DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels

Keyu Man
kman001@ucr.edu
University of California, Riverside

Zhiyun Qian
zhiyunq@cs.ucr.edu
University of California, Riverside

Zhongjie Wang
zwang048@ucr.edu
University of California, Riverside

Xiaofeng Zheng
zhengxiaofeng@qianxin.com
Qi-AnXin Group
Tsinghua University

Youjun Huang
huangyj@cernet.edu.cn
Tsinghua University

Haixin Duan
duanhx@tsinghua.edu.cn
Tsinghua University
Qi-AnXin Group

ABSTRACT

In this paper, we report a series of flaws in the software stack that leads to a strong revival of DNS cache poisoning — a classic attack which is mitigated in practice with simple and effective randomization-based defenses such as randomized source port. To successfully poison a DNS cache on a typical server, an off-path adversary would need to send an impractical number of 2^{32} spoofed responses simultaneously guessing the correct source port (16-bit) and transaction ID (16-bit). Surprisingly, we discover weaknesses that allow an adversary to "divide and conquer" the space by guessing the source port first and then the transaction ID (leading to only $2^{16} + 2^{16}$ spoofed responses). Even worse, we demonstrate a number of ways an adversary can extend the attack window which drastically improves the odds of success.

The attack affects all layers of caches in the DNS infrastructure, such as DNS forwarder and resolver caches, and a wide range of DNS software stacks, including the most popular BIND, Unbound, and dnsmasq, running on top of Linux and potentially other operating systems. The major condition for a victim being vulnerable is that an OS and its network is configured to allow ICMP error replies. From our measurement, we find over 34% of the open resolver population on the Internet are vulnerable (and in particular 85% of the popular DNS services including Google's 8.8.8.8). Furthermore, we comprehensively validate the proposed attack with positive results against a variety of server configurations and network conditions that can affect the success of the attack, in both controlled experiments and a production DNS resolver (with authorization).

CCS CONCEPTS

• **Networks** → **Cross-layer protocols**; • **Security and privacy** → **Network security**.

KEYWORDS

DNS cache poisoning, side channel, off path attack, ICMP rate limit

CCS '20, November 9–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7089-9/20/11.
https://doi.org/10.1145/3372297.3417280

ACM Reference Format:
Keyu Man, Zhiyun Qian, Zhongjie Wang, Xiaofeng Zheng, Youjun Huang, and Haixin Duan. 2020. DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3372297.3417280

1 INTRODUCTION

Domain name system (DNS) is an essential part of the Internet, originally designed to translate human-readable names to IP addresses. Nowadays, DNS has also been overloaded with many other security critical applications such as anti-spam defenses [38], routing security (e.g., RPKI) [10]. In addition, DNS also plays a crucial role in bootstrapping trust for TLS. TLS certificates are now commonly acquired by proving the ownership of a domain [2]. Therefore, compromising the integrity of DNS records can lead to catastrophic security failures, including fraudulent certificates being issued that can compromise the underpinning of public key cryptography [9].

Historically, the very first widely publicized DNS cache poisoning attack was discovered by Kaminsky [39] in 2008, who demonstrated that an off-path attacker can inject spoofed DNS responses and have them cached by DNS resolvers. This has led to a number of DNS defenses being deployed widely, including source port randomization [35] and "birthday protection" [29, 30]. Other defenses such as 0x20 encoding [19] and DNSSEC [6] have also gained some traction. Unfortunately, due to reasons such as incentives and compatibility, these two defenses are still far from being widely deployed as reported in recent studies [13, 16, 17, 36, 46, 53]. To summarize, source port randomization becomes the most important hurdle to overcome in launching a successful DNS cache poisoning attack. Indeed, in the past, there have been prior attacks that attempt to derandomize the source port of DNS requests [30, 32]. As of now, they are only considered nice conceptual attacks but not very practical. Specifically, [32] requires an attacker to bombard the source port and overload the socket receive buffer, which is not only slow and impractical (unlikely to succeed in time) but also can be achieved only in a local environment with stringent RTT requirement. In [30], it is assumed that a resolver sits behind a NAT which allows its external source port to be derandomized, but such a scenario is not applicable to resolvers that own public IPs.

In contrast, the vulnerabilities we find are both much more serious and generally applicable to a wide range of scenarios and conditions. Specifically, we are able to launch attacks against all layers of

caches which are prevalent in modern DNS infrastructure [5, 8, 54], including application-layer DNS caches (e.g., in browsers) [5], OS-wide caches [5], DNS forwarder caches [34] (e.g., in home routers), and the most widely targeted DNS resolver caches. The vulnerabilities also affect virtually all popular DNS software stacks, including BIND [18], Unbound [43], and dnsmasq [40], running on top of Linux and potentially other OSes, with the major requirement being the victim OS allowed to generate outgoing ICMP error messages. Interestingly, these vulnerabilities result from either design flaws in UDP standards or subtle implementations details that lead to side channels based on a global rate limit of ICMP error messages, allowing derandomization of source port with great certainty.

To demonstrate the impact, we devise attack methods targeting two main scenarios, including DNS forwarders running on home routers, and DNS resolvers running BIND/Unbound. With permissions, we also tested the attack against a production DNS resolver that serves 70 million user queries per day, overcoming several practical challenges such as noises, having to wait for cache timeouts, multiple backend server IPs behind the resolver frontend, and multiple authoritative name servers. In our stress test experiment, we also evaluate the attack in even more challenging network conditions and report positive results.

In this paper, we make the following contributions:

- We systematically analyze the interaction between application- and OS-level behaviors, leading to the discovery of general UDP source port derandomization strategies, the key one being a side channel vulnerability introduced by a global rate limit of outgoing ICMP error messages.
- We research the applicability of the source port derandomization strategies against a variety of attack models. In addition, to allow sufficient time in conducting the derandomization attack, we develop novel methods to extend the attack window significantly, one of them again leveraging the rate limiting feature (this time in the application layer).
- We conduct extensive evaluation against a wide variety of server software, configuration, and network conditions and report positive results. We show that in most settings, an attacker needs only minutes to succeed in an end-to-end poisoning attack. We also discuss the most effective and simple mitigations.

2 CURRENT STATE OF DNS CACHE POISONING ATTACKS

The classic DNS cache poisoning attack in 2008 [39] targeted a DNS resolver by having an off-path attacker tricking a vulnerable DNS resolver to issue a query to an upstream authoritative name server. Then the attacker attempts to inject rogue responses with the spoofed IP of the name server. If the rogue response arrives before any legitimate ones, and if it matches the "secrets" in the query, then the resolver will accept and cache the rogue results. Specifically, the attacker needs to guess the correct source/destination IP, source/destination port, and the transaction ID (TxID) of the query. The transaction ID is 16-bit long. At the time when both the source and destination port (i.e., 53) were fixed, 16-bit is the only randomness. Thus an off-path attacker can simply brute force all possible values with 65,536 rogue responses, not to mention a few optimizations such as birthday attacks that can speed the attack even further.

2.1 State-of-the-Art Defenses

A number of defenses have since then been promoted to mitigate the threat of DNS cache poisoning. They effectively render the original attack no longer feasible. We describe below the most widely known and deployed defenses.

• Randomization of source port [35] is perhaps the most effective and widely deployed defense as it increases the randomness from 16 bits to 32 bits. As an off-path attacker has to guess both the source port and TxID at the same time.

• Randomization of capitalization of letters in domain names, i.e., 0x20 encoding [19]. The offered randomness depends on the number of letters and can be quite effective also, especially for long names. Unfortunately, even though it is a simple change to the protocol, in practice it has significant compatibility issues with authoritative name servers encountered on the Internet [17, 21]. Therefore, most popular public resolvers now refrain from using 0x20 encoding by default. For example, Google DNS uses it only for a set of whitelisted name servers [21]; Cloudflare has even recently disabled 0x20 encoding altogether [16]. At the time of writing, we find only two (i.e., openNIC and Verisign) out of the 16 popular public DNS services we measured (see the other 14 in Table 2) use it by default to a test name server we setup. And the result roughly matches what was observed in a recent study [53].

• Randomization of the choice of name servers (server IP addresses) [31]. The offered randomness depending on the number of name servers. In practice, most domains employ less than 10 name servers, translating to only 2 to 3 bits. In addition, it has been shown that an attacker can induce query failures against certain name servers and therefore effectively "pinning" a resolver to the one remaining name server [30].

• DNSSEC [6]. The success of DNSSEC depends on the support of both resolvers and authoritative name servers. However, only a small fraction of domains is signed — 0.7% for .com domains, 1% for .org domains, and 1.85% for Top Alexa 10K domains, as reported in 2017 [13]. In the same study, it is also reported that only 12% of the resolvers enabling DNSSEC actually attempt to validate the received records. As a result, the overall deployment rate of DNSSEC is far from satisfactory.

2.2 New Attack Surface in the DNS Hierarchy

As alluded to earlier, modern DNS infrastructure has multiple layers of caching. Figure 1 provides a concise view: a client application often initiates a DNS query (through an API call such `gethostbyname()`) to an OS stub resolver — typically a separate system process that maintains an OS-wide DNS cache. The stub resolver does not perform any iterative queries; instead, it always forwards the request to the next layer up, a DNS forwarder which also forwards queries to its upstream recursive resolver. DNS forwarders are commonly found in Wi-Fi routers (e.g., in a home) and they maintain a dedicated DNS cache also. It is the recursive resolver that does the real job to iteratively query the authoritative name servers. The answers are then returned and cached in each layer.

All layers of caches are technically subject to the DNS cache poisoning attack. Unfortunately, most newly proposed attacks were focused on resolvers [9, 29, 30, 32], and very limited investigations have been done on stub resolvers [5] and forwarders [55].

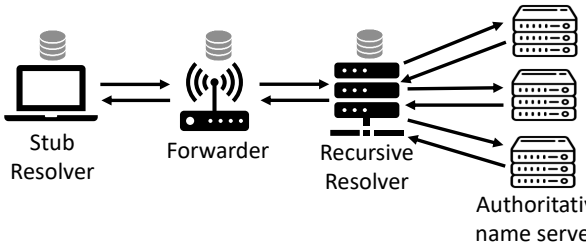

Figure 1: DNS Infrastructure with Multiple Layers of Caching

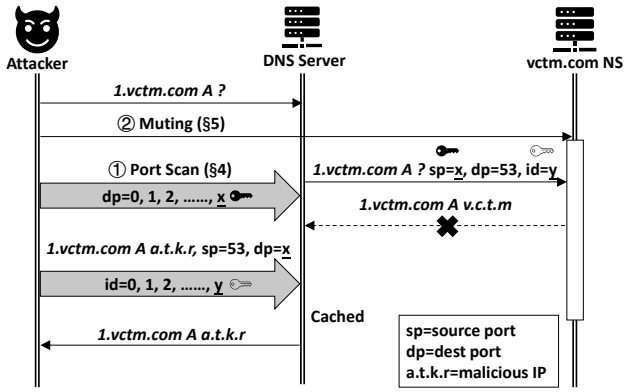

Figure 2: Attack Workflow

3 ATTACK OVERVIEW

We propose a general and novel attack, applicable to all modern DNS software stack, influencing all layers of DNS caching. The key characteristic is that it defeats the most effective and commonly deployed defense — randomization of source port.

Threat Model. In this paper, we focus on the attacks against DNS forwarders and resolvers due to their high impact. Similar to the classic DNS cache poisoning attack, we assume the attacker is off-path (not able to eavesdrop traffic between a forwarder and resolver), and capability of IP spoofing. According to a recent study in 2019 [47], 30.5% of ASes do not block packets with spoofed source IP addresses. In practice, an attacker only needs to find one node that can spoof IPs to carry out the attack. To demonstrate the ease of this, we rented a bullet-proof-hosting node specifically publicly advertised as IP-spoofing-capable ($50/month with unlimited data) and found that it indeed can spoof "arbitrary IPs".

In addition, the attacker needs to control a machine that is able to trigger a request out of a forwarder or resolver. In the case of a forwarder attack, this can happen when the attacker is located in a LAN managed by a wireless router. For example, an attacker can join a public wireless network in a coffee shop, a shopping mall, or an airport. The attacker can also control a puppet whose sole responsibility is to query the forwarder to launch the attack if direct access to the LAN is impossible. In a resolver attack, this can include any network (enterprise, organization, or institution) where the attacker is an insider or owns a compromised machine. Moreover, any public resolvers on the Internet also satisfy the requirement.

Attack Workflow. Regardless of a forwarder or resolver, as illustrated in Figure 2, our newly proposed attacks always start from triggering either one to send a DNS query, followed by two key steps as outlined below:

① *Inferring source port.* To overcome the randomization of source port, we leverage a novel and universal side channel in networking stacks to scan and discover which source ports were used to initiate a DNS query, at a speed of at most 1,000 guesses per second.

② *Extending attack window.* Normally an outstanding query will receive a reply from the upstream server in a matter of tens or hundreds of milliseconds. This is insufficient, given that the attacker needs time to infer the source port and to inject rogue DNS replies. We discover effective and novel strategies (different for forwarder and resolver attack) that can greatly extend the attack window to at least seconds (and even more than 10s), allowing realistic cache poisoning opportunities. We will discuss this in §5.

Once the source port number is known, the attacker simply injects a large number of spoofed DNS replies bruteforcing the TxIDs, which can be done in high speed, given that most servers have sufficient network bandwidth.

4 INFERRING DNS QUERY'S SOURCE PORT

In this section, we will describe the idea and procedure of inferring DNS source ports. We will also measure the vulnerable software and in-the-wild population when feasible.

4.1 Analysis of UDP Source Port Scannability

UDP is a stateless protocol and hence fundamentally different from TCP. More specifically, it is stated in the UDP programming guideline (RFC 8085 [24]) that "UDP datagrams may be directly sent and received, without any connection setup. Using the sockets API, applications can receive packets from more than one IP source address on a single UDP socket." Furthermore, to ensure that an application will receive data from only one particular source address, "these applications MUST implement corresponding checks at the application layer or explicitly request that the operating system filter the received packets."

These are surprisingly under-scrutinized statements. On a first glance, they may be interpreted as applicable to UDP servers only, which can bind to a local port, and subsequently receive packets from "any remote IPs". Surprisingly, from our experiments, it applies to UDP clients as well — a client calling `sendto()` on a specific remote IP and subsequently `recvfrom()` on the same socket can technically receive packets from "any other IPs" as well. We have verified this behavior on all modern operating systems, including Windows, Linux, and MacOS.

This nuanced behavior has a profound impact on what an attacker can learn through a trivial UDP port scan — when a DNS server issues a query, its source port effectively becomes open to the public. This allows an attacker to simply scan the ephemeral

port range with any UDP packet, which will trigger nothing upon hitting the correct port (as the probe will be accepted by the OS but discarded at the application layer), or an ICMP port unreachable message upon missing it (by design).

Next, the UDP programming guideline (RFC 8085) further states that "Many operating systems also allow a UDP socket to be connected, i.e., to bind a UDP socket to a specific pair of addresses and ports." Indeed, modern socket APIs allow `connect()` on a UDP socket but "this is only a local operation that serves to simplify the local send/receive functions and to filter the traffic". As a result, when a DNS query is issued from a source port to a particular destination IP address and port, the OS will accept incoming packets from only the same remote IP and port. Specifically, when testing the behavior on real network stacks, we find that they will reject a packet with either a wrong IP or port, and respond with an ICMP port unreachable message (as if the packet was a port scan attempt). This effectively prevents the source port of a DNS query from being scanned directly.

In summary, the scannability of source port is dependent on the implementation of DNS software, i.e., whether a `connect()` API call is issued on the UDP socket. Interestingly, we find that out of the three most popular DNS forwarder and resolver software BIND, Unbound, and dnsmasq, only BIND uses `connect()`. Nevertheless, we develop different scan methods that can work for each (described in §4.3 and §4.4, overcoming the challenge outlined in the next section.

4.2 ICMP Rate Limit Challenge

A major hurdle to scan UDP source ports efficiently is the commonly deployed rate limit of outgoing ICMP error messages on endhosts. Even in the simple case where a source port is public-facing and can be scanned directly by any IP address, an attacker's scanning speed is limited by the number of allowable ICMP packets per second (a signal indicating a source port is not in use).

Historically, ICMP rate limit was first recommended to limit the resource consumption on a router (described in RFC 1812 [7]) where an attacker can force it to generate a high volume of ICMP error messages. Today, the rate limit mechanism is universally implemented by all major OSes. Here we focus on the Linux's ICMP rate limiting behavior as it is the most popular server OS, but will briefly describe the behaviors of other OSes afterwards.

For Linux, there are both a per-IP and global rate limit on how many ICMP error packets can be sent out per second. The per-IP rate limit was historically introduced in the very early versions of Linux, i.e., present in kernel 2.4.10. The global rate limit was introduced in kernel 3.18 as a way to alleviate the expensive per-IP rate limit check (e.g., red-black tree operations) [22].

By default, the per-IP rate limit is one per second (with an accrued max burst of 6) which will severely restrict the scanning speed; the global rate limit is effectively 1,000 (with periodic max allowable bursts of 50). Both are implemented in token bucket style, with the per-IP tokens recovering at a rate one per second and the global token recovering at a "nominal" rate of one per millisecond (but the actual token increment happens only after at least 20ms has elapsed since the last increment). The number of available tokens is capped at 50 at all times.

We also tested Windows Server 2019 (version 1809), MacOS 10.15 and FreeBSD 12.1.0, all of which have global ICMP rate limits. Specifically, their limits are 200, 250 and 200 respectively. Besides, none of them has a per-IP rate limit.

4.3 Public-Facing Source Port Scan Method

Even though a source port can be directly probed by any attacker IP in this case, e.g., as in unbound and dnsmasq, it is imperative to bypass the per IP rate limit (present in Linux primarily) to achieve faster scan speed. We develop three different probing methods that can overcome the ICMP rate limit challenge.

① If the attacker owns multiple IP addresses, either multiple bot machines or a single machine with an IPv6 address, then it is trivial to bypass the per IP limit. IPv6 address allocation states that each LAN is given a /64 prefix [33], effectively allowing any network to use 2^{64} public IP addresses. We have tested this from a machine in a residential network that supports IPv6 and picked several IPs within the /64 to send and receive traffic successfully.

② If an attacker owns only a single IPv4 address, it is still possible to ask for multiple addresses using DHCP. We verified that multiple private IPv4 addresses can be obtained in a home network. In addition, we have tested this in an educational network where a single physical machine is able to acquire multiple public IPv4 addresses through this method as well.

③ If an attacker owns a single IPv4 address and the above method fails for some reason (e.g., statically assigned IPs), then the last method is to leverage IP spoofing to bypass the per IP rate limit, and the global rate limit as a side channel to infer whether the spoofed probes have hit the correct source port or not, i.e., with or without ICMP responses. As have been shown in the context of TCP recently, global rate limit can introduce serious side channels [11, 12, 26]. Here we leverage the ICMP global rate limit to facilitate UDP port scans which we describe next.

Figure 3 illustrates this. In observing the maximum globally allowable burst of 50 ICMP packets in Linux, the attacker first sends 50 spoofed UDP probe packets each with a different source IP (bypassing the per-IP rate limit). If the victim server does not have any source port open among the 50, then 50 ICMP port unreachable messages will be triggered (but they are not directly observable to the attacker). If the victim server does have n open ports, then only 50-n ICMP packets will be triggered (as the n UDP probing packets will be silently discarded at the application layer). Now, the attacker sends a verification packet using its real IP address, e.g., a UDP packet destined to a known closed port, such as 1. It will either get no response (if the global rate limit is drained), or an ICMP reply otherwise.

If no port is found in the first batch, the attacker waits for at least 50ms for the rate limit counter to recuperate, and then start the next round. Effectively, the scanning speed will be capped at 1,000 per second. It therefore takes 60+ seconds to enumerate the entire port range consisting of 65536 ports. Nevertheless, it is a winning battle as the attacker can simply repeat the experiment and the probability that one experiment will succeed increases drastically (we note that this is a simple Bernoulli trial).

Time consideration. This approach does have a strong timing requirement. The only thing the attacker has to make sure is to

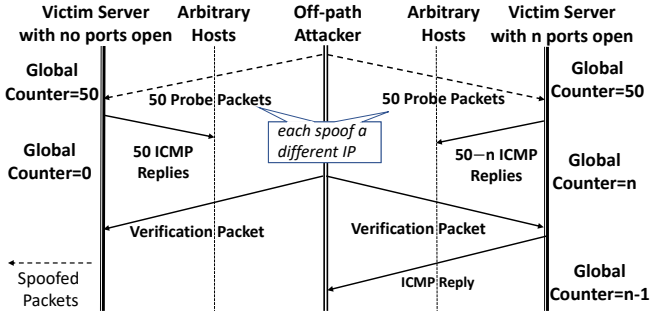

Figure 3: Fast Port Scanning of an Open Source Port

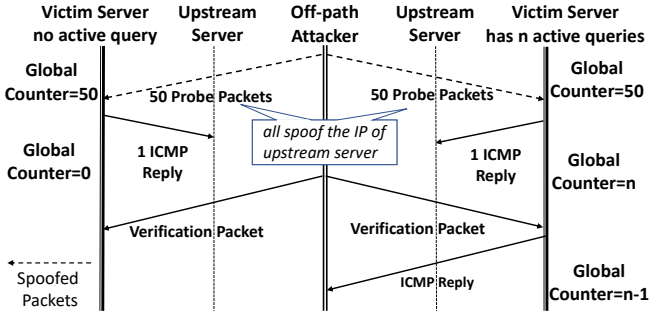

Figure 4: Fast Port Scanning of a Private Source Port

send 50 spoofed probing packets and the verification packet in a burst so that they are all processed within a 20*ms* window; otherwise, the victim may start recovering additional tokens. The other requirement is that the attacker has to wait long enough for the 50 max tokens to recover. If the network condition is not ideal, the attacker can simply wait longer than 50ms.

Binary search to narrowing down to an exact port. Assuming there is a single open port out of the 50 in a specific probing round, we can then employ a simple binary search to quickly narrow down to the exact port. During each round of binary search, we always probe the left half of range first. If it is a match, i.e., 50 spoofed probing packets triggered 49 replies and the attacker can observe one reply to its verification packet, then we continue to search its left half. Otherwise, we assume the port lies in the right half and will conduct a binary search there. Note that we will need to send "padding packets" to ensure the global rate limit is drained when none of the 50 guesses hit a correct port. Padding packets are spoofed packets destined to known closed UDP ports, e.g., 1, that are guaranteed to trigger ICMP replies.

Handling noises. DNS servers usually serve multiple clients at the same time, creating multiple outstanding DNS queries and source ports. As a result, the source port scan will likely discover many irrelevant ports. However, most such queries are transient, and the port scan process can quickly discover an open source port disappearing during the binary search and return to the linear search. In contrast, we assume that the attacker-triggered DNS queries will last significantly longer, e.g., on the order of seconds instead of milliseconds (see §5).

Another source of noises comes from packet losses and reordering. This may lead to both false positives, e.g., loss probing packets or their replies, reordering between verification and probing packets, and false negatives, e.g., lost of the verification packet or its reply (although very rare in practice). To mitigate reordering (which may happen frequently if the jitter is large), we insert a delay, which is empirically determined to be larger than twice the jitter, between probe packets and the verification packet. When false positives do occur, they are handled automatically in the binary search process– it will detect no real port being open and return to linear search.

Even though they can be handled, excessive false positives will drain the per-IP rate limit quickly. Specifically, given the token is recovered at the slow rate of one per second, a false positive rate that is higher than that will force the scan to halt until the token is recovered. Effectively, a per-IP token is a "pass to scan". To solve this problem, the attacker may use two or more real IPs to gain more "passes".

In addition, DNS servers themselves may be subject to random UDP port probing and therefore generate ICMP unreachable messages. This would cause false negatives: we may mistakenly think there is no open port but in fact there is because the verification packet will not trigger any ICMP unreachable replies due to the noise draining the rate limit. Fortunately, not all ICMP replies are subject to rate limit. For example, the most commonly triggered ICMP echo replies are not subject to the limit.

4.4 Private Source Port Scan Method

As described in §4.1, if `connect()` is performed on a UDP socket, the port effectively becomes "private" to the remote peer, invalidating the previous method.

Our idea then is to send spoofed UDP packets with the source IP of the upstream DNS server. In the example of a DNS resolver being the victim, we can send UDP packets probing different source ports with spoofed IP of the authoritative name server. If it hits the correct source port, then no ICMP reply will be generated. Otherwise, there will be. We can then use the same global ICMP rate limit as a side channel to infer if such an ICMP message has been triggered. At first glance, this method can work but at a low speed of *one port per second*, due to the per-IP rate limit on ICMP messages.

Surprisingly, after we analyze the source code of the ICMP rate limit implementation, we find that the global rate limit is checked prior to the per-IP rate limit. This means that even if the per-IP rate limit may eventually determine that no ICMP reply should be sent, *a packet is still subjected to the global rate limit check and one token is deducted*. Ironically, such a decision is consciously made by Linux developers to avoid invoking the expensive check of the per-IP rate limit [22], involving a search process to locate the per-IP data structure.

This effectively means that the per-IP rate limit can be disregarded for the purpose of our side channel based scan, as it only determines if the final ICMP reply is generated but has nothing to do with the global rate limit counter decrement. As a result, we can continue to use roughly the same scan method as efficient as before, achieving 1,000 ports per second. Figure 4 illustrates the slightly modified scan workflow. Similar to Figure 3, the attacker first sends 50 probes where this time all of which uses the spoofed IP of the

Table 1: DNS Forwarder Behaviors in Home Routers

Router	ICMP Reply	Global ICMP Rate Limit	Using `connect()`	Spoofing Public IP in LAN	Vulnerable
Verizon Fios Gateway (G1100)	Y	N	Y	N/A	N
Xiaomi (R3)	Y	N	N	Y	Y1
Huawei A1 (WS826)	N	N/A	N/A	N/A	N
Netgear (WNDR3700v4)	Y	N	N	N	Y2
Arris Spectrum Gateway (TR4400)	Y	N	N	Y	Y1
TP-Link (Archer C59)	Y	N	N	Y	Y1

Y1: vulnerable to an insider attack. Y2: vulnerable to an attack requiring collaboration between an insider and outsider.

upstream server. Due to per-IP rate limit, the victim server will always generate only one ICMP reply (in steady state) as long as there is at least one inactive port scanned, which is the case in both the left and right side of the figure. In the case where the 50 probes hit n private open ports (to the upstream server), the global rate limit counter still decrements to n because the victim attempted to generate $50 - n$ ICMP replies. In contrast, when all 50 probes hit inactive ports (left side of the figure), the counter decrements to 0.

The rest of the procedure is identical as before, where a binary search can be launched to narrow down to a specific port.

Influence on public-facing source port scan. With this knowledge, we can improve method ③ in §4.3 as follows: instead of spoofing 50 different IPs in each round of probing, we only need to use a single spoofed IP (or a 2nd IP the attacker owns) instead of many different IPs (which sometimes can be a hurdle).

Handling noises. We point out that there is inherently less noise in this scan compared to the one on public-facing source ports. This is because every source port is now effectively "open" to only one single remote IP which is originally specified in `connect()`. Therefore, assuming the victim is a resolver, most of its queries (i.e., noise) will be destined to a different name server than a specific attack target. Other noise conditions such as packet loss and reordering still apply. Similarly, noise handling techniques also apply (e.g., using more than one IP to alleviate the per-IP ICMP rate limit).

4.5 Vulnerable DNS Forwarder and Resolver Population

A forwarder or resolver is considered vulnerable if the UDP source port of a DNS query can be inferred successfully, or more specifically if it supports the global ICMP rate limit, and/or if it does not use `connect()` (which makes the port public).

Vulnerable Forwarders. We surveyed six home router devices, all of which act by default as a forwarder supporting DNS caching. Their behaviors are summarized in Table 1.

Only one router (Huawei A1) fails to respond with even the ICMP port unreachable message, which is a basic requirement of the port scan. The Verizon Gateway is not vulnerable because it is the only one using `connect()` yet without the global rate limit. We find that all routers are running old Linux kernel versions in the range of 2.6 to 3.10, which is why global rate limit is not observed. We do believe that routers of newer generations will eventually inherit the global rate limit. Nevertheless, since most of them do not use `connect()` on the UDP socket, the source port of a DNS query can be easily probed without leveraging the side channel based on the global ICMP rate limit. In addition, we also measured the IP spoofing capability within the LAN network. Specifically, if an attacker can spoof the public IP of the resolver from within the LAN network, which often operates on a private IP range, the end-to-end attack can be conducted from a machine in the LAN alone without any external collaborator. The result shows that three routers fall under this category (Y1), and one can be attacked from an outside machine capable of spoofing the resolver's IP (Y2).

Vulnerable Resolvers. We study a list of 14 popular DNS providers shown in Table 2 and show that 12 of them are vulnerable which is very serious. Interestingly, we find that due to firewall policies encountered in several providers, the source port of the probing packet must be set to 53 and the destination port should be in the ephemeral port range in order to trigger ICMP responses on some servers.

Note that we also report the number of backend server IPs behind the anycasted frontend IP (e.g., 8.8.8.8). These backend IPs correspond to the reachable servers on which we can scan ports. The presence of multiple such IPs increases the attack's difficulty as we need to decide which IP(s) to scan. To discover the backend IPs, we simply send 100 queries from the same machine to the frontend and record the observed IPs at an authoritative name server that we own. For the cases where we encounter only a few IPs, we can simply scan all of them simultaneously. For the cases of OpenDNS and AliDNS which have over 100, we discuss possible techniques to handle them later in §6. Note that OpenDNS and AliDNS exhibit more than 100 IPs because our authoritative name server intentionally discards incoming queries and they decide to retry with potentially new IPs every time before giving up.

In addition, we also measured the general population of open resolvers. Compared with public resolvers, which are usually advertised and intended to serve the public, open resolvers, however, are generally unlisted and are intended to serve smaller numbet of clients. We obtain a list of open resolvers from Censys [23] and managed to probe a set of 138,924 live IPs, among which there are 70,503 whose backend and frontend IPs are identical, indicative of the absence of anycast. Further, 41.3% of the 138,924 cases generate ICMP replies (following the same practice of using source port 53 in the probing packets), out of which 67.56% exhibit a global rate limit, and 53.93% use connect() on the socket. Overall, 34.36% of all cases are vulnerable because they either support the global rate limit or do not use `connect()`). Most of them are not vulnerable simply because of the lack of ICMP replies.

5 EXTENDING THE ATTACK WINDOW

The longer the attack window, the more ports an attacker can scan, and also more time to inject rogue records. Therefore, our goal is to "mute" upstream servers and prevent them from being able to

Table 2: Popular Public Resolver Behaviors

Name	Address	Example Backend Addr.	# of Backends	ICMP	Global Rate Limit	Using `connect()`	Vulnerable
Google	8.8.8.8	172.253.2.4	15	Y	Y	N	Y
CloudFlare	1.1.1.1	172.68.135.169	2	Y	Y	Y	Y
OpenDNS	208.67.222.222	208.67.219.11	107	Y	Y	Y	Y
Comodo	8.26.56.26	66.230.162.182	2	Y	Y	N	Y
Dyn	216.146.35.35	45.76.11.166	1	Y	Y	N	Y
Quad9	9.9.9.9	74.63.16.243	11	Y	Y	Y	Y
AdGuard	176.103.130.130	66.42.108.108	3	Y	Y	N	Y
CleanBrowsing	185.228.168.168	45.76.171.37	1	Y	Y	Y	Y
Neustar	156.154.70.1	2610:a1:300c:128::143	2	Y	Y	N	Y
Yandex	77.88.8.1	77.88.56.132	19	Y	Y	Y	Y
Baidu DNS	180.76.76.76	106.38.179.6	16	Y	Y	Y	Y
114 DNS	114.114.114.114	106.38.179.6	11	Y	N	N	Y
Tencent DNS	119.29.29.29	183.194.223.102	45	Y	N	N	N[1]
Ali DNS	223.5.5.5	210.69.48.38	160	N	N/A	N/A	N

[1] Though meeting the requirements, it is not vulnerable due to interference of fast UDP probing encountered (likely caused by firewalls).

respond to the DNS queries triggered by the attacker. Depending on the attack target (i.e., a forwarder or resolver), we come up with two novel strategies. Ironically, one of the strategies again leverage the "rate limiting" feature commonly deployed at the application layer, which can be turned to the attacker's advantage.

5.1 Extending Window in a Forwarder Attack

We propose a novel strategy as follows: the attacker first sends a query of his own domain, e.g., *www.attacker.com* to the forwarder, which will eventually trigger the upstream resolver to query the attacker-controlled authoritative name server. The name server is intentionally configured to be unresponsive so that the forwarder would wait maximum amount of time possible (as the resolver is also halted) while leaving an open source port. At a first glance, this is pointless because we are not interested in poisoning an attacker's own domain. However, due to the unique role of DNS forwarders [34], they rely completely on upstream resolvers to perform validations on responses.

More specifically, according to RFC 8499 [34], recursive resolvers' responsibility is to handle the complete resolution of a name and provide a "final answer" to its client. This includes recursively handling referrals and CNAMEs and assemble a final answer, including any CNAME redirects by design. More importantly, resolvers are required to perform integrity checks such as the bailiwick check [25], whereas forwarders are not. This means that forwarders by design trust the upstream resolvers and its response. This is not a security flaw; rather, it is a design choice to prevent forwarders from duplicating the work of resolvers. This observation is also made a in recent study dedicated to the security of DNS forwarders [60].

As a result, a rogue response (potentially injected by an attacker from either LAN or outside) shown in Figure 5 will be accepted by a forwarder and both the attacker's and victim's domain records will be cached. This strategy is extremely effective because we can impose the maximum wait time on the forwarder (i.e., creating the largest possible attack window). Specifically, most forwarders have a very lenient timeout (sometimes close to a minute e.g., in dnsmasq), and will stop mostly because the upstream resolver failing first (ranging from 5 to 30 seconds) generating a `SERVFAIL` response (or `NXDOMAIN`) message. To prevent resolvers from generating such messages too early, we also employ a technique that can sometimes keep a resolver engaged longer. The trick is to have the attacker-owned authoritative name server respond in a slow pace with a chain of CNAME records, creating an illusion that it is making progress. This can delay resolver's response for over a minute in some cases (e.g., CloudFlare).

Answer	`www.attacker.com`	CNAME	`www.victim.com`
	`www.victim.com`	A	`1.2.3.4`

Figure 5: Example Rogue Response Acceptable by a Forwarder (the victim domain record also cached)

5.2 Extending Window in a Resolver Attack

We propose to take advantage of the security feature of rate limiting in authoritative name servers, as a way to mute name servers and extending window in a resolver attack. Modern DNS name server software such as BIND, NSD, PowerDNS, all support a common security feature called response rate limiting (RRL) [57, 59], as a mitigation of the DNS amplification attack [57] where a large number of malicious DNS queries are issued to authoritative name servers spoofing a victim's IP address. To limit the number of amplified DNS reply packets, the RRL feature allows a configurable per-IP, per-prefix, or even global limit of triggered responses. Specifically, if the limit is reached, then responses are either getting truncated or dropped. There are also dedicated DNS firewalls with similar features [14].

Ironically, this feature can be leveraged maliciously to mute a name server if an attacker can inject spoofed DNS queries (with the target resolver's IP) at a rate higher than the configured limit. Depending on the actual limit (some are configured to be very low), it may be trivial to create a sufficiently high "loss rate" so that the resolver's legitimate query has an extremely low probability of getting a response. To understand how likely such a strategy can

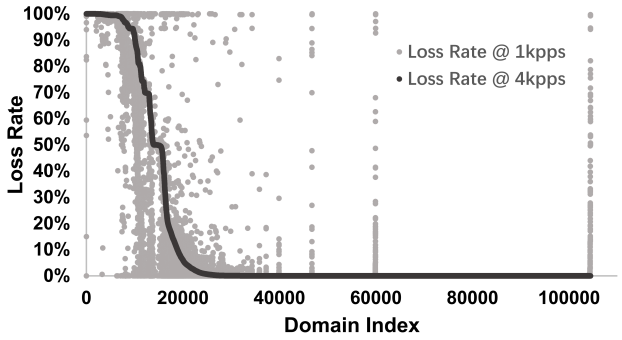

Figure 6: Response Loss Rate under Different Query Rate

succeed, we conduct an experiment to measure the response rate of name servers used by top 10K Alexa websites.

Measurement methodology. To trigger RRL, we send 1K queries per second for 15 seconds, followed by another around of 4kpps test of 15 seconds to each name server IP; the two tests are separated by a two-second gap to avoid interference. If there are multiple name servers for a given domain, we pick the first one. In both cases, the queries are uniformly distributed (instead of sent in bursts) all attempting to ask the A record of the `www` subdomain. The rational is that 1kpps and 4kpps represent sufficiently low throughput, roughly 0.6Mbps and 2.5Mbps respectively, which is easily achievable by any attacker on the Internet.

Ethical considerations. We consciously took a number of measures to limit the impact on the operations of these servers. First, we ask for A records in our queries, which generally result in smaller responses, to conserve the target network's resources; yet, a prior report [58] suggests that rate limiting behaviors are generally agnostic to the type of queries (so this would not impact the result of our measurement). Second, the domain names in the queries are always the same, resulting in minimal processing overhead on the server (the result is likely cached in memory and easy to fetch). Third, we choose to send evenly spaced queries (instead in burst) to avoid stressing the server. In general, the traffic of 4kpps is small compared to a normal load experienced by a name server of a Top Alexa site. Finally, we set up a web server on the IP address used to conduct the probing, serving a webpage with opt out instructions (we also configured the reverse DNS name of the IP to direct visitors to our webpage). In total, we received and honored four requests.

Results. We sort the domains by the loss rate observed in the 4kpps test in descending order and present the results in Figure 6. Overall, there are about 25% domains whose name servers experienced higher than 1% loss rate. This is in line with a recent measurement reporting about 17% cases with loss behaviors [20]. The difference is likely due to their lower rate of queries at 500pps.

We now try to analyze what fraction of these domains are vulnerable (can be muted successfully). Here we define a domain to be vulnerable if its name server exhibits an induced loss rate of 66.7% or higher; the threshold is determined empirically as will be discussed in §7.2. Specifically, there are 13,110 domains that would already satisfy the criteria and fall victim to a simple DoS attack at a rate of 4kpps.

Figure 7: DNS Response Used to Overwrite Cache

Field	Value
Question	`{nonce}.www.victim.com`
Answer	
Authoritative	`www.victim.com NS ns.attacker.com`
Additional	

In addition, we also inspect the remaining cases where the loss rate increased from the 1kpps test to the 4kpps one. There are roughly 5,000 cases where the diff is 2% or higher. We believe that the majority of them can be further increased given increased probe rate, and therefore potentially vulnerable as well. Therefore, we have a total of 18110 (13,110 and 5,000) cases out of the 100K (18%) which we consider vulnerable.

Finally, out of the 75% cases where both 1kpps and 4kpps tests experienced no loss, we believe there may be many more vulnerable cases which we simply cannot uncover due to the relatively low probing speed. Due to ethical concerns, however, we refrain from probing at an even higher speed. To peek into those cases, we manage to obtain permission from a collaborator to test an authoritative name server configured for non-profit website. We are able to probe the server at a much higher rate (late at night to avoid disruption). Initially when probed at a rate 4kpps, no loss is observed. Interestingly, it started to experience loss when the probing rate is increased to 25kpps. Specifically, when the rate is increased to 50kpps, the loss rate jumps to 75%. We checked with our collaborator on whether the server is indeed configured to use such a high rate limit. To our surprise, there is no rate limit configured at all. To understand this behavior, we replicate a BIND server locally (replicating the configuration) and verified that indeed it is fairly easy to trigger high loss rate with comparable probing speeds. We find that it is because the application (i.e., BIND) not reading from the socket queue fast enough, which causes overflows. Indeed, historical DoS attacks similar to this, e.g., by flooding queries with random names, have been observed in practice [44]. To mitigate such threats, the official BIND explicitly guideline recommends rate limit [52], which would paradoxically make it vulnerable to our attack instead.

In addition, we can leverage this technique to extend the attack window against a forwarder since RRL is also deployed on resolvers to limit the rate of incoming queries. By following the same procedure and ethical standard in the previous measurements and a rate of 4kpps probing against the resolver IPs obtained on 14, 2019 from Censys [23], we observe surprisingly 121,195 out of 136,547 exhibit a loss rate of more than 66.7%, indicating it is generally possible to mute resolvers on the Internet.

6 PRACTICAL ATTACK CONSIDERATIONS

Bypassing the TTL of cached records. If an attacker attempts to poison a benign domain such as `www.victim.com` by directly triggering DNS queries of `www.victim.com` on a resolver, it may cache the unwanted legitimate A record, for example, due to occasional failures to mute their upstream servers. This forces the attacker to wait for the cache timeouts before initiating the next attack attempt.

However, according to a recent study [41], the cached A record of `www.victim.com` could be overwritten by injecting a non-existent NS record of `www.victim.com`. Specifically, an attacker always sends queries asking for A records of domain names with random prefixes, e.g., `{nonce}.www.victim.com` where {nonce} is a random value. This forces the resolver to initiate a new query to the authoritative name server of `victim.com` as the record is not cached. Then the attacker attempts to inject a rogue response as shown in Figure 7, claiming that `www.victim.com` is a standalone zone with its own authoritative server `ns.attacker.com`. The resolver will then query `ns.attacker.com` for all future requests asking for A record of `www.victim.com`, after the original cached record expires. This is because the attack has effectively inserted a new NS record of the `www.victim.com` zone. And resolvers are by design advised to use the most accurate delegation it has in the cache, which in this case is the NS record of `www.victim.com` instead of the one of `victim.com` [41]. We have verified that this method works against the latest versions of both BIND and Unbound.

Timeouts and retransmitted queries. When a DNS query is triggered either on a forwarder or resolver and there is no legitimate reply received from their upstream, they will not wait forever. Most of them have a timeout determining when to *close the current socket (and therefore the corresponding source port)* and retransmit. This means that some of these source ports may be short-lived and difficult to catch. Therefore, it is important to understand their behaviors in more depth.

In most DNS software such as BIND and Unbound, we conduct controlled experiments with the help of documentation and source code analysis, and summarize their behaviors (which generally match what we observe in real resolvers). Specifically, when configured as forwarders, they have a similar behavior to resolvers but typically have a longer timeout (and it is generally easier to extend their attack window using different strategies). So, we focus on resolvers' behaviors below.

In the case when there is no failure, both BIND and Unbound maintain a default retransmission timeout (RTO) — 0.8s for BIND, and a dynamically computed value based on RTT (to the authoritative name server) for Unbound. If timeouts occur (e.g., the name server is unresponsive or muted), they will contact another name server in a round robin manner if more than one is available. If all of them failed to respond, they will exponentially back off by doubling the RTO — BIND starts the backoff only after 3 consecutive failure whereas Unbound does it after every failure). Finally, there is another hard-stop condition — default 10s total wait time for BIND and 16 to 32 trials for Unbound (depending on the type of query). A `SERVFAIL` will be sent back to the client if the hard-stop condition is met.

Here we refer to the RTO as the "attack window" as it represents the duration where a source port remains unchanged. When the window ends, a different source port will be chosen, nullifying any previous port scan progress — a new port may happen to pop back into the range that is just scanned. It is important to note that when the attack window is too small (e.g., 1s), even if the port is correctly identified, it will still take time to inject 64K rogue DNS records (at a flooding rate of 100kpps, it may still take a few hundred milliseconds), which may not finish before the window closes.

Generally speaking, if the authoritative name server is muted for an extended duration, we do expect to see larger attack windows (as RTOs double over failed attempts). With BIND being more reluctant in doubling the RTO and having a tighter hard-stop condition (default 10s), we believe it is a more difficult attack target. We describe an experiment against such a difficult case in §8.

Handling multiple authoritative name servers. Many domains in practice are configured with multiple authoritative name server IPs, for redundancy and security. Some consider this as a specific defense against DNS cache poisoning attacks against resolvers (called "IP randomization") [31], as it increases the randomness of a DNS query. According to a recent measurement study [49], second level domains like example.com under TLDs like .com, .net and .org have a median of only 2 NS only (and a mean of 2.3, 2.4, and 2.4); therefore this is not a strong defense by itself.

There are two ways to handle this. First, a general strategy is to simultaneously mute all the authoritative name servers, given that on average few of them exist. This will help the RTO to grow exponentially after a resolver experiences repeated failures when contacting all the name servers.

Second, if a resolver is Unbound, it has a unique behavior where it will stop contacting a name server (blacklisting the server) and switch "permanently" (i.e., minutes) to other available ones, should it repeatedly fail to hear from the originally-contacted server [31]. The authors in [31] therefore take advantage of this behavior to perform what they call "name server pinning". In our case, we need to allow periodic successful responses (by suspending the muting process); this is to avoid the last name server being blocked as well.

Handling multiple backend servers behind DNS resolvers As described in §4.5, many public DNS resolvers have multiple backend servers (with different IPs) that perform the actual queries. Interestingly, we find that the backend server selection is typically heavily skewed towards a few (even when we do see 100+ in total for some providers), likely determined based on location and past performance measurements. This allows us to focus on only a few IPs at the same time, which is easily achievable consider each IP only requires a scan traffic of 1kpps.

7 END-TO-END ATTACKS

In this section, we evaluate our attack in realistic settings, including a forwarder used in a home, and a production resolver with a realistic configuration and network conditions.

7.1 Attacking a Forwarder (Home Router)

Experiment Setup. Given that most vulnerable routers have a fairly similar behavior shown in Table 1, we choose Xiaomi R3 (a Wi-Fi home router) as a representative case study to launch end-to-end attacks. It is used as the one and only gateway in an actual home where 10 to 15 devices are connected to the Internet through the wireless router all the time. In addition, Xiaomi R3's upstream DNS server is set to CloudFlare DNS (`1.1.1.1`). Its DHCP server is by default configured to provide 253 IPv4 addresses in a /24 network. Finally, the attack machine is a Raspberry Pi, which also connects to the router wirelessly.

Since Xiaomi R3 does not deploy global ICMP rate limit and its forwarder software does not call `connect()` on UDP sockets, we

use strategy ② in §4.3 (obtaining multiple IPs through DHCP) to bypass its per-IP rate limit. For extending the attack window, we use strategy 1 as described in §5.1 with a malicious name server.

Attack Process. The attack is divided into two phases, In Phase I, the attacker tries to acquire 240 IP addresses using the DHCP strategy. Afterwards, the attack goes into Phase II where the following repeats: the attacker issues a query to the forwarder asking for an arbitrary subdomain, e.g., `nonce.attacker.com`. If `SERVFAIL/NXDOMAIN` is received or if an attacker has waited for longer than 1 minute, indicating something is wrong, we will repeat the attack process by issuing another query. Otherwise, if a `NOERROR` response is received, it means a forged response is injected successfully. In Phase II, the attacker uses acquired IP addresses to scan open ports on the router. We rotate among the available IPs and make sure that we never go above the per-IP rate limit (which is 1pps in steady state). After a port is found open, we confirm that it stays open for at least one second by repeatedly probing the same port. If it does, we start injecting rogue responses. The experiment is repeated 20 times and we report the success rate, average time-to-succeed, and other statistics.

Results. Overall, the attack is very effective, with a success rate of 100% out of the 20 experiments (we consider it a success if the attack finishes within 30 minutes). The average time-to-succeed is 271s, with a breakdown of 103s in Phase I and 168s in Phase II. The standard deviation of Phase II is 109s with the maximum of 739s and the minimum of 83s. The variance is large because the attack time is mainly determined by the attack window size, which is the timeout before a resolver decides to give up and return SERVFAIL/NXDOMAIN, as mentioned in §5.1, and the timeout on CloudFlares' resolver varies a lot (from seconds to more than one minute for unknown reasons). Also, the attack needs to scan 36,325 ports on average to succeed; the average port scan speed is 210pps, which roughly matches the expected rate of 240pps when using 240 IPs to scan. Besides, the attack generates 78 MB of traffic.

7.2 Attacking a Production Resolver

Even though the attack can work in principle against a large fraction of public DNS resolvers, due to obvious legal and ethical concerns, we refrain from targeting any of them. Fortunately, we obtained authorization to test the attack against a production resolver managed by a collaborator.

Experiment Setup. The resolver processes about 70 million queries daily with thousands of real users across multiple institutions and is configured as an *open* resolver. Because of this, it will be noisy and representing a challenging attack target. Another behavior noteworthy is that it has two backend servers, both of which appear to use `connect()` on the UDP sockets. Interestingly, we were told that they are running `Unbound`, and we suspect that the `connect()`-like behavior can be due to stateful UDP firewalls responsible for filtering out-of-state packets. We are given an attack machine in an adjacent network — 4 hops away from the resolver, which has a 1Gbps Ethernet and can perform IP spoofing.

Also, we setup a test domain and host it on an authoritative server controlled by us so that we poison only our own test domain. We configure the BIND software with a response rate limit at a low rate of 10pps to minimize the impact on the network. Once the limit is reached, we allow 1 out of 5 responses — an effective loss rate of 80%. This forms the setup of *baseline* experiments, and we have conducted 20 rounds of them one in daytime and the other after midnight local time (as shown in Table 3). In addition, to understand the effect of response rate limit on the authoritative name server, we vary the mute level by allowing a loss rate of 75%, 66.7%, to 50% — the lower the loss rate, the more difficult the attack is.

As a comparison, we also simulated more realistic network conditions by imposing additional delay, jitter, and loss on the same attack machine. The exact numbers are presented in Table 3 where the *baseline* represents the unmodified network condition and *altered* represents the simulated condition. We take the numbers with reference to recent Internet measurements [27][15]. We believe an attacker is likely able to find networks with even better conditions. To deal with increased false positives caused by the simulated network condition, we used two IPs to launch the attack in the *altered* experiment; this is to avoid halting the scan too frequently due to the per-IP token being drained (see §4.3).

Finally, we are also interested in understanding the influence of the parameter "name server mute level", on the viability of the attack and will conduct a controlled experiment varying the "mute level" where all other parameters are the same as those in *baseline*.

Attack Process. The process similarly starts from the attacker generating queries asking for `nonce.attacker.com`. Since the resolver has two backend server IPs, we launch the port scans on both IPs simultaneously. At the same time, we mute all authoritative name servers with queries at a rate of 20pps so that the resolver will experience a constant loss rate of 80%. The experiment is repeated 20 times and 5 times for the *baseline* and *altered* respectively.

Results. As shown in Table 3, we achieved a perfect 100% success rate for the first *baseline* experiment Base(D) (at daytime), with an average time of 504s to succeed. The standard deviation is 399s with the maximum being 1404s and the minimum being 13s (which is simply due to luck). On average, only 69 MB of attack traffic is generated, which is similar to that in the forwarder attack even though resolver attacks take much longer to succeed. This is because a forwarder attack is much more likely to enter the TxID bruteforce phase (6 times vs. twice), which generates about 10 MB of traffic every time. Specifically, strategy ② used in the forwarder attack does not have a binary search phase and an open port is simply confirmed twice before it enters the TxID bruteforce phase whereas the binary search phase employed in the resolver attack checks repeatedly the existence of an open port.

After inspecting the detailed log, we found that even though Base(D) experiment has a near perfect network condition, many more packets were sent compared to the forwarder attack. This is because of the frequent change of source ports caused by either resolver retries (i.e., RTOs) or new queries initiated by the attacker (if the resolver happens to receive a legitimate response), resulting in many small and fragmented attack windows. In fact, we find more than half of these fragmented attack windows to be smaller than 1 second, making them undesirable. Interestingly, we do find a decent fraction of large attack windows (10% of them with a 30s or larger). Such long attack windows match the profile of an Unbound resolver — 16 maximum allowed retransmissions, each doubling the RTO. In §8, we demonstrate that a BIND attack with much smaller

Table 3: Production Resolver Attack Results

Exp.	RTT range	Probe loss	Name sever mute level	Average time taken	Success rate
Base(D)	0.2-1.2ms	~0%	80%	504s	20/20*
Base(M)	0.2-1.2ms	~0%	80%	410s	20/20*
Mute Lv.	0.2-1.2ms	~0%	75%	1341s	18/20*
Mute Lv.	0.2-1.2ms	~0%	66.7%	2196s	20/20#
Mute Lv.	0.2-1.2ms	~0%	50%	8985s	9/20#
Altered	37-43ms	0.20%	80%	930s	5/5*

*: 1-hour threshold. #: 3-hour threshold. D: Day. M: Midnight

attack windows appears to be still feasible but taking much longer time to succeed.

As shown in Table 3, the Base(M) experiment has the same exact setup as the Base(D) except that it is conducted after midnight where background traffic and noises will be generally lower. We observe the same 100% success rate and the average time to succeed decreasing from 504s to 410s. This is expected as our attack is sensitive to noises.

In addition, for the *mute level* experiments shown in Table 3, all but 50% mute level (i.e., loss rate) can still achieve a near perfect success rate and can finish generally within an hour (note the threshold of success being 3 hours for the 66.7% mute level). For 50% mute level, the attack succeeded only 9 out of 20 cases. Moreover, the average time taken is 8,985s or 2.5 hours.

Finally, for the *altered* experiment, we also achieved a perfect 100% success rate. Specifically, the time to succeed is 2005s, 538s, 792s, 1287s and 29s respectively. On average, the attack time is 930s and 131 MB of traffic is generated. Note that the scan speed in the *altered* experiment is higher than that in the *baseline* experiment. This is because we used two IPs in the *altered* experiment, reducing the frequency of halting during scans.

We also find that the increased loss rate and jitter causes more false positives, where we incorrectly consider a port discovered (as the verification packet successfully solicits an ICMP). This is commonly caused by any loss of probing packets which can create two problems: (1) we waste much time filtering these false positives during the binary search stage, reducing the effective scanning speed; (2) The scan can still be halted because of frequent draining of the per-IP ICMP tokens even though we used two IPs.

8 DISCUSSION

Attack against Unbound vs. BIND. As mentioned previously, a BIND attack would be much tougher than Unbound as most of the fragmented attack windows will be generally smaller, as it is more reluctant in doubling the RTO and have a tighter hard-stop condition (as discussed in §6). To understand if is ever feasible to attack a BIND resolver, we construct an extreme experiment with 4 name servers, and a default hard-stop condition of 10s wait time on the BIND resolver, resulting in the resolver almost always stuck in a small attack window of 0.8s, as querying 4 name servers for 3 rounds already take 9.6s (before the RTO backoff can kick in). The experiment is conducted in a similar network environment to `baseline`. Surprisingly, we run the experiment twice and both succeeded (one in 0.54 hours and the other in 1.25 hours). We find that it is indeed possible to succeed in scanning a port as well as injecting rogue records all in a 0.8s window. One attack we inspected showed that the port scan took 600ms and the record injection took 200ms.

UDP source port inference on other operating systems. In addition to Linux, we have verified that other major OS kernels are vulnerable as well, albeit with lower global rate limit — 200 in Windows and FreeBSD, and 250 in MacOS. It is concerning that not a single OS is aware of the side channel potential of global rate limits, despite the recent serious side channels specifically leverage a challenge ACK global rate limit in TCP [11]. We argue that all global rate limits in networking stacks need to be scrutinized regardless of their original design goal. We believe this work can serve as another valuable reference.

Other vulnerable protocols. Any protocols based on UDP are affected by the source port inference. A prominent example is QUIC [37] and HTTP/3 [48] which are poised to replace the traditional TCP-based web protocols with a much more efficient UDP-based protocols. They are already widely deployed in Google's web services [56]. In addition, VoIP, video streaming, and delay-sensitive online games may also use UDP, which are subject to port inference, and even off-path packet injection attacks.

Best practices in configuring response rate limiting (RRL). Even though response rate limit on authoritative name servers is an important mitigation against DNS reflection/amplification attacks, if not done carefully, it can allow the extension of attack window in a DNS cache poisoning attack. We endorse the RRL behavior (which was configurable but not always used) where a server still responds with truncated messages when a rate limit is reached [59] instead of being silent. This way, the amplification factor is no longer favorable to a DDoS attacker. Yet, it sends a strong signal to the resolver indicating something bad is going on, and the resolver should immediately react, e.g., either switching the source port and sending a new query, or falling back to TCP altogether (as recommended in [59]). This strategy can reduce the susceptibility of RRL being maliciously taken advantage of, compared to the cases where a server is completely muted (with 100% loss). Unfortunately, as we show in the resolver attack, even a 66.7% drop rate would already make a server vulnerable, not to mention that a determined attacker with more resources can simply flood the server with expensive queries (e.g., to non-existing domains [44]).

8.1 Defenses

The proposed attack is fundamentally an off-path attack and therefore can be mitigated by additional randomness and cryptographic solutions. Besides DNSSEC and 0x20 encoding, there is also an emerging feature called DNS cookie that is standardized in RFC 7873 [1] in 2016. At a high level, it requires both client and server to exchange some additional secrets unknown to an off-path attacker; it therefore has the potential to defeat most (if not all) off-path DNS attacks. Note that this feature requires both resolvers and authoritative name servers to upgrade in order to see benefits. As of now, only BIND has implemented this feature and have it turned on by default in 9.11.0 forward [28] (released in 2016). We find about 5% of the open resolvers that we measured have enabled this feature by default. However, as any other unproven technology (the lesson

regarding 0x20 [16]), it remains to be seen if issues such as compatibility will prevent it from being widely adopted. Interestingly, we already found both DNSPod (operated by Tencent) and a resolver in a private company drop queries with DNS cookie options, likely for compatibility concerns.

In addition, our attack relies on the two fundamental components: (1) inferring source port of a DNS query; (2) extending attack window. Each of them can be a security threat on its own and therefore we discuss how to address both.

For (1), the simplest mitigation is to disallow outgoing ICMP replies altogether (as is done by many servers), at the potential cost of losing some network troubleshooting and diagnostic features. Otherwise, we need to address the global rate limit. As with patches on TCP global counters [51], we suggest a randomized ICMP global rate limit, including possibly randomizing the max allowable burst (currently 50), minimum number of tokens recovered each time (currently 20), minimum idle time to recover tokens (currently 20ms), and number of token recovered per time unit (currently 1 per millisecond). When the side channel is mitigated, we also recommend resolvers adopt the use of `connect()` on their UDP sockets so that their source ports will not be public-facing and directly scannable.

For (2), we have discussed best practices to use RRL to prevent an attacker from muting authoritative name servers easily. Other simple mitigation strategies include: (1) setting the timeout of DNS queries more aggressively (e.g., always below 1s). This way, the source port will be short-lived and disappear before the attacker can start injecting rogue responses. The downside, however, is the possibility of introducing more retransmitted queries and overall worse performance. (2) Employing anycast to make it harder for an attacker to DoS a specific authoritative name server used by a victim resolver.

9 RELATED WORK

DNS blind forgery attacks and cache poisoning. After the major DNS cache poisoning attack presented in 2008 [39], many defenses have been applied, making such blind off-path attacks much more difficult. Several studies continued to investigate the feasibility of new attacks in the presence of state-of-the-art defenses. For example, Herzberg and Shulman [30] proposed a method to derandomize the source port of a resolver behind NAT by occupying all but one port on the NAT with the help of a dummy machine in the same network as the resolver, and also proposed a nameserver-pinning method leveraging IP fragmentation. Unfortunately, the attack is not applicable for resolvers that own a public IP address (which we believe to be the common case). Alharbi et al. [5] conducted a similar attack to exhaust the local ports on a client machine and poisoned the OS-wide DNS cache.

Later, another work by the Herzberg and Shulman [31] proposed a novel IP fragmentation technique to target resolvers. This technique eliminates the requirements of guessing randomized source port number, server address, and query name, etc. Instead, only IPID becomes the secret that needs to be guessed. The key assumption is the response of a victim domain is voluntarily fragmented (e.g., when DNSSEC is enabled). A more recent work by Brandt et al. [9] relaxed the constraint by injecting ICMP fragmentation needed error messages to an authoritative name server to proactively lower its MTU (with respect to the resolver) and induce fragmentation. The attack depends on the exact server configuration, as many simply reject such ICMP packets or uphold a minimum MTU larger than needed to fragment a DNS response. In addition, predicting IPID precisely has become more challenging over time as more randomness is introduced [4]. Despite this, Wang et al. [60] developed a novel attack targeting DNS forwarders by forcing fragmentation using attacker-owned authoritative name servers.

Overall, different from previous works, we leverage a universal network side channel based on the ICMP global rate limit, which we show to be prevalent. In addition, our attack also works perfectly against all layers of DNS caches (not just resolvers).

Network side channel vulnerabilities For decades, researchers have been using network side channels to infer sensitive network information, e.g., port scans [26], TCP sequence number inference [11, 45, 50], and others [3, 42].

The only work that can be classified as a side channel in the DNS source port inference is by Herzberg and Shulman [32]. The authors proposed to use low-rate bursts of packets to overload specific source ports on resolvers that can potentially cause the legitimate DNS reply destined to those ports getting dropped. This creates a timing side channel — longer end-to-end response time (observed by the malicious client triggering the request) indicates that a port is used, and shorter indicates it is not. Unfortunately, this requires an attacker and the resolver to be co-located in low-latency network environment, e.g., LAN, due to its sensitivity to network noise. In contrast, our source port scan technique is much more direct and reliable and hence can be carried out from far away.

In addition, the concept of leveraging global rate limit in network protocols as a side channel has been documented in several important works. For example, TCP RST rate limit [12, 26], TCP challenge ACK rate limit [11] have been demonstrated and reported in the past. The ICMP rate limit is in principle no different, although it is perhaps even more subtle as it shows up during interactions across layers (i.e., UDP and ICMP).

10 CONCLUSION

This paper presents a novel and general side channel based on global ICMP rate limit, universally implemented by all modern operating systems. This allows efficient scans of UDP source ports in DNS queries. Combined with techniques to extend the attack window, it leads to a powerful revival of the DNS cache poisoning attack, demonstrated with real-world experiments under realistic server configuration and network conditions. Finally, we suggest practical mitigations that can be used to raise the bar against such attacks.

11 ACKNOWLEDGMENT

We wish to thank the anonymous reviewers for their valuable comments and suggestions. This work was supported by the National Science Foundation under Grant No. 1652954, 1619391, and 1839511.

REFERENCES

[1] D. Eastlake 3rd and M. Andrews. 2017. RFC 7873, Domain Name System (DNS) Cookies. https://tools.ietf.org/html/rfc7873.

[2] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J. Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, Seth Schoen, and Brad Warren. 2019. Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*.

[3] G. Alexander and J. R. Crandall. 2015. Off-path round trip time measurement via TCP/IP side channels. In *2015 IEEE Conference on Computer Communications (INFOCOM)*.

[4] Geoffrey Alexander, Antonio M. Espinoza, and Jedidiah R. Crandall. 2019. Detecting TCP/IP Connections via IPID Hash Collisions. In *PoPETS*.

[5] Fatemah Alharbi, Jie Chang, Yuchen Zhou, Feng Qian, Zhiyun Qian, and Nael Abu-Ghazaleh. 2019. Collaborative Client-Side DNS Cache Poisoning Attack. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1153–1161.

[6] D. Atkins and R. Austein. 2004. *RFC 3833: Threat Analysis of the Domain Name System (DNS)*. Technical Report. https://tools.ietf.org/html/rfc3833

[7] F. Baker. 1995. *Requirements for IP Version 4 Routers*. Technical Report. https://tools.ietf.org/html/rfc1812

[8] Adib Behjat. 2011. DNS Forwarders. https://www.isc.org/blogs/dns-forwarders/.

[9] Markus Brandt, Tianxiang Dai, Amit Klein, Haya Shulman, and Michael Waidner. 2018. Domain validation++ for MitM-resilient PKI. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2060–2076.

[10] R. Bush and R. Austein. 2017. *RFC 8210: The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1*. Technical Report. https://tools.ietf.org/html/rfc8210

[11] Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, Srikanth V. Krishnamurthy, and Lisa M. Marvel. 2016. Off-Path TCP Exploits: Global Rate Limit Considered Dangerous. In *Proceedings of the 25th USENIX Conference on Security Symposium* (Austin, TX, USA) *(SEC'16)*. USENIX Association, USA, 209–225.

[12] Yue Cao, Zhongjie Wang, Zhiyun Qian, Chengyu Song, Srikanth V. Krishnamurthy, and Paul Yu. 2019. Principled Unearthing of TCP Side Channel Vulnerabilities. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 211–224. https://doi.org/10.1145/3319535.3354250

[13] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. 2017. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1307–1322. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/chung

[14] CloudFlare. [n.d.]. Shield Your DNS Infrastructure From DDoS Attacks With Cloudflare's DNS Firewall. https://www.cloudflare.com/dns/dns-firewall/.

[15] European Commision. 2014. Quality of Broadband Services in the EU. http://ec.europa.eu/newsroom/dae/document.cfm?action=display&doc_id=10816.

[16] Cloudflare community. 2018. Case randomization recently disabled? https://community.cloudflare.com/t/case-randomization-recently-disabled/61376.

[17] Cloudflare community. 2018. Incorrect resolution for my domain. https://community.cloudflare.com/t/incorrect-resolution-for-my-domain/17966.

[18] Internet Systems Consortium. 2020. BIND 9. https://www.isc.org/bind/.

[19] David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. 2008. Increased DNS Forgery Resistance through 0x20-Bit Encoding: Security via Leet Queries. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS '08)*.

[20] Casey Deccio, Derek Argueta, and Jonathan Demke. 2019. A Quantitative Study of the Deployment of DNS Rate Limiting. In *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 442–447.

[21] Google Public DNS. 2019. Introduction: DNS security threats and mitigations. https://developers.google.com/speed/public-dns/docs/security.

[22] Eric Dumazet. 2014. icmp: add a global rate limitation. https://github.com/torvalds/linux/commit/4cdf507d54525842dfd9f6313fdafba039084046.

[23] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. 2015. A Search Engine Backed by Internet-Wide Scanning. In *22nd ACM Conference on Computer and Communications Security*.

[24] L. Eggert, G. Fairhurst, and G. Shepherd. 2017. *RFC 8085: UDP Usage Guidelines*. Technical Report. https://tools.ietf.org/html/rfc8085

[25] R Elz and R Bush. 1997. RFC 2181: Clarifications to the DNS specification. https://tools.ietf.org/html/rfc2181.

[26] Roya Ensafi, Jong Chun Park, Deepak Kapur, and Jedidiah R. Crandall. 2010. Idle Port Scanning and Non-Interference Analysis of Network Protocol Stacks Using Model Checking. In *Proceedings of the 19th USENIX Conference on Security* (Washington, DC) *(USENIX Security'10)*. USENIX Association, USA, 17.

[27] FCC. 2018. Eighth Measuring Broadband America Fixed Broadband Report. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-fixed-broadband-eighth-report#_Toc512944594.

[28] Suzanne Goldlust, Cathy Almond, and Mark Andrews. 2017. DNS Cookies in BIND 9. https://kb.isc.org/docs/aa-01387.

[29] Amir Herzberg and Haya Shulman. 2011. Unilateral antidotes to DNS poisoning. In *International Conference on Security and Privacy in Communication Systems*. Springer, 319–336.

[30] Amir Herzberg and Haya Shulman. 2012. Security of Patched DNS. In *ESORICS 2012*, Sara Foresti, Moti Yung, and Fabio Martinelli (Eds.).

[31] Amir Herzberg and Haya Shulman. 2013. Fragmentation considered poisonous, or: One-domain-to-rule-them-all. org. In *2013 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 224–232.

[32] Amir Herzberg and Haya Shulman. 2013. Socket Overloading for Fun and Cache-Poisoning. In *Proceedings of the 29th Annual Computer Security Applications Conference (ACSAC '13)*.

[33] R. Hinden and S. Deering. 2006. *IP Version 6 Addressing Architecture*. Technical Report. https://tools.ietf.org/html/rfc4291

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

[35] A. Hubert and R. van Mook. 2009. *RFC 5452: Measures for Making DNS More Resilient against Forged Answers*. Technical Report. https://tools.ietf.org/html/rfc5452

[36] Geoff Huston. 2019. The state of DNSSEC validation. https://blog.apnic.net/2019/03/14/the-state-of-dnssec-validation/.

[37] Ed. J. Iyengar, Ed. andM. Thomson. 2020. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Technical Report. https://tools.ietf.org/html/draft-ietf-quic-transport-27

[38] A. J. Kalafut, C. A. Shue, and M. Gupta. 2011. Touring DNS Open Houses for Trends and Configurations. *IEEE/ACM Transactions on Networking* 19, 6 (2011), 1666–1675.

[39] Dan Kaminsky. 2008. Black ops 2008: It's the end of the cache as we know it. *Black Hat USA* (2008).

[40] Simon Kelley. 2020. Dnsmasq - network services for small networks. http://www.thekelleys.org.uk/dnsmasq/doc.html.

[41] Amit Klein, Haya Shulman, and Michael Waidner. 2017. Internet-wide study of DNS cache injections. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 1–9.

[42] Jeffrey Knockel and Jedidiah R. Crandall. 2014. Counting Packets Sent Between Arbitrary Internet Hosts. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*. USENIX Association, San Diego, CA. https://www.usenix.org/conference/foci14/workshop-program/presentation/knockel

[43] NLnet Labs. 2020. Unbound DNS Resolver. https://nlnetlabs.nl/projects/unbound/about/.

[44] Cricket Liu. 2015. A new kind of DDoS threat: The "Nonsense Name" attack. https://www.networkworld.com/article/2875970/a-new-kind-of-ddos-threat-the-nonsense-name-attack.html.

[45] lkm. 2007. Blind TCP/IP Hijacking is Still Alive. http://phrack.org/issues/64/13.html.

[46] Chaoyi Lu, Baojun Liu, Zhou Li, Shuang Hao, Haixin Duan, Mingming Zhang, Chunying Leng, Ying Liu, Zaifeng Zhang, and Jianping Wu. 2019. An End-to-End, Large-Scale Measurement of DNS-over-Encryption: How Far Have We Come?. In *Proceedings of the Internet Measurement Conference* (Amsterdam, Netherlands) *(IMC '19)*. Association for Computing Machinery, New York, NY, USA, 22–35. https://doi.org/10.1145/3355369.3355580

[47] Matthew Luckie, Robert Beverly, Ryan Koga, Ken Keys, Joshua A. Kroll, and k claffy. 2019. Network Hygiene, Incentives, and Regulation: Deployment of Source Address Validation in the Internet. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 465–480. https://doi.org/10.1145/3319535.3354232

[48] Ed. M. Bishop. 2020. *Hypertext Transfer Protocol Version 3 (HTTP/3)*. Technical Report. https://datatracker.ietf.org/doc/draft-ietf-quic-http/

[49] Moritz Müller, Giovane C. M. Moura, Ricardo de O. Schmidt, and John Heidemann. 2017. Recursives in the Wild: Engineering Authoritative DNS Servers. In *Proceedings of the 2017 Internet Measurement Conference* (London, United Kingdom) *(IMC '17)*. Association for Computing Machinery, New York, NY, USA, 489–495. https://doi.org/10.1145/3131365.3131366

[50] Zhiyun Qian and Z Morley Mao. 2012. Off-path TCP sequence number inference attack-how firewall middleboxes reduce security. In *2012 IEEE Symposium on Security and Privacy*. IEEE, 347–361.

[51] Alan Quach, Zhongjie Wang, and Zhiyun Qian. 2017. Investigation of the 2016 Linux TCP Stack Vulnerability at Scale. *SIGMETRICS Perform. Eval. Rev.* (2017).

[52] Vicky Ris, Suzanne Goldlust, and Alan Clegg. 2020. BIND Best Practices - Authoritative. https://kb.isc.org/docs/bind-best-practices-authoritative.

[53] Paul Schmitt, Anne Edmundson, Allison Mankin, and Nick Feamster. 2019. Oblivious DNS: Practical Privacy for DNS Queries. In *PoPETS*.

[54] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. 2013. On measuring the client-side DNS infrastructure. In *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 77–90.

[55] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. 2014. DNS Record Injectino Vulnerabilities in Home Routers. http://www.icir.org/mallman/talks/schomp-dns-security-nanog61.pdf. Nanog 61.
[56] Sergio De Simone. [n.d.]. The Status of HTTP/3. https://www.infoq.com/news/2020/01/http-3-status//.
[57] US-Cert. 2019. Alert (TA13-088A) - DNS Amplification Attacks. https://www.us-cert.gov/ncas/alerts/TA13-088A.
[58] Paul Vixie. 2019. On the Time Value of Security Features in DNS. http://www.circleid.com/posts/20130913_on_the_time_value_of_security_features_in_dns/.
[59] Paul Vixie and Vernon Schryver. 2012. DNS Response Rate Limiting (DNS RRL). https://ftp.isc.org/isc/pubs/tn/isc-tn-2012-1.txt.
[60] Xiaofeng Zheng, Chaoyi Lu, Jian Peng, Qiushi Yang, Dongjie Zhou, Baojun Liu, Keyu Man, Shuang Hao, Haixin Duan, and Zhiyun Qian. 2020. Poison Over Troubled Forwarders: A Cache Poisoning Attack Targeting DNS Forwarding Devices. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 577–593. https://www.usenix.org/conference/usenixsecurity20/presentation/zheng