Zombie Awakening: Stealthy Hijacking of Active Domains through DNS Hosting Referral

Eihal Alowaisheq
Indiana University
King Saud University
ealowais@indiana.edu

Siyuan Tang
Indiana University
tangsi@indiana.edu

Zhihao Wang
Institute of Information Engineering,
Chinese Academy of Sciences
wangzhihao@iie.ac.cn

Fatemah Alharbi
Taibah University
fmhharbi@taibahu.edu.sa

Xiaojing Liao
Indiana University
xliao@indiana.edu

XiaoFeng Wang
Indiana University
xw7@indiana.edu

ABSTRACT

In recent years, the security implication of stale NS records, which point to a nameserver that no longer resolves the domain, has been unveiled. Prior research studied the stale DNS records that point to expired domains. The popularity of DNS hosting services brings in a new category of stale NS records, which reside in the domain's zone (instead of the TLD zone) for an active domain. To the best of our knowledge, the security risk of this kind of stale NS record has never been studied before. In our research, we show that this new type of stale NS record can be practically exploited, causing a stealthier hijack of domains associated with the DNS hosting service. We also performed a large-scale analysis on over 1M high-profile domains, 17 DNS hosting providers and 12 popular public resolver operators to confirm the prevalence of this security risk. Our research further discovers 628 hijackable domains (*e.g.,* 6 government entities and 2 payment services), 14 affected DNS hosting providers (*e.g.,* Amazon Route 53), and 10 vulnerable public resolver operators (*e.g.,* CloudFlare). Furthermore, we conducted an in-depth measurement analysis on them, thus providing a better understanding of this new security risk. Also, we explore the mitigation techniques that can be adopted by different affected parties.

CCS CONCEPTS

• **Security and privacy** → *Authentication*; **Security protocols**; *Vulnerability management*.

KEYWORDS

Domain hijacking; DNS cash poisoning; DNS hosting services

ACM Reference Format:
Eihal Alowaisheq, Siyuan Tang, Zhihao Wang, Fatemah Alharbi, Xiaojing Liao, and XiaoFeng Wang. 2020. Zombie Awakening: Stealthy Hijacking of Active Domains through DNS Hosting Referral. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3372297.3417864

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

ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3417864

1 INTRODUCTION

The Domain Name System (DNS) has been the cornerstone of the Internet and its immense scalability. It provides the vital distributed directory service which associates information with the domain names given to various online entities and translates between domains and their Internet Protocol (IP) addresses. For this purpose, the DNS operates through a complicated hierarchical infrastructure to delegate the management of the domains within a portion of the namespace (called DNS zone) and their mapping of Internet resources to a set of authoritative nameservers. Through this infrastructure, a client-side resolver can recursively discover and query the nameservers in charge of different zones, from the root to the Top Level Domain (TLD, *e.g.,* .com), to the Second Level Domain (SLD, *e.g.,* example.com), and so on, until an authoritative answer (*e.g.,* an IP address) is provided by a server. Such an infrastructure is security-critical. If misconfigured, it could inflict damage of devastating impact, ranging from denial of access to Internet resources [9, 23, 24, 37, 43, 61, 65] to hijacking of domains from their legitimate owners [52, 75]. Despite the long-standing effort from the security community to safeguard this infrastructure, its complexity and distributed nature continue to expose it to emerging security threats.

Menaces of stale NS records in the SLD zone. In recent years, researchers have identified the security implications of stale NS records, where the nameserver that the record points to no longer resolves the domain. For instance, prior research [52] looks into dangling NS (Dare-NS) records, where the nameserver domains that NS records point to are expired and the adversary could purchase the domain to hijack this resource. Another example of domain hijacking through stale NS records emerges with the popularity of DNS hosting services (*e.g.,* Amazon Route 53 [6] and GoDaddy DNS hosting [36]). At these services, users host their DNS records in the service provider's nameservers. Once these records become stale, an adversary can claim the nameserver domain and direct the traffic. Some blog posts discussed the exploitation of this vulnerability [12, 13]. However, the proposed attack works effectively if stale NS records are in the TLD zone. Once a domain is hijacked, it could be easily noticed by the domain owner because such misconfiguration appears in the normal resolution path.

In our research, we found that the popularity of DNS hosting services brings in a new category of stale NS records – stale NS records in the SLD zone: unlike Dare-NS, the nameserver pointed to by the record still exists. Also, those stale NS records are in the SLD zone instead of TLD zone, which makes the misconfiguration difficult to discover. Specifically, the attacker can exploit this vulnerability to hijack a domain through a "hidden" resolution path. For example, stale NS records in the SLD zone exist when importing the domain's zone information from one DNS hosting provider into a new DNS server, where the nameserver provided by the hosting provider no longer resolves the domain. After that, during the domain resolution, the stale NS record at the SLD zone will not be normally used unless cached, since the nameserver received from the TLD will directly return the A record to find out the domain's IP address, as long as the NS records in the TLD zone (*e.g.,* .com) are up-to-date (*i.e.,* only pointing to the current nameserver). Our research shows that the stale NS records at the SLD can actually be practically exploited, causing a stealthy hijack of active domains.

Zombie awakening attack. More specifically, we found those stale NS records can be reactivated within a DNS hosting service when the adversary makes an unauthorized claim on another party's domain to host the DNS records. It indicates a practical scenario for the attacker to inject a valid NS response to the resolver and further hijack the domain. In particular, by strategically querying a resolver for the domain's NS records, the adversary will make the resolver cache NS records that reside at the SLD zone as the valid NS response. So a follow-up query for the IP address of the same domain could trigger the stale NS record, which opens the attack avenue by poisoning the resolver with the adversary's A record configured at the DNS hosting side. Interestingly, even in the presence of inconsistency (that is, the absence of the stale record in the TLD zone), the NS records in the SLD zone will be used due to higher trust level [32]. We call the threat *zombie awakening* or *Zaw* attack, and the vulnerable stale NS records *zombie referrals* or *Zrefs*.

In our research, we found that the Zaw threat is completely realistic: we successfully exploited our own domains at various leading DNS hosting providers and the resolvers maintained by major operators. Also, we demonstrated that this threat is serious, through a large-scale analysis that discovered 628 vulnerable domains from Alexa's top 1M domains, including those of education institutions, government entities, and companies. For this purpose, we designed a semi-automated approach, called *ZreFinder*, to automatically discover Zrefs. ZreFinder systematically collects the NS records for each target domain, looks for the inconsistency between the TLD zone and the SLD zone, and identifies the stale records associated with DNS hosting providers. For each DNS hosting provider, our approach further determines if a domain can be claimed without authorization at the provider to build an illicit resolution path.

Measurement and mitigation. Most importantly, by running ZreFinder on Alexa's top 1M domains and a set of .edu and .gov domains, we found that this new threat is indeed significant and impactful. More specifically, we discovered 628 vulnerable domains, including 6 government domains associated with Colombia, Malaysia, and Saudi Arabia, and domains for 3 universities. Also flagged are those belonging to critical public services and big companies, *e.g.,* Pittsburgh airport, Croatia airlines, and the FastSpring E-commerce platform. Also, we found that the Zrefs of such domains have been out there for a while: 89.97% hijackable domains have Zrefs for at least 30 days.

Furthermore, our research shows that prominent DNS hosting services and popular public resolvers are vulnerable to the Zaw attack. Among them are 14 leading hosting providers such as Amazon Route 53, Hetzner Online GmbH, and more than 7K resolvers, including those operated by CloudFlare, Quad9, and OpenNIC. Such resolvers turned out to be easily poisoned: merely 6.5 queries on average are found to be adequate to contaminate their cache with malicious A records implanted on the hosting service side. This enables an adversary to stealthily control the traffic to the target domain whenever the poisoned resolvers are used, for purposes such as phishing, malware distribution, *etc.*

We further discuss how to mitigate this new security risk. Our ZreFinder can be utilized to find Zrefs so they can be removed by domain owners. DNSSEC can also be leveraged to defeat Zaw attacks, although it has not been extensively deployed. On the side of the DNS hosting provider, we present a simple verification technique to ensure that a domain cannot be claimed without authorization by requesting the domain owner to add a randomly-generated NS record at the TLD level.

Contributions. The contributions of the paper are outlined as follows:

- We discovered a new security risk, Zaw attack, in the DNS infrastructure. To our knowledge, this Zaw attack has *never* been shown practically before in the public, which demonstrates the grave risk posed by a stale NS record in the SLD zone, particularly with the emergence of DNS hosting services.
- We developed a new technique for automatic discovery of the domains with Zrefs. By scanning over 1M high-profile domains, we identified 628 hijackable domains, affecting government agencies, public services, large corporations, *etc.*
- We conducted a large-scale measurement study to understand Zaw attack, including identifying 14 affected DNS hosting providers and 10 vulnerable public revolver operators, and investigating the attack complexity (e.g., the selection rate of Zref) in the real world.
- We provided suggestions for the affected parties to mitigate this new threat.

Roadmap. The rest of the paper is organized as follows: Section 2 presents the background of the research; Section 3 describes the Zaw attack; Section 4 elaborates on the technique for finding Zrefs for a large-scale measurement study; Section 5 provides our measurement findings; Section 6 discusses the potential mitigations; Section 7 compares our work with prior research and Section 8 concludes the paper.

2 BACKGROUND

In this section, we provide background information about the DNS structure and its hosting providers, along with the assumptions made in our research.

2.1 DNS Resolution and Caching

DNS structure and resolution. As mentioned earlier, the Domain Name System (DNS) resolves Fully Qualified Domain Names

(FQDNs) to their corresponding IP addresses (and vice versa) through a hierarchical infrastructure. At the top of the hierarchy is the root ("."), under which are a series of Top Level Domains (TLDs, *e.g.,* `.com`, `.net`, and `.org`). They are followed by Second Level Domains (SLDs, *e.g.,* `example.com`), which their owners register with registrars. More details of this hierarchy can be found in RFCs [57, 58]. Throughout the paper, we refer to SLD as domain. In the DNS hierarchy, each node includes referral information for its child nodes, which is essential to the recursive DNS resolution process. In other words, the root zone contains an `NS` RR set (called RRSet), with each RR pointing to a nameserver in charge of its children zone at the TLD level. Similarly, TLDs have the NS RRSets that point to SLDs.

The DNS resolution is usually done recursively by a DNS resolver. For example, when a client sends a DNS query to retrieve the IP address of an FQDN `www.example.com`, if the cache of the resolver carries no information about the requested domain, the resolver first forwards the query to the root server to trigger the recursive process. The root server then redirects the resolver to the `.com` TLD server, which further refers the resolver to the authoritative nameserver of `example.com` to get the IP address as a response, then the resolver sends the response back to the client. To optimize this process, resolvers can choose to cache any received DNS records for future resolutions.

DNS response. Resolution information is organized into Resource Records (RRs) with the following format: *name, Time-to-Live (TTL), class, type,* and *data*, as illustrated in Figure 1. The *<name, type, class>* serves as the key when searching for a record. Here, *name* states the name of the record, *type* specifies the record type, such as `A`, `NS`, and `CNAME`, and *class* defines the protocol, *e.g.,* `IN` for the Internet protocol in Figure 1. Also, each record has a TTL (time-to-live) field that determines the lifespan (in seconds) of the record when cached by a resolver. Note that resolvers may not adhere to the TTL and may, instead, set its minimum and maximum limit for a record [42].

A DNS response includes a series of RRs in three main sections: *answer*, *authority*, and *additional*. Figure 1 shows a snippet of a DNS response to an `A` record query for `example.com`. The answer section contains the RR that answers the query, *i.e.,* the domain's IP address. The authority section holds a set of the authoritative nameservers for the domain. The additional section carries other RRs related to the query, the IPs of these authoritative nameservers in the example. It is common for a domain to have more than one authoritative nameservers. For example, in Figure 1, the domain `example.com` has two nameservers (`[a,b].iana-servers.net`). During the resolution process, the resolver usually randomly picks one to balance the workload over the nameservers, or chooses the one statistically based on shorter *Round-Trip Time* (RTT) [80].

DNS caching. Resolvers often cache resolution results to improve performance. For each query, the resolver first checks its cache and directly replies to the client if the corresponding RR is found (a cache hit). A security risk of the caching is DNS poisoning [5, 44, 45, 72]: the attacker may inject a malicious DNS response to the cache to direct victims to a server under his control. For this purpose, the attacker needs to win the race against a legitimate response from the domain's nameservers. The attack becomes increasingly hard due to the protection in place for today's resolvers.

```
;; ANSWER SECTION:
example.com.         46294   IN   A      93.184.216.34

;; AUTHORITY SECTION:
example.com.         43020   IN   NS     a.iana-servers.net.
example.com.         43020   IN   NS     b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.  118512  IN   A      199.43.135.53
b.iana-servers.net.  35275   IN   A      199.43.133.53
```

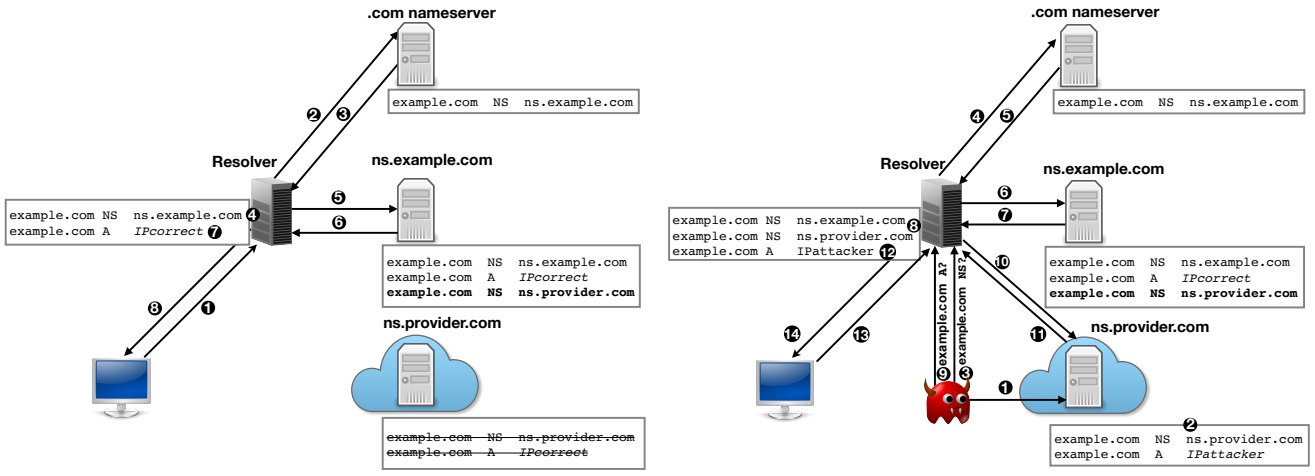
Figure 1: The structure of the DNS response to an `A` record query for `example.com` showing the three main sections.

More specifically, such protection falls into three categories: (1) challenge-response (*e.g.,* Source Port Randomization [22]) ; (2) cryptographic defense (*e.g.,* DNSSEC [77]); and (3) caching rules, including *Bailiwick rule* [72], and *Credibility rule* [32]. The bailiwick rule protects the cache by ignoring out-of-bailiwick records in the authoritative or additional sections of a DNS response. For example, if a response to `.com` returned a mapping of `.org` in the additional or authority sections, this record will not be cached. The credibility rule determines when to overwrite existing records in the cache. Each DNS record is assigned a trust level, or ranking. The resolver overwrites an existing record if it receives a new one with a higher/equal trust level. The trust level is based on two criteria: (1) whether the response is received from an authoritative nameserver or not; and (2) in which section of the response the record is placed. For instance, a DNS response, which is an authoritative answer in the authority section from an authoritative nameserver (*e.g.,* `iana-servers.net`), will have a higher trust level than a response from non-authoritative nameservers (*e.g.,* `.net` zone file).

Some new attacks can be used to circumvent the protection. However, they are all based upon strong assumptions and have difficulty succeeding in practice. For example, prior research [46] shows that an attacker can abuse caching rules to place a poisonous record in a resolver's cache, requiring the attacker to craft a DNS response to bypass the challenge-response protection. As another example, the study [42] found that the attacker can revive a revoked domain by extending its TTL value, when the domain's nameserver is assumed to be under his control, without providing a practical scenario in which the assumption is possible. By comparison, we do not make these assumptions in our research, and our attack works in practice whenever a Zref is found (Section 4). Also, the deployment of DNSSEC [77] may mitigate the risk due to the authorization between parent/child zones. However, the protection still has not been widely used [16, 73, 74, 79].

2.2 DNS Hosting

Operations of DNS hosting services. A DNS hosting provider is a service that provides authoritative nameservers to help its customers manage the DNS records of their domains. To use such a service, one first creates an account with the provider and adds a domain under her account. Then, the provider assigns a set of nameservers to manage the domain and respond to its DNS queries. This is achieved in various ways across different hosting services. For instance, DigitalOcean [26] offers a fixed set of nameservers (*i.e.,* `ns[1,2,3].digitalocean.com`) to all of its customers, while CloudFlare [18] and GoDaddy DNS hosting [36] randomly select two or more nameservers (*e.g.,* `[hank, val].ns.cloudflare.com`; `[ns51,52].domaincontrol.com`) from their nameserver pools for



(a) DNS configuration and resolution for a vulnerable domain

(b) DNS configuration and resolution for a hijacked domain

Figure 2: Overview of the threat model. (a) shows the case of vulnerable domain (*i.e.,* `example.com`). (b) shows the case of the domain being hijacked (Bold represents the stale NS records. For simplicity, here we ignore queries to the root DNS servers.)

each user. In our study, we observe that all DNS hosting providers (Table 1) validate the absence of an active zone for a domain in their services before hosting it, except for the Amazon Route 53 [6], which allows the same domain to be hosted under different accounts and assigns different nameservers to manage it for these accounts. Similar protection measures are taken by most providers to prevent subdomain hijacking.

Security risks. Once a customer stops using the hosting service, the provider's nameservers stop responding to the queries for her domain[1] and return the REFUSED status code instead. If the customer does not clear up her NS records (in the TLD zone or the new SLD zone) that point to these deactivated hosting nameservers, then these records become stale and pose security risks [12, 13]. In particular, we found that most providers do not validate their customers' ownership of the domains they claim under their accounts, possibly due to the belief that to have the domains fully resolved, the NS records have to be properly updated at the TLD level (*i.e.,* at the registrar) and therefore are under the full control of authorized parties. Even when the TLD zone contains the stale RRs, the misconfiguration is conspicuous, since failure in resolving is likely to happen and be noticed by the domain owner. Our study, however, shows that it is possible to trigger an unexpected resolution path through the stale NS record at the SLD zone, even when the record is not present in the TLD zone. This opens a new and stealthy avenue to exploit the stale record for hijacking a domain.

3 ZOMBIE AWAKENING ATTACK

Threat model. We present the Zaw attack, assuming the presence of at least one stale NS record in a target domain's SLD zone that points to a DNS hosting service that is not verifying the ownership of the domain being claimed. This enables an adversary to stealthily hijack a domain with ongoing service, by exploiting both the nameserver at the hosting service and the targeted resolvers, to divert the traffic the domain receives. Later we show that these assumptions are *realistic* (Sections 4.3 and 5.2): our analysis of over 1M high-profile domains has brought to light 628 hijackable [2] domains with Zrefs that enables such a Zaw attack.

Stale NS record at SLD zone. As mentioned earlier, the Zaw attack aims at the stale NS record at the SLD zone. The presence of such a record is less noticeable since it does not normally affect the resolution of a domain in the absence of the attack. As an example, Figure 2a shows a client that attempts to connect to a vulnerable domain `example.com`. In this case, the NS record at both the TLD and SLD zones points to `ns.example.com`. A domain typically has multiple NS records, and ideally, its NS RRSets are identical at the TLD and the SLD zones. The SLD zone further contains its A record that points to the IP address set by the domain's owner. In reality, the domain's zone may include a stale NS record pointing at a DNS hosting provider `ns.provider.com` that no longer manages the domain. The existence of such a record could be caused by the importation of the SLD zone information from the provider (`ns.provider.com`) to the new DNS server (`ns.example.com`) when moving the domain's management out of the provider. Oftentimes, the RRs that do not affect the normal resolution of the domain are left there without being cleaned up.

More specifically, when the stale NS record remains in the TLD zone, it can be used to locate the authoritative nameserver of the domain (which will fail) and, therefore, could make the domain temporarily unavailable. However, if it just appears in the SLD zone, it becomes almost unnoticeable. Figure 2a illustrates the resolution process in the presence of the stale record. As mentioned earlier (Section 2.1), when a resolver receives a request to map the domain `example.com` to its IP address (❶) and cannot find the answer from

[1] CloudFlare is an exception and continues to resolve the domain until the referral records are changed.

[2] Through out this paper we use the terms hijackable, vulnerable, and exploitable interchangeably.

its cache, it forwards the query to the TLD's DNS server (❷), which in turns responds with the domain's NS records at the TLD (pointing to `ns.example.com`) (❸). This response (❹) could be cached by the resolver, which continues to query the returned nameserver at the SLD zone (❺) to get the domain's A record (*i.e.,* $IP_{Correct}$)(❻). This response is then sent back to the client (❽) and cached by the resolver until its TTL expires (❼). As we can see here, during this process, the stale NS record at the SLD zone is not used[3], and therefore, its presence is almost oblivious.

Attack. Such a hidden stale record, however, can still be exploited by an adversary to build and later trigger a resolution path that leads to domain hijacking. Figure 2b illustrates how this zombie awakening attack works: the adversary first claims the vulnerable domain at the DNS hosting provider and then poisons the cache of the targeted resolver with the A record through the stale NS record.

Specifically, the adversary sets an account at the hosting provider directed by the stale NS record of the domain `example.com` (❶) to claim the domain and set its RRs (❷) at the same nameserver on the stale record (*i.e.,* `ns.provider.com`) (see Section 2.2). Here, the adversary creates an A record for the domain that points to an IP address under his control (*i.e.,* $IP_{Attacker}$).

To hijack the domain, the adversary activates the stale resolution path through `ns.provider.com` to attack the A record by poisoning a target resolver. For this purpose, he strategically queries the resolver for the NS records of the domain (❸). If no authoritative answer is found in the resolver's cache, the resolver sends the request to the TLD's nameserver (❹) and receives a NS RR with `ns.example.com` (❺). This nameserver is further contacted for NS RRs in the SLD zone (❻), which results in all such records, including the one with `ns.example.com` and the stale record pointing to `ns.provider.com` to be cached at the resolver side (❼). Note that in this case, the NS RR from the TLD zone is replaced by the RRs (including the stale one) from the domain zone (❽) as the authoritative response from the SLD zone has a higher trust level than a referral from the TLD zone.

After that, the adversary can query the resolver for the domain's A record (❾). Given that both NS records are cached, if chosen randomly, the stale record has a 50% of chance to be used to find the A record (❿). When this happens, the entire attack resolution path is activated: `ns.provider.com` returns the attack A record (⓫), which is cached by the resolver until its TTL expires (⓬). The adversary can ensure the success of such contamination by repeatedly querying the resolver until he receives $IP_{Attacker}$. Later, when the resolver is contacted to resolve the domain (⓭) by a legitimate client, $IP_{Attacker}$ is given as the response, and the traffic of the follow-up visit is then directed to the attacker's IP address (⓮).

In our research, we implemented this attack on our domains to check 17 hosting providers (*e.g., AWS Route 53* and *GoDaddy DNS hosting*) and 12 resolver operators (*e.g., CloudFlare* and *Quad9*). The results show a high success rate (Sections 4.3), confirming that the threat is indeed realistic and serious. We further demonstrate that vulnerable domains with the exploitable stale NS records widely exist using ZreFinder (Section 4).

[3] An exception is that the nameserver could be configured to also send the SLD NS RRs to the resolver.

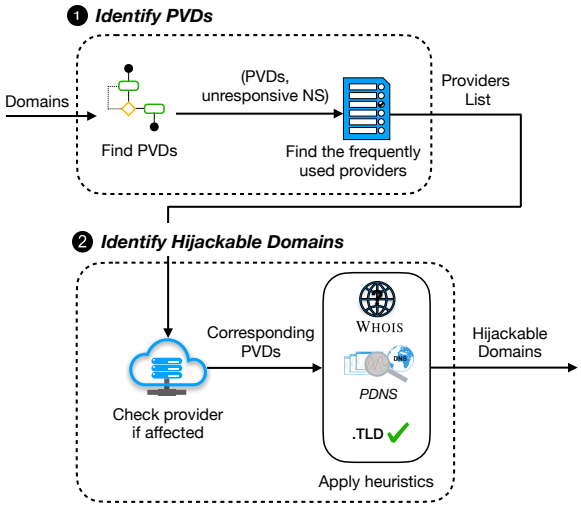


Figure 3: Workflow of ZreFinder to identify hijackable domains, where PVDs represents the potential vulnerable domains.

4 UNDERSTANDING ZAW RISKS

In this section, we first present ZreFinder, a semi-automatic methodology for finding Zrefs, and how to utilize it for a large-scale discovery of domains vulnerable to the Zaw attack. We also describe a study to analyze whether open DNS resolvers can be used to "awaken" zombie resolution paths for domain hijacking.

Figure 3 illustrates the workflow of ZreFinder: it first runs an automated mechanism to scan public domains and identify those containing stale NS records at the domain level (❶), called *potentially vulnerable domains* or *PVDs*; then, from the nameserver in those records, our approach determines those associated with DNS hosting services. Furthermore, we manually analyze these services to find out whether they allow a zombie resolution path to be built and whether some hidden policies on handling public domains are in place to prevent the PVDs from being exploited (❷). The outputs of the analysis are a list of vulnerable domains and their stale NS records, which are considered Zrefs.

In our study, we collected Alexa's top-1M [4] domains, together with around 20K education and government domains collected from Farsight's DNSDB [33], Table 2. From these domains, ZreFinder reported 4,914 PVDs with unresponsive NS records in their SLD zones (Section 4.1). Furthermore, our approach discovered 12 vulnerable providers that allow one to make unauthorized claims on their related PVDs among 17 popular DNS hosting providers. As a result, 628 domains were confirmed to be hijackable (Section 4.2). Finally, we present our study on resolvers which included: the 46 most popular resolvers [78], 11K open resolvers [28], and an organizational resolver, show that the vast majority of them can be easily manipulated to trigger the domain hijacking attack.

4.1 PVD and Provider Discovery

Methodology. To identify PVDs, we built a scanner to automatically inspect a large number of domains. Given a domain *dn*, our approach first locates all its NS records in the SLD zone (but not in the TLD zone) and then evaluates whether the nameservers they point to still resolve *dn*'s DNS query requests. Algorithm 1

describes process. Specifically, for each domain (*e.g.,* `example.com`), we collect two `NS` record sets: the TLD NS RRSet (*TLDns*) and the SLD NS RRSet (*SLDns*) for a differential analysis. Here the TLDns are found by querying the domain's TLD authoritative nameserver (*e.g.,* `a.gtld-servers.net`) through `dig NS example.com @a.gtld-servers.net`. We then continue to query all the returned nameservers (*e.g.,* `ns[1,2].example.com`) to acquire the SLDns, *e.g.,* through `dig NS example.com @ns[1,2].example.com`. After that, our approach compares the TLDns and the SLDns to identify the nameservers that appear on the `NS` records in SLDns but not in the TLDns. For each such nameserver, we further look into whether its `NS` record is indeed stale; that is, the target domain *dn* is not configured at the nameserver [68], which causes any query for the domain on the server to return a `REFUSED` status code [25]. When the domain is confirmed to include such a stale `NS` record in its SLD zone, it is considered to be a PVD.

Once PVDs have been discovered, our approach analyzes them to identify the potential DNS hosting providers involved (pointed to by their stale `NS` records), particularly the popular ones. To this end, our approach first extracts the domain name from all nameservers on the stale records and then selects the most prevalent ones to find their providers. In most cases, the domain name for the nameserver carries the provider's name (*e.g.,* `hank.ns.cloudflare.com`). Otherwise, we have to resort to `WHOIS` information for the provider's domains: for example, the nameserver (`ns51.domaincontrol.com`) does not provide any indicator about who the provider is, but the `WHOIS` record for the domain shows that the provider is GoDaddy.

Findings. To study PVDs and their providers in the wild, we scanned 1,016,449 domains, including Alexa's top 1M websites, and education and government domains gathered from the Farsight passive DNS records (PDNS), based upon their sponsored TLDs (sTLDs) such as .edu, and .gov. We also included some country-specific sTLDs such as China (CN) and Saudi Arabia (SA). The selection of SA domains was made due to the massive increase of attacks recently targeting Saudi Arabian services [3, 54, 66]. Thus, an adversary may leverage the trust inherited from vulnerable domains under the sTLDs to execute a more deceiving attack. Similarly, CN domains were selected due to the observed pervasiveness of DNS resolution interception, in which some were abused for illicit traffic monetization [51]. This suggests that adversaries may also consider launching a Zaw attack against sTLDs domains. Table 2 (in Appendix A) presents the number of the domains under each sTLDs used in our dataset. Scanning all these 1,016,449 domains, ZreFinder discovered 4,914 PVDs along with their corresponding SLD stale `NS` records and further reported the top 11 most prevalent DNS hosting providers associated with these records. Also, we included 6 additional popular providers that offer free service, according to an online list [69]. Table 1 presents these services along with the number of associated PVDs. These services were further manually inspected to verify whether the PVDs are indeed exploitable (Section 4.2).

4.2 Hijackable Domain Identification

Just because a PVD has Zref (*i.e.,* a stale `NS` record pointing to a DNS hosting provider) it does not necessarily mean that the domain is hijackable. The provider may have a protection in place, presenting a different set of nameservers than what the requested domain already has as stale records. An interesting finding is that even in the absence of the protection, as happened in most of the providers we investigated, some PVDs still may not be hijackable. This is due to the observation that the domain could still be possessed by another account at the provider and resolved by a different nameserver than the one in the stale record. Also to be considered is the possibility that the provider imposes restrictions on the domains it hosts.

Algorithm 1: ZreFinder to identify PVDs

```
Input  : D // List of domains in our dataset
Output : PVDs // List of potential vulnerable domains
for dn ∈ D do
    TLDns = ∅
    SLDns = ∅
    SLDns_only = ∅
    PVDs = ∅
    TLDns = Query NS records at the TLD level
    for ns ∈ TLDns do
        // Query NS records at the domain's level
        answer = query ns for NS records
        SLDns.add(answer)
    end for
    SLDns_only = SLDns − TLDns
    for ns ∈ SLDns_only do
        response = query ns for A record
        if response_status == (REFUSED) then
            PVDs.add(dn, ns)
        end if
    end for
end for
```

Apparently, the simplest way to validate a vulnerable domain, regardless of these concerns, is just to claim it at a given DNS hosting provider. This approach, however, could not ensure that no damage will be inflicted on the domain since a successful claim could result in an automatic assignment of an IP to the domain by some providers. Also, there is a slim chance the stale `NS` record could be used by a resolver, and if so, the outcome could be cached because the provider nameserver no longer responds with `REFUSED`. Therefore, our methodology avoids such a direct claim and instead runs a series of tests, including an *unauthorized claim analysis*, to determine whether any ownership check is performed on the provider side, and three *domain exploitability assessments* (*DEAs*) to understand whether a PVD can indeed be hijacked on an unprotected provider.

Unauthorized claim analysis. This analysis is used to find out whether a domain can be claimed on a DNS hosting provider by an unauthorized party. Specifically, we first register a domain and then open two accounts with a provider: *victim* and *attacker*. Through the victim account, an `A` record is configured to point to $\texttt{IP}_{\texttt{victim}}$ and `NS` records pointing to the provider is set at the SLD zone. After confirming that the domain is indeed active, we remove the domain from the victim account. Then, we confirm that the assigned nameservers respond with `REFUSED` status code when queried about

the domain to verify that the domain is no longer active at the provider. Next, we attempt to claim the domain and set a different A record pointing to $IP_{attacker}$ through the attack's account with the same nameserver assigned to the victim (so the stale NS record can be used to hijack the domain). In the case that the provider randomly assigns its nameservers, we continue to try until the same nameserver shows up to serve the claim. This process can be automated when the DNS hosting provider offers APIs (*e.g., Amazon Route 53*) for adding a zone and setting the RRs.

DEA 1: resolvable by another nameserver at provider. In our research, we observed cases in which a domain has a Zref in addition to an active NS record both belong to the same provider. This indicates that the domain is active under another account at the provider. When this happens, an attempt to claim the domain with the provider could fail. However, inspecting all PVDs' TLD and SLD zones is not a common approach to find out whether they are in this category as the active NS records may not show up in the zones. Our solution is to utilize PDNS data to identify all the nameservers associated with a DNS provider and then query all of them with each PVD whose stale record points to the provider. If any of them does not respond with REFUSED, we suspect that the domain may not be exploitable. To confirm this, we further utilize the two accounts on the provider to claim our own domain: if the attempt fails, the PVD is considered not vulnerable.

DEA 2: registration data at provider. Also possible is the situation that the DNS hosting service is actually operated by a registrar: for example, *GoDaddy* offers both DNS hosting and domain registration services. In this case, the provider is in a position to check the ownership although it may not do that. To evaluate the exploitability of a PVD, our approach checks its WHOIS information to find out whether its registrar is also the provider that the Zref points to. Then, for all the registrars/providers discovered in this manner, we further register with each of them a domain under our control through one account and try to claim it from a different account through its DNS hosting service to determine whether the domain can be captured by one without proper ownership.

DEA 3: TLD restriction at provider. Another observation of our research is that some providers may stop serving domains with specific TLDs (mainly country code TLDs operated by some countries). As a result, the PVDs with these TLDs can no longer be exploited through the providers. To identify the presence of such restrictions, our approach utilizes two strategies. For each DNS provider, we first find out all their PVDs' TLDs. Then for each TLD, we try to claim a non-existing domain with the TLD at the provider. If the provider fails to approve the claim, a justification will be returned about the restriction. Although this strategy works for most providers, some (actually, only *GoDaddy* found in our research) do not allow claiming an unregistered domain. In this case, we resort to another strategy, utilizing PDNS to identify all records with domains under the TLD in question pointing to the provider's nameserver. The domains returned in PDNS are then used to send real-time queries to its corresponding nameserver at the provider: if all of them fail to resolve, we hypothesize that a restriction has been placed on the TLD by the provider.

Findings. In our research, we evaluated all 1,304 PVDs that are associated with the selected providers using the four tests and obtained the results presented in Table 1. Among the 17 providers studied, 14 allow unauthorized domain claims resulting in 628 PVDs. *Oracle Dyn* no longer offers DNS hosting service, so all associated PVDs cannot be claimed and thus not included in our experiments. The two remaining providers are *SEO web hosting* and *CloudFlare*. From our analysis, we found that *SEO web hosting* refuses claims for a domain removed from their service stating that the domain already exists in their servers. This prevented reclaiming our domain from the attackers account after we deleted it from victim account. However, it is not clear if this was a result of a security check or a failure to properly clean up the removed domains from their system. We also found that *CloudFlare*, in particular, has a strong verification mechanism to prevent unauthorized claims of a domain. Specifically, to prevent the abuse of stale records, when a client requests to add a domain to its service, it will first check the domain's current records through DNS queries: if the SLDns of the domain already contains any nameserver pointing to *CloudFlare* it will assign a different set of nameservers to it, thus requiring the client to update the domain's current records in order to activate the domain at this service.

In the experiment for DEA 1, we analyzed all PVDs associated with 11 affected providers. PVDs with Zref pointing to *Amazon Route 53* were excluded from this test because *Amazon Route 53* allows a domain to be active under more than one account with different NS records (Section 2.2). Thus, all its 75 associated PVDs can be exploited. For the PVDs associated with the 11 providers, 155 of them have at least one nameserver that did not return REFUSED. Therefore, they were considered not exploitable.

When it comes to DEA 2, from the registrar information of the PVDs, 3 providers were found offering domain registration service, *GoDaddy*, *Hetzner Online GmbH* and *RU Center*. Among them, only *GoDaddy* has protection in place to prevent one from claiming the domain not registered through his account. As a result, 38 PVDs that have a Zref pointing to *GoDaddy* turned out to be not exploitable, since they are all registered through *GoDaddy*. By running DEA 3 on all the providers, we observed that Domain.com stopped supporting the .ir domains, which leads to dropping 104 PVDs from our list.

4.3 DNS Resolver Analysis

For a PVD associated with an unprotected provider, a zombie resolution path (*i.e.,* claim of the domain with an A record pointing to an attack server) can be constructed on the DNS provider side (Section 4.2). However, a Zaw attack can only succeed once this path is "awakened," being utilized to answer queries on the domains at resolvers. In our research, we further investigated whether public recursive resolvers can indeed be manipulated to enable the attack. For this purpose, we analyzed three sets of recursive resolvers: 46 popular public resolvers offered by 12 well-known DNS service operators [78], around 11K open resolvers on a public list [28], and a resolver of an organization serving its members through DHCP. In addition, we performed experiments on six DNS implementations (*e.g.,* Bind [1], Unbound [50] and Microsoft DNS [56]). *Compared*

DNS Hosting Provider	Affected?	DEA1?	DEA2?	DEA3?	# hijackable domains / # PVDs
CloudFlare [18]	No	-	-	-	0/193
Amazon Route 53 [6]	Yes	○	○	○	75/75
GoDaddy DNS hosting [36]	Yes	●	●	○	19/200
Oracle Dyn [27]	NA	-	-	-	0/38
Domain.com [29]	Yes	○	○	●	82/185
Contabo [19]	Yes	○	○	○	91/91
Hetzner Online GmbH [38]	Yes	○	○	○	244/244
CentOS Web Panel [60]	Yes	○	○	○	65/65
RU Center [67]	Yes	●	○	○	15/24
DNS Made Easy [30]	Yes	●	○	○	14/15
DigitalOcean [26]	Yes	○	○	○	14/14
NS1 [59]	Yes	●	○	○	4/6
SEO Web Hosting [39]	No	-	-	-	0/149
Hurricane Electric Hosted DNS [31]	Yes	○	○	○	3/3
ClouDNS [17]	Yes	○	○	○	2/2
GeoScaling [35]	Yes	-	-	-	-*
1984 Hosting [34]	Yes	-	-	-	-*
Total					628/1304

*No PVDs found in our dataset.

Table 1: The study of DNS hosting providers where the adversary can make unauthorized claims on another party's domain. (●: PVDs dropped; ○: no PVDs dropped)

with prior research [42, 46], our study involves more resolvers, including those never analyzed before, such as the CloudFlare public resolver.

To perform domain hijacking in an ethical way (Section 4.4), we utilized a domain under our control. Specifically, we registered a domain, configured its RRs to include a Zref (an exploitable stale NS record in SLDns) and further built up a zombie resolution path with a DNS hosting provider (*i.e.,* DigitalOcean), as stated in Section 3. Then, we sent a series of DNS queries on the domain to each resolver to verify whether it can be utilized to execute the Zaw attack. We designed two experiments: one against public resolvers, and the other against popular DNS implementations. In the following, we elaborate on the study.

Public resolvers – experiment setting. In our study, we configured our own domain so that the TLDns were *not consistent* with its SLDns. Specifically, the domain has two NS records in its TLD zone and four NS records in its SLD zone. Among four records in SLDns, two of them are identical to the NS records in the TLDns and two others considered to be under the attacker's control. This configuration setting is chosen to obtain a more realistic measurement because, based on our observation in Section 5.2 there is an overlap in the NS records between TLDns and SLDns which affects the probability of selecting the Zref. Furthermore, TLDns were configured to respond with the A record carrying the IP set by the domain owner ($IP_{correct}$), while the two nameservers in SLDns were set to respond with the attacker's IP ($IP_{attacker}$). We also confirmed that our DNS client did not cache any DNS response.

We chose a short TTL value (*i.e.,* 30 seconds) for the A record carrying $IP_{correct}$ to make it quickly expire in the resolver's cache so we could force the resolver to launch a recursive resolution process for follow-up requests[4]. Also, we set the TTL for the A record of $IP_{attacker}$ as 4 hours to ensure that it would stay in the resolver once it is cached and, therefore, increases the chance to find a cache-hit at one of the multiple instances of the resolver. In particular, it applies to resolvers that implement *anycast* addressing where the endpoint address has multiple routing paths to two or more destinations for load balancing [2].

Public resolvers – experiment. During the experiment, we issued unique queries for each resolver in our dataset to avoid the answer being cached by any intermediate server, as in [21]. In particular, each subdomain (*i.e.,* `resolverID.ourDomain.tld`) is used to query its corresponding resolver.

Using such queries, we performed a 3-step domain hijacking attack for each resolver in our dataset. First, we queried the A record for our generated subdomain to cause the resolver to cache the record with $IP_{correct}$. The purpose here is to mimic the real world scenario in which resolvers cache DNS records for popular domains. Second, we queried the NS record of our domain to evaluate whether the resolver could overwrite cached TLDns with the SLDns (*i.e.,* the four NS in SLDns with two records controlled by the attacker). Third, we repeated the first step until we received $IP_{attacker}$ as a response. In our experiment, this has been done up to 50 times. For each unsuccessful attempt, we queried the NS records again (the second step) to improve the chances of success in the next attempt to receive $IP_{attacker}$. This is because popular public DNS resolvers use *anycast* for load balancing [51], and by repeating this second step, we make it more likely to have the SLDns (including the Zrefs) cached by as many affiliated resolvers as possible. The limit on the number of attempts was chosen for ethical reasons to avoid overloading the resolvers.

To confirm that the resolver cached the attacker's A record, we further queried each popular resolver for the A records of the subdomains. This test was conducted within one hour after the third step. We found that all vulnerable resolvers had the record cached. In our research, we repeated this experiment five times on each resolver to calculate the average number of attempts required to have the attacker's A record cached by vulnerable resolvers.

Public resolvers – findings. In our experiment, we performed the above 3-steps probing all resolvers.

The result with 12 top public resolver operators [78] is presented in Table 3 (in Appendix B): 10 out of them (83.33%) have at least one affected resolver. The operators include CloudFlare public DNS, OpenDNS, and Quad9. In addition, we found that a successful attack took 1 to 28 attempts with an average of 6.5 over all affected providers and a median of 3.

We further investigated whether these high-profile resolvers validate DNSSEC-signed domains, which can mitigate the Zaw threat (see Section 6). Specifically, we queried each resolver for a domain with a broken DNSSEC configuration (*i.e.,* `www.dnssec-failed.org`). If the resolver returns an A record, we know that it does not

[4]Note that the TTL setting here just serves the purpose of understanding whether public resolvers can be manipulated and in Section 5.2 we report another study that measures the attack complexity under real-world TTLs.

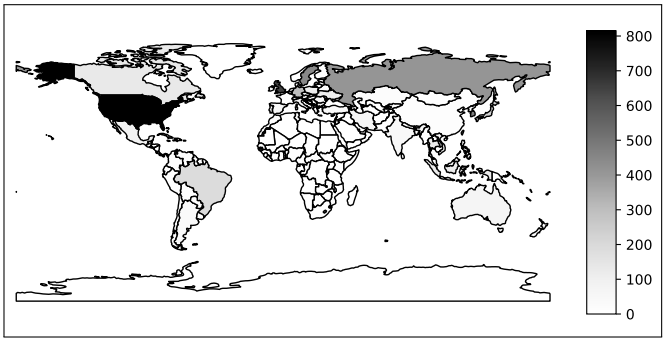


Figure 4: Geolocation distribution of the affected resolves.

support DNSSEC [70]. In our study, we observed that all top popular resolvers support DNSSEC except for *Yandex.DNS*. However, among all 628 vulnerable domains discovered, only one is properly DNSSEC-signed (Section 5.2).

Among 11,613 open resolvers [28], 11,072 responded to our queries and 7,044 are vulnerable to our cache poisoning attack (63.62% of the responding resolvers). Interestingly, only 1541 of the vulnerable resolvers (21.88%) support DNSSEC. Figure 4 shows the distribution of these resolvers' geolocations: most of them are in US, followed by Russia. We also concluded that the resolver of the organization is affected. The attacker's IP address was found to be still in the cache four hours after our experiment.

DNS implementation – experiment setting. We further looked into the behaviors of six DNS implementations (*i.e.,* Bind, Unbound, Microsoft, PowerDNS, MaraDNS, and DJB dnscache) [5] in the presence of the Zaw attack. Specifically, we investigated whether an explicit query of the NS record is required to activate the Zaw attack (step 3 in Figure 2b). The experiment setting is very similar to that for the study on the public DNS, except that we assumed no overlap between TLDns and SLDns, and all SLD nameservers under the attacker's control, for the purpose of removing nondeterminism in nameserver selection.

DNS implementations – experiment. Our experiments were performed in the following two scenarios: 1) immediately querying for the A record to find out whether the resolver will query any SLD nameserver in its default resolution path; 2) explicitly querying for an NS record before querying for the A record. Whenever $\texttt{IP}_{\texttt{attacker}}$ is returned as a response, the resolver is considered vulnerable. Note that the cache of the resolver was cleared before the experiment under each scenario.

DNS implementation – findings. As shown in Table 4 (Appendix B), none of the tested DNS implementations include SLDns in their standard resolution path under their default settings [6]. However, Bind, Unbound, PowerDNS, Microsoft DNS, and DJB dnscache are confirmed to be vulnerable to the Zaw attack when the NS record is explicitly queried for.

[5] DNSmasq was omitted since it is not a standalone recursive resolver, and its behavior depends on the selected upstream DNS server. Therefore, its susceptibility to our attack depends on whether the upstream DNS server is vulnerable. Also, Big IP was not included due to some technical challenges and limited support to our trial account.
[6] Activating "harden-referral-path" option in Unbound will cause the attack to succeed even if the NS was not queried explicitly.

4.4 Ethical Consideration

To avoid the potential negative impact of our experiments on real world online services, *i.e.,* poisoning the cache of DNS resolvers with invalid records of active domains, we directly sent our queries to TLDs nameservers and PVDs' authoritative servers. None of these queries were delivered through a caching resolver (Section 4.1). Also, we only conducted the experiments on affected resolvers on domains that we control and avoided affecting resolutions of the PVDs (Section 4.3). Moreover, we have reported our findings to the owners of vulnerable domains and will help to address the problem when needed. Also, we plan to share our findings with DNS hosting provider and resolver operators.

5 ANALYSIS AND MEASUREMENT

In this section, we discuss our findings based on the in-depth analysis of 628 hijackable domains identified in our research and measure the efforts required by an attacker to launch a successful Zaw attack.

5.1 Characteristics of Hijackable Domains

Landscape. In total, we discovered 628 hijackable domains under the threat of Zaw attack. Four vulnerable domains are under the selected sponsored TLDs, and the remaining (624) are among the Alexa's top 1M. Figure 5a (in Appendix C) shows the top-11 categories of found hijackable domains. These were categorized based on information from *Website Categorify* [15], with 195 domains uncategorized. The hijackable domains fall under a wide range of different categories. The most prevalent category is Sports with 63 domains, which mainly consists of sports betting websites (*e.g.,* `correctscore1x2.com`), followed by business websites with 34 domains (*e.g.,* `quecentre.com`), financial websites (*e.g.,* `sharmastocks.com`) with 31 domains, and shopping websites with 31 domains (*e.g.,* `brandhousedirect.com.au`). Exploiting these domains may cause an imminent financial loss if hijacked. Also, we identified 18 vulnerable domains categorized as "cloud/hosting" (*e.g.,* `avrohost.com`), which are regarded as security-critical since the security of these services could affect all the businesses hosted there. In addition, we found 6 vulnerable domains that belong to government entities: 4 Saudi Arabian, one Colombian, and one Malaysia, all under their corresponding ccTLD (i.e., `gov.ccTLD`). Also found are 31 education-related websites; for example, one university in the US (`swau.edu`), and 2 in Russia (`msun.ru`, and `usma.ru`). An attacker can exploit the high trust level of these government and education domains to orchestrate a wide range of illicit activities, such as phishing attacks to collect identity information, and malware distribution. We elaborate on the cases with high security impact in Section 5.3

The distribution of the hijackable domains among the affected providers from our methodology in Section 4.2 is shown in Figure 5b (in Appendix C). We observed that 244 of the vulnerable domains (38.85%) have Zrefs pointing to *Hetzner Online GmbH*. Given that its DNS hosting service is free, an adversary can exploit any of these domains with no cost.

TLDs of hijackable domains. Figure 5c (in Appendix C) shows the top-10 TLDs of the hijackable domains observed in our research. The most prevalent TLD is `.com` with 380 domains (*i.e.,* 60%). This

is expected due to its popularity [41]. In regard to country code TLDs (*i.e.,* ccTLDs), .gr and .ir are found to have most hijackable domains (42 and 22 domains, respectively). We argue that in some cases the existence of the Zref is due to the transition of the DNS hosting from the affected provider to a new DNS server (as investigated below in *Zref origin investigation*). We also observed in Section 4.2 that domain.com is not currently supporting .ir domains. Domain.com claims that this decision is due to Federal laws that prohibit providing service to some locations, including Iran. So, the large number of hijackable .ir domains that have a Zref pointing to other providers could be due to a burst of transition from international providers to in-house DNS servers, as an attempt to maintain the domains' availability. To support this claim, we manually sampled some .ir domains and found that their active NS records are indeed pointing to Iranian hosting services (*e.g.,* darkoobhost.com). However, the Zref still existed in the SLDns. For the .gr, we observed the transition to in-house services (*e.g.,* datapack.net); however, the justification for the burst is not clear.

Exploitable Duration. To investigate the duration of exploitability, we conducted a longitudinal study spanning over 91 days. Figure 6 (in Appendix C) shows the distribution of the duration in which hijackable domains hold Zrefs. Our results showed that 565 (89.97%) out of the 628 domains were vulnerable for at least 30 days, while 410 (65.29%) domains remained vulnerable for the whole analysis duration. The long vulnerability duration provides an adversary with a good opportunity to act maliciously and exploit it. However, only 31 (4.93%) domains were found vulnerable for fewer than 10 days. When manually sampling these domains to explore the origin, we found 8 IT related domains, all either personal blogs or small IT business. This could indicate a careful clean up by domain owners.

Zref origin investigation. We argue that part of the reason why a Zref exists is when the domain's zone information from one DNS hosting provider is imported to a new DNS server. To this end, we utilized PDNS data to examine the historical DNS records for all the hijackable domains. Specifically, if a Zref occurs at the TLDns and then later disappears, it suggests a transition has happened to a new DNS server. As a result, we indeed observed 109 (17.36%) hijackable domains showed the transition behaviors. It is important to note that PDNS data is mainly dependent on the resolution requests for the domains, thus, the number was estimated as a lower bound.

5.2 Measuring Attack Complexity

We measure the complexity of a Zaw attack in terms of: (1) the possibility for an adversary to find a time slot (i.e., a valid A record not in the cache) to poison the records of the hijackable domains, (2) the probability for an affected resolver to select a Zref, and (3) proper deployment of DNSSEC on hijackable domains. A measurement on the maximum TTL value limit for popular resolvers is included in Appendix C.

Find a cache-miss. The Zaw attack can only be launched when a vulnerable domain's valid A record is not in the cache of an affected resolver (Section 3). To estimate how often such cache-miss appears, we query affected resolvers occasionally to resolve exploitable domains. Specifically, we queried 40 affected resolvers for the A record of all 628 hijackable domains with a "+norec" option, which asks the resolver to respond only from its cache. Given the response with NOERROR status, if an A record is received in the answer section, it means that the domain is in the cache of the resolver; however, if the answer section is empty, we considered the domain not to exist in the cache (*i.e.,* cache-miss). We consistently ran this experiment for 5 days. To avoid overloading the resolver, we sent a query every 30 seconds. In total, we queried each domain for 25 times and reported the responses.

Looking into the responses from the 5 days, we observed inconsistent behaviors by resolvers. Specifically, some resolvers (e.g., *CloudFlare*) responded with NOERROR status and then responded with an error status (*e.g.,* SERVFAIL). This could be due to the utilization of anycast addressing in which some instances of the resolver support "+norec" option while others disabled it. In addition, some resolvers (e.g., *Dyn DNS*) always respond with an error status. This could indicate that these resolvers disabled the "+norec" option for privacy concerns to prevent cache snooping [47].

Table 5 (in Appendix C) shows the cache-miss rates of resolvers supporting "+norec". Here, we measured the *consistency rate*, which represents the ratio of the number of NOERROR responses to the total number of responses. A higher rate indicates consistent support for "+norec" among the instances of the resolver. Also calculated is the *cache-miss rate*, which represents the rate of the number of cache-misses over the total number of NOERROR responses. As resolvers which belong to the same operator showed similar consistency rates and cache miss rates, we only present the average of consistent rates and cache miss rates of resolvers under the same operator. Among the 10 resolver operators, we observed that 6 (*e.g.,* Dyn DNS) providers have low support for "+norec". However, some error messages may be caused by internal configurations such as filtering. The other 4 operators generally accept the "+norec" option (*e.g.,* OpenNIC). More importantly, the cache-miss rate for the exploitable domains at these resolvers is nearly 99%. It means that it is not challenging for an attacker to launch a Zaw attack against these domains.

Zref selection. A resolver selects a specific nameserver from the returned NS RRSet for any follow-up queries of the corresponding domain based on different criteria (see section 2.1). To measure the attack's success rate, we are interested in investigating the probability that an affected resolver selects a Zref. We define the probability that a resolver selects a Zref as:

$$\mathrm{P}(Zref) = \mathrm{P}(NS_{TLDns}) \,.\, \mathrm{P}(Zref_{SLDns} \mid NS_{TLDns}) \qquad (1)$$

where, $\mathrm{P}(NS_{TLDns})$ is the probability of selecting a nameserver from TLDns and $\mathrm{P}(Zref_{SLDns} \mid NS_{TLDns})$ is the probability of selecting a Zref from SLDns given that a nameserver has been selected from TLDns. For each domain, we averaged $\mathrm{P}(Zref)$ over all the snapshots. Here we assumed that the probability of selecting any nameserver from the NS RRSet are equal.

Figure 5d (in Appendix C) illustrates the distribution of the Zref selection probability among the hijackable domains. We found that the Zref selection probability is relatively high in our identified hijackable domains. Specifically, we found 391 (62.26%) of these domains have a Zref selection probability of 0.5 or higher. Even worse, 270 (42.99%) domains have a Zref selection probability of 1.

DNSSEC-signed domains. Domains with properly deployed DNSSEC are protected against this attack due to the chain of trust provided

by this protocol. DNSSEC is an extended version of DNS that utilizes public key infrastructure (PKI) to provide data integrity on DNS responses [77]. We investigated the deployment of DNSSEC on the identified hijackable domains to further assess this vulnerability. DNSSEC provides the integrity of DNS records by introducing three main record types, including DNSKEY records, RRSIG (Resource Record Signature) and DS (Delegation Signer) records. DNSKEY contains the public keys used to sign the records. For each RRSet, there exists a corresponding RRSIG record that contains a digital signature for it. Also, a DS record is placed at the parent zone and holds a hash of the child's signing key. A properly signed domain must include and configure these records correctly.

To check the level of the DNSSEC deployment for the hijackable domains, we first automatically queried each domain for DNSSEC records that facilitate the validation process, *i.e.,* DNSKEY, RRSIG, and DS. Domains that do not have these three records configured are not considered signed properly [16]. For domains that set all these records, we then utilized a DNS checking tool (*i.e.,* Zonemaster [81]) to investigate if the signatures are expired or broken.

Our study reveals that a large number of the hijackable domains have none of the three record types configured. Specifically, 619 domains (98.56%) did not configure any DNSKEY, RRSIG, or DS records. In contrast, we only found 8 (1.27%) domains that have at least DNSKEY record configured, which suggests there was an attempt of deploying DNSSEC for these domains. In addition, only one domain (moodysanalytics.com) was found to deploy DNSSEC properly.

5.3 Case studies

Here we introduce several typical domains with Zrefs, which were used in critical services. In particular, we found 2 domains (twigaw allet.com for TWIGA wallet, onfastspring.com for FastSpring) used as payment facilitation services, and one domain that is operated by a US airport (flypittsburgh.com for Pittsburgh International Airport). The Zref selection probabilities for these domains are (0.33, 0.5, 0.2), respectively (Section 5.2). For FastSpring, an SaaS e-commerce platform, we utilized PublicWWW [64], a source code search engine, to estimate the number of domains that used this service to process the online payment, and found 768 websites connecting to this domain. TWIGA wallet facilitates payment exchange and online purchases. Hijacking such a service is critical as an adversary will be able to steal payment information.

It is important to note that none of the security-critical domains discussed in this case study are DNSSEC-signed. The lack of the utilization of DNSSEC poses an increased risk of domain hijacking. Furthermore, all these domains were found exploitable for a relatively long time, at least during whole duration of our analysis, *i.e.,* 91 days.

6 DISCUSSION

Recommendation for DNS hosting providers. Our study uncovered shortcomings within the domain ownership verification procedures implemented by most of the DNS hosting providers (Section 4.2). To mitigate this issue, for DNS hosting services that use a fixed set of nameservers (*e.g., DigitalOcean*), the provider can require a customer to add a randomly generated NS record at the TLD level for the domain ownership verification. Specifically, such a random NS record could be of the format of random.provider.TLD. The random label could be generated based on the costumer's identity information such as his/her account email address and IP. When validating the domain ownership, the provider could require the customer to add this random record at the registrar. The provider should only activate a domain once it observes this random record in the domain's resolution path. Note that this random nameserver does not need to be a stand-alone nameserver; it could have its IP pointing to one of the actual nameservers used by the provider. For DNS hosting services who randomly assign their nameservers (*e.g., GoDaddy*), we suggest checking the existence of stale NS records at SLDns by enumerating all possible resolution paths for each domain of their customers. When such record is found, a completely different set of NS records should be assigned.

In addition, we argue that one cause for Zaw attacks is that some DNS hosting providers will copy all the current DNS records for the added domain, including the NS record (Section 5.1). This design choice is adopted to make the new service transfer of the domain transparent and to limit the website's downtime. However, this decision has its drawbacks, especially when these NS records are associated with another service. Although the provider allows the customer to edit these records, the customer is not aware of the risks posed by keeping these records, especially when they become stale. Thus, we suggest that the DNS service providers alert their customers to delete these records to mitigate this risk. Also, DNS hosting providers can gradually decrease the TTL values of these NS records and then eventually delete them proactively from the domain's zone. This will allow sufficient time for these records to be cleared from the caching systems that have cached them in the past and thus will not negatively affect the domain's availability.

Protection against DNS cache poisoning. Our study showcases a practical attack scenario of cache poisoning exploiting stale NS records at SLD zone, which circumvents the protection of current caching rules. We suggest that the caching rules should be improved to mitigate Zaw attack. Specifically, as suggested by Jiang *et al.*, the bailiwick rule could be updated to cache authoritative data that resides at the TLD level [42]. Also, the credibility rule could assign a higher trust level to data from the TLD zone than the data from the SLD zone. As an example, MaraDNS [55], a DNS implementation, has already adopted the suggested bailiwick rule [42]. The importance of deploying these suggestions becomes more apparent, especially with the popularity of DNS hosting service which allow any customers to create a zone for a domain of deactivated accounts even if s/he do not own the domain. Note that a recent RFC draft [40] is proposed to improve the DNS standards in RFC 1034 and RFC 1035 [57, 58]. This draft suggests explicitly validating the NS sets with the child (the SLD zone), as already implemented with 'harden-referral-path' in unbound. As a result, the attack we propose becomes straight-forward to execute, as an explicit NS query via the affected resolver is *no longer necessary*. Hence, we suggest that operators of major DNS resolvers do *not* implement this, until DNS hosting providers have widely addressed the issue we uncovered in this paper.

In addition, properly DNSSEC-signed domains are protected from cache poisoning attacks if the users accessing theses domains are using DNSSEC validating resolvers [7]. In particular, an attacker cannot properly sign new records for the hijacked domains. That is because she does not have the private signing key, which will

be used to generate correct RRSIG for the new records. Even if the attacker tries to generate a new signing key, it will not match the key stored as a DS record that is placed at the TLD. As a result, the chain of trust will be broken. However, recent studies [16, 79] showed that the current deployment of DNSSEC by domain owners or resolvers is far from perfect. More importantly, our results showed that only one of the vulnerable domains is properly signed. (Section 5.2). Hence, we advocate the faster deployment of DNSSEC.

Limitations. In the experiment of affected resolvers (Section 4.3), we made the minimum effort to have the attacker's IP address cached at the resolvers. It is worth noting that this does not guarantee that all the traffic of an affected domain will be diverted to the attacker's IP address. Particularly, a client's local cache at the browser or the operating system may still point to correct IP address. In addition, due to the use of *anycast* addressing by DNS resolvers, for a large-scale attack, the adversary needs to successfully poison as many affiliated resolver servers as possible for a more effective attack. However, we believe that these obstacles could be bypassed easily. In particular, the local cache will clear out eventually and then the client will query the recursive resolver. In addition, the attacker can run his/her attack from different geographical location to infect a broader range of resolvers such as injecting his/her attack code in JavaScript advertisement. However, to carry out a stealthy version of this attack, the adversary could just target limited instances of the resolvers' servers.

Also, the measurement on the cache-miss rate (Section 5.2) was conducted from one server. Ideally, it could be executed from different geolocations to observe a wider range of caching samples. This measurement serves as initial estimate and a large-scale version of it is considered for future work.

7 RELATED WORK

DNS security. A body of work have covered different aspects on DNS resolution process and the behavior of public resolvers. Yu *et al.*, [80] studied the aspects that influence resolvers to select a specific NS record form a set of authority servers for further queries. Other research investigated manipulation resolution process due to malware, censorship, or monetizing incentives [21, 49, 51, 62].

Meanwhile, different models were proposed to poison and protect the cache. For example, Son *et al.* [72] presented an analysis of different types of cache poisoning. The Bailiwick rule and Credibility rule are introduced [76] to protect resolvers from a variety range of attacks against the cache [8]. However, Kaminsky [44] introduced a type of attack that exploits the credibility rule to overwrite cached data with the response from referral section. Different defense mechanisms were introduced in [20, 63]. Also, Alharbi *et al.* [5] designed an attack to poison local cache of different operating system. In addition, Jiang *et al.* [42] showed that by abusing the Bailiwick rule and the Credibility rule, some revoked domains may still be resolvable by extending their TTL value. Similarly, klein *et al.* presented different attack scenarios to overwrite the cache [46]. Unlike prior efforts, in our research, we showed that abusing weak verification processes at DNS hosting providers makes these attack scenarios practical.

Domain Hijacking. A wealth of research has been conducted on domain hijacking [10, 14, 52, 75]. For example, Liu *et al.* showed the possibility of hijacking a domain through its stale records that are still pointing to publicly available resources. Specifically, they surveyed stale NS records that point to only expired domains. In addition, they investigated CNAME, MX records pointing to publicly available resources, such as cloud providers, expired domains, and measured A records using deactivated cloud providers' accounts [52]. Vissers *et al.* presented different domain hijacking scenarios, such as nameserver dependency, outdated Whois records, typosquatting and bitsquatting [75]. Bryant showed that he was able to take control over all .io TLD domains through the TLD's stale NS records [14]. Borgolte *et al.* analyzed a use-after-free vulnerability that causes a domain takeover by abusing stale A records pointing to cloud providers: an adversary can claim the corresponding IP address of the stale record at the provider and then issue an SSL certificate through automated certificate authority (*e.g.,* Let's Encrypt). They also proposed a new approach to validate domains in the automated certificate management environments of CAs [10].

Other research presented detection methodologies to detect hijacked domains, such as [11, 53]. In contrast to previous work, we investigated stale NS record at the domain's level. Unlike stale NS records that exist at the TLD level, stale NS records at the domain's level are stealthier and more difficult to be noticed by the domain owner. In addition, in our research, we showcased the attack scenario, where by exploiting the weak validation procedures of DNS hosting providers, attackers can hijack vulnerable domains that have stale records residing at their domain zones.

DNS Misconfigurations. Numerous studies have looked into various kinds of DNS misconfiguration [9, 23, 24, 37, 43, 48, 61, 65, 71], including the NS records inconsistency between the TLDns and the SLDns. Although it's known that this type of misconfiguration affects the availability of a domain, in our research, we showed that inactive NS records in SLDns can be easily acquired and activated by an attacker, thus, leading to domain hijacking.

8 CONCLUSION

This paper comprises the first large-scale study on the menaces of stale NS records in the SLD zone. We have highlighted that these records can be easily exploited, causing a stealthy hijacking of active domains associated with DNS hosting services. By scanning over 1M high-profile domains, we identified 628 hijackable domains, affecting government agencies, public services, and large corporations. Our research further shows prominent DNS hosting services (*e.g.,* Amazon Route 53) and popular public resolvers (*e.g.,* CloudFlare) are all vulnerable to the attack. Moving forward, we investigated a set of mitigation strategies to help the affected parties defend against this new security risk.

ACKNOWLEDGMENTS

We thank our shepherd Tobias Fiebig and the anonymous reviewers for their insightful comments. This work was supported in part by the National Science Foundation under CNS-1838083, 1801432, 1618493, 1850725 and 1937143. Any opinions, findings, conclusions or recommendations expressed in this paper do not necessarily reflect the views of the NSF.

REFERENCES

[1] Bind 9. 2020. Internet Systesm Consortium. https://www.isc.org/bind/.

[2] Joe Abley and Kurt Erik Lindqvist. 2006. *Specification for DNS over Transport Layer Security (TLS).* RFC 4786. RFC Editor. https://tools.ietf.org/html/rfc4786

[3] Salem Alelyani and Harish Kumar. 2020. Overview of Cyberattack on Saudi Organizations. *Journal of Information Security and Cybercrimes Research* 2 (2020).

[4] Alexa. 2018. Alexa - Top sites. https://www.alexa.com/topsites.

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

[6] Amazon. 2017. Amazon Route 53. https://aws.amazon.com/route53/.

[7] Derek Atkins and Rob Austein. 2004. *Threat Analysis of the Domain Name System (DNS).* RFC 3833. RFC Editor. https://tools.ietf.org/html/rfc3833

[8] Steven M Bellovin. 1995. Using the Domain Name System for System Break-ins.. In *proceedings of the USENIX Security Symposium.*

[9] Petar D Bojović and Slavko Gajin. 2017. An approach to evaluation of common DNS misconfigurations. *arXiv preprint arXiv:1711.05696.*

[10] Kevin Borgolte, Tobias Fiebig, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. 2018. Cloud strife: mitigating the security risks of domain-validated certificates. In *proceedings of the Network and Distributed System Security Symposium.*

[11] Andreas Borgwart, Spyros Boukoros, Haya Shulman, Carel van Rooyen, and Michael Waidner. 2015. Detection and forensics of domains hijacking. In *proceedings of IEEE Global Communications Conference.* 1–6.

[12] Matthew Bryant. 2016. Floating Domains – Taking Over 20K DigitalOcean Domains via a Lax Domain Import System. https://thehackerblog.com/floating-domains-taking-over-20k-digitalocean-domains-via-a-lax-domain-import-system/.

[13] Matthew Bryant. 2016. The Orphaned Internet – Taking Over 120K Domains via a DNS Vulnerability in AWS, Google Cloud, Rackspace and Digital Ocean. https://thehackerblog.com/the-orphaned-internet-taking-over-120k-domains-via-a-dns-vulnerability-in-aws-google-cloud-rackspace-and-digital-ocean/.

[14] Matthew Bryant. 2017. The .io Error – Taking Control of All .io Domains With a Targeted Registration. https://thehackerblog.com/the-io-error-taking-control-of-all-io-domains-with-a-targeted-registration/.

[15] Categorify. 2020. Website Categorify. https://categorify.org/.

[16] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. 2017. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. In *proceedings of the USENIX Security Symposium.* 1307–1322.

[17] CloudDNS. 2020. Free DNS hosting, Premium DNS hosting and Domain names | CloudDNS. https://www.cloudns.net/main/.

[18] Cloudflare. 2019. CloudFlare - Managed DNS. https://www.cloudflare.com/dns/.

[19] Contabo. 2020. Hosting solutions: Webspace, VPS, Dedicated Server. https://contabo.com/.

[20] David Dagon, Manos Antonakakis, Kevin Day, Xiapu Luo, Christopher P Lee, and Wenke Lee. 2009. Recursive DNS Architectures and Vulnerability Implications.. In *proceedings of the Network and Distributed System Security Symposium.*

[21] David Dagon, Chris Lee, Wenke Lee, and Niels Provos. 2008. Corrupted DNS resolution paths: The rise of a malicious resolution authority. In *proceedings of the Network and Distributed System Security Symposium.*

[22] CERT Vulnerability Notes Database. 2014. Multiple DNS implementations vulnerable to cache poisoning. http://www.kb.cert.org/vuls/id/800113.

[23] Casey Deccio, Chao-Chih Chen, Prasant Mohapatra, Jeff Sedayao, and Krishna Kant. 2009. Quality of name resolution in the domain name system. In *proceedings of the IEEE International Conference on Network Protocols.* 113–122.

[24] Casey Deccio, Jeff Sedayao, Krishna Kant, and Prasant Mohapatra. 2010. Measuring availability in the domain name system. In *proceedings of the IEEE INFOCOM Conference on Computer Communications.* 1–5.

[25] John Dickinson, Sara Dickinson, Ted Lemon, and Tom Pusateri. 2019. *DNS Stateful Operations.* RFC 8490. RFC Editor. https://tools.ietf.org/html/rfc8490

[26] DigitalOcean. 2019. DigitalOcean – The developer cloud. https://www.digitalocean.com/.

[27] Dyn DNS. 2020. Managed DNS Hosting | Oracle Dyn. https://dyn.com/managed-dns-hosting/.

[28] Public DNS. 2017. Public DNS Server List. https://public-dns.info/.

[29] Domain.com. 2020. Website Domains Names & Hosting | Domain.com. https://www.domain.com/.

[30] DNS Made Easy. 2019. DNS Made Easy | Fastest and Most Reliable Enterprise DNS Provider. https://dnsmadeeasy.com/.

[31] Hurrican Electric. 2018. Hurricane Electric Hosted DNS. https://dns.he.net/.

[32] Robert Elz and Randy Bush. 1997. *Clarifications to the DNS specification.* RFC 2181. RFC Editor. https://tools.ietf.org/html/rfc2181

[33] Farsight. 2020. Farsight Security. https://www.farsightsecurity.com/.

[34] FreeDNS. 2019. FreeDNS - 1984 Hosting. https://www.1984hosting.com/product/freedns/.

[35] GeoScaling. 2015. GeoScaling DNS2 - Free Managed DNS, redirect by Country, City, Network Name. http://www.geoscaling.com/.

[36] GoDaddy. 2019. Domain Manager - GoDaddy. https://dcc.godaddy.com/domains/dnsHosting/add.

[37] Cristian Hesselman, Giovane CM Moura, Ricardo de Oliveira Schmidt, and Cees Toet. 2017. Increasing DNS security and stability through a control plane for top-level domain operators. *IEEE Communications Magazine* 55, 1 (2017), 197–203.

[38] Hetzner. 2020. Dedicated Root Server, VPS &; Hosting - Hetzner Online GmbH. https://www.hetzner.com/.

[39] SEO Hosting. 2015. Multiple DNS Hosting - SEO Web Hosting. http://7cloudcomputing.com/.

[40] Shumon Huque, Paul Vixie, and Ralph Dolmans. 2020. *Delegation Revalidation by DNS Resolvers.* draft-huque-dnsop-ns-revalidation-01. https://tools.ietf.org/html/draft-huque-dnsop-ns-revalidation-01

[41] ICDSoft. 2020. What are the Most Popular TLDs/Domain Extensions in 2020? https://www.icdsoft.com/blog/what-are-the-most-popular-tlds-domain-extensions/.

[42] Jian Jiang, Jinjin Liang, Kang Li, Jun Li, Haixin Duan, and Jianping Wu. 2012. Ghost domain names: Revoked yet still resolvable. (2012).

[43] Jian Jiang, Jia Zhang, Haixin Duan, Kang Li, and Wu Liu. 2018. Analysis and measurement of zone dependency in the domain name system. In *proceedings of IEEE International Conference on Communications.* 1–7.

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

[45] Amit Klein. 2007. BIND 9 DNS cache poisoning. *Report, Trusteer, Ltd.*

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

[47] ISC Knowledge. 2018. What is DNS Cache snooping? https://kb.isc.org/docs/aa-00509/.

[48] John Kristoff. 2018. DNS inconsistency. https://blog.apnic.net/2018/08/29/dns-inconsistency/.

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

[50] NLnet Labs. 2020. Unbound. https://nlnetlabs.nl/projects/unbound/about/.

[51] Baojun Liu, Chaoyi Lu, Haixin Duan, Ying Liu, Zhou Li, Shuang Hao, and Min Yang. 2018. Who is answering my queries: Understanding and characterizing interception of the DNS resolution path. In *proceedings of the USENIX Security Symposium.* 1113–1128.

[52] Daiping Liu, Shuai Hao, and Haining Wang. 2016. All your DNS records point to us: Understanding the security threats of dangling DNS records. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security.* 1414–1425.

[53] Daiping Liu, Zhou Li, Kun Du, Haining Wang, Baojun Liu, and Haixin Duan. 2017. Don't Let One Rotten Apple Spoil the Whole Barrel: Towards Automated Detection of Shadowed Domains. In *proceedings of the ACM SIGSAC Conference on Computer and Communications Security.* 537–552.

[54] Sean Lyngaas. 2020. Saudi cyber authority uncovers new data-wiping malware, and experts suspect Iran is behind it. https://www.cyberscoop.com/saudi-arabia-iran-cyberattack-soleimani/.

[55] MaraDNS. 2020. MaraDNS - a small open-source DNS server. https://maradns.samiam.org.

[56] Microsoft. 2019. Microsoft DNS. https://docs.microsoft.com/en-us/windows-server/networking/dns/dns-top.

[57] Paul Mockapetris. 1987. *Domain Names - Concepts And Facilities.* RFC 1034. RFC Editor. https://tools.ietf.org/html/rfc1034

[58] Paul Mockapetris. 1987. *Domain Names - Implementation And Specification.* RFC 1035. RFC Editor. https://tools.ietf.org/html/rfc1035

[59] NS1. 2018. Managed & Private DNS That's Smart, Efficient & Fast. https://ns1.com/.

[60] CentOS Web Panel. 2018. CentOS Web Panel | Free Linux Web Hosting Control Panel. http://centos-webpanel.com/.

[61] Vasileios Pappas, Zhiguo Xu, Songwu Lu, Daniel Massey, Andreas Terzis, and Lixia Zhang. 2004. Impact of configuration errors on DNS robustness. In *proceedings of the ACM SIGCOMM conference on Applications, technologies, architectures, and protocols for computer communications.* 319–330.

[62] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. 2017. Global Measurement of DNS Manipulation. In *proceedings of the USENIX Security Symposium.* 307–323.

[63] Roberto Perdisci, Manos Antonakakis, Xiapu Luo, and Wenke Lee. 2009. WSEC DNS: Protecting recursive DNS resolvers from poisoning attacks. In *proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks.* 3–12.

[64] PublicWWW. 2019. Source Code Search Engine - PublicWWW . https://publicwww.com/.

[65] Venugopalan Ramasubramanian and Emin Gün Sirer. 2005. Perils of transitive trust in the domain name system. In *proceedings of the ACM SIGCOMM Internet Measurement Conference.* 35–35.

[66] Marwa Rashad. 2020. Saudi Aramco sees increase in attempted cyber attacks. https://www.reuters.com/article/us-saudi-aramco-security/saudi-aramco-sees-increase-in-attempted-cyber-attacks-idUSKBN2002N2.

[67] RU-Center. 2018. DNS Hosting - RU-CENTER. https://www.nic.ru/en/catalog/for-domain-use/dns-hosting.

[68] Serverfault. 2018. What RFC encourages DNS servers to reply REFUSED to queries for unknown domains? - Server Fault. https://serverfault.com/questions/892622/what-rfc-encourages-dns-servers-to-reply-refused-to-queries-for-unknown-domains.

[69] Shashank Singh. 2019. 11 Best Free DNS Hosting Services. https://technumero.com/free-dns-hosting-services/.

[70] Internet Society. 2013. DNSSEC Test Sites. https://www.internetsociety.org/resources/deploy360/2013/dnssec-test-sites/.

[71] Raffaele Sommese, Giovane CM Moura, Mattijs Jonker, Roland van Rijswijk-Deij, Alberto Dainotti, KC Claffy, and Anna Sperotto. 2020. When parents and children disagree: Diving into DNS delegation inconsistency. In *proceedings of the International Conference on Passive and Active Network Measurement*. 175–189.

[72] Sooel Son and Vitaly Shmatikov. 2010. The hitchhiker's guide to DNS cache poisoning. In *proceedings of the International Conference on Security and Privacy in Communication Systems*. 466–483.

[73] StatDNS. 2020. TLD Zone File Statistics. https://www.statdns.com/. Online; accessed 2 March 2020.

[74] stats.labs.apnic.net. 2018. Use of DNSSEC-ECDSA Validation for World (XA). https://stats.labs.apnic.net/ecdsa/XA. Online; accessed 2 March 2020.

[75] Thomas Vissers, Timothy Barron, Tom Van Goethem, Wouter Joosen, and Nick Nikiforakis. 2017. The wolf of name street: Hijacking domains through their nameservers. In *proceedings of the ACM SIGSAC Conference on Computer and Communications Security*.

[76] Paul Vixie. 1995. DNS and BIND Security Issues.. In *proceedings of the USENIX Security Symposium*.

[77] Samuel Weiler and David Blacka. 2013. RFC 6840 - Clarifications and Implementation Notes for DNS Security (DNSSEC). https://tools.ietf.org/html/rfc6840.

[78] Wikipedia. 2020. Public recursive name server. https://en.wikipedia.org/wiki/Public_recursive_name_server.

[79] Matan Ben Yosef, Haya Shulman, Michael Waidner, and Gal Beniamini. 2017. Factoring DNSSEC: Evaluation of Vulnerabilities in Signed Domains. In *proceedings of the Symposium on Networked System Design and Implementation*.

[80] Yingdi Yu, Duane Wessels, Matt Larson, and Lixia Zhang. 2012. Authority server selection in DNS caching resolvers. *proceedings of the ACM SIGCOMM Computer Communication Review*, 80–86.

[81] Zonemaster. 2020. Zonemaster. https://zonemaster.iis.se/.

A SUPPLEMENTARY TLDS

Domains Source	# of unique domains
.edu domains	4,808
.gov domains	6,131
.edu.cn domains	3,336
.gov.cn domains	1,580
.edu.sa domains	1,453
.gov.sa domains	1,887
Total	19,195

Table 2: Distribution of domains under selected sponsored TLDs analyzed in our study.

B EXPERIMENTAL RESULTS ON POPULAR RESOLVERS AND DNS IMPLANTATION

Operator	DNSSEC	IP address	Avg. successful attempts	Affected?
Google Public DNS	Yes	8.8.8.8	-	No
	Yes	8.8.4.4	-	No
CloudFlare Public DNS	Yes	1.1.1.1	3	Yes
	Yes	1.0.0.1	2	Yes
OpenDNS	Yes	208.67.222.222	6.6	Yes
	Yes	208.67.220.220	2.4	Yes
	Yes	208.67.222.123	2.4	Yes
	Yes	208.67.220.123	1.6	Yes
Quad9	Yes	9.9.9.9	3	Yes
	Yes	149.112.112.112	4.8	Yes
	No	9.9.9.10	3.2	Yes
	No	149.112.112.10	3.2	Yes
OpenNIC	Yes	185.121.177.177	14.4	Yes
	Yes	169.239.202.202	6.2	Yes
Dyn DNS	Yes	216.146.35.35	21	Yes
	Yes	216.146.36.36	2.6	Yes
Comodo Secure DNS	Yes	8.26.56.2	3.4	Yes
	Yes	8.20.247.20	3.4	Yes
VeriSign Public DNS	Yes	64.6.64.6	-	No
	Yes	64.6.65.6	-	No
Neustar DNS Advantage	Yes	156.154.70.1	2.4	Yes
	Yes	156.154.71.1	2.2	Yes
	Yes	156.154.70.2	2	Yes
	Yes	156.154.71.2	1.4	Yes
	Yes	156.154.70.3	3.2	Yes
	Yes	156.154.71.3	3	Yes
	Yes	156.154.70.4	2.2	Yes
	Yes	156.154.71.4	1.8	Yes
	Yes	156.154.70.5	1.8	Yes
	Yes	156.154.71.5	1.6	Yes
Clean Browsing	Yes	185.228.168.168	27.4	Yes
	Yes	185.228.169.168	-	No
	Yes	185.228.168.10	1.2	Yes
	Yes	185.228.169.11	1.2	Yes
	Yes	185.228.168.9	23.6	Yes
	Yes	185.228.169.9	1	Yes
AdGuard DNS	Yes	176.103.130.130	3.4	Yes
	Yes	176.103.130.131	3.4	Yes
	Yes	176.103.130.132	1.4	Yes
	Yes	176.103.130.134	1.4	Yes
Yandex.DNS	No	77.88.8.1	-	No
	No	77.88.8.8	18	Yes
	No	77.88.8.2	23	Yes
	No	77.88.8.88	15.8	Yes
	No	77.88.8.3	25	Yes
	No	77.88.8.7	10.4	Yes
Avg(Average # of successful attempts)			6.5	
Median(Average # of successful attempts)			3	

Table 3: The result of our experiment on popular public DNS resolvers, along with their DNSSEC support and the average number of attempts to carry a successful attack against our domain.

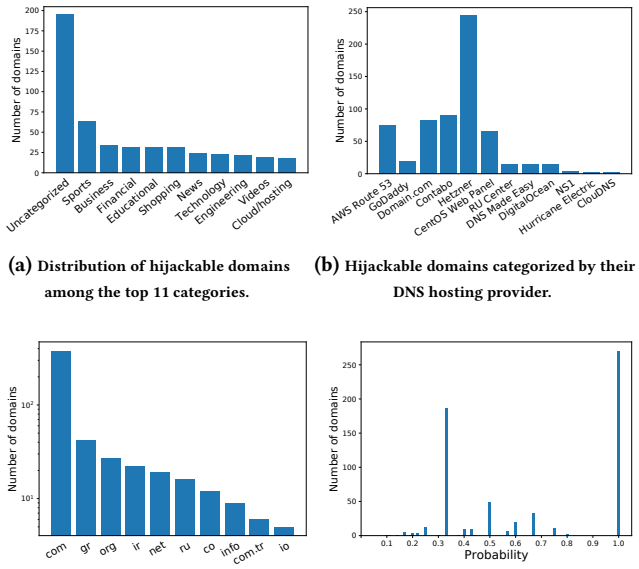
DNS Vendor	Version	Vul.? (Default)	Vul.? (NS explicitly)
BIND	1.9.4	No	Yes
Unbound	1.6.7	No *	Yes
Microsoft DNS	Windows Server 2019	No	Yes
PowerDNS	Recursor 4.2.1	No	Yes
MaraDNS	Deadwood 3.5.0011	No	No
DJB dnscache	1.05	No	Yes

*Vulnerable if "harden-referral-path" option is set

Table 4: The results of our experiment on popular DNS implementations.

C MEASUREMENTS ON HIJACKABLE DOMAINS

C.1 Hijackable Domains Characteristics



(a) Distribution of hijackable domains among the top 11 categories.

(b) Hijackable domains categorized by their DNS hosting provider.

(c) Distribution of hijackable domains based on the top 10 TLDs.

(d) Probability of Zref selection.

Figure 5: Hijackable domains analysis.

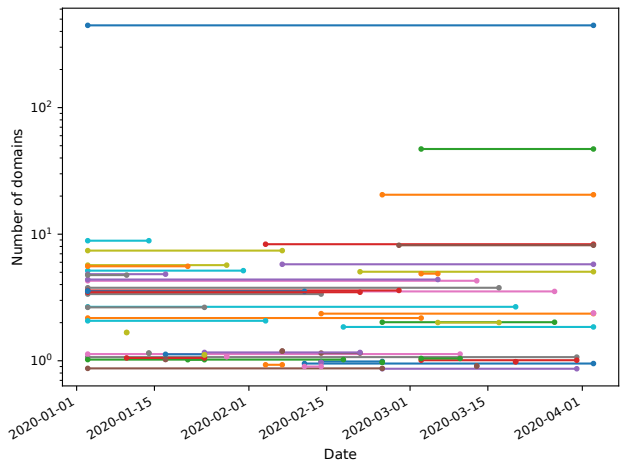


Figure 6: The duration in which hijackable domains remain vulnerable.

C.2 Details on Attack Complexity

Operator	Maximum TTL	Consistency Rate	Cache-miss Rate
Dyn DNS	604800	0	0
Comodo Secure DNS	604800	0	0
Neustar DNS Advantage	604800	0	0
CloudFlare Public DNS	604800	0.13%	0
AdGuard DNS	604800	0.79%	0
OpenDNS	604800	8.17%	0.06%
Quad9	43200	48.38%	99.93%
OpenNIC	604800	99.62%	100.00%
Clean Browsing	86400	99.73%	99.76%
Yandex DNS (77.88.8.8) Yandex DNS (77.88.8.[2,3,7,88])	86400 10800	100.00%	99.44%

Table 5: The attack complexity on the vulnerable DNS resolvers.

Maximum TTL limit. Some resolvers set a limit on the maximum TTL value they accept, which provides an estimate on how long a poisoned A record can survive in the cache of the resolvers. Hence, we measure the maximum TTLs of the affected resolvers. To this end, we registered a domain and created 40 subdomains to test 40 affected resolvers (similar to our methodology in Section 4.3). The TTL value for all subdomain's A record is set to be 604800 seconds (*i.e.,* one week), the maximum TTL limit for most resolvers [42]. Then each resolver is queried to achieve the returned TTL value.

Table 5 shows the returned TTLs. As the resolvers from the same operator share the same returned TTLs, except for *Yandex DNS*, we present the value per operator. We observe the majority of the tested resolvers accepts the set TTL value (*i.e.,* 604800 seconds), while resolvers operated by *Clean Browsing* and *Quad9* accept smaller TTL values, i.e., 86400 seconds and 43200 seconds, respectively. Note that it is possible that a poisoned record was cleaned up from the cache before its TTL expires. This is because resolvers may purge cache periodically. In this measurement study, we are interested in obtaining a high-level estimation about the lifetime of a poisoned record.