NRDelegationAttack: Complexity DDoS attack on DNS Recursive Resolvers [*]

Yehuda Afek [†]
Tel-Aviv University
afek@tauex.tau.ac.il

Anat Bremler-Barr [‡]
Tel-Aviv University
anatbr@tauex.tau.ac.il

Shani Stajnrod
Reichman University
shaaniba93@gmail.com

Abstract

Malicious actors carrying out distributed denial-of-service (DDoS) attacks are interested in requests that consume a large amount of resources and provide them with ammunition. We present a severe complexity attack on DNS resolvers, where a single malicious query to a DNS resolver can significantly increase its CPU load. Even a few such concurrent queries can result in resource exhaustion and lead to a denial of its service to legitimate clients. This attack is unlike most recent DDoS attacks on DNS servers, which use communication amplification attacks where a single query generates a large number of message exchanges between DNS servers.

The attack described here involves a malicious client whose request to a target resolver is sent to a collaborating malicious authoritative server; this server, in turn, generates a carefully crafted referral response back to the (victim) resolver. The chain reaction of requests continues, leading to the delegation of queries. These ultimately direct the resolver to a server that does not respond to DNS queries. The exchange generates a long sequence of cache and memory accesses that dramatically increase the CPU load on the target resolver. Hence the name non-responsive delegation attack, or NRDelegationAttack.

We demonstrate that three major resolver implementations, BIND9, Unbound, and Knot, are affected by the NRDelegationAttack, and carry out a detailed analysis of the amplification factor on a BIND9 based resolver. As a result of this work, three common vulnerabilities and exposures (CVEs) regarding NRDelegationAttack were issued by these resolver implementations. We also carried out minimal testing on 16 open resolvers, confirming that the attack affects them as well.

[*]Supported by a grant from the Blavatnik Interdisciplinary Cyber Research Center, Tel Aviv University.

[†]Member of the Checkpoint Institute of Information Security.

[‡]The majority of this research was carried out while the author was at Reichman University. Member of the Checkpoint Institute of Information Security.

1 Introduction

In a system like the DNS, what is the worst case scenario for resource consumption while processing a single request? Malicious actors carrying out distributed denial-of-service (DDoS) attacks are interested in requests that consume a large amount of resources and provide them with effective ammunition. DNS system designers on the other hand, strive to ensure that no request uses more than a very limited amount of resources.

The DNS system has been, and remains, an attractive prime target for DDoS attacks. Even back in 1987, while more concerned about human errors than malicious activity, Mockapetris wrote in his initial design RFC1034 [24]:

> The recommended priorities for the resolver designer are:
>
> 1. Bound the amount of work (packets sent, parallel processes started) so that a request can't get into an infinite loop or start off a chain reaction of requests or queries with other implementations EVEN IF SOMEONE HAS INCORRECTLY CONFIGURED SOME DATA.

Even after Mockapetris' clear warning about the overuse of the resolver resources, attackers still found a way to impose excess strain on the resolver resources. Most recent attacks, such as the NXNSAttack [5], water-torture, and other flood attacks, abuse the resolver resources to trigger a large number of queries between the resolver and the hierarchy of authoritative servers. The chain reaction of events revealed in this paper leads to a complexity vulnerability in which an attacker request, together with a particular malicious authoritative response, overloads memory and compute resources on the resolver. Interestingly, the mechanisms introduced to mitigate a recent DDoS attack [5] play a key role in our DDoS complexity amplification attack.

The chain reaction of events at the heart of the NRDelegationAttack we present starts with a large referral response received from a malicious authoritative server. Two of the

mechanisms in the chain reaction are:

1. When the server resolves each NS name in the referral response, the victim resolver is delegated to a server that does not respond to DNS queries.

2. In response to each such delegation, the resolver **re**starts the resolution process, ignoring the state of the current resolution, including some important safety limitation counters. As a result, the resolution process restarts repeatedly until either a timeout or the restart-limit is reached. At this point, a FAIL response is saved in the cache and returned to the client.

We measured the effect of a single NRDelegationAttack attacker sending 15,000 malicious packets per second and using a referral list that contains 1,500 NS names, on BIND9 [18], Unbound [29], and Knot [13] implementations. Our evaluation showed that as a result of the attack, the throughput for benign users is reduced by a factor of 609, 58, and 388 respectively. In addition, we tested the NRDelegationAttack on the top 16 open resolvers and demonstrated its impact in the wild. Our results show that all the resolvers tested were affected to a certain extent by the attack. To get more comprehensive measurements, we conducted a detailed analysis on a BIND9 server implementation.

This work includes suggestions for minimal modifications that can be made to the recursive resolver algorithm to mitigate the NRDelegationAttack and significantly limit the resources used by the resolver during the attack. We tested our mitigation technique and proved its efficiency against the NRDelegationAttack.

After reviewing the mechanisms that play a role in the NRDelegationAttack chain reaction in Section 2, we discuss the threat model in Section 3. In Section 4, we show how the different pieces of the model interact to form the NRDelegationAttack. In Section 5, we analyze the complexity factor of the NRDelegationAttack on BIND9. Section 6 covers its impact on recursive resolver implementations and on open resolvers. Alternative mechanisms to mitigate the NRDelegationAttack are suggested in Section 8 and related work is shared in Section 9. We review the responsible disclosure procedure in Section 7, and draw conclusions in Section 10.

2 Basic DNS Mechanisms

2.1 SLIST

The "SLIST" mechanism is presented in RFC-1034 [24]). It is used as a scratch pad memory in the resolver implementation to record the status of each intermediary name server resolution that is performed while processing a client query. Before starting to resolve any new NS name(s) (e.g., from the referral list) the resolver looks at the SLIST to see whether a previous resolution makes this new one superfluous; if necessary, it adds the new name(s) to the SLIST and starts its resolution process. In BIND9, this data structure is called address-DB (ADB) [17]. Further details and an example of how the SLIST is used are given in Appendix A.

2.2 Delegation Response

To answer a resolver query, an authoritative server can decide to delegate the resolution work to another name server. For example, this occurs when .org delegates the resolution of www.usenix.org to .usenix.org, or when a resolver gets a referral response (RR).

Motivated by fault-tolerance and low latency response requirement, referral response is a multi delegation response. It contains a list of name-servers and the resolver continues the resolution by delegating its query to any one of the name servers in the list. The number of name servers in the RR can be large and may not always include their corresponding IP addresses, known as glue records. One reason why the referral-list response does not include glue records is because authoritative servers are not allowed to provide IP addresses for domains whose origin resides outside their zone; these are known as *Out-of-Bailiwick* name servers [16]. This policy was created to protect the servers from DNS poisoning attacks.

The important aspect for the NRDelegationAttack is that any delegation response will trigger a "restart" event in the resolver algorithm.

2.3 Restart Event

In many cases, the resolution process in a resolver is delegated to a different name server, in which case it restarts and continues the resolution at the new name server. Consequently, a restart event was introduced in RFC-1034 [24] to allow this.

The "new authoritative" server to which it is redirected is recorded in the ADB (see Section 2.1). In response to the restart, the resolver clears and resets a few flags. Among them is the "No_Fetch" flag, which was introduced to mitigate the NXNSAttack (see Section 2.4). After clearing the flags, the resolver continues the resolution of the name-at-point.

2.4 Referral Response Limit

A recent DNS DDoS attack, named the NXNSAttack [5], employs a malicious NS referral response (Section 2.2) with a long list of non-existent NS names. The NXNSAttack is amplified and extremely effective because the resolver starts simultaneously resolving all the names in the referral list immediately upon receiving the referral response. To mitigate this attack, a new limit was introduced on the number of NS names being resolved from a referral response, per client request. Upon receiving a referral response with many NS names, the resolver starts resolving only k names (the referral

limit) per client query (in BIND9 and KNOT implementations k equals 5, and in UNBOUND implementation 6). If all k attempts return NXDOMAIN, the resolution is aborted and an NX response is returned to the client.

This limit is called the referral-response-limit ("NS_RR_LIMIT" in BIND9). Throughout this paper, we use the term "referral-limit". A special flag, denoted by "DNS_ADBFIND_NOFETCH" or "No_Fetch," is set when the limit is reached. As shown in Section 4 and analyzed in Section 5.2, this limitation plays a key role in the NRDelegationAttack.

3 Threat Model

To mount an NRDelegationAttack on a recursive resolver, an attacker needs to:

1. Control one or more clients from which it can issue the malicious queries.
2. Control an authoritative name server configured to respond with a particular referral response that will be used in Phase II of the attack.
3. Supply a list of server domain names, with servers that do not respond to DNS queries. Or, in some variants, the attack will control a "delegation authoritative" that is configured to respond with the IP address of a server that doesn't respond to DNS queries (see Section 4.2).

Using the authoritative server, the attacker responds using NS referral responses in which it controls the response content, such as the number of NS names, the names themselves, and any additional information. Authoritative name servers can be easily and cheaply acquired by first buying and registering new domain names. As part of our experiments, we purchased several domain names for less than $1 each in less than 5 minutes. These domain names can be dynamically associated with any authoritative server on the Internet. Alternatively, as shown by recent DNS hijacking attacks [14, 23], attackers can compromise the DNS operators' credentials and manipulate zone-files, sometimes even gaining access to their registrar records.

4 NRDelegationAttack

The attack is formed based on the specific interplay of several DNS mechanisms in the resolver, some of which were introduced only recently. In this section, we explain these key mechanisms and their interplay before describing a variant of the attack in Section 4.1, Figure 1 and Figure 2. The attack proceeds as follows:

1. Following a malicious query, the malicious authoritative server sends a large referral response (LRR, see Section 2.2) to the target victim recursive resolver. The referral list contains n (≤ 1500) different domain names, with no glue records. The recursive resolver will try and resolve the query with any one of the n NS names in the list. As a first step, the resolver has to resolve these NS names.
2. Before starting to resolve any of the NS names in the LRR, the recursive resolver checks in the cache and the ADB to see if any of the n names already has a resolution that it can use. When n is large, this operation consumes an enormous amount of memory and CPU resources; we denote its compute cost by $CC(n)$. This is the core source for the complexity of the NRDelegationAttack, which is significantly amplified by being repeated in a loop. This loop is due to the NXNSAttack mitigation mechanism, which forces the n names to be processed in chucks of k names at a time, as explained in the next 4 steps.
3. This kind of LRR, with n nonexistent (NX) domain names and no glue records, was used in the "NXNSAttack" [5]. In the NXNSAttack, when the resolver receives the LRR, n resolution processes start simultaneously, essentially flooding both the victim recursive resolver and the authoritative appointed by the NX domain names in the LRR.
4. To mitigate the NXNSAttack, a new 'referral limit' (e.g., $= 5$) denoted by k was imposed on the number of referrals that can be resolved out of the n names for each client query; the value of k may differ slightly in different implementations (see Section 2.4). If all k attempts return NXDOMAIN or FAIL, the resolution is aborted and an NX response is returned to the client, thus cutting down the amplification factor of the NXNSAttack. In the following section, we compare the effects of the NRDelegationAttack to that of the NXNSAttack. We refer to the resolver versions before the NXNSAttack as pre-NXNS and those after the NXNSAttack disclosure as NXNS-patched versions.
5. In the NRDelegationAttack, rather than n nonexistent NS names, the attacker uses n **existing** NS names without glue records. Each such name is a name of (or leads to) a server that is non-responsive (NR) to DNS_queries server (see Section 2.2), hence the name "NRDelegation". Upon receiving such an LRR, NXNS-patched resolvers will initially resolve k NR names, and will try to ask each of these servers to resolve the query.
6. As explained in Section 2.2, each delegation response triggers a "restart" event. Hence, k restart events are triggered when each of the first k delegation responses are received by the recursive resolver (NXNS-patched version).
   Each restart event clears the resolver indication that the

"referral-limit" was reached for this LRR. As a result, the resolver starts the process from the beginning, except this time it starts with the resolutions of k other names from the LRR. Consequently, each restart caused by each delegation initiates the resolution of k additional names from the LRR. This ultimately results in k more delegations. The process enters an expanding loop in which k^2 resolutions of NS names from the LRR are started in the subsequent iteration of the loop. The bad news is that after each of the restart events, as long as there are names in the LRR whose resolution has not yet started, the recursive resolver again checks whether any one of the n names in the LRR already has a usable resolution in either the cache or the ADB (see Step 2). The resulting compute load is $(k+k^2)\cdot CC(n)$, which is the core reactor of this complexity attack. In principle, after the next iteration, the accumulated complexity is $(k+k^2+k^3)CC(n)$ but no more than a maximum of n.

7. In pre-NXNS versions, the n resolutions of each of the domain names from the LRR start at once. Thus, although it issues many queries, the load complexity on the resolver is much lower than in the NRDelegation attack on NXNS-patched versions because the resolver flags the LRR after going through all of the LRR domain names. It will not repeat Step 2 as it does in the patched versions. In addition, the NXNSAttack will have a larger packet amplification factor because each authoritative responds and exchanges packets with the resolver. In the NRDelegationAttack, the non-responsive servers simply do not respond. On the other hand, the NXNSAttack is terminated faster; once an NX response is received by the resolver, it ends the client resolution with an NX response. In the NRDelegationAttack, the resolver keeps waiting in the hope that one of the (non-responding) servers will respond. While waiting, the resolver also keeps processing additional NS names from the LRR until it hits the safety threshold counters (see below). As a result, the NRDelegationAttack on pre-NXNS resolver versions has nearly twice the CPU complexity of the NXNSAttack on the same version (292 million instructions vs. 180 million per attacker query), while on the NXNS-patched version it is 38 times more complex (3415 million instructions vs. 90 million per attacker query, see Figure 3).

8. In real implementations of the recursive resolver, either pre or patched NXNSAttack, there are safety limitation counters on the number of restarts allowed in each phase and in total (= 100 restarts). These limitations prevent the process from continuing exponentially until reaching n. However, these safety bounds are high (e.g., 100 restarts before the resolution aborts) and do not restrain the power of the attack, which is still enough to deny service at the target.

9. Because there are several different delegation methods in the DNS system, the NRDelegationAttack has different variants, depending on the delegation method being used. In the most straightforward version, but not the simplest to craft, the attacker can use a list of existing domain names that are not responsive to DNS queries. For example, this may include different web servers such as www.google.com, www.usenix.org, etc. Alternatively, 'broken' or stale domain-to-IP mappings can be used as domain names in the referral list. Using a modular method, the attacker can also use his own delegation server, in which several different response options direct the resolver to a resource IP that is non-responsive to DNS queries. We refer to this as the "delegation server" variant. For example, the attacker can use an "NXDOMAIN-Redirection" [33] response (see Appendix B) in which, instead of replying with NX to non-existent domain names, the delegation authoritative redirects the resolver to an NR resource. Additional delegation options are given in Section 4.2 and Table 1.

4.1 NRDelegationAttack Construction

The following is a detailed step-by-step description of the attack scenario presented in Figure 1; this is the most involved variant of the attack. The attack has three major phases:
Phase I: Victim resolver receives a large referral response (LRR).
Phase II: Victim resolver starts processing the LRR.
Phase III: Victim resolver loops through delegation response and restarts Phase II.

Each phase has the following steps:
Phase I:

1. Attacker's client queries the victim resolver about xyz.referral.com (Step 1 in Figure 1).

2. Victim resolver queries referral.com (Step 2 in Figure 1).

3. Authoritative referral.com responds with a large referral response (LRR) (Step 3).

Phase II:

1. The resolver looks up each of the n names to see if any one of them already has a resolution it can use. This is a total of $2n$ lookups. This step is the core source of the high complexity, especially since it will be repeated many times, as explained in the next phase (Step 4 in Figure 1).

2. The resolver starts resolving referral-limit names in the LRR, e.g., 5 in BIND9 implementation (Step 5 in Figure 1). In the first execution of this step, it begins the

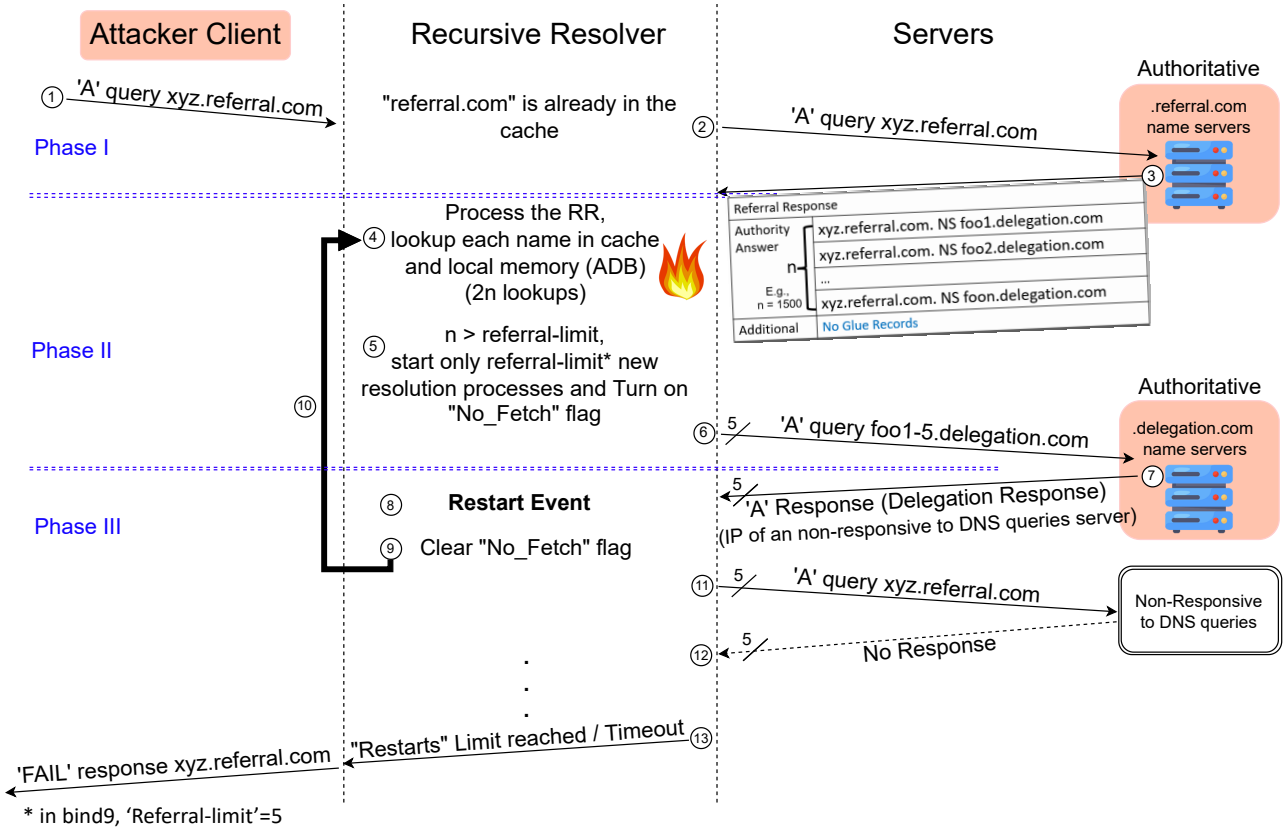


Figure 1: "Delegation server" variant of the NRDelegationAttack flow overview, focused on the resolver requests and responses.

resolution of the first referral-limit NS names from the LRR. Each subsequent execution begins resolving the next referral-limit NS names, until the resolution of all the names in the LRR has been invoked.

3. The resolver turns on the No_Fetch flag (Step 5), which is part of the NXNSAttack mitigation and is intended to limit the number of referred NS names resolved per client query.

4. The resolver issues the query for these referral-limit NS names (Step 6).

Phase III: Victim loops multiple times through referral response-restart loop.

1. Each such name leads the resolver to an authoritative, which responds with a server IP address that is non-responsive to DNS queries and issues the delegation response (Step 7 in Figure 1).

2. Upon receiving a delegation response, the resolver triggers a restart event and restarts the resolution for the LRR (Step 8 in Figure 1).

3. The victim resolver clears the "No_Fetch" flag (Step 9), because retaining it could break the resolution of a valid domain name. For example, if domain ".com" has more than "referral-limit" delegation options, and domain ".example.com" also has delegations that need to be resolved, then the resolver cannot preserve the "No_Fetch" flag between the ".com" and ".example.com" resolutions.

4. The victim resolver does the following in parallel:

   (a) Restarts the RR-list processing (Step 10 in Figure 1).

   (b) Queries the resources received in Step 7 and receives no helpful information in return (i.e., no response at all, as shown in Steps 11-12 in Figure 1).

5. Following the above restart, go to Phase II.

6. The victim loops through Phases II and III until a timeout or restart limit is reached and a 'FAIL' response is returned to the client (Step 13 in Figure 1).

The source of the heavy load produced by the attack is mostly from the numerous executions of Step 1 in Phase II. (In BIND9, it is at most 100 times.) If the number of names n in the referral list is large, e.g., $1,500$, then each execution of this step costs $3,000$ memory accesses and at least

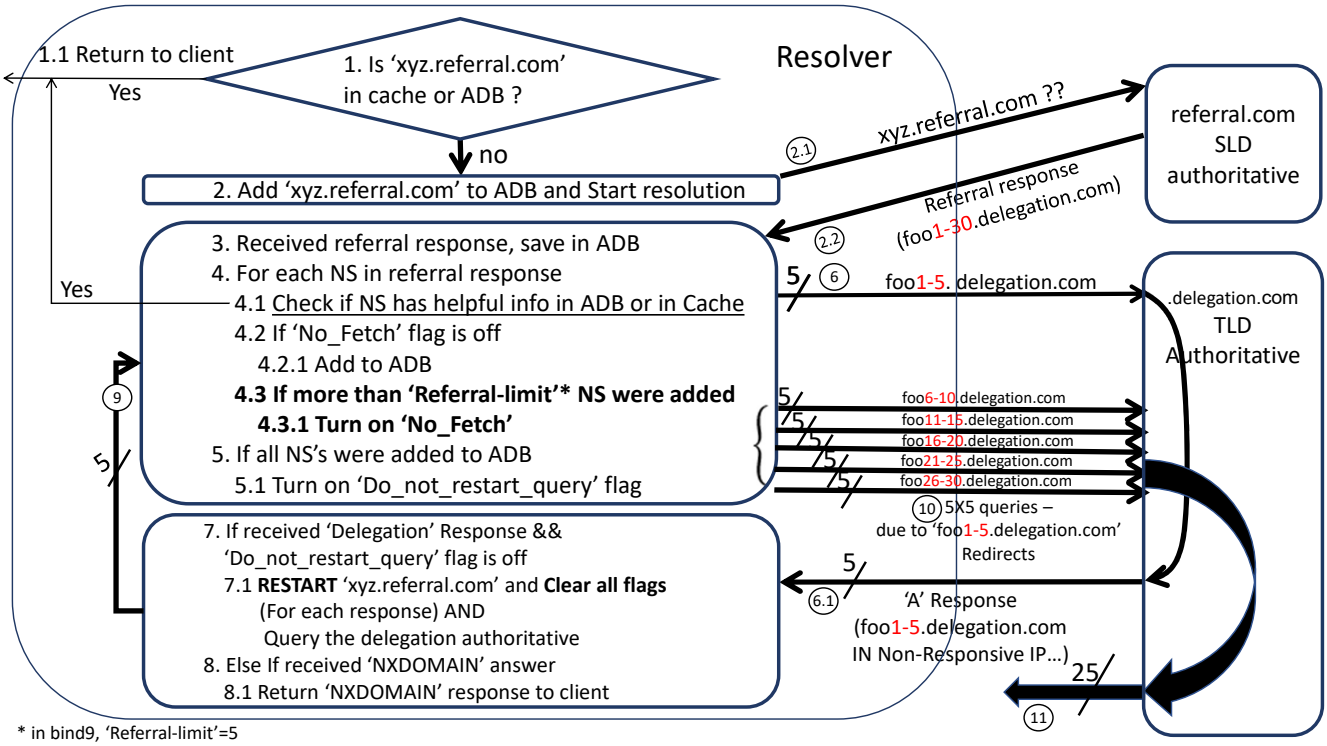


Figure 2: Delegation server variant of the NRDelegationAttack flowchart; the resolver resolution algorithm during the attack.

5,600 times more CPU instructions relative to a benign query (1,100,000,000:195,000 instructions). The NRDelegationAttack amplification factor is given in Section 5.2. Figure 2 presents a flowchart of the relevant parts of the resolver algorithm.

Although the NRDelegationAttack is mainly explained here using a BIND9 implementation, we also tested it on Unbound, and Knot implementations, and 16 open DNS resolvers (see Section 6.2 and Section 6.3). Our evalutions show that all the resolvers under test suffer degradation in their performance under the NRDelegationAttack. This occurs primarily due to the excessive use of the resolver resources when it receives the NRDelegationAttack malicious query (e.g., the number of cache lookups done per one malicious query). As part of the responsible disclosure discussion with the vendors, excessive use of the resolver resources indeed proved to be the main reason for the impact of NRDelegationAttack on the resolvers.

4.2 NRDelegationAttack Delegation Options

It is important that the referral list received at the end of Phase I includes only domain names leading to a server that is non-responsive to DNS queries. Otherwise, if the resolver receives useful resolution, it may obtain a reply to the query thereby ending the attack chain reaction and preventing the repetition of Phase II. There are several delegation options that can be used to ensure the NRDAttack will succeed, as presented in Table 1.

In some resolver implementations there may be limitations regarding the fetches allowed per zone or IP address; these limitations can be bypassed using wildcard records. In this case, the attacker can either look for wildcard records where the IP address does not host a DNS server, or use arbitrary subdomains under his "delegation server" and direct them to different IP addresses that do not host a DNS server.

We tested all the versions mentioned above using our InnerSimulator, described in Section 5.1.

5 NRDelegationAttack Complexity Factor

In this section, we analyze and measure the NRDelegationAttack complexity amplification factor.

5.1 Isolated Lab Setup

The experimental setup includes a client, a resolver and three authoritative name servers. The resolver is a BIND9 recursive resolver with both the NXNSAttack patched version (BIND9 version 9.16.6) and a pre-NXNS version (BIND9 ver-

Variant (Response in Step 7)	Pros	Cons	DNS Mechanisms used to Arm the Attack	RCODE Received in Step 7 (Example)
NXDOMAIN-Redirection	Attacker controls the whole chain	Attacker needs to have a "delegation" server	QNAME Minimisation, CNAMEs, Wildcard Records	'A' (11.22.33.44)
NR-Delegation	Attacker can use some of ".com" servers OR use his own sub domain and control the whole chain	Not all ".com" servers can be used OR attacker needs to have a "delegation" server	QNAME Minimisation, CNAMEs, Wildcard Records	'A' (11.22.33.44)
NR-DNS names	The attacker may use real domains which are not in his control	Attacker needs to have a "delegation" server and create such domain lists	QNAME Minimisation, CNAMEs, Wildcard Records	'NS' (google.com)
'Broken' or stale domain-to-IP mappings	No need for a "delegation" server	Attacker needs to create such domain lists	Wildcard Records (in some cases, only if the domain found has it configured)	'A' (11.22.33.44)

Table 1: NRDelegationAttack delegation options.
* NR-DNS is short for "Non-Responsive to DNS queries" server

sion 9.16.2). The three authoritative name servers include: a 'root' authoritative name server, an attacker authoritative name server, and a malicious redirection authoritative. Most of our tests were carried out on a BIND9 version 9.16.6 resolver compiled to work with our local 'root' authoritative name server. We implemented our authoritative name servers with Name Server Daemon (NSD) [30] version 4.3.3. The clients were deployed on the same machine, which was configured to send DNS queries directly to our local recursive resolver. The setup configuration, referred to as Inner-Emulator, is available in [34].

5.2 NRDelegationAttack Complexity Factor

As noted previously, most of the heavy load produced by the attack stems from the numerous executions of Step 1 in Phase II, which is recursively called in Phase III (see Section 4). We used the Inner-Emulator setup and Valgrind tool [4] with a BIND9 implementation to compute the complexity factor function ($cost(n)$). This represents the number of machine instructions executed to resolve one malicious NRDelegationAttack request, where n is the number of NS names in the referral-list. The major term in this function is the cost of executing Step 1 in Phase II multiplied by the number of times it is executed:

Let *benignCost* be the number of instructions executed to resolve one benign query; the average number over different benign queries is $BenignCost = 195,000$ instructions.

Let $Rstrts(n)$ be the number of restart events raised in a resolution as a function of n. The value of $Rstrts(n)$ is at most 100, at which point the resolution is aborted. $Rstrts(n)$ is measured as follows:

$$Rstrts(n) = \begin{cases} 1 \le n \le 10: & 3 \\ 11 \le n \le 15: & 3+(n-10) \\ 16 \le n \le 20: & 8+5*(n-15) \\ 21 \le n & min(33+25*(n-20), 100) \end{cases}$$

$RR\_Sent(n)$ is the total number of NS name queries sent by the resolver as a function of n and is bounded by the restarts limit ($= 100$). When the restarts limit is reached, the resolution process is aborted. $RR\_Sent(n)$ is measured as follows:

$$RR\_Sent(n) = \begin{cases} 1 \le n \le 35: & n \\ 36 \le n: & 35 \end{cases}$$

Note that the restarts limit is reached when $n = 23$. However, because the resolver is multi-threaded, the restarts limit is reached concurrently with other threads that are looking up referrals for NS names in the ADB. As a result, there is some variance in the number of NS names that are inserted into the ADB and have been queried, when the restarts limit is reached. That is, RR_Sent(n) when $n \ge 30$, may be due to race conditions that occur between n being 25 and 35.

Let $ResolutionLoop(n)$ be the cost of processing the referral-response (Phase II in Section 4). Notice its high dependency on n, even if $n >> 35$. Our measurements show that $ResolutionLoop = 21,000$ in BIND9 for each referral in the referral list.

$QSentCost(n)$ refers to the number of instructions executed when queries are sent out for the NS names in the resolution process and a corresponding response is received. Its value is as follows:

$$QSentCost(n) = RR\_Sent(n) * 2 * 21,000$$

Note that 21,000 is the approximate number of instructions it takes to send one query and to receive its response. This number is multiplied by the number of referrals queried ($RR\_Sent(n)$) and by 2 because each referral is queried in both IPV4 and IPV6. Putting it all together:

$$Cost(n) = Rstrts(n) * ResolutionLoop(n) + QSentCost(n)$$
$$= Rstrts(n) * n * 21,000 + RR\_Sent(n) * 2 * 21,000$$

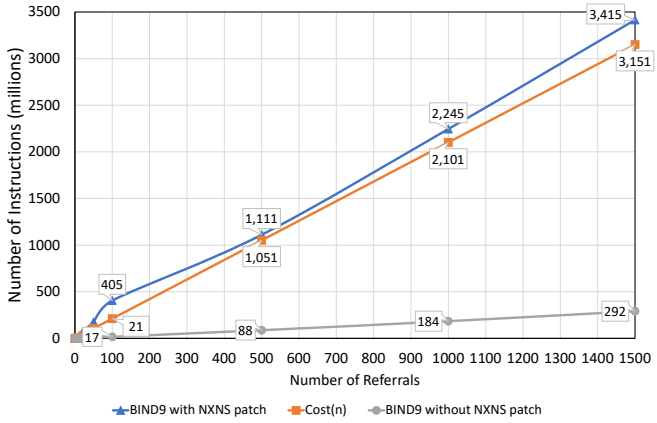


Figure 3: Instructions executed on the resolver processor per one malicious client request (in millions), tested on BIND9 with NXNS patched and non-patched versions, compared to our NRDelegationAttack instructions prediction function.

We measured the resolver server's actual performance with Valgrind [4] using different values of n, and compared the results with $Cost(n)$. We tested the NXNSAttack patched and non-patched versions of BIND9 (version 9.16.2 and 9.16.6, respectively). The results are given in Figure 3. As can be observed, $Cost(n)$ closely predicts the number of instructions executed for each malicious attacker query. The instructions' amplification factor is at least 5,600 relative to a benign query (1,100,000,000:195,000 instructions). In addition, the number of instructions on a pre-NXNSAttack version of BIND9 is significantly lower than the number on an NXNS-patched version (1,100,000,000:88,000,000), as explained at the beginning of Section 4.

6 NRDelegationAttack Measurements

In the previous section we measured and analyzed an NRDelegationAttack with one malicious query in a closed setup disconnected from the Internet. Here we test it on a cloud setup and on open resolvers in the network. We measured the effect of different-sized attacks on the resolvers' (BIND, Knot, and Unbound) performance, and a weak attack on open resolvers. All of this was executed while taking into account ethical considerations.

6.1 Cloud Setup

Our cloud setup resided in the Azure cloud and included the different attacker DNS servers, along with benign users that make requests in parallel. This allowed us to evaluate the impact of the attack's effect on benign users as shown in Figure 4. The following machines were used:

1. Resolver machine with either BIND9 (9.16.6), Knot (5.1.3) or Unbound (1.13.1) resolvers; all were NXNSAttack patched versions.
2. Two client machines, a benign client and an attacker client, each equipped with a Resperf tool [31].
3. Authoritative server that is used to craft the malicious referral-response (referral.com in Figure 1).
4. Two machines simulating the authoritative (and its backup) that owns the domain referred to by the NS names in the referral response (e.g., delegation.com in Figure 1). This authoritative was configured to respond with different delegations to non-responsive DNS query servers e.g., NXDOMAIN-Redirection (see Appendix B).
5. Two machines simulating the authoritative and the backup to which benign users are referred.

We placed the client, resolver, and authoritative servers in the same Azure region so our measurements would not be impacted by any significant Internet delays. All the machines used in the cloud-setup have Intel Xeon Platinum 8272CL with 4 virtual CPUs, 16GB RAM, and the Ubuntu 18.04 operating system.

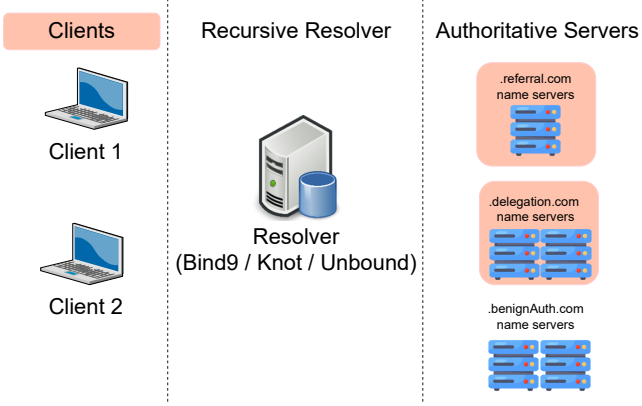


Figure 4: Cloud-setup testing environment simulation

6.2 Resolver Throughput Under NRDelegationAttack

In this section, we analyze the impact of an NRDelegationAttack on benign queries in a resolver that is under attack.

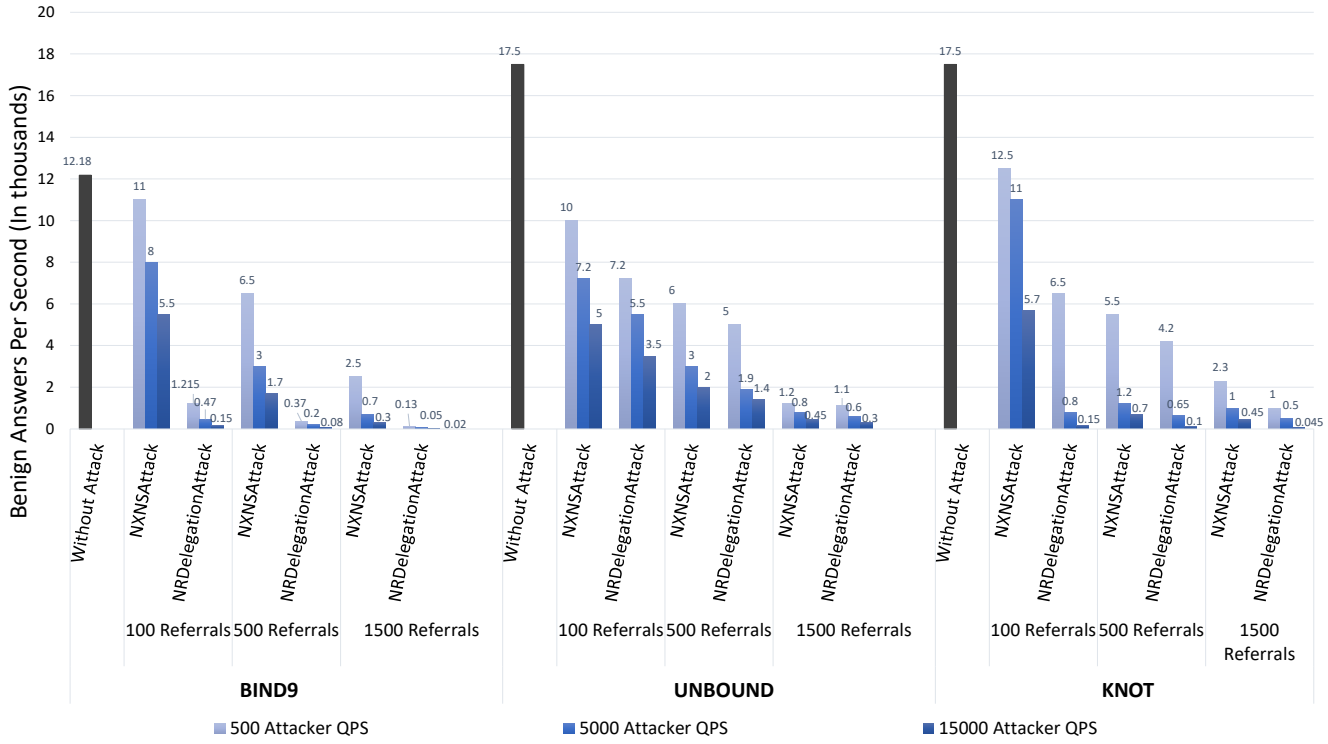


Figure 5: Resolver throughput tested on NXNSAttack patched versions of BIND9, Knot, and Unbound implementations under and without either NRDelegationAttack or NXNSAttack.

We tested BIND9, Unbound, and Knot implementations, all with NXNS-patched versions, with different attack scenarios, i.e., with different sizes of referral response. We used *two* Resperf tools [31]: the first simulates the attacker and issues queries each time at a fixed rate, and the second tool ramps up the benign user requests until things start to fail. All this is done in the cloud-setup. For each combination of attack parameters and rate, we tested the impact of both attacks: NRDelegationAttack and NXNSAttack.

As shown in Figure 5, all resolver implementations exhibited a significant performance degradation in their throughput measurements during the NRDelegationAttack. The impact of the NRDelegationAttack is significantly more severe than that of the NXNSAttack, with BIND9 having a dramatically lower throughput during the NRDelegationAttack compared to the NXNSAttack. Furthermore, the longer the malicious referral-list, the bigger the impact of the NRDelegationAttack on the tested resolver. This is because all the names in the referral-list are searched for in the ADB and the resolver cache, even if they have no resolution process. We also measured the attack's impact with different attacker QPS rates. These results demonstrate that as the malicious QPS increases, the resolver throughput decreases in all the tested implementations. We also observed that increasing the length of the referral list is more destructive to the resolver than increasing the attack rate by same proportion.

6.3 NRDelegationAttack Impact on Open Resolvers

In this section, we demonstrate the effect of the NRDelegationAttack in the wild on 16 popular open resolvers. To test the attack on real open resolvers, we used our Azure cloud-setup.

We were doubly restricted when testing the NRDelegationAttack on real open resolvers. First, we could not cause any harm to any of the open resolvers, as per the ethical statement about these measurements given in Section 6.4. Second, we could not measure the resolver itself. We could only make an indicative measurement of the latency and number of referral resolution attempts ("sent" in Table 1) per attacker request for each open resolver.

To avoid any disturbance of the open resolvers' operations, we used a rather weak version of the attack that had only 1 malicious request with 20 names in the referral-list response configured in the malicious authoritative. We repeated each of our tests 5 times, spread over some period of time.

The summary of our tests is presented in Table 2. As can be seen in the table, even when compared to the NXNSAttack, all the open resolvers tested show a significant performance degradation during our weakened version of the NRDelegationAttack. Our client's DNS timeout value was 15 seconds; if there was no response within that period, we received a

client-timeout.

The impact of the NRDelegationAttack is visible in both an increase of the resolver latency and the number of referrals sent to our authoritative servers. In most of the resolvers tested, the number of referrals that were sent during the NRDelegationAttack is much higher than during the NXNSAttack on an NXNS-patched resolver. This could be caused by the restart events that occur during the NRDelegationAttack but we do not know the implementation details of the open resolvers. Moreover, all the resolvers tested showed a significant increase in latency when responding to the attacker's request. In some cases, we even received a client-timeout and got no response. While the attack took place on a BIND9 resolver in the cloud-setup environment, we saw a correlation between the attacker request latency, the amount of resources used on the resolver, and the benign users' latency, as shown in Table 2.

The above experiments suggest that the NRDelegationAttack can impact the performance of open resolvers, even with a weakened version of the attack.

6.4 Ethical Statement

As mentioned before, when testing the NRDelegationAttack on real open resolvers, we could not cause harm to any of the open resolvers. Consequently, we could only take an indicative measurement using a weakened version of the NRDelegationAttack. Each test point on an open resolver sent only one request that initiated a response of at most 20 referrals. This was repeated 5 times at different points in time to ensure the accuracy of the measured latency. To verify that our tests did not harm the open resolvers, we first tested the weakened version of the NRDelegationAttack on the resolver in our internal setup. According to these tests, each such request is equivalent to at most 100 benign requests. Moreover, we notified the owners of all the open resolvers used in the responsible disclosure procedure.

7 Responsible Disclosure Procedure

Following the discovery of the NRDelegationAttack, we initiated a responsible disclosure procedure and corresponded with 20 different vendors and open resolver implementations.

To improve and synchronize the process, the DNS community "DNS.OARC" has created a specific channel in dns-oarc.net/community/channels and invited DNS related vendors and SP's. In this channel we shared our technical report and corresponded with different vendors and SP's including ICANN, Cisco, PowerDNS, Google and others. In addition we continued to correspond with several parties one-on-one through encrypted email. Furthermore, we have shared on the OARC channel our cloud setup with instructions on how to test NRDelegationAttack using the setup.

Three CVEs were issued, (CVE-2022-2795 [1]) by BIND9, (CVE-2022-40188 [3]) by Knot, and (CVE-2022-3204 [2]) by Unbound. The three vendors responded swiftly and cooperated to patch their software and servers, most of them by limiting the number of referrals processed or the cache searches per request. See Section 8 for the details of the mitigation techniques and the performance measurements on the vendors patched versions. Here is a quote from one of the large parties in the disclosure, which rated the original report as having high severity: "By flooding the target resolver with queries exploiting this flaw, an attacker can significantly impair the resolver's performance, effectively denying legitimate clients access to the DNS resolution service."

8 NRDelegationAttack Mitigation Options

We present several approaches to mitigate and reduce the effect of the NRDelegationAttack. The NRDelegationAttack builds upon very basic resolution mechanisms such as delegation and referral response. Therefore, when mitigating the attack, special care must be taken to ensure that the successful resolution of benign queries is not affected in any way. The following are examples of such techniques.

Consider only k of the NS names in the referral response: For all intents and purposes, the resolver considers only the first k (e.g., 20) NS names from the list of names in the referral response. Nevertheless, the entire list is kept in the cache unless the resolution results in an IP address. Only the first k names are looked up each time in the ADB and cache, and the resolver attempts to resolve at most k names. While this approach does not provide complete mitigation, it significantly reduces the effect of the NRDelegationAttack as shown in Figure 6. This is the easiest approach to implement since it requires minimal changes in the resolver implementation. The approach came up in the discussions we had with some of the vendors in the responsible disclosure procedure and some of them use this technique (BIND and Knot). As noted in [5], the top million domains have an average of 2.52 NS names in their corresponding RR, and 99% of them have less than 7. Because many root servers respond with 13 NS names in the RR, $k = 20$ was chosen as a safe number that is unlikely to affect the resolution of any benign name. Notice that had this mitigation technique been applied to an unpatched against NXNSAttack version (pre-NXNS version), then it would reduce the effect of the NXNSAttack. However, the limit set in the NXNSAttack mitigation (see Section 2.4) is more restrictive (send at most 5 NS name resolutions per client query) than the limit set here ($= 20$). Thus applying this mitigation on an NXNSAttack patched version is not expected to effect the behavior under an NXNSAttack. Except that the

Open Resolver			NXNS Attacker			NRDelegationAttack Attacker		
Name	IP	Benign Query Latency (msec)	Latency (msec)	Latency Increase (%)	# Referrals Sent (IPV4 & IPV6)	Latency (msec)	Latency Increase (%)	# Referrals Sent (IPV4 & IPV6)
VeriSign	64.6.64.6	70	90	28.57	3 (Only IP4)	Client-Timeout	INF	35
ADGuard	94.140.14.14	78	200	156.41	14	11000	14002.5	40
Dyn	216.146.35.35	97	160	64.95	3 (Only IP4)	Client-Timeout	INF	34
Quad9	9.9.9.9	79	130	64.55	5 (Only IP4)	8000	10026	38
Yandex	77.88.8.8	94	250	165.95	4 (Only IP4)	Client-Timeout	INF	40
Comodo-Secure	8.26.56.26	76	130	71.05	12	1600	2005.26	13
OpenDNS	208.67.222.222	75	170	126.6	12	1200	1500	22
Google	8.8.8.8	74	130	75.67	14	4700	6251.35	14
Cloudflare	1.1.1.1	80	560	600	10	5350	6587.5	12
DNS.Watch	84.200.69.80	70	100	42.85	3 (Only IP4)	13000	18471.4	34
FreeDNS	37.235.1.174	86	225	161.62	5 (Only IP4)	8000	9202.32	20 (Only IP4)
Hurricane	74.82.42.42	72	210	191.6	5 (Only IP4)	6100	8372.22	20 (Only IP4)
Level3 ($\|RR\| = 20$)*	209.244.0.3	90	550	511.11	20 (Only IP4)	11000	12122.22	40
Level3 ($\|RR\| = 30$)*	209.244.0.3	90	5000	5455.5	20 (Only IP4)	14000	15455.5	60
Norton	199.85.127.10	80	115	43.75	3 (Only IP4)	Client-Timeout	INF	30
SafeDNS	195.46.39.39	86	90	4.65	6 (Only IP4)	5085	5812.8	7 (Only IP4)
Ultra	156.154.71.1	70	100	42.85	3 (Only IP4)	Client-Timeout	INF	24
BIND9	Cloud-Setup-Testing env	50	67	34	10	10050	20000	40
KNOT	Cloud-Setup-Testing env	72	110	52.7	10	17575	24309.7	14
UNBOUND	Cloud-Setup-Testing env	90	200	122.2	12	17540	19388.8	20

Table 2: Latency for BIND9, Knot, Unbound, and 16 open resolvers in responding to one NRDelegationAttack or one NXNSAttack malicious query with 20 NS names in the referral-list ($\|RR\| = 20$), and the number of referral queries sent by the resolver (only those sent in Step 6 in Figure 1) while processing these 2 queries.
* Because the initial NXNSAttack test on level3 showed 20 referral queries sent (which is $= \|RR\|$), we verified that this resolver limit on the number of referrals sent $= 20$, by testing it with $\|RR\| = 30$.

CPU and Memory load would be somewhat reduced since only 20 names would be looked up in the cache and ADB.

Preserving the No_Fetch: We also suggest not clearing the No_Fetch flag upon restart. In this way, after the restart, the resolver knows it already issued the resolution of 5 names and that the referral_limit has been reached; thus, it will not query about any more names from the LRR. Either way, only 5 queries are issued by the resolver and the recursive repetition of Phase II is eliminated, further eliminating Phase III. Because this flag is implemented in the code, it must be reset before processing a new referral response. This is not trivial since it is an event that is not simple to distinguish. Clearly, this flag must be reset between client queries.

No RR lookup on No_Fetch: To completely eliminate the repetition of Phase II, it is possible to add a condition that prevents looking up the referral-list in the ADB and the resolver cache if the flag No_Fetch is turned on. This mitigation ensures that Phase II of the attack is executed only the first time the referral response is processed, which we believe follows the spirit of the NXNSAttack mitigation.

The mechanisms "Preserving the No_Fetch" and "No RR lookup on No_Fetch" are difficult to implement while ensuring that legitimate client queries are not affected. For example, if the No_Fetch flag is preserved in the resolution process, a client querying "foo.example" in which the ".example" domain name has more than 5 name servers in the referral list, could receive a FAIL response. This happens because the resolver will stop the

resolution process after adding the first 5 names to ADB and will not resolve the "foo.example" domain name.

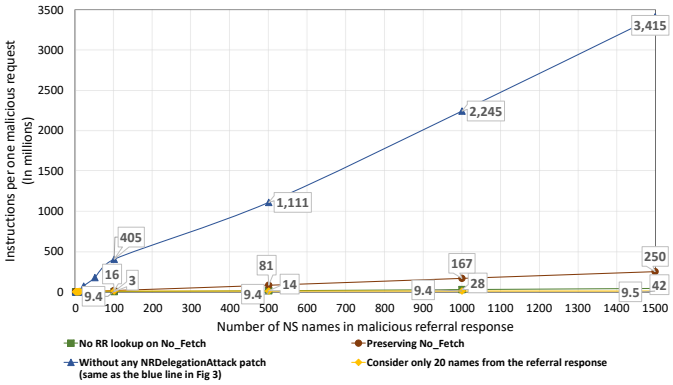


Figure 6: Comparing performance of the first three mitigation suggestions by the number of instructions in the resolver per one malicious NRDelegationAttack client request, on BIND9 version 9.16.6 (NXNSAttack patched). Blue is used to designate the performance of the unpatched version under attack (as in Figure 3). The other lines show the three other patches.

Cache NR responses: Another suggestion to mitigate the NRDelegationAttack is to add a new RCODE, "NR", to mark a domain name in the cache that resolves to a server that is non-responsive to DNS queries or to queries in general. Caching "NR" domains with a configurable TTL prevents the resolver from asking the same non-responsive server over and over again. We also suggest adopting the NR RCODE in the resolver algorithm in a way that stops the resolution process when more than x of the resolutions leads to an NR resource.

We tested the first three NRDelegationAttack mitigation suggestions above on BIND9 version 9.16.6 and compared the number of instructions executed with and without the three mitigations. As shown in Figure 6, the three suggestions significantly reduced the amount of resolver resources consumed during the NRDelegationAttack. As can be seen in Figure 6, the load of the attack with the "No_Fetch" mitigation is about 7.3% of the load without any mitigation. We also saw a change of 1.2% with the "No RR lookup on No_Fetch" mitigation and 0.3% with the "Consider only k of the NS names in the referral response" mitigation. As shown for the "Consider only k of the NS names in the referral response" mitigation, when k equals 20, the number of instructions is around 9.5 million, regardless of the number of NS names in the referral response. This is indeed the number of instructions executed upon receiving a malicious referral response with 20 NS names.

As part of the responsible disclosure procedure, we tested the NRDelegationAttack patch of each of the three vendors, BIND, Unbound, and Knot. The corresponding results are presented in Figure 7, showing that all three patches present a substantial improvement in performance under the NRDelegationAttack.

9 Related Work

Complexity attacks exploit algorithmic deficiencies in the implementation of targeted services. Many network systems such as Bro intrusion detection [12], TCP [20], and hashes [6, 12] are vulnerable to complexity attacks [7]. For example, Crosby et al. [12] show how a maliciously crafted input compromises the data structure of the server, leading to malfunction of the server in the worst-case scenario. The authors demonstrate their complexity attacks on the Perl, Squid, and Bro systems.

Pfeifer et al. [32] raised issues regarding configuration errors and the risk of unavailability due to the complexity of the DNS system. They also examine possible improvements to reduce the risk. Although they targeted amplification and flood attacks [19], to the best of our knowledge, no specific complexity attack was observed on DNS servers. Moura et al. [26] in their TsuNAME amplification attack, found a way to use maliciously crafted authoritative configuration with cyclical dependencies on DNS records. As a result, some recursive resolvers can greatly amplify queries, potentially resulting in a denial-of-service to DNS servers. Luo et al. [21] analyzed the prevalence and characteristics of the NXDOMAIN and water torture attacks. They showed that the volume of the DNS water torture attack is significantly larger than the volume of disposable domains or DGA queries. In addition, they demonstrated that the client IPs launching the DNS water torture attack are all random generated fake addresses. NXNSAttack [5] is another flood attack that exposed a vulnerability causing a flood of queries between the recursive resolver and the authoritative, resulting in an overload on both of them and producing an amplified DDoS effect. Bushart et al. [10] exposed an application-layer DoS attack in which they chained CNAME records and forced resolvers to perform deep name resolutions. This leads to an overload on the target authoritative name server and considerably reduces its availability.

Maury [22] presents a different attack named iDNS, which also exploits the delegations of name servers in a referral response and causes a packet amplification factor (PAF) of 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. Some measures were taken by different DNS vendors such as BIND9 and Unbound following the disclosure of iDNS, as described in [22]; however, these measures do not influence or weaken the NRDelegationAttack.

Wang [35] focused on the DNS security implications of glue records. He describes how recursive resolver implementations such as BIND9 and Unbound treat glue records. How-

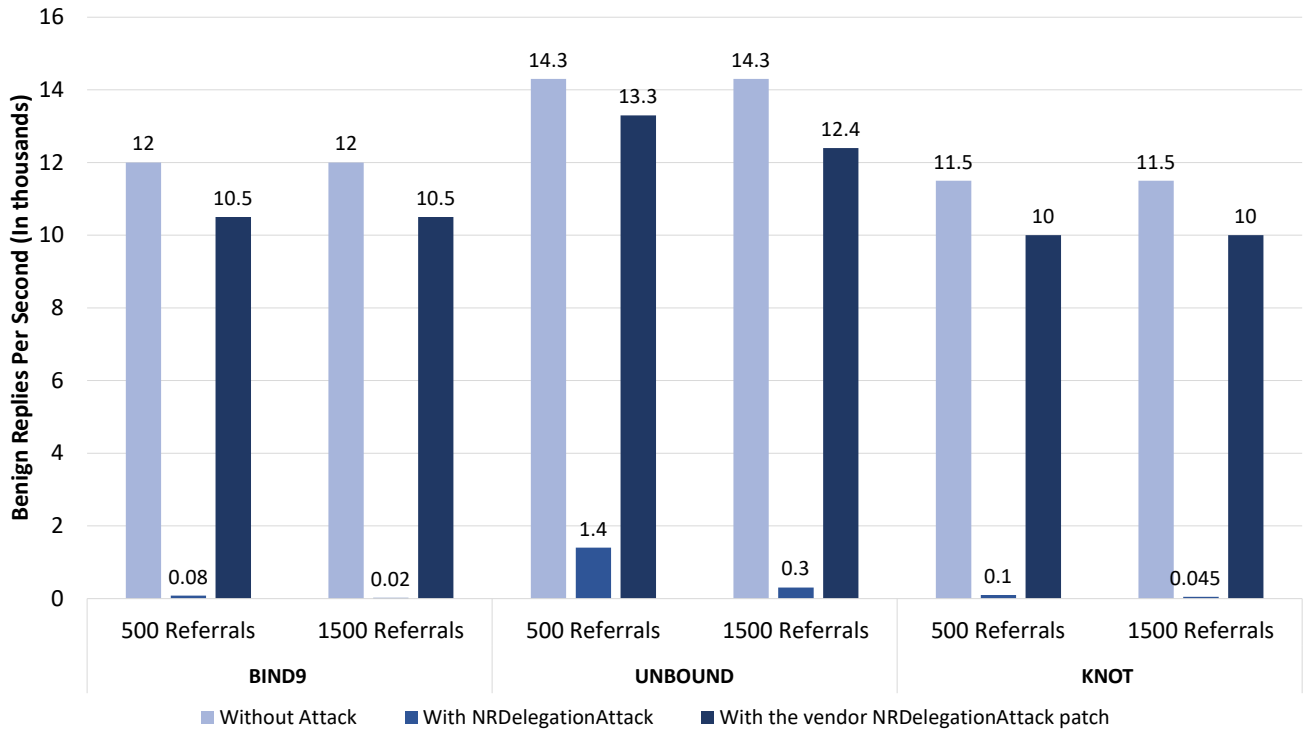


Figure 7: Resolver throughput during NRDelegationAttack, tested on NRDelegationAttack patched and non-patched versions of BIND9, Knot, and Unbound implementations.

ever, the focus of his work is on cache poisoning vulnerabilities rather than the impact on the recursive performance as in our paper.

Muller et al. [28] perform a comprehensive measurement using the RIPE atlas to analyze how recursive resolvers select the name server with which to interact, out of a set of multiple authoritative servers. These authors show that, upon arrival of client requests, most recursive resolvers interact with all the name servers of a given domain to distribute the load and maintain latency for RTT algorithms. They focus on how and when the recursive resolvers query a set of multiple authoritative servers. In another work, Moura et al. [27] 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 [25], Moura et al. measure and show the impact of caching and long TTL while discussing DNS defenses during a DDoS attack.

The DNS infrastructure is also facing abuse by various entities that use it for applications for which it was not intended. In this case, a large volume of temporary domain names (known as disposable domains [15]) is commonly used to help these services communicate via DNS queries. A study [15] of large scale DNS traffic recordings showed that 60% of all distinct resource records observed daily are disposable. Hao et al. [11] examined the negative impact of disposable domains on recursive caching. They proposed a classification based on domain name features to increase the cache hit-rate.

10 Conclusions

Two conclusions follow from this paper regarding the DNS system and how to fix its problems. First, the DNS system appears to be simple, with a straightforward interface at a high level. However, under the hood, it is a complex and complicated system whose behavior is hard to analyze and predict. This paper, along with other works and recent attacks, exposes a set of complex mechanisms and intricate relations between procedures and protocols that compose the DNS system. Specifically, as this paper shows, it is difficult to understand and analyze how many compute and/or communication resources one query may consume. This brings us to the second conclusion, that it is hard to predict the consequences of a fix made to a problem. The mitigation of one attack can open the door to another one. Here, we showed that the techniques used to mitigate the NXNSAttack proved to be a major driver in the current NRDelegationAttack.

The NRDelegationAttack introduces a serious DDoS complexity attack on DNS resolvers. We analyzed the attack and prove its severity through several different tests. The attack can easily deny service from any target resolver of choice. In addition, we presented the complexity factor function and explained the source of the complexity.

Finally, we suggested and tested several simple and effective mitigation techniques, including methods used by different vendors.

Acknowledgements: The authors are grateful to the USENIX Security shepherd and referees of the paper for their comments which have considerably improved the paper. We are thankful to Michał Kępień and Petr Špaček from ISC (BIND), Wouter Wijngaards and George Thessalonikefs from NLnet Labs (Unbound), Vladimir Cunat from CZ.NIC (Knot) and the other members of the DNS-OARC forum for their many comments and discussions on an earlier version of the paper that have considerably improved it, and for their analysis and testing of the attack. In addition, the authors would like to thank Ron Stajnrod, Adiel Vaintraub and Tom Legkov for their helpful comments on earlier versions of the paper.

References

[1] Cve-2022-2795. https://nvd.nist.gov/vuln/detail/CVE-2022-2795.

[2] Cve-2022-3204. https://nvd.nist.gov/vuln/detail/CVE-2022-3204.

[3] Cve-2022-40188. https://nvd.nist.gov/vuln/detail/CVE-2022-40188.

[4] Valgrind debugging and profiling. https://valgrind.org/.

[5] Yehuda Afek, Anat Bremler-Barr, and Lior Shafir. NXNSAttack: Recursive DNS inefficiencies and vulnerabilities. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 631–648. USENIX Association, August 2020. https://www.usenix.org/conference/usenixsecurity20/presentation/afek.

[6] Udi Ben-Porat, Anat Bremler-Barr, and Hanoch Levy. Evaluating the vulnerability of network mechanisms to sophisticated ddos attacks. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 2297–2305. IEEE, 2008.

[7] Udi Ben-Porat, Anat Bremler-Barr, and Hanoch Levy. Computer and network performance: Graduating from the "age of innocence". *Computer Networks*, 66:68–81, 2014. https://www.sciencedirect.com/science/article/pii/S1389128614001236.

[8] S. Bortzmeyer. Nxdomain: There really is nothing underneath. November 2016. https://datatracker.ietf.org/doc/html/rfc8020.

[9] S. Bortzmeyer. Dns query name minimisation to improve privacy. Internet Engineering Task Force (IETF), November 2021. https://datatracker.ietf.org/doc/html/rfc9156.

[10] Jonas Bushart and Christian Rossow. Dns unchained: Amplified application-layer dos attacks against dns authoritatives. In Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis, editors, *Research in Attacks, Intrusions, and Defenses*, pages 139–160, Cham, 2018. Springer International Publishing.

[11] 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.

[12] Scott A. Crosby and Dan S. Wallach. Denial of service via algorithmic complexity attacks. In *12th USENIX Security Symposium (USENIX Security 03)*, Washington, D.C., August 2003. USENIX Association. https://www.usenix.org/conference/12th-usenix-security-symposium/denial-service-algorithmic-complexity-attacks.

[13] CZ.NIC. https://www.knot-resolver.cz/, 2021.

[14] FireEye. Global DNS Hijacking Campaign: DNS Record Manipulation at Scale. https://www.fireeye.com/blog/threat-research/2019/01/global-dns-hijacking-campaign-dns-record-manipulation-at-scale.html, August 2019.

[15] 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.

[16] P. Hoffman, A. Sullivan, and K. Fujiwara. RFC 8499–DNS Terminology. https://tools.ietf.org/html/rfc8499, 2019.

[17] ISC. https://kb.isc.org/docs/aa-01463, 2019.

[18] ISC. https://www.isc.org/downloads/bind, 2021.

[19] Marc Kührer, Thomas Hupperich, Christian Rossow, and Thorsten Holz. Exit from hell? reducing the impact of amplification ddos attacks. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 111–125, 2014.

[20] A. Kuzmanovic and E. W. Knightly. Low-rate tcp-targeted denial of service attacks (the shrew vs. the mice and elephants). In *ACM SIGCOMM*, 2003.

[21] 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.

[22] Florian Maury. The iDNS attack. In *OARC 15*, 2015.

[23] Warren Mercer and Paul Rascagneres. Talos blog: DNSpionage campaign targets middle east. https://blog.talosintelligence.com/2018/11/dnspionage-campaign-targets-middle-east.html, August 2019.

[24] P. Mockapetris. Domain names - concepts and facilities, 1987. https://datatracker.ietf.org/doc/html/rfc1034.

[25] 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.

[26] Giovane C. M. Moura, Sebastian Castro, John Heidemann, and Wes Hardaker. Tsuname: Exploiting misconfiguration and vulnerability to ddos dns. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, page 398–418, New York, NY, USA, 2021. Association for Computing Machinery. https://doi.org/10.1145/3487552.3487824.

[27] 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 *Internet Measurement Conference*, pages 255–270, 2016.

[28] Moritz Müller, Giovane C. M. Moura, Ricardo de O. Schmidt, and John Heidemann. Recursives in the wild: Engineering authoritative DNS servers. In *Internet Measurement Conference*, pages 489–495, New York, NY, USA, 2017.

[29] NLNETLABS. https://www.nlnetlabs.nl/projects/unbound/about/, 2021.

[30] NLNETLABS. https://www.nlnetlabs.nl/projects/nsd/about/, 2021.

[31] Nominum. resperf(1) - Linux man page. https://linux.die.net/man/1/resperf, May. 2019.

[32] Gert Pfeifer, Andre Martin, and Christof Fetzer. Reducible complexity in dns. 09 2010.

[33] ICANN Security and Stability Advisory Committee (SSAC). Preliminary report on dns response modification. https://www.icann.org/en/system/files/files/sac-032-en.pdf, June 2008.

[34] Shani Stajnrod. Dns full protocol simulator. https://github.com/ShaniBenAtya/dnssim.

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

11 Appendices

A SLIST

To demonstrate the use of the SLIST in the resolution process, we use BIND9 implementation (in which the SLIST is called "ADB"). In the following example (Figure 8) we explain how the resolver behaves in an attempt to resolve the query as quickly as it can, after receiving a referral-list response with no glue records from the authoritative.

Upon receiving a new client request (i.e., e1.example.com, Step 1 in Figure 8), the resolver first checks whether the name server is already in the server's cache or was previously inserted into the ADB. The main purpose for checking these components is to avoid wasting time and sending unnecessary requests if the resolver already knows the answer to the requested name server. Assuming the resolver does not find the answer, it inserts the name server into the ADB and starts a new resolution process to resolve it. First, the resolver searches the example1.com name server, assuming the name is already presented in its cache. The resolver issues a query to the example1.com name server asking about e1.example.com (Step 2). In this example, the example1.com authoritative answers with a referral-list response, which contains two name servers that may have the answer to the requested name server (www1.example2.com and www2.example2.com as seen in Step 3). Assuming all the names in the referral list have no other information relevant to the resolution process (i.e., no additional glue records), the resolver checks whether it has helpful information about each one of them in the cache or ADB (described as Step (a) in Figure 8). If the resolver cannot find any helpful information about the name servers in the referral response, it tries to invoke queries about all of them in parallel (Steps 4-5).

Upon receiving the responses to the queries described in Steps 4-5 (Steps 4R and 5R), the resolver adds them to the ADB (as described in (b)), and continues to search for their IP address with example2.com authoritative server (Steps 4.1 and 5.1). Then, the resolver receives their IP addresses (Steps 4.1R and 5.1R) and saves them in the ADB (as can be seen in (c)). In Steps 4.2 and 5.2, the resolver searches for the original client query (e1.example1.com) and receives the final IP address in Steps 4.2R and 5.2R. Lastly, in the table described in (d), we can see the final state of the ADB, which is flushed to the resolver cache after returning the final answer to the client (step 6).

B NXDOMAIN-Redirection Response

Sometimes rather than responding with an NXDOMAIN to a non-existent query, an authoritative server may redirect the querying resolver to another authoritative server that can provide an alternate resolution (as described in [33]). There are different uses for this redirection response. For example, if the TLD authoritative server receives a query for a non-existent domain name (for instance foofake.example), it can use the NXDOMAIN-Redirection configuration and return an 'NS' response instead of an NXDOMAIN. The purpose of this mechanism could be, for example, to direct the client to a page offering to sell the requested domain or to obtain advertising revenue from third-parties. The use of NXDOMAIN-Redirection is optional and, for example, is off by default in BIND9.

C Delegation Tests on .com TLD

To determine whether an authoritative name server supports delegation response or not, the attacker needs to have access to a resolver, e.g., the attacker can set up a BIND9 resolver. Then, the following steps are done: First, the attacker configures his computer to use his own resolver; then, he sends a non-existent query to the desired authoritative. Finally, the attacker follows the chain of requests and responses on his resolver to see whether the authoritative answered with an NXDOMAIN response or a delegation.

We tested .com to see if it supports delegation responses. We also checked whether any of the resources we were redirected to are suitable for NRDelegationAttack. (See Section 4 for detailed information about NRDelegationAttack requirements.) The results of our tests are presented in Table 3.

D Testing QNAME Minimisation

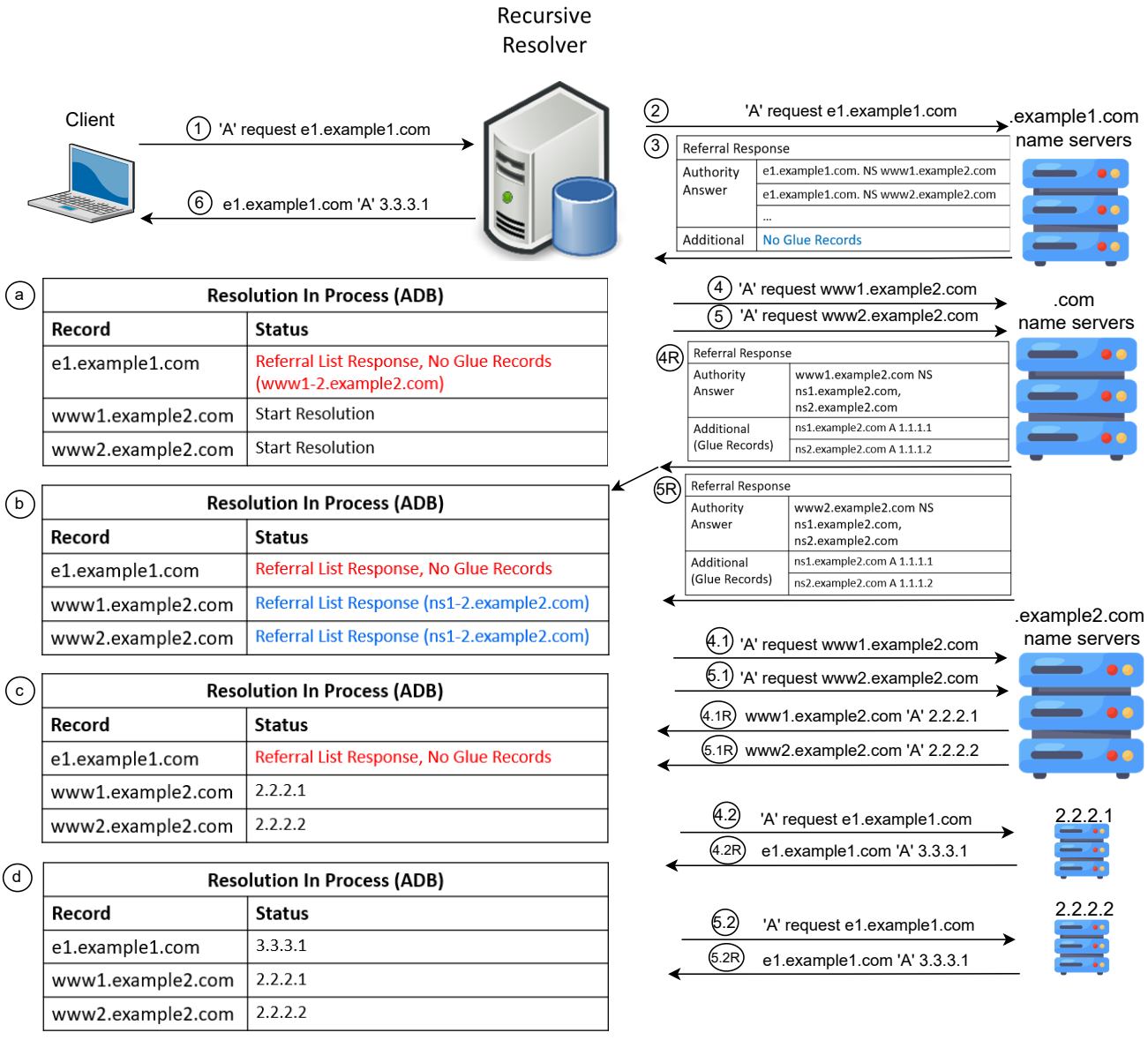
By saving the previous queries in its cache and using them instead of resolving the same name server over and over again, the resolver is able to operate efficiently, and even more importantly - quickly. To achieve efficiency and speed, both the resolver query mechanism and the cache pretend to be hierarchical. But, in this research, we discovered that both of them may not be as hierarchical and efficient as expected.

There are two main options for the resolver to query the authoritative chain about a client query. We describe them as:

1. The "traditional" method of querying, in which the resolver always asks about the fully qualified domain name.

2. Using query minimization [9] to disclose QNAME Minimisation information to the authoritative and increase privacy.

Figure 9 shows the traditional method in which the resolver queries the authoritative chain. After the resolver receives a new query to resolve test1.fake.com from the client (Step 1), and assuming the requested name server is not saved in the resolver's cache (Step 2), the resolver simply starts querying the authoritative chain about the full name server (Steps 3,4



a

Resolution In Process (ADB)	
Record	**Status**
e1.example1.com	Referral List Response, No Glue Records (www1-2.example2.com)
www1.example2.com	Start Resolution
www2.example2.com	Start Resolution

b

Resolution In Process (ADB)	
Record	**Status**
e1.example1.com	Referral List Response, No Glue Records
www1.example2.com	Referral List Response (ns1-2.example2.com)
www2.example2.com	Referral List Response (ns1-2.example2.com)

c

Resolution In Process (ADB)	
Record	**Status**
e1.example1.com	Referral List Response, No Glue Records
www1.example2.com	2.2.2.1
www2.example2.com	2.2.2.2

d

Resolution In Process (ADB)	
Record	**Status**
e1.example1.com	3.3.3.1
www1.example2.com	2.2.2.1
www2.example2.com	2.2.2.2

Figure 8: Resolution process - network and Address DB operations view, part of the Appendix

.

Response Ffrom (.com Authoritative IP)	Delegation Response Domain Name	Suitable for NRDelegationAttack
192.48.79.30	nsg1.namebrightdns.com	YES
192.48.79.30	nsg2.namebrightdns.com	YES
192.48.79.30	ns1.redehost.com.br	YES
192.48.79.30	ns2.redehost.com.br	YES
192.41.162.30	ns1.dan.com	YES
192.41.162.30	ns2.dan.com	YES
192.48.79.30	ns1.dynadot.com.br	NO
192.48.79.30	ns2.dynadot.com	NO
97.74.111.55, 173.201.79.55	ziyuan.baidu.com	NO
192.52.178.30	ns1.hover.com	NO
192.52.178.30	ns2.hover.com	NO
192.48.79.30	pdns11.domaincontrol.com	NO
192.48.79.30	pdns12.domaincontrol.com	NO
192.48.79.30	ns35.domaincontrol.com	NO
192.48.79.30	ns36.domaincontrol.com	NO

Table 3: Delegation responses tested on .com TLD.

and 5). The final response, which in this case is the non-existent name server (NXDOMAIN), is received from .com authoritative server (Step 6) and saved in the cache (Step 7). Finally, the response is returned to the client (Step 8).

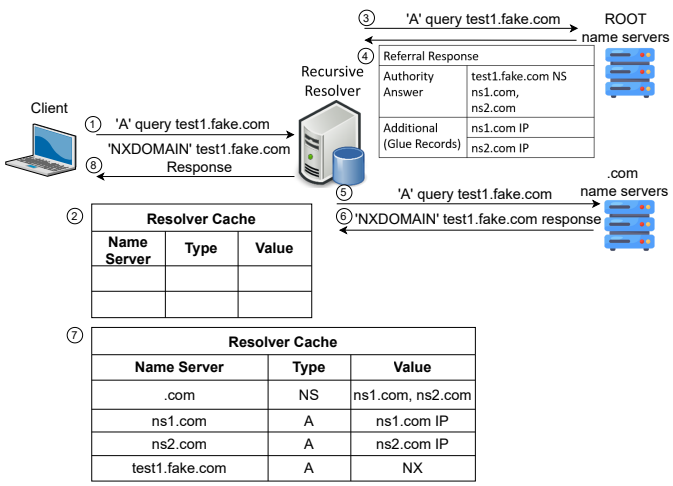


Figure 9: The traditional form of querying the authoritative chain and the result saved in the resolver cache.

If the resolver is using QNAME Minimisation (as shown in Figure 10), after the resolver receives a new query to resolve test1.fake.com from the client (Step 1), and assuming the requested name server is not saved in the resolver's cache (Step 2), the resolver first queries the root authoritative about just the suffix .com of the requested domain name (Steps 3 and 4). Then, the resolver queries the .com authoritative about the second level of the domain name fake.com (Step 5), which in this case does not exist. Hence, the resolver gets an NXDOMAIN response about the requested domain fake.com (Step 6). Because the response is not validated with DNSSEC, even though an NXDOMAIN response is received about the fake.com suffix according to the NXDOMAIN RFC [8], the resolver may still query about the full domain name (Steps 7 and 8). The final response, which in this case is NXDOMAIN for the whole queried name, is then saved in the cache (Step 9) and returned to the client (Step 10).

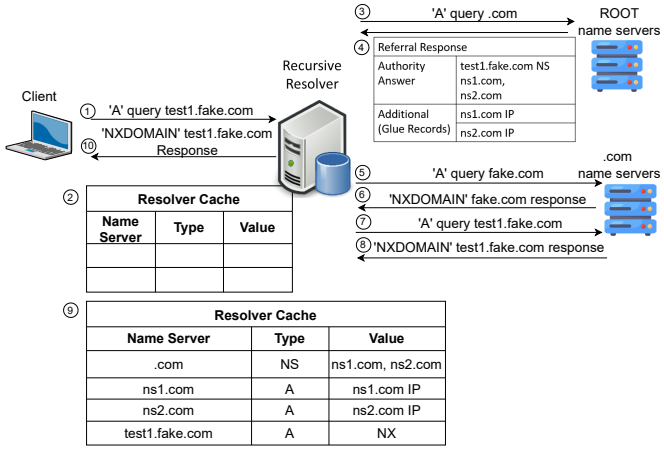


Figure 10: Querying the authoritative chain using QNAME Minimisation and saving the result in the resolver cache.

We tested the QNAME Minimisation mechanism on 10 different implementations by querying them multiple times on the same non-existent domain name, which has our own authoritative suffix. Then, we recorded the requests received on our authoritative. We noticed that half of the resolvers tested did not save the fake.com suffix in their cache but kept asking about it from the same resolver IP address over and over again. This led to unnecessary queries between the resolver and authoritative servers. Table 4 presents the results of our QNAME Minimisation and describes the querying and caching mechanisms for each implementation.

Even though fake.com is already known not to exist, this chain of queries about every subdomain of fake.com received from the client results in a large number of unnecessary queries to the subdomain of fake.com. This behavior is mutual to both the "traditional" and QNAME Minimisation techniques. We use this behavior in the NRDelegationAttack to create a query with a long list of referrals that have the same

suffix (e.g., redrct.com) and direct the resolver to an authoritative server that supports NXDDOMAIN-Redirection. This causes a significant number of queries generated by a single attack query, resulting in an amplification effect.

Resolver Implementation	Querying Mechanism	Caching Mechanism
BIND9 (version 9.16.6)	QNAME Minimisation only on TLDs	Caching only TLDs and the whole query
ADGuard(94.140.14.14)	QNAME Minimisation	Caching only the whole query
Comodo-Secure(8.26.56.26)	QNAME Minimisation	Caching all queries but keeps querying the whole prefix even if it is under a non-existent domain
Dyn(216.146.35.35)	QNAME Minimisation	Caching only the whole query
Cloudflare(1.1.1.1)	QNAME Minimisation	Caching all queries but keeps querying the whole prefix even if it is under a non-existent domain
OpenDNS(208.67.222.222)	QNAME Minimisation	Caching all queries but keeps querying the whole prefix even if it is under a non-existent domain
Quad9(9.9.9.9)	QNAME Minimisation	Caching all queries but keeps querying the whole prefix even if it is under a non-existent domain
VeriSign(64.6.64.6)	QNAME Minimisation	Caching all queries but keeps querying the whole prefix even if it is under a non-existent domain
Google(8.8.8.8)	No QNAME Minimisation	Caching only the whole query
Yandex(77.88.8.8)	No QNAME Minimisation	Caching only the whole query

Table 4: QNAME Minimisation mechanism tested on 10 different open resolvers.