Assessing DNS Vulnerability to Record Injection*

Kyle Schomp[†], Tom Callahan[†], Michael Rabinovich[†], Mark Allman[‡]

[†]Case Western Reserve University, Cleveland, OH, USA
[‡]International Computer Science Institute, Berkeley, CA, USA

Abstract. The Domain Name System (DNS) is a critical component of the Internet infrastructure as it maps human-readable names to IP addresses. Injecting fraudulent mappings allows an attacker to divert users from intended destinations to those of an attacker's choosing. In this paper, we measure the Internet's vulnerability to DNS record injection attacks—including a new attack we uncover. We find that record injection vulnerabilities are fairly common—even years after some of them were first uncovered.

Keywords: Domain Name System (DNS), Measurement, Security, Cache Poisoning

1 Introduction

The Domain Name System (DNS) is a critical component of the Internet infrastructure. DNS maps human-readable hostnames (e.g., "amazon.com") to IP addresses and is involved to some degree in most Internet transactions. Given the foundational role of DNS in today's Internet, DNS security has a profound effect on the overall security, trust, and operability of the network. In particular, substituting an authoritative mapping with a fraudulent record allows an attacker to divert user access to nefarious hosts with implications ranging from replacing the original content and phishing attacks to installing malware on client hosts. In this paper, we measure the prevalence of DNS vulnerabilities to attacks designed to substitute the authoritative mapping. Collectively, these attacks are known as "record injection" attacks. We consider known attacks and a new vulnerability we uncover, as well as the extent of the adoption of suggested best-practice defenses.

Fraudulent hostname-to-IP address mappings originate in two places: (i) a component in the hostname resolution machinery (e.g., a local DNS resolver) or (ii) a man-in-the-middle that can monitor DNS transactions and either change or inject responses. A variant of the first is a cache poisoning attack whereby an attacker populates the cache of a DNS resolver with an illegitimate record, which the resolver then uses to satisfy subsequent requests for the given hostname.

Cache poisoning attacks generally rely on open DNS resolvers that will act upon DNS requests from arbitrary Internet hosts. Open resolvers have long been a known security issue. However, the prevalence of such resolvers is increasing—from 15M in 2010 [11] to 30M in 2013 [14]. While not all open resolvers are vulnerable, their increasing numbers provide a larger attack surface that we must understand. Moreover, as

* Work supported in part by NSF grants CNS-0831821, CNS-1213157 and CNS-1237265.

we discuss below, open resolvers often give attackers a vector to attack closed resolvers, which further weakens the overall system.

The Internet engineering community has spent considerable energy fortifying DNS with DNSSEC [1] which cryptographically protects the integrity of the authoritative bindings set by the holder of a name. While DNSSEC is the long-term security strategy for the DNS, deployment is currently low—with only about 1% of the resolvers validating DNSSEC records [6, 9]. Given the low DNSSEC deployment, understanding the security landscape of DNS without DNSSEC remains of critical importance.

Unfortunately, assessing the extent of security threats within the DNS infrastructure is anything but straightforward. The path a DNS transaction takes through a maze of intermediate resolvers is often both complex and hidden from external view. This paper develops techniques to attribute vulnerabilities to various actors in this infrastructure. Our key observations are: (i) that some closed resolvers are still vulnerable to cache poisoning, (ii) while vulnerability mitigations exist, deployment is not ubiquitous, and (iii) 7–9% of home networks are vulnerable to a simple new cache poisoning attack we uncover. Our general finding is that DNS security soft spots are not rare—even for vulnerabilities that have been known for years. Finally, note that our datasets are available for community use [13].

2 Terminology and Methodology

The architecture of the client-side DNS resolution infrastructure varies across providers—which we discuss in depth in companion work [14]. Here we provide a short overview of our terminology. Generally, client systems do not query authoritative DNS servers (“ADNS”) directly, but rather rely on a recursive resolver, which we denote “RDNS”, to handle these interactions and return the final address mapping. An RDNS may optionally leverage additional RDNS servers in the lookup process. We denote open resolvers that will answer arbitrary requests as “ODNS”. We often find that ODNS resolvers do not perform recursive lookup themselves, but rather simply forward requests to an RDNS. We denote a forwarding ODNS as an “FDNS”. The RDNS querying our ADNS for an FDNS is an indirect RDNS, which we denote “RDNS$_i$”.

Our basic methodology for studying the vulnerability of the client-side DNS infrastructure is to probe the Internet in search of ODNS resolvers, similar to previous efforts [5, 14]. We register a domain and deploy an ADNS for this domain.[1] We then use approximately 100 PlanetLab [3] nodes to randomly scan the IP address space with DNS requests for various hostnames within our domain. We embed the IP address of the target of our scan in the hostname request. Therefore, the queries arriving at our ADNS illuminate the set of ODNS servers. Additionally, the ADNS can use the source IP address to discover the set of RDNS resolvers. Given these two pieces of information, we can distinguish between ODNS resolvers that are themselves performing recursive lookup from those that are merely forwarding the requests to another resolver—i.e., the set of FDNS servers. Table 1 provides information about the datasets we discover and

[1] Note, unless otherwise stated, we always work within our own unused namespace as to not interfere with users’ normal activities.

Scan	Begin	Dur. (Days)	# ODNS	# RDNS
S_1	2/29/12	17	1.09M	69.5K
S_2	3/1/13	11	40.5K	5.3K
S_3	7/19/13	12	2.31M	86.1K

Table 1. Collected Datasets

Observation	RDNS	
	No.	%
Total	69K	100%
Unclassified	12K	18%
Classified	57K	82%
Complex Trans. ID Seq.	57K	100%
Var. Ephemeral Port	48K	84%
0x20 Encoding	195	0.3%

Table 2. RDNS Characteristics

utilize in the remainder of the paper. While the general methodology we sketch here applies to all our experiments, the specifics vary across experiments as we study different aspects of the infrastructure. The specifics are given in the relevant sections below.

Note, we return to methodological issues in § 8. In particular, we use the techniques we develop in the paper to address two specific issues. First, we aim to understand whether the ODNS servers we find are actually in operational use by real users. Second, since we do not probe the entire Internet address space, we seek to understand if our sample is representative of the broader Internet.

3 Kaminsky's Attack

Kaminsky [10] describes a DNS cache poisoning attack which leverages the connectionless nature of typical UDP-based DNS requests to insert an NS record[2] into the victim's cache. The Kaminsky attack proceeds with the attacker A sending a large number of requests for hostnames within a domain to be poisoned, $P.com$, to a victim RDNS V in the form of queries for $random\_string.P.com$. A legitimate response to such requests must (i) be from the ADNS for $P.com$, (ii) be directed to the correct ephemeral UDP port number (the source port listed in the request message), (iii) contain the query string from the request and (iv) use the transaction ID assigned in the request. However, A knows the query string and can readily determine and spoof the IP address of the ADNS—leaving only checks (ii) and (iv) as protection against illegitimate responses. By sending a large number of requests with different query strings, A can then use brute force guessing of port numbers and transaction IDs in forged replies until a reply is accepted by V.

Mitigating the Kaminsky attack involves increasing the amount of entropy in DNS requests such that the average cost of mounting a successful attack is prohibitively high. Resolvers can increase entropy by randomizing both the DNS transaction ID and the ephemeral port number. While randomizing only the DNS transaction ID is insufficient protection, randomizing both values is an effective strategy [10]. Another technique to increasing entropy is "0x20 encoding" [4] in which the RDNS randomly changes the capitalization throughout query strings. Authoritative servers should be case insensitive

[2] An NS record contains the hostname of the ADNS for all hostnames within a particular domain. For instance, Google has an NS record that indicates the authoritative source for the binding of "news.google.com" to an IP address.

when resolving the query yet retain the capitalization in their response [12]. Hence, checking that the capitalization in the request and response matches is another way to decrease an attacker's likelihood of forming an acceptable response.

To understand the vulnerability to the Kaminsky attack we assess the adoption of the strategies for enhancing entropy by sending multiple requests for unique hostnames to each ODNS. Then, in our S_1 dataset, we check successive queries from a single RDNS at our ADNS for variation in the ephemeral port selection, DNS transaction ID, and for the use of 0x20 encoding. Table 2 shows our results. First, our dataset does not contain enough requests,[3] to accurately characterize 12K (or 18%) of the RDNS resolvers. For the remainder, we find that nearly all RDNS resolvers employ a complex (presumably random) method for selecting DNS transaction IDs.[4] Further, 84% of the classified RDNS resolvers use some variation in their ephemeral port selection. That means 9K RDNS servers *use a static ephemeral port on all transactions*! Per the discussion above, both the ephemeral port and the transaction ID values must be random to thwart attacks and therefore roughly 16% of classified RDNS servers are vulnerable to the Kaminsky attack. Furthermore, we observe RDNS resolvers using static source ports in 37% of the autonomous systems in our dataset, which illuminates the breadth of the issue. Additionally, we find that 0x20 encoding is in use by roughly 0.3% of RDNS resolvers—showing that resolvers are generally not using this strategy for increasing the entropy of requests.[5] These results are nearly identical when only considering the RDNS$_i$ subset of RDNS resolvers that we know to serve FDNS clients.

Finally, we note that the use of *RDNS pools* (e.g., [7, 14]) serves to mitigate the Kaminsky attack as well. Regardless of the IP address the attacker uses as entry point into the pool, the IP address used to communicate with the ADNS is chosen according to an algorithm unknown to the attacker. Therefore, to launch a Kaminsky attack against an RDNS pool, the attacker must either target every RDNS in the pool simultaneously, know how the pool distributes requests internally, or guess the destination IP address.

4 Bailiwick Rules Violations

Bailiwick rules prevent malicious ADNS servers from inserting fraudulent records into resolvers' caches [2]. Under this attack, a legitimate response from $X.com$ also includes an "additional answers" section that supplies arbitrary unrelated bindings—e.g., for $www.Y.com$. To potentially save time later, a susceptible resolver adds $www.Y.com$ to its cache. During our S_1 scan, we test for this vulnerability by returning legitimate responses from our domain that also include information for a non-existent google.com subdomain.[6] We then query the ODNS for the google.com subdomain and determine

[3] We require at least ten transactions for the results in this paper, but in other experiments we find the insights are not sensitive to the exact threshold.

[4] We conclude that resolvers do not use static, incrementing, or decrementing transactions IDs by observing a high standard deviation in the transaction ID sequence.

[5] Our results may be a lower bound on the adoption of 0x20 encoding as at least one major RDNS pool—Google Public DNS [8]—uses 0x20 encoding on a white-listed set of domains. Unfortunately, we have no information on the prevalence of white-listing.

[6] This will not interfere with regular Google traffic as the hostnames involved are not in use.

whether the response includes our poisoned result or an error message from Google indicating a non-existent domain.

Preventing this attack can be accomplished through the implementation of bailiwick rules—such as checking that any records in the “additional answers” section belong to the domain owned by the responding ADNS. In the most simplistic attack, we find 675 cases where client-side DNS infrastructure readily caches a DNS response for a mapping we provide for a bogus google.com subdomain. Furthermore, we observe 231 cases where the resolvers cache any additional record from a response to an MX query (these are queries for the mail server for the domain in question) and 203 cases where the resolver caches any additional CNAME-type record. Overall, there are a total of 749 cases where we find a resolver falling prey to at least one of these record injection attacks. While a relatively small number, these RDNS resolvers are completely exposed to crude poisoning by malicious ADNS servers, with no guessing involved.

5 Preplay Attack

The Kaminsky attack requires an attacker to forge an acceptable DNS response. However, in the course of our investigation we determined that FDNS servers were vulnerable to a previously unknown injection attack. While FDNS servers do not themselves recursively look up mappings, they often do have caches of previous lookups. The FDNS servers populate these caches with the responses from upstream RDNS resolvers. In some cases we find that FDNS servers fail to validate the DNS responses. This leaves these FDNS servers vulnerable to the crudest form of cache injection: a “preplay” attack whereby an attacker sends a request to a victim FDNS and then, before the legitimate response comes back, the attacker answers the request with a fraudulent response. The FDNS will then forward the fake response to the originator and cache the result. An FDNS that (i) forwards requests with a new random ephemeral port number and DNS transaction ID and (ii) verifies these and the upstream RDNS’ IP address on returning responses would be protected against the preplay attack. Such protections would reduce an attacker to guessing a variety of values in the short amount of time before the legitimate response from the RDNS arrives. However, we find a non-trivial number of FDNS servers simply forward on the packets received and/or do not verify the values on DNS responses. This leaves the door open for a crude attack whereby an attacker does not have to guess these values, but can just use those from the original request.

To assess the extent of this vulnerability during our S_2 and S_3 experiments, we send a request for a hostname within our domain to each ODNS and immediately issue a fraudulent response containing IP address X. On the other hand, our ADNS responds to these requests with a binding to IP address Y. The probing host issues a subsequent request and determines which IP address is in the ODNS’ cache.

In its most primitive form, the preplay attack does not involve spoofing or guessing to make the fraudulent response appear legitimate—we use the ephemeral port number and DNS transaction ID from the original request. Additionally, we use the probing machine’s genuine IP address. We use variants of this attack that attempt to leverage information arriving at the ADNS (e.g., the RDNS server’s IP address) to craft DNS responses that look more legitimate. However, to date our variants do not point to higher vulnerability rates. Finally, while we use the DNS default port 53 as the ephemeral port in the results herein, using a random ephemeral port number shows similar results.

We first test the preplay attack during the S_2 scan. Unlike our other scans which were performed from PlanetLab nodes, the S_2 scan leverages a single node in a residential network.[7] For each ODNS we attempt each attack variant three times to reduce any impact from packet loss. Of the roughly 41K ODNS servers we test, we find 3.5K (or 8.6%) to be vulnerable to the preplay attack. Therefore, we conclude that ODNS servers are failing to take three simple measures to thwart this attack: (i) use a new and random DNS transaction ID, (ii) verify that the source IP address in DNS responses matches the IP address of the upstream RDNS, and (iii) verify the destination port number on responses. The latter is particularly intriguing as it suggests these devices are not running a traditional protocol stack in which packets arriving on an unbound port number are dropped. Given we find no increase in the success rate with our attempts at spoofing, we return to PlanetLab with the S_3 scan to assess the vulnerability at a larger scale. Of the 2.3M ODNS servers we test, we find 170K (or 7.3%) to be vulnerable to the preplay attack.

6 DNS Message Rewriting

We now examine DNS record modification by network operators. Depending on one's perspective this may or may not be considered a security issue. However, we believe that responses deviating from the authoritative intent are at least worth understanding.

NXDOMAIN Rewriting: A DNS request for a non-existent name evokes a response with the "NXDOMAIN" return code. Previous anecdotal observations indicate that such responses are prone to interception and replacement with valid addresses by some ISPs and DNS providers. This practice is generally attempting to monetize the unfulfilled request (e.g., by trying to sell the domain or sending a user to a similar page in an attempt to meet their intent). In our S_1 experiment, we send a request to each ODNS that causes our ADNS to return an NXDOMAIN message. We find that roughly 258K (23.7%) of ODNS servers are subject to NXDOMAIN rewriting as we receive an address in response to our invalid query, which is close to previous measurements [15][8] To understand who may be responsible for rewriting, we analyze the set of RDNS resolvers on the path of rewritten messages. We determine an RDNS is a probable rewriter if more than half the open resolvers served by the RDNS experience rewriting. We find over 100 ISPs/DNS providers that we suspect of performing rewriting by default, including Qwest/Centurylink, OpenDNS, Frontier, Rogers, Airtel, RoadRunner, and TE Data.

Search Engine Hijacking: Previous work shows several ISPs alter DNS responses from major search engines in an effort to place a proxy between the user and the search results [15–17]. This allows the ISPs to monetize users' searching (e.g., by placing ads

[7] Due to some of our (unreported) tests using spoofed addresses (against PlanetLab's AUP).

[8] As a methodological note, one must be careful in selecting query strings. For instance, we initially misclassified OpenDNS as not performing rewriting because our queries began with dotted-quad IP addresses—a pattern OpenDNS excludes from its rewriting process. A second pass with a different query string correctly classifies OpenDNS as a rewriter. Thus, our findings are conservative due to other potential edge cases.

on the results). Since our strategy allows us to assess ISPs' RDNS resolvers, we investigate this behavior and find no evidence the practice is now in widespread use. Still, we find 18 smaller regional ISPs that appear to rewrite DNS responses for google.com.

7 Implications

Duration of Record Injection: The injection attacks we discuss above can only be successful when part of the DNS infrastructure caches a fraudulent record and then returns that record in response to a normal user request. An assessment of the caches of FDNS and RDNS resolvers [14] finds (i) little evidence of cache evictions based on capacity limits and (ii) that records with long TTLs—which can be set in injected records—stay in the cache for at least one day in 60% of the RDNS$_i$ resolvers and 50% of the FDNS servers. This shows the impact of record injection can be long-lived.

Indirect Attacks: It is not enough for RDNS resolvers to act on requests only from authorized devices as these devices are in turn commonly globally accessible and open RDNS resolvers to indirect attacks. The large and growing set—doubling to over 30M in the last three years—of open resolvers [14] represents an attack vector to otherwise inaccessible RDNS resolvers. For instance, we find that 62% of the RDNS resolvers in the S_1 dataset do not answer external queries and yet we are still able to probe these servers. Further, using ODNS servers to indirectly attack other portions of the DNS ecosystem provides a layer of obfuscation that helps attackers escape attribution.

Phantom DNS Records: A class of denial-of-service attacks relies on placing a large DNS record in a cache (at an RDNS, say) and then spoofing requests that will cause the record to be sent to some victim. This can both hide the actual origin of the attack, as well as amplify (in volume) an attackers traffic by using records that are larger than requests. To date this requires attackers to register a domain and serve large records to insert them into the various caches or find an ADNS that is already serving large records. However, using record injection techniques, an attacker does not need to be bound to any centralized infrastructure. In fact, any domain could be readily inserted in the cache and then used in a subsequent attack. This leaves less of a paper trail that can potentially trace back to an attacker. The preplay attack allows such record injection into millions of devices with trivial effort.

8 Context

We now return to contextual issues surrounding our measurements, as sketched in § 2.

Are Open Resolvers Used? We first turn to the question of whether ODNS servers in fact serve users or are active, yet unused artifacts. This bears directly on whether the preplay attack represents a real problem. First, in companion work we use several criteria—including scraping any present HTTP content on the ODNS, consulting blacklists of residential hosts and observing UDP protocol behavior—to determine that "78% [of ODNS servers] are likely residential networking devices" [14]. Using the same criteria against the FDNS servers in the S_3 scan, we find that 91% of the FDNS servers that are vulnerable to the preplay attack are likely residential network devices. While this result does not speak directly to use, our experience is that these devices act as DNS forwarders for devices within homes and therefore we believe this suggests actual use.

Additionally, we seek to test directly for evidence that the FDNS servers we probe are in use by some client population. We start by gathering round-trip time (RTT) samples for each FDNS and the corresponding RDNS. For the FDNS we use the preplay attack to measure the RTT by taking the time between sending a fraudulent response to the FDNS and receiving the response back from the FDNS at our client. Measuring the RTT to the RDNS is more complicated. The process starts with the client requesting some name N from our ADNS. The ADNS responds with some CNAME N', which the RDNS then resolves and our ADNS returns a random address A. The mapping between N and A then returns to the FDNS and ultimately our client. The client then issues a request for N'—which will presumably be in the RDNS' cache, but given the primitive nature of preplay-vulnerable FDNS devices not in the FDNS' cache. The response for N' will be A when the RDNS answers the request from the cache.

After we obtain RTTs for both FDNS and RDNS, we seek to understand whether popular web site names are in the FDNS cache as a proxy for whether the FDNS is in use by a population of users. We therefore issue DNS requests for the Alexa[9] top 1,000 web sites and time the responses.[10] Given unreliable TTL reporting by FDNS servers [14], we determine that a given hostname is in the FDNS cache using the time required to resolve the name. Since we expect individual FDNS to have diurnal variation, we perform the lookups on each FDNS every 4 hours for one day. Our own queries will populate the FDNS cache and therefore we must exercise care with subsequent probes lest we wrongly conclude users employ the FDNS when it is our own probes we observe. We mitigate this issue in two ways. First, we probe all names with an authoritative TTL of 4 hours or more only once, accounting for 415 names. Second, we inject records into FDNS cache from our ADNS with the same TTLs as the remaining 1,585 records in our corpus (all of which are less than 4 hours). At each 4 hour interval we check our own records and if the FDNS incorrectly returns a record that had an initial TTL of x we exclude all but the initial query for popular names with an initial TTL of at least x.

We determine that a given hostname is in the FDNS cache if the time required to resolve the name during our S_3 scan does not exceed the median FDNS RTT. Figure 1 shows the distribution of the fraction of FDNS servers that hold a given number of records in their cache. The "All" line shows the distribution for all preplay-vulnerable FDNS servers. We find 81% of the FDNS servers have at least one popular name in their cache at some point during the experiment. However, the distribution also shows that over 30% of the FDNS servers have at least 100 hostnames in the cache. This seems unlikely and we believe these represent cases where our heuristic is not properly delineating between the FDNS and RDNS cache. Therefore, in an effort to better delineate the FDNS and RDNS caches we plot the subset of FDNS servers where the maximum FDNS RTT is at least 10 msec less than the minimum RDNS RTT, which we denote as "Far from RDNS". This subset encompasses 8.4K FDNS servers and we do see the tail of the distribution fall away. Within this subset, 53% of the FDNS servers are in use.

[9] www.alexa.com

[10] Note, we augment the list of sites by prepending each web site name from Alexa with "www"—which is not included in the list—and we therefore probe for 2,000 names.

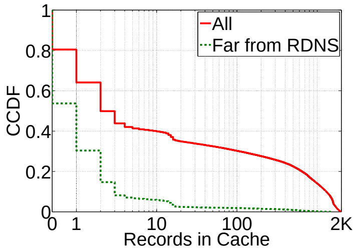

Fig. 1. Distribution of popular websites in FDNS server caches.

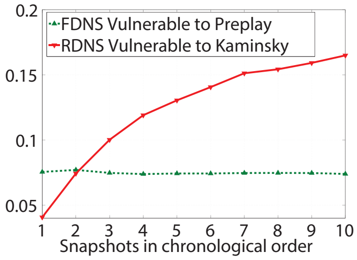

Fig. 2. Vulnerability frequency at snapshots during discovery.

Additionally, we examine the subset of FDNS servers that are accessible for our entire 24 hour experiment. In this subset, we find more in-use FDNS servers—90% of all FDNS servers and 68% of FDNS servers that are far from their corresponding RDNS resolver. We exclude the 24 hour lines from the graph for readability.

Note, our heuristic provides a lower bound on the number of in-use FDNS servers since we only measure a fraction of the 24 hour period. Indeed, the median TTL for the popular names is 10 minutes. Assuming the median TTL, an FDNS that enforces the TTL and an FDNS available for 24 hours, our strategy provides a one-hour window into the FDNS' cache—or, just over 4% of the day. Further, our extensive probing of the FDNS' cache may actually overflow the cache thus pushing out records added via use. Therefore, we believe that many of the FDNS servers that do not show as in use are in fact in use, but that short TTLs and our coarse and extensive probing conspire to hide the use. Our general conclusion is that the FDNS servers we find are in fact in use by people during their normal browsing.

Representativeness: Finally, we return to the issue of representativeness of our results as mentioned in § 2. Since our scans do not encompass the entire Internet our insights could be skewed by our scanning methodology. To check this we divide our scans into ten chronological slices and derive the cumulative vulnerability rate at each slice for the Kaminsky and preplay attacks. The slices are equal in size in terms of the number of vulnerable RDNS servers and vulnerable ODNS servers for the Kaminsky and preplay vulnerabilities, respectively. The cumulative vulnerability rate should plateau once the dataset is typical of the broader population. Figure 2 shows the cumulative vulnerability rate across the ten slices for both attacks. The FDNS vulnerability rate reaches steady state immediately, illustrating that we are in fact capturing a representative sample of FDNS servers with random sampling of IP addresses—which is not surprising.

On the other hand, the figure shows that for the Kaminsky attack, the vulnerability rate increases as the scan progresses, indicating that the RDNS resolvers we discover at the beginning of the scan are *less* vulnerable to the Kaminsky attack than those we discover later in the scan. While we choose ODNS servers at random, we only indirectly discover RDNS resolvers. In particular, the probability of discovering an RDNS resolver is directly proportional to the size of the FDNS population that it serves. Thus, as the scan proceeds the discovery rate decreases and we find smaller scale RDNS resolvers. We believe these results sum to indicate that busier RDNS servers are better maintained and less vulnerable to the Kaminsky attack. The plot also indicates that our estimate of the Kaminsky vulnerability rate is a lower bound.

9 Conclusion

In this study, we assess the susceptibility of the client-side DNS infrastructure to record injection attacks. We find that many open resolvers are still vulnerable to record injection. Further, these devices provide a back door to attack shared DNS infrastructure. Through active probing, we assess the extent of known record injection threats and the deployment of known protective techniques. We further uncover and measure a new attack vector—the preplay attack. We find 7–9% of the open DNS resolvers are vulnerable to the preplay attack and 16% of recursive DNS servers are vulnerable to the Kaminsky attack. Therefore, we conclude that the client-side DNS ecosystem is non-trivially vulnerable to record injection attacks.

References

1. R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. *RFC 4033*, 2005.
2. D. Bernstein. `http://cr.yp.to/djbdns/notes.html`.
3. B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM CCR*, 33(3), 2003.
4. D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee. Increased DNS Forgery Resistance Through 0x20-bit Encoding: Security via Leet Queries. In *ACM CCS*, 2008.
5. D. Dagon, N. Provos, C. Lee, and W. Lee. Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. In *NDSS*, 2008.
6. K. Fujiwara. Number of Possible DNSSEC Validators Seen at jp. In *DNS-OARC Workshop*, 2012.
7. Google Public DNS. Performance Benefits. `https://developers.google.com/speed/public-dns/docs/performance`.
8. Google Public DNS. Security Benefits. `https://developers.google.com/speed/public-dns/docs/security`.
9. O. Gudmundsson and S. Crocker. Observing DNSSEC Validation in the Wild. In *Workshop on Securing and Trusting Internet Names (SATIN)*, 2011.
10. D. Kaminsky. Black Ops 2008: It's the End of the Cache As We Know It. *Black Hat USA*, 2008.
11. D. Leonard and D. Loguinov. Demystifying Service Discovery: Implementing an Internet-Wide Scanner. In *ACM Internet Measurement Conference*, 2010.
12. P. Mockapetris. Domain Names Implementation and Specification. *RFC 1035*, 1987.
13. K. Schomp, T. Callahan, M. Rabinovich, and M. Allman. Client-Side DNS Infrastructure Datasets. `http://dns-scans.eecs.cwru.edu/`.
14. K. Schomp, T. Callahan, M. Rabinovich, and M. Allman. On Measuring the Client-Side DNS Infrastructure. In *ACM Internet Measurement Conference*, 2013.
15. N. Weaver, C. Kreibich, B. Nechaev, and V. Paxson. Implications of Netalyzr's DNS Measurements. In *Workshop on Securing and Trusting Internet Names (SATIN)*, 2011.
16. N. Weaver, C. Kreibich, and V. Paxson. Redirecting DNS for Ads and Profit. In *Workshop on Free and Open Comm. on the Internet*, 2011.
17. C. Zhang, C. Huang, K. Ross, D. Maltz, and J. Li. Inflight Modifications of Content: Who Are The Culprits? In *LEET*, 2011.