Should I Trust You? Rethinking the Principle of Zone-Based Isolation DNS Bailiwick Checking

Yuxiao Wu*§, Yunyi Zhang†§, Chaoyi Lu‡ and Baojun Liu†‡¶
*Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University
†Tsinghua University ‡Zhongguancun Laboratory
wuyuxiao25@mails.tsinghua.edu.cn, yunyizhang@mail.tsinghua.edu.cn, lucy@zgclab.edu.cn, lbj@tsinghua.edu.cn

***Abstract*—DNS cache poisoning attacks covertly hijack domain access by injecting forged resource records into resolvers. To counter this, resolvers employ bailiwick checking, a critical defense mechanism designed to filter potentially malicious records from DNS responses. However, in the context of third-party services, a misalignment between domain ownership and the traditional, top-down zone delegation model has emerged, posing significant challenges to the effectiveness of bailiwick checks.**

In this paper, we present a systematic analysis of the design and implementation of bailiwick checking. We demonstrated that mainstream resolvers generally adopt a conservatism principle: *they will cache any resource record that satisfies minimal constraints, regardless of its direct relevance to the originating query*. Building on this finding, we propose a novel cache poisoning attack (termed CUCKOO DOMAIN): by controlling one single subdomain, attackers can compromise its parent domain or its sibling domains. The results of our testing revealed that seven major DNS resolver implementations, including BIND9 and Microsoft DNS, are vulnerable. Through a large-scale measurement study, we confirmed that 44.64% of open resolvers and 21 major public DNS providers are also at risk. In addition, we found that over a million subdomains provided by 7 providers—including No-IP, ClouDNS, and Akamai—are potentially vulnerable to hijacking through this attack. We have conducted a responsible disclosure, reporting the affected software vendors and service providers. BIND9, Unbound, PowerDNS and Technitium have acknowledged our reports and assigned 3 CVEs. We call upon the community and software vendors to address the new challenges that modern service ecosystems pose to the effectiveness of bailiwick checking.

I. INTRODUCTION

The Domain Name System (DNS), as a key part of the Internet infrastructure, is responsible for transforming user-friendly domain names into machine-understandable IP addresses. Its security and stability are directly related to the normal operation of upper-layer applications, such as PKI [1] and Email [2]. DNS cache poisoning injects forged resource records into resolvers to hijack traffic for a target domain, severely undermining the security of the Internet.

§ Both are first authors. ¶ Corresponding author.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240330
www.ndss-symposium.org

Constructing specially-crafted response packets is a simple and effective method for achieving DNS cache poisoning. In the early stages of DNS, resolvers were designed with limited security considerations and would indiscriminately cache all resource records returned by an authoritative nameserver, even when the records are owned by a *different* zone (e.g., records owned by **al**`ternic.net` can be injected into responses for **in**`ternic.net` queries and then cached by resolver [3]), which led to the first generation of cache poisoning attacks. An attacker could inject forged records of an arbitrary domain, by simply configuring an authoritative nameserver for a domain they controlled. To sanitize such malicious records in DNS responses, major resolver software vendors designed and implemented *bailiwick* checks. While mostly effective against such attacks, recent research [4] has demonstrated that bailiwick checks in mainstream resolver software may still be bypassed using carefully crafted responses.

Research Gap. Bailiwick checking aims to filter malicious resource records from DNS responses, but what constitutes a "malicious" record is not only formally undefined but also notoriously difficult to specify. This lack of a standard has led software vendors to independently implement their own versions of the check. As a result, there is currently no systematic understanding of bailiwick checking implementations within the security community.

Conservative Bailiwick Checking. In this work, we fill the gap by systematically analyzing the principles and implementation of bailiwick checks by reviewing relevant RFCs and auditing the source code of mainstream resolver software. We found that while the specific logic of their bailiwick check implementations varies, the core mechanism is fundamentally *zone*-based. They rely on the top-down hierarchy of the domain namespace, setting the current zone based on a "closest" principle. Each record in a response is then compared against this current zone to determine whether it should be filtered. For example, when resolving *example.com* with no pre-existing cache entries, a resolver will first initialize the current zone to the root (i.e., ".") and subsequently to *example.com*. Moreover, we confirmed that most implementations adopt a conservatism policy, aiming to *cache as many potentially useful records as possible*. This validation consists only of confirming that a record's domain name is part of the current zone, either by being identical to the zone or by being one of its subdomains.

CUCKOO DOMAIN Attack. We found that conservative checking policies lead to an over-trust of resource records from the authoritative nameservers of upper-level zones, which introduces new security risks. This principle is predicated on the traditional assumption that, within the DNS hierarchy, a upper-level zone holds administrative authority over its lower-level zone. However, in the context of third-party services, administrative control is often demarcated by the service itself, not by traditional zone delegation. For instance, with a Dynamic DNS (DDNS) service, a user registers and configures a subdomain from the provider; in this case, administrative authority for that subdomain belongs to the user, not the service provider who owns the parent domain. Due to conservative checking policies, the resolver will cache the subdomain's records when they are served by the parent zone's authoritative nameserver. By exploiting it, an attacker with control over a single such subdomain can configure specific records to create a new attack vector, affecting any other sibling subdomain or even the parent domain. We call it CUCKOO DOMAIN, derived from the cuckoo's practice of brood parasitism, where it deposits its eggs in the nests of unsuspecting hosts, and its offspring then displace or attack the host's own young.

This attack pattern is analogous to well-known threats in the web security field. For example, attackers can exploit platforms that provide subdomains, such as GitHub Pages [5], to conduct cross-site scripting attacks [6] or supercookies [7]. To address this, the Public Suffix List [8] was proposed to limit the trust scope of each domain. In light of the CUCKOO DOMAIN, it is necessary to rethink the effectiveness of current bailiwick checking constraints in the context of modern business scenarios.

Evaluation of Impact. To evaluate the attack's real-world impact, we first designed and implemented 20 distinct test payloads across three categories to systematically evaluate the bailiwick implementations of eight major resolver software. We found that seven of the eight software packages were vulnerable to at least one payload, and two (PowerDNS [9] and Microsoft DNS [10]) were vulnerable to all three. We then extended our testing to 588,624 stable open resolvers and 30 popular public DNS services. Our results show that 262,779 (44.64%) of these open resolvers and 21 major public DNS providers are impacted by at least one of our test payloads (e.g., Quad9 DNS [11], Clean Browsing DNS [12]), demonstrating the widespread nature of the threat.

Furthermore, we validated the attack on seven major third-party services, including No-IP [13] and Akamai [14], confirming its practical exploitability. We demonstrated that on these services, an attacker can easily register a subdomain and configure resource records. This capability serves as a stepping stone to poison responses from the provider's nameserver, leading to cache poisoning on resolvers. Finally, we used Passive DNS (PDNS) to evaluate the real-world impact scale of the attack. Our analysis shows that across seven service providers, this threat affects millions of customer subdomains.

Mitigation and Disclosures. Based on our findings, we propose mitigation for DNS software and third-party service providers. We evaluated these schemes on popular domain service platforms, and our results confirm that the proposed solutions do not interfere with legitimate domain resolution. Furthermore, we responsibly disclosed the issues to affected DNS software, public DNS services, and third-party providers. To date, BIND9 [15], Unbound [16], PowerDNS [9] and Technitium [17] have acknowledged our report, assigned 3 CVEs and released patched versions.

Contributions. We make the following contributions:

Systematic analysis of bailiwick principle. We bridged the gap in the study of bailiwick checks with a systematic analysis of the design and implementation. By reviewing 470 RFCs and auditing 8 major DNS software, we uncover a significant divide between the protocol standards and their implementation in practice.

New threat model. We proposed a novel threat model that allows attackers to poison the cache of their sibling or even parent domains in the same zone by controlling any subdomain within the zone.

Comprehensive evaluation of new attacks. We performed a comprehensive threat assessment across major software implementations, third-party service providers, open resolvers, and public DNS providers. The evaluation results show that seven major software implementations, 44.64% of open resolvers, and 21 public DNS providers are vulnerable to this threat. Furthermore, over a million customer domains hosted by seven third-party service providers face potential hijacking risks.

II. BACKGROUND

In this section, we introduce the DNS concepts, including the DNS namespace and DNS packet structure. Then we describe the bailiwick rules.

A. DNS Overview

DNS Namespace and Resolution. The DNS namespace is a tree structure where each node represents a domain name. As shown in Figure 1, the apex of the structure is the DNS root zone (denoted as ".") , followed by a number of Top-Level Domains (TLDs), such as `com` and `net`. The Second-Level Domains (SLDs) further down the hierarchy are managed by registrars, such as GoDaddy [18] and Dynadot [19], which provide registration services while transferring ownership of the domain and its subdomains to the domain administrator. The authoritative data for a domain is stored in its corresponding authoritative name server according to the domain name and data type, and a group of domains which are in the same authoritative server is called a DNS zone.

After receiving a DNS query, the DNS resolver first checks its local cache to see if it has the answer. If not, it recursively queries the DNS root server, TLD server, SLD server, until it reaches the authoritative name server for the queried domain (step ❷, ❹, and ❻ in Figure 1). During this process, name servers may return `referral` or `answer` responses, as shown in Figure 2 (a) and (b). A `referral` response is used to indicate that the queried domain name does not belong to the current zone, and it provides the information of a name server that is "closer" to the answer in `authority` and `additional` sections. An `answer` response contains the requested resource records for the queried domain in `answer` section, such as `A` record for IPv4 address, `NS` record for name server, and `CNAME` record for a canonical name.

For example, when facing a query for `example.com`, the name server for `com` return a `referral` to indicate the authoritative server for `example.com` (step ❺), while the server for `example.com` returns an `answer` containing final answers (step ❼). In order to optimize the DNS resolution process, the resolvers will cache the DNS records in referral and answer packets for a period of time.

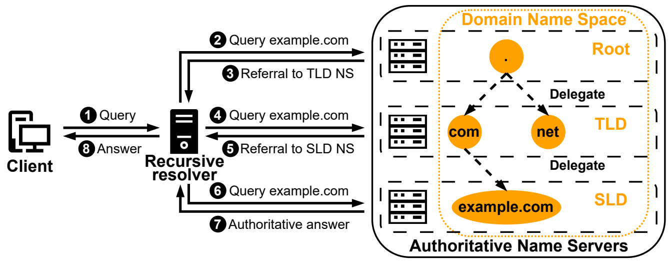


Fig. 1. Standard DNS resolution for `example.com`.

DNS Packet Structure. Both DNS query and response packets consist of the header flags and four sections containing information about the current resolution process, namely `question`, `answer`, `authority`, and `additional`. Figure 2 illustrates two response packets whose `QR` bit in the header flags used to distinguish the query and response packets are both set, and the `AA` flag is set in Figure 2 (b) to indicate it is an authoritative `answer` response. The `question` section contains the domain and the type of resource record requested by the client. A server which is not authoritative for the queried domain uses the `authority` and `additional` sections to provide referral information, while an authoritative server fill the `answer` section with the requested resource records.

A 5-tuple structure is used to describe the elements of the resource record in the three sections except for the `question` section: $\langle$`RNAME`,`RTYPE`,`RCLASS`,`TTL`,`RDATA`$\rangle$. When the resolver try to cache a new records whose `RNAME`, `RTYPE`, and `RCLASS` are the same as the one in the cache, it will decide whether to update the cache based on the data trust level. Though different DNS software have different implementations of the cache update policy [20], they all follow two basic principles to determine the trust level of the records: the ones in authoritative `answer` responses have a higher trust level than those in other responses; and the ones in the `answer` section in the same packet has a higher trust level.

B. Bailiwick Rules

Back to 1990s, the implementations of DNS resolvers were relatively simple, and they would cache all resource records in the response packets without any checking. Generally, a name server should not provide records outside of its zone [21]. However, the attacker who controlled a malicious authoritative name server could inject arbitrary records into the each section of the response packets, as shown in Figure 2 (c), (d) and (e).

Packet (c) spoof a `A` record for `victim1.com.` in the `answer` section, while packet (d) injects a `NS` record for `victim2.com.` in the `authority` section. These two malicious records can be directly cached by the resolver, and will be returned to the clients. Packet (e), on the other hand, modifies the glue record [22] in the `additional` section, which will only be used for subsequent queries for `com.` by the resolver, but still achieves the goal of domain hijacking.

In order to prevent malicious user from poisoning the resource records of other users' domains, the bailiwick checking principle is implemented in mainstream DNS software [4]. The key idea of bailiwick checking is to filter out records that are not supposed to appear in the response packet based on their `RNAME`. However, how to implement this principle is complex in practice, thus different software vendors are implementing as they see fit to defend the very type of cache poisoning since there is no standardization of bailiwick checking.

The principle of DNS bailiwick checking is the key mechanism to prevent malicious responses. However, with the development of the Internet, especially the popularity of hosting-based services, it is unclear whether the independent implementation of DNS software can effectively cope with the new challenges. We reveal the vulnerabilities in Section III, and detail the implementation and the differences of bailiwick checking among various software in Section IV.

III. Attack Overview

In this section, we propose a novel threat model based on the vulnerabilities in the implementations of zone-based isolation bailiwick checking, and analysis the feasibility of the attack.

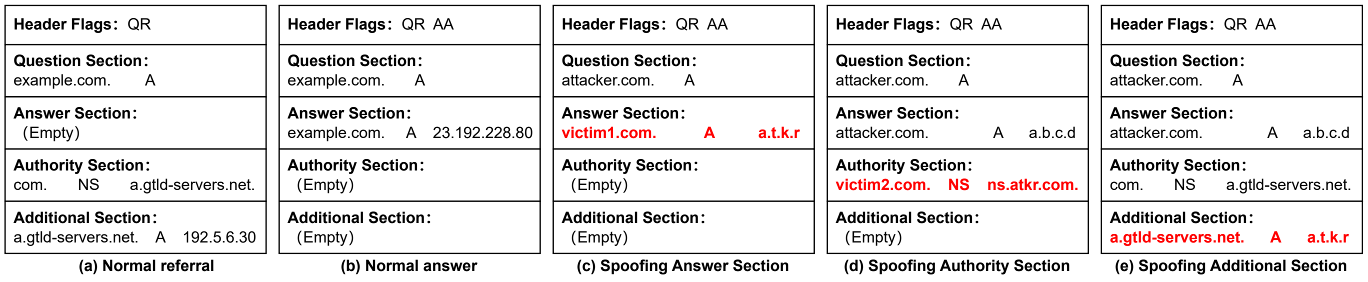
A. Threat Model

Our new threat model focuses on third-party hosting providers, where an attacker's control of a single subdomain can be weaponized. The objective is to poison the DNS cache for other sibling subdomains or even the parent service domain itself. Under this scenario, the attacker can directly impact a large number of domains belonging to other users of the same service. Figure 3 illustrates our threat model.

First, a prerequisite for the attacker is to gain control over the resource record configuration of an arbitrary subdomain under the target SLD that is not zone-cut. A lack of a zone cut implies that the subdomain shares the same authoritative servers as its parent domain. This prerequisite is frequently met in practical hosting-based environments where different subdomains are often partitioned for distinct services or business units. We introduce such real-world scenarios in Section III-C.

Second, similar to existing cache poisoning attacks [4, 23–25], the attacker needs to send DNS queries to the target resolver to trigger the attack. This is naturally satisfied for open resolvers, such as Google Public DNS [26] and Cloudflare DNS [27]. For resolvers that are private, the attacker can either enter the internal network or leverage internal hijacked machines to send DNS queries.

Additionally, due to the fact that only a few domains in the real world are configured with DNSSEC [28, 29], and



(a) Normal referral	(b) Normal answer	(c) Spoofing Answer Section	(d) Spoofing Authority Section	(e) Spoofing Additional Section
Header Flags: QR	**Header Flags:** QR AA	**Header Flags:** QR AA	**Header Flags:** QR AA	**Header Flags:** QR AA
Question Section: example.com. A	**Question Section:** example.com. A	**Question Section:** attacker.com. A	**Question Section:** attacker.com. A	**Question Section:** attacker.com. A
Answer Section: (Empty)	**Answer Section:** example.com. A 23.192.228.80	**Answer Section:** victim1.com. A a.t.k.r	**Answer Section:** attacker.com. A a.b.c.d	**Answer Section:** attacker.com. A a.b.c.d
Authority Section: com. NS a.gtld-servers.net.	**Authority Section:** (Empty)	**Authority Section:** (Empty)	**Authority Section:** victim2.com. NS ns.atkr.com.	**Authority Section:** com. NS a.gtld-servers.net.
Additional Section: a.gtld-servers.net. A 192.5.6.30	**Additional Section:** (Empty)	**Additional Section:** (Empty)	**Additional Section:** (Empty)	**Additional Section:** a.gtld-servers.net. A a.t.k.r

Fig. 2. Normal responses (a) and (b) for A record query of `example.com`. Spoofing responses (c), (d) and (e) for A record query of `attacker.com`.

the insufficient deployment of resolvers that perform DNSSEC validation [30], we do not consider DNSSEC protection [31].

Comparison with Related Works. CUCKOO DOMAIN differs from previous DNS cache poisoning attacks in the aspects described below. First, the CUCKOO DOMAIN attack exploits improper domain ownership delegation in hosting-based services, rather than software implementation flaws like forwarders lacking resource record validation [25, 32], or operating system side-channels [33–35]. Second, while prior research has examined bailiwick checks, these studies have often lacked a systematic analysis of real-world bailiwick implementations, or have focused narrowly on specific resolver types (e.g., CDNS [4]). Consequently, a comprehensive, systematic analysis of bailiwick rules remains absent.

A body of work has also studied cache poisoning via in-bailiwick responses. For instance, the well-known Kaminsky attack [24] leverages `referral` responses to achieve cache poisoning. Other studies have focused on response packet forgery through methods such as IP fragmentation [23, 36, 37] or birthday attacks [38]. These approaches primarily introduced novel methods for response packet forgery, compelling recursive resolvers to accept spoofed packets directly, thereby bypassing bailiwick checks. However, our work does not focus on devising new response packet forgery techniques. Instead, we conduct an in-depth analysis of the adaptability of recursive resolvers' bailiwick rules to contemporary and novel scenarios, revealing the limitations of "legacy" bailiwick checks. Moreover, prior attack methods can be integrated within our proposed threat model.

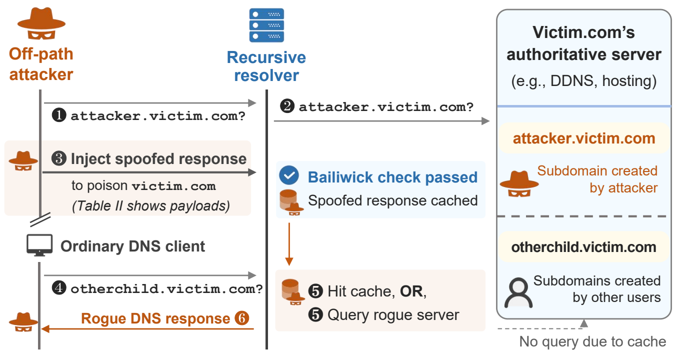


Fig. 3. Threat model and attack flow of CUCKOO DOMAIN.

B. Attack Workflow

The attack comprises two steps: (1) obtaining a subdomain for the target domains and (2) poisoning the cache of recursive resolvers by exploiting the limitations of current bailiwick rules under the subdomains scenario. It is a common practice for large enterprises to host multiple distinct services within a single DNS zone, assigning these services to different subdomains. Certain public offerings, such as Dynamic DNS (DDNS) or free subdomain services, allow users to register a subdomain to host their own services. In the first step, an attacker can gain control over a subdomain by simply registering for the service. We introduce practical scenarios of how an attacker acquires a subdomain in Section V-C.

For the second step, the attacker follows the general cache poisoning procedure illustrated in Figure 3. First, the attacker initializes a query for a subdomain (e.g., *attacker.victim.com*) under their control to the target recursive resolver. Next, the recursive resolver issues a query to the authoritative server for that subdomain. The attacker then injects a crafted, spoofed response (step ❸ in Figure 3). The response contains forged `NS` data designed to be compliant with existing bailiwick rules, thus evading checks (detailed in Section V-A). For example, the authoritative server for `victim.com` or `otherchild.victim.com` is maliciously set to `ns.rogue-ns.com`. In our threat model, the attacker is off-path and can leverage existing off-path techniques to inject the spoofed packet. For instance, an attacker could configure an oversized resource record for their subdomain to force IP fragmentation, enabling the insertion of the spoofed packet through a fragmentation-based attack [23, 25, 36, 37, 39]. We discuss the practical feasibility of such off-path attacks in Section III-C, and confirm that mainstream service providers allow users to configure resource records and IP fragmentation in Section V-C. Finally, the recursive resolver accepts and caches the spoofed response. As a result, when a victim client queries the targeted victim domain for the resolver, the resolver is redirected to query the attacker's rogue authoritative server.

C. Practical Considerations of CUCKOO DOMAIN

In this section, we analyze the feasibility of the CUCKOO DOMAIN by discussing the practical factors. First, we describe how current bailiwick rules can be exploited in the context of this attack. Next, we introduce several common subdomain services that furnish realistic scenarios in which the attack can be mounted. Finally, we discuss the feasibility of injecting the required spoofed data packets.

Limitations of Bailiwick Rules. The core objective of bailiwick rules is to prevent forged resource records. However, as network services have evolved, the boundaries of domain ownership have shifted, rendering existing bailiwick rules insufficient in certain contemporary scenarios. This allows an attacker to trigger a new attack vector by controlling any subdomain, leading to cache poisoning attacks. Through systematic analysis, we tested 20 different payloads against mainstream DNS software implementations and, based on the experimental results, designed three new attack variants (Section V-A). Moreover, we also demonstrate the prevalence of vulnerable open resolvers in the real world in Section V-B.

Subdomain Service Provider. In practice, many online services allocate subdomains to users to provide specific functionalities. For instance, GitHub [5] assigns subdomains to users for hosting blogs. However, not all such subdomain services meet the requirements of our threat model. An exploitable service needs to satisfy two conditions: First, the user needs to be able to configure resource records, which can contribute to launching a off-path attack. Second, the subdomain still resides in the same zone as its parent domain without a separate zone cut, meaning the subdomain and its parent share the same nameservers. After a detailed investigation, we identified the following services that satisfy these requirements, and measure their vulnerability and prevalence in Section V-C:

Dynamic DNS (DDNS). DDNS [40] is a service that allows users to update their DNS records dynamically, which is often used by home users and small businesses. It provides users with a convenient way to access certain services by establishing a mapping between a fixed domain name and a dynamic IP address. Most service providers (e.g., No-IP [13], Dynv6 [41]) usually allow users to configure some other resource records, such as `CNAME` and `TXT` records. We take Dynv6 as an example and show the configuration interface of its DDNS service in Appendix A. Users can configure their domain and related DNS resource records on this interface.

Free subdomain service. Some zone administrators (e.g., ClouDNS [42] and DNS Exit [43]), provide free subdomains within the zone to users for various purposes, such as personal websites and custom emails. These domains are not cut off from the DNS zone when users register, which means that the resource records directly configured by users still reside directly in the authoritative server of the zone.

Load balancing service. To achieve load balancing during the resolution process, some service providers offer Global Traffic Management (GTM) or Content Delivery Network (CDN) services, which allow users to configure a `CNAME` record pointing to the vendor's subdomain. The service provider then responds to the resolver using different nodes with information such as the resolver's geographic location and server status to manage traffic. In GTM services such as Akamai [14], users can configure various records for the service provider's subdomain to align with our threat model.

Others. In addition to the above three publicly available subdomain services, we also find that some IoT vendors (e.g., Synology [44] and ASUSTOR [45]) and large companies assign different subdomains and their record configuration rights to different internal users or devices. For the former, they provide a subdomain for their branded devices (e.g., NAS) to support various functions, including but not limited to DDNS, email services, and communication protocol specification. For the latter, it is common in large companies where different departments or teams are assigned different subdomains under the same zone for easier management and resource isolation.

Attack Implementation. The fragmentation attack is a powerful off-path attack method as it only requires the attacker to forge a small number of random bits in the response packet. However, existing attacks [23, 25, 36, 37] either rely on DNSSEC validation-enabled data packets, making it difficult for malicious data to pass validation, or struggle to identify authoritative servers in real-world environments that can return oversized response packets, limiting their impact range. CUCKOO DOMAIN, on the other hand, presents a natural advantage for overcoming these limitations. An attacker can manipulate the resource records of their subdomain to force the generation of oversized DNS response packets, thereby facilitating the execution of fragmentation attack. It is worth noting that fragmentation is only one pathway to carry out our attack. We provide attackers who perform port inference attacks [33, 46] with a variety of payloads.

Feasibility of fragmentation attack. Due to the severity of fragmentation attacks, RFC 9715 [47] outlines a series of mitigation measures. However, since packet fragmentation and reassembly are handled by the operating system, the resolver cannot detect whether a packet has been fragmented. They only apply bailiwick checking to examine the recived packets from OS. Our tests confirm the practical viability of this threat (Section V-C). The results show that the authoritative servers of 7 tested service providers can be induced to send fragmented responses, indicating their vulnerability to this attack vector. We also demonstrated an end-to-end fragmentation attack on the resolver in Appendix B.

IV. SYSTEMATIC ANALYSIS OF THE SPECIFICATION AND IMPLEMENTATION OF BAILIWICK PRINCIPLE

In this section, we first present a comprehensive analysis of the definition of bailiwick in DNS RFCs. Then, we systematically review the implementation of bailiwick rules in mainstream DNS software and summarize their workflow and differences in bailiwick checking.

A. Bailiwick in RFCs

We begin by analyzing bailiwick rules from the perspective of protocol standards, collecting and organizing relevant specifications of the bailiwick. Specifically, we gathered 470 DNS-related RFCs and searched for the keyword "bailiwick" within their contents. This process ultimately yielded only six relevant RFCs, as shown in Figure 4. We present relevant passages from these RFCs in Appendix C

Upon manual review of these six RFCs, we categorized them into two types: those imposing requirement constraints and those providing definitions. We observed that a formal definition of bailiwick appeared after the requirement constraints, which implies that bailiwick checking was initially implemented across different software as a de facto standard rather than a formally specified one.

Three RFCs (6763 [48], 7477 [49], and 9199 [50]) of requirement constraints do so for specific responses, requiring them to satisfy bailiwick rules. For instance, RFC 6763, published in 2013, required resolvers to "verify that any records they receive from a given authoritative name server are in bailiwick for that server, and ignore them if not." However, it does not further elaborate on the precise meaning of bailiwick.

The remaining three RFCs (7719 [51], 8499 [52], and 9499 [53]), published in 2015, 2019, and 2024 respectively, attempted to provide a formal definition of bailiwick. However, acknowledging the difficulty of providing a precise dictionary definition, they ultimately recommended treating "bailiwick" as a historical term, and it is sufficient to understand its technical significance.

Summary. Bailiwick rules emerged as a de facto standard from efforts to mitigate DNS cache poisoning attacks and have been implemented independently across different resolver software. The community once made efforts to try to propose a specific normative description of bailiwick in the RFC, ultimately pivoting away from the dictionary definition, instead recommending an understanding of its technical meaning.

B. Systematic Analysis of Bailiwick Checking in DNS software

To gain an in-depth understanding of how different DNS software implements the bailiwick checking, we systematically analyzed mainstream DNS software. Specifically, we selected eight of the most prevalent DNS software, which have also been analyzed in prior research [20, 54–58], including six open-source software and two closed-source software. For open-source software, we analyzed their bailiwick implementations and runtime behavior using a hybrid approach combining static and dynamic analysis. In conjunction with official documentation and relevant RFCs [21, 52, 59, 60], we conducted a four-week manual analysis of the bailiwick implementation source code for all six software. Additionally, we performed dynamic analysis using GDB [61] to collect runtime data and resolver interaction logs. For the closed-source software, Microsoft DNS and Simple DNS Plus, we first reviewed their official documentation [10, 62]. Subsequently, we employed the testing methodology described in Section V-A to analyze the behavior of their respective bailiwick check implementations.

Next, we present the results of our analysis on bailiwick check implementations and summarize the discrepancies identified among the different DNS software, as shown in Table I.

(I) Query Initialization. After receiving a query `Q` from the client, the resolver first parses and retains the queried domain (i.e., `QNAME`) and record type (i.e., `QTYPE`), while ignoring the records added in the `answer` and `authority` sections. Then, the resolver will search its cache for records that match the `QNAME` and `QTYPE`. If existing, it will return the cached data as the response `R`. If not, it will use the zone closest to `QNAME` in cache as the starting point for recursive queries and *initialize the variable* `QZONE` *to that zone for bailiwick checking*. For example, when querying the `A` record of `example.com`, if there are no records in the resolver's cache, it will initialize `QZONE` to "`.`" and send a query to the root server. If there are `NS` records of `com`, it will mark `QZONE` to `com` and send the query. The bailiwick check verifies whether the name of the resource record (i.e., `RNAME`) is equal to, or a subdomain of, the `QZONE`, formally denoted as $RNAME = QZONE$ or $RNAME < QZONE$.

(II) Referral Processing. Prior to receiving the `answer` response from the target authoritative server, the resolver iteratively queries each name server in a top-down fashion, constructing a complete resolution chain with the aid of `referral` responses. The main difference between `referral` and `answer` responses is that `referral` does not contain records in `answer` section, but only carries the relevant information for the "closer" name servers. Thus, the resolver only checks and caches the records in `authority` and `additional` sections. Different software implementations exhibit significant variations in how they process resource records within `referral` responses.

- *PowerDNS, Technitium, Microsoft DNS, and Simple DNS Plus cache records in* `authority` *section if and only if* $RNAME < QZONE$. They check each record in the `authority` section individually. Regardless of the relationship between `RNAME` and `QNAME`, as long as the condition $RNAME < QZONE$ is met, the record is cached and utilized in future resolution processes.
- *All resolvers (e.g., BIND9, PowerDNS) cache and use glue records in* `additional` *section if and only if* $RNAME < QZONE$. To mitigate resolution loops under in-domain delegation, RFC 1034 [59] introduced glue records and required their use exclusively within `referral` responses. However, we found that all resolver software only check the relationship between the `RNAME` of a glue record in the `additional` section and the `QZONE`; if $RNAME < QZONE$, they will cache the record and utilize it in future resolution processes.
- *Knot and PowerDNS cache records in* `authority` *and* `additional` *sections when* $RNAME = QZONE$. However, they differ in their caching behavior: Knot will not update its cache with new resource records that have a trust level equal to or lower than that of existing records, whereas PowerDNS will update its cache with new resource records of the same trust level.
- *MaraDNS strictly requires records in* `authority` *section that* $RNAME \leq QZONE$ *and* $RNAME \geq QNAME$*, and cache the corresponding glue records.*

After the above processing, the resolver will update `QZONE` with the "closer" domain in `NS` records, which will be used for subsequent bailiwick checks.

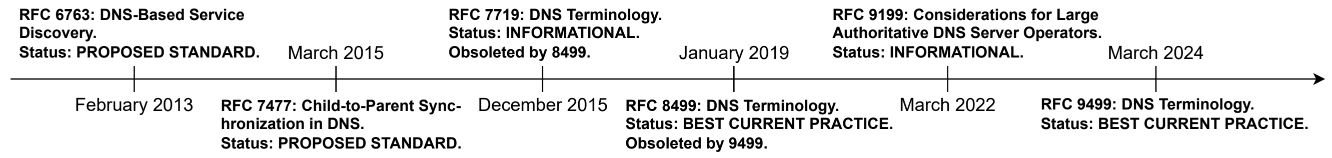

RFC 6763: DNS-Based Service Discovery. Status: PROPOSED STANDARD.
February 2013
March 2015
RFC 7477: Child-to-Parent Synchronization in DNS. Status: PROPOSED STANDARD.
RFC 7719: DNS Terminology. Status: INFORMATIONAL. Obsoleted by 8499.
December 2015
January 2019
RFC 8499: DNS Terminology. Status: BEST CURRENT PRACTICE. Obsoleted by 9499.
RFC 9199: Considerations for Large Authoritative DNS Server Operators. Status: INFORMATIONAL.
March 2022
March 2024
RFC 9499: DNS Terminology. Status: BEST CURRENT PRACTICE.

Fig. 4. The timeline of the RFCs related to bailiwick checking.

TABLE I
IMPLEMENTATION DIFFERENCE IN BAILIWICK CHECKING OF MAINSTREAM DNS SOFTWARE.

Functional implementation		BIND9[15]	Knot[63]	Unbound[16]	PowerDNS[9]	Technitium[17]	MaraDNS[64]	Microsoft DNS[10]	Simple DNS Plus[62]
Version		9.20.13	6.0.9	1.22.0	4.9.9[1]	13.3	3.5.0036	2025	9.1.116
Check matched RTYPE records in AN		✓	✓	✓	✓	✓	✓	✓	✓
Check NS from a referral response		✓	✓	✓	✓	✓	✓	✓	✓
Check AR from a referral response		✓	✓	✓	✓	✓	✓	✓	✓
Check NS from an answer response		✓	✓	✓	✓	✗	✗	✓	✓
Check AR from an answer response		✓	✓	✓	✓	✗	✗	✓	✓
Cache sibling record in referral response		✗	✗	✗	✓	✓	✗	✓	✓
Update delegation in referral response		✗	✓	✗	✓[2]	✗	✗	✗	✗
Cache unmatched RNAME in AN		✗	✗	✗	✓[3]	✗	✗	✓	✗
Cache sibling record in answer response		✓	✓	✗	✓	-	-	✓	✓
Update delegation in answer response		✓	✓	✓	✓	-	-	✓	✗
Cache overwrites with the same level data		✓	✗	✓	✓	✓	✓	✓	✓
Vulnerable	**T1 Attack**	✗	✗	✗	✓[3]	✗	✗	✓	✗
	T2 Attack	✓	✓	✓	✓	✓	✗	✓	✓
	T3 Attack	✓	✓	✓	✓	✗	✗	✓	✓

AN: Answer Section. NS: Authority Section. AR: Additional Section. [1] Due to the extended testing period, we evaluated three versions of PowerDNS: 4.9.9, 5.1.7 and 5.2.5. [2] PowerDNS versions 5.1.7 and 5.2.5 implement correctly. [3] PowerDNS version 5.2.5 implement correctly.

(III) Answer Sanitization. Authoritative `answer` responses from a target domain's authoritative server are assigned the highest trust level. Consequently, resolver software processes the records within the `answer` response differently than it does for `referral` responses. Moreover, with the exception of Technitium and MaraDNS—which only process the `answer` section while ignoring the records in `authority` and `additional` sections—all other tested resolver implementations inspect every record across all sections of the response, caching any records that pass the requisite checks.

- *PowerDNS and Microsoft cache records in the `answer` section when the condition $RNAME \leq QZONE$ is met.* Other implementations, in contrast, apply a stricter check, additionally requiring that `RNAME` equals `QNAME`.
- *BIND9, Knot, PowerDNS, and Microsoft DNS cache and use resource records in `authority` and `additional` sections, requiring only that $RNAME \leq QZONE$ and not the stricter condition of $RNAME = QNAME$.* As previously described, Knot still will not update its cache with a new record that has a trust level equal to or lower than that of the existing record.
- *Unbound caches and uses resource records in `authority` and `additional` sections, when $RNAME \leq QZONE$ and $RNAME \geq QNAME$.*
- *Simple DNS Plus only requires $RNAME < QZONE$.*

Summary. Most resolver implementations perform bailiwick checking by validating resource records based on the `QZONE`. They apply different rules for different response types, such as `referral` and `answer`, and assign varying trust levels to different resource records. However, some resolver neglect to validate the `RNAME` of records within `answer` responses using `QNAME`. In scenarios where subdomain ownership boundaries are ambiguous, our threat model (Section III-A) exploits this widespread neglect of `RNAME` validation to achieve cache poisoning of sibling domains and even the parent domain itself. Our large-scale evaluation in Section V demonstrated the widespread, real-world impact of this threat.

V. EVALUATION OF CUCKOO DOMAIN IN THE WILD

The analysis in Section IV revealed that mainstream DNS software implementations primarily focus on `QZONE` for their bailiwick checks. This creates a vulnerability to CUCKOO DOMAIN in scenarios where subdomain ownership is ambiguous, potentially leading to cache poisoning.

Building on these findings, in this section, we first design corresponding attack test packets and evaluate the susceptibility of 8 mainstream software. Next, we conduct a large-scale impact assessment on public DNS services and open resolvers. Finally, we analyze the extent to which subdomain service providers are affected. Our experimental results confirm the widespread, real-world impact of the CUCKOO DOMAIN.

A. Evaluating Mainstream DNS Software

1) Test Payload Design: As shown in Section IV-B, the bailiwick checking mechanism is implemented differently in different sections. Thus, we design distinct test payloads for each section to comprehensively evaluate the susceptibility of each software, as shown in Table II. We assume an attacker obtains the subdomain *atkr.victim.com* from a subdomain service provider with the SLD *victim.com*, and gains the authority to configure arbitrary resource records for it. Our payloads were designed according to the following principles:

- For the `answer` section, since all implementations inspect the `RTYPE` of resource records, we design two test scenarios: (i) utilizing various `RTYPEs` while ensuring the injected resource record remains consistent with the `QTYPE`, including payload 1 (`A`), payload 2 (`TXT`), and payload 6 (`CNAME`). (ii) inserting special resource records, such as payload 5 (`CNAME`), into the data packet.
- For the `authority` section, we poison the target domain's authoritative server information by constructing various `NS` records. For example, payload 9 configures the SLD's authoritative server to be the attacker's own subdomain, while payload 11 points to an authoritative server in a different TLD.
- For the `additional` section, we try to overwrite the IP address of the parent domain's nameserver using glue records. For example, payload 17 injects an `NS` record for the parent domain into the `authority` section, followed by a corresponding spoofed glue record. Payload 19 injects a spoofed IP address for the parent's authoritative server by leveraging the attacker's own subdomain.

In addition to targeting the parent domain, we also designed test payloads targeting other subdomains (i.e., the sibling domain of the subdomain controlled by attackers) denoted by payloads with a "-c" suffix. For example, payload 3 directly injects an A record for a subdomain (*sibling.victim.com*).

This approach, which targets three sections while simultaneously considering sibling domains and parent domains, enables us to propose more (13/20, i.e. PN1-8, 13-16 and 19) test cases beyond existing studies [58, 65].

In total, we design and implement 20 payloads, which we have subsequently categorized into the following three types based on their attack effectiveness and scope of impact:

T1: Arbitrary Record Injection (PN1-4). By exploiting a validation flaw in how resolvers handle resource records within the `answer` section—specifically, that they only require the resource record type (`RTYPE`) to match the query type (`QTYPE`)—an attacker can directly inject forged records for parent or sibling domains.

T2: Authority Record Takeover (PN5-16). Leveraging a flaw in the way resolvers validate authority records, the attacker can inject a fake `CNAME` record in the `answer` section to directly point the target domain to a controlled domain, or inject a fake `NS` record in the `authority` section to hijack all queries to the target domain.

T3: Glue Record Poisoning (PN17-20). By exploiting flaws in the way resolvers handle glue records, the attacker can indirectly hijack the target domain by overwriting the glue records in the resolver's cache.

2) Test Results of Mainstream DNS Software: We design a local DNS environment[1] based on `DNS-Builder`. Except for MaraDNS, each of the other seven software implementations we tested was susceptible to at least one payload. In particular, PowerDNS and Microsoft DNS were vulnerable to all three payload categories. The test results are shown in Table II.

[1]https://github.com/fly1ngpengu1ns/DNS-Bailiwick

- The bailiwick checking in PowerDNS and Microsoft DNS are the most permissive, rendering both susceptible to all three categories of attack payloads. Specifically, PowerDNS was found to be vulnerable to all 20 of our tested payloads, while 16 of these payloads were effective against Microsoft DNS.
- BIND9, Knot, Unbound, and Simple DNS Plus are all susceptible to both T2 and T3 threats. Moreover, for each of these software implementations, at least five distinct test payloads were effective.
- Technitium is vulnerable to T2 attacks because it improperly caches and uses the records of sibling domain from the `authority` and `additional` sections.

B. Evaluating DNS Resolvers in the Wild

To evaluate the real-world impact of CUCKOO DOMAIN, we conducted a large-scale measurement of resolvers in the wild.

1) Collecting Available DNS Resolvers: We first collect our test subjects, which include major public DNS providers and a large number of stable open resolvers.

Public DNS Providers. We collect a list of 30 popular public DNS vendors that are widely used in the real world, based on search engines and previous studies [20, 35, 54, 57, 66–69] as listed in Appendix D, including Google Public DNS [26] and Quad9 DNS [11].

Open Resolvers. To assess the susceptibility of open resolvers in the wild, we utilized XMAP [70] to conduct a scan of the entire IPv4 address space on June 1, 2025. Specifically, we probe the entire routable IPv4 address space with DNS queries, classifying hosts that provide a successful response as open resolvers. To prevent any interference with regular user activity, we utilize a domain name registered exclusively for our scanning purposes. Prior research [71] has confirmed that open resolvers are dynamic, with a high churn rate over short periods. Accordingly, we repeated our scan on June 7, 2025, and defined the final set of open resolvers as the IP addresses present in both scan results. Finally, we collected 588,624 stable open resolvers, as shown in Table III.

2) Measurement Setups: To conduct a large-scale impact evaluation on in-the-wild DNS resolvers without disrupting legitimate users, we configured a controlled experimental environment. This setup comprises two domains and four servers with public IP addresses. The first domain, *victim.com*, serves as the service domain of the target subdomain hosting provider. The second, *attack.net*, is an attacker-controlled domain used to host authoritative nameservers for a cross-TLD delegation. The 4 servers are configured with distinct roles: (1) a client to simulate queries; (2) an authoritative nameserver for zone division; (3) the legitimate authoritative nameserver for victim.com; and (4) a rogue authoritative nameserver, controlled by the attacker, returns forged resource records and is the server the target resolver queries if an attack is successful.

TABLE II
INFORMATION ABOUT THE PAYLOADS AND WHETHER THE DNS RESOLVER CACHES MALICIOUS INFORMATION.

Payload Name[1]	DNS Section	Values in DNS Section[2]	Cache the Malicious Records of victim.com/sibling.victim.com?[3]							
			BIND9	Knot	Unbound	PowerDNS	Technitium	MaraDNS	Microsoft DNS	Simple DNS Plus
1. AN-a	QD	A? sub.atkr.victim.com	No	No	No	Yes[4]	No	No	Yes	No
	AN	sub.atkr.victim.com A a.t.k.r								
		victim.com A a.t.k.r								
2. AN-txt	QD	TXT? sub.atkr.victim.com	No	No	No	Yes[4]	No	No	Yes	No
	AN	sub.atkr.victim.com TXT aktr-txt								
		victim.com TXT aktr-txt								
3. AN-a-c	QD	A? sub.atkr.victim.com	No	No	No	Yes[4]	No	No	Yes	No
	AN	sub.atkr.victim.com A a.t.k.r								
		sibling.victim.com A a.t.k.r								
4. AN-txt-c	QD	TXT? sub.atkr.victim.com	No	No	No	Yes[4]	No	No	Yes	No
	AN	sub.atkr.victim.com TXT aktr-txt								
		sibling.victim.com TXT aktr-txt								
5. AN-cname	QD	A? sub.atkr.victim.com	No	No	No	Yes[4]	No	No	Yes	No
	AN	sub.atkr.victim.com A a.t.k.r								
		victim.com CNAME atkr.net								
6. AN-cname-2	QD	CNAME? sub.atkr.victim.com	No	No	No	Yes[4]	No	No	Yes	No
	AN	sub.atkr.victim.com CNAME sub.atkr.net								
		victim.com CNAME atkr.net								
7. AN-cname-c	QD	A? sub.atkr.victim.com	No	No	No	Yes[4]	No	No	Yes	No
	AN	sub.atkr.victim.com A a.t.k.r								
		sibling.victim.com CNAME atkr.net								
8. AN-cname-2-c	QD	CNAME? sub.atkr.victim.com	No	No	No	Yes[4]	No	No	Yes	No
	AN	sub.atkr.victim.com CNAME sub.atkr.net								
		sibling.victim.com CNAME atkr.net								
9. NS-in-domain	QD	A? sub.atkr.victim.com	Yes	No	Yes	Yes	No	No	Yes	No
	AN	sub.atkr.victim.com A a.t.k.r								
	NS	victim.com NS ns.atkr.victim.com								
	AR	ns.atkr.victim.com A a.t.k.r								
10. NS-in-domain-referral[6]	QD	A? sub.atkr.victim.com	No	No	No	Yes[5]	No	No	No	No
	NS	victim.com NS ns.atkr.victim.com								
	AR	ns.atkr.victim.com A a.t.k.r								
11. NS-out-zone	QD	A? sub.atkr.victim.com	Yes	No	Yes	Yes	No	No	Yes	No
	AN	sub.atkr.victim.com A a.t.k.r								
	NS	victim.com NS ns.atkr.net								
12. NS-out-zone-referral[6]	QD	A? sub.atkr.victim.com	No	No	No	Yes[5]	No	No	No	No
	NS	victim.com NS ns.atkr.net								
13. NS-in-domain-c	QD	A? sub.atkr.victim.com	Yes	Yes	No	Yes	No	No	Yes	Yes
	AN	sub.atkr.victim.com A a.t.k.r								
	NS	sibling.victim.com NS ns.atkr.victim.com								
	AR	ns.atkr.victim.com A a.t.k.r								
14. NS-in-domain-referral-c[6]	QD	A? sub.atkr.victim.com	No	No	No	Yes	Yes	No	Yes	Yes
	NS	sibling.victim.com NS ns.atkr.victim.com								
	AR	ns.atkr.victim.com A a.t.k.r								
15. NS-out-zone-c	QD	A? sub.atkr.victim.com	Yes	Yes	No	Yes	No	No	Yes	Yes
	AN	sub.atkr.victim.com A a.t.k.r								
	NS	sibling.victim.com NS ns.atkr.net								
16. NS-out-zone-referral-c[6]	QD	A? sub.atkr.victim.com	No	No	No	Yes	Yes	No	Yes	Yes
	NS	sibling.victim.com NS ns.atkr.net								
17. AR-victim-domain	QD	A? sub.atkr.victim.com	No	No	Yes	Yes	No	No	Yes	Yes
	AN	sub.atkr.victim.com A a.t.k.r								
	NS	victim.com NS ns.victim.com								
	AR	ns.victim.com A a.t.k.r								
18. AR-victim-domain-referral[6]	QD	A? sub.atkr.victim.com	No	Yes	No	Yes	No	No	No	No
	NS	victim.com NS ns.victim.com								
	AR	ns.victim.com A a.t.k.r								
19. AR-attack-domain	QD	A? sub.atkr.victim.com	No	No	Yes	Yes	No	No	Yes	Yes
	AN	sub.atkr.victim.com A a.t.k.r								
	NS	sub.atkr.victim.com NS ns.victim.com								
	AR	ns.victim.com A a.t.k.r								
20. AR-attack-domain-referral	QD	A? sub.atkr.victim.com	Yes[7]	Yes	Yes	Yes	No	No	No	No
	NS	sub.atkr.victim.com NS ns.victim.com								
	AR	ns.victim.com A a.t.k.r								

[1] The naming format is: section where the malicious records is located - type of malicious records / relationship between malicious records and `QZONE` - either `referral` or `answer` - whether it targets sibling domains. [2] We use red color to highlight malicious records. [3] We use red color to highlight the vulnerable resolvers.
[4] PowerDNS version 5.2.5 is immune to this payload. [5] PowerDNS versions 5.1.7 and 5.2.5 are immune to this payload.
[6] These payloads can be used directly for port inference attacks, but fragmentation attacks require additional benign `NS` records, as demonstrated in Appendix B.
[7] BIND9 overwrites the original glue record with the attacker address a.t.k.r, but continues to use the original glue record during the subsequent 10 seconds of resolution.

When using a specific payload from Table II to test a open resolver r, the testing process involves following three stages:

i. *Zone division.* To assess each target resolver against 20 payloads, we employ a method that isolates each test to prevent cross-payload interference. Specifically, to test a payload (e.g., payload 4) against a resolver r, we use a unique domain such as `payload-4-r.victim.com`. A dedicated name server within our experimental environment handles the delegation for these domains. Notably, while this subdomain-based approach reduces the number of SLDs required for testing, we acknowledge that resolver behavior may be influenced by the domain's level in the hierarchy. We validated this testing method against major DNS resolver software to confirm its effectiveness.

ii. *Attack launch.* When the resolver r sends a query for `[payload-id]-[r].victim.com`, the authoritative name server will respond the malicious packet according to the payload information in the query. Some payloads, such as the packet 11, involve a out-of-zone `NS` record, where the `RDATA` will be replaced with *attack.net*, and all forged authoritative name servers will ultimately point to our controlled server with IP address *a.t.k.r*.

iii. *Attack verify.* We determine the outcome of an attack by examining the response received at the client. For a T1 attack, success is confirmed if the resource record in the response from the victim's authoritative nameserver has been replaced with the attacker's IP address (i.e., *a.t.k.r*). For T2 and T3 attacks, a successful attempt results in the client receiving a response directly from the attacker's rogue authoritative nameserver, which also contains the attacker's IP address (i.e., *a.t.k.r*).

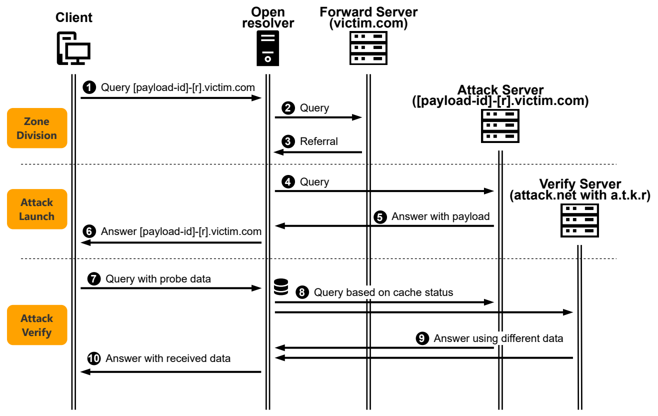

Client
Open resolver
Forward Server (victim.com)
Attack Server ([payload-id]-[r].victim.com)
Verify Server (attack.net with a.t.k.r)
Zone Division
Attack Launch
Attack Verify
1 Query [payload-id]-[r].victim.com
2 Query
3 Referral
4 Query
5 Answer with payload
6 Answer [payload-id]-[r].victim.com
7 Query with probe data
8 Query based on cache status
9 Answer using different data
10 Answer with received data

Fig. 5. Identify vulnerable open resolver with a specific payload.

3) Evaluation Results: Our large-scale measurement results reveal that *44.64% of open resolvers are vulnerable to at least one threat*, as listed in Table III. We found that 6.98% of resolvers are susceptible to the T1 attack, as they cached and utilized at least one of payloads 1–4. A larger portion, 39.95% of resolvers, face the T2 attack, where malicious authority records can be injected via payloads 5–16. Among these, payload 9 was the most impactful, successfully inserting an `NS` record and its corresponding glue record by the `authority` and `additional` sections of the `answer` responses for over 30% of the resolvers. Lastly, the T3 attack threatens 23.59% of resolvers, indicating that these resolvers place excessive trust in unsolicited glue records.

Cache status of three sections. The validation policies for resource records result in significant differences in how various resolvers cache and utilize the `answer`, `authority`, and `additional` sections of a DNS response. As shown in Table III, the number of resolvers caching records in the `answer` section (payloads 1 to 8) is generally lower than other sections. In contrast, the `authority` section has the highest number of vulnerable resolvers. This result is consistent with the findings in Table II, indicating that resource records in the `answer` section are afforded a higher trust level and are consequently subject to stricter validation policies. In contrast, the checks for the `authority` and `additional` sections are comparatively more lenient.

Cache status of subdomains. The results show that all eight subdomain-related payloads were cached by tens of thousands of resolvers, and the most effective payload (Payload 10) was adopted by 18.02% of them. This finding demonstrates that by controlling a single subdomain, an attacker can compromise the resource records for any other subdomain under the same parent domain. We highlight the pervasiveness and significant danger posed by this attack vector, especially given the popularity of current service hosting services.

Furthermore, we analyzed the geographical distribution of all vulnerable resolvers using the GeoLite database [72], as depicted in Figure 6. These vulnerable resolvers are widely distributed globally, with particularly large numbers in China, United States, and Russia (these resolvers account for 67%, 18%, and 60% of their total resolvers, respectively), demonstrating the widespread nature of this threat.

TABLE III
VULNERABLE OPEN DNS RESOLVER STATISTICS.

Attack Payload	# IP	%	Attack Payload	# IP	%
DNS resolver on Jun. 1 2025	1,044,825	-	DNS resolver alive on Jun. 7 2025	588,624	100%
1. AN-a	28,611	4.86%	2. AN-a-c	26,216	4.45%
3. AN-txt	28513	4.84%	4. AN-txt-c	28,545	4.85%
5. AN-cname	28,697	4.88%	6. AN-cname-c	26,061	4.43%
7. AN-cname2	28,607	4.86%	8. AN-cname2-c	26,239	4.46%
9. NS-in-domain	179,175	30.44%	10. NS-in-domain-c	106,049	18.02%
11. NS-in-domain -referral	46,776	7.95%	12. NS-in-domain -referral-c	39,224	6.66%
13. NS-out-zone	105,837	17.98%	14. NS-out-zone-c	99,886	16.97%
15. NS-out-zone -referral	5,850	0.99%	16. NS-out-zone -referral-c	24,636	4.19%
17. AR-victim-domain	89,591	15.22%	18. AR-attack-domain	46,309	7.87%
19. AR-victim-domain -referral	76,121	12.93%	20. AR-victim-attack -referral	54,961	9.34%
T1 Attack	**41,104**	**6.98%**	**T2 Attack**	**235,171**	**39.95%**
T3 Attack	**138,856**	**23.59%**	**Total Vulnerability**	**262,779**	**44.64%**

For each public DNS service in our dataset, we executed the testing procedure 10 times to mitigate potential inconsistencies caused by multiple backends. The complete experimental results are presented in Table VI of Appendix D. Our findings show that 21 mainstream public DNS services are affected by at least one payload, including Quad9 DNS [11], DynDNS [73], and OpenDNS [74]. Moreover, six of these major

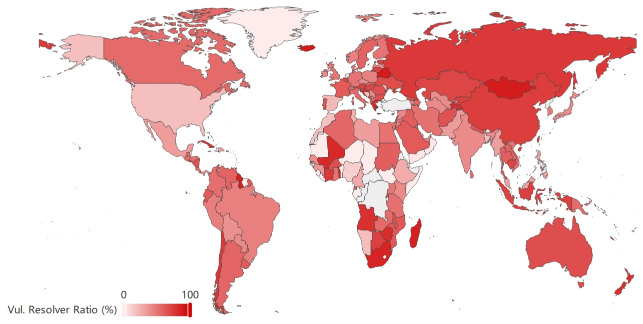


Fig. 6. Geographical view of vulnerable resolvers in each region, with different colors representing their percentages.

services, such as Clean Browse DNS [12] and OpenDNS [74], were found to be vulnerable to all three threat types. Table IV details the vulnerability status of 10 representative providers.

TABLE IV
VULNERABILITIES OF 10 OUT OF 30 PUBLIC RESOLVER VENDORS.

Public DNS Vendors	**IPv4 Address**	**Vulnerable?**		
		T1	T2	T3
CNNIC sDNS [75]	1.2.4.8	✗	✓	✓
Quad9 DNS [11]	9.9.9.9	✗	✓	✓
Strongarm DNS [76]	52.3.100.184	✓	✓	✓
Hurricane Electric DNS [77]	74.82.42.42	✓	✓	✓
ControlD DNS [78]	76.76.2.0	✓	✓	✓
LibreDNS [79]	88.198.92.222	✓	✓	✓
Safe Surfer DNS [80]	104.155.237.225	✓	✓	✗
OneDNS [81]	117.50.10.10	✗	✓	✓
Clean Browsing DNS [12]	185.228.168.10	✓	✓	✓
Dyn DNS [73]	216.146.35.35	✗	✓	✓

✓: Vulnerable. ✗: Notvulnerable.

C. Evaluating Vulnerable Service Providers

To implement CUCKOO DOMAIN, our threat model requires the attacker to control a subdomain that is still within the target zone. It is a common practice for large enterprises to delegate different subdomains for distinct business functions. Concurrently, some business models involve assigning subdomains to users to provide specific services, such as DDNS. Following the introduction in Section III-C, we evaluated the impact scope of our threat model in real-world scenarios.

Specifically, we collect our test targets from public reports and industry forums. For DDNS service, we choose No-IP [13] and Dynv6 [41] as targets. No-IP, established in 1999, is one of the oldest and most well-known DDNS providers, while Dynv6 is widely used due to its native support for IPv6 and totally free service. For free subdomain service, we select DNS Exit [43] and ClouDNS [42]. DNS Exit, founded in 1998, is an ICANN accredited registrar, while ClouDNS is the largest global hosting DNS provider in Europe. For load balancing service, we choose Akamai GTM [14], which has the largest distributed platform in the world, as our target for further analysis. For other subdomain service, we select Synology [44] and ASUSTOR [45], which are well-known NAS vendors that provide subdomain services for their users.

Furthermore, we register a test account with each provider to validate its vulnerability. To prevent any service disruption, our assessment for each provider adhered to the conditions outlined in our threat model. First, we directly apply a subdomain or read their official documentation to confirm our ability to configure its resource records. Second, using techniques from prior work [25, 82] (sending PTB to lower down targets' MTU), we determine whether the provider's name servers support fragmentation. We ran three trials from three vantage points and marked a name server as supporting fragmentation only if fragmented responses appeared in all trials.

In addition, to evaluate the real-world impact of these service providers, we measure the scope and scale of their operations using Passive DNS (PDNS)[2] data. We first collect the domain suffixes provided by these vendors from their official websites. Then, we collect and count the number of domains containing these domain suffixes and the average daily queries from PDNS for April 2025, to reflect the potential number and impact of CUCKOO DOMAIN's attack targets.

Results. Our tests revealed that all seven providers allowed users to register subdomains, and permitted users to configure arbitrary resource records for their subdomain, like `TXT` record. Further testing of fragmentation capabilities showed that the name servers of these providers supported fragmentation, fragmenting the packet that exceeded 548 bytes.

Moreover, our analysis of PDNS data reveals the widespread impact of these vulnerable providers. We found that these vendors each expose hundreds of thousands of user subdomains to security risks, collectively handling a daily DNS query volume that exceeds ten million, as shown in Table V.

VI. DISCUSSION

In this section, we first propose mitigation strategies for both software vendors and third-party service providers and validate the feasibility of these solutions through a large-scale evaluation of popular domains. Then, we discuss the lessons learned from our study. Finally, we describe our responsible disclosure process.

A. Mitigation

Based on our findings, we propose the following enhancements to the bailiwick checking logic for software vendors:

Validating the `RNAME` against the `QNAME` when caching records, rather than only validating the `QZONE`. This stricter policy would prevent resolvers from caching excessive out-of-bailiwick records unrelated to the current query, thereby effectively mitigating the T1 and T2 attacks.

[2]The Passive DNS data we use comes from a large security company we work with. The data has been strictly anonymized, and no user privacy-related data is involved in the analysis process.

TABLE V
AFFECTED VENDORS AND THE POPULARITY OF THEIR SERVICES.

Vendor	Available Domain	Name Server	Number of Subdomains	Daily Queries
No-IP[1] [13]	∗.ddns.net	nf1.no-ip.com	862,426	14,986,250.7
	∗.zapto.org	nf2.no-ip.com	310,049	4,402,439.6
	∗.hopto.org	nf3.no-ip.com	305,116	4,197,433.4
	∗.sytes.net	nf4.no-ip.com	300,657	1,337,583.6
	∗.ddns.me		274,537	255,232.4
Dynv6 [41]	∗.dynv6.net	ns1.dynv6.com	31,337	2,749,345.5
	∗.dns.army	ns2.dynv6.com	9,058	3,114,435.7
	∗.v6.army	ns3.dynv6.com	3,390	693,529.5
	∗.dns.navy	ns2.dynv6.net[2]	3,364	734,668.9
	∗.v6.rocks	ns3.dynv6.net[2]	3,111	428,565.6
	∗.v6.navy		2,140	542,523.8
DNSExit [43]	∗.linkpc.net	ns10.dnsexit.com	9,339	4,548,094.2
	∗.publicvm.com	ns11.dnsexit.com	7,838	162,293.4
	∗.work.gd	ns12.dnsexit.com	7,825	199,321.1
	∗.run.place	ns13.dnsexit.com	2,776	14,355.8
ClouDNS [42]	∗.ip-ddns.com	ns61.cloudns.net ns62.cloudns.com	17,684	149,697.8
	∗.ddns-ip.net	ns63.cloudns.net ns64.cloudns.uk	5,202	74,109.3
Akamai [14]	∗.akadns.net	a1-128.akadns.net[4] a18-128.akagtm.org	124,354	3.282×10^9
Synology[5] [44]	∗.myds.me	ddns-ns1.quickconnect.to	1,481,000	8,925,267.2
	∗.synology.me	ddns-ns2.quickconnect.to	655,159	15,916,461.5
	∗.i234.me	ddns-ns3.quickconnect.to	32,338	5,028,393.2
	∗.dsmynas.com	ddns-ns4.quickconnect.to	10,386	1,095,862.7
	∗.dscloud.biz		7,116	2,860,361.9
ASUSTOR [45]	∗.myasustor.com	ns1.myasustor.com ns2.myasustor.com	11,681	14,576.2
Total Vul.[6]	-	-	**6,400,327**	**3.3×10^9**

[1] No-IP has 82 TLDs as subdomain services, only the top 5 most frequently used domains are shown in the table.
[2] Only `dynv6.net` has `NS` records of `ns2.dynv6.net` and `ns3.dynv6.net`.
[3] FreeDNS has over 20,000 shared TLDs, only the top 5 most frequently used domains are are counted and displayed.
[4] Akamai GTM has a total of 9 authoritative servers, whose SLDs are `akadns.net` and `akadns.org`. The specific list is not detailed here.
[5] Synology has 12 TLDs as subdomain services, only the top 5 most frequently used domains are shown in the table.
[6] The number count all available domains, including those not listed, from all affected vendors.

Restricting the caching and use of glue records from `referral` responses. Ideally, such records should not be cached, and under no circumstances should they be used to overwrite existing resource records. The fundamental role of glue is to address resolution loops, and it should not be trusted outside of this narrow context. Previous work [54] had demonstrated that over-trusting glue records can lead to significant domain hijacking vulnerabilities.

For third-party service providers, we recommend that they configure their authoritative nameservers to disallow IP fragmentation or use the latest version software, as this can effectively limit an attacker's capabilities. The dangers of UDP fragmentation are well-documented within the security community, with a recommendation to prohibit it. The standard mechanism for handling oversized DNS responses is to re-issue the query over TCP. Additionally, we think implementing a zone cut when assigning subdomains is an effective countermeasure, though it seems cumbersome.

Evaluation of Mitigation. Considering the complexity of real-world network environments, altering a resolver's bailiwick checking logic could potentially impact resolution efficiency or even lead to resolution failures. To evaluate the impact of our mitigation, we conducted a large-scale test using the Tranco list [83]. Specifically, we collect the `A`, `NS`, and `TXT` resource records for the top 100,000 domains from the Tranco list and systematically checked each record to determine if it would be filtered by our proposed mitigation measures. We analyze the following three types packets that are related to our mitigation, and display the results in Figure 7.

1) **Unmatch Records in Answer's AN.** We look for packets which contain records in the `answer` section that do not match the `QNAME` of the query. These records should not be cached, as they are not relevant to the original query.
2) **In Zone Records in Answer's NS.** We search for answer packets that contain `NS` records in the `authority` section, where the `RNAME` is located below the zone apex and does not belong to the `QNAME` or its subdomains.
3) **Sibling's Records in Referral's NS.** We also search for `referral` response packets that contain `NS` records in the `authority` section, where the `RNAME` is a sibling domain of the `QNAME`.

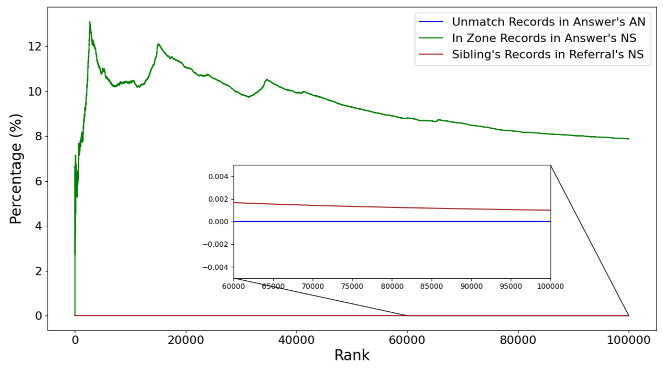


Fig. 7. Results of recursive resolution for 100,000 domains in Tranco list.

Results. Our experimental results show that for the top 100K domains, only a few of query processes generated responses that would be flagged by our mitigation measures. And, this did not impact the normal resolution process for the domains.

Firstly, for the `answer` responses containing unmatched answer records, we only find one case that using `NS` as `QTYPE` to query `csrc.gov.cn`, which places the glue record in the `answer` section instead of the `additional` section.

Next, for the `answer` responses containing in-zone `NS` records, we found that 6% of these responses included additional `NS` records. These records are typically used by a domain's authoritative nameserver for `NS` records updates, and not caching them does not impact the domain resolution process. Our proposed mitigation advises against directly caching these resource records. Instead, performing an active query, such as directly requesting the `NS` records for the domain, provides a more accurate and secure method for obtaining this information (see Appendix E for details).

Lastly, for the packets containing sibling domain `NS` records, we only find the `tc.qq.com` (ranked 47,472), which contains the record for `tc.qq.com.qq.com`.

B. Lessons Learned

The CUCKOO DOMAIN attack reveals the limitations of traditional solutions in new scenarios, demonstrating that even the most well-designed defenses can develop flaws as technology and use cases evolve.

As a de facto standard, bailiwick checking is widely adopted by major resolver software implementations and serves as a cornerstone of DNS security. However, with the evolution of the internet—particularly the changing principles of subdomain ownership delegation in modern business scenarios—we must re-examine whether traditional, zone-based implementations can effectively defend against threats in these new contexts. This serves as a critical reminder that, in the face of an ever-evolving internet landscape, continuous effort is required to safeguard the security of the DNS.

Greater effort is required to bridge the gap between theoretical understanding and practical implementation. During our analysis of the BIND9 source code, we found that the developers stated in the comments that "BIND9 caches the records in the authority section of answer response, whose `RNAME` is a subdomain of the domain being queried"[3]. However, it actually caches records of subdomains of the zone apex.

The current deployment status of authoritative name servers is also concerning. Our measurements reveal that servers of several well-known providers still support IP fragmentation, which should not exist in standard DNS authoritative servers. While these providers may deploy customized software to meet specific operational needs, we highlight that security risks should not be overlooked, and they should adhere to best practices for standard DNS authoritative server implementations.

C. *Responsible Disclosure*

We have responsibly disclosed the threat to all affected DNS software vendors, public DNS providers, and third-party service providers , and are actively discussing mitigation strategies with them. To date, BIND9, Unbound, PowerDNS and Technitium have acknowledged our disclosureand have released patched versions. Furthermore, BIND9, Unbound, and PowerDNS have each assigned a CVE to this vulnerability. Both BIND9 and Technitium incorporate `QNAME` into bailiwick checking, while Unbound and PowerDNS take a more aggressive approach by discarding records in `authority` and `additional` sections within the `answer` response. We are currently awaiting responses from the other vendors.

VII. Related Works

DNS Cache Poisoning Attacks. Among the vulnerabilities in DNS, the most notorious is the cache poisoning attack. To achieve cache poisoning, an attacker can use various methods to send a malicious response packet to the target DNS resolver before the genuine response from the authoritative server arrives. Schuba [84] in 1993 implemented off-path attacks by brute-forcing the 16-bit Transaction ID in DNS response packets. Stewart [38] in 2003 utilized the birthday paradox concept, where multiple clients simultaneously send queries, enabling attackers to construct only hundreds of packets for off-path cache poisoning attacks, significantly improving attack efficiency. Kaminisky [24] in 2008 proposed an attack using random subdomains to bypass cache restrictions.

Due to the deployment of various enhanced DNS packet randomization schemes, such as source port randomization [85], 0x20 encoding [86], and DNS Cookie [87], the state space of packets has grown exponentially. Consequently, many side-channel attack methods based on IP Fragmentation [23, 25, 36, 37] and ICMP response packets [33, 34] or pseudo-random number generator [35] have been proposed to reduce the state space for brute-force attacks.

Bailiwick Rules and Cache Mechanism Analysis. In 1997, Kashpureff [3] implemented a significant cache poisoning attack by injecting out-of-zone data into response packets, directly prompting the design and implementation of bailiwick rules in various resolver software. Schomp et al. [32] in 2014 analyzed the response validation practices of routers and found many of them still lacked bailiwick checking. The bailiwick principle is the cornerstone of DNS security, but few studies have systematically analyzed its design and implementation.

Li et al. [4] discussed the differences in `QZONE` initialization between forwarding and recursive modes of CDNS, which allowed the injection of malicious information into the public cache of CDNS. However, this essentially relies on flaws in the initialization implementation of CDNS, while this paper focuses on the bailiwick checking of the response. Li et al. [20] analyzed the cache update strategy of resolvers for `NS` records and demonstrated how to maintain the existence of an entire zone using subdomains. However, their focus was on the subtle timing between the deletion of old cache and the insertion of new cache records, rather than on how to poison the caches of other domains. Some studies [58, 88] have tested how resolvers check and cache data using various test cases, but they failed to provide a comprehensive discussion on the bailiwick principle and did not propose a practical threat model for attackers to implement attacks.

VIII. Conclusion

Bailiwick checking is a cornerstone of DNS security and a critical defense against DNS cache poisoning attacks. In this paper, we conducted a systematic analysis of the design and implementation of bailiwick checking in major resolver software. We revealed that conservative checking policies face significant challenges in modern third-party service scenarios. Based on these findings, we proposed a new threat model, the Cuckoo Domain attack, where an attacker who controls a single subdomain can compromise its parent domain or other sibling subdomains. Our evaluation shows that seven mainstream DNS resolver implementations, including BIND9 and Microsoft DNS, are susceptible to this threat. Furthermore, our large-scale measurements confirmed that 44.64% of open resolvers and 21 public DNS providers, including Quad9 DNS and DynDNS, are at risk. We further validated that on seven third-party service platforms, including No-IP, ClouDNS, and Akamai, millions of customer subdomains are exposed to potential hijacking risks. We have engaged in

[3] https://gitlab.isc.org/isc-projects/bind9/-/blob/v9.20.13/lib/dns/resolver.c?ref_type=tags#L8751-8753

a responsible disclosure process with the affected vendors; BIND9, Unbound, PowerDNS and Technitium have confirmed our findings, assigned 3 CVEs and released patched versions. We call upon the community and software vendors to address the new challenges facing DNS security in the context of modern service paradigms.

Acknowledgment

We thank all anonymous reviewers for their valuable and constructive feedback. This work is supported by the National Key Research and Development Program of China (No. 2023YFB3105600), and the National Natural Science Foundation of China (Grant No. 62102218).

Ethics Considerations

Our experiments involve many DNS software, open resolvers, and name servers, so we have fully considered ethical issues during the actual experimental process. Our institution does not have an IRB, but we refer to and strictly follow the Menlo Report [89] and the best practices of network measurements [90]. Specifically, we have taken the following measures to ensure the ethicality of our experiments:

Software Testing. Regarding the software testing in Section V-A, all open-source software tests were conducted on a controlled local machine. These software were downloaded from their official websites and compiled and run in different Docker containers. We set up a complete DNS resolution system in this testing environment, including multiple name servers for the root and other domains, ensuring that this environment does not interact with the real world.

Internet Measurement. First, during the measurement process in Section V-B, we registered all the domains needed for the tests and pointed them to our controlled DNS servers. These servers only respond to DNS requests for the test domains and do not handle requests for any other domains or provide other services. Second, We ensured that all DNS records used in the tests point to domains or IPs controlled by us, without affecting any other real-world DNS components. We configured the test domains with special `TXT` records indicating the purpose of the test and providing contact information. Third, we used a cloud server to initiate the DNS queries in Section VI-A and strictly adhered to the service provider's terms of use [91]. Fourth, we strictly limited the scanning rate to avoid potential negative impacts. When querying target resolvers, we limited it to below 5 QPS per resolver. For probing the 100,000 domains in Section VI-A, we queried one domain per second, and the overall measurement time did not exceed 30 hours to avoid overloading the cloud server. Finally, we set the TTL of all DNS records in test to 60 seconds to prevent any abuse of cached records.

Vulnerability Disclosure. We have proposed several suggestions to mitigate the risk of Cuckoo Domain, and have reported vulnerabilities to the affected DNS and subdomain service vendors in July, 2025, leaving sufficient time (nearly half a year) for them to confirm and develop patches before the paper publication.

References

[1] Y. Zhang, B. Liu, C. Lu, Z. Li, H. Duan, J. Li, and Z. Zhang, "Rusted anchors: A national client-side view of hidden root cas in the web pki ecosystem," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[2] K. Shen, C. Wang, M. Guo, X. Zheng, C. Lu, B. Liu, Y. Zhao, S. Hao, H. Duan, Q. Pan, and M. Yang, "Weak links in authentication chains: A large-scale analysis of email sender spoofing attacks," in *30th USENIX Security Symposium*, Aug. 2021.

[3] J. Kornblum, "AlterNIC founder arrested," https://www.cnet.com/tech/services-and-software/alternic-founder-arrested, 1997.

[4] X. Li, C. Lu, B. Liu, Q. Zhang, Z. Li, H. Duan, and Q. Li, "The maginot line: Attacking the boundary of DNS caching protection," in *32nd USENIX Security Symposium, USENIX Security 2023*, 2023.

[5] GitHub, https://pages.github.com/, 2025.

[6] A. Klein, "Dom based cross site scripting or xss of the third kind," in *Web Application Security Consortium*, Jul. 2005.

[7] A. Barth, "HTTP State Management Mechanism," RFC 6265, Apr. 2011.

[8] Mozilla, https://publicsuffix.org/, 2025.

[9] PowerDNS, https://www.powerdns.com/, 2025.

[10] M. D. N. System, https://learn.microsoft.com/en-us/windows-server/networking/dns/dns-overview, 2025.

[11] Quad9DNS, "Quad9," https://www.quad9.net/, 2025.

[12] CleanBrowsing, "Cb DNS," https://cleanbrowsing.org/, 2025.

[13] No IP, https://www.noip.com/, 2025.

[14] Akamai, https://www.akamai.com/products/global-traffic-management, 2025.

[15] BIND, https://www.isc.org/bind/, 2025.

[16] Unbound, https://nlnetlabs.nl/projects/unbound/about/, 2025.

[17] Technitium, https://technitium.com/dns/, 2025.

[18] GoDaddy, https://www.godaddy.com/, 2025.

[19] Dynadot, https://www.dynadot.com/, 2025.

[20] X. Li, B. Liu, X. Bai, M. Zhang, Q. Zhang, Z. Li, H. Duan, and Q. Li, "Ghost domain reloaded: Vulnerable links in domain name delegation and revocation," in *30th Annual Network and Distributed System Security Symposium*, 2023.

[21] P. Mockapetris, "Domain names - implementation and specification," RFC 1035, Nov. 1987.

[22] M. P. Andrews, S. Huque, P. Wouters, and D. Wessels, "DNS Glue Requirements in Referral Responses," RFC 9471, Sep. 2023.

[23] A. Herzberg and H. Shulman, "Fragmentation considered poisonous, or: One-domain-to-rule-them-all.org," in *2013 IEEE Conference on Communications and Network Security (CNS)*, 2013, pp. 224–232.

[24] D. Kaminsky, "It's the end of the cache as we know it,"

in *Black Hat*, 2008.
[25] X. Zheng, C. Lu, J. Peng, Q. Yang, D. Zhou, B. Liu, K. Man, S. Hao, H. Duan, and Z. Qian, "Poison over troubled forwarders: A cache poisoning attack targeting DNS forwarding devices," in *29th USENIX Security Symposium (USENIX Security 20)*, Aug. 2020.
[26] Google, "Google Public DNS," https://dns.google/, 2025.
[27] CloudFlare, "Cloudflare," https://1.1.1.1/dns/, 2025.
[28] G. Huston, "Measuring the use of DNSsec," https://labs.apnic.net/index.php/2023/09/09/measuring-the-use-of-dnssec/, Sep. 2023.
[29] SIDN, "None of the biggest internet services are DNSsec-enabled," https://www.sidn.nl/en/news-and-blogs/none-of-the-biggest-internet-services-are-dnssec-enabled, Jan. 2025.
[30] APNIC, "Use of DNSsec validation for world," https://stats.labs.apnic.net/dnssec/XA, 2025.
[31] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, "Protocol Modifications for the DNS Security Extensions," RFC 4035, Mar. 2005.
[32] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman, "Assessing DNS vulnerability to record injection," in *Passive and Active Measurement - 15th International Conference, PAM*, 2014.
[33] K. Man, Z. Qian, Z. Wang, X. Zheng, Y. Huang, and H. Duan, "Dns cache poisoning attack reloaded: Revolutions with side channels," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20, 2020.
[34] K. Man, X. Zhou, and Z. Qian, "Dns cache poisoning attack: Resurrections with side channels," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21, 2021.
[35] A. Klein, "Cross layer attacks and how to use them (for DNS cache poisoning, device tracking and more)," in *42nd IEEE Symposium on Security and Privacy*, 2021.
[36] A. Herzberg and H. Shulman, "Security of patched DNS," in *Computer Security – ESORICS 2012*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.
[37] H. Schulmann and M. Waidner, "Fragmentation considered leaking: Port inference for DNS poisoning," in *Applied Cryptography and Network Security - 12th International Conference*, 2014.
[38] J. Stewart, "Dns cache poisoning-the next generation," Jan. 2003.
[39] X. Feng, Q. Li, K. Sun, K. Xu, B. Liu, X. Zheng, Q. Yang, H. Duan, and Z. Qian, "PMTUD is not panacea: Revisiting IP fragmentation attacks against TCP," in *29th Annual Network and Distributed System Security Symposium*, 2022.
[40] P. A. Vixie, D. S. Thomson, Y. Rekhter, and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)," RFC 2136, Apr. 1997.
[41] Dynv6, https://dynv6.com/, 2025.
[42] ClouDNS, https://www.cloudns.net/, 2025.
[43] DNSExit, https://freedomain.one/, 2025.
[44] Synology, https://www.synology.com/en-global, 2025.
[45] ASUSTOR, https://www.asustor.com/, 2025.
[46] K. Man, Z. Wang, Y. Hao, S. Zheng, X. Zhou, Y. Cao, and Z. Qian, " SCAD: Towards a Universal and Automated Network Side-Channel Vulnerability Detection ," in *2025 IEEE Symposium on Security and Privacy*.
[47] K. Fujiwara and P. A. Vixie, "IP Fragmentation Avoidance in DNS over UDP," RFC 9715, Jan. 2025.
[48] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," RFC 6763, Feb. 2013.
[49] W. Hardaker, "Child-to-Parent Synchronization in DNS," RFC 7477, Mar. 2015.
[50] G. Moura, W. Hardaker, J. Heidemann, and M. Davids, "Considerations for Large Authoritative DNS Server Operators," RFC 9199, Mar. 2022.
[51] P. E. Hoffman, A. Sullivan, and K. Fujiwara, "DNS Terminology," RFC 7719, Dec. 2015.
[52] P. E. Hoffman, A. Sullivan, and K. Fujiwara, "DNS Terminology," RFC 8499, Jan. 2019.
[53] P. E. Hoffman and K. Fujiwara, "DNS Terminology," RFC 9499, Mar. 2024.
[54] Y. Zhang, B. Liu, H. Duan, M. Zhang, X. Li, F. Shi, C. Xu, and E. Alowaisheq, "Rethinking the security threats of stale DNS glue records," in *33rd USENIX Security Symposium*, Aug. 2024.
[55] X. Li, W. Xu, B. Liu, M. Zhang, Z. Li, J. Zhang, D. Chang, X. Zheng, C. Wang, J. Chen, H. Duan, and Q. Li, "Tudoor attack: Systematically exploring and exploiting logic vulnerabilities in DNS response pre-processing with malformed packets," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024.
[56] H. Lee, A. Gireesh, R. van Rijswijk-Deij, T. T. Kwon, and T. Chung, "A longitudinal and comprehensive study of the DANE ecosystem in email," in *29th USENIX Security Symposium*, Aug. 2020.
[57] P. Jeitner and H. Shulman, "Injection attacks reloaded: Tunnelling malicious payloads over DNS," in *30th USENIX Security Symposium*, Aug. 2021.
[58] A. Klein, H. Shulman, and M. Waidner, "Internet-wide study of DNS cache injections," in *IEEE Conference on Computer Communications*, 2017.
[59] P. Mockapetris, "Domain names - concepts and facilities," RFC 1034, Nov. 1987.
[60] B. Hubert and R. Mook, "Measures for Making DNS More Resilient against Forged Answers," RFC 5452, Jan. 2009.
[61] GDB, https://www.sourceware.org/gdb/, 2025.
[62] S. D. Plus, https://simpledns.plus/download, 2025.
[63] Knot Resolver, https://www.knot-resolver.cz/, 2025.
[64] MaraDNS, https://maradns.samiam.org, 2025.
[65] W. Wijngaards, "Resolver side mitigations," Internet-Draft draft-wijngaards-dnsext-resolver-side-mitigation-01, Feb. 2009, work in Progress.
[66] A. Klein, H. Shulman, and M. Waidner, "Counting in the dark: DNS caches discovery and enumeration in the

internet," in *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2017.

[67] A. Randall, E. Liu, G. Akiwate, R. Padmanabhan, G. M. Voelker, S. Savage, and A. Schulman, "Trufflehunter: Cache snooping rare domains at large public DNS resolvers," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20, 2020.

[68] Y. Afek, A. Bremler-Barr, and L. Shafir, "NXNSAttack: Recursive DNS inefficiencies and vulnerabilities," in *29th USENIX Security Symposium*, Aug. 2020.

[69] Y. Afek, A. Bremler-Barr, and S. Stajnrod, "NRDelegationAttack: Complexity DDoS attack on DNS recursive resolvers," in *32nd USENIX Security Symposium*, 2023.

[70] X. Li, B. Liu, X. Zheng, H. Duan, Q. Li, and Y. Huang, "Fast ipv6 network periphery discovery and security implications," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 88–100.

[71] M. Kührer, T. Hupperich, J. Bushart, C. Rossow, and T. Holz, "Going wild: Large-scale classification of open DNS resolvers," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15, 2015.

[72] MAXMIND, https://dev.maxmind.com/geoip/geolite2-free-geolocation-data/, 2025.

[73] Dyn, "Dyn DNS," https://help.dyn.com/internet-guide-setup/, 2025.

[74] Cisco, "Opendns," https://www.opendns.com/, 2025.

[75] C. sDNS, "Cnnic sDNS," https://www.sdns.cn/, 2025.

[76] StrongarmDNS, "Strongarmdns," https://strongarm.io/, 2025.

[77] H. Electric, "He DNS," https://dns.he.net/, 2025.

[78] ControlD, "Controld DNS," https://controld.com/free-dns/, 2025.

[79] LibreDNS, "Libredns," https://libredns.gr/, 2025.

[80] SafeSurfer, "Safesurferdns," https://safesurfer.io/, 2025.

[81] OneDNS, "Onedns," https://onedns.net/, 2025.

[82] M. Brandt, T. Dai, H. Shulman, A. Klein, and M. Waidner, "Domain validation++ for mitm-resilient pki," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Oct. 2018.

[83] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, 2019.

[84] C. Schuba, "Addressing weaknesses in the domain name system protocol," Master's thesis, Aug. 1993.

[85] M. Larsen and F. Gont, "Recommendations for Transport-Protocol Port Randomization," RFC 6056, Jan. 2011.

[86] D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee, "Increased DNS forgery resistance through 0x20-bit encoding: security via leet queries," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, 2008.

[87] D. E. E. 3rd and M. P. Andrews, "Domain Name System (DNS) Cookies," RFC 7873, May 2016.

[88] S. Son and V. Shmatikov, "The hitchhiker's guide to DNS cache poisoning," in *Security and Privacy in Communication Networks*, 2010, pp. 466–483.

[89] D. Dittrich and E. Kenneally, "The menlo report: Ethical principles guiding information and communication technology research," *SSRN Electronic Journal*, 2012.

[90] C. Partridge and M. Allman, "Ethical considerations in network measurement papers," *Commun. ACM*, vol. 59, no. 10, p. 58–64, Sep. 2016.

[91] Alibaba Group, "Alibaba cloud services," https://www.alibabacloud.com/, 2025.

[92] Level3, "Level3 DNS," https://www.publicdns.xyz/public/level3.html, 2025.

[93] Comodo, "Comodo DNS," https://www.comodo.com/secure-dns/, 2025.

[94] Yandex, "Yandex.dns," https://dns.yandex.com/, 2025.

[95] DNS for Family, "DNS for Family," https://dnsforfamily.com/, 2025.

[96] Freenom, "Freenom DNS," https://www.freenom.com/en/index.html, 2025.

[97] AdGuard DNS, "Adguard DNS," https://adguard-dns.io/, 2025.

[98] Quad101DNS, "Quad101 DNS," https://101.101.101.101/indexen.html, 2025.

[99] 360, "360 secure DNS," https://sdns.360.net/, 2025.

[100] 114DNS, "114dns," https://www.114dns.com/, 2025.

[101] Tencent, "DNSPod," https://www.dnspod.com/, 2025.

[102] CIRA, "Cira shield DNS," https://www.cira.ca/cybersecurity-services/canadian-shield, 2025.

[103] DNS Forge, "DNS forge," https://dnsforge.de/, 2025.

[104] Baidu, "Baidu DNS," https://dudns.baidu.com/, 2025.

[105] CZ.NIC, "CZ.NIC," https://www.nic.cz/odvr/, 2025.

[106] SkyDNS, "Skydns," https://www.skydns.ru/, 2025.

[107] SafeDNS, "Safedns," https://www.safedns.com/, 2025.

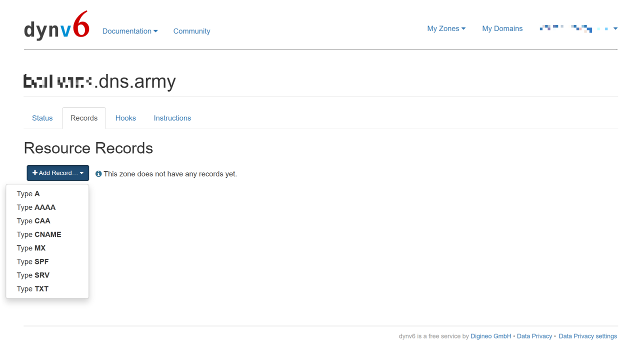
[108] CenturyLink, "Centurylink DNS," https://www.centurylink.com/home/help/internet/dns.html, 2025.

Appendix A
Configure Subdomain Records for Dynv6

As mentioned in Section III-C, Dynv6 [41] provides a service that allows users to configure subdomain resource records for their domains. We exhibit the interface of Dynv6 in Figure 8 and Figure 9, where users can configure various types of resource records (such as `A`, `MX`, `TXT`, etc.) for subdomains and fill in the corresponding values. We employed the techniques described in Section V-C to generate the fragmented response packets, as shown in Figure 10.

Appendix B
End-to-end Fragmentation Attack

We use Docker to set up a local experimental environment to simulate real-world end-to-end fragmentation attacks. Specifically, we employed PowerDNS 4.9.9 with default settings (except for disabling DNSSEC) as the victim resolver and used Python scripts to simulate nameserver that can generate



[redacted]

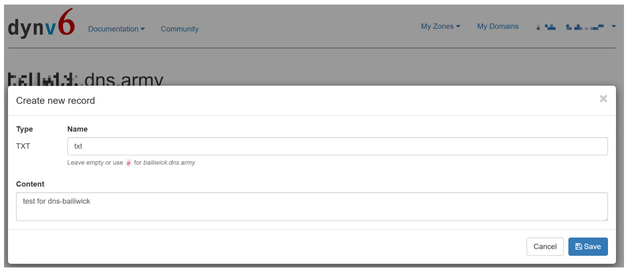
[redacted].dns.army

Resource Records

This zone does not have any records yet.

dynv6 is a free service by Digineo GmbH • Data Privacy • Data Privacy settings

Fig. 8. The configurable record type in Dynv6.



[redacted]

[redacted].dns.army

Create new record

Type	Name
TXT	txt

Leave empty or use @ for *bailiwick.dns.army*

Content

test for dns-bailiwick

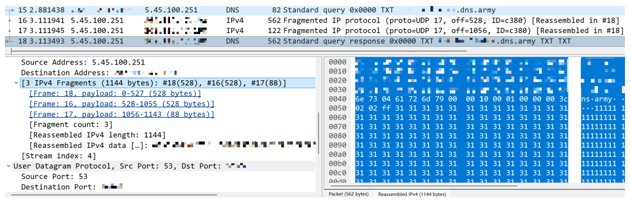
Fig. 9. The process of configuring TXT record in Dynv6.

fragment responses. The script runs on the authoritative server and monitors all incoming traffic; upon detecting a DNS query for our test domain, it intercepts the request and immediately replies with a fragmented DNS response. We launched a fragmentation attack using payload 16 from Table II. To achieve fragmentation, we first utilized three NS records belonging to the attacker's domain (`attacker.root.hit`), then inserted malicious record into the forged second fragment, ultimately hijacking `victim.root.hit`, as shown in Figure 11.

Appendix C
Relevant Texts on Bailiwick in RFCs

We analyze the relevant texts on bailiwick in RFCs in Section IV-A. The following are the original texts in the RFCs:

1) The section 12 of RFC 6763 [48] mentioned "Recursive name servers that talk to multiple authoritative name servers should verify that any records they receive from a given authoritative name server are 'in bailiwick' for that server, and ignore them if not.";
2) The section 3.2.2 of RFC 7477 [49] mentioned "The A and AAAA type flags indicates that the A and AAAA address glue records for in-bailiwick NS records within the child zone should be copied verbatim (with the exception of the TTL field, for which the parent MAY want to select a different value) into the parent's delegation information";
3) RFC 7477 [49] use "Out-of-Bailiwick NS Records" as the title of section 4.3;
4) The section 6 of RFC 7719 [51] mentioned "In-bailiwick: (a) An adjective to describe a name server whose name is either subordinate to or (rarely) the same as the zone origin. In-bailiwick name servers require glue records in their parent zone (using the first of the definitions of 'glue records' in the definition above).";
5) The section 7 of RFC 8499 [52] mentioned "Bailiwick: 'In-bailiwick' is a modifier to describe a name server whose name is either a subdomain of or (rarely) the same as the origin of the zone that contains the delegation to the name server. In-bailiwick name servers may have glue records in their parent zone (using the first of the definitions of 'glue records' in the definition above). (The word 'bailiwick' means the district or territory where a bailiff or policeman has jurisdiction.)" and "'In-bailiwick' names are divided into two types of names for name servers: 'in-domain' names and 'sibling domain' names.";
6) The section 3.6.1 of RFC 9199 [50] mentioned "Multiple record types exist or are related between the parent of a zone and the child. At a minimum, NS records are supposed to be identical in the parent (but often are not), as are corresponding IP addresses in 'glue' A/AAAA records that must exist for in-bailiwick authoritative servers.";
7) The section 7.2.46 of RFC 9499 [53] mentioned "Bailiwick: 'In-bailiwick' and 'Out-of-bailiwick' are modifiers used to describe the relationship between a zone and the name servers for that zone. The dictionary definition of bailiwick has been observed to cause more confusion than meaning for this use. These terms should be considered historic in nature.".

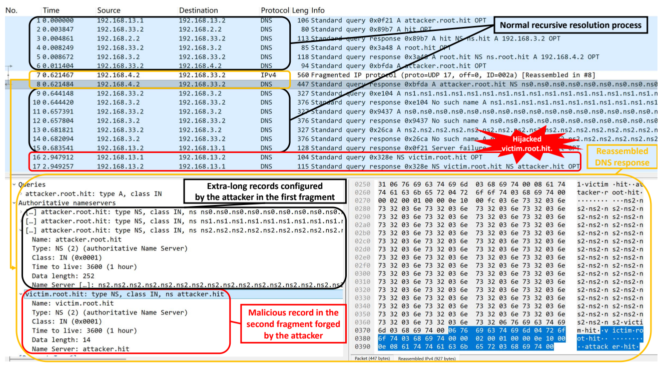

```
15 2.881438 [redacted]        5.45.100.251   DNS    82 Standard query 0x0000 TXT [redacted].dns.army
16 3.111941 5.45.100.251      [redacted]     IPv4  562 Fragmented IP protocol (proto=UDP 17, off=528, ID=c380) [Reassembled in #18]
17 3.111945 5.45.100.251      [redacted]     IPv4  122 Fragmented IP protocol (proto=UDP 17, off=1056, ID=c380) [Reassembled in #18]
18 3.113493 5.45.100.251      [redacted]     DNS   562 Standard query response 0x0000 TXT [redacted].dns.army TXT TXT

    Source Address: 5.45.100.251
    Destination Address: [redacted]
  [3 IPv4 Fragments (1144 bytes): #18(528), #16(528), #17(88)]
      [Frame: 18, payload: 0-527 (528 bytes)]
      [Frame: 16, payload: 528-1055 (528 bytes)]
      [Frame: 17, payload: 1056-1143 (88 bytes)]
      [Fragment count: 3]
      [Reassembled IPv4 length: 1144]
      [Reassembled IPv4 data […]: [redacted]
    [Stream index: 4]
User Datagram Protocol, Src Port: 53, Dst Port: [redacted]
    Source Port: 53
    Destination Port: [redacted]

0000  [redacted]
0040  6e 73 04 61 72 6d 79 00  00 10 00 01 00 00 00 3c   ns·army· ·
0050  02 02 ff 31 31 31 31 31  31 31 31 31 31 31 31 31   ···11111 1
0060  31 31 31 31 31 31 31 31  31 31 31 31 31 31 31 31   11111111 1
0070  31 31 31 31 31 31 31 31  31 31 31 31 31 31 31 31   11111111 1
0080  31 31 31 31 31 31 31 31  31 31 31 31 31 31 31 31   11111111 1
0090  31 31 31 31 31 31 31 31  31 31 31 31 31 31 31 31   11111111 1
00a0  31 31 31 31 31 31 31 31  31 31 31 31 31 31 31 31   11111111 1
00b0  31 31 31 31 31 31 31 31  31 31 31 31 31 31 31 31   11111111 1
00c0  31 31 31 31 31 31 31 31  31 31 31 31 31 31 31 31   11111111 1

Packet (562 bytes)   Reassembled IPv4 (1144 bytes)
```

Fig. 10. A packet returned by Dynv6 as a fragment response.


```
No.  Time       Source         Destination    Protocol Leng Info
 1 0.000000   192.168.13.1   192.168.13.2   DNS    106 Standard query 0x0f21 A attacker.root.hit OPT
 2 0.003847   192.168.33.2   192.168.2.2    DNS     80 Standard query 0x89b7 A hit OPT
 3 0.004861   192.168.2.2    192.168.33.2   DNS    113 Standard query response 0x89b7 A hit NS ns.hit A 192.168.3.2 OPT
 4 0.008249   192.168.33.2   192.168.3.2    DNS     85 Standard query 0x3a48 A root.hit OPT
 5 0.008672   192.168.3.2    192.168.33.2   DNS    118 Standard query response 0x3a48 A root.hit NS ns.root.hit A 192.168.4.2 OPT
 6 0.011404   192.168.33.2   192.168.4.2    DNS     94 Standard query 0xbfda A attacker.root.hit OPT
 7 0.621467   192.168.4.2    192.168.33.2   IPv4   560 Fragmented IP protocol (proto=UDP 17, off=0, ID=002a) [Reassembled in #8]
 8 0.621484   192.168.4.2    192.168.33.2   DNS    447 Standard query response 0xbfda A attacker.root.hit NS ns0.ns0.ns0.ns0.ns0.ns0.ns0.ns0.ns0
 9 0.644148   192.168.33.2   192.168.3.2    DNS    327 Standard query 0xe104 A ns1.ns1.ns1.ns1.ns1.ns1.ns1.ns1.ns1.ns1.ns1.ns1.ns1.ns1.n
10 0.644420   192.168.3.2    192.168.33.2   DNS    376 Standard query response 0xe104 No such name A ns1.ns1.ns1.ns1.ns1.ns1.ns1.ns1.ns1.ns1
11 0.657391   192.168.33.2   192.168.3.2    DNS    327 Standard query 0x9437 A ns0.ns0.ns0.ns0.ns0.ns0.ns0.ns0.ns0.ns0.ns0.ns0.ns0.n
12 0.657804   192.168.3.2    192.168.33.2   DNS    376 Standard query response 0x9437 No such name A ns0.ns0.ns0.ns0.ns0.ns0.ns0.ns0.ns0
13 0.681821   192.168.33.2   192.168.3.2    DNS    327 Standard query 0x26ca A ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.n
14 0.682094   192.168.3.2    192.168.33.2   DNS    376 Standard query response 0x26ca No such name A ns2.ns2.ns2.ns2.ns2.ns2.ns2
15 0.683541   192.168.13.2   192.168.13.1   DNS    128 Standard query response 0x0f21 Server failure OPT
16 2.947912   192.168.13.1   192.168.13.2   DNS    104 Standard query 0x328e NS victim.root.hit OPT
17 2.949257   192.168.13.2   192.168.13.1   DNS    115 Standard query response 0x328e NS victim.root.hit NS attacker.hit OPT

Queries
  attacker.root.hit: type A, class IN
Authoritative nameservers
  […] attacker.root.hit: type NS, class IN, ns ns0.ns0.ns0.ns0.ns0.ns0.ns0.ns0.ns0.
  […] attacker.root.hit: type NS, class IN, ns ns1.ns1.ns1.ns1.ns1.ns1.ns1.ns1.ns1.r
  […] attacker.root.hit: type NS, class IN, ns ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.r
    Name: attacker.root.hit
    Type: NS (2) (authoritative Name Server)
    Class: IN (0x0001)
    Time to live: 3600 (1 hour)
    Data length: 252
    Name Server […]: ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2.ns2
  victim.root.hit: type NS, class IN, ns attacker.hit
    Name: victim.root.hit
    Type: NS (2) (authoritative Name Server)
    Class: IN (0x0001)
    Time to live: 3600 (1 hour)
    Data length: 14
    Name Server: attacker.hit

0250  31 06 76 69 63 74 69 6d  03 68 69 74 00 08 61 74   1·victim ·hit··at
0260  74 61 63 6b 65 72 04 72  6f 6f 74 03 68 69 74 00   tacker·r oot·hit·
0270  00 02 00 01 00 00 0e 10  00 fc 03 6e 73 32 03 6e   ········ ···ns2·n
0280  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
0290  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
02a0  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
02b0  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
02c0  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
02d0  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
02e0  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
02f0  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
0300  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
0310  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
0320  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
0330  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
0340  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
0350  73 32 03 6e 73 32 03 6e  73 32 03 6e 73 32 03 6e   s2·ns2·n s2·ns2·n
0360  73 32 03 6e 73 32 03 6e  73 32 06 76 69 63 74 69   s2·ns2·n s2·victi
0370  6d 03 68 69 74 00 06 76  69 63 74 69 6d 04 72 6f   m·hit··v ictim·ro
0380  6f 74 03 68 69 74 00 00  02 00 01 00 00 0e 10 00   ot·hit·· ········
0390  0e 08 61 74 74 61 63 6b  65 72 03 68 69 74 00      ··attack er·hit·

Packet (447 bytes)   Reassembled IPv4 (927 bytes)
```
Normal recursive resolution process
Hijacked victim.root.hit.
Reassembled DNS response
Extra-long records configured by the attacker in the first fragment
Malicious record in the second fragment forged by the attacker

Fig. 11. A demonstration of an end-to-end fragmentation attack against PowerDNS using payload 16.

Appendix D
Public DNS Resolver Behavior

We collect 30 popular public DNS resolvers and use the test cases in Table II to find the differences in their DNS resolution and cache processes, as shown in Table VI.

TABLE VI
DNS RESOLUTION AND CACHE BEHAVIOR OF 30 POPULAR PUBLIC DNS RESOLVER VENDORS.

Public DNS Vendors	IPv4 Address	T1 Attack				T2 Attack												T3 Attack			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Cloudflare DNS[27]	1.1.1.1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
CNNIC sDNS[75]	1.2.4.8	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✓
Level3 DNS[92]	4.2.2.1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Google Public DNS[26]	8.8.8.8	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Comodo Secure DNS[93]	8.20.247.10	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Quad9 DNS[11]	9.9.9.9	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗
Strongarm DNS[76]	52.3.100.184	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✗
Hurricane Electric DNS[77]	74.82.42.42	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✗	✓	✓	✓	✗	✓	✗
ControlD DNS[78]	76.76.2.0	✓	✓	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓	✓	✗	✗	✓	✗
Yandex.DNS[94]	77.88.8.1	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
DNS for Family[95]	78.47.64.161	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Freenom World DNS[96]	80.80.80.80	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
LibreDNS[79]	88.198.92.222	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	✓	✗	✓	✗
AdGuard DNS[97]	94.140.14.14	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Quad101 DNS[98]	101.101.101.101	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
360 Secure DNS[99]	101.226.4.6	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗
Safe Surfer DNS[80]	104.155.237.225	✗	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
114DNS[100]	114.114.114.114	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
OneDNS[81]	117.50.10.10	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
DNSPod Public DNS+[101]	119.28.28.28	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
CIRA Shield DNS[102]	149.112.121.10	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
DNS Forge[103]	176.9.1.117	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗
Baidu DNS[104]	180.76.76.76	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
CZ.NIC ODVR DNS[105]	185.43.135.1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Clean Browsing DNS[12]	185.228.168.10	✓	✓	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓	✓	✗	✗	✗	✓	✗	✗	✗
SkyDNS[106]	193.58.251.251	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
SafeDNS[107]	195.46.39.39	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓
CenturyLink DNS[108]	205.171.2.26	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗
OpenDNS[74]	208.67.220.120	✗	✗	✓	✗	✓	✓	✓	✓	✓	✗	✗	✓	✗	✗	✗	✓	✓	✗	✓	✓
Dyn DNS[73]	216.146.35.35	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓

✓: Vulnerable. ✗: Not Vulnerable.

APPENDIX E
ANALYSIS OF RESPONSES WITH NS RECORDS

Since the number of `answer` response containing `NS` records is relatively large, we conduct a more detailed analysis in Table VII besides Section VI-A.

First, we find that 36.8% of the `answer` packets carry `NS` records in the `authority` section. Next, we classify the `NS` records based on their `RNAME` and find that the vast majority are records for the zone apex, confirming that no data for sibling domains appears in the `answer` response. Finally, based on the `NS` records obtained during the top-down recursive resolution process, we find that most of the records carrying zone apex content are consistent with those in the `referral`, and only 2,524 domains attempt to provide `NS` records that differ completely from those in the `referral`.

TABLE VII
ANALYSIS OF ANSWER PACKETS CONTAINING NS RECORD.

Records type provided by `answer` packets	**# Packets**	**%**
Total number of `answer` packets	127,484	-
Total number of `answer` packets with NS records	46,893	100%
Provide NS records for a domain out of the zone	60	0.13%
Provide NS records for zone apex	46,404	98.96%
Provide `NS` records consistent with `referral`	43,880	93.57%
Provide `NS` records inconsistent with `referral`	2,524	5.38%
Provide NS records for a parent domain of QNAME	229	0.49%
Provide NS records for a domain under the QNAME	200	0.43%