Internet-Wide Study of DNS Cache Injections

Amit Klein, Haya Shulman and Michael Waidner
Fraunhofer Institute for Secure Information Technology SIT
Darmstadt, Germany

***Abstract*—DNS caches are an extremely important tool, providing services for DNS as well as for a multitude of applications, systems and security mechanisms, such as anti-spam defences, routing security (e.g., RPKI), firewalls. Subverting the security of DNS is detrimental to the stability and security of the clients and services, and can facilitate attacks, circumventing even cryptographic mechanisms.**

We study the caching component of DNS resolution platforms in diverse networks in the Internet, and evaluate injection vulnerabilities allowing cache poisoning attacks. Our evaluation includes networks of leading Internet Service Providers and enterprises, and professionally managed open DNS resolvers. We test injection vulnerabilities against known payloads as well as a new class of *indirect attacks* that we define in this work. Our Internet evaluation indicates that more than 92% of the Internet's DNS resolution platforms are vulnerable to records injection and can be persistently poisoned.

I. Introduction

Domain Name System (DNS), [RFC1034, RFC1035], plays a key role in the Internet. However, its significance also made it a target of attacks, most notably, DNS cache poisoning, [1], [2], [3], [4], [5], [6], [7], [8]. In the course of a DNS cache poisoning attack, the attacker provides spoofed records in DNS responses, in order to redirect the victims to incorrect hosts. DNS cache poisoning can facilitate credentials theft, malware distribution, censorship and more. DNS cache poisoning attacks are known to be practiced by governments, e.g., China, [9], USA with the QUANTUMDNS program [10], as well as by cyber criminals.

The recent wave of cache poisoning vulnerabilities as well as evidence for attacks against DNS in the wild stimulated awareness within the operational and research communities, and a number of studies measuring incorrect DNS responses in the wild were conducted, [11], [12], [13], [14]. Nevertheless, it is not clear *how effective the injection of DNS records is and whether, and under what conditions, such records poison the caches of victim DNS platforms*. In particular, different resolvers' software apply different logic when deciding if to accept and cache the records and if to return them to clients. In this work we perform the first study of caches in popular DNS resolution platforms in the Internet, and evaluate the effectiveness of records injection different networks, including networks of large Internet Service Providers (ISPs), public DNS service operators and enterprise networks. The findings of our study are alarming: 92% of the measured networks are vulnerable to records injection.

Adoption of DNSSEC, [RFC4033-RFC4035], would prevent the vulnerabilities. Unfortunately, recently [15] found that many domains are signed with vulnerable DNSSEC keys.

Related Work: Prior work studied the impact of caching in the resolvers on DNS performance and on the latency perceived by the clients, e.g., [16], [17]. In a recent work, [4], showed that a large fraction of nameservers use caches for reducing the requests' volume and to protect the nameservers from Denial of Service (DoS) and other attacks. [18], measured the client side of the DNS infrastructure, in order to identify all the actors in DNS resolution platforms that provide open recursive resolution. In their study Schomp et al used a similar host discovery technique to the one proposed in [19] – both scanned a fraction of the IPv4 addresses requesting hostnames in domains that they owned, and checking for requests arriving at the nameservers authoritative for those domains. A study of approaches for services discovery (including DNS) via scanning of the IPv4 address block was done in [20]. To optimise content distribution networks (CDNs) [21] ran a study of associating DNS resolvers with their clients.

To study ranking assignment by caches on DNS records, [22] applied a ProVerif formal verification program on the formal models of the semantics of 3 DNS resolvers. Recommendations for handling records in DNS responses, caching and returning them to clients were discussed with respect to Unbound DNS software in [23].

Contributions: In this work we perform an extensive study of the caches in DNS resolution platforms in well managed networks, comprising: (1) ISPs, (2) enterprises and (3) popular networks operating open resolvers. For our data collection we utilise three different approaches: (1) a distributed ad-network, (2) email servers in networks of popular enterprises, (3) open resolvers used in popular domains (according to Alexa websites ranking service, `www.alexa.com`). Through evaluation and measurements we find that more than 92% of the studied networks are vulnerable to at least one injection attack resulting in cache poisoning; this breaks down to 97% of the networks operating open resolvers, 74% of the enterprise networks measured via email servers, and 68% of the ISPs measured via ad-network.

Our study of records injections uses a comprehensive set of payloads, which consists of different combinations of records and records' types in DNS responses and in caches. We also define a new type of payloads which allow *indirect injection* of spoofed records for cache poisoning attacks.

Organisation: In Section II we present our study methodology and data collection. In Section III we define the payloads that we use in our evaluation of records injection. In Section IV we perform Internet evaluation of records injection against popular DNS resolution platforms and public DNS services.

978-1-5090-5336-0/17/$31.00 ©2017 IEEE

We conclude this work in Section V.

II. Methodology

In this section we describe the attacker model that we consider in this work, and the setting that we use for our study and the data collection methodology.

A. Attacker Model

We consider an attacker that can inject valid DNS responses into the communication between a victim DNS resolver and a nameserver. Namely, the attacker does not need to guess the challenge-response authentication parameters, such as the source port and the DNS transaction identifier (TXID). We assume that these are either known to the attacker, e.g., if the challenge values are not randomised, or that the attacker can guess them efficiently. For instance, a man-in-the-middle (MitM) attacker sees the DNS requests and hence can simply copy the challenge-response authentication parameters from the request to the response. An off-path attacker does not see the requests, therefore has to guess the challenge values. To that end, the attacker can apply fragmentation [8], which allows to bypass guessing the parameters, or can apply different techniques to guess the values, e.g., in vulnerable implementations or via side-channels, [24], [7].

Since the attacker is assumed to be able to bypass the challenge authentication (i.e., craft a DNS response with a correct destination port, TXID and other values), the remaining part is to provide such records to the victim resolver that would be accepted, cached and provided to the clients. Caching behaviour and which records the resolvers return to the clients, both depend on the records that are already in cache, the caching policies and the trust level that the DNS resolvers assign to the records that they receive in responses. Hence the attacker needs to find such records' sequence, records' type and hostnames that would enable the attacker to overwrite the values of the already cached legitimate DNS records with spoofed values. For instance, overwriting a legitimate IP address of a nameserver or a website with a spoofed one.

B. DNS Resolution Platforms

In our study, we consider a general model of DNS resolution platforms, illustrated in Figure 1. The platform consists of a set (2^{32-x}) of ingress DNS resolvers which handle DNS queries from the clients, a set of n caches, and a set (2^{32-y}) of egress DNS resolvers, which communicate with the nameservers if the queries from the clients cannot be satisfied from (one of) the caches. The load balancers apply logic for selection of the caches to probe, and for selection of an egress resolver's IP address (in case a requested record cannot be satisfied from caches).

This infrastructure corresponds to complex platforms such as Google Public DNS, but can also be abstracted to incorporate very simple DNS resolution platforms with a single IP address which performs both the ingress and egress functionalities and uses a single cache.

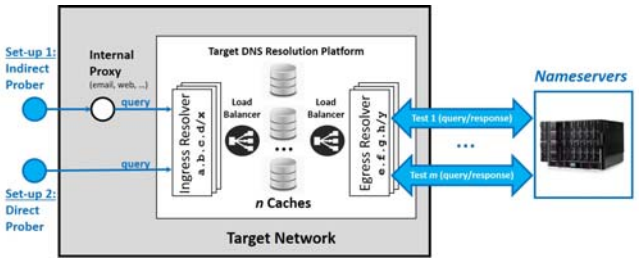

Fig. 1. DNS resolution platforms and our caches evaluation infrastructure.

Our test infrastructure in Figure 1 communicates with the egress resolvers, and uses two set-ups to communicate with the ingress resolvers: with direct and indirect probers, for triggering the caches study. A direct prober is one that can send queries to the ingress resolver directly, whereas indirect uses proxies, such as web browsers or email servers (details follow). Our infrastructure uses a dedicated domain, for simplicity lets assume it is `test.example` and allocates a number of subdomains, under `test.example`, for testing the caches. We setup a number of nameservers, authoritative for `test.example`, and nameservers authoritative for the subdomains of `test.example`.

Our implementation uses the Stanford::DNSserver Perl authoritative DNS server.

For the purpose of our study we evaluate poisoning the records of the domain `test.example` (which is under our control) and its subdomains. Specifically, we first insert into the caches the real values of records under `test.example` and then test the caching vulnerabilities. During that phase we attempt to replace the real values with spoofed ones. For evaluation of injection of spoofed values we set up a dedicated infrastructure which 'simulate' attacker hosts. The attacker's hosts are located on a different network block then the `test.example` infrastructure and use a distinct set of IP addresses. For instance, we replace an IP address, *a.b.c.d*, of a website under `test.example` with a malicious IP address *6.6.6.6* which is allocated to one of the hosts of the attacker. Then we issue a request to the evaluated DNS platform, say for IP address of the website under `test.example`, and check which value we receive in response: the real IP address *a.b.c.d* or the poisoned value *6.6.6.6*.

C. Data Collection and Statistics

In this section we describe our data collection in DNS resolution platforms in three common networks: (1) networks of popular Alexa domains (www.alexa.com) that operate open recursive resolvers, (2) networks of Internet Service Providers (ISPs), and (3) networks of popular enterprises.

1) Collecting DNS Platforms with Open Resolvers: A number of studies were conducted on open resolvers, e.g., [18], [12], [25]. The studies scan the IPv4 address block, looking for hosts that serve DNS requests on port 53. However, [19], [26] showed that most such open resolvers are either

(misconfigured) home routers and mismanaged (security oblivious) networks or malicious networks operated by attackers (e.g., where the open DNS resolver is set up for malware communicaton to the command and control servers). In this work we take a complementary approach and study open resolvers in *popular* and *well managed* networks.

Our population of networks running open resolvers includes Google Public DNS and OpenDNS and 1K networks operating open resolvers among top-10K Alexa networks (taken from `www.alexa.com`).

We obtain the IP addresses of 1K domains containing open resolvers out of 10K-top Alexa networks in two steps: (1) we queried the 10K-top Alexa domains for nameserver (NS) records, and their corresponding IP addresses (A records); (2) we select the first 1K domains that provide open DNS resolution services, by querying these IP addresses for records in our domain.

We check that our networks with open resolvers do not operate open relays, contain correctly configured PTR records, and are stable over time. Specifically, it was shown that open resolution service provided by malware or home routers is unstable, with IP addresses going offline within a period of hours, [27]. In contrast, the IP addresses of the open resolvers that we used remained stable over a period of 2015-2017.

2) Collecting DNS Platforms with Email Servers: In addition to popular domains, Alexa service provides listing of popular networks according to categories. We select the 1K-top enterprise networks, and collect the IP addresses and MX (mail exchanger) records of the SMTP (Email) services on those networks. For each SMTP email server, we establish an SMTP session, over which we sent an email message to a *non-existing* email-box in the target domain. Since the destination in the received email is a non-existing recipient, the receiving SMTP email server must generate a Delivery Status Notification (DSN) message to the originator of the email message informing the sender that the message could not be delivered. The rule to send bounce messages is mandated by [RFC5321], to enable the originator of email messages to detect and fix problems and prevent email messages from silently vanishing.

The email server, sending the bounce message, has to perform some DNS resolution via its local DNS resolver, typically searching for `MX` and `A` records of the target email server. Other query types are also possible, for instance networks that support DNS-based email sender authentication mechanisms (against spam), [28], may also issue DNS requests for corresponding DNS records. Indeed, during our evaluation of the servers in the Internet, we also received requests for Sender Policy Framework with `SPF/TXT` records [RFC4408] (69.6% of the requests), DomainKeys Identified Mail with `DKIM` [RFC6376] (0.3%), Domain based Message Authentication, Reporting and Conformance `DMARC` [RFC7489] (35.3%).

3) Collecting DNS Platforms with Web-Browsers: We use a popular ad-network to collect data from the resolvers used by web clients. To that end, we embedded our Javascript code in a SCRIPT tag in an ad network page, at a static URL. Our script is wrapped in an IFRAME by the ad network, and IFRAME is placed on webpages. When downloading the web page, the clients generate DNS requests via (ingress) recursive DNS resolvers on the networks of their ISPs to our test infrastructure.

We received more than 12K web clients, in which an AJAX call was made to our web server (indicating that the page was loaded and functional, e.g., Javascript running). Our test ran as a *pop-under* and needed several minutes to complete, hence only 1:50 of the cases were successful, and produced valid statistics for our study.

Our Javascript is non intrusive and only triggers DNS requests to our test infrastructure.

III. Injection Payloads

In this section we list and explain the payloads, which we use for evaluation of injection vulnerabilities in DNS resolution platforms.

A. Injection Methodology

The evaluation of record injections consists of three phases: seeding, poisoning, and validating. During the seeding phase we plant 'honey' DNS records into caches. Specifically, we cause the DNS resolution platforms to issue queries for records in the 'victim' domain. During the poisoning phase we attempt to replace the authentic records by injecting spoofed records with one of the payloads in Table I. During the validation phase we check whether the poisoning attack succeeded by probing the values of the 'honey' records. During the validation of honey records in payloads 1-14 in Table I we request a new hostname, that is different from the hostname cached during the seed phase. Then we check if the response arrives from the real nameserver or from the attacker's nameserver, and we do not request the NS and A records – which we attempt to overwrite during the poisoning phase – of the nameserver directly. In the other payloads 15-18 in Table I we check if the cache contains the values injected during the poisoning phase – if so, then the attack succeeded. Otherwise, if the records contain values planted during the seeding phase, the attack failed.

These three phases are essential not only for validation of successful cache-injection attacks but they also reflect the real world – the DNS resolution platforms typically contain cached records which the attacker wishes to replace with its own values.

B. Payloads for Overwriting Cached Records

Injecting spoofed records does not guarantee that the resolvers will overwrite the original values of the cached records with the new ones, nor does it guarantee that the resolvers will provide these new values in responses to clients. In particular, each DNS software assigns a different trust level (i.e., rank) to the records that it receives in DNS responses. The higher the rank assigned to the cached records, the more difficult it is to replace their values with new values. For instance, consider pointing a real nameserver to an attacker's IP address, instead

of the authentic IP address of the nameserver. If records in the attacker's response receive lower rank than the cached records, their value will be ignored. DNS resolvers also apply different caching policies when deciding whether to accept and use a record, cache it and if to return it to applications and clients. The ranking of DNS records in responses is discussed in [RFC2181] – this is however interpreted and implemented differently by different DNS software.

In what follows we provide our evaluation of the injection payloads against popular DNS software and against DNS resolution platforms in the wild.

We list the payloads, that we use for overwriting the values of cached records, in Table I. The payloads correspond to different combinations of hostnames and types of records, and sections in DNS responses where the records reside. Each payload has a name and a sequence number in column Test Name. The columns DNS Fields and Values in DNS Fields contain the three sections (`answer`, `authority`, `additional`) in a DNS response, and the records in those sections – these are the values that comprise the (malicious) payloads that are designed to override (authentic) cached records during the *poisoning* phase. The cached records, which the injected payloads attempt to override, appear in column Overrides Cached. These cached records are seeded by us during the *seeding* phase, and they correspond to the different values that can be present in the victim cache during the poisoning phase.

The column Auth Ref. indicates whether a DNS response (during the *seeding* phase) arrives from the authoritative nameserver, namely, contains `authority` and `additional` sections and the `AA` bit is set to 1; e.g., see payloads 5 and 6 in Table I. Authoritative responses are assigned a higher trust level (rank 5) by the resolvers than non-authoritative (rank 3). For instance, cached records from authoritative response cannot be overwritten with records from a non-authoritative response. The Auth Ref. type response is relevant for attacks that attempt to change the value of cached nameserver (NS) records.

In all the payloads we assume that the authentic IP is `a.b.c.d` and that the parent domain is `TAIL`, e.g., `test.example`. We use different subdomains under a `TAIL` domain, as names of DNS records in Figure I. We assume that the attacker uses IP address `ATTACKER`.

The payloads can be categorised according to the injected spoofed records during the poisoning phase. Injected records can be nameserver hostname (NS), nameserver IP address (A), hostnames of other services, such as email with MX records, or IP addresses, such as of web server or email server. Nameserver records can be provided in `authority` and `additional` sections or in an `answer` section. Records of other services are provided in the `answer` section. The payloads can be grossly categorised according to the following classes:

a) Overwrite Any Record Indirectly: we define a new type of poisonous payloads (15-18 in Table I), which we call the *indirect attacks*. In contrast to direct attacks, the indirect attacks are effective after some other record expires. Specifically, we inject into a victim cache a poisonous record which does not immediately impact the resolution process, but becomes effective after an authentic record expires from the cache. Then, the next resolution request to that name will return the spoofed record. The attacker seeds the poisonous record in advance and it is guaranteed that the payload will take effect on the first resolution for the target record after it expires. The indirect record injection allows to attack records which are otherwise difficult to target, e.g., due to a high trust level that may be assigned to the already cached records. To clarify the concept we present the following examples:

(Payload 16. ak1) the attacker injects a `CNAME` record mapping `www.victim.example` to `www.attacker.example`. As long as the victim resolver has a valid `A` record (IP address) for `www.victim.example` it will use it. Once the `A` record expires, the resolver falls back to using the (already cached poisoned) `CNAME` record. Same applies to other records, e.g., injecting a spoofed `NS` record, so that it is requested for resolving a new `MX` record once the previous one expires.

(Payload 17. w11) attacker's goal is to point `www.victim.example` to attacker's address. Assume that the victim DNS resolver has a cached A record for `www.victim.example`:
`www.victim.example A a.b.c.d`
The attacker crafts a malicious record in DNS response: `www.victim.example NS ns.attacker.example`. Notice that `www.victim.example` is not a zone but a hostname pointing at a webserver. Since DNS software uses the most accurate delegation it has in the cache, which in this case is an NS record, the attacker turns it into a zone in the malicious payload that it crafts stating that it has a nameserver `ns.attacker.example`. When the A record with value `a.b.c.d` expires, next time a resolver receives a query for A of `www.victim.example` it will go and ask the *most accurate server* (namely the one that is authoritative for that domain). Hence it will send a query to `ns.attacker.example` (as the nameserver) instead of the nameserver of the parent domain `victim.example` (which has lower precedence). Then, the attacker returns a response which points the webserver to a malicious IP address instead of the real one.

b) Overwrite Glue Records: this category is comprised of payloads that replace cached glue records with spoofed glue records (7,9-11,13,14 in Table I). The payloads override cached glue records (NS or A records of the authentic nameserver) with spoofed glue records (NS or A records). The newly cached glue records are assigned low trust level by the resolvers, e.g., will not be returned to the clients and applications. Notice that overwritten nameserver records can subsequently be used to poison spoofed non-nameserver records, such as applications and services, for instance mail exchanger with MX record, or web server with A record. Specifically, when the cached target victim records, e.g., MX,

expire from the cache, the resolver will query the malicious nameserver (via cached glue A/NS records). Cached glue records can also be used to inject new subdomains, e.g., for phishing attacks.

c) Overwrite Non-Nameserver Records: the payloads in previous categories were designed to override A or NS records of nameservers. In this category, we use a DNAME to override records of any other service, e.g., MX. This category includes payloads 12 and 15.

Consider for example that a victim resolver has a cached record for `www.victim.example`. During the poisoning phase, the attacker injects `victim.example DNAME attacker.example` and `www.attacker.example A ATTACKER`. As a result, when the A record for `www.victim.example` expires from the cache, the queries for `www.victim.example` will be resolved to `www.attacker.example`.

IV. Internet Measurement of Payloads Injection

In this section we present the results of our evaluation of the payloads injection against our dataset of networks. We explain our evaluation against DNS resolution platforms with a single cache, then in Section IV-C we extend our study to multiple caches.

In the first step, we use our infrastructure to evaluate known DNS caching resolvers – those which DNS software and configuration is known to us before the evaluation, and then report on our study of unknown caching resolvers. We evaluate our tool against known and popular DNS caches, hence creating a 'caching behaviour fingerprint' of those DNS servers. Using the fingerprints we can then identify and characterise the caches in DNS resolution platforms during our Internet measurements.

A. Methodology

Our study is performed against DNS resolution platforms in each of the three datasets that we collected (see Section II). The evaluation is initiated by a prober either in set-up 1 or in set-up 2 (Figure 1) depending on whether we have a direct access to the ingress DNS resolver or not. The prober generates DNS requests to the ingress resolver for records within our domains hosted on our test infrastructure. Then, the communication with our nameservers is carried out by the egress resolver. The evaluation keeps the egress resolver in resolution iteration by utilising standard DNS behaviour, e.g., by redirecting it with referral responses to our subdomains. During the evaluation we check the 18 payloads in Table I.

The evaluation consists of three phases, as described in Section III-A. For each of the 18 payloads, we perform the three phases. During the seed phase we plant a record into the tested cache (Table I column Overrides Cached). Then, during the poison phase we override the cached record with a record in column Values in DNS Fields. We then check the success by requesting the 'honey record' (i.e., the record that indicates whether the poisoning was successful and replaced the value of the seed).

Handling Packet Loss: Packet loss introduce noise into the test results, and may make successful tests appear as failed, for instance, if the poisonous payload is lost and hence not cached, the cache will contain the original value of the 'honey' record. Most networks exhibit a typical packet loss of 1% while others, such as Iran, have almost 12% packets' loss.

To cope with packet loss we evaluate poisoning of each payload in Table I against a target DNS resolution platform k times (with k different queries). The parameter k should be a function of a packet loss in the measured network. To ensure accuracy of our evaluation we used $k = 10$, i.e., repeating each iteration (seed $\rightarrow$ poison $\rightarrow$ validate) for each payload 10 times.

B. Evaluation of DNS Platforms

a) Known DNS Caches: Table 2 summarises the results of our evaluation of the DNS resolution platforms with the popular caches. These include caches of public DNS providers such as Google public DNS, as well as networks with open source, or proprietary DNS software, and appliances. The first column corresponds to the name and sequence number of the payload in Table I. Subsequent columns show for each DNS software (and version) which of the tests it was found vulnerable to – namely, the evaluation resulted in successful injection of payload into its cache. MaraDNS as well as Nominum exhibit almost identical caching behaviour, and are vulnerable to only two of the payloads.

The fingerprinting of the software or cache type is done based on caching strategies and cache overwriting behaviour using the payloads in Table I. The fingerprints enable us to approximate the fraction of the caches in the Internet that match our models reflecting the types of caching behaviour.

b) Unknown DNS Software: We then apply our evaluation over the DNS platforms in the dataset of networks from Section II. Evaluation against unknown caches allows to identify the fraction of vulnerable caches even without having a fingerprint of its software; namely caches we have not 'seen' before. We list the results for our three networks populations in Figure 3. All the DNS software were found vulnerable to our *indirect attacks*, except for payload 15 (dname) – some resolvers, such as google public DNS, do not cache/support DNAME records. Furthermore, tests (3,4,13,14) in Figure 3, cannot be performed with SMTP email servers, since the tests require an A record query.

Figure 3 shows that the networks operating open resolvers contain DNS platforms with most vulnerabilities – over 97% of the networks are vulnerable to at least one payload. This is typically due to direct communication with the ingress DNS resolvers which provides better control of the timing and the values of queries' names and queries' types, in contrast to Email or web browsers.

The ISP networks evaluated via ad-network provide lowest success rate, i.e., a bit less than 70% of the networks are vulnerable to one or more injection payloads. The reason for lower success is that many of the tests are interrupted in the middle of the evaluation by the clients or due to timeout in

Test Name	DNS Fields	Values in DNS Fields	Overrides Cached	Auth Ref.	Direct	Defence
1. NS0	Q An Au Ad	A? two.test-ns0.TAIL two.test-ns0.TAIL A a.b.c.d test-ns0.TAIL NS ns2.test-ns0.TAIL ns2.test-ns0.TAIL A ATTACKER	test-ns0.TAIL NS ns.test-ns0.TAIL	No	No	(A)/(B)
2. NS0-auth	Q An Au Ad	A? two.test-ns0-auth.TAIL two.test-ns0-auth.TAIL A a.b.c.d test-ns0-auth.TAIL NS ns2.test-ns0-auth.TAIL ns2.test-ns0-auth.TAIL A ATTACKER	test-ns0-auth.TAIL NS ns.test-ns0-auth.TAIL	Yes	No	(A)/(B)
3. NS	Q An Au Ad	A? ns2.test-ns.TAIL ns2.test-ns.TAIL A ATTACKER test-ns.TAIL NS ns2.test-ns.TAIL —	test-ns.TAIL NS ns.test-ns.TAIL	No	No	(A)
4. NS-auth	Q An Au Ad	A? ns2.test-ns-auth.TAIL ns2.test-ns-auth.TAIL A ATTACKER test-ns-auth.TAIL NS ns2.test-ns-auth.TAIL —	test-ns-auth.TAIL NS ns.test-ns-auth.TAIL	Yes	No	(A)
5. NS2	Q An Au Ad	A? two.test-ns2.TAIL two.test-ns2.TAIL A a.b.c.d test-ns2.TAIL NS ns2.magic-ns2.TAIL —	test-ns2.TAIL NS ns.test-ns2.TAIL	No	No	(A)
6. NS2-auth	Q An Au Ad	A? two.test-ns2-auth.TAIL two.test-ns2-auth.TAIL A a.b.c.d test-ns2-auth.TAIL NS ns2.magic-ns2-auth.TAIL —	test-ns2-auth.TAIL NS ns.test-ns2-auth.TAIL	Yes	No	(A)
7. b4	Q An Au Ad	A? two.test-b4.TAIL — sub.test-b4.TAIL NS ns.test-b4.TAIL ns.test-b4.TAIL A ATTACKER	ns.test-b4.TAIL A a.b.c.d	N/A	No	(B)
8. u1-auth	Q An Au Ad	A? two.test-u1-auth.TAIL — test-u1-auth.TAIL NS ns2.test-u1-auth.TAIL ns2.test-u1-auth.TAIL A ATTACKER	test-u1-auth.TAIL NS ns.test-u1-auth.TAIL	Yes	No	(A)/(B)
9. u3-2	Q An Au Ad	A? two.test-u3-2.TAIL two.test-u3-2.TAIL A a.b.c.d test-u3-2.TAIL NS ns.test-u3-2.TAIL ns.test-u3-2.TAIL A ATTACKER	ns.test-u3-2.TAIL A a.b.c.d	N/A	No	(B)
10. u3-3	Q An Au Ad	A? two.test-u3-3.TAIL — test-u3-3.TAIL NS ns.test-u3-3.TAIL ns.test-u3-3.TAIL A ATTACKER	ns.test-u3-3.TAIL A a.b.c.d	N/A	No	(B)
11. u3-4	Q An Au Ad	A? two.sub.test-u3-4.TAIL — sub.test-u3-4.TAIL NS ns.test-u3-4.TAIL ns.test-u3-4.TAIL A ATTACKER	ns.test-u3-4.TAIL A a.b.c.d	N/A	No	(B)
12. w-dname	Q An Au Ad	A? two.test-w-dname.TAIL test-w-dname.TAIL DNAME magic-w-dname.TAIL — —	(all).test-w-dname.TAIL All types	N/A	No	no DNAME from cache
13. w7	Q An Au/Ad	A? two.test-w7.TAIL two.test-w7.TAIL CNAME ns.test-w7.TAIL; ns.test-w7.TAIL A ATTACKER —	ns.test-w7.TAIL A a.b.c.d	N/A	No	[break] CNAME chain
14. w8	Q An Au/Ad	A? ns.sub.test-w8.TAIL sub.test-w8.TAIL DNAME test-w8.TAIL; ns.test-w8.TAIL A ATTACKER —	ns.test-w8.TAIL A a.b.c.d	N/A	No	[break] DNAME chain
15. dname	Q An Au Ad	A? zwei.test-dname.TAIL test-dname.TAIL DNAME magic-dname.TAIL — —	(all).test-dname.TAIL ALL types	N/A	Yes	no DNAME from cache
16. ak1	Q An Au/Ad	A? zwei1.test-ak1.TAIL zwei1.test-ak1.TAIL CNAME one1.test-ak1.TAIL; one1.test-ak1.TAIL CNAME one1.magic-ak1.TAIL —	one1.test-ak1.TAIL ALL TYPES	N/A	Yes	[break] CNAME chain
17. w11	Q An Au Ad	A? zwei.one1.test-w11.TAIL — one1.test-w11.TAIL NS ns2.magic-w11.TAIL —	one1.test-w11.TAIL NS ANY	N/A	Yes	(A)
18. w11bis	Q An Au Ad	A? zwei.one1.test-w11bis.TAIL — one1.test-w11bis.TAIL NS ns2.test-w11bis.TAIL ns2.test-w11bis.TAIL A ATTACKER	one1.test-w11bis.TAIL NS ANY	N/A	Yes	(A)/(B)

TABLE I
PAYLOADS FOR DNS RECORDS' INJECTION. DEFENCES LEGEND: (A) AUTHORITY QUERY FOR NS AFTER REFERRAL; (B) AUTHORITY QUERIES FOR NAMESERVER IP ADDRESSES.

the browser; for instance, Chrome 48 times-out long before the tests are completed (due to a bug which was later fixed in Chrome 49).

The results for SMTP are slightly better than for ad-networks. In this set-up, as is the case with browser based evaluation, indirect access introduces noise and higher failure rates. In particular, local caches (of browsers and operating systems), as well as lack of control of the timing of the queries, make the evaluation more difficult.

Despite the higher failure rates among ISPs (with ad-nets) and enterprises (with SMTP servers), the fraction of networks that were found to be vulnerable to at least one injection payload is alarmingly high, and the majority of the networks are vulnerable to at least one injection payload. The combined

	Name	BIND 9.10.2-P2	BIND 9.4.1	Unbound 1.5.4	MaraDNS 3.2.07 Deadwood	PowerDNS 3.7.3	MS DNS 6.1 Win Server'08 R2 6.1.7601)	MS DNS 6.2 Win Server'12 6.2.9200	MS DNS 6.3 Win Server'12 R2 6.3.9600	Google Public DNS	Open DNS	BIND 9.10.2-P2 w/DNSSEC	Nominum Vantio CacheServe v5	Nominum Vantio CacheServe v7
1	ns0	no	yes	yes	no	yes	yes	yes	yes	no	yes	no	no	no
2	ns0-auth	no	yes	yes	no	yes	no	no	no	no	no	no	no	no
3	ns	yes	yes	yes	no	yes	yes	yes	yes	no	yes	no	no	no
4	ns-auth	yes	yes	yes	no	yes	no	no	no	no	no	no	no	no
5	ns2	yes	yes	yes	no	yes	yes	yes	yes	no	yes	no	no	no
6	ns2-auth	yes	yes	yes	no	yes	no	no	no	no	no	no	no	no
7	b4	no	no	no	no	yes	yes	yes	yes	no	yes	no	no	no
8	u1-auth	no	no	yes	no	yes	no	no	no	no	no	no	no	no
9	u3-2	no	no	yes	no	yes	yes	yes	yes	no	yes	no	no	no
10	u3-3	no	no	yes	no	yes	yes	yes	yes	no	no	no	no	no
11	u3-4	yes	yes	yes	no	yes	yes	yes	yes	no	yes	yes	no	no
12	w-dname	yes	yes	no	no	no	no	yes	yes	no	no	yes	no	no
13	w7	no	no	no	no	yes	yes	yes	yes	no	yes	no	no	no
14	w8	yes	yes	no	no	yes	yes	yes	yes	no	yes	no	no	no
15	dname	yes	yes	no	no	no	no	yes	yes	no	no	yes	no	no
16	ak1	yes	yes	no	no	yes	yes	yes	yes	yes	yes	no	no	no
17	w11	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
18	w11bis	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes

Fig. 2. Evaluation of the cache poisoning payloads against popular DNS software and public DNS services.

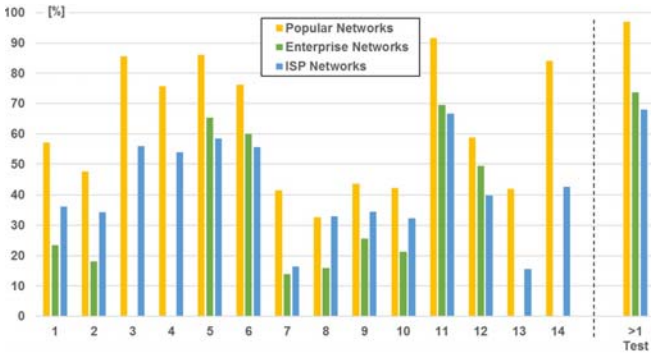

Fig. 3. Evaluation of payloads injections against networks in our dataset.

tests (including indirect attacks) across all network populations exhibit more than 92% success rate.

The DNS platforms typically more than a single cache, we next explain how our evaluation handles multiple caches.

C. Extension to Multiple Caches

Often DNS resolution platforms use more than one cache, e.g., for efficiency. We show how to extend our study to evaluate DNS platforms with multiple caches. The main idea behind multiple caches is using multiple number of DNS requests that would ensure probing all the caches used by a given DNS resolution platform.

When a resolution platform has more than one cache we cannot guarantee that all the payloads will be evaluated in sequence against the same cache. In particular, often the queries will be distributed among different caches, hence, it can often happen that the seed would be placed in one cache, and the malicious payload would be directed to another cache. To cope with multiple caches, instead of having the same cache handle the attack cycle (seed $\rightarrow$ poison $\rightarrow$ validate), we "bombard" with (N) multiple "parallel" steps of (seed $\rightarrow$ poison $\rightarrow$ check). Namely, during the seeding phase we send N seeds: $s_1, s_2, ..., s_N$. Then, during the poisoning phase we send N instances of the same payload: $p_1, ..., p_N$. Finally, we initiate the validate phase requesting the 'honey' records: $c_1, ..., c_N$.

A prerequisite is that N (number of copies in the attack) is larger than n (number of caches). Specifically, the expected part of the n caches that is not covered in N attempts is roughly $exp(-\frac{N}{n})$.

Overall, if the target DNS resolver is vulnerable, in the last phase we expect success rate of $N \cdot (1 - exp(-\frac{N}{n}))^2$; as $\frac{N}{n}$ grows, this asymptotically reaches N.

Our measurements show that typically the resolution platforms in the Internet use more than a single IP address and more than a single cache. In Figure 4 we plot our measurements for number of caches supported by resolution platforms in the three common networks' populations that we studied. The networks running open resolvers use the least number of caches, 70% have 1 to 2 caches per IP address. About 60% of DNS platforms operated by ISPs use 1-3 caches, and 65% of networks measured via email servers use 1-4 caches per ingress IP address.

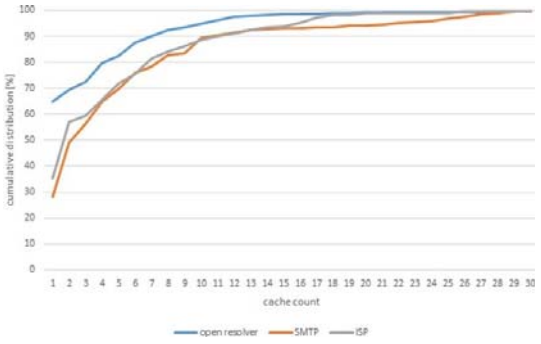

Fig. 4. The number of caches supported by resolution platforms.

The results in Figures 6 and 7 provide the distribution of the number of ingress IP addresses vs. caches for networks operating open resolvers, enterprise networks and ISPs. The

circles' area corresponds to the number of measured networks that fall within that set, i.e., the larger the circle is the more networks fall within that set. The center of the circle corresponds to the (x, y) coordinate on the graph. The majority of the networks with open resolvers have similar properties: use 1 ingress IP address and 1 cache – this corresponds to the largest circle in Figure 7. Specifically, as Figure 5 shows, almost 70% of the networks with open resolvers use DNS resolution platforms with one IP address and one cache.

Smaller circles on y axis show that many other networks have less than 10 IP addresses. On the other hand, very few networks also use more than 500 IP addresses, with more than 30 caches (top right circles in Figure 7).

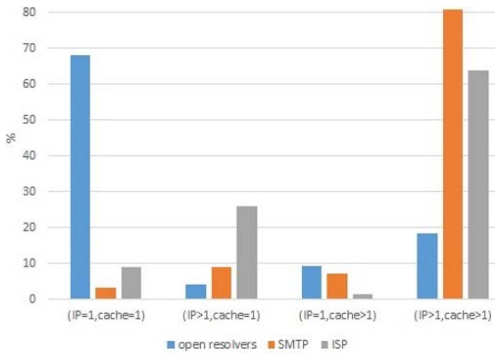

Fig. 5. IP addresses vs caches count across three networks populations.

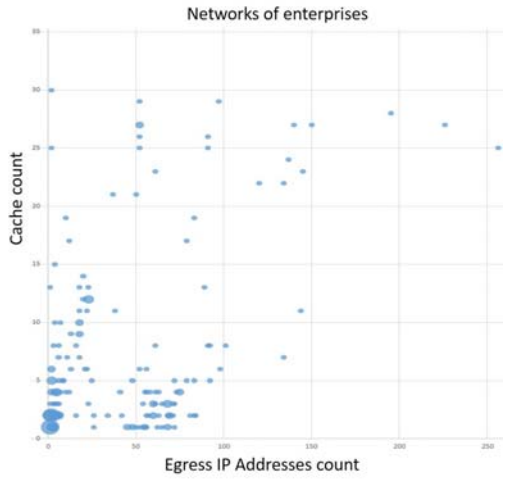

Fig. 6. IP addresses vs caches count in SMTP population.

In contrast, the results for enterprise networks and networks of ISPs are more scattered, with a more even distribution and significantly less IP addresses. ISP networks appear to use least caches and have the smallest number of IP addresses. Less than 10% of ISP networks and less than 5% of enterprises use a single address and cache, see Figure 5. The majority of ISPs and of enterprise networks use more than one address and more than one cache (almost 65% of IPSs and more than 80% of enterprises).

In this section we analyse the upper bound on the number of DNS requests, needed to enumerate the caches.

Assume that each cache, out of n caches, is equally likely to be selected as a candidate for a given query. In each iteration exactly one cache is probed, and the experiment has to be repeated until each of n caches has been probed at least once. The cache selection is an independent random variable, and a cache i is selected with probability $p_i = \frac{1}{n}$. We then consider the following problem: *What is the expected number of queries q that needs to be issued in order to probe each of n caches?*

Theorem 4.1: Let X be the random number of queries that need to be issued in order to probe all n caches, such that $X = X_1 + ... + X_n$ and each X_i $(\forall 0 < i \leq n)$ denotes the number of queries required to probe cache i, after probing cache $i-1$. Then, the expected number of queries $q = E(X)$ to probe all n caches is: $E(X) = E(X_1) + ... + E(X_n) = \frac{1}{p_1} + ... + \frac{1}{p_n} = \frac{n}{n} + \frac{n}{n-1} + ... + \frac{n}{1} = n \times (\frac{1}{1} + \frac{1}{2} + ... + \frac{1}{n}) = n \times \Sigma_{i=1}^{n} \frac{1}{i} = n \times H_n = n(\log n + \Theta(1)) = n \log n + O(n) = \Theta(n \log n)$.

Proof. The first cache is probed with first query, i.e., $X_1 = 1$. The probability to probe cache $i+1$, after probing i caches for $i \in \{1, ..., n\}$, is $\frac{n-i}{n}$. Since the caches are selected with uniform probability X_i has a geometric distribution with parameter $\frac{n-i+1}{n}$. By linearity of expectations, we obtain that the expected number of queries to probe all the caches is $E(X) = E(X_1) + ... + E(X_n) = n \times \Sigma_{i=1}^{n} \frac{1}{i} = n \cdot H_n$, where H_n is the harmonic series, $H_n = \Sigma_{i=1}^{n} \frac{1}{i}$. For $n \to \infty$, the series converges to $H_n = \log n + \epsilon + \frac{1}{2n} + O(\frac{1}{n^2})$. Hence $E(X) = n \log n + n\epsilon + \frac{1}{2} + O(\frac{1}{n})$. We obtain $E(X) = n \log n + O(n) = \Theta(n \log n)$. ∎

V. Conclusions

Our study of the injection vulnerabilities in DNS resolution platforms revealed significant security problems. We showed that the vast majority of the platforms – including those operated by large ISPs and enterprises – are vulnerable: DNS caches can be poisoned persistently, with little effort and in a stealthy way. This represents a huge risk for the Internet overall and for systems and applications relying on DNS.

Obviously, it is important to raise awareness for these facts and motivate adoption of countermeasures. Our study serves as an important guidance for DNS software vendors and operators for design and configuration of secure DNS platforms. We provide recommendations for defences in Table I column Defences. Patching the caches would even in some cases provide defences against Man-in-the-Middle attackers.

In our study we focus on IPv4 DNS resolvers, and use queries IPv4 (A) resource record (RR) addresses, as well as payloads only with IPv4 addresses. Extending our study to mixed (IPv6 and IPv4) environments may pose an interesting challenge and opportunities, and we leave it for future work.

VI. Acknowledgements

We thank Amir Herzberg and Stephane Bortzmeyer for their helpful comments on our manuscript. The research reported

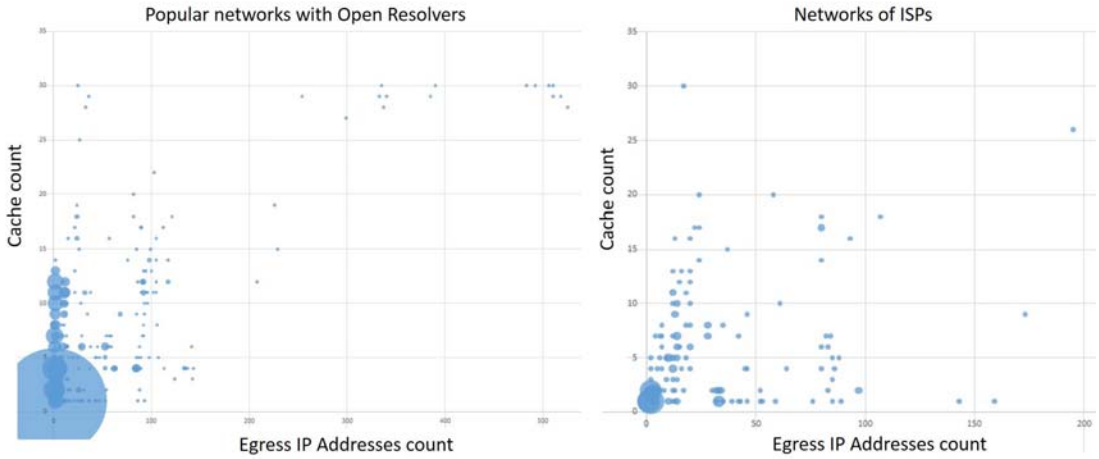

Fig. 7. IP addresses vs caches count in networks with open resolvers (left) and ISP networks (right).

in this paper has been supported in part by the German Federal Ministry of Education and Research (BMBF) and by the Hessian Ministry of Science and the Arts within CRISP (www.crisp-da.de/). We are grateful to Microsoft Azure Research Award, which enabled us to host our infrastructure on Azure platform.

REFERENCES

[1] J. Stewart, "Dns cache poisoning–the next generation," 2003.

[2] D. Kaminsky, "It's the End of the Cache As We Know It," in *Black Hat conference*, August 2008, http://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf.

[3] A. Herzberg and H. Shulman, "Security of patched DNS," in *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, 2012, pp. 271–288.

[4] H. Shulman and M. Waidner, "Towards security of internet naming infrastructure," in *European Symposium on Research in Computer Security*. Springer, 2015, pp. 3–22.

[5] ——, "Fragmentation Considered Leaking: Port Inference for DNS Poisoning," in *Applied Cryptography and Network Security (ACNS), Lausanne, Switzerland*. Springer, 2014.

[6] A. Herzberg and H. Shulman, "Vulnerable delegation of DNS resolution," in *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, 2013, pp. 219–236. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40203-6_13

[7] ——, "Socket Overloading for Fun and Cache Poisoning," in *ACM Annual Computer Security Applications Conference (ACM ACSAC), New Orleans, Louisiana, U.S.*, C. N. P. Jr., Ed., December 2013.

[8] ——, "Fragmentation Considered Poisonous: or one-domain-to-rule-them-all.org," in *IEEE CNS 2013. The Conference on Communications and Network Security, Washington, D.C., U.S.* IEEE, 2013.

[9] D. Anderson, "Splinternet behind the great firewall of china," *Queue*, vol. 10, no. 11, p. 40, 2012.

[10] M. Hu, "Taxonomy of the snowden disclosures," *Wash & Lee L. Rev.*, vol. 72, pp. 1679–1989, 2015.

[11] P. Levis, "The collateral damage of internet censorship by dns injection," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 3, 2012.

[12] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman, "Assessing dns vulnerability to record injection," in *Passive and Active Measurement*. Springer, 2014, pp. 214–223.

[13] M. Wander, C. Boelmann, L. Schwittmann, and T. Weis, "Measurement of globally visible dns injection," *Access, IEEE*, vol. 2, pp. 526–536, 2014.

[14] A. Borgwart, S. Boukoros, H. Shulman, C. van Rooyen, and M. Waidner, "Detection and forensics of domains hijacking," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[15] M. Ben-Yossef, H. Shulman, M. Waidner, and G. Beniamini, "Factoring DNSSEC: Evaluation of Vulnerabilities in Signed Domains," in *NSDI*, 2016.

[16] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "Dns performance and the effectiveness of caching," *Networking, IEEE/ACM Transactions on*, vol. 10, no. 5, pp. 589–603, 2002.

[17] D. Wessels, M. Fomenkov, N. Brownlee, and K. Claffy, "Measurements and laboratory simulations of the upper dns hierarchy," *Passive and Active Network Measurement*, pp. 147–157, 2004.

[18] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman, "On measuring the client-side dns infrastructure," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 77–90.

[19] D. Dagon, N. Provos, C. P. Lee, and W. Lee, "Corrupted dns resolution paths: The rise of a malicious resolution authority." in *NDSS*, 2008.

[20] D. Leonard and D. Loguinov, "Demystifying service discovery: implementing an internet-wide scanner," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 109–122.

[21] C. A. Shue and A. J. Kalafut, "Resolvers revealed: Characterizing dns resolvers and their clients," *ACM Transactions on Internet Technology (TOIT)*, vol. 12, no. 4, p. 14, 2013.

[22] S. Son and V. Shmatikov, "The hitchhikers guide to dns cache poisoning," in *Security and Privacy in Communication Networks*. Springer, 2010, pp. 466–483.

[23] W. Wijngaards, "Resolver side mitigations," 2009.

[24] A. Klein, "BIND 9 DNS cache poisoning," Trusteer, Ltd., 3 Hayetzira Street, Ramat Gan 52521, Israel, Report, 2007.

[25] M. Kührer, T. Hupperich, J. Bushart, C. Rossow, and T. Holz, "Going wild: Large-scale classification of open dns resolvers," in *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*. ACM, 2015, pp. 355–368.

[26] J. Zhang, Z. Durumeric, M. Bailey, M. Liu, and M. Karir, "On the mismanagement and maliciousness of networks," in *to appear) Proceedings of the 21st Annual Network & Distributed System Security Symposium (NDSS14), San Diego, California, USA*, 2014.

[27] J. Mauch, "Open resolver project," in *Presentation, DNS-OARC Spring 2013 Workshop (Dublin)*, 2013.

[28] A. Herzberg, "DNS-based email sender authentication mechanisms: A critical review," *Computers & Security*, vol. 28, no. 8, pp. 731–742, 2009.