and queries as "hits" at the leaf nodes. Upon receiving a query (hit), the DNS server uses a best-matching algorithm to search the cache for the *closest match* and starts a recursive lookup from there. A poisoned closest match can misdirect the query to a malicious name server that returns poisoned response, causing poison to propagate in the cache. In Figure 3, a query for "gb.gov.au" is best-matched to ".au", which is poisoned. The subsequent recursive queries cause "gov.au" and "gb.gov.au" to *inherit* the poisoned (P) status of ".au". The same scenario was observed on March 2005 [2] when the ".com" entry was poisoned and subsequently many sub-domains under ".com".

C. Poisoning the Entire Cache

The DNS server, resolver and client programs are also subject to common issues like buffer overrun and vulnerabilities of the operating system. These include buffer overrun vulnerabilities of early versions 4 and 8 of BIND (Berkely Internet Name Daemon) and malwares that modify the host file of a browser or a end-host. In such cases, one must assume the entire cache is poisoned. Additionally, these poisons are persistent since they cannot be flushed out by the TTL mechanism.

IV. The Antidote: Domain Name Cross Referencing (DoX)

If a DNS lookup is affected by a poisoned cache, the response must be different from what dictated by the authoritative server. Naturally, one could verify a record with its authoritative server, assuming the record about the authoritative server is correct. However, verifying every record with the authoritative server is not a viable approach since it renders caching ineffective and hence the DNS unscalable. In this section, we propose a domain name cross-referencing (DoX) system in which peers collaboratively monitor their DNS lookup results. We design a DoX peer to intercept communication between a DNS query and its response. DoX peers perform consistency checks with each other and consult the authoritative server only if suspicious records are noticed.

A. DNS Inconsistency

Although DNS records should all be consistent with entries dictated by the authoritative servers, there are reasons other than cache poisoning that lead to inconsistencies among clients.

1) Stale Cache: DNS cache uses a TTL-based weak consistency mechanism that could store and return stale records. This can be resolved if peers ignore the cache and *reload* the record from authoritative servers. As a side effect, DoX detects inconsistencies caused by modifications and resolves them by synchronizing with the authoritative servers.

2) Load Distribution: DNS servers commonly use *round robin* load distribution which rotates the order of the records mapped to the same domain name. Consequently, for a domain name that has three addresses $A1, A2, A3$ mapped to it, one peer could receive $A2, A3, A1$ while the other could receive $A3, A1, A2$. DoX uses *set* comparison for consistency check to handle the permuted records.

3) Multiple Views: Some modern DNS servers, e.g BIND 9.2 or newer, can provide *multiple views* (also known as "split horizon") of the zone data. DNS servers using multiple views can return different records based on the IP address of clients. This is often used to separate internal clients from external ones, offering the latter a limited, probably modified, view into the internal networks. Consequently, two peers receiving different *views* will not be able to resolve the inconsistency by querying the authoritative server. DoX avoids raising false

alarms in this case by verifying a *record update* instead of a record itself. If the view assignment is static, the older version of the record can serve as an indicator to whether the two peers are under the same view.

4) Dynamic Mapping: Some content distribution networks (CDNs) use DNS-based redirection for performance improvement and load balancing. Clients could be assigned to servers that are closer or less-loaded based on the current status. DoX introduces the concept of verification channels so that peers can avoid checking names served by CDNs.

B. DoX Network

1) Verification Channel: The entire DNS name space is enormously large. Each DNS server is *interested* in resolving only a very small subset of the name space. If two peers both want to ensure the accuracy of a particular record, the consistency check between them is mutually beneficial. Otherwise, the consistency check is helpful to only one peer and presents only overhead to the other. In addition, if two peers share common queries, it will be more likely they can verify for each other based on local cache. To reduce the overhead and latency of verifications, we propose to construct a *verification channel* to a group of peers that share common interests . A verification channel could be defined in one of the following ways.

- Topic channel: Topic channels use a similar concept as mailing lists or IRC channels by defining *suitable topics* in a certain channel. A topic refers to a set of the DNS names, the correctness of which peers wish to collectively guard. It can be all sub-domains under a certain domain, e.g. *intel.com* or all domain names listed under a yahoo directory, e.g. the "Government/Military/" directory.
- Community channel: A community channel exploits the likelihood that peers from the same community will have more common interests, or at a bare minimum, are willing to bear the overhead for each other. A typical community could be "everybody in the same company".

A verification channel is described by a *tracker file* and is coordinated by a *tracker*, which is a daemon program that tracks participating peers and announces them to each other. A peer joins a channel by opening the tracker file and connecting to the tracker for bootstrapping. Every peer in a channel are assigned k peers randomly so that the peers form a k-connected network. We leave it to the tracker administrator to setup the actual definition of a channel and peers choose the suitable channel based on their own interest.

2) Verification Cache (vCache): If a record has been verified earlier and remains unchanged, it is unnecessary to be verified again, even though it might have expired from the local cache and therefore was obtained through DNS lookup again. For this reason, we augment every peer with a separate verification cache (vCache) to store previously verified results. A vCache is similar to a normal DNS cache but stores records for an unlimited amount of time. It flushes the old records only if the memory limit is reached.

A natural consequence of using vCache is that a peer will only verify a record with the DoX network if (1) the record does not exist in the vCache or (2) the peer observes a record update in which the version stored in vCache (R_v) is different from the version obtained through standard DNS lookup (R_d). We denote the older version of the record as R_o, the newer version as R_n and the transition as $R_o \rightarrow R_n$. The first case is considered as a special case where $R_o = None$. It can be avoided if we allow a safe startup phase for peers to build up their vCache first.

C. DoX Consistency Check Algorithm

Algorithm 1 $DoXCheck(Q)$

```
R^d ← DNS_Lookup (Q)
R^v ← vCache_Lookup (Q)
if R_d = R_v then
  return OK
else
  R_o ← R^v, R_n ← R^d
  Send R_o → R_n to k peers and get first m results
  if #Disagree = 0 then
    return OK
  else
    R^a ← Authoritative_Server_Lookup (Q)
    if R^a ≠ R_n then
      Poison Detected
    else if R^a = R_n and #Agree > threshold then
      return OK
    else
      WARNING
    end if
  end if
end if
```

Algorithm 1 describes the DoX checking algorithm. The version stored in vCache (R^v) was verified previously and the current version obtained through DNS lookup (R^d) might be a newer version. If $R^v \neq R^d$, the peer construct a verification request in the form of $< Q, R_o \rightarrow R_n >$. The purpose of including R_o in the verification request is to help determine if the verifying and requesting peers are receiving different views, which we elaborate shortly. A verification request is sent to n remote peers simultaneously but the requester only wait for the first m ($m < n$) responses to guard against peer failures and to achieve lower latency. If the local record is poisoned, there will be "Disagrees" as long as at least one peer is not. (The remote peers decide if they agree based on the algorithm we will describe shortly.) In this case, the local peer will use an "iteration-only" lookup to obtain an authoritative copy (R^a) directly from the authoritative server. In the case of local poisoning, R^a will not be consistent with R_n. If $R^a = R_n$ and a sufficient number of peers "Agree" with the verification request, we consider the verification a success since there might be malicious peers. Otherwise, we raise a warning.

A peer will perform consistency check upon receiving a verification request in the form of $< Q, R_o \rightarrow R_n >$. It replies to the requesting peer with a verification response in the form of <Decision, Info> where the decision can be one of "Agree", "Disagree" or "DiffView" and $Info$ contains additional information about this check.

The verifying peer first checks R_n with its local vCache version (R^v). If they are consistent, the verifying peer can "Agree" without further checks since it must have verified this earlier. Otherwise, the verifying peer request an authoritative copy (R^a) from the authoritative server and perform the following checks.

- $R^v = R_o$: The two peers had consistent history record.
  - $R^a = R^n$ (Agree): The verifying peer observes the same update from the authoritative server. The reason it did not observe this update earlier is likely a stale record in its local DNS cache. Therefore, this verification request actually helps the verifying peer to update the stale record faster than its TTL expiration.
    A verifying peer *forwards* a record update to other peers if it agrees with the requesting peer. Consequently, a record update is made known to every peer in a DoX network after the first peer notices the update. This can significantly reduce obsolete records and failures caused by them. To avoid global synchronization, a peer set the TTL to be a random value between 1 and the original TTL.
  - $R^a \neq R^n$ (Disagree): The verifying peer does not observe the same update. In particular, if $R^a = R_o$, the verifying peer does not observe a change at all. The requesting peer is most likely *poisoned*.
- $R^v \neq R_o$: The two peers did not have consistent history record. If $R^a = R^n$ (Agree), the authoritative server is probably merging the two peers into the same view. Even if $R^a \neq R^n$ (DiffView), it is possible that they are assigned different *views* by the authoritative server and the verifying peer.
- $R_o = None$ or $R^v = None$: Either the requesting or the verifying peer does not have this record in its vCache. The verifying peer agrees if $R^a = R_n$ and disagrees otherwise. This case can be avoided using a safe startup phase to initialize vCache.

V. Simulation Evaluation

A. Simulation Setup

To study the performance of DoX, we implemented an event-based simulator for the DNS infrastructure and the proposed DoX scheme. The name space is based on the 2.7 million names listed on *dmoz.org* [13]. We queried the authoritative server of individual names to find TTL values and found that several popular ones, e.g. 1 hour, 2 hours, 1 day, and 2 days dominate. Queries arrive with a Poisson arrival process of rate λ_q to leaf nodes and the popularity of names follows a Zipf-distribution. Every node in the namespace, except at the root node, has equal opportunity to be modified by their respective administrators. Collectively, modifications follow a Poisson arrival of rate λ_m. We exclude the root node for the possibility of modification and poisoning since every DNS server has a local copy of *root hint* file and record about the root servers remains unchanged for many years.

B. Evolution of A Standard DNS Cache

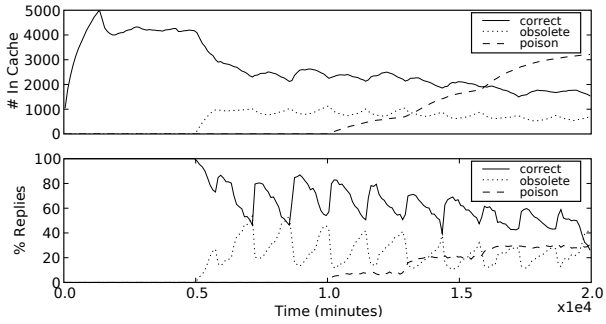


Fig. 4. Evolution of a Standard DNS Cache

Figure 4 presents the evolution of a standard DNS cache under various conditions. Ideally, one hopes more records are cached correctly so that queries can be answered locally. Arrivals of queries cause records to be cached but the TTL expiration mechanism also flushes records. Since all records have a TTL expiration, a certain arrival rate will only keep a certain number of records in the DNS cache. This is observed in Figure 4 before $t = 5000$.

Starting from $t = 5000$, modifications to DNS records are introduced. Consequently, certain cached records become obsolete and this affects the accuracy of some replies. The actual ratio of obsolete records depends on the arrival of modifications and the TTL values of records.

Poison is injected at time 10000 for ".com", at 12500 for ".net" and at 15000 for ".org" respectively. We notice that the poison does not start to propagate or cause poisoned records being returned for a small period. This is because a typical application does not query ".com" directly. Instead, they query for leaf nodes like "www.cnn.com" and if "www.cnn.com" or "cnn.com" is cached, the recursive lookup will start from the best-match instead of ".com". However, once queries starts to reach the poisoned nodes, it propagate down and spreads in the cache.

To inflict maximum damage, malicious attackers are likely to set a very large TTL value for the poisoned record. Although most DNS servers have a maximum allowable TTL value (default is 7 days for BIND), it is significantly longer than the TTL values of most DNS records. Consequently, we observe a steady increase of poisoned cache over time until all names under the ".com"(or ".net" or ".org") branch is poisoned. Because of this poison propagation mechanism, the attacker need to succeed only once to affect a large number of users.

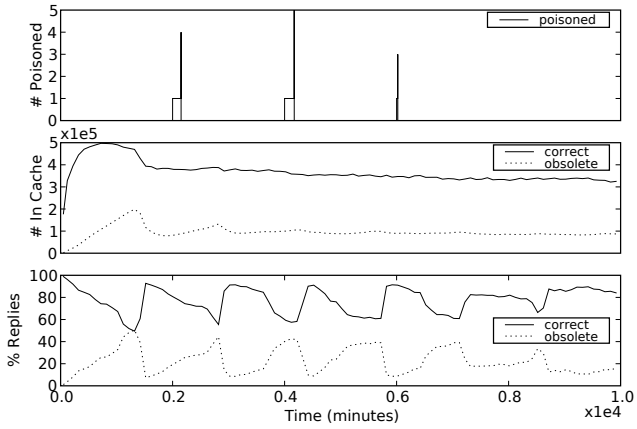


Fig. 5. Poison Detection and Removal

C. DoX for Poison Detection and Removal

Figure 5 presents the evolution of DoX-protected DNS cache based on a strict strategy in which every answer obtained through DNS lookup is checked before it is relayed to the querier. As long as there is at least one verifying peer that is not poisoned, the requesting peer will be able to detect the poison and remove it by flushing. In Figure 5, poison is injected at time 2000 for ".com", at 4000 for ".net" and at 6000 for ".org". In the top figure, the injected poison can stay in the cache for a while until a query *hits* it. However, it is detected by DoX and removed immediately if it causes any poisoned response to be returned. Since DoX stops poison from propagating, we do not observe poisoned records in the cache or poisoned replies as in Figure 4. In addition, the percentage of correctly cached records is significantly increased.

D. DoX for Improving Cache Consistency

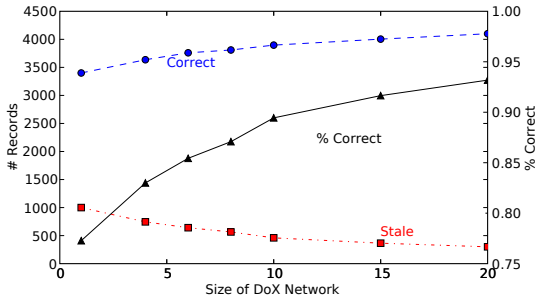


Fig. 6. Improving Cache Consistency

In DoX, a verifying peer will forward a valid record update to other peers in addition to send the confirmation to the requesting peer. Consequently, all peers in the DoX network will update to the current record as soon as one peer notices the update, independent of the DoX topology. Figure 6 shows that the average number of obsolete records decreases when the size of the DoX network increases. Therefore, peers in a large DoX network will observe negligible amount of obsolete records.

E. DoX Overhead

The overhead incurred by DoX to the DNS infrastructure is determined by the volume of verification requests the entire DoX network is generating (λ_v). Although a DoX peer verifies every response it receives, the vCache limits it to checking with remote peers only if there is *record update*. Furthermore, a valid record update is propagated in a DoX network once it is observed. In a large DoX network, every record update will be first observed by a certain peer and made known to the entire network. Consequently, λ_v equals the modification rate λ_m instead of the query arrival rate λ_q. Under normal condition, one would expect λ_m to be small.

The number of verifying peers used for each verification request (N_p) determines the strength of the DoX verification algorithm. N_p can also be regarded as the node degree of DoX topology. It does not impact the overhead to the DNS infrastructure because the verifying peer first checks a record update with its local vCache and avoids repeated checks through the DNS infrastructure. However, waiting for the responses from a large number of peers will increase the latency of the verification process. This latency is experienced only by peers noticing the record update first and affects little on the average latency. However, it still presents a tradeoff between the strength and the worst-case performance of DoX.

VI. Related Work

Cao and Liu [14] studied several approaches to achieve strong cache consistency for the web, including adaptive TTL, polling-every-time, and proactive server invalidation. A server invalidation scheme can also be used for improving DNS cache consistency; however, the main difficulty is that the DNS server would have to maintain states for known clients. Invalidation, of course, does not protect against cache poisoning.

Cohen and Kaplan [15] proposed proactive caching of DNS records in which they use renewal policies to refresh selected cache entries by issuing unsolicited queries. In addition, they propose a *simultaneous-validation* (SV) in which the end-host will use an expired DNS entry to connect to the web server, but the content will be served only if the DNS entry is validated by the SV queries. Their focus is on improving cache hit rate.

Substantial work has been done using structured overlays to provide scalable DNS lookups. Overlook [16] is a new name service based on Pastry. DDNS [17] is a P2P lookup service based on Chord. CoDNS [18] is based on CoDeeN and CoDoNS [19] is based on Beehive. The common aim of Overlook, DDNS, CoDNS and CoDoNS is to provide an alternative for the current aged DNS system. Exploiting the scalability and reliability of the underlying overlay network, these new DNS protocols achieve faster lookups and fewer failures. However, none of these works addresses the accuracy of DNS lookups. In fact, relying on peers for DNS lookups could be dangerous if they are untrustworthy.

In contrast, the focus of DoX is on the accuracy of DNS records. For a mission-critical system like DNS, its accuracy should receive as much attention as, if not more than, performance. DoX uses peers for consistency checks but never stores nor forwards records from peers directly. A malicious peer might cause DoX to unduly check with the authoritative servers in the worst case. However, the records will remain correct since a peer only uses records that it looks up itself.

The Netcraft Toolbar [20] aims to counter phishing attacks by displaying the hosting location of the IP address to the user. Users can manually detect poisoning in certain cases, e.g. if a cache poison directs a client to a server in one country while the client knows the server should be located in another counter. This can restrict the maneuverable range of poisoning but is not a complete solution.

VII. Conclusions

In this paper, we presented a new approach for combating DNS poisoning attacks based on the idea of cooperative verification among a set of peers. The scheme is simple, avoids any modifications to the current DNS infrastructure, and imposes only modest cost in terms of latency and overhead. Furthermore, the scheme assists in improving the coherence of DNS caches. The scheme was evaluated using simulation. We have also developed an analytic model for the scheme; the details of which will be presented elsewhere.

Future work on the subject includes a more comprehensive evaluation of the scheme both via simulation and analysis and formal proofs of correctness of the algorithms proposed.

References

[1] Symantec Corporation, "Symantec gateway security products DNS cache poisoning vulnerability," http://securityresponse.symantec.com/avcenter/security/Content/2004.06.21.html, June 2004.

[2] K. Haugsness and the ISC Incident Handlers, "DNS cache poisoning detailed analysis report version 2," http://isc.sans.org/presentations/dnspoisoning.php, Mar 2005.

[3] I. Green, "DNS spoofing by the man in the middle," http://www.sans.org/rr/whitepapers/dns/1567.php, Jan 2005.

[4] Netcraft Ltd., "DNS poisoning scam raises wariness of 'pharming'," http://news.netcraft.com/archives/2005/03/07/dns_poisoning_scam_raises_wariness_of_pharming.html, March 2005.

[5] J. Evers, "DNS servers–an Internet Achilles' heel," http://news.com.com/DNS+servers--an+Internet+Achilles+heel/2100-7349_3-5816061.html, Aug 2005.

[6] S. Kwan, P. Garg, J. Gilroy, L. Esibov, J. Westhead, and R. Hall, "Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG)," 2003.

[7] K. Kant, R. Iyer, and P. Mohapatra, "Architectural impact of secure socket layer on internet servers," in *International Conference on Computer Design (ICCD 2000)*, Sep 2000, pp. 7–14.

[8] D. E. Kaminsky, "Black Ops of TCP/IP 2005," http://www.doxpara.com/, 2005.

[9] V. Ramasubramanian and E. G. Sirer, "Perils of transitive trust in the Domain Name System," Cornell University, Computing and Information Science, Ithaca, New York, Tech. Rep., May 2005. [Online]. Available: http://www.cs.cornell.edu/people/egs/beehive/paper.php

[10] C. Schuba, "Addressing weakness in the Domain Name System protocol," Master's thesis, Purdue University, 1993.

[11] "Description of the DNS server secure cache against pollution setting," http://support.microsoft.com/kb/316786/EN-US/, Apr 2005.

[12] J. Stewart, "DNS cache poisoning - the next generation," http://www.securityfocus.com/guest/17905, Jan 2003.

[13] "dmoz - open directory project," http://www.dmoz.org.

[14] P. Cao and C. Liu, "Maintaining strong cache consistency in the world wide web," *IEEE Transactions on Computers*, vol. 47, no. 4, April 1998.

[15] E. Cohen and H. Kaplan, "Proactive caching of DNS records: Addressing a performance bottleneck," in *Proceedings of the Symposium on Applications and the Internet*, 2001.

[16] M. Theimer and M. B. Jones, "Overlook: Scalable name service on an overlay network," in *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, 2002.

[17] R. Cox, A. Muthitacharoen, and R. T. Morris, "Serving DNS using a peer-to-peer lookup service," in *Proceedings of IPTPS' 02*, 2002.

[18] K. Park, V. Pai, L. Peterson, and Z. Wang, "CoDNS: Improving DNS performance and reliability via cooperative lookups," in *Proceedings of the Sixth Symposium on Operating Systems Design and Implementation(OSDI '04)*, 2004.

[19] V. Ramasubramanian and E. G. Sirer, "The design and implementation of a next generation name service for the internet," in *Proceedings of SIGCOMM*, Portland, Oregon, August 2004.

[20] Netcraft Ltd., "Netcraft toolbar," http://toolbar.netcraft.com, 2005, released in 2004?