Poison Over Troubled Forwarders: A Cache Poisoning Attack Targeting DNS Forwarding Devices

Xiaofeng Zheng[*,†], Chaoyi Lu[*], Jian Peng[*], Qiushi Yang[†],
Dongjie Zhou[§], Baojun Liu[*], Keyu Man[‡], Shuang Hao[¶], Haixin Duan[*,†*] and Zhiyun Qian[‡]

[*] Tsinghua University, [†] Qi An Xin Technology Research Institute,
[§] State Key Laboratory of Mathematical Engineering and Advanced Computing,
[‡] University of California, Riverside, [¶] University of Texas at Dallas

Abstract

In today's DNS infrastructure, DNS forwarders are devices standing in between DNS clients and recursive resolvers. The devices often serve as ingress servers for DNS clients, and instead of resolving queries, they pass the DNS requests to other servers. Because of the advantages and several use cases, DNS forwarders are widely deployed and queried by Internet users. However, studies have shown that DNS forwarders can be more vulnerable devices in the DNS infrastructure.

In this paper, we present a cache poisoning attack targeting DNS forwarders. Through this attack, attackers can inject rogue records of arbitrary victim domain names using a controlled domain, and circumvent widely-deployed cache poisoning defences. By performing tests on popular home router models and DNS software, we find several vulnerable implementations, including those of large vendors (e.g., D-Link, Linksys, dnsmasq and MS DNS). Further, through a nationwide measurement, we estimate the population of Chinese mobile clients which are using vulnerable DNS forwarders. We have been reporting the issue to the affected vendors, and so far have received positive feedback from three of them. Our work further demonstrates that DNS forwarders can be a soft spot in the DNS infrastructure, and calls for attention as well as implementation guidelines from the community.

1 Introduction

The Domain Name System (DNS) serves as one of the fundamental infrastructures of the Internet. It provides translation of human-readable domain names to numerical addresses, and is the entry of almost every action on the Internet. According to its initial standard, when a domain name needs to be resolved, a DNS client sends a query to a recursive resolver. The recursive resolver in turn fetches answers from authoritative servers.

[*]Haixin Duan is the corresponding author.

However, as the DNS ecosystem has evolved dramatically, the system now consists of multiple layers of servers [62]. Specifically, *DNS forwarders* refer to devices standing in between DNS clients and recursive resolvers. Upon receiving DNS queries, the devices do not resolve the domain name by themselves, but pass the requests to other servers (e.g., an upstream recursive resolver). To name a few use cases, DNS forwarders can serve as convenient default resolvers, load balancers for upstream servers, and gateways of access control. Meanwhile, for clients in a local network, using DNS forwarders can mitigate security risks, as the devices are not directly exposed to Internet attackers [49].

Because of the advantages, DNS forwarders are fairly prevalent devices in the DNS infrastructure. It has been reported that over 95% open DNS resolvers are actually forwarders [62], and that a vast number of them run on residential network devices [57, 64]. Forwarding is also widely implemented in DNS software (e.g., BIND [25], Unbound [27], Knot Resolver [13] and PowerDNS [18]) and home routers (e.g., TP-Link [21], D-Link [5] and Linksys [4]).

Given its prevalence, though, there have been only few studies on the understanding and security status of DNS forwarders. In addition, works have shown that DNS forwarders can actually be a soft spot in the DNS infrastructure. For instance, a considerable number of such device fail to perform checks on ephemeral port numbers and DNS transaction IDs, and are vulnerable to cache poisoning attacks or DoS [49, 63, 64]. The discoveries call for deployments of cache poisoning defences, such as randomizing port numbers [52], 0x20 encoding [36] and DNSSEC [30].

In this paper, we further demonstrate that DNS forwarders can be vulnerable devices in the ecosystem, by proposing a cache poisoning attack. Using our attack methods, an adversary can use a controlled domain name and authoritative server to *inject records of arbitrary domain names*. In addition, the attack *bypasses widely-deployed defences* including randomized ephemeral port numbers and 0x20 encoding. We also perform tests on current implementations of DNS forwarders, and find several home router models and

DNS software vulnerable to this attack. The vulnerable implementations include those from popular vendors, such as D-Link [5], Linksys [4], dnsmasq [7] and MS DNS [8]. We have been reporting the issue to the affected vendors, and so far have received positive responses from three of them. Furthermore, we perform a nationwide measurement of the affected client population, and estimate the scale of Chinese mobile devices which are using the vulnerable devices. In the end, we find that the industry have diverse understanding on the role of DNS forwarders, and there is still a lack of forwarder implementation guidelines in the DNS specifications.

Contributions. In this paper, we make the following contributions.

New attack. We propose a type of cache poisoning attack targeting DNS forwarders. Through this attack, an adversary can use a controlled domain name to inject DNS records of arbitrary victim domain names, and circumvent current cache poisoning defences.

New findings. We find several home router models and DNS software vulnerable to the attack, including those by large developers. We have been reporting the vulnerability to affected vendors.

Put together, this paper demonstrates an attack targeting DNS forwarders, and sheds light on their security problems. DNS forwarders are prevalent devices in the ecosystem, yet we show that they can be more vulnerable to cache poisoning attacks. Therefore, we believe more attention should be paid from the community to DNS forwarder specifications and security.

Paper organization. The remainder of this paper is organized as follows. Section 2 gives an overview on prior DNS cache poisoning attacks. Section 3 describes the role of forwarders in the DNS ecosystem. Section 4 illustrates our attack model. Section 5 elaborates our tests on vulnerable DNS forwarder software. Section 6 performs a nationwide measurement study on the population of affected clients. Section 7 discusses the implementation and specification of DNS forwarders. Section 8 extends the attack model and proposes mitigation. Section 9 summarizes related work and Section 10 concludes the paper.

2 Prior DNS Cache Poisoning Attacks Targeting Recursive Resolvers

DNS cache poisoning attacks have been known for long, and they pose serious threats to Internet users [65, 67, 69]. In this section we first give an overview on two major types of known attack methods, and discuss their limitations.

2.1 Forging Attacks

The goal of forging attacks is to craft a rogue DNS response and trick a resolver into accepting it. In detail, a DNS response is accepted when the following fields matches a DNS query: question section, DNS transaction ID, source/destination addresses and port numbers. If an attacker forges a DNS response with the correct metadata before the authenticated response arrives, the rogue response can be accepted by the resolver and the attack succeeds. The most influential case of forging attacks is the Kaminsky Attack [53] in 2008, which affects nearly all software designed to work with DNS.

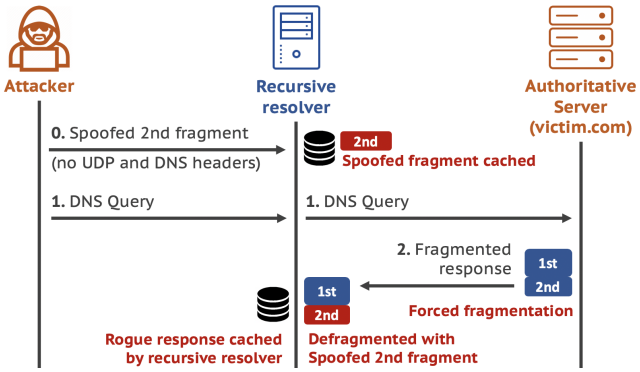

Figure 1: Defragmentation cache injection attacks targeting recursive resolvers.

Limitations. The key to mitigating forging attacks is to increase the randomness of DNS query packets. As required by RFC 5452 [52], resolver implementations now must use randomized ephemeral port numbers and DNS transaction IDs. Meanwhile, resolvers also adopt 0x20 encoding [36] to mix the upper and lower spelling cases of the name in the question section. As a result, the widely-deployed defences have significantly increased resolvers' resistance to forging attacks.

2.2 Defragmentation Attacks

Recent studies [33, 47–49, 66] have uncovered a new type of DNS cache poisoning attack based on IP defragmentation. The attack exploits the fact that the 2nd fragment of a fragmented DNS response packet does not contain DNS or UDP headers or question section, so it can *bypass randomization-based defences against forging attacks*. As shown in Figure 1, an attacker first crafts a spoofed 2nd fragment with rogue DNS records, and issues a DNS query of the victim domain name. The response from an authoritative server is forced to be fragmented by the attacker (through a separate process ahead of time). At the recursive resolver, the legitimate 1st fragment is reassembled with the spoofed 2nd fragment, which produces a rogue DNS response. As a result, the rogue records are cached by the recursive resolver and the attack succeeds. We provide more background on IP fragmentation in Appendix A.

The core challenge of defragmentation attacks is to force a fragmentation of the DNS response, and there are two ap-

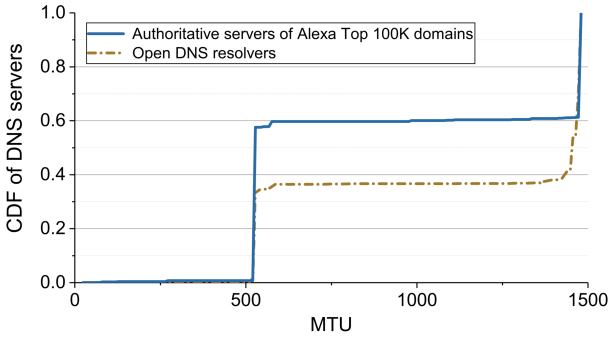

Figure 2: CDF of lowered MTU of a) authoritative servers of Alexa Top 100K domains, and b) 2M open DNS resolvers from an Internet-wide scan.

proaches proposed so far. The first approach is to lower the Path Maximum Transmission Unit (PMTU) between the recursive resolver and authoritative servers [33]. We term this type of attack as *PMTU-based defragmentation attack*. By contrast, the second approach is to send DNSSEC queries to solicit enlarged DNS responses with DNSSEC records, so that they reach limits of MTUs (e.g., 1,500 bytes for Ethernet) and will be fragmented [48]. We term this type of attack as *DNSSEC-based defragmentation attack*.

Limitations. Previous defragmentations have high requirements on the PMTU behavior of authoritative name servers, as well as the victim domains. Specifically, PMTU-based defragmentation requires an attacker to send specially-crafted *ICMP fragmentation needed error* messages to an authoritative server claiming a small PMTU and trick it to lower the PMTU for a specific resolver. However, we find this is impractical in most cases. As shown in Figure 2, for authoritative servers of Alexa Top 100K domains, only 0.7% are willing to reduce their MTU to less than 528 bytes. Since DNS responses are typically smaller than 512 bytes, it is not likely that they will be forcibly fragmented. As for DNSSEC-based attacks, they require non-validating recursive resolvers and can be mitigated through proper DNSSEC deployment and validation. Moreover, the attack only works for DNSSEC-signed victim domains. Currently, DNSSEC deployment is still low among domain names (e.g., less than 1.85% for popular domains in 2017 [34]), thus the target of DNSSEC-based defragmentation attacks is limited.

3 DNS Forwarder

Traditionally, a DNS resolution process involves a DNS client (or stub resolver), a recursive resolver and authoritative servers. When a domain needs to be resolved, the DNS client sends a query to a recursive resolver, which in turn fetches answers from authoritative servers. For maximum protocol capability, it is recommended that DNS clients use a full-service resolver directly [31]. However, in reality, the DNS infrastructure has become far more complex than

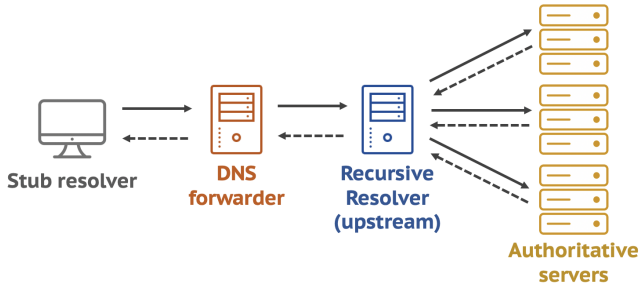

Figure 3: DNS infrastructure with forwarders

this simple model, often involving multiple layers of servers. One of the new roles introduced in the infrastructure [29,70], as shown in Figure 3, is *DNS forwarders*[1]. They sit in between stub and recursive resolvers, and often serve as ingress servers for DNS clients (e.g., home wireless routers). When a DNS forwarder receives a query, instead of performing the resolution recursively, it simply forwards the query to an upstream recursive resolver. To name a few use cases, it can serve as a default local resolver (with *caching capability*) for clients (e.g., clients using DHCP to obtain network configurations in LAN), perform load balancing among upstream recursive resolvers, and can be used to enforce access control.

DNS forwarder vs. recursive resolver. In the latest RFC on DNS terminology (i.e., RFC 8499 [50]), recursive resolvers are resolvers which "act in recursive mode". When it receives a DNS query, a recursive resolver accesses other servers, and should respond with the *final answer* to the original query. As such, recursive resolvers should handle referrals to other servers and aliases to other names (i.e., CNAMEs), and aggregate the resource records into one final answer. Recursive resolvers should also perform integrity checks such as the bailiwick check [39] and DNSSEC validation [30]. In contrast, a DNS forwarder does not recursively resolve queries, and instead relies on the integrity of its upstream server. As a result, DNS forwarders do not handle referrals, and are typically not in the position to verify the responses. Otherwise, forwarders will be repeating the work of resolvers, e.g., checking each referral, defeating the purpose of having another layer of indirection. As we will articulate later, this is a key weakness of DNS forwarders which enables our cache poisoning attack.

4 Defragmentation Attacks Targeting DNS Forwarders

As we have seen, previous defragmentation attacks have limitations regarding the ability to trigger fragmentation. In this section, we propose a novel modified defragmentation attack that works perfectly against DNS forwarders due to its unique role in the DNS infrastructure.

[1] Also defined as "FDNS" in literature.

4.1 Attack Overview

Threat Model. Studies have discovered a large number of DNS forwarders running on *residential network devices*, such as home routers [64]. As such, in our threat model we assume the attacker is located in the same LAN as the DNS forwarder, and can issue DNS queries. This can occur in an open Wi-Fi network (e.g., at coffee shops and airports) without strong security protection or password. This can also happen in some enterprise networks where a guest, insider, or compromised machine acts as an attacker. In some cases, forwarders on home routers can also be open to public due to misconfigurations [57].

Our attack starts out by asking the question: can we force fragmentation reliably and deterministically? It turns out that we can, if the query is sent towards an authoritative name server under an attacker's control – as the server can intentionally send an oversized response. At a first glance, this is meaningless because that would mean that the domain hosted on the attacker's authoritative name server also belongs to the attacker already (e.g., attacker.com). It is useless to poison the attacker's own domain. However, our key insight is that forwarders have total reliance on upstream resolvers to perform response validation (see Section 3). Due to the unique role of DNS forwarder, it is actually possible to inject spoofed fragments containing records of other domains (e.g., victim.com) and trick the forwarder to cache such records.

Workflow. Figure 4 illustrates this idea. After probing the current resolver IPID (step 0), an attacker feeds the victim DNS forwarder with a spoofed 2nd fragment containing rogue DNS records (step 1) and launches a DNS request (step 2). The aggregated final response from the attacker's authoritative servers (oversized, larger than Ethernet MTU) is fragmented when leaving the recursive resolver (step 3b), and defragmented at the DNS forwarder (step 3c). In particular, at defragmentation *the legitimate 1st fragment is reassembled with the spoofed 2nd fragment*, producing a rogue response. Consequently, the rogue DNS records are then written into the forwarder's cache (as forwarders are not in the position to validate upstream responses), and handed over to downstream devices. As is the case with prior defragmentation attacks, an attacker no longer needs to guess DNS and UDP metadata (e.g., DNS transaction ID and ephemeral port numbers), which does not exist in the 2nd fragment. Using oversized responses, our new attack can overcome the key limitation in prior defragmentation attacks that forcing fragmentation is difficult.

4.2 Forcing DNS Response Fragmentation

Forcing DNS response fragmentation (see Figure 4, step 3b) is one of the key steps in defragmentation attacks. As discussed in Section 2, previous studies use two different methods to force fragmentation: reducing PMTU and enlarging DNS responses with DNSSEC. Let us reason about whether these two methods can be applied to DNS forwarders successfully.

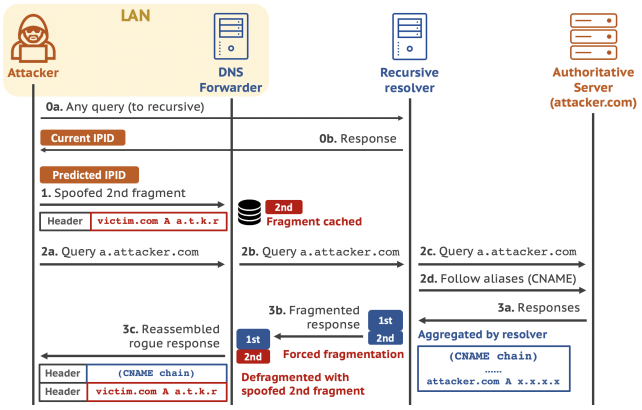

Figure 4: Defragmentation cache poisoning attack targeting DNS forwarders in the same LAN (e.g., DNS forwarders of residential network devices).

Reducing PMTU: ineffective for DNS forwarder attacks. We first consider borrowing from PMTU-based defragmentation attacks, where an attacker lowers PMTU to force response fragmentation. According to our attack model, the DNS response needs to be fragmented between the recursive resolver and the DNS forwarder (see Figure 4, step 3b), thus an attacker should attempt to lower the MTU of the upstream recursive resolver. Using the same approach as in [33] (i.e., sending ICMP fragmentation needed error messages), we perform a measurement on 2M open DNS resolvers in the wild. In the end, as also shown in Figure 2, the results turn out to be unsatisfying: only 0.3% resolvers can reduce their packet size to below 512 bytes, and less than 37% reduce to below 600 bytes.

DNSSEC-based fragmentation: even less effective against DNS forwarders. We already know that leveraging DNSSEC is very limited as it only works for a limited range of domains and servers. In addition, DNS forwarders in this case also need to support DNSSEC. Otherwise, the upstream recursive resolver will not even send DNSSEC responses.

Solution: oversized DNS response using CNAME. As mentioned earlier, an attacker-controlled authoritative name server can intentionally create an oversized DNS response larger than the Ethernet MTU (i.e., larger than 1,500 bytes), such that it will *always be fragmented at the recursive resolver*.

As shown in Figure 5, the method to create such large responses is through *a chain of CNAME records*, followed by one final A record. When handling this query, recursive resolvers will query the aliases in the chain (see Figure 4, step 2d) and aggregate the CNAME records into the final response. The attacker fills the chain with enough

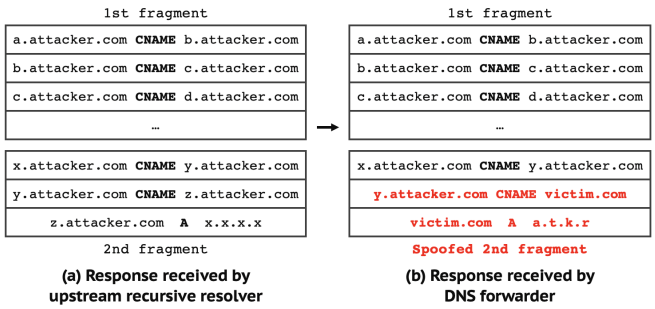

Figure 5: Oversized DNS response using CNAME

dummy CNAME records to make the final response larger than the Ethernet MTU, such that it will always be fragmented at the recursive resolver. In the spoofed 2nd fragment (sent to the DNS forwarder), the attacker "tampers" with the last CNAME record by pointing it to a victim domain (`victim.com`), and the last A record by pointing it to a rogue address (`a.t.k.r`). After the response is defragmented at the forwarder, the rogue A record will be cached.

The key here is that the recursive resolver sees only a legitimate oversized response from the authoritative name server (Figure 5(a)), without violating bailiwick rules. Therefore, it will attempt to relay this response as a whole back to the forwarder, with fragmentation. However, what the forwarder sees on its end is actually a tampered response (Figure 5(b)), due to the spoofed 2nd fragmented injected ahead of time. Had the resolver seen such a response (where the attacker.com eventually points to victim.com), it will reject the response during recursive queries of the aliases. This is exactly the reason our attack targets DNS forwarders as they are not in the position to perform validations.

The use of oversized DNS responses requires that all DNS servers in our attack model support Extension Mechanisms for DNS (EDNS(0)) [37]. As an important DNS feature, EDNS(0) provides support to transfer DNS packets larger than 512 bytes over UDP, and is being increasingly supported by software vendors and DNS operators. Currently it has been implemented by mainstream DNS software (e.g., BIND [25], Knot DNS [13], Unbound [27] and PowerDNS [18]) and supported by most recursive resolvers [61]. To indicate EDNS(0) support, servers use one OPT record in the additional section of a DNS packet to carry EDNS options.

4.3 Crafting Spoofed Fragments

For fragmented DNS responses, only the 1st fragment contains DNS and UDP headers (see Appendix A for more background of IP fragmentation). As a result, to craft a spoofed 2nd fragment, an attacker does not need to predict ephemeral port numbers and DNS transaction IDs. However, for successful defragmentation, an attacker needs to craft the following IP header fields of the spoofed 2nd fragment.

IPID prediction. IP identification (IPID) is a 16-bit field in the IP header, which is used to determine which datagram a fragment belongs to. For successful defragmentation, the IPIDs of the spoofed 2nd fragment and the legitimate 1st fragment (from the upstream resolver) should agree. As such, an attacker should be able to predict the IPID assignment of the upstream resolver (see Figure 4, step 1). In general, this is a well studied topic in the literature and a number of techniques have been proposed [28, 49]. We give a summary below on how we can take advantage of predictable IPID assignment and then conduct a measurement to show how most DNS resolvers in the wild can be exploited.

IPID assignment algorithms. There are three major IPID assignment algorithms: global IPID counter, hash-based IPID counter [44], and random IPID assignment. Global IPID counter increases by one for every sent packet, which is highly predictable [55]. Hash-based IPID counter algorithms first use a hash function to map an outgoing packet to one in an array of IPID counters, and then increase the selected counter by a random amount, chosen from a uniform distribution between 1 and the number of system ticks (typically milliseconds) since the last packet transmission that used the same counter [28]. If the two probes are sent close enough in time, then the IPID increments from the responses are very predictable. In fact, since the defragmentation cache can typically buffer 64 fragments [56], an attacker can make a prediction on a range of IPID values instead of a single one. The hash function determines which IPID counter is used, based on the source and destination IP address of the sent packet (the same source and destination IP pair will therefore always result in the same IPID counter getting selected). In our attack, an attacker can first probe for the current IPID value of the upstream resolver, and use one or more predicted IPIDs to place the spoofed 2nd fragment. The probing response (see Figure 4, step 0b) and the 1st fragment (see Figure 4, step 3b) are both sent to the "NAT-ed" public address of the LAN, so they are guaranteed to use the same IPID counter on the upstream resolver. As the attacker initiates the entire sequence of packets and controls the timing of these packets, it can make sure that the gap between the initial IPID and the later one (in the resolver's response) is small enough and hence predictable (because they are generated close in time, e.g., a few milliseconds apart).

Operating systems. As reported by previous studies [28, 33, 55, 71], early versions of Windows (prior to Windows 8) use global IPID counters, and recent Windows and Linux versions use hash-based IPID counters. By setting up virtual machines and probe their IPID assignments, we confirm that the latest versions of Windows 10 (Professional, Version 1909 (18363.657)) and Ubuntu (5.3.0-29-generic) both use hash-based IPID counters. Since most servers (including recursive resolvers) on the Internet are equipped with Windows or Linux, we believe this technique covers most of the ground.

Table 1: IPID assignment of egress resolvers

IPID Assignment	Name	Address	# Tested Resolvers
Hash-based IPID counters (Exploitable)	Cloudflare	1.1.1.1	64
	Quad9	9.9.9.9	8
	Comodo	8.26.56.26	2
	OpenDNS	208.67.222.222	14
	Norton	199.85.126.10	2
Random	Google	8.8.8.8	15
	Verisign	64.6.64.6	24

Open DNS resolvers. We leverage the open DNS resolver scanning result of Censys [38] on Jan 8, 2020. For each resolver in the list, we send three DNS queries in a row of our own domain name, and check whether the IPIDs in the corresponding DNS response packets are increased by a fixed value. As a result, 4,988,186 resolvers respond to all three queries, and 4,235,342 (84.9%) use incremental IPID counters which can be exploited in the attack.

Popular public DNS services. Public DNS services often use anycast for load balancing. For example, DNS queries to Google's 8.8.8.8 can exit from hundreds of "egress" resolvers (e.g., 74.125.19.*). From a client's perspective, because DNS responses come from different egress resolvers, the public DNS services appear to use random IPID assignment. However, in our defragmentation attack, because the authoritative server is under an adversary's control, an attacker can *break the load balancing* by responding to only one selected egress resolver address. If the selected egress resolver uses incremental IPID counters, the attack is still possible.

To begin our measurement, we build a custom authoritative server for our own domain name (termed as `echo.dnsaddr`). On receiving a DNS query (e.g., `[nonce].echo.dnsaddr`), the authoritative server records the source IP address of the DNS query (i.e., egress resolver address), and echoes the resolver address through an A record in the DNS response[2]. Using this technique, we can separate DNS responses sent from different egress resolvers, and observe their IPID assignment respectively.

We choose 7 popular public DNS services for our tests: Cloudflare [2], Google [10], Quad9 [19], OpenDNS [1], Verisign [22], Comodo [3] and Norton [16]. Our vantage points send DNS queries of `[nonce].our.domain` (to avoid caching) to each public DNS service and capture the DNS response packets.

As shown in Table 1, we find that five public DNS services use hash-based IPID counters on their egress resolvers, which can be exploited in the attack. Google and Verisign use unpredictable IPIDs, which are not exploitable. Due to space limit, we put more detailed results in Appendix B. To confirm that the public DNS services are exploitable, in Section 5 we also launch real attacks using a public DNS service as upstream resolver.

Other header fields. For successful defragmentation of the 1st fragment and the spoofed 2nd fragment, the attacker should also craft the following header fields in the spoofed 2nd fragment.

Fragment offset. The fragment offset in the spoofed 2nd fragment should indicate its correct position in the original datagram. Since contents of the oversized DNS response are fully controlled by the attacker (see Figure 5), the offset of the 2nd fragment can be calculated.

IP source address. The spoofed 2nd fragment should come from a spoofed address of the upstream recursive resolver. To learn the address of the upstream recursive resolver, an attacker can leverage the `echo.dnsaddr` method in our public DNS service measurement (i.e., send a query of `echo.dnsaddr` to the DNS forwarder, and check the resolver address encoded in the DNS response). An attacker may also setup an authoritative server of a controlled domain, query the DNS forwarder for the domain name, and observe the upstream recursive resolver address at the authoritative server. In networks of residential devices (i.e., LAN), IP spoofing is generally allowed.

Fitting the UDP checksum. The UDP checksum (in the legitimate 1st fragment) is calculated from the IP header, UDP header and the entire UDP payload. Tampering with records in the spoofed 2nd fragment produces a checksum mismatch, so an attacker should also adjust other bytes in the spoofed 2nd fragment to fit the original checksum. In fact this task is easy, as in our model the contents of the DNS response are fully controlled by the attacker, thus the original checksum of the DNS response is already sknown. As a result, the attacker can adjust other bytes in the spoofed 2nd fragment with simple calculation (as in [33]) to fit the UDP checksum.

4.4 Conditions of Successful Attacks

Driven from our threat model, a DNS forwarder should satisfy the following conditions to be successfully attacked.

EDNS(0) support. EDNS(0) allows large DNS packets over UDP. As an important DNS feature, we expect that it is being increasingly supported by software vendors and DNS operators.

No truncation of DNS response. Despite supporting EDNS(0), several of our tested forwarder implementations actively truncate large DNS responses, even when they do not reach the Ethernet MTU (e.g., truncate all responses at 512 or 1,280 bytes, see Table 2 in Section 5). In such case, the truncated DNS responses are not fragmented, thus the defragmentation attack will fail.

[2]Our authoritative server is similar to Akamai's `whoami.akamai.net` tool [12]. The difference is that our server replies to `*.echo.dnsaddr`, while Akamai's tool does not support queries of arbitrary subdomain.

No verification of DNS response. The aggregated oversized DNS response consists of a CNAME chain, and the attacker tampers with the last two records. To detect the rogue records, a possible solution is for the DNS forwarder to "re-query" the domains and aliases (i.e., `*.attacker.com` and `victim.com`) in the aggregated response (i.e., perform recursive queries). Alternatively, if the victim domain is DNSSEC-signed, it can also perform full DNSSEC validation. However, this defeats the purpose of a forwarder as it is significantly increasing the amount of workload.

DNS caching by record. From the smallest unit of each DNS cache entry, we find DNS forwarders cache the answers either by response as a whole (i.e., the entire response forms one cache entry) or by record (i.e., each resource record forms individual cache entries). For example, when the defragmented DNS response in Figure 5(b) is cached by response, it only forms one cache entry for `a.attacker.com`. As a result, querying `victim.com` does not hit the cache, so the spoofed record will not be returned. In contrast, when it is cached by record, querying any name in the response (e.g., `y.attacker.com` and `victim.com`) will hit the cache. Because the victim domain is located only in the last record of the response, the attack requires that the DNS forwarder cache by record. Caching by record has a performance advantage as more records will be cached in a single response.

5 Vulnerable DNS Forwarder Software

In this section, we first measure the DNS forwarding behaviors of home routers and DNS software, to check whether they fit our defragmentation attack conditions. We then perform actual defragmentation attacks to confirm their vulnerabilities.

5.1 Home Routers

A number of DNS forwarders have been recognized to run on residential network devices. In a typical setting, the devices receive DNS requests from clients, and forward them to upstream recursive resolvers. As a very representative case, we start from testing the prevalent home routers, which commonly support DNS forwarding.

We perform our tests on real home router models that we purchase from their official online stores. According to a report on the home router market [60], we select models from leading vendors including TP-Link [21], D-Link [5], NETGEAR [15], Huawei [11] and Linksys [4], as well as other prominent players like Tenda [20], ASUS [23], Gee [9] and Xiaomi [14]. In total, we perform tests on 16 router models from different vendors. For each router, we test if it fits all attack conditions proposed in Section 4.4.

Test results. Table 2 presents the DNS forwarding behaviors of home routers. Among 16 router models, we find that 8 (50%, in the first section of Table 2) satisfy all our attack conditions, which are vulnerable to the defragmentation cache poisoning attack. 5 models (in the second section of Table 2) are immune to the attack because they either do not support DNS caching or do not cache by record. The remaining 3 models (in the third section of Table 2) are not affected by the attack, because they have problems handling oversized DNS responses. They either do not support EDNS(0) at all, or actively truncate the response to a smaller size. As expected, we do not find any router model that "re-queries" the names to verify the DNS response.

Table 2: DNS forwarding behaviors of home routers. The first eight models are confirmed vulnerable by real attacks.

Brand	Model	EDNS(0)	No Truncation	Cache by Record	Vulnerable
D-Link	DIR 878	✓	✓	✓	✓
ASUS	RT-AC66U B1	✓	✓	✓	✓
Linksys	WRT32X	✓	✓	✓	✓
Motorola	M2	✓	✓	✓	✓
Xiaomi	3G	✓	✓	✓	✓
GEE	Gee 4 Turbo	✓	✓	✓	✓
Wavlink	A42	✓	✓	✓	✓
Volans	VE984GW+	✓	✓	✓	✓
Huawei	Honor router 2	✓	✓	✗	✗
Tenda	AC1206	✓	✓	✗	✗
FAST	FER1200G	✓	✓	–[1]	✗
TP-Link	TL-WDR5660	✓	✓	–	✗
Mercury	D128	✓	✓	–	✗
NetGear	R6800	✗	✗[2]	–	✗
H3C	MSR830-WiNet	✗	✗[2]	✓	✗
Cisco	RV320	✓	✗[3]	✓	✗

[1] DNS caching not supported.
[2] Truncate at 512 bytes.
[3] Truncate at 1280 bytes.

5.2 DNS Software

DNS forwarding is also implemented by mainstream DNS software. For instance, it can be enabled by the `forward-zone` keyword of Unbound, or the `server` keyword of dnsmasq. We test the DNS forwarding behaviors of seven kinds of mainstream DNS software: BIND [25], Unbound [27], Knot Resolver [13], PowerDNS [18], DNRD [6], dnsmasq [7] and MS DNS [8].

Test results. Table 3 presents the DNS forwarding behaviors of DNS software. We find that dnsmasq and MS DNS satisfy all attack conditions, which are vulnerable to the defragmentation cache poisoning attack. Particularly, dnsmasq is used by embedded systems like OpenWRT [17], so the attack can affect more router models than our tested ones. DNRD is not vulnerable because it caches DNS responses as a whole. Surprisingly, BIND, Unbound, Knot Resolver and PowerDNS are immune to the attack, because they re-query the CNAME chain and verify the oversized response, even when configured as DNS forwarders.

Table 3: DNS forwarding behaviors of DNS software. The first two are confirmed vulnerable by real attacks.

Software	Version	EDNS(0) & No truncation	Cache by Record	No Verification	Vulnerable
dnsmasq	2.7.9	✓	✓	✓	✓
MS DNS	2019	✓	✓	✓	✓
BIND	9.9.4	✓	✓	✗	✗
Unbound	1.7.2	✓	✓	✗	✗
Knot Res	3.2.0	✓	✓	✗	✗
PowerDNS	4.1.8	✓	✓	✗	✗
DNRD	2.20.3	✓	✗	✓	✗

5.3 Confirmation of Attacks

To confirm that the selected software (listed in Tables 2 and 3) is vulnerable to the defragmentation cache poisoning attack, we launch real attacks in controlled environments.

Clean controlled experiment. In a simple case, we build our testing environment according to the attack model (see Figure 4). The attacker machine and the DNS forwarder locate in the same LAN. We configure the DNS forwarder to use a recursive resolver (for which we use Unbound [27]) as upstream, which is not open to public. Also, we build the attacker's authoritative server (which is located outside of the LAN) and create an oversized DNS response according to Figure 5. Finally, we confirm that the attack succeeds, if the rogue record of `victim.com` (in the spoofed 2nd fragment) is cached by the DNS forwarder. As a result, all 8 router models and 2 DNS software are confirmed vulnerable with this experiment.

Complex network experiment. To confirm that the attack is feasible in the real world, we also test the attack in a more complex environment.

Home router. We select a home router which runs on the latest version of OpenWRT operating system (19.07.1 r10911-c155900f66). As mentioned, OpenWRT by default uses dnsmasq as its DNS forwarder, thus home routers built over this system are vulnerable to the attack.

Clients and attacker. To add more background traffic, we add 13 other clients (e.g., mobile phones, tablets and laptops) into the LAN of the home router. On the clients we start tasks such as file downloading, video streaming and web browsing. On average, the home router receives 7.95Mbps/753.3Kbps of inbound/outbound traffic in a 3-minute window. The attacker retries each failed or timed-out DNS query every five seconds.

Upstream recursive resolver. We configure the DNS forwarder to use Norton ConnectSafe (at 199.85.126.10). According to our measurement results in Table 1, its egress resolvers use incremental IPID counters which are exploitable.

Authoritative server. We also create the oversized DNS response according to Figure 5. To break load balancing of the resolver, we configure our authoritative server such that it only responds to queries from one selected egress resolver address of Norton ConnectSafe (e.g., 156.154.38.*).

In the end, a successful attack in this environment takes 58 seconds to complete. In more detail, the attacker first tries to probe the current IPID value of the selected egress resolver (see Figure 4, step 0), which takes 22 seconds and 7 retries. The attacker then uses sequentially incremented IPID values in the spoofed 2nd fragments, and start querying the attacker domain name (see Figure 4, step 2). On the 10th retried DNS query, the legitimate 1st fragment and the spoofed 2nd fragment are reassembled, and the attack succeeds. Because of resolver load balancing (i.e., not every DNS query goes to the selected egress resolver) and possible packet loss, the attack takes longer and requires more retries of DNS queries.

5.4 Responsible Disclosure

We have been reporting the issue to the affected vendors, by submitting vulnerability reports and contacting via emails. So far, we have received responses from 3 home router manufacturers (ASUS, D-Link and Linksys). ASUS and D-Link have released firmware patches to fix the DNS cache poisoning vulnerability, where DNS responses are now cached as a whole (see Section 8 for detailed mitigations). Linksys has accepted our report via the Bugcrowd [26] platform.

6 Client Population: A Nationwide Measurement Study

In Section 5, we find several home routers vulnerable to defragmentation attacks. Further, we seek answer to the question "how many real-world clients are using the susceptible devices?" In this section, we elaborate our methodology on measuring the client population of such devices, and report our findings.

6.1 Methodology

Unlike our tests on forwarder software, from real clients we cannot launch defragmentation attacks to check if the devices are vulnerable due to ethical considerations. While fingerprinting methods like [68] seem straightforward, we find it difficult to use these methods to reveal the exact model of the routers.

Measurement overview. Alternatively, we can reach the same goal by checking whether the conditions (listed in Section 4.4) of the attack are satisfied. As such, from a high-level view, we need to collect real-world clients as our vantage points, and check from client side whether the conditions are satisfied by their DNS forwarders.

To perform the measurement study, we collaborate with our industrial partner who develops network diagnosis software for mobile users. They implement our checking methods in the diagnosis tool, which obtains permission to collect

1st fragment

[uuid].attacker.com	**CNAME**	a.attacker.com
a.attacker.com	**CNAME**	b.attacker.com
b.attacker.com	**CNAME**	c.attacker.com
…		
x.attacker.com	**CNAME**	y.attacker.com
y.attacker.com	**CNAME**	uuid.final.attacker.com
[uuid].final.attacker.com	**A**	(nonce address)

2nd fragment

Figure 6: Oversized response of `[uuid].attacker.com`

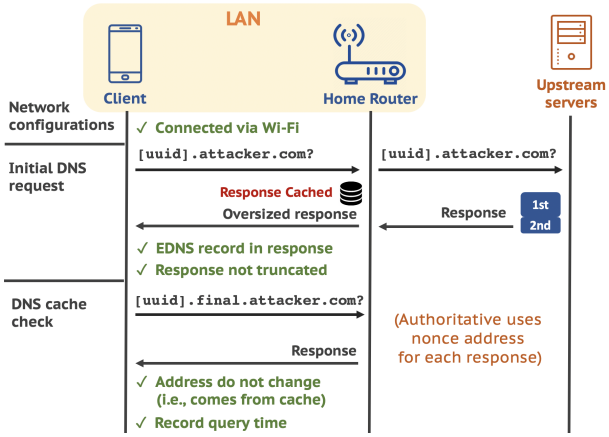

Figure 7: Workflow of client-side measurement

fine-grained DNS data. When run by mobile users, the tool performs several checks on the attack conditions and sends the collected data back to the company's server. The results are further provided to us for deeper analysis. The software has active users mostly located in China. Each mobile client is assigned with a unique ID (termed as `uuid`).

To perform the measurement, the only component we need to configure is the authoritative server (i.e., `attacker.com`). Figure 6 shows the oversized response we create for `[uuid].attacker.com`. A slight difference here is that in the last A record, the authoritative server generates a nonce IP address for each query. Using this technique, from a client we can distinguish whether a DNS response comes from DNS cache.

Attack condition filters. Following the workflow in Figure 7, the checking procedure contains the following steps.

Network configurations. To perform checks on home routers (i.e., through Wi-Fi), the software first check the network environment of each client and remove clients which use mobile data. It also checks basic network configurations, such as client IP address and gateway address.

Initial DNS request. To begin with, each client sends an initial DNS query of `[uuid].attacker.com` with EDNS(0) options[3], to port 53 of the gateway address. If the query times out, it suggests that the router does not support DNS forwarding, and the client is removed from our data. Otherwise, the software checks if an EDNS(0) OPT record presents in the response, which suggests EDNS(0) support (**Filter 1**). It also checks whether the oversized response is truncated by checking the integrity of the CNAME chain (**Filter 2**). If the final A record of `[uuid].final.attacker.com` is intact, it reports the IP address in the record (termed as `addr_init`). Note that if the DNS forwarder supports caching, the initial response should have been written into its DNS cache.

DNS cache check. The client sends queries of `[uuid].final.attacker.com` and report the IP addresses in the responses (termed as `addr_cache`). If the initial response is cached by record, this query should hit the cache, and therefore `addr_init` and `addr_cache` should be the same (**Filter 3**). Otherwise, the authoritative server should be queried again and give another nonce response, thus `addr_init` and `addr_cache` should differ, and we remove the client from our dataset.

Meanwhile, when `addr_init` equals `addr_cache` (i.e., response comes from DNS cache), we need to check whether the response is from the cache of the DNS forwarder or upstream recursive resolvers. Traditionally, one can use non-recursive queries to snoop the DNS cache of recursive resolvers [32]. However, we find that this approach is infeasible for DNS forwarders, as several router models that we test forward non-recursive queries to other servers. As such, we choose to infer the caching position based on timing of the response. On each client, the software repeats the final DNS request of `[uuid].final.attacker.com` for 10 times (due to traffic limit of the diagnosis tool), and calculates the average DNS query time of this cached domain. Based on the average DNS query time, we perform measurements to select clients that are affected (**Filter 4**).

Limitations. We acknowledge that using a timing-based approach is only an estimation of actual affected clients. However, we find that more accurate methods of cache snooping (e.g., non-recursive queries) are not applicable for DNS forwarders. To make the conclusion more reliable, we perform an additional analysis on the DNS query time (hitting a forwarder cache vs. resolver cache), and justify the results based on real-world measurements. Also, because of the software coverage, we can only perform measurements on mobile Wi-Fi users in China. Although we may underestimate the actual population of affected clients, we believe the test results still provide us with an opportunity to understand the impact of the newly discovered attack.

Ethics. The checking method is implemented by our industrial partner on their network diagnosis tool for mobile users, which obtains permission to send and collect network traffic. It is important that on each client, the software *does not launch real attacks to DNS forwarders*, but only checks the attack conditions. Regarding implementation, it

[3] UDP buffer size=4096

only performs ∼10 DNS queries of our controlled domain name exclusively registered for this study. Upon receiving the DNS answers, it does not make connections to the server addresses. Throughout the experiments, no personally identifiable information (PII) or privacy data is collected. In addition, the checking tool uses an encrypted channel to send back collected data to the company's servers.

6.2 Analysis of Affected Population

In the end, we collect valid measurement results from 20,113 mobile clients. The collected clients cover all 31 provinces of mainland China (excluding Hong Kong SAR, Macao SAR and Taiwan), and are distributed in more than 300 (almost all) cities. Also, our clients cover 127 autonomous systems.

When applying our attack condition filters, 79.3% mobile clients are removed by Filters 1-3. In detail, 8,211 (40.8%) clients are using forwarders without EDNS(0) support, 5,695 (28.3%) receive truncated DNS responses, and forwarders of 2,035 (10.1%) clients do not cache the DNS response by record.

For the remaining 20.7% (4,172) mobile clients, we check their average query time of the repeated DNS queries of `[uuid].final.attacker.com` (i.e., Filter 4). Note that because the mobile clients have already passed Filter 3, the repeated queries here should all come from the DNS cache, either of the DNS forwarder of the upstream resolver. Our goal is to keep mobile clients which get the responses from DNS forwarder cache, i.e., exclude mobile clients which obtain responses from recursive resolver cache. To this end, we take the opportunity to measure how long it generally takes for a Wi-Fi client to probe the cache of its upstream recursive resolver. Because learning the upstream resolver address needs manual effort, we choose to perform the measurement using 30 controlled vantage points in China. The vantage points are all connected to home routers through Wi-Fi (i.e., the same environment as the large-scale measurement), which span 11 Chinese provinces and 6 major Chinese ISP networks. We learn the upstream resolver addresses manually from the router configuration pages. On each vantage point, we send cache-probing queries *directly* to the upstream resolver, and record the average time. Figure 8 shows the CDF of upstream resolver cache probing time. We term the ratio of clients which spend *more than t* ms to get a response from the cache of the upstream resolver as $P(t)$, which is the opposite of the CDF. For instance, from Figure 8, $P(10) = 0.7$, $P(11) = 0.6$. Later, we will use this ground truth distribution to extrapolate and estimate how many clients in the complete dataset are hitting the forwarder cache.

For mobile clients that passes Filters 1-3, Figure 9 shows their average time to retrieve a response from the DNS cache. For instance, a total of 139 mobile clients spend 10ms to get a response from DNS cache (either of the DNS forwarder or the upstream resolver). Here we know $P(10) = 0.7$ in

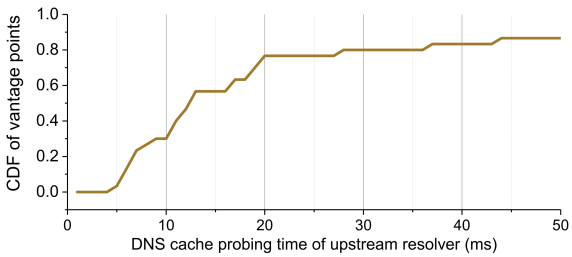

Figure 8: CDF of DNS cache probing time of upstream recursive resolver (30 vantage points).

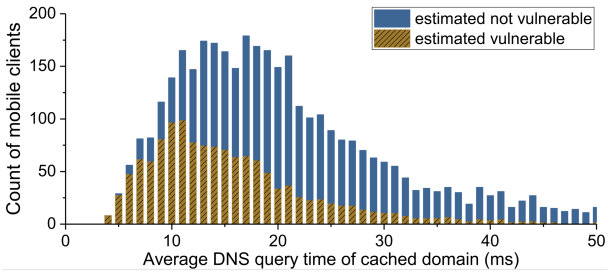

Figure 9: Average DNS query time of a cached domain

the ground truth dataset which we assume will generalize to the complete dataset. This means that 70% (97) of the 139 clients should require more than 10ms to hit upstream resolver cache, so their responses can only come from the DNS forwarder cache (i.e., are vulnerable). Similarly, for the 165 clients which spend 11ms to retrieve a cached response, because $P(11) = 0.6$, we estimate that 60% (99) of the 165 clients are vulnerable. Finally, summing up the client numbers for each time value (i.e., the yellow bars in Figure 9) together, we get an estimation of 1,346 vulnerable clients – 6.6% of the total clients measured in the wild). As expected, when the DNS query time gets longer, it is less likely that the responses come from DNS forwarder cache.

Summary. Overall, a significant portion of the tested DNS forwarders (6.6%) is estimated to be vulnerable to our new defragmentation attack. Different from prior works [33, 48] which have a different set of attack conditions (e.g., on the configuration of authoritative servers), our attack conditions are focused more on the behaviors and configurations of DNS forwarders (and also partly resolvers). Therefore, we do not have constraints on which domains can be attacked. Rather, our constraint is more on which client networks can be attacked. In addition, we estimate that the vulnerable DNS forwarders in the wild will rise because our results indicate that the major attack conditions unsatisfied are EDNS(0) support (40.8%) and correctly handling oversized responses (28.3%). As the new DNS features are getting promoted and increasingly supported by vendors, more users will be affected.

7 Reflection on DNS Forwarders

Our attack further demonstrates that DNS forwarders can be a soft spot in the DNS infrastructure. From our tests in Section 5 we have seen different variations of DNS forwarder implementations. In this section, we further give a discussion on the role of DNS forwarders in the ecosystem. We begin with observations on current implementations, and then discuss the specifications related to DNS forwarders.

7.1 DNS Forwarder Implementations

A general notion of DNS forwarders is that the devices do not resolve queries themselves, but pass the queries to another server. They rely on the integrity and logic checks of the upstream recursive resolvers, and are often not in the position to verify the DNS responses. For instance, none of the home routers that we test verifies the CNAME chain in the response. As a result, vulnerable DNS forwarders are not able to distinguish the rogue responses, which are tampered with after checked by the upstream resolvers. However, if a DNS forwarder performs response verification itself to void the attack (e.g., by "re-querying" or full DNSSEC validation), it is acting in recursive mode, which could not be wanted because of performance overhead.

In fact, from the DNS forwarder implementations that we test in Section 5, we find that the industry does not agree on the role of DNS forwarders in the ecosystem. They can act as transparent DNS proxies, or exhibit behaviors of recursive resolvers. As listed in Table 2 and Table 3, software vendors could disagree on whether their DNS forwarders should have caching abilities, whether they should handle fragmented DNS packets, and whether they should issue queries on their own (e.g., to verify CNAME chains).

7.2 DNS Forwarder Specifications

After researching RFC documents related to DNS, we find that the diverse implementations of DNS forwarders can be caused by the vague definitions in these specifications. In the very original specification of DNS (i.e., RFC 1034 [58]), there is no discussion on DNS forwarding, and the major components of DNS only include name servers and resolvers. As the ecosystem evolves, it now contains multiple layers of servers, including forwarding devices. While DNS forwarders are prevalent in current use, there is still a lack of specific guidelines on their implementation details in the standard documents.

History: two definitions of "forwarder". In Table 4 we list the RFC documents which refer to DNS forwarding. In fact, we find that the standard documents themselves disagree on the definition of DNS forwarders, and have different names for them. Put together, there have been two different descriptions of DNS forwarding devices.

In early specifications, DNS forwarding devices appear to serve as *upstream servers of recursive resolvers*. The devices are leveraged to access authoritative servers, and typically have better Internet connection or bigger caching abilities. The first description of "DNS forwarding" appears in RFC 2136 [70], which refers to an authoritative zone slave forwarding `UPDATE` messages to their master servers. Later, RFC 2308 [29] gives a definition of "DNS forwarder", which implies that forwarders are used to only query authoritative servers. It also says that DNS forwarders are bigger machines which can share their cached data to downstream servers. This term is again used in RFC 7626 [32] on DNS privacy, which suggests that forwarders receive queries from recursive resolvers.

On the other hand, another definition says that DNS forwarders *locate between clients and recursive resolvers*. The devices take queries from clients, and instead of resolving, they pass the requests on to another server. Starting from RFC 3597 [45], the document first describes that forwarders are used by the client. In RFC 7871 [35], "Forwarding Resolvers" use recursive resolvers to handle their queries. For hosts behind broadband gateways, RFC 5625 [31] provide guidelines on the implementations of their DNS proxy devices, which are included as "simple DNS forwarders".

It is not until the very recent specification on DNS terminology (i.e., RFC 8499 [41]) that the definition on DNS forwarders is clarified. According to their common use, DNS forwarders "often *stand between stub resolvers and recursive resolvers*". It also defines DNS forwarding as the process of "sending DNS queries with the `RD` bit set to 1 to another server".

Lacking implementation guidelines. While the term of DNS forwarders has been updated, the specifications do not discuss much about the implementation details. That is, the answer to "what should a DNS forwarder do" is still vague, such as how they should handle DNS responses, whether they should have caches, or whether they can perform like a full-service resolver (e.g., handle referrals and aliases).

The only document we find related to DNS forwarder implementation is RFC 5625 [31], which provides guidelines to DNS proxies (i.e., a subset of DNS forwarders in one specific network). It recommends that DNS proxies should be *as transparent as possible*, and that they should ensure DNS packets are forwarded and returned *verbatim* to their destinations. It is also recommended that DNS proxy devices should be able to forward UDP packets up to 4,096 octets. As a result, a consequence is that a DNS proxy cannot distinguish a spoofed response, if it is tampered with on its way back to the forwarder. In particular, defragmentation attacks have made the tampering task simple, since there is not much entropy for and adversary to guess in the 2nd fragment.

Table 4: DNS forwarder descriptions in RFC documents

RFC No.	Title	Description
2136 [70] (Apr 1997)	**Dynamic Updates in the Domain Name System (DNS UPDATE)**	When a **zone slave** forwards an UPDATE message upward toward the zone's primary master server, it must allocate a new ID and prepare to enter **the role of "forwarding server"**.
2308 [29] (Mar 1998)	**Negative Caching of DNS Queries (DNS NCACHE)**	Forwarder is a nameserver used to resolve queries instead of directly using the authoritative nameserver chain. The forwarder typically either has better access to the internet, or maintains a bigger cache which may be **shared amongst many resolvers**.
3597 [45] (Sept 2003)	**Handling of Unknown DNS Resource Record (RR) Types**	... and in some cases also at caching name servers and **forwarders used by the client**.
5625 [31] (Aug 2009)	**DNS Proxy Implementation Guidelines**	(DNS) proxies are usually **simple DNS forwarders**, but typically do not have any caching capabilities. The proxy serves as a convenient default DNS resolver for clients on the LAN, but **relies on an upstream resolver** (e.g., at an ISP) to perform recursive DNS lookups.
7626 [32] (Aug 2015)	**DNS Privacy Considerations**	DNS recursive resolvers sometimes forward requests to other recursive resolvers, ... these **forwarders are like resolvers**, except that they do not see all of the requests being made.
7871 [35] (May 2016)	**Client Subnet in DNS Queries**	Forwarding Resolvers essentially appear to be Stub Resolvers to whatever **Recursive Resolver is ultimately handling the query**, but they look like a Recursive Resolver to their client.
8499 [41] (Jan 2019)	**DNS Terminology**	In current use, however, forwarders often stand **between stub resolvers and recursive servers**.

8 Attack Model Extension and Mitigation

In this section, we extend our attack model to open DNS forwarders. Further, we propose mitigation to the new defragmentation attack.

8.1 Extending the Attack Model

In our extended model, we remove the requirement that the attacker and the DNS forwarder have to be located in the same LAN. For example, the attack can also be possible for open DNS forwarders out on the Internet. [62] proposes a method on how to detect such open forwarders. As we show in Figure 10, a major difference here is that it is much harder for the attacker to predict the IPID from the resolver to the DNS forwarder, unless the resolver uses a globally incrementing IPID counter, in which case such open forwarders will be obviously vulnerable. In the case of hash-based IPID counter, the recursive resolver is likely going to have two separate IPID counters for the forwarder and attacker (depending on if there is a hash collision). Therefore, it is difficult for the attacker to predict the IPID value of the resolver's response packets sent to the forwarder.

However, prior defragmentation attacks have proposed techniques such as meet-in-the-middle [42, 49], which can still infer the current IPID counter despite that the attacker, using its own IP address sending probes, would only observe a different IPID counter (due to the attacker's IP hashing into a different counter). A recent technique also suggested that an attacker who controls multiple IP addresses can probabilistically force a hash collision, in which case the attacker would still succeed. We believe such attacks are promising and would affect many more users. Due to reasons such as ethics, we leave it as future work to validate such attacks in practice.

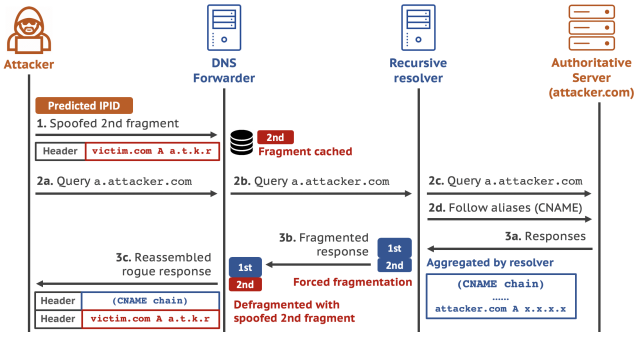

Figure 10: Defragmentation cache poisoning attack targeting open DNS forwarders

8.2 Mitigation

Recall that in Section 4.4, we list several conditions of a vulnerable DNS forwarder implementation. Intuitively, breaking any of the conditions will void the attack. However, measures like removing DNS cache or EDNS(0) support are not advised as they are compromising new and important functionalities.

Response verification. The first solution is for the DNS forwarder to verify the oversized DNS response. In detail, it can re-query all names and aliases in the CNAME chain, or perform full DNSSEC validation. For example, in our DNS software test (see Table 3), we find that BIND and other 3 kinds of software adopt the "re-query" approach. As a result, the rogue records should not pass verification, and the

attack fails. However, this approach requires that the DNS forwarder should be able to perform recursive queries, which could not be wanted in certain use cases. Meanwhile, it brings significant performance overhead, which contradicts with the purpose of DNS forwarders, and might not be feasible for devices with limited resources (e.g., home routers).

DNS caching by response. An ad hoc approach to void the defragmentation attack is to change how forwarders cache the responses. As discussed in Section 4.4, vulnerable devices cache DNS responses by individual records. By caching them as a whole, the rogue records in the last part of the response (see Figure 5(b)) will not hit the cache. The approach is practical, as it only requires changes on the forwarder itself. From the disclosure responses (see Section 5.4), the updated firmware of ASUS router adopts this defence. We recommend this solution as a short-term countermeasure. However, due to the uncertainty of the role of forwarders, it is unclear what their expected behaviors should be (as RFCs do not specify this), and whether caching by responses will hurt performance.

0x20 encoding on DNS records. Similar to previous defences of DNS cache poisoning, the essence of this mitigation is to increase randomness of the response (specifically, the 2nd fragment). As the 2nd fragment lacks DNS and UDP metadata, its entropy can be increased by encoding the DNS records, using an upgraded version of 0x20 encoding [36]. While the original 0x20 encoding only mixes cases of query names in the question section, here we need recursive resolvers to encode names and aliases *in all records of the answer section* oversized response. To go along with this, the DNS forwarder should also check the cases of each record when receiving a DNS response. The downside of this mitigation is that it needs changes from upstream recursive resolvers, thus cannot be deployed shortly.

Randomizing IPID values. Random IPID values makes any defragmentation-based attacks (including ours) much more difficult, as they require the prediction of future IPID values. Interestingly, as we have tested and described earlier in Section 4.3, major operating systems such as Windows and Linux do not exhibit such a random IPID behavior. Yet in our measurement, we do find Google and Versign's resolvers appear to have such behaviors. We suspect that it is either because they have used uncommon/customized operating systems and network middleboxes (that rewrite the IPIDs), or that there are actually still multiple hosts sitting behind the same egress IP address (e.g., through NAT). In any event, random IPID values are not impossible to guess, especially given that the attacker can place 64 guessed values (out of 64K possible values). Furthermore, if the attack is repeated multiple times, the likelihood of success will increase as well. As a result, it is not a bullet-proof mitigation.

9 Other Related Work

Security risks of DNS forwarders. As mentioned earlier, a DNS forwarder does not perform recursive DNS lookup themselves, but simply forwards DNS requests to an upstream resolver. In order to mitigate the security risks of DNS cache poisoning and denial of service attacks, DNS forwarders are widely implemented in network products related with DNS protocol, such as home routers, as it not directly exposed to Internet attackers [49]. It is also recommended by some DNS experts, e.g., Kaminsky [24].

Unfortunately, many DNS forwarders themselves are not patched and are vulnerable to DNS cache poisoning attacks [49]. In some cases, DNS forwarders fail to validate the DNS responses, such as the DNS transaction ID, source IP address and the destination port number. A measurement study shows that at least 8.6% open DNS resolvers in the wild are vulnerable to the DNS cache poisoning attacks [63]. Therefore, in spite of the availability of DNSSEC, DNS record injection vulnerabilities are still fairly common among DNS forwarders until now.

Compared to previous works, in this paper we further present a type of cache poisoning attack targeting DNS forwarders. The methods can circumvent traditional defences against cache poisoning attacks. Combined with previous attacks, our work further demonstrates that DNS forwarders can be a soft spot in the infrastructure.

10 Conclusion

As the DNS infrastructure has evolved dramatically, today it involves multiple layers of servers. DNS forwarders are widely-deployed devices, however we show that they can be a soft spot that is more vulnerable to cache poisoning attacks. Using fragmented DNS packets and oversized response, an attacker can inject rogue DNS records of arbitrary domain names into the forwarders' cache, and bypass common defences including randomized ephemeral ports and 0x20 encoding. By testing on current implementations, we find several home router models and DNS software vulnerable to this attack, including those of large vendors. Meanwhile, through a nationwide measurement study, we assess the affected population of mobile clients using the vulnerable devices. From the implementations we find a diversity in the industry on understanding the role of DNS forwarders. Also, there is still a lack of implementation guidelines on forwarding devices in the DNS specifications. As such, we believe more attention should be raised from the community to the understanding and the security status of DNS forwarders.

Acknowledgements

We sincerely thank all anonymous reviewers for their valuable comments to improve the paper. We also thank the GeekPwn Cyber Security Competition.

This work is supported by National Key R&D Program of China, Grant No. 2017YFB0803202; NSFC Grant No. U1836213, U1636204; State Key Laboratory of Computer Architecture (ICT, CAS) under Grant No. CARCH201703; National Science Foundation under Grant No. 1652954, 1646641 and 1619391.

References

[1] Cloud delivered enterprise security by opendns. https://www.opendns.com/.

[2] Cloudflare Resolver. https://cloudflare-dns.com/.

[3] Comodo secure dns. https://www.comodo.com/secure-dns/.

[4] Create your perfect wifi system - linksys. https://www.linksys.com/us/.

[5] D-link: Consumer. https://www.dlink.com/en/consumer.

[6] Dnrd, domain name relay daemon. http://dnrd.sourceforge.net/.

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

[8] Domain name system (dns) overview. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh831667(v=ws.11).

[9] Gee hiwifi. https://www.hiwifi.com/.

[10] Google Public DNS. https://developers.google.com/speed/public-dns/.

[11] Huawei wireless routers-huawei official site. https://consumer.huawei.com/eg-en/support/smart-home/wireless-routers/.

[12] Introducing a new whoami tool for dns resolver information. https://developer.akamai.com/blog/2018/05/10/introducing-new-whoami-tool-dns-resolver-information.

[13] Knot DNS. https://www.knot-dns.cz/.

[14] Mi router 3 - mi.com. https://www.mi.com/mea/mi-router-3/.

[15] Netgear, howpublished = https://www.netgear.com/.

[16] Norton connectsafe. https://www.publicdns.xyz/public/norton-connectsafe.html.

[17] Openwrt project. https://openwrt.org/.

[18] Powerdns. https://www.powerdns.com/.

[19] Quad9 DNS: Internet Security & Privacy In a Few Easy Steps. https://www.quad9.net/.

[20] Tenda wireless router. http://simulator.tendacn.com/N301v2/.

[21] Tp-link: Wifi networking equipment for home & business. https://www.tp-link.com/us/.

[22] Verisign public dns offers dns stability and security. https://www.verisign.com/en_US/security-services/public-dns/index.xhtml.

[23] Wireless routers | networking | asus global. https://www.asus.com/Networking/Wireless-Routers-Products/.

[24] Dan kaminsky's blog. http://dankaminsky.com/2008/07/21/130/, 2008.

[25] Bind 9 - versatile, classic, complete name server software. https://www.isc.org/bind/, 2019.

[26] Bugcrowd. https://www.bugcrowd.com/, 2019.

[27] Nlnet labs - unbound. https://nlnetlabs.nl/projects/unbound/about/, 2019.

[28] Geoffrey Alexander, Antonio M Espinoza, and Jedidiah R Crandall. Detecting tcp/ip connections via ipid hash collisions. *Proceedings on Privacy Enhancing Technologies*, 2019(4):311–328, 2019.

[29] Mark Andrews. Negative caching of dns queries (dns ncache). 1998.

[30] R Arends, R Austein, M Larson, Daniel Massey, and Scott W Rose. Protocol modifications for the dns security extensions rfc 4035. Technical report, 2005.

[31] Ray Bellis. Dns proxy implementation guidelines. 2009.

[32] Stephane Bortzmeyer. Dns privacy considerations. 2015.

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

[34] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. A longitudinal, end-to-end view of the {DNSSEC} ecosystem. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1307–1322, 2017.

[35] Carlo Contavalli, Wilmer van der Gaast, David C Lawrence, and Warren Kumari. Rfc 7871-client subnet in dns queries. 2016.

[36] David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. Increased dns forgery resistance through 0x20-bit encoding: security via leet queries. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 211–222. ACM, 2008.

[37] Joao Damas, Michael Graff, and Paul Vixie. Extension mechanisms for dns (edns (0)). 2013.

[38] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A search engine backed by Internet-wide scanning. In *22nd ACM Conference on Computer and Communications Security*, October 2015.

[39] R Elz and R Bush. Rfc 2181: Clarifications to the dns specification. Technical report, Updates RFC 1034, RFC 1035, RFC 1123. Status: Proposed standard, 1997.

[40] Kazunori Fujiwara. Measures against dns cache poisoning attacks using ip fragmentation. https://indico.dns-oarc.net/event/31/contributions/692/attachments/660/1115/fujiwara-5.pdf.

[41] Kazunori Fujiwara, Andrew Sullivan, and Paul Hoffman. Dns terminology. 2019.

[42] Yossi Gilad and Amir Herzberg. Fragmentation considered vulnerable: blindly intercepting and discarding fragments. In *Proceedings of the 5th USENIX conference on Offensive technologies*, pages 2–2. USENIX Association, 2011.

[43] Yossi Gilad, Amir Herzberg, and Haya Shulman. Off-path hacking: The illusion of challenge-response authentication. *IEEE Security & Privacy*, 12(5):68–77, 2013.

[44] Fernando Gont. Rfc 7739-security implications of predictable fragment identification values. 2016.

[45] Andreas Gustafsson. Handling of unknown dns resource record (rr) types. 2003.

[46] John W Heffner, Ben Chandler, and Matt Mathis. Ipv4 reassembly errors at high data rates. 2007.

[47] Amir Herzberg and Haya Shulman. Security of patched dns. In *European Symposium on Research in Computer Security*, pages 271–288. Springer, 2012.

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

[49] Amir Herzberg and Haya Shulman. Vulnerable delegation of dns resolution. In *European Symposium on Research in Computer Security*, pages 219–236. Springer, 2013.

[50] Paul Hoffman, Andrew Sullivan, and K Fujiwara. Dns terminology. Technical report, 2019.

[51] Charles Hornig. A standard for the transmission of ip datagrams over ethernet networks. Technical report, 1984.

[52] A Hubert and R Van Mook. Measures for making dns more resilient against forged answers. Technical report, RFC 5452, January, 2009.

[53] Dan Kaminsky. The massive, multi-vendor issue and the massive, multi-vendor fix. Technical report, 2008.

[54] Christopher A Kent and Jeffrey C Mogul. *Fragmentation considered harmful*, volume 17. 1987.

[55] Amit Klein and Benny Pinkas. From IP ID to device ID and KASLR bypass. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1063–1080, Santa Clara, CA, August 2019. USENIX Association.

[56] Jeffrey Knockel and Jedidiah R Crandall. Counting packets sent between arbitrary internet hosts. In *4th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 14)*, 2014.

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

[58] Paul Mockapetris. Rfc-1034 domain names-concepts and facilities. *Network Working Group*, page 55, 1987.

[59] Jeffrey C Mogul and Steven E Deering. Path mtu discovery. Technical report, 1990.

[60] VC NewsNetwork. Wifi home router market 2019 global analysis, opportunities and forecast to 2025. https://www.reuters.com/brandfeatures/venture-capital/article?id=105961, 2019.

[61] Vicky Risk. Edns (in) compatibility. https://www.isc.org/docs/DNS-OARC-EDNS-Compliance.pdf, 2015.

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

[63] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. Assessing dns vulnerability to record injection. In *Proceedings of the Passive and Active Measurement Conference*, 2014.

[64] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. Dns record injectino vulnerabilities in home routers. http://www.icir.org/mallman/talks/schomp-dns-security-nanog61.pdf, 2014.

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

[66] Haya Shulman and Michael Waidner. Fragmentation considered leaking: port inference for dns poisoning. In *International Conference on Applied Cryptography and Network Security*, pages 531–548. Springer, 2014.

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

[68] Yves Vanaubel, Jean-Jacques Pansiot, Pascal Mérindol, and Benoit Donnet. Network fingerprinting: Ttl-based router signatures. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 369–376. ACM, 2013.

[69] Paul Vixie. Dns and bind security issues. In *Usenix Security Symposium*, 1995.

[70] Paul Vixie, S Thomson, Y Rekhter, and J Bound. Rfc 2136: Dynamic updates in the domain name system (dns update), 1997.

[71] Xu Zhang, Jeffrey Knockel, and Jedidiah R Crandall. Onis: Inferring tcp/ip-based trust relationships completely off-path. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2069–2077. IEEE, 2018.

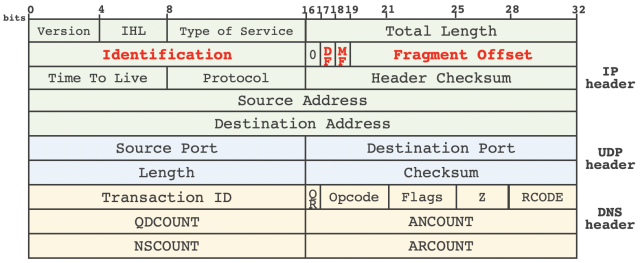

Figure 11: Headers in a DNS packet

Appendices

A IP Fragmentation

IP fragmentation allows IP datagrams to be transmitted through networks which limit packets to a small size. On an arbitrary internet path, Path Maximum Transmission Unit (PMTU) defines the size limit of IP packets, and datagrams larger than PMTU will be fragmented. PMTU equals the minimum MTU of each hop in the path, and can be discovered using a technique described in [59]. Particularly, the MTU of Ethernet is 1,500 bytes [51].

As shown in Figure 11, IP fragmentation and reassembly is supported by using several fields of the IP header: Identification (IPID), Don't Fragment bit (`DF`), More Fragment bit (`MF`) and Fragment Offset. If a sender does not desire a datagram to be fragmented, the `DF` flag is set. The `MF` flag indicates whether this is the last fragment of the datagram, and is cleared in the last fragment. Fragment Offset shows the position of current fragment in the original datagram. Most importantly, fragments of one IP datagram have the same IPID, in order to be correctly reassembled.

Specifically for DNS packets, they contain IP header, UDP header and DNS header. If a DNS packet is fragmented, *only the first fragment* will have UDP header and DNS header.

Fragmentation considered "harmful". Despite being one of the IP basic functions, there has been long discussions on the problems caused by IP fragmentation. The earliest report on the issue dates back to 1987 [54], which shows that fragmentation can lead to poor performance and complete communication failure. As documented by [46], IP fragmentation can also result in frequent data corruption. In recent studies, IP fragmentation can be used to circumvent DNS cache injection defences [43, 48], or cause CAs to issue fraudulent certificates [33]. Because of the security issues, there have been discussions on completely avoiding fragmentation behaviors [40].

B IPID Assignment of Public DNS Services

Using the technique described in Section 4, we test the IPID assignment of egress resolvers of 7 public DNS services. We

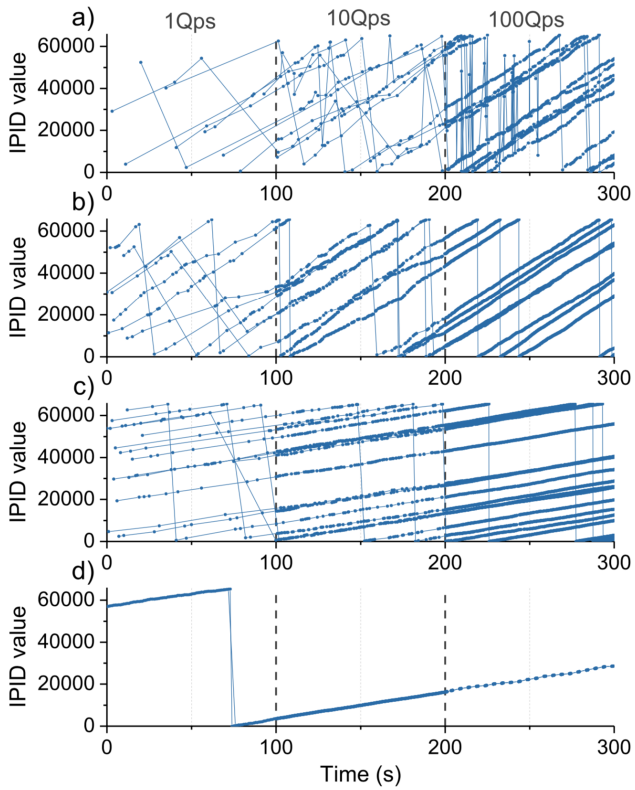

Figure 12: IPID assignment of a) Cloudflare DNS, b) Quad9 DNS, c) OpenDNS, and d) Comodo Secure DNS, observed from one vantage point. Each line represents one egress resolver, and each dot marks one DNS response packet.

use two vantage points as DNS clients, and start the measurement on both machines at the same time. We change the speed of DNS queries every 100 seconds (from 1Qps, 10Qps to 100Qps).

Hash-based IPID counters. We find that egress resolvers of Cloudflare, Quad9, OpenDNS, Comodo and Norton use hash-based IPID counters. Figure 12 shows the IPIDs of DNS responses received by one DNS client. After separating responses from different egress resolvers (i.e., lines in Figure 12), we find that the egress resolvers use predictable

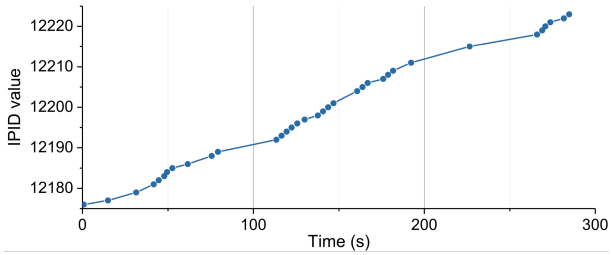

Figure 13: IPID assignment of Norton ConnectSafe DNS for fragmented DNS responses (egress resolver: 156.154.180.*).

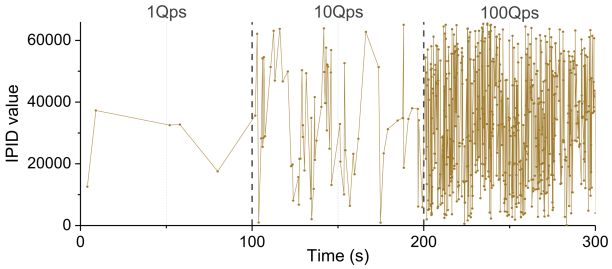

Figure 14: IPID assignment of Google Public DNS (egress resolver: 172.253.0.*).

incremental IPID counters. The increments are linear with time, because in hash-based algorithms each IPID counter is shared by an array of destination addresses. We confirm that the algorithm is hash-based, because the IPIDs of DNS reponses sent to our two vantage points are not related.

Particularly, as shown in Figure 13, Norton ConnectSafe uses hash-based IPID counters for fragmented DNS responses only, and uses zero IPID values when they are not fragmented. This design has made IPID prediction easier, as most DNS packets on the Internet are not fragmented, so the IPID counters are hardly increased by normal responses.

Random IPID assignment. As shown in Figure 14, egress resolvers of Google and Verisign use random IPID assignment. As upstream resolvers, the two services cannot be exploited in the attack.