NXNSAttack: Recursive DNS Inefficiencies and Vulnerabilities

Lior Shafir
Tel Aviv University
lior.shafir@gmail.com

Yehuda Afek
Tel Aviv University
afek@post.tau.ac.il

Anat Bremler-Barr
The Interdisciplinary Center
bremler@idc.ac.il

Abstract

The Domain Name System (DNS) infrastructure, a most critical system the Internet depends on, has recently been the target for different DDoS and other cyber-attacks, e.g., the notorious Mirai botnet. While these attacks can be destructive to both recursive and authoritative DNS servers, little is known about how recursive resolvers operate under such attacks (e.g., NXDomain, water-torture). In this paper, we point out a new vulnerability and show an attack, the *NXNSAttack*, that exploits the way DNS recursive resolvers operate when receiving NS referral response that contains name-servers but without their corresponding IP addresses (i.e., missing glue-records). We show that the number of DNS messages exchanged in a typical resolution process might be much higher in practice than what is expected in theory, mainly due to a proactive resolution of name-servers' IP addresses. We show how this inefficiency becomes a bottleneck and might be used to mount a devastating attack against either or both, recursive resolvers and authoritative servers. The *NXNSAttack* is more effective than the NXDomain attack: i) It reaches an amplification factor of more than 1620x on the number of packets exchanged by the recursive resolver. ii) Besides the negative cache, the attack also saturates the 'NS' resolver caches. In an attempt to mitigate the attack impact, we propose enhancements to the recursive resolvers algorithm to prevent unnecessary proactive fetches. Finally, we implement our Max1Fetch enhancement on the BIND resolver and show that Max1Fetch does not degrade the recursive resolvers performance, throughput and latency, by testing it on real-world traffic data-sets.

1 Introduction

The Domain Name System (DNS) infrastructure, a most critical highly dynamic system on which almost any access to a resource on the Internet depends, has recently been an attractive target to a variety of DDoS attacks [4, 26]. As seen in the Mirai attack [4], a degradation or outage of part of the DNS service disrupts many popular websites such as Twitter, Reddit, Netflix, and many others, impacting millions of Internet users. Moreover, recent large scale attacks were directly trying to take down parts of the DNS system by flooding the DNS servers with well-structured requests of randomly-generated non-existent sub-domains known as NXDomain attacks [28] (or *water-torture* attack [16, 26], or *PRSD attack*).

These attacks can be destructive to both recursive resolvers and authoritative DNS servers. While authoritative servers are usually the prime target of such attacks [3, 4, 16, 22], the damages are often not limited to the target itself but involve severe collateral damages to recursive resolvers as well. Indeed, it has been reported [24, 26] that ISP recursive resolvers were knocked down during such attacks (and more such cases go unreported). Still, little is known about how recursive resolvers operate under NXDomain and similar complexity attacks.

This paper focuses on recursive resolvers and has three major parts. First, we perform an in-depth analysis of the complex DNS recursive behavior and the interaction between its various algorithms and components using the popular BIND [11] server implementation, to expose their inefficiencies. Secondly, we uncover a new vulnerability in their algorithms, and display a new attack, the *NXNSAttack* that exploits the way DNS recursive resolvers operate. Finally, we suggest and analyze a modification to the recursive resolver algorithms, called Max1Fetch, that drastically reduces the effectiveness of the *NXNSAttack* attack. Notice that investigating recursive resolvers and measuring their performance is challenging due to their interaction with many live name servers.

At an abstract level, the DNS system has two parts, each of which is a highly distributed large system around the globe: a hierarchical and dynamic database of *authoritative* name-servers storing the DNS data, and a huge

number of client-facing *resolvers*, located either locally at the service providers and local organizations, or as cloud public services (e.g., CloudFlare 1.1.1.1), that query the hierarchical structure to provide domain names resolutions to IP addresses. The focus of the current paper is on the interaction between the recursive resolvers and the authoritative hierarchical structure.

During the resolution process, the recursive resolver walks through the authoritative hierarchy to obtain the definitive answer for its request. This iterative walk usually involves delegations from one authoritative zone to another. The delegation messages are called name servers (NS) referral responses. In this delegation message, an authoritative server tells the recursive resolver: "I do not know the answer, go and query these and these name servers, e.g., ns1, ns2, etc., instead". One of our main observations is that the information in the NS referral responses, at the different recursive steps, and the actions taken by the recursive resolvers as a result, may introduce large undesired overheads.

The main reason for these overheads is that the name-servers in the NS referral response are not always provided with their corresponding IP addresses (known as glue records). Top-level authoritative domains (TLDs), second-level domains (SLDs), and other authoritative servers are not allowed to provide IP addresses for domains that do not reside in the same zone origin (known as *Out-of-Bailiwick* name-servers [2]). This is mostly to protect from DNS poisoning attacks.

We study the implications and prevalence of this phenomenon. Our first observation (given in §2) is that the number of packets involved in a typical resolution process is much higher in practice than expected in theory, mainly due to proactive extra resolutions of name-servers IP Addresses.

Secondly, we show how the proactive resolution of *all* the name servers in the referral response becomes a major bottleneck in recursive servers such as BIND. BIND is considered as the de-facto standard for DNS software. We present a new attack, called *NXNSAttack* (§3), that exploits this vulnerability and is more effective against authoritative and recursive servers than the NXDomain attack (§4.5). We show three variants of this attack (*a*, *b*, and *c* in Table 1) analyzing their impact on a BIND based recursive resolver and authoritative servers. (§4). The NXNSAttack simulations saturate the recursive resolver's cache (with NX & NS records) and reach a packet amplification factor (PAF) of more than 1600x (variant *a*). The key enabler for the attack is the ease with which an attacker can acquire and control an authoritative server.

Thirdly, we enhance the BIND DNS resolver algorithms to remove unnecessary proactive fetches (§5), thus alleviating the vulnerability and measure the performance improvements of the enhancement, called Max1Fetch. In particular, we show it has no negative impact on either the latency or throughput of the enhanced recursive resolver.

		Attack target (victim)	Max Amplification factor	
			Bytes	Packets
NXDomain Attack (Mirai [4])		Authoritative name-server	2.6x	2x
NXNSAttack	a	Recursive resolver	163x	1621x
	b	Authoritative SLD	21x	75x
	c	Root / TLD	99x	1071x

Table 1: Three variants of the NXNSAttack, and the NXDomain attack [28] empirical evaluation under BIND 9.12.3

Finally, (§6), we quantify the pervasiveness of domains with *out-of-bailiwick* name-servers in: (i) the top million domains resolutions, and (ii) in a real-life DNS traffic trace observed at our campus. Since the inefficiency and vulnerability we uncover are associated mostly with referral responses that contain many name-servers without an associated IP address, we study how common is this phenomena. We notice that in 60% of the domains, all their name-servers are *out-of-bailiwick* name-servers.

Related work is given in §7, Disclosure procedure in §8 and conclusions are provided in §9.

2 Background: DNS Resolution Process Overhead

The DNS infrastructure consists mostly of two types of servers, authoritatives and resolvers, from each of which there are millions of servers spread all over the world. The DNS directory itself is stored in the authoritative name-servers, which are organized in a delegated hierarchical structure. These servers are authorized to provide the DNS data (including IP addresses) for a specific zone without performing requests to other DNS servers.

In contrast, DNS resolvers perform the DNS resolution on behalf of clients that query them, to translate a domain name to its corresponding current IP address. Recursive resolvers (located at ISPs, large organizations, or public cloud resolvers) perform a full DNS resolution by sending queries to the authoritative hierarchical structure.

Recursive resolvers (and stub-resolvers) use a cache to significantly reduce the number of requests that reach the authoritative hierarchy. A TTL is associated with each record in the cache, indicating for how long this entry is valid. The TTL is provided by the authoritative that sent the corresponding response. The cache records are also labeled by the type of query/response

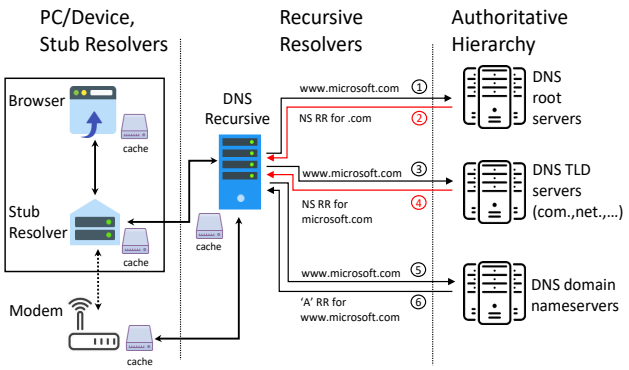


Figure 1: The resolution process, in theory, for the domain www.microsoft.com. The red steps represent NS referral responses.

that produced them: A, AAAA, NS, or NX corresponding to the IPv4 address of a particular host, IPv6 address, the authoritative name-servers for the domain or zone, a domain name that does not exist in the appropriate authoritative server, respectively.

2.1 The Resolution Process: In Theory

When an answer is not found in the recursive resolver's cache, it walks through the authoritative hierarchy to obtain the answer as shown in Figure 1, where the recursive resolver resolves the domain name `www.microsoft.com`, assuming that the recursive resolver's cache is empty; thus, it issues a query to one of the root servers (e.g., A.ROOT-SERVERS.NET, whose IP address is hard-coded into the recursive resolver), asking for the address of `www.microsoft.com` (see step 1 in Figure 1). The root server returns an NS referral response delegating the query to one of few TLD (Top Level Domain) name servers, that are responsible for the '.com' zone (step 2). The recursive resolver selects one of these name servers and issues another query (step 3) asking the chosen TLD name server (assuming it has its IP address) for the address of `www.microsoft.com`. The TLD (.com) server returns an NS referral response (step 4) with the names of a few SLD (Secondary Level Domain) name-servers responsible for the zone 'microsoft.com'. In a similar way to the previous iteration, the recursive resolver selects one of these name servers and issues another query asking for the address of `www.microsoft.com` (step 5). The SLD authoritative server owns the DNS records for all the domains under 'microsoft.com', thus returns an 'A' response with the requested IP address (step 6).

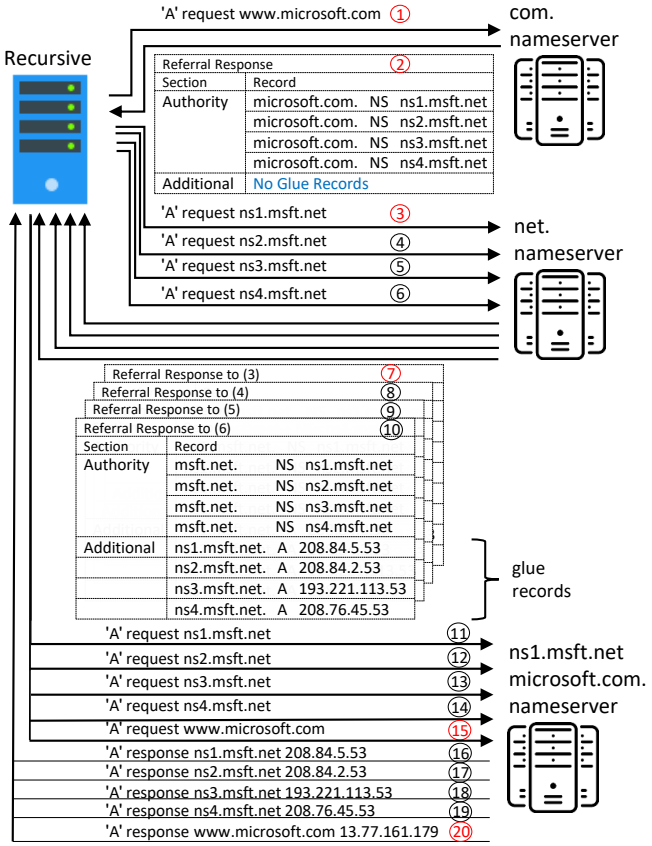


Figure 2: The resolution process, in practice, for the domain www.microsoft.com by BIND 9.12.3 recursive resolver (parallels the diagram in Fig. 1). The .net and .com TLD nameservers are already cached at the beginning of the process. The red steps are the mandatory messages required to answer the client query.

2.2 The Resolution Process: In Practice

One of our main observations is that the information in the NS referral response at the different recursive steps (appear as responses (2), (4) in Figure 1) and the actions taken by the recursive resolvers in response, may introduce large undesired overheads.

We analyze thousands of resolutions taken from real-life scenarios, popular websites, and campus DNS data, inspecting the type and number of packets involved in each resolution. Our focus is on the interaction between the recursive resolvers and the authoritative hierarchical structure. We performed a test on a recursive resolver using BIND 9.12.3 installed on an AWS EC2 machine, as well as on a local machine to inspect the code and to analyze its internal components and algorithms. We found out that while the procedure described in Figure 1, results in a total of three requests and replies (if relevant

information is not in the cache), in real life results in many more messages (see the procedure described in Figure 2), sometimes hundreds, even if the cache is already filled by many previous but different requests. For example, `microsoft.com` resolution requires 54 IPv4 packets (actually there are 126 packets involved, but we exclude TCP handshake and control packets that are used in the case that responses are too long due to additional records and EDNS), `twitter.com` resolution requires 388 packets and `www.gov.uk` requires 102 packets.

In §5.3 we describe the data-sets that we use and provide more details on how many queries resolutions incur more messages than the one expected in theory. For example, we show that 61.56% of the 'A' queries for the top million domains introduce such overhead.

The main reason for these extra messages is that the recursive resolver initiates new independent fetches (additional 'A' resolutions) to resolve *all* the IP addresses of *out-of-bailiwick* (this term is explained in the sequel) name servers it received in the NS referral response. These additional resolutions are not considered in the theoretical procedure as depicted in Figure 1, but often occur in practice (see Figure 2).

At a high level, for each client query, BIND resolver maintains a recursion state to keep track of the current resolution level in the hierarchical tree. Upon receiving an NS referral response, in order to select the next name-server to which it sends its query, the resolver walks through each one of the name-servers it received in the referral response, and checks whether it has its IP address in its local cache.

If not present in the cache, either the resolver starts a new resolution recursively, to resolve the names of the name-servers in the list, or the NS referral response includes the IP addresses of the NS's, which are called *glue records*, eliminating the need for the additional recursive resolution. Each such recursive resolution is independent of the original client query, and new resolution states are maintained for it.

This procedure results in pro-active resolutions for *all* the non-cached name-servers that appear in the NS referral without a corresponding glue-record, and is not part of the configuration nor can be disabled.

Name Server Selection. The NS referral records that appear as the responses (2), (4) in the example in Figure 1 are used to list all the name-servers that are delegated to a child zone (i.e., any of these name-servers can provide the next level in the recursive resolution process). The DNS standard [RFC 1035] requires a minimum of two authoritative name servers for every operational zone for redundancy. However, in practice, service operators at all levels often use more than two name-servers (see §6) that are physically distributed over multiple servers using anycast and load-balancing techniques.

The resolver has to select to which name-server from the list it sends the query. Most of the recursive resolvers implementations use algorithms to distribute the load among the different name-servers and achieve lower latency over time when sending queries to authoritative name servers. For example, BIND uses an sRTT (smoothed Round Trip Time) algorithm with decaying factor, in which it tracks the response time of each name server. Other algorithms perform an initial round-robin over the name servers followed by measured latency-aware selections.

Glue Records and the Bailiwick rule. As mentioned before, the name-servers in an NS referral message are specified by their domain names. To indicate the location of the delegated name-servers, their IP address may also be present in the NS referral response as additional A records. These additional A records appear in an NS record, are called *glue records*. Glue records may be present for none, some or all the name-servers in an NS record. The DNS specifications do not provide clear guidelines on when glue record should be present nor how to process them on the recursive side. An old definition in RFC 1034 [19] states that glue records are required only if the NS is lying within or below the zone or domain for which it acts as a name-server. This condition is known as the *Bailiwick* rule. The reason for this requirement is to avoid a query deadlock for NS referrals that contain name servers within the domain being queried. For example, if the recursive is performing a resolution for the URL `www.example.com` and the TLD returns a referral containing `ns5.example.com` as a delegated name-server which resides within the example.com domain, but without its IP address, the recursive will then initiate another A query asking to resolve `ns5.example.com`. It will be again referred to `ns5.example.com`, which leads to a live-lock. A recent wider definition of the *Bailiwick rule* appears in RFC 8499 [2], suggesting that sibling domains (name-servers whose name is subordinate to the zone origin, and not subordinate to the owner name of the NS resource record) may also have glue records in their parent zone (e.g., .com may provide a glue record for `ns.domain.com` as a name-server for `example.com`). In practice, as we measure in §6, authoritative servers provide glue records according to the wider definition in order to eliminate extra overhead.

Notice that another motivation for the *Bailiwick* rule is to avoid and reduce the risk possed by cache poisoning attacks [27, 29].

How recursive resolvers accept and treat glue records returned by the authoritative servers is not defined in the DNS specification and left to their implementations. To prevent such cache poisoning attacks using malicious glue records, many recursive implementations store glue records as 'A' record in their cache only if they comply

with the *bailiwick rule*. If the glue record refers to a name server that resides within the domain for which it is a name server (*in-bailiwick*, for example, `ns.example.com` as a name server for the example.com zone) the recursive will honor the IP address provided in the glue record associated with `ns.example.com` and store it in its cache. Otherwise (*out-of-bailiwick*, for example, ns.example.net as a name server for the example.com zone), it will discard the glue record. Generally, without getting into different variations and implementation details, BIND [11] recursive implementation, which we analyze in this paper, as well as Unbound [15], PowerDNS [1] and Microsoft DNS all discard out-of-bailiwick glue records. Other solutions to eliminate cache poisoning attacks as a result of out-of-bailiwick glue records include DNSSEC - authenticating the authoritative responses by verifying their signature, but these have a low adoption rate.

Figure 2 illustrates the additional out-of-bailiwick requests that the recursive issues for `www.microsoft.com`. In this case, the TLD (.com and .net) name servers are already in the cache as a result of previous requests. The .com authoritative responds with an NS referral (step 2) containing four out-of-bailiwick name servers (`ns*.msft.net` for the queried zone microsoft.com). The recursive then initiates four additional resolution fetches for all these name servers. Note that even after it receives their IP addresses in the referral responses (steps 7-10) as glue records, it still performs additional resolution requests for them (steps 11-14). This is because their corresponding requests' recursion state was already initiated independently with an indication that they are not cached.

2.3 Out-of-Bailiwick overhead implications

It is important to distinguish between the time in which an authoritative name server from the list in the NS referral response is resolved (to obtain its IP address) and the actual time in which it is queried [23]. As shown in [23], in each client request, a different authoritative name-server from the list is likely to be queried in order to improve latency. In this paper, we focus on the timing of the resolution and propose it should be spread and amortized over several client requests (see §5), and not as it is now, where all the resolutions are performed in the first client request. Moreover, many domains outsource their authoritative name-servers to cloud operators such as Cloudflare, domaincontrol.com, and these operators often choose short TTL values (30 or 60 seconds). This in turn, causes many server resolutions that have been performed by the recursive resolver to be outdated by the time the resolver wants to use them. As a result, the resolver has to redo the corresponding resolution.

This gap between the number of resolution packets per query expected in theory and the one observed in practice raises several questions:

1. Does this phenomena introduce new vulnerabilities?
2. Should the recursive resolver resolve all the missing name-servers IP addresses upon arrival of the first client query? Or should these extra queries be amortized over future client queries?
3. What is the prevalence of Out-of-Bailiwick nameservers?
4. What is the impact on the normal operation of recursive resolvers? Does the cache mitigate the overhead over time?

To answer the first question, we present the NXNSAttack in §3. We evaluate the attack and present our experiments in §4.

Section 5 suggests a solution to mitigate the NXNSAttack, and proposes an answer to the second question.

To evaluate our solution, and provide answers to the third and fourth questions, we present our experiments and measurements in §5 and §6. We measure the prevalence of *Out-of-Bailiwick* domains, and measure the *Out-of-Bailiwick* additional resolutions overhead on two different data-sets: (i) the top million domains list. (ii) a real-life DNS trace from our campus.

Note that we observed additional reasons for the high number of messages in DNS resolutions, which are (i) too long NS responses that include multiple name servers and other options such as RRSIG and NSEC3 data in the additional records, leaving no place for all the glue records in a 512-byte UDP packet, forcing the recursive to resend the request using TCP, or by using the UDP EDNS0 4096-byte option. (ii) Canonical NAME records (CNAME) that reside in different domains than the queried one, these sometimes have to be resolved with an additional fetch starting from the root-servers. However, we identify the additional resolutions of name-servers IP addresses as the major reason for the high number of messages. Moreover, many of the other reasons mentioned above are associated with new resolutions that are issued due to the missing IP addresses of name-servers.

3 NXNSAttack

Here we exploit the multi name-server referral response and the resulting extra resolutions to carry out a new attack, NXNSAttack (NoneXistent NameServers Attack), on different elements of the DNS infrastructure.

As shown in the previous section, for each name-server name without an associated IP address, in the NS referral response, the recursive resolver initiates a new resolution

procedure. This is the core of our attack. The attacker uses the authoritative that it owns to craft a response to a resolver with a referral that contains n new and nonexistent name-server names without an associated IP address, and as a result, this resolver starts the process of F new resolutions. As shown later, the maximum F can be in the range, $74 \leq F \leq 2 \cdot n$ (in BIND implementation, $2n$ requests to resolve the IPv4 and IPv6 addresses). When the attacker generates many such referral responses repeatedly, she gets a DDoS attack on either the resolver or on a corresponding authoritative server, with an amplification factor of $O(F)$ packets, sometimes much larger than F. There are several parameters and variants of this basic principle that we investigate in this paper.

3.1 Threat Model

To mount a NXNSAttack on either a recursive resolver or an authoritative server, an attacker should:

1. Have access to one or more DNS clients on the Internet (May use a botnet such as in the Mirai IoT botnet [4] or through an ad-network [13]).

2. Own or compromise an authoritative name-server. This can be easily achieved by e.g., buying a domain name. An adversary who acts as an authoritative server has the ability to craft any NS referral response as an answer to different DNS queries. It controls the information that appears in the referral response, such as the number of name-servers, their names, and their glue records (as well as the absence of glue records).

Controlling and acquiring a huge number of clients and a large number of authoritative NSs by an attacker is easy and cheap in practice. Authoritative name-servers are easily acquired by first buying and registering new domain names, which is cheap (for our experiments, we purchased several domain names for less than $1 per name) and takes less than 5 minutes. Secondly the acquired domain names can be dynamically associated with any authoritative server in the internet. Alternatively, attackers today have the capabilities to compromise DNS operators' credentials and to manipulate zone-files and sometimes even gain access to their registrar records, as exemplified by recent DNS hijacking attacks [9, 30]. Notice that recently attackers achieved much harder to get capabilities [4, 26] that use IoT botnets, Booters (DDoS for hire services [12]) and dynamic C&C servers.

3.2 The Amplifier

The core building block of the NXNSAttack is the *amplifier* (Figure 3), which is composed of three components: two attacker components and one innocent recursive resolver. The two attacker's components are a client and an authoritative name-server. Essentially the attacker issues many requests for sub-domains of domains authorized by its own authoritative server (step 1 in Fig. 3). Each such request is crafted to have a different sub-domain to make sure it is not in the resolver's cache, thus forcing the resolver to communicate with the attacker's authoritative server to resolve the queried sub-domains (step 2). The attacker authoritative name-server then returns an NS referral response with n name-server names but without their glue records(step 3), i.e., without their associated IP addresses, forcing the resolver to start a resolution query for each one of the name-server names in the response, whether these name-servers are *in-bailiwick* or *out-of-bailiwick* because it does not have their IP addresses in its cache (step 4). The attacker's authoritative referral response issues n new and different delegated name-servers each time it receives a query for a sub-domain from the recursive resolver.

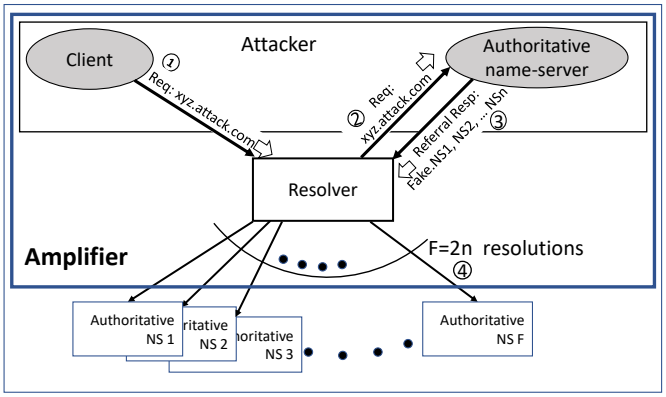


Figure 3: The amplifier

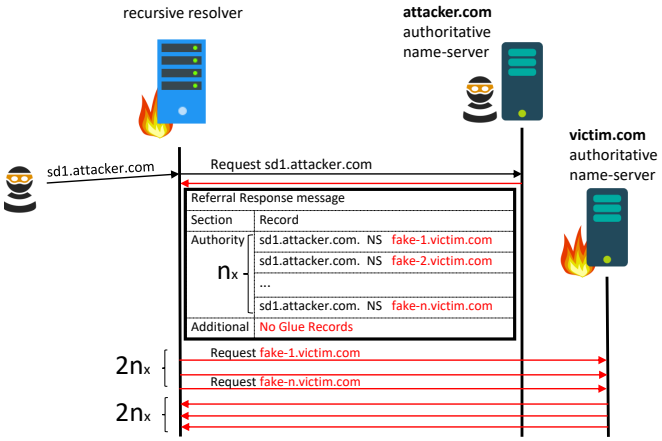
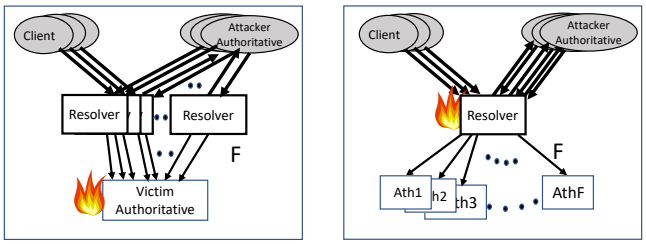


Figure 4: A different view of the messages exchanged in Figure 3



(a) (b)

Figure 5: NXNSAttack targeting the authoritative server (a) and the recursive resolver (b)

An attacker can use the amplifier building block to attack different targets and in a variety of ways, depending on three parameters:

1. The name-servers that appear in the referral might be at different levels of the DNS authoritative hierarchy: Root, TLD, or SLD. Different levels result in different maximum F factors as shown in §4.
2. Using multiple clients, communicating to multiple resolvers to target a single SLD authoritative victim (Fig. 5a), or using multiple name-servers to target a particular recursive resolver (Fig. 5b).
3. Extend the attack recursively, by setting self-delegations in the first malicious referral from the attacker's authoritative, which leads to $F1$ additional subsequent requests and corresponding referral responses (see Figure 6), each of which contains n_2 delegations, thus achieving up to $2n_2 \cdot F1$ name-servers referrals.

These parameters are controlled by the attacker and may be used to generate different attacks. Here we focus on three basic attacks: against a recursive resolver, against an authoritative SLD victim (e.g., victim.com name-server), and against the ROOT/TLD servers (.com, and ".". see Table 1 for a summary of the amplification factors).

Recursive resolver attack. (Fig. 5b) Here the maximum packet amplification factor (PAF) is 1620x (both by the model and empirically see §4) which is achieved when the referral delegations are to different TLD name-servers, (e.g., `fake1.com`, `fake2.com`, ...,`fake1.net`, ...). For each two packets the attacker components generate (one from the client, and one from the authoritative name-server) the victim recursive resolver processes 3,242 packets, out of which 1,081 are DNS packets and the rest are TCP connection control packets. The corresponding bandwidth amplification (BAF) is 132x, see §4.3.

Authoritative SLD attack. (Fig. 5a) In this attack, all the name-servers in the malicious referral

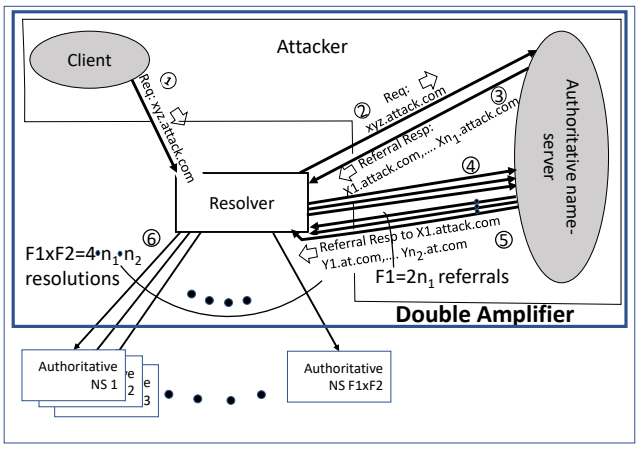


Figure 6: Illustration of the double amplification attack using self delegations in the first referral response. This attack variant (c) reaches a firepower of $F = F_1 \cdot F_2 = 37 \cdot 2 \cdot 135 \cdot 2 = 19,980$ (see §4.1).

are sub-domains of a victim SLD (second-level domain, e.g., `fake-1.victim.com`, `fake-2.victim.com`, ...). The maximum packets amplification factor is 74x, and the corresponding bandwidth amplification factor is 21x, see §4.3 for the cost and amplification factor analysis.

ROOT/TLD attack: Here the attacker uses the self-delegations technique (Fig. 6) to increase the number of concurrent referrals to the ROOT name-servers. In our empirical tests, the victim processes up to 81,428 packets (14,126,945 bytes) for each client request (and corresponding 75 referral packets) that the attacker generates (it is "only" 81,428 because many were lost). The high victim cost is because the first referral response from the attacker contains delegations to 37 new and different sub-domains of the attacker (e.g., `sd1.attacker.com`, ... , `sd37.attacker.com`), which results in 74 more requests (IPv4 and IPv6 for each delegated ns) to the attacker from the recursive resolver. The attacker's authoritative name-server then responds with 74 crafted referrals, each of which contains 135 delegations to the ROOT server (e.g., `domain.fake` or `domain.tld` where the tld name-servers are not cached in the recursive resolver) which in turn receives 18,980 concurrent requests see §4.3 for the cost and amplification factor analysis.

4 NXNSAttack Analysis Evaluation

4.1 F, The Amplifier Firepower

The traffic fan-out of the amplifier as a result of one client request is measured by either, the number of generated resolution requests, or the number of packets sent, or the number of bytes (bandwidth, bw) sent. In this section we

present the corresponding numbers that were measured in our test bed setup, and we provide an analysis that accurately explains these numbers.

Theoretically if the recursive resolver receives a referral response that delegates the original request to n name-server names, without providing their IP address (no glue records), the recursive then generates in response, $2n$ requests to resolve the IPv4 addresses and the IPv6, of each of these n names. However, there are two parameters that limit this number. First the maximum number of delegation names that fit into the referral response, denoted n_{max} or just n. From our experiments n_{max} is a function of the DNS packet size (including EDNS(0) extensions [6] and DNS over TCP) and the number of characters in the domain names. In our tests n_{max} turned out to be 135. Second, in BIND, the *max-recursion-queries* parameter that sets the maximum total number of requests a recursive resolver can send in the process of resolving one client request. As written in the BIND 9.12 manual "max-recursion-queries: Sets the maximum number of iterative queries that may be sent while servicing a recursive query. If more queries are sent, the recursive query is terminated and returns SERVFAIL. Queries to look up top level domains such as 'com' and 'net' and the DNS root zone are exempt from this limitation. The default is 75". We denote max-recursion-queries as Max_rq.

Since in step 3 in Fig. 3 the recursive already sends one request, the remaining Max_rq budget is 74. Which is sufficient to resolve 37 names, requesting separately the IPv4 and IPv6 address of each. Unless the requests are sent to either the root or a TLD name-server, in which case, n_{max} is the only limiting factor and the firepower F is $270 = 2n_{max}$.

4.2 Experimental Setup

We deployed an experimental setup that looks like Figure 3, on the AWS cloud in Ohio. Some testing is prevented since it requires attacking live operational name-servers. The setup includes a client, a recursive resolver, two authoritative servers: one for the attacker, and one for the victim. For each component, we use a large EC2 machine with 16Gb RAM and 4 vCPUs. The authoritative server runs BIND 9.12.3 in authoritative operation mode, while the recursive resolver runs BIND 9.12.3 in recursion mode. The client is deployed on a different machine, configured to send DNS requests directly to our recursive resolver.

We chose BIND because it is known as the most widely-used DNS server implementation [11, 20], and is considered as the de-facto standard for DNS servers. Moreover, a recent work [14] shows that the majority of open DNS resolvers operate BIND. We tested multiple versions of BIND in our experiments (different minor versions of 9.11 and 9.12), with no notable differences. Here we present the results as experimented with BIND 9.12.3.

To show that the vulnerability is not unique to BIND, we also provide in §4.4 our results on open recursive resolvers including Google, CloudFlare, Dyn and others. All the open resolvers that we tested were observed with a considerable amplification when sending a single request of the NXNSAttack.

4.3 Cost and Amplification Analysis

In Subsection 4.1 we computed F, the amplifier firepower, which is the total number of DNS requests generated by the amplifier, which was $2(\min(n, (Max\_rq-1)/2))$ if the attack is on an SLD domain, and $2n$ if the attack is on a TLD or root servers (results in $F = 74$ and 270 respectively). The 2 factor here is due to requesting IPv4 address and IPv6 separately. But how many packets and bytes does it translate into? We both measure it in our setup and explain (calculate) the observed numbers by analyzing the protocol used by BIND.

We claim that the cost to the victim in packets, denoted C_v^{pkt}, as a result of one client request, as a function of F, is:

$$C_v^{pkt} = 2 \cdot F \cdot (1 + 5 \cdot TC) \quad (1)$$

Where TC, the value of the truncate bit in the DNS protocol, equals 1 if the F requests fall back to TCP, and 0 otherwise. The TC bit indicates whether the UDP DNS request/response has failed due to UDP packet size limitation and is retried in TCP. This often happens when the delegated name-servers support DNSSEC signing (e.g., TLD servers, as we observed in our evaluation in §4.3). In such cases, the resolver retry (request and response) involve additional TCP control packets. In our evaluation in §4.3 we observe that each such TCP request response involves a total of 10 packets: DNS request, DNS response, and 8 TCP control packets (3 for handshake, and 5 for session termination).

The factor of 2 in (1) is because we count both the packets sent and received by the recursive resolver or the authoritative victim towards their attack-cost. Traditionally, in DDoS bandwidth attacks the packets/bytes amplification factor is the number of packets/bytes that are sent to the victim divided by the number of packets/bytes the attacker sends. However, the NXNSAttack is both a bandwidth and a complexity attack, consuming different resources at the victim. The victim name-server is forced to receive many packets, process them, make memory accesses, consume cache/memory capacity, and respond with new DNS request or response packets including TCP connections. Therefore, towards the analysis of the amplification factor we consider the packets the

victim (the recursive or the authoritative) both receives and sends.

Equation (1) provides C_v^{pkt}, the cost incurred by the victim (recursive resolver or authoritative server) when attacked by the amplifier. In calculating the packet amplification factor (PAF) of the different attacks, we need to divide the victim cost by the cost incurred by the attacker, denoted C_a^{pkt}. In both attack a and b (following Fig. 3) the attacker sends two messages, the client request, and the referral response from the attacker controlled authoritative name-server. In attack *c*, Fig. 6, the attacker's authoritative server sends 74 packets making $C_a^{pkt} = 75$.

PAF is the ratio between the number of packets processed by the victim and the number of packets sent by the attacker, i.e., $PAF = \frac{C_v^{pkt}}{C_a^{pkt}}$. Similarly, the bandwidth amplification factor, $BAF = \frac{C_v^{bw}}{C_a^{bw}}$. Where, C_a^{bw} and C_v^{bw} are the number of bytes that the attacker and victim have to send/process respectively.

The costs discussed above are incurred with every client request because the attacker's authoritative server issues referral requests with new and different fake (non-existent) names each time. In addition, there are one time costs that we ignore but will show up in our measurements. These represent the packets exchanged between the recursive resolver and the ROOT/TLD authoritative name-servers to resolve the attacker and the victim name-servers respectively. Since these name-servers are cached after the first client request, we do not consider them in the packets cost analysis.

In Table 2 and below we analyze each attack variant, describing the variant and comparing its measured cost to its calculated cost according to the model above.

(a) Recursive resolver attack (row a in Table 2). Here each attacker's referral (step 3 in Figure 3) contains delegations to many new and different name-servers of the `.com` zone. The zone file contains millions of NS records, and looks like this:

```
ORIGIN sd0.attacker.com.
sd0.attacker.com. IN NS ns1.fakens0.com.
sd0.attacker.com. IN NS ns1.fakens1.com.
...
sd0.attacker.com. IN NS ns1.fakens-n.com.
```

Considering that .com and other TLD name-servers are external to our setup, we initiated only a few requests for `sd*.attacker.com`, while monitoring the recursive resolver behavior.

First, we experimentally measured that $n_{max} = 135$ in our setup. The resulting firepower is thus 270 requests that are sent to one of the `.com` TLD name-servers, asking 'who is `ns1.fakens*.com`?'. The `.com` name-server responds with negative responses (NXDOMAIN). However, all TLD responses also contain a SOA record, RRSIG and multiple NSEC3 signatures (DNSSEC signatures), thus exceeding the maximum response size, of 512 bytes, therefore, setting the TC bit on, and forcing the resolver to repeat the 270 queries over TCP (which in addition creates a lot of overhead on the resovler and the authoritatives to handle these TCP connections). Thus, C_v^{pkt} by equation (1) is 3240. In the setup we measured 3243 due to the initial one time non-recurring resolution of the attacker's authoritative server, and the victim recursive resolver addresses. The PAF is thus $\frac{C_v^{pkt}}{2} = 1620$.

We also measured the BAF in our setup which turned out to be 163, which is very close to what expected when taking into account the sizes of the different packets.

Note that here the `.com` TLD can also be considered as a victim because it processes the same packets as the recursive resolver under attack. Moreover, as described in Figure 5a, several resolvers may be used to mount a massive attack on any TLD or root server. We also performed this experiment with other TLDs (`.live` and `.online`) and received the exact same results.

(b) Authoritative SLD attack: To attack a particular SLD server, each attacker's referral is crafted to contain delegations to many new and different sub-domains of the victim SLD (e.g., `fakens1.victim.com`, `fakens2.victim.com`, ...).

In this attack variant, BIND *max-recursion-queries* threshold does limit the number of iterative requests to 75. To test this attack we used two name-servers, one as the attacker's and one as the victim. Since our authoritative victim does not use DNSSEC, no TCP retries are involved. Thus, $C_v^{pkt} = 2 \cdot 74 = 148$ and PAF is $\frac{C_v^{pkt}}{2} = 74$x. The victim bytes cost is $C_v^{bw} = 22,073$ bytes, and the attacker bytes cost is 1,049 bytes which leads to BAF of 21x. As before the measurement on one client request is 150 rather than 148 due to the one time resolution of the attacker and resolver servers, which should not be counted towards the PAF or BAF calculations.

(c) ROOT TLD attack. To attack a TLD or root servers (a tough challenge since there are hundreds of them, using any-cast and load balancing) one can try variant a, or with much fewer client requests as described here (using variant c). In this experiment, the attacker uses the self-delegations technique (Fig. 6 as described in §3.2) to achieve a double amplification effect and launch a long-lived attack against the ROOT or TLD name-servers. The attacker's first referral (step 3 in the figure) contains n_1 different sub-domains of itself (e.g.,

	Victims	Cost Factors		Packets Cost		PAF	Bytes Cost		BAF
		Firepower (F)	TC bit retry TCP	Attacker C_a^{pkt}	Victim C_v^{pkt}		Attacker C_a^{bw}	Victim C_v^{bw}	
a a	recursive resolver, TLD name-server	270	1	2	M 3,243 C 3,240	**1621x**	3,967	647,107	**163x**
b b	SLD name-sever, e.g., victim.com	74	0	2	M 150 C 148	**74x**	1,049	22,073	**21x**
c	ROOT or TLD ns	74x270	1	76	M81,428 C239,760	**1071x**	142,487	14,126,945	**99x**

Table 2: Different attack variants cost as a result of one client request, using BIND (M - is measured, C - is calculated).

`sd1.attacker.com`, ... , `sdF1.attacker.com`), forcing the resolver to send $2n_1$ additional queries (step 4) to resolve the IPv4 and IPv6 addresses of these delegated name-servers. The attacker server then responds to these (step 5) with $2n_1 = F1$ referral delegations each with n_2 delegations. This results in total of $2 \cdot F1 \cdot n_2$ delegations, each of which is a name of a fake TLD server (e.g., `ns.fake1`, `ns.fake2`, ... , `ns.fakef1xf1x2`). $F1$ is constrained by the *max-recursion-queries* parameter to 74 and n_2 to 135 by the n_{max}. Resulting in a maximum amplifier fan-out in this variant of $74 \cdot 270$.

In this experiment there is a huge discrepancy between the measured and calculated victim cost (81,428 vs. 239,760). This is since many packets and requests are lost in the experiment, because the resolver has to send and receive 19,980 requests at the same time, which it fails to do.

Long-lived attacks simulation. So far we mostly discussed the attack power as a result of one client request. Since the attack is using non-existent domain names, the cache mechanisms do not help, and the attack is long-lived. To show this we simulate a long-lived attack using variant b that does not interact with external authoritative servers; hence we could experiment it on our setup without leaking any attack packets outside of the virtual lab. As appears in Table 2, F of this variant is 74, thus we include 37 name-servers names in each NS referral response. We monitor the packets processed in both the recursive resolver, and the victim authoritative server in the test-bed. We used the *resperf* tool [25] on the client machine (acts as the attacker) to send a query stream consisting of many unique DNS 'A' requests to `sd*.attacker.com`. As seen in Figure 7 (See the 'Original BIND' line), 10,000 attacker requests result in 1,500,319 packets involved in the recursive resolutions, producing a constant PAF of 75x. Here each client query ends with a SERVFAIL, but the recursive resolver's cache is filled with 740,000 NXDOMAIN records (each client request triggers 37 IPv4 resolutions, and 37 IPv6 resolutions), and 10,000 NS records. Thus both having a large PAF, and saturating the cache and the memory very fast.

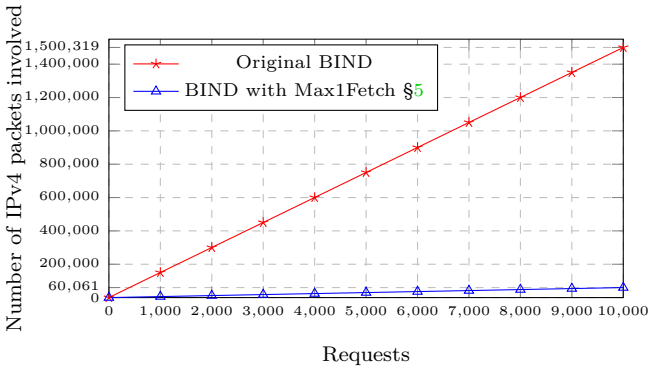


Figure 7: NXNSAttack long-lived simulation against an SLD authoritative server. A Constant PAF of 75x in the original BIND compared with PAF 3x of Max1Fetch (see §5). Recall that attacker cost is $2 \cdot$#requests.

4.4 Public DNS servers

Here we tested whether public DNS servers (such as, cloudflare, Google, Quad9, etc.) could be used as the resolver in the amplifier and what firepower one can get when using them. The DNS software among the public resolvers varies, some have their own proprietary implementation. Our test setup is as in Figure 3, where the recursive resolver is the public DNS server we test. We used attack variation b, in which an SLD authoritative is the victim, and we used our own ns.victim.com as the victim. Since we cannot really mount an attack using a public DNS server, we tested each with one client request at a time, for several requests until finding the maximum firepower. The results are given in Table 3. We could not test variants a, and c, because these require monitoring the recursive resolver or the TLD/ROOT servers. To this end, we deployed 'malicious' name-server that responds to queries for `xxx.attacker.live` and sent few queries to each one of the public resolvers, requesting `sdX.attacker.live`. For each such request, our name-server, ns.attacker.live, responded with a referral response with a different number and sub-domains of the victim (our name-server, `victim.online`). For each request sent, we monitored how many requests arrive at the victim name-server. All the public DNS resolvers

that we tested exhibited a large PAF, on a single request of the NXNSAttack. Some have a higher PAF than the one observed in BIND for this variant b (74x).

Public DNS recursive resolver (IP)	Max # of delegations = F/2	Victim cost C_v^{pkt}	PAF
CloudFlare (1.1.1.1)	24	96	48x
Comodo Secure (8.26.56.26)	140	870	435x
DNS.Watch (84.200.69.80)	135	972	486x
Dyn (216.146.35.35)	50	408	204x
FreeDNS (37.235.1.174)	50	100	50x
Google (8.8.8.8)	15	60	30x
Hurricane (74.82.42.42)	50	98	49x
Level3 (209.244.0.3)	135	546	273x
Norton ConnectSafe (199.85.126.10)	140	1138	569x
OpenDNS (208.67.222.222)	50	64	32x
Quad9 (9.9.9.9)	100	830	415x
SafeDNS (195.46.39.39)	135	548	274x
Ultra (156.154.71.1)	100	810	405x
Verisign (64.6.64.6)	50	404	202x

Table 3: Firepower and PAF of public resolvers as a response to a single request in the NXNSAttack.

4.5 NXNSAttack vs. NXDomain Attack and its effects on the DNS system

Both NXDomain and NXNSAttack use non-existing domain names to bypass the recursive caches and reach different name servers. The NXDomain attack (*water torture* [16, 28]) is easier to launch, because it does not require a malicious authoritative server. However, the NXNSAttack is more destructive in two parameters; First, its PAF is ranging from 74x to 1602x, while the PAF of the NXDomain attack is 3x. Secondly, while the NXNSAttack continuously consumes memory and ns-cache records in the resolver, whether or not negative caching is enabled, the NXDomain attack does not have this effect on the memory or the cache of the resolver, it only grows the negative cache (NX records) and at a slower pace. Notice that some ISPs have decided to disable negative caching due to considerations such as the increased pervasiveness of one-time signals and disposable domains [10], thus completely eliminating the cache growth during an NXDomain attack. Therefore, since it has been reported [24, 26] that ISP recursive resolvers were knocked down by the NXDomain attacks, we conclude that the NXNSAttack would also take down these servers, perhaps even faster.

Attack effectiveness Comparison. While variant b of the NXNSAttack is the least effective, with the smallest PAF and likely also the smallest cache consumption rate, it is the only variant we can easily compare against the NXDomain attack in a stress test in our setup. We used the same setup as in the long-lived test in Section 4.3.

In the comparison we measured, MaxQps, the maximum rate of attacker client requests before the victim resolver or the authoritative server starts to lose requests. We prepared a file that contains one million requests for each attack (each has different bogus requests to instigate the attack) and gave it to *resperf* stress tool by Nominum [25], running on the client (We did not use *queryperf* [7] of BIND because it has been reported [25] to produce poor results.) The MaxQps throughput is determined as the point at which the server starts dropping queries, and the response rate stops growing, meaning that the server capacity has been exceeded.

The results show that the MaxQps of the BIND recursive resolver significantly degrades under the NXNSAttack with a peak of 932 Qps. The resolver throughput under the NXDomain attack is 3708 Qps. This mainly attests to the much higher PAF of the NXNSAttack, which requires much fewer malicious client's requests to saturate the resolver. As a reference, the max throughput that we measured under non-attack traffic (e.g., our campus DNS trace and top million domains) varies between 6,000 (in case that most of the requests are not cached) to more than 100,000 Qps (where most of the requests are already in the cache).

4.6 Saturating the DNS server

We do not have access to a real authoritative, or resolver servers in order to show how they fail under attack. As an alternative, we measure here the maximum rate of NXNSAttack type requests that each such server installed on a strong EC2 machine can handle before losing requests. Since this rate of requests is easily attained by the NXNSAttack we deduce that the attack can easily reach the level required to take down these servers. We used an xlarge EC2 machine (4 vCPU with 16GB memory) with BIND 9.12.3 in both resolver mode and authoritative mode. In resolver mode it starts to lose requests at a rate of 932 client requests per second (same requests that are issued by attacking clients in NXNSAttack). In this experiment, we observed a large CPU resource difference between the victim and the attacker; the victim 4 vCPU resolver load exceeded 390%, while at the same time, the attacker's authoritative 1 vCPU load was only 3%. In authoritative mode we fed the authoritative two different streams of requests. The first, a stream of 'A' requests, and it starts to lose requests when reaching a rate of 68,208 Rps. In the second, a stream of NXDOMAIN random requests same as the requests that are sent to an authoritative victim in an NXNSAttack (e.g., in step 4 in Figure 3), and the maximum rate achieved was 65,418 Rps. Therefore, to overwhelm the authoritative an NXNSAttack of 1,000 client requests per second (with PAF=x75) would be sufficient.

5 Attack Mitigation: Max1Fetch

5.1 Possible and Existing Measures

The first mitigation techniques that come to mind are methods to identify and detect authoritative name-servers that issue many malicious NS referral responses. These are servers whose referral responses result with many SERVFAIL resolutions of the delegated servers. However, this requires sophisticated identification algorithms to distinguish miss-behaving authoritative servers, and the attackers will insert legitimate delegations in between to miss-lead the detection. Moreover, such malicious name-servers can dynamically change their name and IP address (in the same manner as malicious C&C servers do).

Following the NXDomain attack, recent versions of BIND have new manual rate limiting features designed to throttle queries from a resolver to authoritatives that are under attack. These rate-limiters, (e.g., 'fetch-limits', fetches-per-server, and fetches-per-zone.) are however a double-edged sword, and can become a way to DDoS an authoritative by issuing many requests to hit the threshold and block legitimate requests. Moreover, setting a rate-limit per authoritative zone or per authoritative name-server does not protect the recursive resolver from the NXNSAttack.

5.2 Max1Fetch

We propose to amortize the resolution of multiple delegations for a zone over multiple requests that use that zone, one or few resolutions per request, rather than resolving all the delegations of the zone at once, when the referral for the zone first arrives. Thus, in general, each request while using an already delegated name-server, resolves the IP address of an additional delegated name-server to be ready for the next request. Until all the delegations that were provided in a referral response are resolved. Here we only provide a high level description of Max1Fetch. A detailed discussion is relegated to a followup paper.

We have modified BIND 9.12.3 resolver algorithm to implement Max1Fetch. The max number of external fetches (additional resolutions) we enforce at each level is configurable. Max$k$Fetch allows the resolution of k additional delegations that do not have an associated IP address, per request. In Max1Fetch a resolver that uses a zone z while resolving a request, checks if there are unresolved delegations for z, in z's NS record. If such a delegation is found, the resolver initiates the resolution of this delegation, while continuing in parallel the resolution of the original request, using an already resolved delegate for zone z. Notice that the first request that uses zone z (which has also received the corresponding referral response) may have to wait for the resolution of the first delegate if all of them came without a glue record in the referral response (or all are *out-of-bailiwick*.) In this case the second request that uses zone z will use the same delegate that the first one used (one may consider resolving two delegations in the first request, something we have not implemented).

It is important to note that Max1Fetch does not have negative effect on the latency of a request resolution (see latency analysis in §5.3 and §5.4), nor does it disturb the RTT estimation algorithms (such as sRTT). Most recursive resolvers perform latency-wise algorithms to decide about the next-server to approach. However, Max1Fetch does not disrupt these algorithms because it allows a resolution of an additional name-server that may be selected in the next client request, and after enough requests all the delegations are resolved. The resolution of an additional name-server is not adding to the latency of a response since each request, except the first, uses a previously resolved name-server while issuing the additional resolution in parallel.

In the next sub-sections we evaluate and compare the original BIND and Max1Fetch. We focus on the impact on the latency and the number of packets, per client request, under normal traffic and under attack.

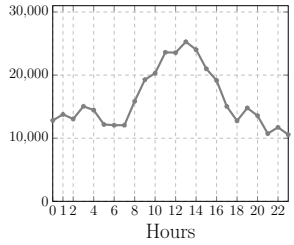
5.3 Max1Fetch evaluation under NXNSAttack

In Figure 7 (§4.1) we compare the PAF of the original BIND to that of the Max1Fetch variant, during a long-lived simulated NXNSAttack against an SLD victim. The blue line shows that Max1Fetch enhancement avoids most of the additional resolutions, since it initiates only two additional requests, one IPv4, and one IPv6 per reqest. Instead of 1,500,319 packets exchanged by the original BIND recursive resolver (as a result of 10,000 malicious client requests), Max1Fetch exchanges only 60,061 packets (The measured Mac1Fetch PAF is reduced from $75x$ to $3x$).

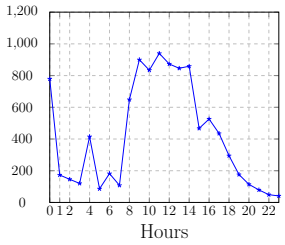
	Orig Bind 9.12.3	Max1Fetch
Max requests/sec	932	3390
Avg. Latency (ms)	4.31	1.32
Median Latency (ms)	4	1
std Latency	4.51	1.37

Table 4: Comparing BIND resolver performance under NXNSAttack with and without Max1Fetch.

We have also repeated the stress testing as in §4.5 to measure the maximal number of client queries per second that BIND resolver is able sustain under NXNSAttack with and without Max1Fetch. As seen in Table 4, BIND with Max1Fetch is capable of processing many more attack requests, 3,390 vs. 932 under the NXNSAttack



(a) Num. 'A' requests per hour



(b) Num. unique 'A' requests per hour

Figure 8: Data-set $\mathcal{B}$ (Campus trace) requests per hour profile.

(and 3708 orig. BIND under the NXDomain attack §4.5). We also compared the latency of attack requests with and without Max1Fetch. The latency values are observed at the attacker client that generates requests during a simulation of the NXNSAttack against an SLD victim in our test bed. Table 4 shows the average, median, and std latency, under attack, with and without Max1Fetch.

5.4 Max1Fetch in normal operation

Here we evaluate the recursive resolver operation under real-life scenarios. We compare the original BIND and Max1fetch to measure: (i) the latency of client queries. (ii) number of IPv4 packets processed by the resolver in the resolution of real-life queries. The purpose is twofold: first, we verify that the Max1Fetch modification does not incur query delays or resolution failures (i.e., the number of SERVFAIL and NOERROR responses is not higher than the one observed in the original BIND). Second, we measure the impact of the *Out-of-Bailiwick* overhead on the recursive resolver under normal operation, to determine whether the cache mitigates this overhead over time.

5.4.1 Data-sets

Two data-sets are used to study the normal operation of the BIND resolver:

Dataset $\mathcal{A}$: A list of the **top million domains** [17]. Here we executed DNS 'A' requests (IPv4 resolution) for the first 100,000 domains in this list.

Dataset $\mathcal{B}$: Campus DNS trace. A 24 hours trace of live DNS traffic observed at our campus DNS server. We exclude all internal queries for domains that reside within the campus zone. This leaves us with 1,027,359 incoming requests, out of which we take only the 'A' requests, which result in 386,736 queries. Notice that only 10,092 are unique requests. Figures 8a ,8b show how the total 386,736 queries and the 10,092 unique queries, respectively, are spread over the day.

Ethical Consideration: Dataset $\mathcal{B}$ is a sequence of DNS queries with their timestamps but without the IP addresses that originated them.

	Dataset $\mathcal{A}$ (100K top domains)		Dataset $\mathcal{B}$ (Campus trace)	
Resolver Impl.	Original BIND	Max1-Fetch	Original BIND	Max1-Fetch
Total Req.	100,000	100,000	386,691	386,691
Unique Req.	100,000	100,000	10,092	10,092
Total recursion packets	**747,494**	**650,864**	**454,032**	**422,946**
NOERROR			363028	363031
SERVFAIL			18911	18910
NXDOMAIN			4752	4750
Latency (ms)				
Mean	**157.37**	**155.95**	**41.50**	**40.97**
Median	53	52	13	13
Std	298.63	293.37	101.03	95.81

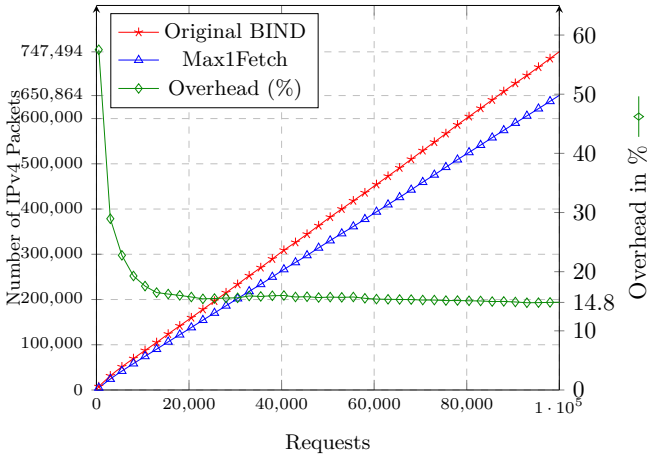
Table 5: Comparing between original BIND and Max1Fetch during the resolution of query streams of Datasets $\mathcal{A}$ and $\mathcal{B}$

With each data-set, we send its query stream (100,000 queries in Dataset $\mathcal{A}$, and 386,736 queries in Data-set $\mathcal{B}$) to both original BIND and BIND with Max1Fetch. The resolvers cache is empty at the beginning of each experiment. The cache size is 1GB, which is sufficient to store all the responses of the input query stream. We record the traffic between the recursive resolver and the authoritative hierarchy, as well as collecting the BIND statistics.

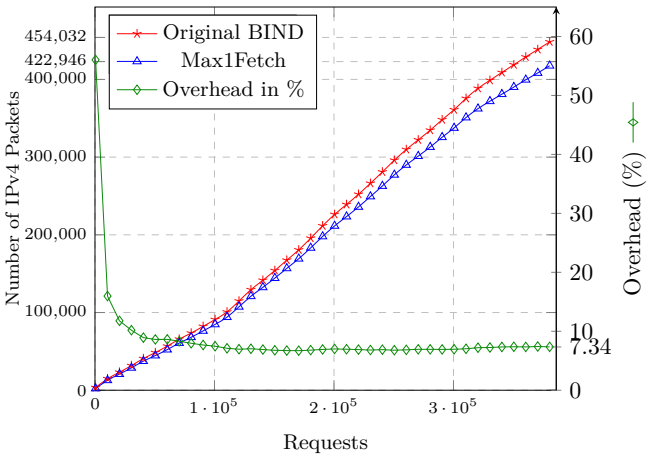
5.4.2 Results

Number of extra resolutions. First we measure whether Max1Fetch reduces the overhead that was observed in the resolution process in practice (see §2.2). Figure 9 and the fourth row (Total recursion packets) in Table 5 show the number of packets processed by the recursive resolver (with and without Max1Fetch) in each of the data-sets. The resolver using original BIND exchanges 14.84% more packets in the resolution of queries from Data-set $\mathcal{A}$ than the Max1Fetch variant (747,494 vs. 650,864). Similarly, in Dataset $\mathcal{B}$ (Campus DNS trace), original BIND exchanged 7.34% more packets (454,032 vs. 422,946).

The green lines in Figures 9a, and 9b show the resolution overhead in percentages $((\frac{Packets_{orig}}{Packets_{max1fetch}} - 1) \cdot 100)$. As seen, in both datasets, the gaps after the first 1000 requests show a high overhead of more than 50%. The gap in percentages is decreasing as long as we send more requests since the cache is filled with answers to previous requests that share the same name-servers. However, after 20K requests, the gap remains stable (at around 15% in Dataset $\mathcal{A}$, and 7% in Dataset $\mathcal{B}$) until the end of the experiment. Indeed, we see that Max1Fetch does not introduce more SERVFAIL than original BIND

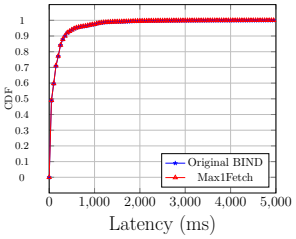


(a) 100K Top Domains

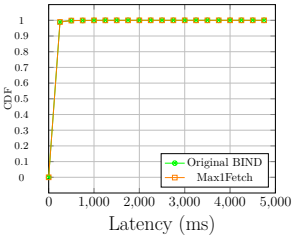


(b) Campus 1-Day Trace

Figure 9: The number of recursion packets exchanged by a BIND resolver (with and without Max1Fetch) in the resolution of Data-set $\mathcal{A}$ and $\mathcal{B}$ query streams. The green line shows the overhead that is relative to Max1Fetch.



(a) 100K top domains



(b) Campus trace

Figure 10: Latency of queries in Datasets $\mathcal{A}$ and $\mathcal{B}$: Comparison between original BIND and Max1Fetch

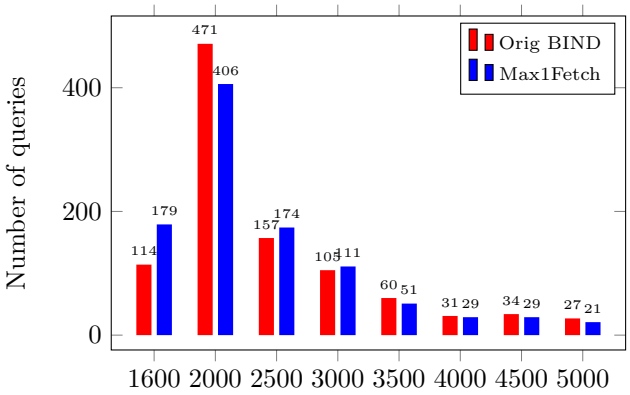


Figure 11: 100K Dataset: 99th percentile latency comparison

in the resolution the 386,691 queries in Dataset $\mathcal{B}$ (see the fifth row in Table 5).

Latency. The last row in Table 5 shows the average, median and std latency, in both datasets, with and without Max1Fetch. The results exhibit a slightly faster response time using Max1Fetch: 157.37ms using original BIND vs. 155.95ms using Max1Fetch in Dataset $\mathcal{A}$ (top domains), and 41.5ms vs. 40.97ms in Dataset $\mathcal{B}$ (campus trace). Notice that in Dataset $\mathcal{B}$, the majority of the queries are served by the resolver cache because not all the requests are unique (see Figure 8b). Thus, when calculating the average, median and std calculations, we exclude queries with zero latency (we consider only 114,570 out of 386,691 queries).

Figures 10a, 10b show the cumulative distribution of the queries latency values with and without Max1Fetch in Datasets $\mathcal{A}$ and $\mathcal{B}$ respectively. The latency values are between 0 and 5 seconds. In both datasets, the original BIND and Max1Fetch CDF lines are overlapping, exhibiting a similar distribution.

The 99th percentile latency distribution in Dataset $\mathcal{A}$ (top domains) is provided in Figure 11. The quantile values (cut points of the 99th percentile) for original BIND and Max1Fetch are 1,414ms and 1,382ms respectively. Similar 99th percentile distribution in Dataset $\mathcal{B}$ appears in Figure 12.

Figure 13 presents the latency differences per domain request (between original BIND and Max1Fetch) in the top domains dataset. Here, for each domain d request we calculate $L^d_{orig} - L^d_{m1f}$, where L^d_{orig} is the latency of the query for the domain d using original BIND, and L^d_{m1f} when using Max1Fetch. Figure 13 shows the distribution of the calculated values (vary from -5000 to 5000), where positive values represent domain requests for which Max1Fetch performed faster.

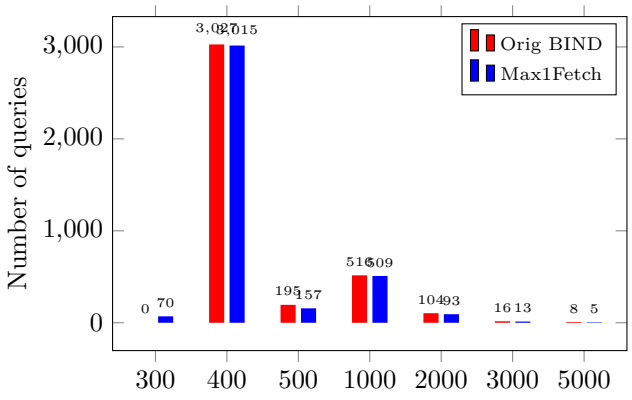


Figure 12: Campus Dataset: 99th percentile latency comparison

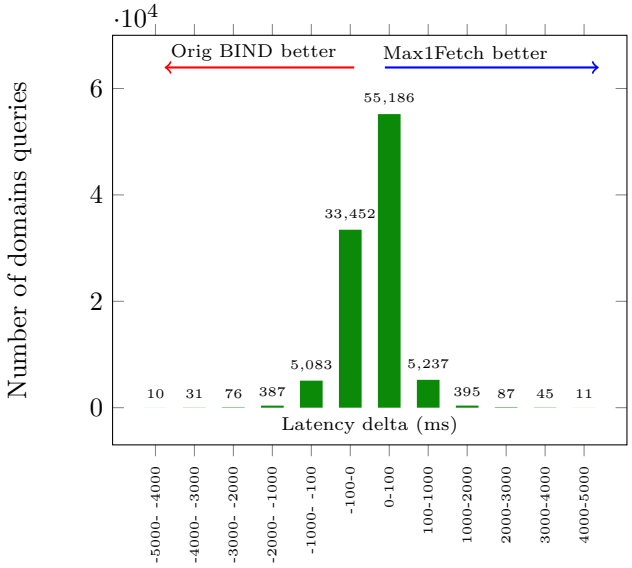


Figure 13: 100K websites Dataset : OrigBIND - Max1Fetch latency per domain histogram

5.5 Additional remedies

Here we point out additional directions to mitigate the NXNSAttack. We intend to investigate these approaches in a followup work.

MaxBreadth: The Max1Fetch proposal mitigates and significantly reduces the PAF (and BAF) of the attack, however, the attack still consumes large amounts of memory and cache (NX, NS records) per client request, in particular variant *c*. To mitigate this additional attack vector we suggest to adopt recommendations to restrict the breadth, i.e., the number of delegation name-servers in an NS record of a domain/zone. This restriction is supported by the observations made in §6 in particular Table 15 shows that about hundredth of a percent of the top 1M domains have more than 13 name-servers and less than one percent have more than 7 name-servers. The limitation should be a function of the level of the zone and of the authoritative, from which the referral that creates the NS record, arrives. Thus for an SLD zone a default restriction of 4 makes sense. Investigating the exact limits and effects of this MaxBreadth proposal is beyond the scope of the current paper, and we plan to continue evaluating and investigating this restriction in a followup work.

Behavioral analysis: In the spirit of IPS's it is possible to monitor the referral messages incoming to resolvers and detect abnormally large referrals for zones that appear only once or a small number of times. Heavy hitters and distinct heavy hitters Algorithms, such as in [8] may be used for this. The disadvantage of this approach is that operators will have to deal with yet one more package and the upgrade path is not clear. Can such algorithms be integrated into BIND or other DNS software?

6 The Pervasiveness of Out-of-Bailiwick Nameservers

Here we measure the prevalence of domains with *out-of-bailiwick* name servers. We show that the majority of the domains out of the top 1M popular sites [17] have *out-of-bailiwick* name-servers. Two controlled experiments to monitor the resolvers operation and to examine the NS referral responses in the resolutions of the top 1M domains, are performed.

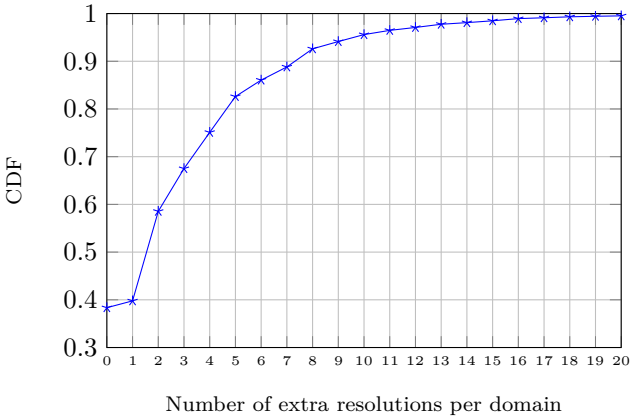


Figure 14: Number of domains that their resolution incurs initiation of extra resolutions by BIND (over top million domains).

In the first controlled experiment we measured how many recursive resolutions a BIND based resolver performed when resolving each of the top 1M domains. The

cache has been cleared before issuing each client request. We measure the extra resolutions as in the difference between Figure 1 and 2. That is, at each level of the hierarchy one resolution is not counted for. Figure 14 shows the cumulative distribution of domains that trigger additional resolutions (fetches). The figure shows that 60.22% of the domain requests initiate more than one additional fetches. We see that 374,498 domain requests do not initiate any additional resolution (38.34%, notice we count only requests with NOERROR responses).

In the second experiment, we record the communication between the recursive resolver and the authoritative structure during the resolution of the 1M domains. In this case, we do not focus on the BIND operation, but rather inspect the NS referral responses that are received from the authoritative hierarchy to measure: (i) how many name-servers are returned for each domain, (ii) how many name-servers are not provided with their corresponding IP addresses (missing glue-records), and (iii) which name-servers are *out-of-bailiwick*. When counting the ammount of *out-of-bailiwick* name-servers, we consider both definitions as we discuss in §2.2 (RFC 8499). The first strict definition, describes a name-server whose name is subordinate to the owner name of the NS resource record (e.g., ns.child.example.com as name-server for the domain 'example.com'). The second wider definition, refers to a name-server's name that is subordinate to the zone origin and not subordinate to the owner of the NS resource record (e.g., ns.another.com as name-server for the domain 'example.com').

We start by counting in Figure 15 the number of name servers for each domain. While most of the domains have two name servers, 33% of the domains have three or more. Results show that top million domains have an average of 2.52 name servers per domain.

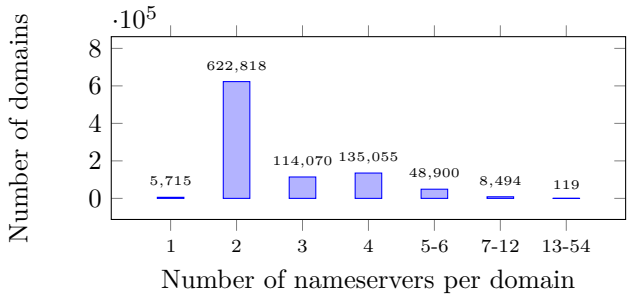


Figure 15: Number of name servers per domain over top million domains

We show the results in Table 6. Only 869,140 out of 2,394,475 (36.3%) name-servers that appear in the NS referral responses of the 1M domains are both in-bailiwick and include a corresponding IP address (glue-recrod). 1,525,335 (63.7%) name-servers are missing a

Measurement	Number
Requests	1,000,000
Answers	1,000,000
NXDOMAIN	20,025
SERVFAIL	20,110
NOERROR	959,865
CNAME Response	1,717
Empty Response	11,498
Domains with nameservers (valid)	946,650
Domains that all their NSs with glue (IP)	342,429
Domains that all their NSs w/o glue.	567,450
Total name-servers in answers	**2,394,475**
In-bailiwick name-servers (strict def.)	70,596
Out-of-bailiwick name-servers (strict def.)	2,323,879
In-bailiwick name-servers (wider def.)	1,081,876
Out-of-bailiwick name-servers (wider def.)	**1,312,599**
Name-servers with glue records	869,140
Name-servers w/o glue records	**1,525,335**

Table 6: Pervasiveness of authoritative nameservers with missing glue records over top million domains

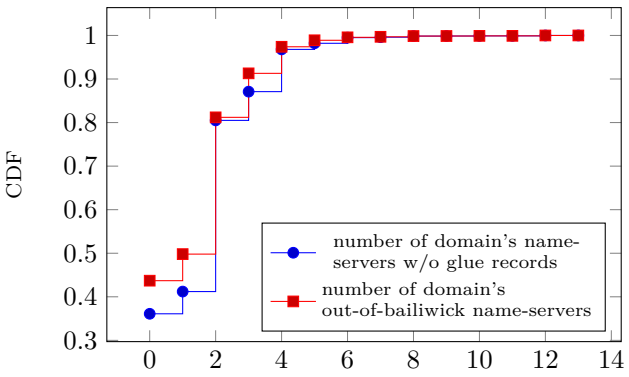


Figure 16: CDF of out-of-bailiwick nameservers per domain

corresponding glue record, from which 1,312,599 are out-of-bailiwick, showing that some *in-bailiwick* name-servers are not provided with their glue-records by their parent authoritative name-servers. Here we refer to the wider definition of *bailiwick*; the results show that most authoritative name-servers provide glue-records according to this definition. An additional note is that according to strict definition of *in-bailiwick*, we find that only 70,596 name-servers out of 2,394,475 (2.95%) are *in-bailiwick*, i.e., their name-servers names are within the domain name (for example, 'ns.example.com' as a name-server for the domain 'example.com').

The blue line in Figure 16 shows the distribution of the number of name-servers without a glue-record per domain. The majority of the domains (567,450 out of 946,650 domains with NOERROR responses, 59.94%) have *all* their name-servers received without a corresponding glue record (in the NS referral response from the TLD, or sometimes from an SLD). One reason for this high number of domains with *out-of-bailiwick* name-servers is that many domains outsource their DNS authoritative service to the same vendors. Out of the 1 million

we tested, 218,747 (21%) domains use ns.cloudflare.com and 129,789 use domaincontrol.com.

7 Related Work

Luo et al. [16] analyze the prevalence and characteristics of the NXDomain and water torture attacks. Using one month of real-world DNS traffic, they compare the attack behavior with DGA malware and disposable services.

Recently the DNS infrastructure is facing abuse by various entities which use it for applications for which it was not intended. In this case, a large volume of temporary domain names (aka disposable domains [10]) would be commonly used to benefit these services to communicate via DNS queries. A study [10] from large scale DNS traffic shows that 60% of all distinct resource records observed daily are disposable. Hao et al. [5] examine the negative impact of disposable domains on recursive caching. They propose a classification based on domain name features to increase the cache hit-rate.

Maury [18] presents a different attack that also exploits the delegations of name-servers in a referral response. However, the attack (called iDNS attack) PAF is at most 10x. In iDNS the attacker's name-server sends self-delegations (back and forth to the attacker's name-server) up to an infinite depth. A major difference from our work is that the glueless name-servers in the iDNS attack are never used against an external server such as a victim name-server. Some measures have been taken by different DNS vendors such as BIND and UNBOUND following the disclosure of iDNS described in [18], however these measures do not affect and do not weaken the NXNSAttack.

Wang [31] focuses on the DNS security implications of glue records. He describes how recursive resolver implementations such as BIND and Unbound treat glue records, but the focus is on cache poisoning vulnerabilities rather than the impact on the recursive performance, which is the focus of the current paper.

Muller et al. [23] perform a comprehensive measurement using the RIPE atlas to analyze how recursive resolvers select which name server to interact with, out of a set of multiple authoritative servers. The focus is on how and when the recursive resolvers *query* a set of multiple authoritative servers, while in this paper we extend the discussion and focus on how and when recursive servers *resolve* the IP addresses of a set of authoritative name-servers. In another work [22], Moura et al. analyze the root DNS service during a specific DDoS attack. However, the analysis refers to authoritative servers rather than recursive behavior. In a recent work [21], Moura et al. measure and show the impact of the caching and long TTL on dissecting DNS defenses during a DDoS attack.

8 Disclosure

A responsible disclosure procedure has been performed following the discovery of the NXNSAttack described in this paper. The following vendors and providers have been approached and have patched their software and servers, most of them according to our MaxFetch(c) approach: ISC BIND (CVE-2020-8616), NLnet labs Unbound (CVE-2020-12662), PowerDNS (CVE-2020-10995), CZ.NIC Knot Resolver (CVE-2020-12667), Cloudflare, Google, Amazon, Microsoft, Oracle (DYN), Verisign, IBM Quad9 and ICANN.

9 Conclusions

You never know what you might find when you go off looking for your lost donkey. Our initial goal was to investigate the efficiency of recursive resolvers and their behavior under different attacks, and we ended up finding a new seriously looking vulnerability, the NXNSAttack.

The key ingredients of the new attack are (i) the ease with which one can own or control an authoritative name-server, and (ii) the usage of non-existent domain names for name-servers and (iii) the extra redundancy placed in the DNS structure to achieve fault tolerance and fast response time.

We note that some of the possible remedies, such as various rate limiters are a double edge sword remedy, that a sophisticated attacker may use to deny service to legitimate clients, by hitting the limiters thresholds with malicious requests.

Notice that DoH (DNS over Http) is irrelevant to this paper because it deals with the communication channel between a client and its recursive resolver while the issues discussed in this paper are on the communications between the recursive resolver and the authoritative structure.

10 Acknowledgements

We would like to thank Michael McNally and Cathy Almond of ISC, Ralph Dolmans, Wouter Wijngaards and Benno Overeinder of NLnet Labs and Petr Spacek of NIC.CZ for their cooperation upon disclosing this issue, as well as Eyal Ronen and Yair Kaldor for their help with this project.

References

[1] PowerDNS. https://www.powerdns.com/, 2019.

[2] RFC 8499–DNS Terminology. https://tools.ietf.org/html/rfc8499, 2019.

[3] Akamai. Whitepaper: DNS Reflection, Amplification, and DNS Water-torture , 2019.

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, 2017.

[5] Yizheng Chen, Manos Antonakakis, Roberto Perdisci, Yacin Nadji, David Dagon, and Wenke Lee. DNS noise: Measuring the pervasiveness of disposable domains in modern DNS traffic. In *DSN*, pages 598–609. IEEE Computer Society, 2014.

[6] J. Damas, M. Graff, and P. Vixie. Extension Mechanisms for DNS (EDNS(0)), 2013.

[7] ISC DNSperf. Performance testing of recursive servers using queryperf, Oct. 2018.

[8] Shir Landau Feibish, Yehuda Afek, Anat Bremler-Barr, Edith Cohen, and Michal Shagam. Mitigating dns random subdomain ddos attacks by distinct heavy hitters sketches. *HotWeb*, pages 8:1–8:6, 2017.

[9] FireEye. Global DNS Hijacking Campaign: DNS Record Manipulation at Scale, August. 2019.

[10] Shuai Hao and Haining Wang. Exploring domain name based features on the effectiveness of dns caching. *ACM SIGCOMM Computer Communication Review*, 47(1):36–42, 2017.

[11] ISC. Bind: Internet systems consortium. https://www.isc.org/downloads/bind, May 2019.

[12] Mohammad Karami, Youngsam Park, and Damon McCoy. Stress testing the booters: Understanding and undermining the business of ddos services. In *Proceedings of the 25th International Conference on World Wide Web*, pages 1033–1043. International World Wide Web Conferences Steering Committee, 2016.

[13] Amit Klein, Haya Shulman, and Michael Waidner. Counting in the dark: Dns caches discovery and enumeration in the internet. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 367–378. IEEE, 2017.

[14] Marc Kührer, Thomas Hupperich, Jonas Bushart, Christian Rossow, and Thorsten Holz. Going wild: Large-scale classification of open dns resolvers. In *Proceedings of the 2015 Internet Measurement Conference*, pages 355–368. ACM, 2015.

[15] NLnet Labs. Unbound. https://nlnetlabs.nl/projects/unbound, 2019.

[16] Xi Luo, Liming Wang, Zhen Xu, Kai Chen, Jing Yang, and Tian Tian. A large scale analysis of dns water torture attack. In *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*, pages 168–173. ACM, 2018.

[17] Majestic. Top Million Root Domains List , May. 2019.

[18] Florian Maury. The idns attack. In *OARC 15*, 2015.

[19] Paul Mockapetris. RFC-1034 Domain Names-Concepts and Facilities. *Network Working Group*, page 55, 1987.

[20] Don Moore. Dns server survey, 2004.

[21] Giovane Moura, John Heidemann, Moritz Müller, Ricardo de O Schmidt, and Marco Davids. When the dike breaks: Dissecting dns defenses during ddos. In *Proceedings of the Internet Measurement Conference 2018*, pages 8–21. ACM, 2018.

[22] Giovane C.M. Moura, Ricardo de O. Schmidt, John Heidemann, Wouter B. de Vries, Moritz Muller, Lan Wei, and Cristian Hesselman. Anycast vs. ddos: Evaluating the november 2015 root dns event. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, pages 255–270, New York, NY, USA, 2016. ACM.

[23] Moritz Müller, Giovane C. M. Moura, Ricardo de O. Schmidt, and John Heidemann. Recursives in the wild: Engineering authoritative dns servers. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, pages 489–495, New York, NY, USA, 2017. ACM.

[24] Water Torture Nishida K. A Slow Drip DNS DDoS Attack on QTNet, May. 2019.

[25] Nominum. resperf performance tool manual , May. 2019.

[26] Radware. DNS: Strengthening the Weakest Link, 2018.

[27] Christoph Schuba. Addressing weaknesses in the domain name system protocol. *Master's thesis, Purdue University, West Lafayette, IN*, 1993.

[28] Secure64:. Water torture, a slow drip DNS DDoS attack, Feb. 2014.

[29] Joe Stewart. Dns cache poisoning–the next generation, 2003.

[30] Talos. DNSpionage Campaign Targets Middle East, August. 2019.

[31] Zheng Wang. The availability and security implications of glue in the domain name system. *CoRR*, abs/1605.01394, 2016.