TuDoor Attack: Systematically Exploring and Exploiting Logic Vulnerabilities in DNS Response Pre-processing with Malformed Packets

Xiang Li[¶], Wei Xu[¶], Baojun Liu[¶], Mingming Zhang[¶‡], Zhou Li[†✉], Jia Zhang[¶‡], Deliang Chang[£], Xiaofeng Zheng[¶£], Chuhan Wang[¶], Jianjun Chen[¶‡], Haixin Duan[¶‡§✉], and Qi Li[¶✉]

[¶]*Tsinghua University*, [†]*University of California, Irvine*, [‡]*Zhongguancun Laboratory*
[§]*Quan Cheng Laboratory*, [£]*QI-ANXIN Technology Research Institute*, [✉]*Corresponding Author(s)*

***Abstract*—DNS can be compared to a game of chess in that its rules are simple, yet the possibilities it presents are endless. While the fundamental rules of DNS are straightforward, DNS implementations can be extremely complex. In this study, we intend to explore the complexities and vulnerabilities in DNS response pre-processing by systematically analyzing DNS RFCs and DNS software implementations. We present the discovery of three new types of logic vulnerabilities, leading to the proposal of three novel attacks, namely the TuDoor attack. These attacks involve the use of malformed DNS response packets to carry out DNS cache poisoning, denial-of-service, and resource consuming attacks. By performing comprehensive experiments, we demonstrate the attack's feasibility and significant real-world impacts of TuDoor. In total, 24 mainstream DNS software, including BIND, PowerDNS, and Microsoft DNS, are affected by TuDoor. Attackers can instigate cache poisoning and denial-of-service attacks against vulnerable resolvers using a handful of crafted packets within 1 second or circumvent the query limit to deplete resolution resources (e.g., CPU). Besides, to determine the vulnerable resolver population in the wild, we collect and evaluate 16 popular Wi-Fi routers, 6 prevalent router OSes, 42 public DNS services, and around 1.8M open DNS resolvers. Our measurement results indicate that TuDoor could exploit 7 routers (OSes), 18 public DNS services, and 424,652 (23.1%) open DNS resolvers. Following the best practice of responsible disclosure, we have reported these vulnerabilities to all affected vendors, and 18 of them, including BIND, Chrome, Cloudflare, and Microsoft, have acknowledged our findings and discussed mitigation solutions with us. Furthermore, 33 CVE IDs are assigned to our discovered vulnerabilities, and we provide an online detection tool as one of the mitigation measures. Our research highlights the urgent need for standardization of DNS response pre-processing logic to enhance the security of DNS.**

1. Introduction

The Domain Name System (DNS) serves as a crucial component of the Internet infrastructure, translating human-readable domain names to machine-readable IP addresses and vice versa. DNS is extensively used to access websites and support fundamental security mechanisms, including email communication [118], certificate validation [32], blacklists [86], and sinkholing [6]. Consequently, DNS has become a notorious target for various network attacks, e.g., DNS cache poisoning and Denial-of-Service (DoS) attacks.

The domain name resolution process consists of three major components: the client-resolver side, which receives queries and returns final answers; the internal portion, which operates data and manages cache; and the resolver-server side, which requests resource records and handles responses.

Prior work. Previous studies demonstrated that attackers often employ DNS flood [31], [113] and random subdomain attacks [67] to overwhelm the client-resolver side, resulting in network bandwidth and resolution resource depletion and thus reducing the availability of DNS services [15]. Recent studies also revealed a series of vulnerabilities existing in the internal portion of DNS resolvers, including flawed cache management and update strategies [57], [62], [69], as well as unexpected network side-channels [74], [75]. These vulnerabilities can result in cache poisoning and traffic amplification attacks. As for the resolver-server side, few previous works focused on the DNS response processing or only a fraction of it, such as the cache poisoning model [35] and bailiwick checking implementations [71], [103]. Until now, *there has been no systematic study attempting to understand the whole DNS response pre-processing procedure and explore potential logic vulnerabilities.*

Our study. To address this research gap, we systematically reviewed DNS RFCs and "reverse engineered" 28 mainstream DNS software, including BIND, Unbound, Knot, and PowerDNS, by inspecting their source code and conducting black-box testing. The in-depth analysis enabled summarizing the general workflow of DNS response pre-processing with state machines. Based on valuable insights and states, we designed malformed DNS response packets for testing and identified different implementations of DNS response pre-processing. The analysis uncovered three novel logic vulnerabilities exploitable for cache poisoning (DNSPoisoning), denial-of-service (DNSDoS), and resource consuming (DNSConsuming) attacks via *injecting malformed packets through DNS resolvers' response receiving "door", resulting in our proposed "TuDoor attack" (In history, TuDoor, in Chinese characters 突门, was a hidden door in the Great Wall for scouts to pass through [121]).*

Specifically, we developed a state machine for representing DNS response pre-processing. While all software attempted to await a time frame for normal responses, we discovered that they failed to handle several corner cases of malformed packets. The vulnerabilities in the state machine could be exploited for V_{CP} (cache poisoning), V_{DS} (DoS), and V_{RC} (resource consuming) attacks (see Section 4). *The vulnerabilities can affect 24 DNS software in total. (i)* V_{CP} provides a covert side-channel for pinpointing the source port used for resolution, allowing the fastest cache poisoning attack (DNSPOISONING) within only one second towards arbitrary domains, including TLDs like `.com` and `.net`. *(ii)* With V_{DS}, attackers can conduct a DoS attack (DNSDOS) on vulnerable resolvers and clients within one second, preventing valid responses from SLDs' or even TLDs' servers by sending a few crafted responses rather than occupying network bandwidth. *(iii)* V_{RC} enables DNSCONSUMING to bypass the resolvers' query limit employed for mitigating attacks like [3], [83] and force more queries, resulting in resource (such as CPU) consuming attacks. (see Section 5).

We also investigated the prevalence of vulnerable resolvers (see Section 6). We evaluated 16 prominent Wi-Fi routers, 6 prevalent router operating systems, 42 prominent public DNS services, and about 1.8 million open DNS resolvers through Internet-wide scanning. Through experiments with ethical considerations, we uncovered a significant population of vulnerable resolvers. We identified 7 routers (OSes) that are vulnerable to cache poisoning attacks, while one public DNS service (114DNS) is influenced by DNSPOISONING and 17 other public DNS services are vulnerable to DNSDOS. Furthermore, we found that *423,652* (23.1%) open resolvers are exploitable by the TUDOOR attack. Our measurement results demonstrated the real-world impact of the TUDOOR attack.

Disclosure and feedback. We have reported discovered vulnerabilities to all the affected parties, including 22 DNS software vendors, 18 public DNS service providers, 7 router (OS) producers, and network operators (see Section 7). We have received acknowledgments from 18 vendors, including BIND, ChromeOS, Cloudflare, and Microsoft, and have worked closely with them to develop mitigation strategies. We are still waiting for responses from the other vendors. To protect DNS resolvers from the TUDOOR attack, we also provide a comprehensive set of mitigation solutions and detection approaches. In addition, we release an online checking tool that customers can use to examine their resolvers at this website: https://test.tudoor.net. Our discovered vulnerabilities have been assigned 33 CVE IDs, and some vendors have already released patched versions according to our suggestions. In summary, our study calls for standardizing the DNS response pre-processing logic and implementation, as there is presently no rigorous specification.

Contributions. We make the following contributions:

- We provide an in-depth analysis of the DNS response pre-processing logic, summarizing its generic workflow and finite-state machine, and presenting various flawed implementations.
- We identify three novel types of DNS vulnerabilities in the pre-processing logic and propose three corresponding attacks, collectively named the TUDOOR attack.
- We conduct comprehensive experiments to evaluate the population of vulnerable resolvers in the wild and demonstrate TUDOOR's real-world impact.
- We report our newly discovered vulnerabilities to all affected vendors, along with detailed mitigation and detection solutions.

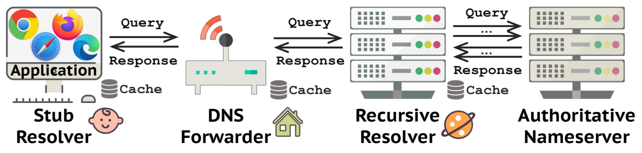


Figure 1. General DNS resolver roles and domain name resolution process.

2. Background

In this section, we provide an overview of DNS concepts and several crucial resolution mechanisms. Additionally, we discuss ICMP messages and their impact on the application.

2.1. DNS Overview

DNS concepts and resolution process. The Domain Name System (DNS) serves to improve the usability of IP applications by providing domain name-to-IP address mappings, and vice versa. The DNS infrastructure comprises two primary components: the DNS namespace and DNS resolution. The DNS namespace is a distributed hierarchical database consisting of zones separated by a period ("`.`") that is managed by authoritative nameservers at each level. Each zone contains authoritative resource records for corresponding domains, such as type `A` (IPv4 addresses) and `AAAA` (IPv6 addresses). The upper zone contains reference information of all subzones underneath it, i.e., `NS` (nameservers) records. Figure 3 shows several examples of resource records. For instance, the domain `example.com` is made up of three zones: the root zone "`.`", the top-level domain (TLD) zone `.com`, and the second-level domain (SLD) zone "`example.com`".

Figure 1 illustrates the general DNS resolution process. When the client browses a website such as `example.com`, the IP application utilizes the OS application programming interface (API), such as `getaddrinfo()` [28], to communicate with the stub resolver inside the TCP/IP stack, such as `systemd-resolved` [105]. The stub resolver formulates a DNS request to its pre-configured DNS forwarder, which is often integrated into home or Wi-Fi routers [27], [60] to improve resolution speed by sending DNS queries on behalf of users. Upon receiving the request, the DNS forwarder forwards the request to the recursive resolver, which performs the iterative DNS resolution process to retrieve the final results. Specifically, the recursive resolver sends queries from the root to TLD to SLD authoritative nameservers. The upper nameserver returns referral information to inform the

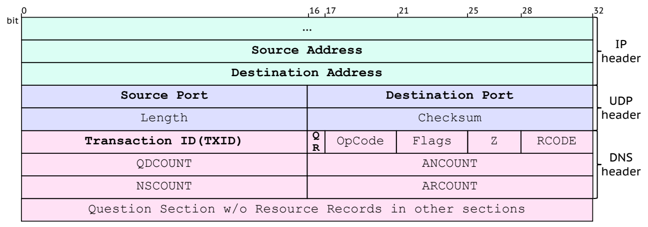
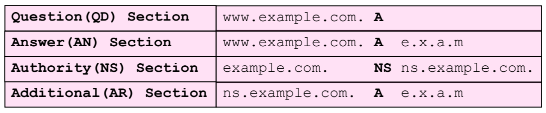


Figure 2. DNS packet format on UDP.



Question(QD) Section	www.example.com. **A**
Answer(AN) Section	www.example.com. **A** e.x.a.m
Authority(NS) Section	example.com. **NS** ns.example.com.
Additional(AR) Section	ns.example.com. **A** e.x.a.m

Figure 3. Exemplar DNS sections and resource records.

recursive resolver of the next "closer" nameserver. Finally, the nameserver of SLD `example.com` replies with authoritative data, and the resolution process is complete. If resolvers are equipped with the cache mechanism, they will store the response in the local cache databse for future use.

DNS packets. Originally specified in RFC 1034-1035 [81], [82], DNS typically utilizes UDP to transmit query and response payloads. A DNS packet consists of a 12-byte DNS header and a DNS body, which is illustrated in Figure 2. The DNS header records a transaction ID (`TXID`) value for authentication, a `QR` flag that denotes a query or response packet, and other fields such as the operation code (`OpCode`), return code (`RCODE`), and the number of resource records. Meanwhile, the DNS body carries question and answer data and includes four sections that are shown in Figure 3. The question section is present in both the query and response packets, indicating the query name and type. The answer section is utilized to return the final resolution results (with `AA` in `Flags`) from the authoritative nameserver, whereas the authority and additional sections provide the referral information (with no `AA` in `Flags`) from the upper nameserver.

2.2. DNS Resolution Mechanisms

DNS resolvers process queries and responses in a similar way. In the following section, we describe several crucial resolution mechanisms involved in response processing.

DNS response processing. The DNS response processing procedure proceeds as follows for each response that arrives [82]. First, the response is parsed to ensure it conforms to the valid DNS packet format. Second, the response is compared to the current resolution request using the `TXID`. Third, DNS data is handled based on specified standards and rules, such as question matching and bailiwick rules [23], [44]. *Any invalid responses should be discarded.* Finally, *resolvers consider the first valid DNS packet that arrives as the final answer.* If no valid reply is received, a `ServFail` response is sent to clients. The above operation is a common practice among DNS developers, as discussed in Section 4.

Despite the fact that DNS standards [82] require DNS resolvers to discard malformed responses, no specific implementation guidelines are provided. In Section 4, we present a comprehensive analysis of DNS response pre-processing through empirical studies, including different software implementations, and identify multiple logic vulnerabilities.

DNS query retransmission. After receiving a client's request, the resolver transforms it into a query and sends it to upstream servers on port 53. Subsequently, the resolver will wait for a response within a specified timeout window. If the response is invalid or fails to arrive within the allotted time frame, the resolver will resend queries several times, using previously sent or newly-constructed packets, until it reaches the total resolution timeout limit and returns a `ServFail` response [23], [82]. To prevent aggressive retransmission, resolvers implement query detection approaches that restrict the number of queries [3], [17]. For example, Knot Resolver limits the number of retries to a maximum of 9 [63].

DNS negative caching. DNS caching reduces query latency and overload by storing previously received DNS data, as well as non-response results (negative caching) from resolution failures, such as "server failures", "timeouts", or "server refused" [10], [23], [119]. In cases of resolution failures, the RFC draft [119] recommends a maximum retry value of 2 and a typical timeout value ranging from 3 to 30 seconds. If no valid answer is received, the negative response is cached for at least 5 seconds and a `ServFail` response is returned.

2.3. ICMP Messages and Impact

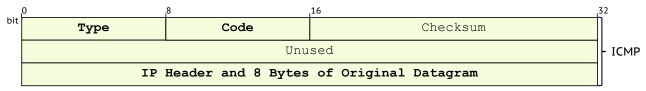


Figure 4. ICMP message format.

The Internet Control Message Protocol (ICMP) is used to report operational information and error messages to the sender, which is specified in RFC 792 [90] for IPv4 and in RFC 4443 [49] for IPv6. When a transmission error occurs en route, the intermediate node or the receiver generates an ICMP error message, which includes at least the four-tuple (source address, destination address, source port, and destination port) and the first 8 data octets of the original datagram, as shown in Figure 4.

According to RFCs [22], [47], the OS kernel should forward ICMP error messages to the application layer only if the wrapped four-tuple corresponds to an existent socket; then, the application (including DNS) is responsible for processing the ICMP error message. For example, in the case of a port unreachable error message, the resolver may close the receiving connection or ignore it. However, none of the TCP, UDP, and ICMP RFCs [22], [89], [90], [91] recommend any validation checks on received ICMP error messages. In 2006, Gont et al. [46] demonstrated various ICMP attacks against TCP, such as TCP connection reset. Nevertheless, there has been little discussion of ICMP attacks on UDP, especially for the application layers such as DNS [9].

In this paper, we present a comprehensive analysis and testing of current DNS implementations on ICMP error mes-

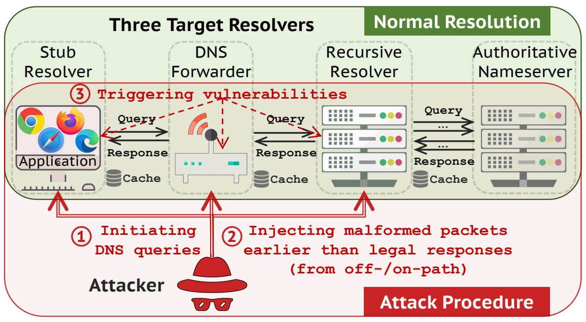


Figure 5. Threat model of the TuDoor attack.

sage processing and identify huge differences. Specifically, we use the ICMP Destination Unreachable Error message (type 3) for testing, covering code 0 (Net Unreachable), code 1 (Host Unreachable), code 2 (Protocol Unreachable), and code 3 (Port Unreachable).

3. TuDoor Attack Overview

In this paper, we discover novel logical vulnerabilities in DNS response pre-processing that facilitate the injection of malformed packets into resolvers through a covert and effective channel (see Section 4.3). This channel enables a range of serious attacks, including cache poisoning, denial-of-service, and resource consuming, which can be conducted by attackers using our newly proposed techniques (see Section 5). We refer to them as *TuDoor* attacks (a convert door).

In this section, here, we first describe the threat model of the TuDoor attack, and then dive deeper into the high-level workflow of TuDoor attacks. We leave more exploitation and experimental details in Section 5.

3.1. Threat Model

The TuDoor attack targets various resolver roles in the DNS system, including stub resolvers, DNS forwarders, and recursive resolvers. In this paper, we assume that an adversary has the following capabilities: the ability to initiate a DNS query to target resolver and the ability to obtain the egress IP addresses of target resolver. The threat model of TuDoor attacks is illustrated in Figure 5.

For open resolvers distributed in the Internet, attackers can directly send DNS queries targeting the resolver from any location [101]. For DNS resolvers that serve limited networks, such as those located in home or enterprise networks, attackers can collect vantage points from large-scale measurement platforms [99] or residential proxy networks [73], [79] to generate DNS queries. For stub resolvers, an attacker can employ online advertisements and spam emails that embed the attacker's controlled domain and direct clients to launch DNS queries. Notably, because TuDoor primarily exploits UDP response pre-possessing, attackers could initiate client queries via any connection type, including UDP, TCP, HTTPS, and TLS. Particularly affected are responses received via TCP, HTTPS, and TLS for Knot Resolver.

Prior to starting the TuDoor attack, an adversary also need to collect the egress IP address of the target resolver for packet injection. To achieve this, an attacker could query the target resolver or make the machine running a stub resolver visit his or her own website in advance. The attacker can then obtain the egress IP address from the perspective of his or her authoritative nameserver.

Besides, in the cases of cache poisoning and DoS attacks, the attacker is assumed to be off-path and needs to spoof the source IP address of a malformed DNS response packet. According to the latest statistics from CAIDA [26], over 19% of the IPv4 ASes are classified as IP-spoofable, making it still possible for an attacker to use any bulletproof hosting service [7] within these ASes to spoof the source address. Specifically, for cache poisoning, we do not consider resolvers that enable DNSSEC validation [14] and 0x20 encoding [36]. Regarding resource consuming attacks, we assume that the attacker is on-path. Nonetheless, these on-path attackers have limited packet operating capabilities and can only control their own domains to send responses like normal users. In addition, to ensure that malformed packets are delivered to the target resolvers before legitimate responses, we assume that the attacker can generate response packets from a neighboring host. Later, we will show that the TuDoor attack is highly efficient.

3.2. Attack Workflow

In general, in the TuDoor attack, an adversary starts by initiating a DNS query towards the victim resolver (step ①), followed by the injection of malformed packets (step ②). Finally, based on the triggered logic vulnerabilities of DNS responses pre-processing (step ③), the attacker launches one of the following three TuDoor attacks below.

DNSPoisoning: DNS cache poisoning attack. The vulnerability V_{CP} opens up a covert side channel, which enables an attacker to identify the source port for domain name resolution without any guesswork. Once the attacker has pinpointed the source port, he or she can easily poison arbitrary domains on the target resolver by simply brute-forcing the `TXID`. This includes TLD domains like `.com` and `.net`. Our subsequent experiments demonstrate the feasibility and practicality of this attack, which affects mainstream DNS software such as Microsoft DNS and can be carried out in less than one second.

DNSDoS: DNS denial-of-service attack. To exploit the vulnerability V_{DS}, an attacker must send malicious DNS response packets to the target resolver prior to the arrival of the legal packets. These packets cover all source ports of the target resolver. Due to the vulnerability in DNS response pre-processing, software such as PowerDNS prematurely terminates the normal resolution process that employs a hit source port. As a result, the target resolver would consider the remote authoritative nameserver to be unavailable, leading to DoS for itself, its clients, and future queries.

DNSConsuming: DNS resource consuming attack. Typically, to prevent resource consuming, a fixed threshold controls the retransmission of DNS queries. However, the

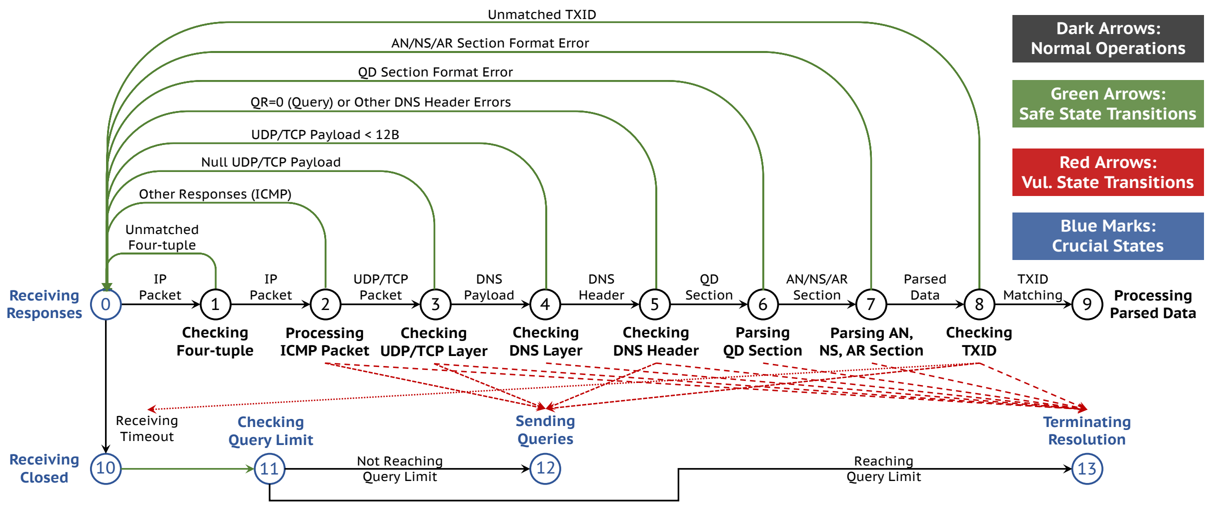


Figure 6. General state machine model of DNS response pre-processing (Except for the red dotted arrows).

vulnerability V_{RC} allows on-path attackers to repeatedly send malformed packets to the target resolver from their authoritative nameservers, thereby evading the retransmission limit and exhausting the resolver's resolution resources, including CPU and UDP source ports. This type of vulnerability affects software such as BIND, Unbound, and Knot Resolver.

4. Systematic Analysis of DNS Response Pre-processing

According to DNS RFC documents [23], [81], [82], the DNS resolution procedure can be divided into three components: the "client-resolver" part that accepts requests from clients and responds to them; the "resolver-server" part, which sends queries from resolvers and processes server responses; and the "internal operation" that processes data and cache. Although the client-resolver and internal operation parts could be exploited to conduct various attacks, such as DDoS [16], [66], [96], [97] and cache injection [57], [62], [69], the most notorious attacking target is still the resolver-server side, whereby attackers use numerous newly emerging techniques to perform the influential cache poisoning [5], [37], [50], [51], [52], [55], [56], [58], [61], [71], [74], [75], [102], [104], [124] and DoS attacks [3], [4], [24], [78], [83]. However, none of these studies provided an in-depth analysis of the resolver-server component to determine possible attack vectors, especially for the DNS response pre-processing component, which is our primary area of focus.

In this section, we first present the generic and common workflow of DNS response pre-processing by systematically reviewing DNS-related RFCs [23], [81], [82] and "reverse-engineering" 28 DNS software. Table 1 shows the DNS software that we analyze, encompassing all DNS resolver roles and mainstream software implementations (of latest versions): 8 recursive resolvers, 10 DNS forwarders, 6 stub resolvers, and 4 DNS programming libraries.

Considering complex and inconsistent DNS software implementations [71], [103], we determine using a state machine to present a comprehensive response pre-processing logic similar to [38] analyzing TLS, and leveraging it to discover vulnerabilities. Unlike [38], which employs state machine learning, we primarily construct state machines through line-by-line source code inspection and GDB debugging to ensure their completeness. Due to the relative simplicity of DNS response pre-processing, we can manually enumerate all states. Additionally, we combine states showing similar behaviors and summarize a general model.

Through identified states, we design various malformed packets to troubleshoot each software and monitor their runtime behaviors and logs. Besides, we read official documents for closed-source software like Microsoft DNS and MacOS and test them using malformed packets constructed from open-source software. The total core source code has ∼1.1M lines, and it took us 4 to 5 weeks to review and test them.

After that, we detail response pre-processing differences between each software. In the end, we summarize newly discovered logic vulnerabilities that attackers could exploit to launch various attacks, such as cache poisoning, denial-of-service (DoS), and resource-consuming attacks. *These vulnerabilities enable attackers to launch the fastest-ever cache poisoning or DoS attacks within a second and bypass the query limit, which are distinct from previous attacks.*

4.1. Generic Workflow of DNS Response Pre-processing

As described in Section 2.2, regarding a DNS response packet, all resolvers must receive it via a network socket, parse it based on each network layer, check the format, and then handle the parsed DNS data as per the specifications [23], [44]. We refer to the entire procedures before processing parsed DNS data as *DNS response pre-processing* and present its generic workflow in Figure 6. The whole pre-processing component operates as a finite-state machine, depending on the input and output result. The resolver transitions between various states and stops until it obtains a valid response or the resolution terminates.

State ⓪: receiving responses. After delivering the resolver's query transformed from clients' requests or generated internally to upstream servers, the resolver enters the receiving response state, where it awaits incoming packets.

State ⓪→① & ①→⓪. Upon the arrival of each incoming IP packet, the network kernel scrutinizes its four-tuple (source address, destination address, source port, and destination port) and designates the packet to a pre-existing socket belonging to the API caller (i.e., the DNS resolver). Any packet that consists of an unmatched four-tuple will be rejected (goto state ⓪, same as below).

State ①→② & ②→⓪. When the network kernel encounters an ICMP error message, it notifies the DNS application with an error return code and leaves application to decide on the subsequent action. Most DNS software will ignore the message and proceed to state ⓪. In such situations, the four-tuple used in state ⓪→① is embedded in the ICMP payload, as we discussed in Section 2.3.

State ②→③ & ③→④ & ③→⓪ & ④→⓪. If the UDP/TCP payload is either empty or less than 12 bytes (the DNS header's length), the packet must be discarded and the resolver should wait for more promising responses.

State ④→⑤ & ⑤→⓪. For a valid-length DNS header, resolvers must check the `QR` flag and reject the query packet whose `QR` field is 0 when receiving responses.

State ⑤→⑥ & ⑥→⑦ & ⑥→⓪ & ⑦→⓪. For each DNS response packet whose `QR` field is 1, resolvers should parse QD, AN, NS, and AR sections in accordance with their respective formats depicted in Figure 3 and exclude malformed sections.

State ⑦→⑧ & ⑧→⓪ & ⑧→⑨. The final stage of pre-processing involves TXID validation. Responses with a wrong TXID shall be discarded, as they could potentially be attack packets. In contrast, a response with a correct TXID signifies the completion of DNS response pre-processing and the initiation of data processing.

State ⓪→⑩ & ⑩→⑪. As mentioned in Section 2.2, resolvers will await a short period to receive a promising response. They will close the receiving socket if no valid response is received within the specified time frame.

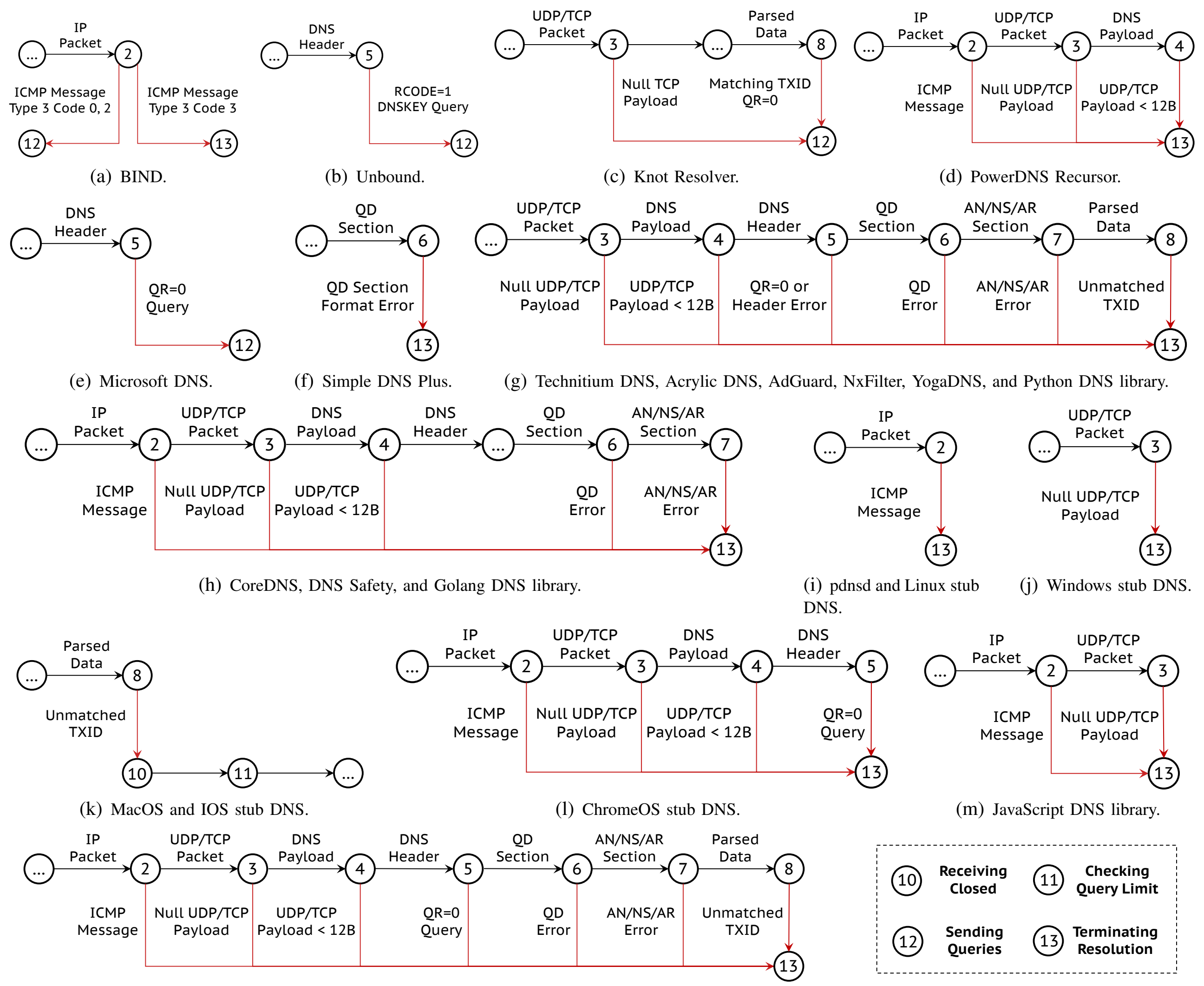
State ⑪→⑫ & ⑪→⑬. Due to the DNS retransmission mechanism (explained in Section 2.2), the resolver will check its internal query limit to determine whether to re-send new queries (if the limit has not been reached) or terminate the current resolution process (if the limit has been reached or if there is no limit checking operation).

4.2. Software Implementations of DNS Response Pre-processing

Here, we summarize the DNS response pre-processing implementation differences from the generic workflow for each analyzed software and illustrate their specific state transitions in Figure 7. We discover that, for the majority of state transitions, every software uses a similar processing logic as Figure 6. Detail-wise, most software excluding Java (Figure 7(n)) attempt to transition to state ⓪ to continue receiving promising responses from at least one of the state ① - ⑧. It demonstrates that *all developers have adhered to the practice that resolvers should await a time frame for any valid response.* However, they implement different operations for several specific response processing states, such as state ② (ICMP error messages) and ④ & ⑤ (invalid DNS headers). In summary, all state transitions denoted by red dotted arrows in Figure 6 are vulnerable. Specifically, each software employs different programming techniques to design the receiving process (same to UDP and TCP).

Recursive resolvers tend to perform similarly to the generic workflow and have fewer state transition variations than the others. Only MaraDNS has no differences with Figure 6.

- **BIND** uses `dns_dispatch_add` [19] to register receiving functions and processes responses with dispatchers. BIND employs `udp_recv` [21] for receiving UDP packets and calls `udp_dispatch_getnext` [20] for the next one if the packet is invalid. If BIND receives an ICMP error message in `udp_recv`, it will cancel the receiving process and transition into one of two states: state ⑫ (type 3 and code 0/2) or state ⑬ (type 3 and code 3).
- **Unbound** employs `iter_operate` [115] to handle every incoming response that is considered as an event. The callback function for a response receiving event is `outnet_udp_cb` [116]. If the packet is malformed, Unbound will call `comm_point_udp_callback` [114] at most 100 times to continue reading the next packet. For a `DNSKEY` query, if the RCODE is 1 (`FormErr`), Unbound will send new queries immediately.
- **Knot Resolver** regards each resolution process as a worker and initiates a queue to store packets [64]. In `worker_submit` [65], differently, Knot first examines the TXID and discards those with an unmatched TXID before checking the format. However, Knot will re-send queries for the malformed TCP packets, or UDP packets with correct TXIDs but `QR`=0 (in the forwarding mode). Malformed packets received via TLS or HTTPS connections (DoT/DoH) cause the same issue as TCP.
- **PowerDNS Recursor** also utilizes the event mechanism to receive packets [93]. Responses are received in `recvfrom` [94]. PowerDNS will return an error code directly to cease resolving [95] for an ICMP error message or DNS payload less than 12 bytes.
- **Microsoft DNS** accepts the query packet (`QR`=0) on its outbound request socket when receiving answers, and issues resolver-queries for resolving new queries.
- **Simple DNS Plus** would stop the resolution process upon receiving a packet with an invalid QD section.
- **Technitium DNS** receives and processes responses in `PostProcessQueryAsync` [110]. However, it just accepts the first-arriving packet before closing the receiving socket [111]. Therefore, if the packet is malformed, Technitium just enters state ⑬.
- **MaraDNS** takes a `bigloop` function [76] to process responses with the select mechanisms until obtaining a valid answer with `get_remote_udp_packet` [77].



(a) BIND. (b) Unbound. (c) Knot Resolver. (d) PowerDNS Recursor. (e) Microsoft DNS. (f) Simple DNS Plus. (g) Technitium DNS, Acrylic DNS, AdGuard, NxFilter, YogaDNS, and Python DNS library. (h) CoreDNS, DNS Safety, and Golang DNS library. (i) pdnsd and Linux stub DNS. (j) Windows stub DNS. (k) MacOS and IOS stub DNS. (l) ChromeOS stub DNS. (m) JavaScript DNS library. (n) Java DNS library.

Figure 7. Specific state transitions of DNS response pre-processing implementations for different software (Red lines represent vulnerable state transitions).

DNS forwarders. Dnsmasq and Pi-hole function identically to the general logic, whereas other software show a variety of distinct actions.

- **Dnsmasq** employs the polling mechanism in a running loop [42] to receive the response until a legal response is returned (`reply_query` [41]) and works as Figure 6.
- **CoreDNS** implements the resolver server portion [34] but uses the Golang DNS library for outgoing queries.
- **Pi-hole** is built on Dnsmasq and behaves similarly.
- **pdnsd** will terminate resolution on ICMP messages.
- **Acrylic DNS**, **AdGuard**, **NxFilter**, and **YogaDNS** are forwarders that provide traffic filtering and work similarly to Technitium accepting malformed DNS packets.
- **DNS Safety** accepts ICMP messages but ignores responses with wrong TXIDs, similar to CoreDNS.
- **Dual DHCP DNS** will discard all malformed packets.

Stub resolvers, except for Android, implement at least one state transition path distinct from the general workflow.

- **Linux stub DNS (systemd-resolved)** also leverages the event technique (`sd_event_add_io` [107]) to receive packets with `on_dns_packet` [106]. If the packet contains an ICMP error message, the resolution will be terminated and other servers will be attempted.
- **Windows stub DNS** will terminate its resolution operation if the response has a null UDP/TCP payload.
- **MacOS and IOS stub DNS** will initiate new TCP queries no more than 2 times (after checking the query limit) [13] if the UDP response's TXID is not correct.
- **Android stub DNS** has the exactly same processing workflow in Figure 6, and the source document is [11].
- **ChromeOS stub DNS** will close the receiving connection in `DoReadResponseComplete` [30] if it encounters an ICMP error message, DNS payload smaller than 12 bytes, or DNS header with `QR`=0.

TABLE 1. THE DNS RESOLUTION MECHANISM AND RESPONSE PRE-PROCESSING IMPLEMENTATIONS OF 28 DNS SOFTWARE (24 VULNERABLE).

Resolver			Resolution		Vulnerable state transition												Vulnerability		
Role	Software	Version	Query count	Negative caching	⑧ ↓ ⑩	② ↓ ⑫	③ ↓ ⑫	⑤ ↓ ⑫	⑧ ↓ ⑫	② ↓ ⑬	③ ↓ ⑬	④ ↓ ⑬	⑤ ↓ ⑬	⑥ ↓ ⑬	⑦ ↓ ⑬	⑧ ↓ ⑬	V_{CP}	V_{DS}	V_{RC}
Recursive	BIND	9.18.14	13	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✓
	Unbound	1.17.1	9	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
	Knot	5.5.3	3	✓	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
	PowerDNS	4.8.3	1	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗
	Microsoft	2022	2	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
	Simple DNS+	9.1.111	3	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗
	Technitium	11.0.2	6	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✗
	MaraDNS	3.5.0036	6	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Forwarder	Dnsmasq	2.89	1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	CoreDNS	1.10.1	3	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✓	✓	✗	✓	✓	✗
	Pi-hole	5.17.1	1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	pdnsd	1.2.9	1	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗
	Acrylic DNS	2.1.1	1	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✗
	AdGuard	7.14	2	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✗
	DNS Safety	1.0	1	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✓	✓	✗	✓	✓	✗
	Dual DHCP DNS	8.00RC	1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
	NxFilter	4.6.7.6	3	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✗
	YogaDNS	1.37	1	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✗
Stub	Linux	253	6	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗
	Windows	2023	5	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗
	MacOS	13.2.1	6	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
	IOS	16.3.1	6	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
	Android	13	4	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	ChromeOS	111.x	5	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗	✓	✗
Library	Python	2.3.0	1	-	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✗	✓	✗
	Golang	2023	1	-	✗	✗	✗	✗	✗	✓	✓	✓	✗	✓	✓	✗	✗	✓	✗
	JavaScript	19.8.1	1	-	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✓	✗
	Java	3.5.2	1	-	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗

'-': Not applicable due to no caching. ✓: Yes. ✗: No. ✓: Vulnerable. ✗: Not vulnerable.

DNS libraries operate most differently from Figure 6.

- **Python DNS library** performs the same operations using `receive_udp` [43] with Technitium DNS.
- **Golang DNS library** runs a loop [45] to receive packets with a matching TXID. If the packet is malformed, the loop will return and the resolution will stop.
- **JavaScript DNS library** accepts an ICMP error message and a null UDP/TCP payload as a resolution end signal and transitions to state ⑬ (see documents [85]).
- **Java DNS library** will take any ICMP error message or UDP/TCP response as a valid DNS answer and end the resolution process without retransmission in `send` [40].

4.3. Novel Logic Vulnerabilities in DNS Response Pre-processing

By conducting a comprehensive analysis of DNS response pre-processing for 28 DNS software, we explore their implementation variations and discover multiple new logic vulnerabilities in 24 of the analyzed software (as listed in the "Vulnerability" column of Table 1). We categorize these vulnerabilities into three groups based on their effects: cache poisoning (V_{CP} in 9 software), DoS (V_{DS} in 20 software), and resource consuming (V_{RC} in 3 software). Notably, V_{CP} and V_{DS} could be exploited for cache poisoning and DoS attacks in less than one second. In this section, we provide a high-level explanation of the root cause of the vulnerabilities and how they can be exploited, while leaving the details of the attack steps to Section 5.

V_{CP}: cache poisoning vulnerability. Different from other DNS software, Microsoft DNS allows a DNS query packet (`QR`=0) to be received on the socket designated for responses, as long as the packet's four-tuple matches (as shown in Figure 7(e)). In particular, this new DNS query packet can contain any query domain name. Upon receiving this query, Microsoft will initiate another recursive resolution procedure to resolve it. However, *this operation creates an effective side channel that attackers can use to locate the source port utilized by Microsoft DNS.* This is because *only the query with a matching four-tuple can be accepted, while packets received on other ports will be ignored.* In Section 5.1, we will demonstrate how attackers could exploit this vulnerability to identify the source port in less than one second (without the need to guess it) and poison arbitrary domain names, resulting in a cache poisoning attack that is significantly faster than all prior ones. Apart from this vulnerability, we find AdGuard does not verify the TXIDs of DNS responses in state ⑧, which satisfies the cache attack condition. In addition, we uncover other cache poisoning

TABLE 2. EXPERIMENT RESULTS OF THREE TYPE OF TUDOOR ATTACKS ON EXEMPLAR SOFTWARE.

TUDOOR attack	**Selected software**	**Average time cost**	**Average traffic rate**	**Success rate**
DNSPOISONING	Microsoft DNS	425ms	103Mbps	19/20
	Technitium	349ms	84Mbps	16/20
DNSDOS	BIND	241ms	35Mbps	18/20
	PowerDNS	185ms	29Mbps	20/20

TUDOOR attack	**Selected software**	**Query limit**	**# Actual query**	**Success rate**
DNSCONSUMING	BIND	13	100	-
	Unbound	9	60+	-
	Knot	3	100	-

'-': Not applicable.

vulnerabilities that are unrelated to malformed packet injection and describe them in Section 5.1. We demonstrate that attackers could exploit these vulnerabilities to poison DNS software in a brief period of time.

V_{DS}**: denial-of-service vulnerability.** In the generic workflow of DNS response pre-processing shown in Section 4.1, any malformed packet or DNS response with an unmatched TXID will be discarded, and resolvers should await a time frame for any future valid response. However, we find that 20 software fail to enter state ⓪ when receiving some malformed packets but immediately close the receiving socket and stop the resolution instead. Afterward, they return a `ServFail` reply to clients. Due to the fact that they accept the malformed packet before verifying TXIDs and never attempt to receive the next legal response, *an attacker could manufacture a small number of malformed responses to DoS the resolver by simply enumerating source ports.* Every outbound query from a source port that is hit would fail. The effect is mitigated for software that falls back to TCP, such as MacOS and IOS (state ⑧→⑩); however, if the remote server does not support TCP, they will also not obtain valid responses. In addition, we will demonstrate that negative caching contributes to our attack in Section 5.2.

V_{RC}**: resource consuming vulnerability.** As described in Section 4.2, upon receiving certain malformed packets, BIND (Figure 7(a)), Unbound (Figure 7(b)), and Knot Resolver (Figure 7(c)) would directly transition into state ⑫ and send new queries without checking the retry limit. After reviewing source code, we find query limit-checking is located after the retry section. Therefore, *attackers can compromise this vulnerability by repeatedly returning malformed packets via a malicious nameserver, which would trigger more queries and exhaust the resolver's CPU resources.*

5. Three TUDOOR Attacks

In this section, we provide detailed exploitation steps for three proposed TUDOOR attacks, including DNSPOISONING, DNSDOS, and DNSCONSUMING, and the end-to-end experiments with controlled network settings. We then compare TUDOOR with previous attacks in detail. In addition, we identify another two types of cache poisoning

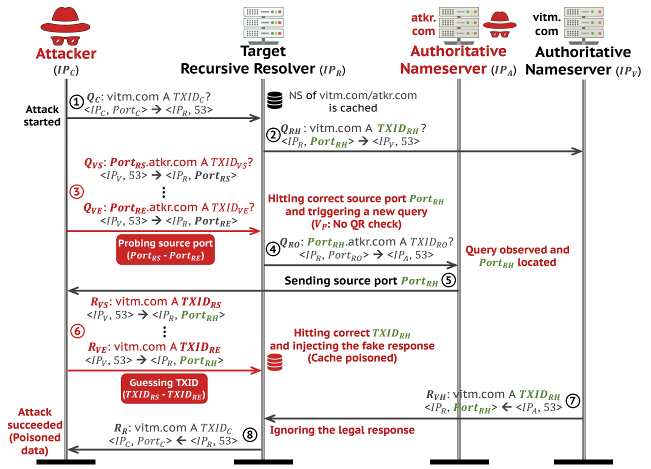

Attacker (IP_C)
Target Recursive Resolver (IP_R)
atkr. com
Authoritative Nameserver (IP_A)
vitm. com
Authoritative Nameserver (IP_V)
Attack started
① Q_C: vitm.com A $TXID_C$? <IP_C, $Port_C$> → <IP_R, 53>
NS of vitm.com/atkr.com is cached
② Q_{RH} : vitm.com A $TXID_{RH}$? <IP_R, $Port_{RH}$> → <IP_V, 53>
③ Q_{VS}: $Port_{RS}$.atkr.com A $TXID_{VS}$? <IP_V, 53> → <IP_R, $Port_{RS}$>
Q_{VE}: $Port_{RE}$.atkr.com A $TXID_{VE}$? <IP_V, 53> → <IP_R, $Port_{RE}$>
Probing source port ($Port_{RS}$ - $Port_{RE}$)
Hitting correct source port $Port_{RH}$ and triggering a new query (V_P: No QR check)
④ Q_{RO}: $Port_{RH}$.atkr.com A $TXID_{RO}$? <IP_R, $Port_{RO}$> → <IP_A, 53>
Query observed and $Port_{RH}$ located
Sending source port $Port_{RH}$ ⑤
⑥ R_{VS}: vitm.com A $TXID_{RS}$ <IP_V, 53> → <IP_R, $Port_{RH}$>
R_{VE}: vitm.com A $TXID_{RE}$ <IP_V, 53> → <IP_R, $Port_{RH}$>
Guessing TXID ($TXID_{RS}$ - $TXID_{RE}$)
Hitting correct $TXID_{RH}$ and injecting the fake response (Cache poisoned)
R_{VH}: vitm.com A $TXID_{RH}$ <IP_R, $Port_{RH}$> ← <IP_A, 53> ⑦
Ignoring the legal response
Attack succeeded (Poisoned data)
R_R: vitm.com A $TXID_C$ <IP_C, $Port_C$> ← <IP_R, 53> ⑧

Figure 8. Attack steps of DNSPOISONING.

vulnerabilities: V_{CPR} and V_{CPA} that could be used to poison DNS software. Here, we describe the experiment setup.

Experiment setup. Since almost all software is vulnerable to TUDOOR (shown in Table 1), we select a subset of these software (listed in Table 2) to conduct attack experiments based on the popularity [72] and our identified distribution share in Section 6.3. We install BIND, Unbound, Knot, and PowerDNS on a server running Ubuntu 20.04 respectively, while Microsoft DNS and Technitium are installed on a Windows Server 2022. The victim's and the attacker's DNS nameservers[1], and the attack program (based on Golang) are hosted on separate Ubuntu Server 20.04 machines. All machines are linked to a local network (1000Mbps bandwidth) and assigned with public IPs. Moreover, we use IP_C (attacker), IP_R (target), IP_A (attacker's nameserver), and IP_V (victim's nameserver) to denote the IPs of different roles.

5.1. DNSPOISONING Attack (Cache Poisoning)

DNS cache poisoning is one of the most potent attacks because it can affect all clients if an attacker successfully inject forged DNS responses. However, even the state-of-the-art attack method like [74], [75] will take an average of 80s to 504s, which is also ineffective to Microsoft DNS[2]. Using the vulnerability V_{CP} (a new side channel), we prove that an DNSPOISONING attacker could inject arbitrary fake responses into vulnerable resolvers (e.g., Microsoft DNS) in less than one second. In addition, we identify another two types of vulnerabilities: *(i)* V_{CPR} of source port or TXID randomization that could be employed to poison Technitium (Acrylic DNS, AdGuard, DNS Safety, Dual DHCP DNS, NxFilter, and YogaDNS) within just one second; *(ii)* V_{CPA} of query aggregation used for poisoning CoreDNS.

V_{CPR}**: source port or TXID randomization vulnerability.** Technitium DNS implements a sequentially increasing

1. We registered two own domains for testing. For anonymity, we use `vitm.com` (for the victim) and `atkr.com` (for the attacker) in the following.
2. Microsoft DNS opens and listens on all 2,500 source ports simultaneously for resolution, which cannot be inferred through side channels.

source port generation algorithm while operating on Windows. After examining the source code, we identify the underlying culprit as the C Sharp socket library used by Technitium [112]. Acrylic DNS and AdGuard also generate the source port sequentially, while YogaDNS sequentially selects both the source port and TXID. Dual DHCP DNS uses a fixed source port after starting. Besides, DNS Safety, Dual DHCP DNS, and NxFilter utilize the same TXID value extracted from the client's DNS query. Using V_{CPR}, an adversary could pinpoint the source port or TXID within a second and poison these software.

V_{CPA}: query aggregation vulnerability. To defend against the "birthday paradox" [103], query aggregation is proposed to ensure that only one outgoing query is issued to resolve the same query domain name. However, CoreDNS permits multiple simultaneous queries for a given query name, rendering it vulnerable to the "birthday paradox"-based cache poisoning attacks. We save this for future work.

Attack design. We show the DNSPOISONING attack steps in Figure 8. To begin with, an attacker initiates a DNS query to the target recursive resolver for the victim domain `vitm.com` (step ①). After receiving the query, the target resolver starts the resolution process by sending a query to `vitm.com`'s nameserver. The query four-tuple and TXID are (IP_R, $Port_{RH} \rightarrow IP_V$, 53) and $TXID_{RH}$ (step ②).

The initial phase of cache injection for the attacker is to determine the source port $Port_{RH}$ through the following two operations. First, by spoofing the source IP address IP_V, the attacker sends a number of DNS queries for his or her controlled domain `atkr.com` towards all possible source port numbers $Port_{RX}$ (from $Port_{RS}$ to $Port_{RE}$) of the target resolver, using the four-tuple (IP_V, $53 \rightarrow IP_R$, $Port_{RX}$) in step ③. Second, the query domain name is encoded with each probed source port number for observing and differentiating the source ports. For example, if the probing query is sent to $Port_{RX}$, its domain name is set as $Port_{RX}$`.atkr.com`. As soon as the probing query hits the correct source port $Port_{RH}$, the target resolver will accept it and deliver another query for resolution (step ④). The attacker could then observe this new query on the controlled nameserver and "steal" the correct source port number (step ⑤).

The next phase involves guessing TXIDs. The attacker can brute-force 65,536 TXID values by injecting fake responses (step ⑥) with the correct four-tuple (IP_V, $53 \rightarrow IP_R$, $Port_{RH}$) before the target resolver receives a legitimate response (step ⑦). The target resolver will then accept the forged response with a correct $TXID_{RH}$ and cache it, which signals the success of the attack (step ⑧).

For attacking a resolver running Technitium DNS, unlike the above procedures, the attacker should initiate the process by sending a query for `atkr.com` to obtain the source port $Port_P$ on the nameserver, and then query `vitm.com`. The target resolver will use $Port_P + 1$ as the source port for resolution. The attacker can simply brute-force all 65,536 possible TXID values to inject forged responses with the four-tuple (IP_V, $53 \rightarrow IP_R$, $Port_P + 1$).

End-to-end attack. We ran the DNSPOISONING attack 20 times against Microsoft DNS and Technitium DNS using two different attack programs implementing the aforementioned techniques. To increase query latency, we delayed one second before returning the legal response of `vitm.com`. Besides, we find that Microsoft DNS only utilizes *2,500* source ports for resolution, further reducing the probing space. Specifically, all these 2,500 source ports are in the open state, making port inference methods in [74], [75] inoperable. The experiment results are shown in Table 2. The respective attack success rate of Microsoft DNS and Technitium DNS are 19/20 and 16/20. On average, it takes only *425ms* (*103Mbps*) and *349ms* (*84Mbps*) to attack Microsoft DNS and Technitium DNS, respectively. Compared to prior attacks that took 80 to 504 seconds [74], [75], DNSPOISONING is approximately 200 to 1,000 times faster.

Discussion. First, to poison an entire SLD or TLD, we could conduct DNSPOISONING for NS queries issued by recursive resolvers. Since DNSPOISONING can inject arbitrary responses within one second, we can seize the entire zone and all of its domains by forging NS records. Second, to ensure the fake responses are received before the legitimate ones, one option is to send the packets from a nearby machine or launch the denial-of-service attack on the nameserver, preventing the nameserver from returning a response by triggering the rate-limiting mechanism [74]. Third, we need to consider the architecture of resolvers and nameservers:

Multiple nameservers. Many domains are configured with multiple authoritative nameserver IP addresses. We analyzed the latest zone files of `.com` and `.net` downloaded from ICANN's CZDS service [53] and found domains under them have a median of 4 nameserver IPs. Since our attack only requires a maximum of 65,536 packets (or even 2,500) to identify the source port, attackers could simultaneously spoof these IPs by sending 65,536×4 packets.

Multiple backend IPs. Public DNS services usually have multiple backend IPs, extending the injection space. However, the selection of backend IPs is typically based on factors such as geolocation and network performance. For example, via experiments from a specific geolocation, we found 17 out of 18 affected public DNS services (tested in Section 6.2) have only 1-2 backend IPs. Thus, attackers only need to focus on spoofing a few IPs at once.

5.2. DNSDoS Attack (Denial-of-Service)

In the *DNSDoS* attack, *an attacker could DoS all DNS resolver roles, including the stub resolvers, DNS forwarders, and recursive resolvers, within just a single second.* As an example, we present the attack procedure for the recursive resolver, which suffers from significant DoS effects on all its clients. The fundamental idea is that an attacker can brute-force the source port of DNS queries and inject malformed packets. By utilizing the vulnerability V_{DS}, the attacker can deceive the target resolver into terminating the resolution process and returning invalid answers (`ServFail`) to clients.

Attack design. We depict the attack steps of DNSDOS in Figure 9. An attacker starts by sending a query for the victim domain `vitm.com` to the target resolver (IP_R, step ①). The target resolver then requests the upstream server

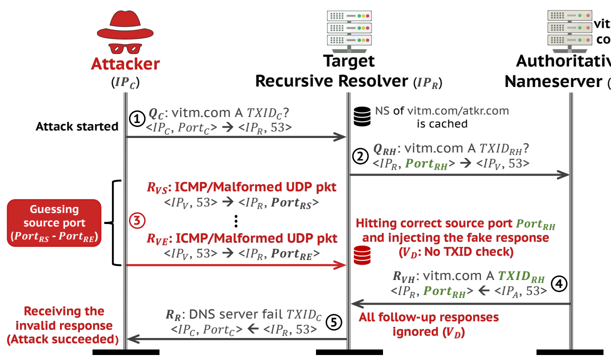

Attacker
(IP_C)
Target
Recursive Resolver (IP_R)
vitm.com
Authoritative
Nameserver (IP_V)
Attack started
① Q_C: vitm.com A $TXID_C$? <IP_C, $Port_C$> → <IP_R, 53>
NS of vitm.com/atkr.com is cached
② Q_{RH}: vitm.com A $TXID_{RH}$? <IP_R, $Port_{RH}$> → <IP_V, 53>
Guessing source port ($Port_{RS}$ - $Port_{RE}$)
③
R_{VS}: ICMP/Malformed UDP pkt <IP_V, 53> → <IP_R, $Port_{RS}$>
R_{VE}: ICMP/Malformed UDP pkt <IP_V, 53> → <IP_R, $Port_{RE}$>
Hitting correct source port $Port_{RH}$ and injecting the fake response
(V_D: No TXID check)
R_{VH}: vitm.com A $TXID_{RH}$ <IP_R, $Port_{RH}$> ← <IP_A, 53> ④
All follow-up responses ignored (V_D)
Receiving the invalid response (Attack succeeded)
R_R: DNS server fail $TXID_C$ <IP_C, $Port_C$> ← <IP_R, 53> ⑤

Figure 9. Attack steps of DNSDoS.

(e.g., vitm.com's nameserver) for responses with the four-tuple (IP_R, $Port_{RH}$ $\rightarrow$ IP_V, 53). Before a legal response is returned, the attacker must determine the source port and inject malformed packets. To achieve this purpose, the attacker can spoof the IP address IP_V and brute-force all 65,536 port numbers (step ③). Once $Port_{RH}$ is hit, the target resolver will accept the malformed response without verifying the TXID and terminate the resolution. Then, all further replies will be disregarded (step ④). All clients who query vitm.com will receive invalid ServFail responses, causing the follow-up connection to fail (step ⑤).

End-to-end attack. In the experiment, we select BIND and PowerDNS as the target resolvers and attack them individually using ICMP error messages (type 3 and code 3) and null UDP payload packets. Particularly, the attacker does not need to forge the source IPs for ICMP messages since the OS kernel merely checks the four-tuple embedded in ICMP payloads. Similar to the DNSPoisoning attack, our nameserver will temporarily hold the response for one second before sending it to the resolver. As listed in Table 2, we execute DNSDoS 20 times with a success rate of 18/20 for BIND and 20/20 PowerDNS. The average attack time taken (traffic rate) for BIND and PowerDNS is just *241ms* (*35Mbps*) and *185ms* (*29Mbps*), respectively. In contrast, traditional DDoS attacks (e.g., [15], [31], [113]) against resolvers require at least 1Gbps of network bandwidth.

Discussion. There are some practical attack considerations.

Attacking NS queries. To DoS the target resolver, an attacker could inject malformed packets for either NS queries or other queries to fetch the final answer. After receiving malformed packets, the target resolver will determine the remote server is inaccessible. Consequently, if the NS query targeting the root or TLD nameserver fails, all subsequent queries under these zones like .com and .net fail as well.

Retransmission and negative caching. Even though the majority of evaluated software provides a retransmission mechanism, none of them resend queries after receiving our malformed packet; instead, they close the receiving socket and return a ServFail response to clients. Besides, 6 of them employ the negative caching method (the "Negative caching" column in Table 1), in which resolvers store the ServFail response for a period of time and do not transmit further requests for the same remote server. For instance,

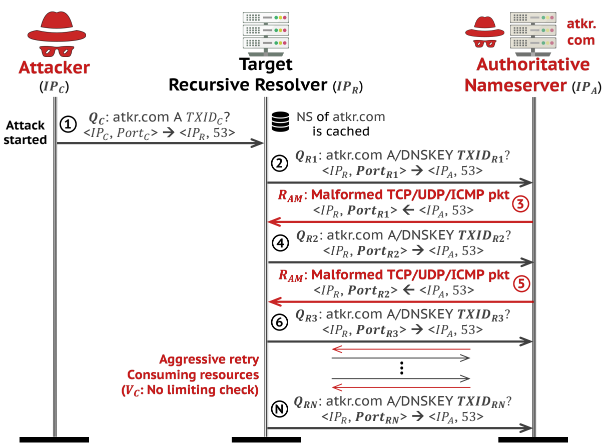

Attacker
(IP_C)
Target
Recursive Resolver (IP_R)
atkr.com
Authoritative
Nameserver (IP_A)
Attack started
① Q_C: atkr.com A $TXID_C$? <IP_C, $Port_C$> → <IP_R, 53>
NS of atkr.com is cached
② Q_{R1}: atkr.com A/DNSKEY $TXID_{R1}$? <IP_R, $Port_{R1}$> → <IP_A, 53>
R_{AM}: Malformed TCP/UDP/ICMP pkt <IP_R, $Port_{R1}$> ← <IP_A, 53> ③
④ Q_{R2}: atkr.com A/DNSKEY $TXID_{R2}$? <IP_R, $Port_{R2}$> → <IP_A, 53>
R_{AM}: Malformed TCP/UDP/ICMP pkt <IP_R, $Port_{R2}$> ← <IP_A, 53> ⑤
⑥ Q_{R3}: atkr.com A/DNSKEY $TXID_{R3}$? <IP_R, $Port_{R3}$> → <IP_A, 53>
Aggressive retry
Consuming resources
(V_C: No limiting check)
Ⓝ Q_{RN}: atkr.com A/DNSKEY $TXID_{RN}$? <IP_R, $Port_{RN}$> → <IP_A, 53>

Figure 10. Attack steps of DNSConsuming.

the default negative cache TTL of Technitium is 300s [109]. PowerDNS will mark the resolver as unavailable for a negative cache TTL time, including the TLD nameservers. These behaviors make it easier for an attacker to send malformed packets between the negative cache TTL interval, during which the attacker does not need to send packets.

Multiple nameservers and backend IPs. To handle multiple nameserver IPs and resolver backend IPs, we essentially use the same techniques as shown in Section 5.1. Besides, because DNSDoS only needs to brute-force 65,536 source port numbers, it is possible for attackers to spoof all the IPs.

5.3. DNSConsuming Attack (Resource Consuming)

The DNSConsuming attack aims to deplete the target resolver's resolution resources by manipulating it to issue more queries. *Leveraging the vulnerability V_{RC}, an attacker could bypasses the query retransmission limit.*

Attack design. Figure 10 shows the detailed DNSConsuming exploitation steps. To launch DNSConsuming, the attacker should be on-path and operate a domain nameserver (e.g., atkr.com) that produces malformed packets. First, the attacker sends a query towards the target recursive resolver for atkr.com (step ①). Second, the target resolver performs recursive resolution from the root to .com in order to obtain atkr.com's nameserver (same as below), then queries the nameserver for answers (step ②). Third, the attacker continues to return carefully crafted packets (step ③) to the correct source port, whereas the target resolver keeps sending new queries to the malicious nameserver (step ④) until it reaches a total resolution limit.

End-to-end attack. We conduct DNSConsuming against BIND, Unbound (with the DNSSEC option enabled), and Knot Resolver with the following different queries (responses): A query (ICMP type 3 and code 0), A query (null TCP payload), and DNSKEY query (RCODE FormErr). As shown in Table 2, DNSConsuming can cause BIND, Unbound, and Knot Resolver to query 100, 60+, and 100 times larger than the retransmission limit of 13, 9, and 3, respectively, which is employed to mitigate the previous attacks [3], [83]. With Valgrind [117], we show DNSConsuming makes BIND and Knot cost 35× and 30× more CPU instructions, respectively. Since the outcome is deterministic, we just test DNSConsuming once for them.

Discussion. An attacker might boost resource consumption in two ways. The first method is to immediately return the malformed packet to overwhelm the CPU. The alternative would be to increase the response packet size. We observe that BIND, Unbound, and Knot Resolver accept packets larger than *4,096* bytes (with a maximum limit of *65,535* bytes) through fragmentation. This increases the potential resource consumption through packet fragmentation [51].

5.4. Comparison with Prior Attacks

The TuDoor attack is novel because it exploits three new logic vulnerabilities (and three new corresponding techniques) found by systematic analysis of DNS response pre-processing. Several previous studies have examined specific aspects of DNS response processing, such as the analysis of cache poisoning [35], amplification DDoS [66], bailiwick checking [71], [103], and cache injection [62]. However, none of them provide a thorough investigation of the entire DNS response pre-processing procedure, as we present in this paper. Although the attack vectors are known, TuDoor differs significantly from existing works in both vulnerability discovery and attack techniques (impacts).

For vulnerability discovery, we use state machines to analyze DNS response pre-processing that has not been thoroughly studied and identify new vulnerabilities, particularly malformed responses. Prior work only studied the processing of legal format [3], [4], [74], [75] or legal with escaped characters encoding (not malformed) [55], [56] responses.

With regard to detailed attack techniques (impacts), three attacks work separately compared with prior attacks:

- DNSPoisoning uses malformed responses (`QR`=0) to pinpoint source-port directly by forcing resolvers to send queries for the attacker's domain without requiring probability-based divide-and-conquer port guessing employed by [74], [75]. Therefore, DNSPoisoning could poison arbitrary domains (including TLDs) in less than 1s, which is about 200 to 1,000 times faster than previous attacks [5], [74], [75]. DNS cache poisoning will benefit from a fast attack, because forged responses must be returned before the legitimate ones. Otherwise, attackers have to delay legal responses like muting nameservers [74], [75], which affects normal resolution and makes it difficult to poison TLDs (too many nameserver IPs and good performance). Besides, our experiments revealed [74], [75] cannot function on Microsoft DNS (see Section 5.1). Moreover, when injecting forged responses, previous Kaminsky-style attacks [58], [74], [75] all send A queries and include fake NS records in the authoritative section because they must try many times with different random subdomains of an SLD in each round based on the guessing probability. Once the attack succeeds, the phony NS records will be cached, and all subsequent queries under that SLD can be hijacked. This type of response can also enhance the effectiveness of DNSPoisoning for poisoning NS records of SLDs. Furthermore, by just sending an A or NS query, DNSPoisoning can directly inject forged A or NS records carried in the answer section for both SLDs and TLDs. Specifically, to inject fake records, DNSPoisoning only needs to send a maximum of 2^{16} malformed response (`QR`=0) packets to pinpoint the source port and another 2^{16} responses to enumerate the 16-bit TXID field. However, in the worst-case scenario, Kaminsky-style attacks have to send 2^{32} responses for each attack query due to 16-bit port and 16-bit TXID space. Comparing the total number of packets required by attackers to have one attack instance to be successful, DNSPoisoning requires roughly 2^{17} packets, which is more effective than Kaminsky-style attacks which require 2^{32} packets in the worst case.
- DNSDoS forces resolvers to terminate resolution via a small number of targeted malformed packets and to cache negative responses for a period of time, affecting future queries (see Section 5.2). Thus, DNSDoS is less expensive and stealthier. In contrast, traditional DoS attacks (including low-rate DoS) [31], [48], [68], [113] must continuously send large volumes of traffic to occupy network bandwidth and cause packet loss. Although one type of our malformed packets ICMP, has been exploited to DoS TCP connections [46], there is little discussion about UDP and DNS in this regard.
- DNSConsuming circumvents query-limits and renders resolvers to consume more CPU resources like [4], since one attack query could trigger such as 100 resolver queries (see Section 5.3). Although DNSConsuming requires on-path attackers, these attackers have limited packet operating capabilities and can only control their own domains to send responses and trigger vulnerabilities, in contrast to traditional on-path packet interception that can observe and modify all packets except those sent to their own nameservers [73].

6. Vulnerable Resolvers in the Wild

In this section, we provide a comprehensive TuDoor exploiting picture in the wild by evaluating 16 Wi-Fi routers, 6 router OSes, 42 public DNS services, and 1.8M open DNS resolvers. We demonstrate via testing that a substantial portion of the resolver population is vulnerable to TuDoor.

6.1. Wi-Fi Routers and Router-OSes

Router and OS list. Through investigation [100], [120], we purchased 16 popular Wi-Fi routers and installed 6 router OSes on our machines, as shown in Table 3.

Testing. We connected our client machine to these routers (OSes) and configured their upstream DNS servers to point straight to our authoritative nameserver. Then, we verified their state transitions from on-path according to Section 4.

TABLE 3. 16 Wi-Fi routers and 6 router OSes testing results.

Type	Vendor	Version	Cache poisoning
Wi-Fi router	ASUS RT-AC66U	9.0.0.4.x	✗
	D-Link DIR-816	17.01.11	✗
	FAST FAC1200R	3.0.x	✗
	FiberHome R1 AX1800	RP0102	✗
	H3C Magic NX15	100R008	✗
	HUAWEI AX3 Pro	3.0.3.213	✗
	LINKSYS EA8100	2.0.4.x	✗
	MERCURY D191G	2.0.2.x	✓
	Netcore N30	1.0.5.14	✗
	Nighthawk RAX70	1.0.14.134	✓
	Skyworth WR9651X	1.0.0	✓
	Tenda AX2 Pro	16.03.29.36	✓
	TOTOLINK X5000R	1.0.0.x	✓
	TP-LINK TL-XDR3230	1.0.22	✗
	Redmi AX3000	1.0.46	✓
	ZTE ZXHN E2631	1.0.0.x	✗
Router OS	DD-WRT	r52189	✗
	Gargoyle	1.13.0	✗
	iKuai OS	3.7.0	✓
	libreCMC	1.5.12	✗
	OpenWrt	22.03.3	✗
	RouterOS	7.8	✗

Ordered according to the alphabet. ✓: Vulnerable. ✗: Not vulnerable.

Results. After testing, we found all 16 routers and 6 OSes function identically as described in Figure 6, because they are built on OpenWrt (Dnsmasq) [87] which is immune to TuDoor. However, we identified other vulnerabilities related to the source port randomization and query aggregation in 7 routers (OSes) (the "Cache poisoning" column in Table 3). *An attacker might poison these 7 routers (OSes) by brute-forcing the source port or exploiting the "birthday paradox" [103].* This attack will be left for future work.

6.2. Public DNS Services

Public DNS service list. According to the statistics from APNIC [12], we selected 42 widely-used public DNS services and exemplar IP addresses in Table 6.

Testing. We examined the response pre-processing state transitions of these 42 public DNS services based on Section 4 utilizing our own domain. Multiple tests were conducted on each of them to guarantee their validity. Additionally, we evaluated their negative caching behaviors.

Results. The overall testing results of 41 public DNS services are shown in Table 6, 18 of which are vulnerable. In detail, *one public DNS service (114DNS) is vulnerable to* DNSPoisoning since it operates similarly to Microsoft DNS, which allows DNS queries when receiving the response (state ⑤→⑫). However, 114DNS only accepts a query with the same query domain name as the one sent on the receiving socket. In order to poison 114DNS, an attacker could exploit this vulnerability to occupy as many as source ports and brute-force remaining source ports like [5]. *Other 17 public DNS services are vulnerable to* DNSDoS, and 10 of them employ negative caching.

TABLE 4. Statistics of vulnerable open resolvers.

Type	Resolver number and percentage	
Collected	Alive on 03/10/2023	**1,837,442 (100.0%)**
Software identified	Microsoft DNS	205,984 (11.2%)
	BIND	54,813 (3.0%)
	Unbound	12,765 (0.7%)
	PowerDNS Recursor	12,750 (0.7%)
	Knot Resolver	45 (0.0%)
	CoreDNS	8 (0.0%)
Vulnerable	DNSPoisoning	205,984 (11.2%)
	DNSDoS	216,317 (11.8%)
	DNSConsuming	67,623 (3.7%)
	TuDoor	**423,652 (23.1%)**

TABLE 5. Top 10 region and ASN distribution of collected open resolvers.

Region	#	%	AS	#	%
China	658,312	35.8%	ASN 4134	247,572	13.5%
India	141,668	7.7%	ASN 4837	126,485	6.9%
United States	135,201	7.4%	ASN 4538	63,151	3.4%
South Korea	84,908	4.6%	ASN 24560	63,062	3.4%
Russia	79,978	4.4%	ASN 17488	54,148	2.9%
Indonesia	66,147	3.6%	ASN 4847	47,276	2.6%
Brazil	52,609	2.9%	ASN 4766	39,880	2.2%
Bangladesh	41,073	2.2%	ASN 4808	30,784	1.7%
Iran	38,739	2.1%	ASN 58224	27,598	1.5%
Japan	26,018	1.4%	ASN 3462	22,900	1.2%
Total 227 regions			Total 24,941 ASes		

6.3. Open DNS Resolvers

Open DNS resolver list. Since the open DNS resolver is volatile [101], we aim to acquire the most recent "screenshot" of the Internet by scanning the IPv4 network for UDP port 53 on our controlled domain with `XMap` [70]. We disregard these returning wrong results. Via scanning on March 10, 2023, we discovered over 1.8 million open DNS resolvers containing DNS forwarders and recursive resolvers as shown in Table 4. Specifically, 1.8M open DNS resolvers are associated with 227 regions and 24,941 ASes (autonomous systems). The top 10 regions and ASes of collected resolvers are listed in Table 5, with China, India, and the United States ranking highest. We acknowledge that the resolver distribution may introduce a bias in which a few regions account for the majority; however, we aim to demonstrate the impact rather than analyze their distribution.

Testing. To analyze 1.8M resolvers' behaviors against TuDoor, we cannot return malformed packets due to ethical reasons; instead, we take the following approaches. *(i)* We identify their software version first using the `version.bind` query [18] and then `fpdns` [39] (a common fingerprinting tool in the DNS community). As all vulnerable software are exploitable in the latest version, we can determine whether a resolver is vulnerable by the software brand. *(ii)* We query these resolvers by first returning ICMP error messages and then normal responses to verify the vulnerability V_{DS}. In the end, all confirmed resolvers are just the lower bound.

TABLE 6. 42 PUBLIC DNS SERVICES TESTING RESULTS (18 VULNERABLE VENDORS).

Public DNS service		Resolution		Vulnerable state transition												Vulnerability		
Vendor	Exemplar IP address	Query count	Negative caching	⑧ ↓ ⑩	② ↓ ⑫	③ ↓ ⑫	⑤ ↓ ⑫	⑧ ↓ ⑫	② ↓ ⑬	③ ↓ ⑬	④ ↓ ⑬	⑤ ↓ ⑬	⑥ ↓ ⑬	⑦ ↓ ⑬	⑧ ↓ ⑬	V_{CP}	V_{DS}	V_{RC}
114DNS	114.114.114.114	5	-	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
360 Secure DNS	101.226.4.6	3	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
AdGuard DNS	94.140.14.14	10	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
AhaDNS	5.2.75.75	1	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Ali DNS	223.5.5.5	2	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Baidu DNS	180.76.76.76	2	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗
CenturyLink DNS	205.171.2.26	6	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
CFIEC Public DNS	240C::6644	12	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
CIRA Shield DNS	149.112.121.10	2	-	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Cisco OpenDNS	208.67.220.123	5	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗
CleanBrowsing DNS	185.228.168.9	1	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✓	✗
CloudFlare DNS	1.1.1.1	2	✓	✓	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✗	✗	✓	✗
CNNIC sDNS	1.2.4.8	20	-	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Comodo Secure DNS	8.26.56.10	2	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
ControlD DNS	76.76.2.0	6	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗
CZ.NIC ODVR DNS	185.43.135.1	1	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
DNS for Family	78.47.64.161	1	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
DNS Forge	176.9.1.117	1	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗
DNS.SB	45.11.45.11	1	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗
DNS.WATCH	84.200.69.80	10	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
DNSlify DNS	185.235.81.1	1	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗
DNSPod Public DNS+	119.28.28.28	34	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Dyn DNS	216.146.35.35	8	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
FDN DNS	80.67.169.12	8	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Freenom World DNS	80.80.80.80	7	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Google Public DNS	8.8.8.8	1	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Hurricane Electric DNS	74.82.42.42	1	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗
Level3 DNS	4.2.2.1	6	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
LibreDNS	88.198.92.222	1	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗
Neustar UltraDNS	156.154.70.1	8	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
NextDNS	45.90.28.118	10	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Norton DNS	199.85.126.10	9	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
OneDNS	117.50.10.10	20	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
OpenNIC DNS	103.1.206.179	10	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗
Quad101 DNS	101.101.101.101	6	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗
Quad9 DNS	9.9.9.9	7	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Safe Surfer DNS	104.155.237.225	1	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗
SafeDNS	195.46.39.39	8	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
SkyDNS	193.58.251.251	8	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Strongarm DNS	52.3.100.184	1	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗
Verisign Public DNS	64.6.64.6	8	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Yandex.DNS	77.88.8.1	8	-	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

'-': Not applicable. Ordered according to the alphabet. ✓: Yes. ✗: No. ✓: Vulnerable. ✗: Not vulnerable.

Results. We list resolvers whose brands can be identified as vulnerable software in Table 4, including Microsoft DNS (*205,984*), BIND (*54,813*), Unbound (*12,765*), PowerDNS Recursor (*12,750*), Knot Resolver (*45*), and CoreDNS (*8*). Since BIND, Unbound, and Knot are vulnerable to V_{RC}, *67,623* (3.7%) resolvers (the sum of them) can be exploited by the DNSCONSUMING attack. Via the ICMP error messages, we determine that *216,317* (11.8%) are vulnerable to the DNSDOS attack, of which 60% employ the negative caching with a TTL value greater than 1 second (identified through our measurements that evaluate the negative caching with continuous cache probing). All *205,984* (11.2%) resolvers running Microsoft DNS are impacted by the DNSPOISONING attack. In total, *423,652 (23.1%) open DNS resolvers are exploitable by the* TUDOOR *attack.*

7. Discussion

Ethical considerations. Since our testing and experiments involve serious vulnerabilities and active resolvers, we take a number of ethical considerations in accordance with the ethical principles of Menlo Report [59] and best practices for network measurements [88]. First, we installed all analyzed software and routers (OSes) on our own machine to evaluate and demonstrate discovered vulnerabilities. Second, regarding 42 public DNS services, we only tested their IPs a handful of times using our own domain and never exploited their bugs. Third, while conducting large-scale experiments against open resolvers, we strictly controlled the probing speed and only sent well-formatted DNS queries (responses) towards (from) our own domain (nameserver), in addition to ICMP error messages. We configured the PTR record and a website utilizing the probing IP to indicate our academic research intent, and no opt-out was received. In the end, we informed affected vendors of all vulnerabilities and received confirmation and acknowledgment from them.

Lessons learned. DNS response pre-processing is an essential facet of the resolution process, which handles both legitimate and malicious packets. However, there is no rigorous specification to guide software implementations, particularly on the receiving side, thus enabling various attacks such as cache poisoning and DoS. After systematic analysis and evaluation against kinds of implementations, we identify multiple new logic vulnerabilities, the majority of which fail to provide a comprehensive consideration for the entire pre-processing mechanism, such as malformed packet handling, query limit checking, and QR validating. Consequently, based on our analysis in Section 4, we call for standardization for the DNS response pre-processing logic and implementation. Besides, manual vulnerability discovery can be hit or miss. We believe that our analysis could provide several guidelines for automated approaches, which will be our future work.

Mitigation solutions. To eliminate the TuDoor attack, we recommend that DNS implementations adopt the state-of-the-art DNS response pre-processing practices described in Section 4.1, which stipulate that *resolvers should await a time window for promising normal responses.* Specifically, they should reject any malformed packets and continue to receive valid responses. In fact, all software has attempted to implement this workflow, but they fail to consider some corner cases. In addition, to prevent cache poisoning, resolvers might enable 0x20 encoding [36] and DNSSEC [14] (incorporated with domain holders signing their domains).

Detection. Network operators could identify potential attack traffic by censoring DNS responses (QR=1) enumerating the TXID or DNS queries (QR=0) sent to non-53 UDP ports.

Online tool. To assess resolvers' resilience to TuDoor, we developed an online evaluation tool for customers, which is made available at this website: https://test.tudoor.net.

Disclosure and feedback. Follow the ethical policy, we have responsibly disclosed found vulnerabilities to affected vendors, including 22 software vendors, 18 public DNS providers, and other parties such as 7 router producers and network operators. We have so far received responses from 29 vendors and discussed mitigation strategies with them, 18 of which confirmed and fixed related vulnerabilities. They will have months to address these issues. We are awaiting responses from others. 33 CVE numbers have been assigned to us (https://tudoor.net), and part of vendors have patched their latest versions. We summarize their feedback below.

- **BIND** confirmed DNSDoS but determined that ICMP attacks were not particular to it; BIND acknowledged DNSConsuming and fixed it [54].
- **Unbound** acknowledged DNSConsuming that they would attempt to send more queries under DNSSEC, although it was indeed resource-consuming.
- **Knot** fixed DNSConsuming with a CVE published.
- **PowerDNS** acknowledged and fixed DNSDoS in the latest version and assigned a CVE to us [92].
- **Microsoft** confirmed the DNSPoisoning attack and issued a CVE [80]. It has patched the DNS component and is reproducing the DNSDoS vulnerability.
- **Technitium** has addressed the DNSDoS and cache poisoning vulnerability [108] with two CVEs.
- **CoreDNS** secured the software against DNSDoS [33].
- **MacOS and IOS** confirmed DNSDoS and employed TCP fallback to mitigate it (as did **CZ.NIC**).
- **ChromeOS** determined DNSDoS as an security vulnerability and is working on the patches [29].
- **AdGuard** confirmed DNSDoS, fixed it by falling back to TCP [2], and requested us to apply for a CVE.
- **Baidu** and **Cloudflare** confirmed the DNSDoS attack and awarded us bounties, respectively.
- **CleanBrowsing**, **ControlD DNS**, **DNSlify**, **Hurricane Electric**, **LibreDNS**, **Netgear**, **OpenDNS**, **ShieldDNS**, **Strongarm**, **Systemd**, and **Xiaomi** are now further evaluating the vulnerability.
- **DNS.SB** replied it used PowerDNS that is vulnerable.
- **iKuai** and **Mercury** confirmed the query-aggregation issue and are discussing mitigation solutions with us.
- **Node.js** confirmed DNSDoS and replied that it used the c-ares library that has been fixed [25] with a CVE.
- **YogaDNS** patched both the cache poisoning and DoS vulnerabilities and awarded a license key to us [123].

8. Related Work

DNS Resource consuming & DoS attacks. Over the years, significant research efforts have been devoted to analyzing DNS resource-consuming issues. The causes of resource-consuming arise from multiple aspects. First, misconfiguration or vulnerabilities of DNS components can be exploited to force aggressive retries or query loops. For example, in NXNSAttack [3] and NRDelegationAttack [4], a recursive resolver may attempt to query each name server separately if it loads a response with lots of malicious NS records, leading to the victim being flooded with a large number of requests. DNS Unchained [24] and TsuNAME [83] respectively use NS and CNAME records to construct dependency loops and force resolvers to repeatedly launch queries toward authoritative servers during resolution. TsuKing [122] coordinates multiple resolvers to amplify DNS traffic level by level.

Second, attackers can abuse large DNS responses for traffic amplification, resulting in DoS. Some types (e.g., ANY [8], [84] and TXT) of DNS queries can yield large responses, while the policy of minimal-sized responses for ANY queries can partially mitigate such attacks [1]. Some DNS extensions (e.g., DNSSEC [98]) can also introduce more records and enlarge DNS responses. In addition, a recent work proposed a systematic fuzzing approach to analyzing UDP-based protocols and digging out query patterns that can yield traffic amplification [66].

DNS cache poisoning attacks. Off-path attackers can exploit vulnerabilities in operating systems or DNS software implementations to poison DNS caches. In 2008, such attacks were popularized by Dan Kaminsky [58]. From then on, several mechanisms, including source port and TXID randomization, were proposed to prevent DNS cache poisoning. However, researchers have continued to challenge the security of these mechanisms. [50] proposed a method to de-randomize the source ports of resolvers behind NATs, and [5] achieved a similar goal by using malware to exhaust the local source ports on clients. Klein et al. [61] de-randomized the Linux kernel's PRNG to predict the local source ports for cache poisoning. Several research delved into the available side channels (e.g., ICMP global rate limit [74] and ICMP fragment or redirection options [75]) for inferring source ports. However, predicting transaction information can usually be slow and difficult, so fragmentation techniques were developed to eliminate the requirements of guessing the source ports or TXIDs [51], [124]. In addition, [55], [56] found that an attacker could use misinterpretation of special encoded characters to trick residential routers into caching the injected DNS records. MaginotDNS [71] first exploited new techniques for poisoning conditional DNS resolvers.

9. Conclusion

In this paper, we perform the first comprehensive research on the security of DNS response pre-processing via source code inspection, black-box testing, and state machine developing. We systematically investigate the ongoing RFC standards and 28 DNS software implementations, identify three types of novel logic vulnerabilities, and propose high-impact TuDoor attack models. We demonstrate that 24/28 mainstream DNS implementations can be exploited to launch DNSConsuming, DNSDoS, and DNSPoisoning attacks. By conducting large-scale measurements, we evaluate the real-world impacts of these attacks. We find that 18/42 public DNS services, 7/22 router OSes, and 23.1% of 1.8M open DNS resolvers are vulnerable to TuDoor.

Acknowledgement

We thank all the anonymous reviewers and our shepherd for their valuable comments and especially for our shepherd's thoughtful and patient guidance in helping us to improve this paper. Authors from Tsinghua University were supported by the National Natural Science Foundation of China (U1836213, U19B2034, 62102218, and 62132011). Authors from University of California, Irvine (UCI) were supported by NSF CNS-2047476. Most of Xiang Li's work was done when visiting UCI as a project specialist.

References

[1] Joe Abley, Olafur Gudmundsson, Marek Majkowski, and Evan Hunt. RFC 8482: Providing Minimal-Sized Responses to DNS Queries That Have QTYPE=ANY. *RFC Proposed Standard*, 2019.

[2] AdGuard. upstreamplain.go. https://github.com/AdguardTeam/AdGuardDNS/blob/master/internal/dnsserver/forward/upstreamplain.go#L160, 2023.

[3] Yehuda Afek, Anat Bremler-Barr, and Lior Shafir. NXNSAttack: Recursive DNS Inefficiencies and Vulnerabilities. In *USENIX Security '20*, 2020.

[4] Yehuda Afek, Anat Bremler-Barr, and Shani Stajnrod. NRDelegationAttack: Complexity DDoS Attack on DNS Recursive Resolvers . In *USENIX Security '23*, 2023.

[5] Fatemah Alharbi, Jie Chang, Yuchen Zhou, Feng Qian, Zhiyun Qian, and Nael B. Abu-Ghazaleh. Collaborative Client-Side DNS Cache Poisoning Attack. In *INFOCOM '19*, 2019.

[6] Eihal Alowaisheq, Peng Wang, Sumayah Alrwais, Xiaojing Liao, XiaoFeng Wang, Tasneem Alowaisheq, Xianghang Mi, Siyuan Tang, and Baojun Liu. Cracking the Wall of Confinement: Understanding and Analyzing Malicious Domain Take-downs. In *NDSS '19*, 2019.

[7] Sumayah Alrwais, Xiaojing Liao, Xianghang Mi, Peng Wang, XiaoFeng Wang, Feng Qian, Raheem Beyah, and Damon McCoy. Under the Shadow of Sunshine: Understanding and Detecting Bulletproof Hosting on Legitimate Service Provider Networks. In *S&P '17*, 2017.

[8] Marios Anagnostopoulos, Georgios Kambourakis, Stefanos Gritzalis, and David K. Y. Yau. Never Say Never: Authoritative TLD Nameserver-powered DNS Amplification. In *NOMS '18*, 2018.

[9] Mark Andrews. Using IP_RECVERR/IPV6_RECVERR on resolver client sockets. https://lists.dns-oarc.net/pipermail/dns-operations/2019-January/018347.html, 2019.

[10] Mark P. Andrews. RFC 2308: Negative Caching of DNS Queries (DNS NCACHE). *RFC Proposed Standard*, 1998.

[11] Android. DNS Resolver. https://source.android.com/docs/core/ota/modular-system/dns-resolver, 2023.

[12] APNIC. DNS Resolvers Use. https://stats.labs.apnic.net/rvrs, 2023.

[13] Apple. MacOS & IOS libresolv. https://opensource.apple.com/source/libresolv/libresolv-68/res_send.c.auto.html, 2023.

[14] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. RFC 4033: DNS Security Introduction and Requirements. *RFC Proposed Standard*, 2005.

[15] Arstechnica. Major DNS Provider Hit by Mysterious, Focused DDoS Attack. https://arstechnica.com/information-technology/2016/05/major-dns-provider-hit-by-mysterious-focused-ddos-attack/, 2016.

[16] Hitesh Ballani and Paul Francis. Mitigating DNS DoS attacks. In *CCS '08*, 2008.

[17] BIND. CVE-2020-8616: BIND Does Not Sufficiently Limit... https://kb.isc.org/docs/cve-2020-8616, 2020.

[18] BIND. How Do I Change the Version that BIND Reports When Queried for version.bind? https://kb.isc.org/docs/aa-00359, 2021.

[19] BIND. dns_dispatch_add. https://gitlab.isc.org/isc-projects/bind9/-/blob/v9_18_13/lib/dns/resolver.c#L2263, 2023.

[20] BIND. udp_dispatch_getnext. https://gitlab.isc.org/isc-projects/bind9/-/blob/v9_18_13/lib/dns/dispatch.c#L630, 2023.

[21] BIND. udp_recv. https://gitlab.isc.org/isc-projects/bind9/-/blob/v9_18_13/lib/dns/dispatch.c#L501, 2023.

[22] Robert Braden. RFC 1122: Requirements for Internet Hosts – Communication Layers. *RFC Internet Standard*, 1989.

[23] Robert T. Braden. RFC 1123: Requirements for Internet Hosts – Application and Support. *RFC Internet Standard*, 1989.

[24] Jonas Bushart and Christian Rossow. DNS Unchained: Amplified Application-Layer DoS Attacks against DNS Authoritatives. In *RAID '18*, 2018.

[25] c-ares. 0-byte UDP payload Denial of Service. https://github.com/c-ares/c-ares/security/advisories/GHSA-9g78-jv2r-p7vc, 2023.

[26] CAIDA. State of IP Spoofing. https://spoofer.caida.org/summary.php, 2023.

[27] Kenjiro Cho, Kensuke Fukuda, Vivek Pai, Neil Spring, Marc Kührer, Thomas Hupperich, Jonas Bushart, Christian Rossow, and Thorsten Holz. Going Wild: Large-Scale Classification of Open DNS Resolvers. In *IMC '15*, 2015.

[28] Chrome. Chrome Host Resolution. https://source.chromium.org/chromium/chromium/src/+/main:net/dns/, 2023.

[29] Chrome. Chrome/ChromeOS DNS Can Be Made to Fail by Attacker. https://bugs.chromium.org/p/chromium/issues/detail?id=1424084, 2023.

[30] ChromeOS. DoReadResponseComplete. https://source.chromium.org/chromium/chromium/src/+/main:net/dns/dns_transaction.cc;l=340, 2023.

[31] Cloudflare. DDoS Reports. https://blog.cloudflare.com/tag/ddos-reports/, 2023.

[32] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and William Polk. RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. *RFC Proposed Standard*, 2008.

[33] CoreDNS. Continue Waiting after Receiving Malformed Responses. https://github.com/coredns/coredns/pull/6014, 2023.

[34] CoreDNS. Struct Server. https://github.com/coredns/coredns/blob/master/core/dnsserver/server.go#L34, 2023.

[35] David Dagon, Manos Antonakakis, Kevin Day, Xiapu Luo, Christopher P. Lee, and Wenke Lee. Recursive DNS Architectures and Vulnerability Implications. In *NDSS '09*, 2009.

[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 *CCS '08*, 2008.

[37] Tianxiang Dai, Philipp Jeitner, Haya Shulman, and Michael Waidner. From IP to Transport and Beyond: Cross-layer Attacks against Applications. In *SIGCOMM '21*, 2021.

[38] Joeri de Ruiter and Erik Poll. Protocol State Fuzzing of TLS Implementations. In *USENIX Security '15*, 2015.

[39] DNS-OARC. Fpdns. https://www.dns-oarc.net/tools/fpdns, 2021.

[40] dnsjava. send. https://github.com/dnsjava/dnsjava/blob/release/3.5.x/src/main/java/org/xbill/DNS/Resolver.java#L150, 2023.

[41] Dnsmasq. reply_query. https://thekelleys.org.uk/gitweb/?p=dnsmasq.git;a=blob;f=src/forward.c#l1088, 2023.

[42] Dnsmasq. while. https://thekelleys.org.uk/gitweb/?p=dnsmasq.git;a=blob;f=src/dnsmasq.c#l1060, 2023.

[43] dnspython. receive_udp. https://github.com/rthalley/dnspython/blob/2.3/dns/query.py#L511, 2023.

[44] Robert Elz and Randy Bush. RFC 2181: Clarifications to the DNS Specification. *RFC Proposed Standard*, 1997.

[45] Golang DNS Library. connect. https://github.com/coredns/coredns/blob/master/plugin/forward/connect.go#L117, 2023.

[46] Fernando Gont. ICMP Attacks against TCP. *OWASP*, 2006.

[47] Fernando Gont. RFC 5927: ICMP Attacks against TCP. *RFC Informational*, 2010.

[48] Run Guo, Jianjun Chen, Yihang Wang, Keran Mu, Baojun Liu, Xiang Li, Chao Zhang, Haixin Duan, and Jianping Wu. Temporal CDN-Convex Lens: A CDN-Assisted Practical Pulsing DDoS Attack. In *USENIX Security '23*, 2023.

[49] Mukesh Gupta and Alex Conta. RFC 4443: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. *RFC Internet Standard*, 2006.

[50] Amir Herzberg and Haya Shulman. Security of Patched DNS. In *ESORICS '12*, 2012.

[51] Amir Herzberg and Haya Shulman. Fragmentation Considered Poisonous, or: One-domain-to-rule-them-all.org. In *CNS '13*, 2013.

[52] Alden Hilton, Casey Deccio, and Jacob Davis. Fourteen Years in the Life: A Root Server's Perspective on DNS Resolver Security. In *USENIX Security '23*, 2023.

[53] ICANN. Centralized Zone Data Service. https://czds.icann.org/, 2023.

[54] idealeer. ICMP Error Messages Causing BIND9 to Send More Queries than Intended). https://gitlab.isc.org/isc-projects/bind9/-/issues/4005, 2023.

[55] Philipp Jeitner and Haya Shulman. Injection Attacks Reloaded: Tunnelling Malicious Payloads over DNS. In *USENIX Security '21*, 2021.

[56] Philipp Jeitner, Haya Shulman, Lucas Teichmann, and Michael Waidner. XDRI Attacks - and - How to Enhance Resilience of Residential Routers. In *USENIX Security '22*, 2022.

[57] Jian Jiang, Jinjin Liang, Kang Li, Jun Li, Hai-Xin Duan, and Jianping Wu. Ghost Domain Names: Revoked Yet Still Resolvable. In *NDSS '12*, 2012.

[58] Dan Kaminsky. It's the End of the Cache as We Know It. https://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf, 2008.

[59] Erin Kenneally and David Dittrich. The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research. *SSRN Electronic Journal*, 2012.

[60] Amit Klein. DNS Record Injection Vulnerabilities in Home Routers. http://www.icir.org/mallman/talks/schomp-dns-security-nanog61.pdf, 2014.

[61] Amit Klein. Cross Layer Attacks and How to Use Them (for DNS Cache Poisoning, Device Tracking and More). In *S&P '21*, 2021.

[62] Amit Klein, Haya Shulman, and Michael Waidner. Internet-Wide Study of DNS Cache Injections. In *INFOCOM '17*, 2017.

[63] Knot Resolver. Max. # of Retries after Timeout. https://gitlab.nic.cz/knot/knot-resolver/-/blob/v5.6.0/lib/defines.h#L55, 2023.

[64] Knot Resolver. udp_queue_init_global. https://gitlab.nic.cz/knot/knot-resolver/-/blob/v5.6.0/daemon/main.c#L564, 2023.

[65] Knot Resolver. worker_submit. https://gitlab.nic.cz/knot/knot-resolver/-/blob/v5.6.0/daemon/worker.c#L1694, 2023.

[66] Johannes Krupp, Ilya Grishchenko, and Christian Rossow. AmpFuzz: Fuzzing for Amplification DDoS Vulnerabilities. In *USENIX Security '22*, 2022.

[67] Mikael Kullberg. Random Subdomain Attacks. https://www.netnod.se/sites/default/files/2016-11/mikael_kullberg_random_subdomain_attacks.pdf, 2015.

[68] Aleksandar Kuzmanovic and Edward W Knightly. Low-Rate TCP-Targeted Denial of Service Attacks: The Shrew vs. the Mice and Elephants. In *SIGCOMM '03*, 2003.

[69] Xiang Li, Baojun Liu, Xuesong Bai, Mingming Zhang, Qifan Zhang, Zhou Li, Haixin Duan, and Qi Li. Ghost Domain Reloaded: Vulnerable Links in Domain Name Delegation and Revocation. In *NDSS '23*, 2023.

[70] Xiang Li, Baojun Liu, Xiaofeng Zheng, Haixin Duan, Qi Li, and Youjun Huang. Fast IPv6 Network Periphery Discovery and Security Implications. In *DSN '21*, 2021.

[71] Xiang Li, Chaoyi Lu, Baojun Liu, Qifan Zhang, Zhou Li, Haixin Duan, and Qi Li. The Maginot Line: Attacking the Boundary of DNS Caching Protection. In *USENIX Security '23*, 2023.

[72] Liku Zelleke. Top 20 Best Open Source DNS Servers for (Linux / Windows). https://cloudinfrastructureservices.co.uk/top-20-best-open-source-dns-servers-for-linux-windows/, 2023.

[73] Baojun Liu, Chaoyi Lu, Hai-Xin Duan, Ying Liu, Zhou Li, Shuang Hao, and Min Yang. Who Is Answering My Queries: Understanding and Characterizing Interception of the DNS Resolution Path. In *USENIX Security '18*, 2018.

[74] Keyu Man, Zhiyun Qian, Zhongjie Wang, Xiaofeng Zheng, Youjun Huang, and Haixin Duan. DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels. In *CCS '20*, 2020.

[75] Keyu Man, Xinan Zhou, and Zhiyun Qian. DNS Cache Poisoning Attack: Resurrections with Side Channels. In *CCS '21*, 2021.

[76] MaraDNS. bigloop. https://github.com/samboy/MaraDNS/blob/3.5.0035/deadwood-github/src/DwSocket.c#L1228, 2023.

[77] MaraDNS. get_remote_udp_packet. https://github.com/samboy/MaraDNS/blob/3.5.0035/deadwood-github/src/DwUdpSocket.c#L1362, 2023.

[78] Florian Maury. The Indefinitely Delegating Name Servers (iDNS) Attack. In *OARC '15*, 2015.

[79] Xianghang Mi, Xuan Feng, Xiaojing Liao, Baojun Liu, XiaoFeng Wang, Feng Qian, Zhou Li, Sumayah Alrwais, Limin Sun, and Ying Liu. Resident Evil: Understanding Residential IP Proxy as a Dark Service. In *S&P '19*, 2019.

[80] Microsoft. Domain Name System Docs. https://docs.microsoft.com/en-us/windows-server/networking/dns/dns-top, 2023.

[81] Paul V. Mockapetris. RFC 1034: Domain Names - Concepts and Facilities. *RFC Internet Standard*, 1987.

[82] Paul V. Mockapetris. RFC 1035: Domain Names - Implementation and Specification. *RFC Internet Standard*, 1987.

[83] Giovane CM Moura, Sebastian Castro, John Heidemann, and Wes Hardaker. TsuNAME: Exploiting Misconfiguration and Vulnerability to DDoS DNS. In *IMC '21*, 2021.

[84] Marcin Nawrocki, Mattijs Jonker, Thomas C. Schmidt, and Matthias Wählisch. The Far Side of DNS Amplification: Tracing the DDoS Attack Ecosystem from the Internet Core. In *IMC '21*, 2021.

[85] nodejs. DNS of nodejs. https://github.com/nodejs/node/blob/main/doc/api/dns.md, 2023.

[86] Adam Oest, Yeganeh Safaei, Penghui Zhang, Brad Wardman, Kevin Tyers, Yan Shoshitaishvili, and Adam Doupé. PhishTime: Continuous Longitudinal Measurement of the Effectiveness of Anti-phishing Blacklists. In *USENIX Security '20*, 2020.

[87] OpenWrt. DNS and DHCP configuration. https://openwrt.org/docs/guide-user/base-system/dhcp, 2023.

[88] Craig Partridge and Mark Allman. Ethical Considerations in Network Measurement Papers. *Communications of the ACM*, 2016.

[89] Jon Postel. RFC 768: User Datagram Protocol. *RFC Internet Standard*, 1980.

[90] Jon Postel. RFC 792: Internet Control Message Protocol. *RFC Internet Standard*, 1981.

[91] Jon Postel. RFC 793: Transmission Control Protocol. *RFC Internet Standard*, 1981.

[92] PowerDNS. PowerDNS Security Advisory 2023-02. https://docs.powerdns.com/recursor/security-advisories/powerdns-advisory-2023-02.html, 2023.

[93] PowerDNS Recursor. arecvfrom. https://github.com/PowerDNS/pdns/blob/master/pdns/recursordist/pdns_recursor.cc#L318, 2023.

[94] PowerDNS Recursor. recvfrom. https://github.com/PowerDNS/pdns/blob/master/pdns/recursordist/pdns_recursor.cc#L2759, 2023.

[95] PowerDNS Recursor. return. https://github.com/PowerDNS/pdns/blob/master/pdns/recursordist/pdns_recursor.cc#L2779, 2023.

[96] Ryan Rasti, Mukul Murthy, Nicholas Weaver, and Vern Paxson. Temporal Lensing and Its Application in Pulsing Denial-of-Service Attacks. In *S&P '15*, 2015.

[97] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. DNSSEC and Its Potential for DDoS Attacks: A Comprehensive Measurement Study. In *IMC '14*, 2014.

[98] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. DNSSEC and Its Potential for DDoS Attacks: A Comprehensive Measurement Study. In *IMC '14*, 2014.

[99] RIPE. RIPE Atlas. https://atlas.ripe.net/, 2023.

[100] RouterChart. Popular Routers. https://routerchart.com/brands, 2023.

[101] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. On Measuring the Client-side DNS Infrastructure. In *IMC '13*, 2013.

[102] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. Assessing DNS Vulnerability to Record Injection. In *PAM '14*, 2014.

[103] Sooel Son and Vitaly Shmatikov. The Hitchhiker's Guide to DNS Cache Poisoning. In *SecureComm '10*, 2010.

[104] Joe Stewart. DNS Cache Poisoning – The Next Generation. *Secureworks*, 2003.

[105] systemd. systemd-resolved.service and VPNs. https://systemd.io/RESOLVED-VPNS/, 2023.

[106] systemd-resolved. on_dns_packet. https://github.com/systemd/systemd/blob/main/src/resolve/resolved-dns-transaction.c#L1406, 2023.

[107] systemd-resolved. sd_event_add_io. https://github.com/systemd/systemd/blob/main/src/resolve/resolved-dns-transaction.c#L1491, 2023.

[108] Technitium. Technitium Version 11.1 & 11.0.3. https://github.com/TechnitiumSoftware/DnsServer/blob/master/CHANGELOG.md#version-111, 2023.

[109] Technitium DNS. Help: Negative Cache. https://technitium.com/dns/help.html, 2023.

[110] Technitium DNS. PostProcessQueryAsync. https://github.com/TechnitiumSoftware/DnsServer/blob/v11.0.1/DnsServerCore/Dns/DnsServer.cs#L985, 2023.

[111] Technitium DNS. ReceiveFromAsync. https://github.com/TechnitiumSoftware/TechnitiumLibrary/blob/master/TechnitiumLibrary.Net/SocketExtensions.cs#L113, 2023.

[112] Technitium DNS. Socket. https://github.com/dotnet/runtime/blob/main/src/libraries/System.Net.Sockets/src/System/Net/Sockets/Socket.cs#L118, 2023.

[113] The Guardian. DDoS Attack That Disrupted Internet Was Largest of Its Kind in History, Experts Say. https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet, 2016.

[114] Unbound. comm_point_udp_callback. https://github.com/NLnetLabs/unbound/blob/branch-1.17.1/util/netevent.c#L934, 2023.

[115] Unbound. iter_operate. https://github.com/NLnetLabs/unbound/blob/branch-1.17.1/iterator/iterator.c#L4139, 2023.

[116] Unbound. outnet_udp_cb. https://github.com/NLnetLabs/unbound/blob/branch-1.17.1/services/outside_network.c#L1437, 2023.

[117] Valgrind. Valgrind. https://valgrind.org/, 2023.

[118] Chuhan Wang, YASUHIRO KURANAGA, Yihang Wang, Mingming Zhang, Linkai Zheng, lixiang, Jianjun Chen, Haixin Duan, Yanzhong Lin, and Qingfeng Pan. BreakSPF: How Shared Infrastructures Magnify SPF Vulnerabilities across the Internet. In *NDSS '24*, 2024.

[119] Duane Wessels, William Carroll, and Matthew Thomas. RFC Draft: Negative Caching of DNS Resolution Failures. *RFC Draft*, 2023.

[120] Wikipedia. List of Router Firmware Projects. https://en.wikipedia.org/wiki/List_of_router_firmware_projects, 2023.

[121] Xinhua. Ruins of Secret Passages ... https://www.chinadaily.com.cn/a/202301/10/WS63bcb1d5a31057c47eba8995.html, 2023.

[122] Wei Xu, Xiang Li, Chaoyi Lu, Baojun Liu, Jia Zhang, Jianjun Chen, Tao Wan, and Haixin Duan. TsuKing: Coordinating DNS Resolvers and Queries into Potent DoS Amplifiers. In *CCS '23*, 2023.

[123] YogaDNS. Changelog: Version 1.38 (2023.05.22). https://yogadns.com/download/, 2023.

[124] Xiaofeng Zheng, Chaoyi Lu, Jian Peng, Qiushi Yang, Dongjie Zhou, Baojun Liu, Keyu Man, Shuang Hao, Haixin Duan, and Zhiyun Qian. Poison Over Troubled Forwarders: A Cache Poisoning Attack Targeting DNS Forwarding Devices. In *USENIX Security '20*, 2020.

Appendix A. Meta-Review

A.1. Summary

The paper analyzes the response pre-processing pipeline of existing DNS resolvers, forwarders, and libraries for identifying exploitable logical vulnerabilities. Based on insights from DNS RFCs, manual audit of open-source DNS implementations, and black-box analysis of commercial DNS resolvers, the paper discovered logical vulnerabilities in DNS deployments that can result in the following three types of attacks: DNS cache poisoning; Denial-of-Service (DoS); and resource exhaustion attacks. The discovered attacks are responsibly disclosed to the relevant vendors and have been assigned CVEs.

A.2. Scientific Contributions

- Independent Confirmation of Important Results with Limited Prior Research
- Identifies an Impactful Vulnerability

A.3. Reasons for Acceptance

1) The paper identified new logical vulnerabilities in the DNS resolution pipeline that can result in cache poisoning and DoS in DNS resolvers, libraries, and forwarders.
2) The paper has performed a large-scale evaluation (e.g., 1.8 million open resolvers) and identified vulnerabilities in a wide variety of products, including popular Wi-Fi routers, router OSes, public DNS services, and open DNS resolvers.
3) The paper's evaluation has followed the community recommendations of carrying out vulnerability research on deployed systems, and the resulting findings have been responsibly disclosed and acknowledged.

A.4. Noteworthy Concerns

1) Although all three identified attacks have severe repercussions, two of them require making strong assumptions about the attacker's capabilities. Concretely, the resource consumption attack in the paper requires an on-path attacker whereas the cache poisoning attack requires the victim to disable DNSSEC validation and 0x20 encoding.

Appendix B. Response to the Meta-Review

Thank our anonymous reviewers and shepherd for their insightful comments. We concur with the reviewers on the content of this meta-review. With regard to the noteworthy concerns, we acknowledge certain requirements of our attacks. In detail, the resource consumption attack needs attackers to obtain the DNS query on his or her own nameserver to return a malformed response. The cache poisoning attack requires the resolver to disable DNSSEC validation and 0x20 encoding like all DNS cache poisoning attacks.