The Maginot Line: Attacking the Boundary of DNS Caching Protection

Xiang Li[*], Chaoyi Lu[*], Baojun Liu[*✉], Qifan Zhang[†]
Zhou Li[†], Haixin Duan[*‡§], and Qi Li[*§✉]

[*]*Tsinghua University,* [†]*University of California, Irvine*
[‡]*QI-ANXIN Technology Research Institute,* [§]*Zhongguancun Laboratory*

{x-l19}@mails.tsinghua.edu.cn, {luchaoyi}@mail.tsinghua.edu.cn
{lbj, duanhx, qli01}@tsinghua.edu.cn, {qifan.zhang, zhou.li}@uci.edu

Abstract

In this paper, we report MAGINOTDNS, a powerful cache poisoning attack against DNS servers that simultaneously act as recursive resolvers and forwarders (termed as *CDNS*). The attack is made possible through exploiting vulnerabilities in the bailiwick checking algorithms, one of the cornerstones of DNS security since the 1990s, and affects multiple versions of popular DNS software, including BIND and Microsoft DNS. Through field tests, we find that the attack is potent, allowing attackers to take over *entire* DNS zones, even including Top-Level Domains (e.g., `.com` and `.net`). Through a large-scale measurement study, we also confirm the extensive usage of CDNSes in real-world networks (up to 41.8% of our probed open DNS servers) and find that at least 35.5% of all CDNSes are vulnerable to MAGINOTDNS. After interviews with ISPs, we show a wide range of CDNS use cases and real-world attacks. We have reported all the discovered vulnerabilities to DNS software vendors and received acknowledgments from all of them. 3 CVE-ids have been published, and 2 vendors have fixed their software. Our study brings attention to the implementation inconsistency of security checking logic in different DNS software and server modes (i.e., recursive resolvers and forwarders), and we call for standardization and agreements among software vendors.

1 Introduction

The Domain Name System (DNS) provides translation between human-readable domain names and numerical addresses and is relied on by many fundamental Internet security mechanisms such as certificate issuance [95], email authentication [84, 92], and malicious domain takedown [5]. Unfortunately, almost all DNS messages are designed to be sent unencrypted, which are vulnerable to manipulation [63].

Among the reported attacks against DNS, cache poisoning is considered one of the most dangerous. For the earliest version of DNS, malicious authoritative servers can respond with DNS data for arbitrary zones that are outside of their authority to poison any domain [34, 83, 86]. To reject such "unsolicited" responses, *the bailiwick rule* [85] has been implemented by DNS software (e.g., BIND [12] and Unbound [91]), which serves as a fundamental defense against DNS cache poisoning. However, till the writing of this paper, DNS standards [42, 70] have not defined concrete cache algorithms, leaving software developers to implement. This implementation-to-standard gap results in various DNS attacks.

Prior attacks. Previous cache poisoning attacks focused on tricking DNS servers into accepting rogue in-bailiwick responses rather than breaking the bailiwick rule. Proposed delicate methods include UDP source port brute-forcing [46], birthday attacks [85, 86], IP fragmentation [39, 96], and ICMP-based side channels [65, 66]. A major problem with these in-bailiwick attacks is the attack efficiency: the attacker can only poison victim domains (usually Second-Level Domains like `example.com`) one by one. A few works choose to attack devices that do not perform bailiwick checks [34, 81], but only a small fraction of DNS software ignore bailiwick check. Some studies looked into what DNS data could be stuffed in rogue DNS responses without being rejected under the bailiwick rule [55, 85]. In the end, the prior studies suggest that breaking bailiwick is hard, and the attack should focus on the settings that do not violate bailiwick checks.

Our study. In this paper, we revisit the status of bailiwick checking and demonstrate that its logic in DNS software (BIND [12], Knot [77], Microsoft DNS [87], and Technitium [89]) has vulnerabilities that enable new cache poisoning attacks (Section 4), which we termed as MAGINOTDNS. Since MAGINOTDNS breaks the bailiwick checking, it is powerful in that it can poison *delegation data* and hijack arbitrary DNS zones *in its entirety*, even including Top-Level Domains (e.g., `.com` and `.net`).

The key to the discovery of MAGINOTDNS is the inconsistent bailiwick implementations between different DNS modes. In particular, we found new bailiwick vulnerabilities in the *forwarder* mode of DNS software. The vulnerabilities do not harm the regular forwarders as they do not perform recursive

✉ Corresponding authors.

domain resolutions, but for *conditional DNS servers* (CDNS), severe consequences can be caused. CDNS is a prevalent type of DNS server but not yet systematically studied. It is configured to act as recursive resolver and forwarder simultaneously, and the different server modes share the same global cache. As a result, attackers can exploit the forwarder vulnerabilities and "cross the boundary" – attack recursive resolvers on the same server. Under this general idea, we propose an on-path attack and an off-path attack, and demonstrate effective attacks against the latest versions of DNS software, by combining two extra vulnerabilities in source port randomization and `CNAME` chasing discovered by us (Section 5).

After testing the DNS software, we try to assess how many deployed DNS servers are impacted. We encounter a challenge in identifying the target of the attack, CDNS, as no DNS server directly tells if it is CDNS. We propose a novel methodology to find them at a large scale in both open networks and enterprise networks (Section 6). Our measurement results confirm that CDNSes are prevalent servers in the wild (up to 41.8% of all probed open DNS servers), and a large number of them (at least **35.5% of all CDNSes**) are vulnerable to the attacks. After interviews with ISPs, we show a wide range of CDNS use cases and real-world attacks. We have reported all vulnerabilities to the affected DNS software vendors (Section 7) and have received acknowledgements from all of them. So far, 3 CVE-ids have been published, and 2 vendors have patched their software [9].

In the end, our study uncovers the implementation inconsistency of DNS security checking logic, both in different software and server modes (i.e., recursive resolvers and forwarders). The root cause of the vulnerabilities stems from the missing of a rigorous definition of the bailiwick checking logic in the standard and the confusion about the position and functionalities of DNS forwarders. As a result, we recommend that the stakeholders, including the standardization parties, DNS software vendors, and DNS server owners, address the vulnerabilities.

Contributions. We make the following contributions:

New threat model. We present a new threat model against CDNS, a new type of server extensively deployed in the DNS infrastructure that has not been studied.

New attack surface. We review source code and dynamically debug bailiwick checking algorithms in 9 types of DNS software. This yields new vulnerabilities that allow powerful cache poisoning attacks and hijacking of entire DNS zones. We validated the attacks in a controlled environment, and we present attack demos and details at [9].

New methodology and results. We propose a new methodology to identify the new type of server, CDNS, at a large scale in real networks. We also find that a large number of them are vulnerable to our attack.

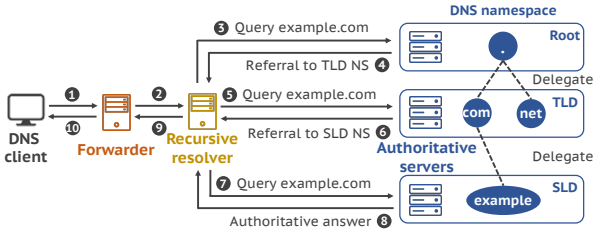

Figure 1: A standard DNS resolution process for domain `example.com` under the DNS namespace.

2 Background

In this part, we provide background on DNS concepts, conditional DNS servers, and caching mechanisms [69, 70].

2.1 DNS Concepts and Infrastructure

DNS namespace and domain resolution. As shown in Figure 1, the DNS namespace is a hierarchical structure comprising multiple layers of *DNS zones*, including Root (denoted by "."), Top-Level Domains (TLDs, e.g., `.com`), Second-Level Domains (SLDs, e.g., `example.com`), etc. In each DNS zone, authoritative data is organized by domain labels and types in the form of *resource record sets* (RRSets), which are distributed by *authoritative servers* (also *nameservers* or *NS* for short). Common RRSet types include `A` that stores IPv4 addresses, `CNAME` that stores domain aliases, and `NS` that stores *delegation* data (nameserver) of child DNS zones.

To resolve a DNS query, servers operating in *recursive mode* (called *recursive resolvers*) contact authoritative servers starting from root (steps ❸ to ❽ in Figure 1). During domain resolution, authoritative servers without local data about the query provide *referrals* that contain information to other servers *"closer"* to the final answer (e.g., a child DNS zone). DNS referral messages include a type `NS` RRSet pointing to the next servers (in the authority section) and *glue records* which are type `A` (or `AAAA`) RRSets specifying the next server addresses[1] (in the additional section). The resolution terminates upon getting an *authoritative answer* (indicated by `AA` flag in the DNS header) from the server whose local DNS zone can provide answers to the query. Figure 2 shows the simplified referral message and authoritative answer to DNS queries for `example.com`.

DNS forwarders. As the DNS infrastructure evolves, *forwarders* are increasingly deployed as ingress servers [82] that sit between DNS clients and upstream servers. They are commonly installed on home routers [96] and can serve as default local resolvers for DNS clients [10]. Unlike recursive resolvers, forwarders do not perform recursive resolution but pass DNS queries to upstream servers (e.g., public DNS services like Cloudflare's `1.1.1.1` [1] or authoritative servers).

[1] Glue records only appear in referrals if the next servers are *in-bailiwick* [42] to the current zone.

Flags: QR RD;	Flags: QR AA RD;
Question section: example.com. A	Question section: example.com. A
Answer section: (Empty)	Answer section: example.com. A 93.184.216.34
Authority section: com. NS a.gtld-servers.net.	Authority section: (Empty)
Additional section: a.gtld-servers.net. A 192.5.6.30	Additional section: (Empty)

Figure 2: Referral to TLD `NS` (left) and authoritative answer to type `A` queries for `example.com` (right).

a)
```
options {
    recursion yes;
                /* Z_R includes the entire namespace */
};
/* Opt-out example.com and creates Z_F */
zone "example.com" {
    type forward;               /* Enables forwarding */
    forwarders { 1.1.1.1; };    /* Specifies upstream */
};
```

b)
```
zone "." {          /* Z_F includes the entire namespace */
    type forward;               /* Enables forwarding */
    forwarders { 1.1.1.1; };    /* Specifies upstream */
};
/* Opt-out example.com and creates Z_R */
zone "example.com" {
    type forward;               /* Enables forwarding */
    forwarders {};
     /* Triggers immediate fallback to recursive mode */
};
```

Figure 3: Example BIND configurations of (a) recursive-default CDNS and (b) forwarding-default CDNS [62].

2.2 Conditional DNS Servers (CDNS)

DNS software usually implements multiple server modes (i.e., forwarder, recursive resolver, and authoritative server) to meet the needs of users, network operators, and DNS service providers. Table 1 summarizes the modes available in mainstream DNS software (list drawn from a survey on DNS software and previous studies [44, 57, 58, 65, 96]). Multiple modes in one software can be enabled *simultaneously*.

In this paper, we study DNS servers that act simultaneously as *recursive resolvers and forwarders* (termed as *conditional DNS servers* or *CDNS*). A CDNS is configured by its operator to install lists of *recursive DNS zones* (termed as Z_R) and *forwarding DNS zones* (termed as Z_F), such that the incoming DNS queries can be served by different modes. All queries for domains in Z_R will be recursively resolved, while queries for domains in Z_F will be forwarded to the upstream servers.

To avoid confusion in serving queries, it is required that $Z_R \cap Z_F = \varnothing$. To ensure DNS queries for any domain can be served, together Z_R and Z_F should cover the entire DNS namespace ($Z_R \cup Z_F = DNSRoot$), which is usually achieved through *domain "opt-out"* – operators either create Z_R on a regular forwarder or create Z_F on a regular recursive resolver, such that the other list will automatically include the remaining DNS namespace (i.e., becoming the *default*). Figure 3 shows two examples that configure recursive-default and forwarding-default CDNS in BIND.

Usage of CDNS. We find that, although not being studied by prior works systemically, CDNSes have been extensively used in practice. Enterprises or ISPs may use them for access control [7], speeding up domain resolution [90], and reducing operational cost (e.g., to reduce out-of-band traffic [63]). In Section 6.3, we interviewed two ISPs to get an in-depth understanding of how CDNS is used.

Forwarding fallback. Through code review and testing, we find some DNS software provide a *fallback* option like reissuing the query in recursive mode, if a forwarding query fails (e.g., when the upstream server times out). Table 1 summarizes the fallback options in DNS software. In BIND, the fallback option is enabled by default with a `type forward` keyword (see Figure 3, termed as *forwarding-first* mode) and turned off by adding an extra line `forward only` (termed as *forwarding-only* mode). By contrast, Unbound [91] disables fallback unless `forward-first` is written in the configuration file. Microsoft DNS also provides fallback but performs incomplete recursive resolution [68] that only queries the upstream server for one round. Other tested software do not provide fallback and returns DNS errors if forwarding queries fail. Although forwarding fallback complicates our attack against CDNS, we found it can be thwarted with new techniques described in Section 5.

2.3 DNS Caching Mechanisms

Recursive resolvers and forwarders both use cache to enhance lookup performance. In general, each cached RRSet is associated with a Time-To-Live (TTL) and will be used to serve clients' queries until the TTL expires. Below we demonstrate other specific rules and our observations about caching.

Bailiwick rule. Authoritative servers should not provide data for DNS zones outside of their authority (i.e., *out-of-bailiwick* data) [35], e.g., responses from `.com` authoritative servers should not contain data for `.net` domains. Most DNS resolvers implement *the bailiwick rule* [85] to reject out-of-bailiwick data from upstream servers (see Table 1). Though DNS standard [70] requires that servers to discard such "unsolicited" responses, no concrete algorithms are given, and there exist various DNS software implementations. In Section 4, we provide detailed analysis of their implementations and describe a new class of vulnerabilities in several DNS software that were never reported before.

Data trust levels and cache overwriting. When new DNS responses contain RRSets that have been previously cached, servers should determine which data to keep. To this end, DNS data is ranked by *trustworthiness* (RFC 2181 [35] defines 7 levels), and the data with a higher ranking will be preferred when put into the DNS cache. In general, authoritative answers (with the `AA` flag set) have higher levels than non-authoritative answers. Within the same DNS response, data from the answer section ranks higher than data from the authority or additional section. Through code review, we confirm that most DNS software adopt this model for cache overwriting, though some use different trust levels (shown in Table 1 and Appendix A).

Table 1: DNS operational modes and functionalities available in mainstream implementations.

DNS software		Server role					Cache protection			Cache poisoning defense	
Brand	**Version**	**Auth**[1]	**Recur**[2]	**Fwder**[3]	**CDNS**	**Fall-back**	**Bailiwick checking**	**Trust level**	**Shared cache**	**DNSSEC**	**0x20**
BIND [12]	9.18.0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
Knot Resolver [77]	5.5.2	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓
Unbound [91]	1.16.2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PowerDNS Recursor [75]	4.7.1	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓
Microsoft DNS [87]	2022[4]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
Technitium [89]	7.0	✓	✓	✓	✓	✗	✓	✗	✓	✗	✓
Simple DNS Plus [73]	9.1.108	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓
MaraDNS [67]	3.5.0022	✓	✓	✓	✓	✗	✓	✗	✓	✗	✓
CoreDNS [22]	1.9.3	✓	✓[5]	✓	✓[5]	✓[5]	✗	-[6]	✗	✓	✗
Dnsmasq [33]	2.86	✗	✗	✓	✗	-	✗	✗	-	✓	✗
DNRD [26]	2.20.3	✗	✗	✓	✗	-	✗	✗	-	✗	✗
YADIFA [94]	2.5.4	✓	✗	✗	✗	-	-	-	-	✓	✓
NSD [72]	4.6.0	✓	✗	✗	✗	-	-	-	-	✓	✓

[1] Authoritative server. [2] Recursive resolver. [3] Forwarder. [4] OS build 20348.740. [5] Available only when compiled with extra Unbound extensions. [6] "-" means not applicable.

Global cache in CDNS. We find in most DNS software that, when being configured as CDNS, the recursive resolver and forwarder modes *share a global cache* (see Table 1). Therefore, the DNS data about forwarding queries can overwrite the cache created under recursive resolving. We will soon demonstrate that this setting allows attackers to poison the global DNS cache through a mode with weaker security (i.e., *forwarder*) and attack other more secured modes (i.e., *recursive resolver*).

3 Attack Overview

Previous studies reported that forwarders are more vulnerable to cache poisoning attacks due to their simplified implementation of defense mechanisms. For example, older versions of forwarders may use predictable UDP port numbers [50], skip validation of port numbers and TXIDs [54], and lack bailiwick checks [81]. By contrast, recursive resolvers are often secured with defenses against cache poisoning attacks [82].

The imbalanced security measures on forwarders and recursives inspire us to investigate the possibilities of attacking the recursive resolver that co-locates with a forwarder on one CDNS. Below we first describe the threat model. Then, we overview the attack workflow. Finally, we compare our attack MAGINOTDNS with other cache poisoning attacks.

3.1 Threat Model

Figure 4 overviews the threat model. We assume the target CDNS is pre-configured with Z_R and Z_F by its operator. The attacker leverages a domain d_{attack} (which falls in Z_F, denoted as $d_{attack} \in Z_F$) and chooses different attack strategies based on which profile (i.e., recursive-default or forwarding-default) is default on the target CDNS. For forwarding-default CDNSes, because all unregistered domains are covered by Z_F, the attacker can register d_{attack} and run its authoritative server. For recursive-default CDNSes, any owner of domains covered by Z_F may weaponize his/her domain as d_{attack}. We find through large-scale experiments that domains with high rankings (e.g., Alexa [3] Top 10K sites) are included in Z_F by a large number of CDNSes (in Section 6.2), making their owners potential attackers. Sometimes, as discussed in Section 6.3, ISPs might forward queries to third-party "authoritative" nameservers, which can be also controlled by attackers.

Same as other cache poisoning attacks [39, 46, 65, 96], we assume the attacker can query the target CDNS and spoof the IP address in the forged DNS responses. For an open CDNS, querying from any source IP is allowed. For a closed CDNS that only responds to queries in the same network (e.g., an enterprise CDNS), the attacker could leverage an insider (e.g., enterprise employee) or a compromised machine in the same network. To spoof the source IP, the attacker can choose a bullet-proof hosting service [6], and studies showed that over 25% of IPv4 ASes currently allow IP spoofing behaviors [15, 64]. Meanwhile, integrity checks including DNSSEC validation [8] and 0x20 encoding [27] defend against cache poisoning attacks, and we do not consider CDNSes that enforce them. In Section 6.2, we show that most of the CDNSes in the wild are qualified under this condition.

3.2 Attack Workflow

The attack comprises two steps: (*i*) identifying a CDNS and (*ii*) poisoning its global DNS cache by exploiting forwarder vulnerabilities. The first step poses a challenge that DNS servers do not tell in their responses how a query is resolved backstage. In Section 6 we propose a novel method to address this challenge.

For the second step, the attacker follows the general cache poisoning scheme as shown in Figure 4. First, the attacker sends a query Q for d_{attack} to the target CDNS (step ❶). Because $d_{attack} \in Z_F$ (step ❷), Q triggers a subsequent query

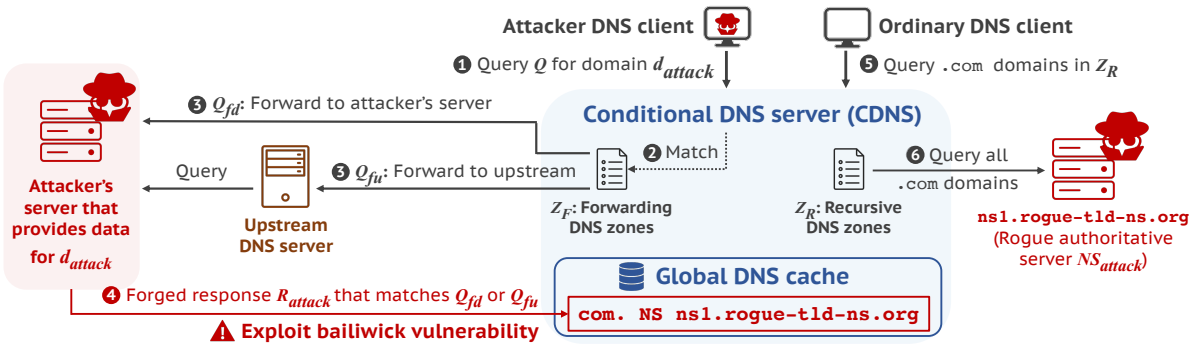

Figure 4: Threat model and attack workflow of MAGINOTDNS.

(step ❸): either a Q_{fd} directly to the attacker's server (e.g., the ISP DNS server example we raised in Section 3.1) or a Q_{fu} to an upstream server (e.g., another recursive resolver). The attacker then generates a rogue response R_{attack} and sends it directly back to the target CDNS (step ❹). R_{attack} contains forged `NS` data (termed NS_{attack}) of out-of-bailiwick domains, e.g., suggesting `ns1.rogue-tld-ns.org` as the nameserver of `.com`, which will be cached by the CDNS and poison Z_R. Finally, when a victim client queries the CDNS for domains in Z_R (step ❺), the recursive resolver will query the rogue authoritative server under the attacker's control (step ❻).

Noticeably, we consider both Q_{fu} and Q_{fd} in step ❸. The former case represents an *off-path* attacker (i.e., the attacker cannot intercept the link between the upstream server and the target CDNS) that has been well studied in cache poisoning. The latter case represents an *on-path* attacker (i.e., the attacker sees queries from the target CDNS) that has received much less attention. Though the attacker needs to run two servers, one provides authoritative data for d_{attack} and the other is NS_{attack}, they can be on the same physical machine.

3.3 Comparison with Related Works

MAGINOTDNS differs from previous DNS cache poisoning attacks in the aspects described below. First, the target of MAGINOTDNS is CDNS, a type of DNS server that has not yet been studied [82]. Second, MAGINOTDNS aims to break the bailiwick rules and can poison arbitrary DNS zones (even TLDs) in its *entirety*. Prior to ours, some works [34, 81] also studied bailiwick rules, but they targeted DNS servers that did not implement bailiwick checks. However, from our test results in Table 1, most mainstream DNS software have fixed this issue. Third, MAGINOTDNS exploits the vulnerabilities of forwarders to attack the recursive resolvers. Although many vulnerabilities about forwarders have been discovered [41, 49–54, 81, 83, 85, 86, 96], none of them conducted a systematic analysis of the bailiwick component.

The more common type of cache poisoning attacks focused on injecting *in-bailiwick* responses, which did not break bailiwick checks. As a result, they can only hijack one domain (typically one SLD) at a time. Though the well-known Kaminsky attack in 2008 [46] can hijack entire zones by poisoning *referrals*, it is still in-bailiwick and has to race against authentic referrals coming from root and TLD nameservers that are out of the attacker's control. These authentic referrals are often responded immediately due to the anycast servers deployed worldwide [60], making the attack window very short. By contrast, our attack is based on poisoning *authoritative answers*, which can be supported by the attacker's server. Hence, the attacker has the full control of the attack window. A series of works focused on methods to predict/eliminate randomized UDP ports and TXIDs (e.g., port brute-forcing [46], birthday attacks [85, 86], IP fragmentation [39, 96], and ICMP-based side channels [65, 66]). When MAGINOTDNS is operated under the off-path setting, we leverage the techniques proposed by these prior works, but the new vulnerabilities discovered by us make MAGINOTDNS more powerful (these prior works cannot poison an arbitrary zone).

4 Systematic Analysis of Bailiwick Checking

In this section, we first present a systematic analysis of the bailiwick checking implementations of popular DNS software. Then we explain how they become vulnerable.

Specifically, we collected a list of DNS software that are also analyzed by previous research [44, 57, 58, 65, 96]. We discard the software which is out-of-date, provides no download links, or cannot function as CDNS (shown in Table 9 of Appendix B). In the end, we narrow down the list to 9 DNS software, as shown in Table 10. Each software has tens or hundreds of thousands of SLOC (Source lines of code), and it took about 3 weeks to analyze them all together. Among the 9 software, 7 have published source code, so we conducted source code review on the files that are relevant to DNS resolution and caching. In addition to code analysis, we also conducted dynamic analysis using CLion [21] and GDB [37], and inspected the software's runtime behaviors and logs. The closed-source software are Microsoft DNS and Simple DNS Plus, and we analyze them through experiments described in Section 5, and by reading their official documents [73, 87]. Next, we describe the high-level logic of bailiwick checking that is summarized from RFCs and DNS software and the differences between their implementations.

4.1 Implementations of Bailiwick Checking

Basic algorithm. Bailiwick checking has been tightly integrated into the DNS resolution process rather than working as a stand-alone component [69, 70]. In Section 2.3, we summarize the high-level guideline offered by several RFCs [35, 43, 69, 70]. In Algorithm 1, we describe the basic DNS resolution procedure and describe bailiwick checking that is embedded in three functions. The detailed explanation of each function and data structure of Algorithm 1 is given in Table 11 and Table 12 from Appendix C.

(*I*) *InitQuery*. After receiving a DNS request from a client, the DNS server transforms it into a query Q that will be issued against the upstream servers and keep three key variables about Q for the later bailiwick checking, including $Q.name$ (query name), $Q.type$ (query type), and $Q.zone$ (current query zone that is owned by nameservers). $Q.zone$ is used to measure how "close" the nameserver is to $Q.name$. According to RFCs [69, 70], resolvers are instructed to query the closest server to get answers. Consequently, $Q.zone$ is initialized with the name of the *closest* `NS` record in cache. For example, if $Q.name$ were `example.com`, and `NS` records of `example.com` and `.com` were cached, resolvers will ask nameservers of `example.com` for answers and mark `example.com` as $Q.zone$. If `NS` records of `example.com` expire, resolvers request nameservers of `.com` and set $Q.zone$ with `.com`. If none of them exists, $Q.zone$ will be the root ".". However, as we found through code analysis, forwarders measure the closeness and modify $Q.zone$ in various ways.

(*II*) *SanitizeRecords*. When the resolver receives a response R matching Q, it will sanitize R with bailiwick rules on every resource record (denoted *RR*) of each section. As for a record in the answer section, if $RR.name \neq Q.name$ or $RR.type \neq Q.type$ (except `CNAME`), the response will be ignored. For a record in the authority section, if $RR.name \in Q.zone$ and $Q.name \in RR.name$ ($\in$ means same as or under), it will be identified as a valid delegation record that is close to the answer. Record from the additional section is trustworthy only if $RR.name \in Q.zone$ and it is a glue record (explained in Section Section 2.1) related to the authority section.

(*III*) *UpdateQuery*. After sanitization, the resolver classifies R into either an answer, a referral (explained in Section 2.1), or a `CNAME`[2]. If R is a referral, the resolver will update $Q.zone$ to the name of `NS` records from the referral and continue recursive queries that are sent to the referred servers. If R is about `CNAME` records, the resolver chases the CNAME chain [16] and validate their authenticity. Otherwise, the resolver returns records from the answer section to clients.

Software implementations. We summarize the implementation differences related to bailiwick checking in Table 2 based on our analysis of DNS software. In summary, we find that bailiwick checking under the recursive mode is implemented similarly and robustly. All DNS software carefully update $Q.zone$ with *name* of the closest `NS` records in cache for Q, consequently rejecting all out-of-bailiwick records whose $name \notin Q.zone$. Then authentic in-bailiwick records are cached and shared between the recursive resolver and the forwarder. However, there exist prominent differences under the forwarding mode. First, when $Q.zone$ is initialized in *InitQuery*:

- **BIND** uses the closest `NS` name in the cache to initialize $Q.zone$ for the default forwarding-first mode while using the closest forwarding zone in Z_F for forwarding-only mode (explained in Section 2.2). This design seems reasonable at first glance, but in reality, the closest `NS` often points to its parent zone or even the root when there is no previous cache for $Q.name$.
- **Knot Resolver**, **Microsoft DNS**, and **Technitium** simply initialize $Q.zone$ with the root ".".
- **Unbound**, **PowerDNS Recursor**, and **Simple DNS Plus** assign the closest forwarding zone in Z_F to $Q.zone$ and just trust answers from that zone. When falling back to recursive queries (fallback is explained in Section 2.2), $Q.zone$ of Unbound is again updated to the closest `NS` name in the cache.
- **MaraDNS** takes a different approach to check the response instead of following bailiwick rules, thus $Q.zone$ is not considered.
- **CoreDNS** does not perform bailiwick checking and $Q.zone$ is not implemented.

Second, after *SanitizeRecords*, Knot Resolver does not stash the additional records when running in forwarding mode. MaraDNS only stores the authority section from a referral. Technitium keeps the authority and additional sections only if they are from a referral. The other software cache all in-bailiwick records.

Third, we also found that the resolver can select a `CNAME` record from all the `CNAME` records embedded in R (during *UpdateQuery*) and query the closest server in the cache, but the implementations differ. BIND, Unbound, MaraDNS, and Simple DNS Plus use the first `CNAME` record to issue the following query Q, while Knot Resolver and PowerDNS Recursor use the last `CNAME` record. Microsoft DNS selects a random `CNAME` record to lookup. Querying the server referred by CNAME leads to another vulnerability described in Section 5.1. Noticeably, a CNAME chain should be verified (explained in [35, 93, 96]) to prune the forged CNAME relations that might be embedded in R. We found the latest versions of the analyzed DNS software all have the correct implementations, which foil previous attacks targeting CNAME chain [96]. Our attack exploits a different issue related to the CNAME chasing.

[2] We only focus on basic query types: `A`, `NS`, and `CNAME` and ignore others.

Table 2: Implementation differences of bailiwick checking by mainstream DNS software.

Key functionality	BIND	Knot Resolver	Unbound	PowerDNS Recursor	Microsoft DNS	Technitium	Simple DNS Plus	MaraDNS
Qry.zone (recursive)	closest `NS` name							-[2]
Qry.zone (forwarding)	closest `NS` name	root	forwarding zone	forwarding zone	root	root	forwarding zone	-
Chase CNAME chains	first	last	first	last	random one	first	first	first
Remove out-of-bailiwick RRs	✓	✓	✓	✓	✓	✓	✓	-
Cache AN matching query	✓	✓	✓	✓	✓	✓	✓	✓
Cache NS from a referral	✓	✓	✓	✓	✓	✓	✓	✓
Cache NS from an answer	✓	✓	✓	✓	✓	✗	✓	✗
Cache AR from a referral	✓	✓[1]	✓	✓	✓	✓	✓	✗
Cache AR from an answer	✓	✓[1]	✓	✓	✓	✗	✓	✗
Send NS or AR to clients	✗	✗	✗	✗	✗	✗	✗	✗
Vulnerable	✓	✓	✗	✗	✓	✓	✗	✗

AN: answer section. NS: authority section. AR: additional section. [1] Not cache if forwarding. [2] '-' means not applicable.
CoreDNS behaves same with Unbound for recursive while no bailiwick checking and caching records by packet for forwarding.

4.2 Vulnerability in Query Zone Initialization

As described above, when BIND, Knot Resolver, Microsoft DNS, and Technitium act in the forwarding mode, they mistakenly initialize *Q.zone* with the ancestor zone of *Q.name* (for BIND it is when a record is not cached) or even the root "." (Knot Resolver, Microsoft DNS, and Technitium). Such behavior seems to manifest in DNS software from a long time ago: we analyzed the first version of BIND9 (BIND 9.0.0 [2]) released in 2000 and found the same issue.

This leads to a vulnerability (denoted $V1$) that records of *any* domain name in the authority or additional section from R will be considered to be *"in-bailiwick"* (i.e., satisfying $RR.name \in Q.zone$ during *SanitizeRecords*) and *cacheable*. According to RFCs [42], delegation (explained in Section 2.1) is allowed only when a parent zone provides nameservers for its child domains. However, leveraging $V1$, *we could inject delegation records of any domain name into the authority section of R, compromise any domain's* `NS` *records, and poison CDNS' global cache.* As such, even TLDs like `.com` and `.net`, and all subdomains under them can be hijacked. For example, provided $d_{attack} \in Z_F$ (e.g., `attacker.com`), if a vulnerable CDNS receives a response like Figure 5(a) when querying d_{attack}, it will use NS_{attack} (`ns1.rogue-tld-ns.org`) to resolve all subdomains under `.com` in the future.

5 End-to-End Attacks under MaginotDNS

In this section, we develop end-to-end cache poisoning attacks exploiting $V1$ against the 4 vulnerable DNS software (BIND, Knot Resolver, Microsoft DNS, and Technitium) and evaluate them in a controlled environment. We first describe the concrete steps for our cache poisoning attack and 2 other vulnerabilities ($V2$ and $V3$) that we discover about the forwarder mode. Then, we elaborate on our attack methodology and evaluation results under the on-path and off-path settings. We present attack demos and details at [9].

Experiments setup. We install BIND, Knot Resolver, and Technitium on a machine with Ubuntu 20.04 as the host OS and Microsoft DNS on another machine with Windows Server 2022 as the host OS. All software are configured as CDNSes. Then we configure d_{attack} (e.g., `attacker.com` for anonymity) as a forwarding zone in Z_F pointed to our authoritative server (outside the LAN) for these CDNSes and leave all other domains under Z_R. There is one attacker machine and another machine holding NS_{attack} (`ns1.rogue-tld-ns.org`), and they are in the same LAN as the victim CDNSes. Then we run 3-5 clients from other machines to use the CDNSes simultaneously. Since we assign a public IP address to our CDNS machine, it is sometimes accessed by unknown DNS clients, which makes our setting more realistic.

5.1 Attack Design

In Section 3.1, we give a high-level overview of the attack workflow. Below we describe the concrete steps, which are illustrated in Figure 6.

Before the actual attack, the attacker needs to collect information, including CDNS' egress IP addresses (termed IP_c) and upstream DNS server's ingress IP addresses (termed IP_u)[3]. When the attacker uses the CDNS to query his/her owned domain[4], IP_c and IP_u can be learned on the authoritative server. If $IP_c \cap IP_u \neq \emptyset$, the attacker launches an on-path attack. Otherwise, the attacker launches an off-path attack.

The actual attack starts by asking the resolver to issue a Q (Q_{fd} or Q_{fu}) for d_{attack} (steps ❶ & ❷ & ❸). Under the on-path setting, the attacker generates a R_{attack} in Figure 5 and sends it back directly (step ❹). Otherwise, under the off-path setting, the attacker needs to guess the source port and TXID of Q_{fu} to create a valid R_{attack} because the upstream resolver will ignore any out-of-bailiwick responses. Since R_{attack} has to

[3] We consider a simple upstream DNS server model with the same ingress and egress IP address because it is pre-configured into the profile.
[4] d_{attack} in Z_F and another registered domain in Z_R.

Flags: QR AA RD;	Flags: QR AA RD;	Flags: QR AA RD;	Flags: QR AA RD;	Flags: QR AA RD;
Question section: attacker.com. A	Question section: attacker.com. NS	Question section: com. NS	Question section: attacker.com. A	Question section: attacker.com. A
Answer section: attacker.com. A a.t.k.r	Answer section: attacker.com. CNAME com.	Answer section: com. NS ns1.rogue-tld-ns.org.	Answer section: (Empty)	Answer section: attacker.com. A a.t.k.r
Authority section: com. NS ns1.rogue-tld-ns.org.	Authority section: (Empty)	Authority section: (Empty)	Authority section: com. NS ns1.rogue-tld-ns.org.	Authority section: (Empty)
Additional section: ns1.rogue-tld-ns.org. A a.t.k.r	Additional section: (Empty)	Additional section: ns1.rogue-tld-ns.org. A a.t.k.r	Additional section: ns1.rogue-tld-ns.org. A a.t.k.r	Additional section: (Empty)
(a)	(b)	(c)	(d)	(e)

Figure 5: DNS responses used for poisoning the delegation data of .com.

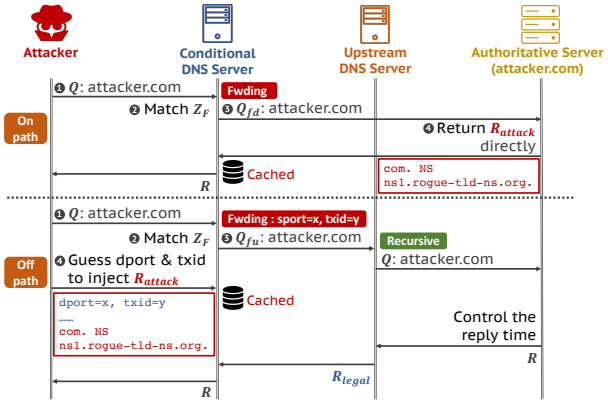

Figure 6: Cache poisoning steps of on-/off-path settings.

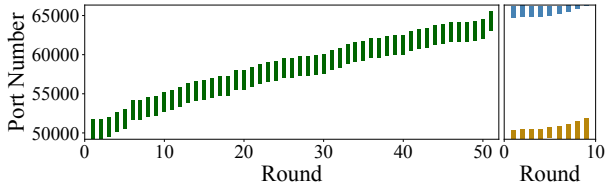

Figure 7: Microsoft DNS source port distribution. We test all 65,535 ports for 60 rounds and show the opened ports. The rounds are sorted by the lowest port. Windows server (and DNS server) restarts in the beginning of each round. We encounter two types of distribution: (*i*) one consecutive range (51 rounds on the left) and (*ii*) two disjoint ranges (the remaining 9 rounds on the right) with an upper-bound of *65534* and lower-bound of *49152*.

compete with the original response R_{legal}, the attacker needs to control the reply time to make R_{attack} arrive earlier than R_{legal} and uses a spoofed IP from IP_u (step ❹). Another extra step under the off-path setting is to send a valid packet before the resolver timeouts on Q_{fu} when bruteforcing the ports and TXID to avoid forwarding fallback (see Section 2.2). The successful attack will force the target resolver to cache R_{attack}.

Other vulnerabilities. In addition to the bailiwick vulnerability $V1$, we also found 2 new vulnerabilities that facilitate our cache poisoning attacks, denoted as $V2$ and $V3$.

$V2$: Flawed CNAME chasing. As described in Section 4.1, a resolver selects a CNAME record in the CNAME chain to issue the next query, which can be exploited against Knot Resolver. Under CNAME chasing, most resolvers issue a new query for the alias name targeting the closest nameservers in the cache. However, under forwarding mode, Knot Resolver issues queries for the alias name to the same upstream server (as in the previous forwarding query), even when the CNAME record points to a different domain. For example, given $d_{attack} \in Z_F$ and $d_{victim} \in Z_R$, and $server_{attack}$ is attacker's authoritative server for d_{attack}. If Knot Resolver receives a CNAME response pointing d_{attack} to d_{victim}, it will launch a new query for records of d_{victim} towards $server_{attack}$, instead of other name servers close to d_{victim}. Attackers holding d_{attack} could poison any domain just by pointing d_{attack} to them and return a rogue response on his/her authoritative server.

$V3$: Insufficient source port randomization. We observe that Microsoft DNS uses a small source port list (only *2,500* ports) to issue *forwarding/recursive* DNS queries by inspecting Windows Server's Resource Monitor and conducting traffic analysis (we send 65,535 different queries in each round). The distribution of ports are shown in Figure 7. Both IPv4 and IPv6 share the same implementation, but the used port lists are different. Surprisingly, once Microsoft DNS starts, the port list becomes constant, which means that attackers could scan ports in advance to determine which port range is in use. Furthermore, scanning the port range from *49152* to *65534* with a global ICMP error rate limit of 200 [65] can be finished in several minutes or, at worst several hours. Besides, if the Windows Server's firewall is shut down, Microsoft DNS does not check the source IP address of a response provided that the source port and TXID match the query.

5.2 On-path Attack

We assume the CDNS issues Q_{fd} for d_{attack} directly to attacker's authoritative server. We test whether the cached .com NS record can be poisoned.

Attacking BIND and Microsoft DNS. Figure 5 shows examples of the rogue response R_{attack}. To overwrite cached .com NS record, we need to inject some records with a higher rank in data trust levels. According to BIND (Table 7 in Appendix), cached NS records obtained from recursive queries have a rank of dns_trust_glue=4. Rogue NS records of R_{attack} in Figure 5(a) with an AA flag in DNS header (explained in Section 2.1) has a rank of dns_trust_authauthority=6, which can overwrite the TLD cache.

For Microsoft DNS, R_{attack} in Figure 5(a) could overwrite any cached NS records, including those from NS queries.

Attacking Knot Resolver. Attacking Knot is more challenging, as NS records in the authority section are always delegated with a trust level of KR_RANK_TRY=2 (Table 8 in Appendix), which rejects the cache overwriting attempt from the response of Figure 5(a). Hence, we leverage *V2* to inject rogue DNS data. Specifically, we query the NS records of d_{attack} and return a CNAME response in Figure 5(b). Then Knot will follow the CNAME chain and query .com's NS records towards our authoritative server. Eventually, records in Figure 5(c) with an AA flag of a trust level KR_RANK_AUTH=16 will be cached.
Attacking Technitium. Technitium only cache rogue NS records from a referral. Thus, to attack Technitium, we remove the answer section and use R_{attack} in Figure 5(d).
Result summary. All the tested resolvers can be poisoned successfully and use the rogue nameserver NS_{attack} to resolve any domain under .com. Different from the off-path attack that needs to guess the random numbers like port and TXID, the consequences of our on-path attack are deterministic.

5.3 Off-path Attack

We forward the d_{attack} zone on the resolvers to a public DNS resolver 1.1.1.1 for the off-path attack. Then we leverage two different methods to attack Microsoft DNS and BIND separately. We did not attack Knot Resolver and Technitium because they are configured with 0x20 protection [77, 89] by default (see Table 1). In addition, a mismatch on any field causes Knot Resolver to ignore incoming UDP responses and re-query over TCP, which makes port and TXID guessing impractical. We also try to poison the cached .com NS record. The attack payload is similar to Figure 5(a).
Attacking Microsoft DNS with brute-force guessing. Before the attack, we scan the ports of the Microsoft DNS resolver to locate the possible in-use DNS ports. Due to *V3*, there is no need to scan all 65,535 ports. But still, because Microsoft DNS listens to all the opened 2,500 ports, scanning all of them is inefficient. Therefore, we choose to guess a small port range, e.g., 20 ports based on the port distribution (see Figure 7), and bruteforce 65,536 TXIDs on these ports. Though in each attack round, the success rate is only $20/2,500$, we could attempt many times to increase the success rate by exploiting "birthday paradox" [85]. For example, the success rate could rise to 99.7% after 720 attack rounds (see the equation below).

$$1-[(2,500-20)/2,500]^{720}=99.7\% \quad (1)$$

What makes the attack conditions more favored to us is that Microsoft DNS has a default forwarding timeout of *5s*, which leaves a sufficiently large time window for attacks.

• *Attack process.* In each round (in total 5s), (*i*) we send a query asking for A records of {nonce}.attacker.com where nonce is a random value used to bypass caching. (*ii*) Our authoritative server returns a valid response in Figure 5(e) at the end of timeout to extend the attack window. (*iii*) We inject $20*65,536$ forged packets to enumerate the combinations of 20 ports and 65,536 TXIDs against the target resolver. Specially, we remove the additional section from responses in Figure 5(a) to reduce the packet size. (*iv*) Finally, we issue a new query to check whether the rogue NS records of .com are cached or not.

• *Attack results.* We run the experiment 20 times. As shown in Table 3, we achieved a *100%* success rate with an average of *802s* to succeed. Among the runs, the maximum time till success is 2,475s, and the minimum is only 10s. During attack, the target resolver received $\leq$ 216 Mbps inbound attack traffic. To further reduce the attack traffic bandwidth, we can decrease the number of guessed ports in each round, which leads to more attack rounds and longer time costs.

Table 3: Microsoft DNS and BIND off-path attack results.

Software	Time of each round	Avg time taken	Max traffic rate	Success rate
MS DNS	5s	802s	216Mbps	20/20
BIND	1.2s	790s	54Mbps	20/20

Attack BIND with SAD DNS [65]. Through source code review, we find that BIND has a forwarding timeout of *1.2s* and an initial recursion timeout of *0.8s*. In addition, BIND uses the default port range of the host OS, e.g., from *32768* to *60999* with a total number of *28,232* ports in Linux [61]. As such, direct brute-forcing is not a feasible option. We solve this challenge by adjusting SAD DNS to attack BIND. Because BIND uses connect() on the UDP socket, we leverage the private source port scan method from [65] and assign 16 IP addresses using the DHCP strategy to our attack machine. Besides, Appendix D describes the mechanisms of SAD DNS in details.

• *Attack process.* In each round, (*i*) We choose to guess only 50 ports at a round because the attack window is just 1.2s. (*ii*) We reply with a valid response in Figure 5(e) at the end of a timeout to prevent fallback queries. If fallback queries occur, BIND will cache the NS records of d_{attack}, and then the attacker needs to wait for the cache timeout or leverage another domain to launch attacks. (*iii*) After identifying a candidate in-use port, we bruteforce 65,536 TXIDs targeting the port. (*iv*) We test whether the rogue .com NS records are cached. Under the birthday paradox, the success rate increases after many rounds: e.g., 99.8% after 3,600 rounds (see the equation below).

$$1-[(28,232-50)/28,232]^{3600}=99.8\% \quad (2)$$

• *Attack results.* We run the attack 20 times and achieved 100% success rate, with an average runtime of *790s* (ranging from 50s to 2,269s). The maximum traffic volume to the resolver is about 54Mbps during attack. We list the result in Table 3.

6 Finding Vulnerable CDNSes at Large

In this section, we propose a methodology that finds CDNSes vulnerable to MAGINOTDNS in open networks. We also conduct interviews with several ISP operators to gain better understanding of CDNSes.

6.1 CDNS Probing Methodology

Technical challenges. CDNSes differ from the regular forwarders and recursive resolvers in that *both* Z_F and Z_R are *non-empty*. Leveraging this observation, we can confirm a target server is a CDNS, if we find it forwards queries for some domains and recursively resolves some other domains. However, neither CDNSes nor regular DNS servers tell in their responses about how the corresponding queries are resolved backstage, plus only operators know the pre-configured lists of Z_F and Z_R. An attacker may observe from authoritative servers how his/her domain d_{attack} is resolved, but lacks vantage points over other domains to confirm that the target server is a CDNS. As a result, identifying CDNSes poses a significant challenge.

Identifying CDNSes with cache probing. We find solving the challenge is possible through a side-channel analysis from DNS clients: DNS servers cache *"intermediate"* data in *referrals* during recursive resolution of domains, but not so during forwarding because no referrals are ever provided. After the resolution of a domain, we can use cache probing methods (i.e., *non-recursive* DNS queries that ask the server only to find answers locally [38]) to reveal whether the data is cached.

As shown in Figure 8, our controlled DNS client first issues a query for the address (type A) of an SLD d to the target server (steps ① and ❶). During recursive resolution (steps ❷ to ❽), the authoritative server data (type NS) of d will be returned through referrals and cached (steps ❸ and ❺). By contrast, if the query is forwarded (steps ② to ⑤), only type A RRSets of d are returned from upstream servers and no authoritative server data is cached. Finally, the DNS client issues a *non-recursive* cache probing query for the authoritative server data (type NS) of d (steps ⑥ and ❾). If the response is empty (step ⑦) then the previous query for d was forwarded and $d \in Z_F$, otherwise (step ❿) recursively resolved and $d \in Z_R$.

Though the pre-configured DNS zones are not known to us, we only need to probe a subset of Z_F and Z_R to confirm a CDNS (the probed subsets are termed as P_F and P_R). To this end, we select the domains from Alexa Top 10K sites [3] and keep 9,429 SLDs with both type NS and type A RRSets (they are both required by our probing methodology). We also test a newly-registered domain d_{custom}, in order to distinguish between the recursive-default CDNSes (if $d_{custom} \in P_R$) and the forwarding-default CDNSes (if $d_{custom} \in P_F$), because we expect that d_{custom} falls in the default list of DNS zones.

Finding vulnerable CDNSes with software versions. For all discovered CDNSes, we first leverage the `version.bind` DNS probing method [11] to reveal their software versions. If the responses show nothing, we further use `fpdns` [32], a community-sponsored tool that identifies software information from fingerprints in DNS responses. The tool is built with fingerprints for a wide range of DNS software, including BIND (versions 9, 8, and older) and Microsoft DNS (versions NT to 2008). For BIND servers, because the off-path type of MAGINOTDNS relies on SAD DNS and requires specific OSes, we also probe their OS fingerprints with `Nmap` [71].

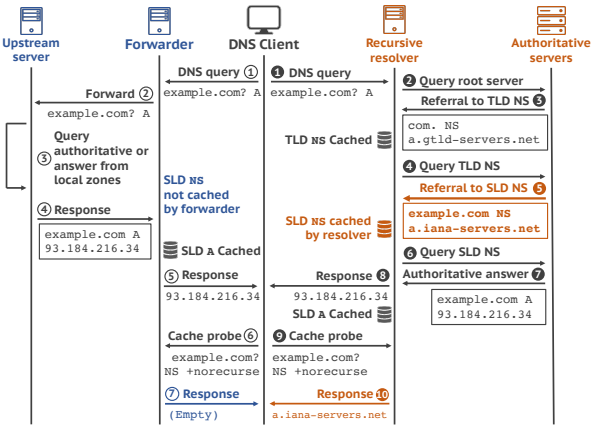

Figure 8: Inferring backstage DNS resolution from cache.

Practical considerations. Our probing methodology works under three requirements:

(*i*) Upstream servers do not stuff type NS RRSets in responses to type A queries (step ④, creating a side channel).

(*ii*) DNS servers follow non-recursive flags in cache probing queries (step ⑥ and ❾) and only find answers locally.

(*iii*) The initial DNS query (steps ① and ❶) and the cache probing request (step ⑥ and ❾) hit the same piece of DNS cache. This is not true if a server IP has multiple backend servers for load balancing, e.g., large public DNS services.

To evaluate whether (*i*) is fulfilled, we perform field tests on all DNS software in Table 1 and public DNS services in Table 13 in Appendix E, which commonly serve as upstream servers. We find that all tested DNS software and most (13 of 16) public DNS services follow this requirement.

For servers that do not satisfy (*ii*) and (*iii*), we identify and remove them from the later analysis. Specifically, we register a domain d_{custom} and set up its authoritative server. For a target DNS server, we first issue non-recursive queries for d_{custom} and remove it if our authoritative server receives subsequent queries (i.e., not satisfying (*ii*)). We then send the server 5 identical DNS queries for d_{custom} at 1 QPS consecutively and remove it if TTLs in the responses do not decrease by one (i.e., not satisfying (*iii*)).

Limitations. We acknowledge that we may underestimate the population of vulnerable CDNSes in the wild, due to limitations of software fingerprinting tools (e.g., DNS servers may ignore `version.bind` probes and `fpdns` lacks fingerprints for Knot Resolver) and cache-probing method (e.g., we

Table 4: Open DNS servers and CDNS statistics.

DNS Server Type	# IP	% of		
		Probed	CDNS	Vuln.
DNS servers on Feb. 14, 2022	1,499,110	–	–	–
DNS servers alive on Mar. 14, 2022	**1,215,918**	–	–	–
– Not following non-recursive	839,017	–	–	–
– Using multiple caches	401,186	–	–	–
– Supports cache-probing	**370,512**	**100%**	–	–
– Version identifiable	237,835	64.2%	–	–
– DNSSEC validation	86,955	23.5%	–	–
– 0x20 encoding	1,619	0.4%	–	–
CDNSes identified by probing	**154,955**	**41.8%**	**100%**	–
– Version identifiable (in CDNS)	117,306	31.7%	75.7%	–
– by `version.bind`	59,419	16.0%	38.3%	–
– by `fpdns`	57,887	15.6%	37.4%	–
– OS identified for BIND (in CDNS)	19,995	5.4%	12.9%	–
– DNSSEC validation (in CDNS)	34,424	9.3%	22.2%	–
– 0x20 encoding (in CDNS)	1,119	0.3%	0.7%	–
Vulnerable CDNSes	**54,949**	**14.8%**	**35.5%**	**100%**
– On-path attack possible*	**54,949**	**14.8%**	**35.5%**	**100%**
– BIND	24,287	6.6%	15.7%	44.2%
– Microsoft DNS	30,662	8.3%	19.8%	55.8%
– Off-path attack possible*	**48,539**	**13.1%**	**31.3%**	**88.3%**
– BIND (OS exploitable)	17,877	4.8%	11.5%	32.5%
– Microsoft DNS	30,662	8.3%	19.8%	55.8%
– Recursive-default	10,445	5.0%	11.9%	33.4%
– Forwarding-default	36,581	9.9%	23.6%	66.6%

* On-/Off-path attack possible: CDNSes equipped with non-empty Z_F and vulnerable software versions/OSes. Because we lack vantage between CDNSes and upstream servers, we can only confirm they are vulnerable to on-/off-path attacks, but cannot further identify which domains in Z_F can be actually exploited by each type of attack.

cannot probe servers that do not follow non-recursive flags). However, as shown in the following sections, we are still able to discover over 54,949 open CDNSes vulnerable to MAGINOTDNS, and our results reflect the lower bound.

6.2 CDNS Probing Results

Open CDNSes. Studies have reported significant churn of open DNS server IPs because most operate on home-owned equipment [19]. We do not expect them to be configured as CDNS and focus on servers that are more stable. To this end, we scanned the entire IPv4 address space for open DNS servers on Feb. 14 and Mar. 14, 2022 using `XMap` [59]. We consider 1,215,918 open DNS servers that appear in both scans (i.e., servers that are up for at least one month).

Our probing of open DNS resolvers for CDNSes started from Mar. 14 to Apr. 13, 2022 (one month). Table 4 shows the statistics of all tested open DNS servers. Using the cache-probing method, we probed 370,512 DNS servers (in 224 countries and 14,212 ASes[5]) satisfying the requirement (*i*)-(*iii*) and confirm **154,955 CDNSes** (41.8% of probed, in 211 countries and 8,900 ASes). As CDNSes can defend against MAGINOTDNS by DNSSEC validation and 0x20 encoding, we filter the CDNSes that these defenses are enabled. To identify servers that enable DNSSEC validation, we follow practices in [20] and deliberately attach bogus signatures to RRSets of our test domain d_{custom}, and query each CDNS for this domain. If the response shows a `SERVFAIL` error, which suggests the server validates the DNSSEC signatures and finds it to be bogus, the server is not vulnerable to MAGINOTDNS. Meanwhile, if the query to our authoritative server carries mixed-case domains, the server enables 0x20 encoding and is not vulnerable to the off-path type of MAGINOTDNS.

[5] Identified using free GeoIP databases [28].

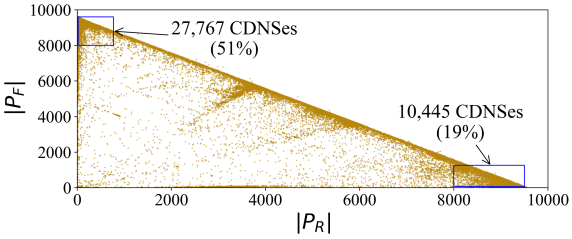

Figure 9: Vulnerable open CDNSes associated with their $|P_F|$ and $|P_R|$. Each dot represents one CDNS.

In the end, we identify **54,949 vulnerable CDNSes** (14.8% of all probed DNS servers and 35.5% of all CDNSes, in 198 countries and 5,839 ASes). All of them are vulnerable to the on-path attack (because they have non-empty Z_F), and 88.3% are also vulnerable to the off-path attack. Due to limitations of software fingerprinting tools, we only find vulnerable CDNSes that use BIND and Microsoft DNS.

Recall that attackers may leverage domains in Z_F to hijack Z_R, and here we show the domain distribution between both sets in vulnerable CDNSes. In Figure 9, we associate vulnerable open CDNSes with their P_F and P_R. We find that for 19% of vulnerable CDNSes (plotted at the bottom-right corner), over 80% of Alexa Top 10K domains falls in P_R, which can become victims. Interestingly, thousands of vulnerable CDNSes have P_F and P_R of similar sizes (plotted in the middle), which seems surprising because putting thousands of domains in configuration files is a cumbersome task for CDNS operators. However, through interviews with network operators (see Section 6.3), we confirm that the configuration process can be automated and prevalent.

Closed CDNSes in the enterprise networks. Enterprise CDNSes are often deployed to split the resolution of *private* and *public* namespaces. For example, operators put the public DNS root in Z_F and private domains in Z_R, such that queries for public names can be forwarded (e.g., public DNS services like Google [31]), while private names are recursively resolved by querying internal authoritative servers.

To find closed CDNSes, we selected 5 leading Internet companies (see Table 6, company names are anonymized) and contacted their network security department. We asked the operators to run a measurement script that probes the default DNS server from their working computers. For ethical reasons, we reduce the domain list to only 2 public domains and 2 private domains (selected by the operators), and we

Table 5: Interview results of CDNS use cases in ISP networks.

Alias			Purpose	Policy			Operation		
ISP	**DSP**	**CDNS**		**Usage**	**Upstream server type**	**Z_F type**	**$\|Z_F\|$**	**Config**	**Query volume**
$I1$	$D1$	$C1$	Access control	Shared	Private authoritative server	Private domain	20+	Manual	-[1]
	$D2$	$C2$	Query fall-backing	Individual	Public DNS service (Google)	Public domain	2	Manual	-
$I2$	$D3$	$C3$	Content caching	Shared	Third-party "authoritative" servers	Public domain	7k+	API	2.5k+ QPS

[1] "-" means the data is not disclosed.

Table 6: Closed CDNSes in enterprise networks.

Enterprise	# Employee	Private domain	Public domain	CDNS
$E1$	100k+	Z_R	Z_F	✓
$E2$	15k+	Z_R	Z_F	✓
$E3$	5k+	Z_R	Z_F	✓
$E4$	200k+	Z_R	Z_R	✗
$E5$	80K+	Z_R	Z_R	✗

report our findings to their DNS operators if vulnerabilities are identified. We also tried to identify their software versions but were unsuccessful. In the end, we find that CDNSes in 3 enterprises are potentially vulnerable to MAGINOTDNS, and their operators confirmed the threat model.

6.3 In-Depth Understanding of CDNS

Though our cache probing method reveals the potentially vulnerable CDNSes, we still lack a good understanding about why CDNS is set up and how it chooses to forward queries. One may argue that the on-path attack has small impact because it might be uncommon that CDNSes forward queries directly to authoritative servers. To gain an in-depth understanding about CDNS, we conducted interviews with 2 major ISPs. Our interviews consist of 3 major questions:

Q1: (Purpose) "For what purpose do you deploy CDNSes in your network?"

Q2: (Policy) "What policies/configurations are enforced by your CDNSes?" CDNS policies include types of domains in Z_F (i.e., potential d_{attack}), destination of forwarded queries (i.e., upstream servers), and whether multiple CDNSes in one ISP network share the same configurations.

Q3: (Operation) "What are the operational statuses of CDNSes?" Considerations include the DNS query volume and how to change configuration files.

Network operators of 2 leading ISPs ($I1$ and $I2$, names anonymized) accepted our interview through online meetings during March and May 2022. The 2 ISPs serve a total of 700 million subscribers. The ISPs outsource their DNS services to 6 DNS service providers (e.g., vendors of DNS services, called *DSPs* for short), and we also interviewed all DSPs.

Results of interviews. Below we summarize answers from ISPs and DSPs to our questions. In Table 5, we also provide information of 3 typical CDNSes ($C1$, $C2$, and $C3$ supported by 3 DSPs $D1$, $D2$, and $D3$) deployed in their networks. $D4$, $D5$, and $D6$ did not describe their specific CDNS use cases.

Q1: Purpose. Both ISPs $I1$ and $I2$ reported a wide range of CDNS use cases in their regional networks. Purposes of deploying CDNSes include: *(i)* access control (e.g., $C1$ splits resolution of private domains and public domains); *(ii)* query fallback (e.g., $C2$ forwards failed queries to public DNS services); *(iii)* content caching (e.g., $C3$ forwards queries about video streaming domains to third-party servers, which claim authority for the domains and resolve them to local cache servers to enhance performance); *(iv)* others, including load-balancing and sinkholing (forwarding queries about malicious domains to sinkhole servers).

Q2: Policy. CDNS policies in one ISP network can be shared among servers ($C1$ and $C3$) or customized for one server only ($C2$). Queries for domains in Z_F are usually forwarded to 3 types of upstream servers: (*i*) closed authoritative nameservers that resolve private domain names ($C1$); (*ii*) public DNS services ($C2$); *(iii)* third-party servers that claim authority for certain types of domains ($C3$).

Q3: Operation. $C1$ forwards 20+ private domains to their internal nameservers, and $C2$ forwards queries of 2 public domains to Google Public DNS (8.8.8.8) due to DNS resolution failure. Because Z_F is small, the configuration files of $C1$ and $C2$ are edited manually by their network operators (similar to Figure 3). By contrast, $C3$ forwards queries for 7k+ public domains directly to third-party "authoritative" servers. Its configuration file is compiled automatically via APIs and can be changed dynamically according to different network conditions. DNS query volumes for domains in Z_F of $C3$ can reach 2.5k+ QPS.

Real-world attacks and impacts. After the interviews, we also discussed the possibility of launching MAGINOTDNS attacks with the ISPs and DSPs. Surprisingly, operators of $C3$ reported evidence of on-path attacks in their networks. From the DNS traffic, they observe *"Third-party authoritative nameservers return rogue* `NS` *records in DNS responses and trick ISP recursive resolvers (C3) into accepting fake nameservers of domains not forwarded to them"*. The websites hosted on those non-forwarded domains are manipulated to insert advertisement pages by them.

Though both $C1$ and $C2$ accept out-of-bailiwick responses on their forwarding modes, $C1$ is less likely to be exploited because private authoritative servers are owned by $I1$ itself. $C2$ is exploitable by off-path attackers targeting forwarded public domains.

7 Discussion

Ethical considerations. All vulnerability analysis and end-to-end validation of MAGINOTDNS are performed in controlled environments. To find vulnerable CDNSes in open networks, we only perform cache probing and collect software version information, and *do not* launch real cache poisoning attacks against them. And we follow the ethical principles of Menlo Report [47] and common practices of measurement studies about DNS servers [19, 56]. When probing Alexa Top 10K domains, we strictly limit our DNS query rate (at below 1 QPS per server) to avoid interrupting their normal services. We also configure reverse DNS records for our client IP addresses with our cache probing experiment information and did not receive any complaints during the measurement period. For all of our interviewed ISPs and enterprises, we receive their consent to report our survey results using anonymized information.

Lessons learned. The vulnerabilities reported in this paper uncover two layers of implementation inconsistency: the security checking logic in different DNS software and different server types (i.e., recursive resolvers and forwarders). While bailiwick rules have been documented since 90s[6], no concrete algorithms are ever given by DNS standards. The lack of specific guidelines for such important security measures results in various implementations by different DNS software.

The standards are also unclear about forwarders due to their unique position in the DNS infrastructure, which creates confusion about whether and how to implement the same functionalities as recursive resolvers. For example, $V1$ shows the forwarder mode of some software initializes $Q.zone$ as root ".", possibly because they rely on the security checks of upstream servers [96]. In addition, forwarding only involves one round of message exchange, so forwarders lack sufficient information to update $Q.zone$ because upstream servers provide no context of the domain resolution (e.g., referrals). This might also explain why $V1$ is manifested. As a result, we call for the standardization of bailiwick checking algorithms and implementation agreements upon DNS forwarders.

Mitigation. For vendors of vulnerable DNS software, we recommend updating implementations of *InitQuery* ($V1$). A possible solution is to follow the implementation of software like Unbound, which initializes $Q.zone$ with the closest forwarding zone in Z_F (see Section 4.1). We also recommend DNS software expand the range of source port randomization ($V2$) and fix the flawed `CNAME` chasing algorithms ($V3$).

For resolver operators and domain holders, DNSSEC [8] is recommended, which attaches cryptographic signatures to records resolvers can validate. Responses failing DNSSEC validation are discarded, effectively eliminating all types of cache poisoning attacks, including ours. However, DNSSEC deployment remains unsatisfactory: as we show in Table 4, only 23.5% of probed resolvers enable DNSSEC validation.

[6] Through code review, we confirm that BIND proposed bailiwick checks from version 4.9.2 in 1993 and implemented it in version 4.9.6 in 1997.

Responsible disclosure. We have reported all the discovered vulnerabilities to DNS software vendors, and all of them have confirmed. We received 3 CVE-ids for BIND [23], Technitium [24], and Knot Resolver [25], and were awarded $1,000 by Microsoft Security Response Center. BIND fixed this issue in 9.18.1 [13] (initializing $Q.zone$ with the closest forwarding zone in Z_F), and Technitium fixed it in 7.1 [88] (ignoring `NS` and glue records for forwarding queries). Microsoft DNS will adopt "defense in depth" in its next major release. Knot Resolver proposes to rewrite its internal modules and published a warning of this risk as a temporary solution [79, 80].

8 Other Related Work

DNS cache poisoning attacks. In Section 3.3, we have compared MAGINOTDNS with most related DNS cache poisoning attacks. Here, we provide additional details and other related works. Back in 90s, [34] exploited recursive resolvers that did not perform bailiwick checks, while [81] found many home routers did not validate DNS responses in 2014. After that, the bailiwick rule is well implemented, and DNS cache poisoning attacks aim to inject in-bailiwick records. [40] proposed a method to de-randomize the source port of a resolver behind NAT, and [4] achieved a similar goal by leveraging malware to exhaust the local port on a client machine. Most recently, [48] presented a cross-layer attack by exploring the weakness of the pseudo-random number generator in the Linux kernel. By contrast, IP fragmentation was developed to eliminate the requirements of guessing the source port number [39]. In addition, a number of other approaches are leveraged for DNS cache poisoning attacks, including cache inconsistency [45], domain misinterpretation [44], and domain collision [17, 18].

Bailiwick rule analysis. The existing RFCs only provide a high-level description of bailiwick rules [35, 43, 69, 70]. Until 2010, [85] developed a formal model of the bailiwick rule and the record overwriting mechanism for modern DNS resolvers. They analyzed different types of well-known cache poisoning attacks and tested the attacks against implementations. However, they did not find new vulnerabilities.

9 Conclusion

In this paper, we provide a systematic security analysis against the implementation of bailiwick rules. Exploiting vulnerable bailiwick checking implementations, we propose MAGINOTDNS against CDNS, a powerful cache poisoning attack that allows an adversary to manipulate arbitrary DNS zones. The vulnerability affects the latest version of several popular DNS software, including BIND and Microsoft DNS. Extensive Internet measurements and interviews were conducted to demonstrate the real-world CDNS use cases and attack impacts. Our research calls for an immediate review of the implementation of DNS security principles.

Acknowledgement

We thank all the anonymous reviewers for their valuable comments to improve this paper and all software vendors and our industry partners for their discussion and support. The authors from Tsinghua University were supported by the National Natural Science Foundation of China (U1836213, U19B2034, 62102218, and 62132011). The authors from University of California, Irvine were supported by NSF CNS-2047476 and gifts from Cisco and Microsoft.

References

[1] 1.1.1.1. https://1.1.1.1/dns/, 2022.

[2] BIND 9.0.0. https://ftp.ripe.net/mirrors/sites/ftp.isc.org/isc/bind9/9.0.0/, 2000.

[3] Alexa. https://www.alexa.com/topsites, 2022.

[4] 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*.

[5] 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*.

[6] 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*.

[7] Anthony E. Alvarez. DNS Forwarding and Conditional Forwarding. https://medium.com/tech-jobs-academy/dns-forwarding-and-conditional-forwarding-f3118bc93984, 2016.

[8] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. RFC 4035: Protocol Modifications for the DNS Security Extensions. *RFC Proposed Standard*.

[9] MAGINOTDNS. https://maginotdns.net/, 2022.

[10] Ray Bellis. RFC 5625: DNS Proxy Implementation Guidelines. *RFC Best Current Practice*.

[11] BIND. How do I change the version that BIND reports when queried for version.bind? https://kb.isc.org/docs/aa-00359, 2021.

[12] BIND. https://www.isc.org/bind/, 2022.

[13] BIND. https://kb.isc.org/docs/cve-2021-25220, 2022.

[14] BIND. Source Code Repository. https://gitlab.isc.org/isc-projects/bind9/-/blob/v9_18_0/lib/dns/include/dns/types.h#L302, 2022.

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

[16] CNAME Chasing. https://cloud.google.com/dns/docs/cnamechasing, 2022.

[17] Qi Alfred Chen, Eric Osterweil, Matthew Thomas, and Z. Morley Mao. MitM Attack by Name Collision: Cause Analysis and Vulnerability Assessment in the New gTLD Era. In *S&P '16*.

[18] Qi Alfred Chen, Matthew Thomas, Eric Osterweil, Yulong Cao, Jie You, and Z Morley Mao. Client-side Name Collision Vulnerability in the New gTLD Era: A Systematic Study. In *CCS '17*.

[19] 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*.

[20] Taejoong Chung, Roland Van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. In *USENIX Security '17*.

[21] CLion. https://www.jetbrains.com/, 2022.

[22] CoreDNS. https://coredns.io/, 2022.

[23] CVE-2021-25220. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-25220, 2021.

[24] CVE-2021-43105. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-43105, 2021.

[25] CVE-2022-32983. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-32983, 2022.

[26] Domain Name Relay Daemon. http://dnrd.sourceforge.net/, 2015.

[27] 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*.

[28] MaxMind GeoIP Databases. https://www.maxmind.com/en/geoip2-services-and-databases, 2022.

[29] Dbndns. https://dbpedia.org/page/Dbndns, 2010.

[30] djbdns. https://cr.yp.to/djbdns.html, 2009.

[31] Google Public DNS. https://developers.google.com/speed/public-dns, 2022.

[32] DNS-OARC. fpdns - DNS Fingerprinting Tool. https://www.dns-oarc.net/tools/fpdns, 2021.

[33] Dnsmasq. https://thekelleys.org.uk/dnsmasq/doc.html, 2022.

[34] Michael Dooley and Timothy Rooney. DNS Security Management. In *John Wiley & Sons, Inc., Hoboken, New Jersey*, page 155.

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

[36] FusionLayer. https://www.fusionlayer.com/products/dns-server, 2022.

[37] GDB. https://www.sourceware.org/gdb/, 2022.

[38] Luis Grangeia. Cache snooping or snooping the cache for fun and profit, 2004.

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

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

[41] Amir Herzberg and Haya Shulman. Vulnerable Delegation of DNS Resolution. In *ESORICS '13*.

[42] Paul Hoffman, Andrew Sullivan, and Kazunori Fujiwara. RFC 8499: DNS Terminology. *RFC Best Current Practice*.

[43] Bert Hubert and Remco van Mook. RFC 5452: Measures for Making DNS More Resilient against Forged Answers. *RFC Proposed Standard*.

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

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

[46] Dan Kaminsky. It's The End Of The Cache As We Know It, 2008.

[47] Erin Kenneally and David Dittrich. The menlo report: Ethical principles guiding information and communication technology research. *SSRN Electronic Journal*.

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

[49] Amit Klein. BIND 8 DNS Cache Poisoning, 2007.

[50] Amit Klein. BIND 9 DNS Cache Poisoning, 2007.

[51] Amit Klein. OpenBSD DNS Cache Poisoning and Multiple O/S Predictable IP ID Vulnerability, 2007.

[52] Amit Klein. Windows DNS Server Cache Poisoning, 2007.

[53] Amit Klein. PowerDNS Recursor DNS Cache Poisoning, 2008.

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

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

[56] Maciej Korczyński, Michał Król, and Michel van Eeten. Zone Poisoning: The How and Where of Non-Secure DNS Dynamic Updates. In *IMC '16*.

[57] Hyeonmin Lee, Aniketh Gireesh, Roland van Rijswijk-Deij, Taekyoung "Ted" Kwon, and Taejoong Chung. A Longitudinal and Comprehensive Study of the DANE Ecosystem in Email. In *USENIX Security '20*.

[58] 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*.

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

[60] Jinjin Liang, Jian Jiang, Haixin Duan, Kang Li, and Jianping Wu. Measuring Query Latency of Top Level DNS Servers. In *PAM '13*.

[61] Linux. https://www.kernel.org/doc/html/latest/networking/ip-sysctl.html, 2022.

[62] Unix & Linux. BIND: forwarding zone does not work when allow-recursive not allowed. https://unix.stackexchange.com/questions/633561/bind-forwarding-zone-does-not-work-when-allow-recursive-not-allowed, 2021.

[63] 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*.

[64] Matthew Luckie, Robert Beverly, Ryan Koga, Ken Keys, and Joshua A Kroll. Network Hygiene, Incentives, and Regulation: Deployment of Source Address Validation in the Internet. In *CCS '19*.

[65] 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*.

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

[67] MaraDNS. https://maradns.samiam.org/, 2022.

[68] Microsoft. Forwarders and conditional forwarders resolution timeouts. https://docs.microsoft.com/en-us/troubleshoot/windows-server/networking/forwarders-resolution-timeouts, 2020.

[69] Paul V. Mockapetris. RFC 1034: Domain Names - Concepts and Facilities. *RFC Standard*.

[70] Paul V. Mockapetris. RFC 1035: Domain Names - Implementation and Specification. *RFC Standard*.

[71] Nmap. https://nmap.org/, 2022.

[72] NSD. https://www.nlnetlabs.nl/projects/nsd/about/, 2022.

[73] Simple DNS Plus. https://simpledns.plus/download, 2022.

[74] Posadis. http://posadis.sourceforge.net/, 2005.

[75] PowerDNS. https://www.powerdns.com/, 2022.

[76] RaidenDNS. http://www.raidendnsd.com/eng_download.html, 2007.

[77] Knot Resolver. https://www.knot-resolver.cz/, 2022.

[78] Knot Resolver. Knot Resolver Source Code Repository. https://gitlab.nic.cz/knot/knot-resolver/-/blob/v5.5.0/lib/resolve.h#L95, 2022.

[79] Knot Resolver. New Policy. https://gitlab.nic.cz/knot/knot-resolver/-/compare/master...new-policy, 2022.

[80] Knot Resolver. Policy Docs. https://github.com/CZ-NIC/knot-resolver/commit/ccb9d9794db5eb757c33becf65cb1cf48ecfd968, 2022.

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

[82] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. On measuring the client-side DNS infrastructure. In *IMC '13*.

[83] Christoph Schuba and Eugene H Spafford. Addressing Weaknesses in the Domain Name System Protocol. *Master's thesis*.

[84] Kaiwen Shen, Chuhan Wang, Minglei Guo, Xiaofeng Zheng, Chaoyi Lu, Baojun Liu, Yuxuan Zhao, Shuang Hao, Haixin Duan, Qinfeng Pan, and Min Yang. Weak Links in Authentication Chains: A Large-scale Analysis of Email Sender Spoofing Attacks. In *USENIX Security '21*.

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

[86] Joe Stewart. DNS Cache Poisoning – The Next Generation. *Secureworks*.

[87] Microsoft Domain Name System. https://docs.microsoft.com/en-us/windows-server/networking/dns/dns-top, 2022.

[88] Technitium. https://github.com/TechnitiumSoftware/DnsServer/blob/master/CHANGELOG.md#version-71, 2021.

[89] Technitium. https://technitium.com/dns/, 2022.

[90] Rick Trader. Windows Server – How to configure a Conditional Forwarder in DNS. https://www.interfacett.com/blogs/windows-server-how-to-configure-a-conditional-forwarder-in-dns/, 2016.

[91] Unbound. https://nlnetlabs.nl/projects/unbound/about/, 2022.

[92] Chuhan Wang, Kaiwen Shen, Minglei Guo, Yuxuan Zhao, Mingming Zhang, Jianjun Chen, Baojun Liu, Xiaofeng Zheng, Haixin Duan, Yanzhong Lin, and Qingfeng Pan. A Large-scale and Longitudinal Measurement Study of DKIM Deployment. In *USENIX Security '22*.

[93] Wouter Wijngaards. Draft: Resolver Side Mitigations. *RFC Draft*.

[94] YADIFA. https://www.yadifa.eu/home, 2022.

[95] Yiming Zhang, Baojun Liu, Chaoyi Lu, Zhou Li, Haixin Duan, Jiachen Li, and Zaifeng Zhang. Rusted Anchors: A National Client-Side View of Hidden Root CAs in the Web PKI Ecosystem. In *CCS '21*.

[96] 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*.

Table 7: Trust levels in BIND.

Definition	Level	Description
dns_trust_ultimate	9	This server is authoritative
dns_trust_secure	8	Successfully DNSSEC validated
dns_trust_authanswer	7	Answer from an authoritative server
dns_trust_authauthority	6	Received in the authority section from an authoritative response
dns_trust_answer	5	Answer from a non-authoritative server
dns_trust_glue	4	Received in a referral response
dns_trust_additional	3	Received in the additional section of a response
dns_trust_pending_answer	2	DNSSEC has not been validated
dns_trust_pending_additional	1	DNSSEC has not been validated
dns_trust_none	0	No data should have this trust level

Summarized from the file bind9/lib/dns/include/dns/types.h#L302 [14].

Table 8: Trust levels in Knot Resolver.

Definition	Level	Description
KR_RANK_SECURE	32	Verified trust chain from the closest TA
KR_RANK_AUTH	16	Authoritative data
KR_RANK_INSECURE	8	Proven to be insecure
KR_RANK_MISSING	7	No RRSIG found
KR_RANK_MISMATCH	6	-
KR_RANK_BOGUS	5	Ought to be secure but isn't
KR_RANK_INDET	4	Unable to determine whether secure
KR_RANK_TRY	2	Attempt to validate
KR_RANK_OMIT	1	Do not attempt to validate
KR_RANK_INITIAL	0	Initial-like states

Summarized from the file knot-resolver/lib/resolve.h#L95 [78].

A Data Trust Levels of DNS Software

Table 7 and Table 8 show the data trust levels implemented by BIND and Knot Resolver, which differ from the rankings defined by DNS standards.

B List of Analyzed DNS Software

Table 9 shows all software we surveyed and explains why we do not analyze bailiwick checking algorithms (in Section 4). Table 10 shows our analysis workload on 9 DNS software.

Table 9: List of not analyzed DNS software.

DNS Software	Version	Analyzed	Comments
Posadis [74]	0.60.6	✗	Too old
djbdns [30]	1.05	✗	Too old
dbndns [29]	1.05-8	✗	Too old
RaidenDNS [76]	1.3	✗	Too old
FusionLayer [36]	-	✗	No links
Dnsmasq [33]	2.86	✗	Not CDNS
DNRD [26]	2.20.3	✗	Not CDNS
YADIFA [94]	2.5.4	✗	Not CDNS
NSD [72]	4.5.0	✗	Not CDNS

C DNS Resolution Algorithm and Structures

The overall DNS resolution algorithm is shown in Algorithm 1, and related DNS definitions and functions are listed in Table 11 and Table 12.

Table 10: Analysis workload on DNS software. "# Days" is how long we spent on software analysis.

Software	# Key SLOC	# Files	# Days
BIND	331,926 (C)	1,030	4
Knot Resolver	71,079 (C)	178	3
Unbound	127,601 (C)	283	3
PowerDNS Recursor	78,320 (C++)	265	3
Microsoft DNS	-	-	1
MaraDNS	54,748 (C)	240	3
CoreDNS	44,777 (Go)	554	2
Technitium	29,709 (C#)	114	2
Simple DNS Plus	-	-	1
Total	738,160	2,664	22

D The SAD DNS Attack

SAD DNS (Side channel AttackeD DNS) [65] is a novel DNS cache poisoning attack defeating UDP port randomization via a side channel of the networking stack in modern operating systems. SAD DNS leverages the ICMP error rate limit side channel to scan ephemeral UDP ports used for DNS queries.

In detail, attackers use UDP messages to probe ports on the target server. If the target port is not open (i.e., not used for DNS queries) an ICMP port unreachable error will be emitted, while no ICMP errors will be returned if the port is open. Linux implements both per-IP (one per second) and global rate-limit (1,000/s) to restrict the number of ICMP error packages sent out per second. SAD DNS leverages the global rate limit of ICMP error messages, which has a periodic maximum burst of 50 and recovers every 20ms. This global limit opens a side channel: during each round (20ms), attackers send 50 probing packets to scan 50 ports with different source IPs. If all 50 ports are closed, 50 ICMP error messages will be triggered. If n ports are in use, then only $50-n$ ICMP error message will be sent. The attacker then send a verification packet to a closed port with his IP address. If no ICMP replies are observed, it means the global ICMP error rate limit has been used up, and all 50 ports are closed. Otherwise, the scanning hits at least one open port. Leveraging the above side channel, the scanning speed will be at most 1,000/s, enabling SAD DNS to enumerate the entire 65,535 port range within 60+ seconds.

Algorithm 1: DNS resolution process

input : A DNS *Request* from clients
output : A DNS *Reply* to clients

```
main()
    step_0: InitQuery(Q, Request)
    step_1: if SeachCache(Q, Cache)
      then
        goto final
    step_2: FindServers(Q, TgtSvrs)
    step_3: SendQuery(Q, TgtSvrs)
    step_4: ProcessResponse(Q, R)
        if ServerIsError(Q, R) then
            goto step 3
        if not MatchQuery(Q, R) then
            goto final
        SanitizeRecords(Q, R)
        if IsReferral(Q, R) then
            if not IsFwding() then
                UpdateQuery(Q)
                goto step 2
        if IsCNAME(Q, R) then
            UpdateQuery(Q)
            goto step 1
        CacheRecords(R, Cache)
    final: ConstructReply(Reply)
    return Reply
InitQuery(Q, Request)
    initialize Q.name, Q.type, Q.zone
    if IsFwding() then
        ModifyFwdQuery(Q)
SanitizeRecords(Q, R)
    for RR ∈ R do
        if OutofBailiwick(RR) then
            remove RR from R
UpdateQuery(Q, R)
    update Q.name, Q.type, Q.zone
```

E Public DNS Services

We collected 16 popular public DNS resolvers via search engines in Table 13, part of which are used in prior works [44,57,65,66,96]. And we test whether they insert NS records into the authority section of DNS responses to type A queries to evaluate requirement a) for CDNS discovery.

Table 11: DNS definitions and notations.

Definition	Description
Request	the DNS request from clients
Reply	the DNS reply to clients
RR	the resource record
name	the name of a resource record, i.e., *RR.name*
type	the type of a resource record, i.e., *RR.type*
Q	the DNS query transformed from *Request*
$Q.name$	the domain name of the Q
$Q.type$	the domain type of the Q
$Q.zone$	the *name* of the closest NS *RRs* to Q
$Q.svrs$	the servers of the closest NS *RRs* to Q
Z_R	the domain specified for recursive queries
Z_F	the domain specified for forwarding queries
$FwdZone$	the closest zone in Z_F to $Q.name$
$FwdSvrs$	the servers to resolve domains in $FwdZone$
Cache	the structure storing query results in local
$TgtSvrs$	the target servers to send current queries
R	the response from $TgtSvrs$
$R.qry$	the question section (*RR*) of R
$R.ans$	the answer section (*RR*) of R
$R.auth$	the authority section (*RR*) of R
$R.add$	the additional section (*RR*) of R
$name_a \in name_b$	$name_a$ is under or same to $name_b$
$\emptyset$	no *RR* in the corresponding field

Table 12: Common DNS server functions and notations.

Function	Description
InitQuery	initialize data structures in Table 11
IsFwding	check if $Q.name \in FwdZone$
ModifyFwdQuery	modify Q to fit for forwarding
SearchCache	look for the answer in local *Cache*
FindServers	find the best servers $TgtSvrs$ to ask
SendQuery	send the best servers $TgtSvrs$ queries
ProcessResponse	receive and analyze R
ServerIsError	check if $TgtSvrs$ is error, e.g., timeout
MatchQuery	check if $R.qry$ matches Q
IsReferral	check if R is a referral
IsCNAME	check if R is a CNAME
SanitizeRecords	remove out-of-bailiwick records
UpdateQuery	update data structure in Table 11
CacheRecords	stash records into local *Cache*
ConstructReply	form a *Reply* to clients
InBailiwick	check if $RR.name \in Q.zone$
OutofBailiwick	check if $RR.name \notin Q.zone$

Table 13: Test results on whether public DNS services stuff type NS RRSets in DNS responses to type A queries.

Stuff NS	Public DNS Services
No	Google, Cloudflare, Quad9, OpenDNS, Verisign, Yandex, AdGuard, Level3, Dyn, FreeDNS, Hurricane, CleanBrowsing, UltraDNS, 114 DNS, AliDNS, Tencent
Yes	Baidu DNS, Comodo Secure, DNS.Watch