Identify DNS misconfigurations and suboptimal configurations

Raffaele Sommese[1], Anna Sperotto[1], Roland van Rijswijk-Deij[1]
Alberto Dainotti[2], Kimberly Claffy[2]

[1]) University of Twente
{r.sommese, a.sperotto, r.m.vanrijswijk}@utwente.nl
[2]) CAIDA, UCSD
{alberto, kc}@caida.org

In this first year of research, we focused on vulnerabilities and misconfigurations in the DNS system. In particular, we investigated two of them: Parent-Child Inconsistency in the DNS hierarchy and Orphan and Abandoned Records.

Parent-Child Inconsistency in the DNS hierarchy

Paper: When parents and children disagree: Diving into DNS delegation inconsistency
(Under review@PAM2020)

In the first case, we studied the consistency of the replicated information along the DNS hierarchy between parent and child zone. In particular, we analyzed the consistency of NS records. The NS records maintain information about the delegation of a domain and are contained both in the parent zone file and in the child zone. This replicated information in the different levels of the hierarchy should be consistent, however, we discovered that is not always the case. We investigated the consequences and the scope of this misconfiguration. We also provided advice for the operators and tried to notify some of them.

Orphan and Abandoned Records

Paper: The Forgotten Side of DNS: Orphan and Abandoned Records (IMC Reproducibility Track rejected)
(We are addressing the comments of the reviewers and extending for submitting@TMA2020 on Feb 10, 2020)

Domains in DNS can be registered for a certain amount of time and expire if not renewed. At the expiration of a domain, all the related records should be removed from the zone file. Nonetheless, we discover dangling information in the zone file of some TLDs operators.

In particular, we identified:

- Orphan Records: Former glue records (A record) for which the delegation records (NS records) of the related domain does not exist. (belonging to potentially expired domains).
- Abandoned Records: Former glue records (A record) for which the delegation records (NS records) of the related domain does not point anymore to it.

We studied the scope, the impact and the security risk related to these records.

When parents and children disagree: Diving into DNS delegation inconsistency

Anonymous authors

Abstract. The Domain Name System (DNS) is a hierarchical, decentralized, and distributed database. A key mechanism that enables the DNS to be hierarchical and distributed is *delegation* [7] of responsibility from parent to child *zones*—typically managed by different entities. RFC1034 [11] states that authoritative nameserver (NS) records at both parent and child should be "consistent and remain so", but we find inconsistencies for over 13M second-level domains. We classify the type of inconsistencies we observe, and the behavior of resolvers in the face of such inconsistencies, using RIPE Atlas to probe our experimental domain configured for different scenarios. Our results underline the risk such inconsistencies pose to the availability of misconfigured domains.

1 Introduction

The Domain Name System (DNS) [11] is one of the most critical components of the Internet, used by virtually every user and application. DNS is a distributed, hierarchical database that maps hosts, services and applications to IP addresses and various other types of records. A key mechanism that enables the DNS to be hierarchical and distributed is *delegation* [7]. In order for delegation to work, the DNS hierarchy is organized in parent and child *zones*—typically managed by different entities—that need to share common information (NS records) about which are the authoritative name servers for a given domain. While RFC1034 [11] states that the NS records at both parent and child should be "consistent and remain so", there is evidence that this is not always the case [9]. However, a full and systematic analysis of the extent of this problem is still missing.

In this paper, we comprehensively analyze this issue by *(i)* providing a broad characterization of inconsistencies in DNS delegations, and *(ii)* investigating and shedding light on their practical impact. Specifically, we first evaluate if there are inconsistencies between parent and child sets of NS records (NSSet) for all active second-level domain names of three large DNS zones: `.com`, `.net`, and `.org` (§3)—together comprising of more than 160M domain names, as well as all top-level domains (TLDs) from the Root DNS zone [20]. We show that while 92% of these domain names exhibit consistency, 8% (i.e., 13 million domains) do not. These inconsistencies affect even large and popular organizations, including Twitter, Intel and AT&T. Overall we find that at least 50k `.com`, `.net`, and `.org` domains of the Alexa Top 1M list are affected.

We then classify these inconsistencies into four categories (§3): the cases (i) in which the parent and child NSSets are *disjoint* sets, (ii) the parent NSSet is a

subset of the child NSSet, (iii) the parent NSSet is a *superset* of the child NSSet and (iv) the parent and child NSSet have a non-empty intersection but do not match (ii) or (iii). These inconsistencies are not without harm. Even in the case in which disjoint sets of NS records resolve to the same IP addresses, case (i) introduces fragility in the DNS infrastructure, since operators need to maintain different information at different levels of the DNS hierarchy, which are typically under separate administrative control. Case (ii) may lead to unresponsive name servers, while case (iii) points to a quite understandable error of modifying the child zone while forgetting the parent, but it offers a false sense of resilience and it results in improper load balancing among the name servers. Finally, case (iv), which we see happening in more than 10% of the cases in which parent and child have a non-empty intersection, suffers all the aforementioned risks.

To understand the practical impact of such inconsistencies, we emulate all four categories (§4) by setting up a test domain name and issuing DNS queries from more than 15k vantage points. Our experiment highlights the impact of delegation inconsistency on query load distribution in the wild. We then investigate how popular DNS resolvers from different vendors deal with such inconsistencies (§5), and find that some resolvers do not comply with RFC specifications. Finally, we conclude the paper discussing our findings and offering recommendations for domain name operators to manage the inconsistencies we identified.

2 Background and Related Work

DNS uses a *hierarchical name space* [11], in which the root node is the *dot* (`.`). Zones under the root—the top-level domains such as `.org`— are referred to as *delegations* [7]. These delegations have second-level delegations of their own such as `example.org`. To create delegations for a *child* zone (such as `example.org`), DNS NS records [11] are added to the *parent* zone (`.org` in Figure 1). In this example, the NS records are `[a,b].iana-servers.net`, which, in practice, means that these records are the *authoritative* name servers for `example.org`, *i.e.,* servers that have definitive information about the `example.org` zone.

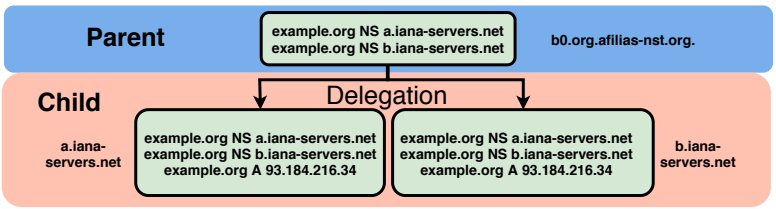


Fig. 1: Domain name delegation: parent and child authoritative servers.

RFC1034 states that the NSSet should be consistent between parent and child authoritative servers. This, however, is far from trivial. Parent and child

zones and servers are almost always maintained by different organizations across administrative boundaries. The most common case is where the parent is a TLD. Delegation changes in the parent go through the so-called Registry-Registrar-Registrant (RRR) channel for almost all TLDs. In this model, the Registry operates the TLD, the Registrar sells domain names under the TLD and the Registrant is the domain holder. If the domain holder wants to change the delegation, they can make the change in their child zone, but need to file a request with the Registry through the Registrar. This process currently always happens via an out-of-band channel (not through the DNS) and in some cases may even require forms on paper. Add to this that domain holders may not always be aware of this complexity and the requirement to keep parent and child in sync, and it is clear to see that keeping the DNS consistent is prone to human errors.

The problem of Parent-Child consistency is addressed in RFC7477 [6], which introduces a method to automatically keep records in the parent in sync through a periodical polling of the child using SOA records and a new type of record (CSYNC). Unfortunately, RFC7477 lacks deployment. Kristoff [9] analysed delegations in `.edu` and finds that 25% of `.edu` delegations suffer some form of inconsistency. In his work, he considers 3 types of inconsistency: superset, subset and disjoint-set. Our work significantly expands on this by considering both the largest generic TLDs `.com`, `.net` and `.org` and the root zone of the DNS.

Liu *et al.* show that dangling delegation records referring to expired resources (e.g., cloud IP addresses or names) left in the parent or child pose a significant risk [10]. An attacker can obtain control of these records through the same cloud services by randomly registering new services, and in this way take control of the domain. Finally, Moura *et al.* [13] have looked into the consistency of time-to-live values [11] of parent and child NS records.

3 Parent and Child NSSet: are they consistent?

DNS NS records must be configured at both parent and child zones [11,5]. We compare `NS` records at parents and children in the wild considering all second-level domains (SLDs) under `.com`, `.net`, and `.org`, on 2019-10-16. We also evaluate the records in the Root DNS zone on 2019-10-30. We make use of OpenINTEL, a large-scale DNS measurement platform [21]. For each SLD, we extract the sets of `NS` records from the parent and child authoritative servers, respectively indicated as P and C. Table 1 shows the results of our comparative analysis. The first row shows the number of responsive SLDs for each TLD zone on the date considered. For the three zones, 92% of the responsive SLDs have a consistent set of `NS` records at both the parent and the child zones. However, ∼8% of SLDs (∼ 13M) do not. For comparison, consider that 13M is almost as many domain names as the largest country-code TLDs (Germany's `.de`, the largest ccTLD, has 16M SLDs [4]). We even see that 53 TLDs in the Root zone have inconsistent NSSets. Out of these, 26 are country-code TLDs (ccTLDs). We are currently notifying these ccTLD operators, in order to resolve these non-conforming setups, since they can have an adverse effect, among others, on load balancing.

	.com SLD	.org SLD	.net SLD	Root TLD	.com Ratio	.org Ratio	.net Ratio
Responsive domains	142,302,090	9,998,488	13,181,091	1528			
$P = C$	130,937,525	9,240,394	12,106,717	1476	92.0%	92.4%	92.0%
$P \neq C$	11,364,565	758,094	1,074,374	52	8.0%	7.8%	8.0%
$P \cap C = \emptyset$	6,594,680	418,269	548,718	16	58.0%	55.2%	51.0%
$IP(P) = IP(C)$	3,046,075	216,130	245,936	16	48.2%	53.9%	46.7%
$IP(P) \neq IP(C)$	3,265,171	184,885	280,988	0	51.8%	46.1%	53.3%
$IP(P) \cap IP(C) = \emptyset$	1,415,838	83,720	137,913	0	43.3%	45.3%	49.1%
$IP(P) \cap IP(C) \neq \emptyset$	1,849,333	101,165	143,075	0	56.7%	54.7%	51.9%
$P \cap C \neq \emptyset$	4,769,885	339,825	525,656	36	42.0%	44.8%	49.0%
$P \subset C$	3,506,090	236,257	369,442	18	73.5%	69.5%	70.2%
$P \supset C$	681,082	64,161	98,345	10	14.3%	18.9%	18.7%
Rest	582,713	39,407	57,869	8	12.2%	11.6%	11.1%

Table 1: Parent (P) and Child (C) NSSet consistency results. "IP" refers to `A` records of the NSSet of P and C.

Inconsistent NSSets classification: We classify inconsistent domain names into four categories: the cases in which (i) the parent and child NSSets are *disjoint*, (ii) the parent NSSet is a *subset* of the child NSSet, (iii) the parent NSSet is a *superset* of the child NSSet and (iv) the parent and child NSSet have a non-empty intersection but do not match (ii) or (iii).

For case (i), we observe that 51–58% of domains have completely *disjoint NSSets* ($P \cap C = \emptyset$). Depending on if resolvers are parent or child-centric, in this case resolvers will trust different `NS` records. Giving the surprising results for disjoint sets, we investigate the IP addresses of the NS records (IP(P, C, lines 4-7 in Table 1).[1] We discover that in half of the cases, domains have disjoint NSSets that point to the same addresses, *i.e.,* there is an inconsistency of names but addresses match. In the other half, there is inconsistency also in addresses. Of these, $\sim$ 45% have completely disjoint sets of IP addresses, for the remaining 55% there is some sort of overlap. Disjoint sets may increase the risk of human error even in the case of name servers resolving to the same IP address, since operators would need to maintain redundant information in the parent and child, thus introducing fragility in the DNS data. Disjoint sets also may lead to lame delegations [7], *i.e.,* pointing resolvers to servers that may no longer be authoritative for the domain name. Finally disjoint sets can be related to another malpractice: *CNAME configured on the Apex* [1]. However, further analysis shows that only a negligible percentage of cases are related to this.

Considering partially matching SLDs ($P \cap C \neq \emptyset$), we observe that 69–73% belong to case (ii), where the parent NSSet is a *subset* of the child NSSet. This may be intentional, e.g. an operator may want to first update the child and observe traffic shifts, and then later update the parent. Alternatively, operators may forget to update the delegation at the parent after updating the child.

[1] This covers 96% of names with disjoint NSSets, the remaining 4% are indeterminate due to unresolvable names in the NSSets.

Case (iii) where the parent NSSet forms a *superset* of the child NSSet ($P \supset C$) occurs in 14-18% of cases. This situation may introduce latency in the resolution process due to unresponsive name servers. Finally, the *Rest* category is case (iv), where the NSSets form neither a superset nor a subset, yet they have a non-empty intersection. Between 11-12% of SLDs fall in this category, and are susceptible to the range of operational issue highlighted for the previous categories.

Note that the OpenINTEL platform performs the measurements choosing *one* of the child authoritative nameservers. To verify how often sibling name servers have different configurations (child-child delegation inconsistency), we execute a measurement on a random sample of $\sim$ 1% of `.org` domains (10k domains). The measurement suggests that $\sim$ 2% of total parent-child delegation inconsistency cases also have child-child delegation inconsistencies, meaning that our results give a lower bound for the problem of parent-child mismatch.

4 Implications of NSSet differences in the wild

We observed that roughly 8% of studied domains have parent/child inconsistencies.In this section, we investigate the consequences of such inconsistencies, by emulating the four categories of NSSet mismatches.We configure parent and child authoritative servers in eight different configurations (Table 2), and explore the impact in terms of query load distribution.

We emulate an operator that (i) has full control over its child authoritative name servers and (ii) uses the same zone file on all authoritative name servers (zones are synchronized). We place all child authoritative servers in the same network, thus, having similar latencies. We expect this to result in querying resolvers distributing queries evenly among child authoritatives [14].

As vantage points, we use RIPE Atlas [18,19], measuring each unique resolver as seen from their probes physically distributed around the world (3.3k ASes). Many Atlas probes have multiple recursive resolvers, so we treat each combination of probe and unique recursive resolver as a vantage point (VP), since potentially each represents a different perspective. We therefore see about 15k VPs from about 9k Atlas probes, with the exact number varying by experiment due to small changes in probe and resolver availability.

4.1 Disjoint Parent and Child NSSet

We have configure our test domain (`AnonDomain.TLD`) for the disjoint NSSet experiment as shown in Figure 2. For this experiment, we set the NSSet at the parent to [ns1, ns3].`AnonDomain.TLD`, while on the child authoritative servers, we set the NSSet to [ns2, ns4].`AnonDomain.TLD` (Table 2).

Zone files: we then configure the zone files of [ns1–ns4] to answer NS queries with [ns2,ns4], if explicitly asked, *i.e.,* the same records pointed to by the child authoritative servers. By doing that, we are able to single out resolvers that are *parent-centric*, since they will only contact [ns1,ns3].

	Disjoint		Subset		Superset		Rest	
Experiment	Min-Off	Min-On	Min-Off	Min-On	Min-Off	Min-On	Min-Off	Min-On
Measurement ID	*Anon.*	*Anon.*	*Anon.*	*Anon.*	*Anon.*	*Anon.*	*Anon.*	*Anon.*
Frequency	600s							
Duration	2h							
Query	`A` $probeid-$timestamp.`AnonDomain.TLD` with 30 seconds TTL							
NSSet Parent	[ns1, ns3]		[ns1, ns3]		[ns1, ns2, ns3, ns4]		[ns1, ns2, ns3, ns4]	
NSSet Child	[ns2, ns4]		[ns1, ns2, ns3, ns4]		[ns1, ns3]		[ns2, ns4, ns5, ns6]	
TTL NS Parent	3600 s							
TTL NS Child	3600 s							
Date	20191003	20191003	20191025	20191025	20191025	20191026	20191027	20191027
Probes	9028	9031	8888	8883	8892	23115432	8875	8875
VPs	15956	15950	15639	15657	15647	15611	15557	15586
Queries	190434	190333	184364	185706	186960	185015	182992	186472
Answers	178428	178416	169224	175200	175080	174804	174288	174504
From ns1, ns3	109661	175124	132179	169482	52233	83607	53944	84709
From ns2, ns4	65527	322	31753	1557	118835	86804	83100	85739
From ns5, ns6	N/A	N/A	N/A	N/A	N/A	N/A	31740	1545
fail	3240	2970	5292	4161	4012	4393	5504	2511

Table 2: Experiments to compare differents in Parent/Child NSSet

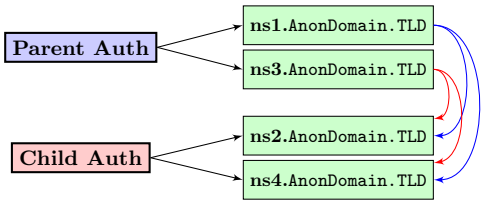
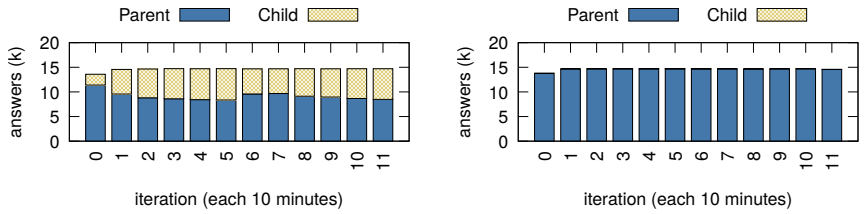


Fig. 2: Disjoint NSSset Experiment for `AnonDomain.TLD`

As vantage points, we use ∼9k Atlas probes, and configure them to send `A` queries through each of their resolvers for $probeid-$timestamp.`AnonDomain.TLD`, which encodes the unique Atlas probe ID and query timestamp, thus avoiding queries of multiple probes interfering with each other. We also set the TTL value of the record to 30 s, and probe every 600 s, so resolver caches are expected to be empty for each round of measurements [12].

Our goal is to determine, *indirectly*, which NS records were used to answer the queries.To do that, we configure [ns1,ns3] to answer our `A` queries with the IP `42.42.42.42`, and [ns2,ns4] with the IP `43.43.43.43`.

Figure 3a shows the results of the experiment. In round 0 of the measurements, we expect resolvers to have a cold cache and to use the NSSet provided by the parent. As the figure shows, this is mostly the case although 253 unique resolver IPs do contact the child name servers. This can be either due to them sending explicit NS queries (and thus learning about [ns2,ns4]) or because some probes share upstream caches. In subsequent rounds, we expect more traffic to go to the child name servers [ns2, ns4]. This is because resolvers learn about the child delegation from the "authority section" included in the response to the `A`



(a) Results for normal responses (b) Results with minimal responses

Fig. 3: Disjoint NSSet experiments

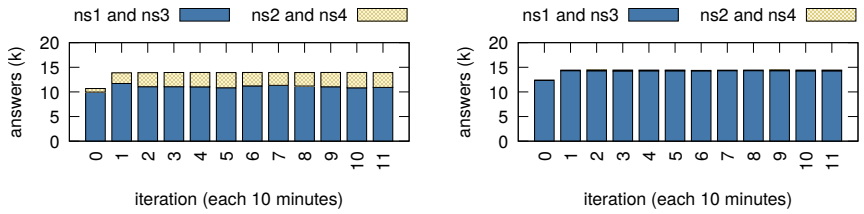
query to ns1 or ns3. According to RFC2181 resolvers may prefer this information over the delegation provided by the parent. Indeed, in rounds [1–11] we see traffic also going to the child name servers. However, not all traffic goes to servers in the child NSSet, because not all resolvers trust data from the "authority section" due to mitigations against the so-called Kaminsky attack [8]. A key takeaway of this experiment is that domain owners may mistakenly assume traffic to go to the name servers in the child NSSet if they change it, whereas for this change to be effective, they must also update the parent NSSet.

The situation is even worse in our second experiment. Here, we configure [ns1–ns4] to answer with *minimal responses*, which prevents these servers from including "extra" records in the authority and additional sections of DNS answers. This means we do not expect resolvers to learn about the existence of [ns2,ns4] at all, since they are no longer present in the "authority section" of responses to the `A` queries. Only if resolvers perform explicit `NS` queries will they learn about [ns2,ns4]. As Figure 3b shows, as expected, almost all resolvers exclusively send their queries to the name servers in the NSSet of the parent. Only about 40 vantage points receive data from the name servers in the child NSSet, indicating their resolvers likely performed explicit `NS` queries. Authoritative name servers are increasingly configured to return minimal responses to dampen the effect of DNS amplification attacks, especially for DNSSEC-signed domains [17]. A key takeaway from this experiment is with this configuration becoming more and more prevalent, it becomes even more important to keep parent and child NSSets correctly synchronized.

4.2 Parent NSSet is a subset of Child

Recall from Table 1 that the majority (69-73%) of cases in which parent and child NSSets differ fall into the category where the child NSSets contains one or more additional `NS` records not present in the parent NSSet. A common reason to add additional `NS` records is to spread load over more name servers, and we assume this to be one of the reasons for this common misconfiguration.

We set up experiments to determine the impact on query distribution if you have this setup. In other words: how many queries will eventually be answered by



(a) Results for normal responses (b) Results with minimal responses

Fig. 4: Subset NS sets experiments

the extra `NS` record? We configure our test domain with [ns1, ns3] at the parent and [ns1, ns2, ns3, ns4] at the child. Like in the previous section, we configure [ns1, ns3] to give a different response to the `A` queries sent by the Atlas probes than [ns2, ns4], so we learn how many queries were answered by the name servers that are only in the child NSSet.

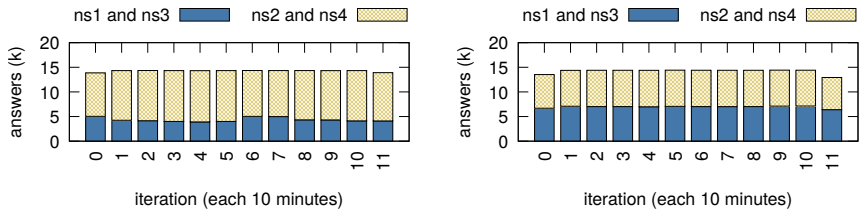
Figure 4a shows the results. Similary to the results shown in §4.1, most resolvers will use the `NS` records provided by the parent. Given that the child NSSet includes the NSSet at the parent, we see that the extra name servers receive only ∼24% of the queries. If in addition we configure the name servers to return minimal responses, we see that, just as in §4.1 virtually no resolvers contact the extra name servers in the child NSSet (Figure 4b). A key takeaway from these two experiments is that the, perhaps, expected even load distribution domain owners are hoping to see will not occur if only the child NSSet is updated. This again underlines the importance of keeping parent and child in sync.

Real-world case: `att.com`: A real-world example that demonstrates that this type of misconfiguration also occurs for prominent domains is the case of `att.com`. We discovered that AT&T's main domain `att.com` had a parent NSSet contains `[ns1...ns3].attdns.com`, whereas the child NSSet had `[ns1...ns4].attdns.com`. We notified AT&T of this misconfiguration and on 2019-10-24 the issue was resolved when the fourth name server (`ns4.attdns.com`) was also added to the parent.

4.3 Parent NSSet is a superset of Child

Roughly 14-18% of domain names that have different NSSet at parent and child have, one or more extra `NS` records at the *parent* ($P \supset C$ in Table 1). This could be due to operators forgetting to remove name servers that are no longer in use at the parent, but also the reverse case of the previous section in which a new name server is added at the parent but not added at the child.

To investigate the consequences of this for resolvers, we carry out experiments using Atlas VPs, setting four `NS` records at the parent ([ns1, ns2, ns3, ns4], as in



(a) Results for normal responses (b) Results with minimal responses

Fig. 5: Superset NS sets experiments

Table 2) and only two at the child ([ns1, ns3]). Our goal is to identify the ratio of queries answered by the extra `NS` records at the parent.

Figure 5a shows the results for the experiment. As can be seen, the servers listed only in the parent ([ns2,ns4]) answer, on average, 68% of the queries. In case minimal responses are configured (Figure 5b), we see the queries being distributed evenly among the NS records in the parent. Consequently, having authoritative servers include an authority section in their answer to the `A` queries seems to cause *some* resolvers to prefer the child NSSet over the one in the parent. For example, Atlas VP (21448, 129.13.64.5) distributes queries only among ns1 and ns3, in the case of normal responses, instead it distributes queries among all name servers in case of minimal responses.

These measurements then confirm that including "authority data" in the authoritative server responses will cause *some* resolvers to prefer only the child authoritative servers.

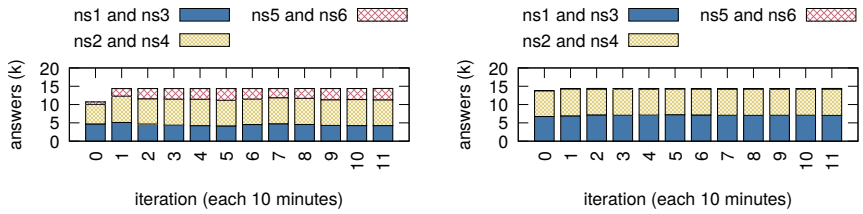
4.4 Mixed NSSets (Rest)

We have shown in Table 1 that in 11% of cases, the NSSet of the parent and child do not have a subset/superset relationship. Instead, some elements are present in both, but both parent and child have at least one NS that is not available in the other. To simulate this scenario, as shown in Table 2, we set four NS records at the parent: [*ns1*,ns2,*ns3*,ns4]. Then, at the child, we set [ns2,ns4,*ns5*,*ns6*], where the highlighted names show the ones not shared.

Figure 6a shows the experiment results. We see that [ns2,ns4], which are listed at both parent and child receive most queries. Then, records set only at the parent ([ns1,ns3]) are second to receive more queries. Finally, records set only at the child ([ns5,ns6]) receive the least amount of queries. In case of minimal responses (Figure 6b), the name servers only present at the child ([ns5,ns5]) receive virtually no traffic.

4.5 Discussion

Having inconsistent NSSets in parent and child authoritative servers impacts how queries are distributed among name servers, which plays an important role



(a) Results for normal responses (b) Results with minimal responses

Fig. 6: Rest NS sets experiments

in DNS engineering. Overall, for all evaluated cases, queries will be unevenly distributed among authoritative servers – and the servers listed at the parent zone will receive more queries than then ones specified in the child.

5 Resolver software evaluation

The experiments carried out in §3 evaluates DNS resolver behavior in the wild. Since we use RIPE Atlas, we do not know what resolver software is used, if probes use DNS forwarders, or what kind of cache policies they use. We, however, see the aggregated behavior among a large set of configurations.

In this section, we focus on evaluating specific DNS resolver software instead, in a controlled environment, in order to understand how they behave towards DNS zones that are inconsistent with regards to their parent/child NSSet. Our goal is to identify which vendors conform to the standards. In particular, we pay attention as to whether resolvers follow RFC2181 [5], which specifies *how* resolvers should rank data in case of inconsistency: child authoritative data should be preferred.

We evaluate four popular DNS resolver vendors *BIND*[2], *Unbound*[15], *Knot*[3], and *PowerDNS*[16]. We do this under popular Linux server distribution releases, using default packages and configurations. In addition, we evaluate resolvers shipped with various Windows server releases. Table 3 shows which vendors and versions we evaluate.

	Bind	Unbound	Knot	PowerDNS	Windows-DNS
Ubuntu-18-04	9.11.3-1	1.6.7	2.1.1	4.1.1	N/A
Ubuntu-16.04	9.10.3-P4	1.5.8	1.0.0	4.0.0	N/A
CentOS 7	9.9.4	1.6.6	2.4.1	4.1.9	N/A
CentOS 6	9.8.2rc1	1.4.20	N/C	3.7.4	N/A
Source	9.14.0	1.9.0	N/C	4.1.9	N/A
Windows	N/C	N/C	N/C	N/C	2008r2, 2012, 2016, 2019

Table 3: O.S. and resolver versions evaluated (N/Available, N/Covered)

	(i) A Query	(ii) NS Query	(iii) A Query Then NS Query		(iv) NS Query Then A Query	
Query			First	Second	First	Second
Answer	C(A)	C(NS)	C(A)	C(NS)	C(NS)	C(A)
Cache	C(A); C(NS)	C(NS)	C(A); C(NS)	C(A); C(NS)	C(NS)	C(NS); C(A)
Minimal response enabled						
Answer	C(A)	C(NS)	C(A)	C(NS)	C(NS)	C(A)
Cache	C(A); P(NS)	C(NS)	C(A); P(NS)	C(A); C(NS)	C(NS)	C(NS); C(A)
Information provided by: C⇒ Child, P⇒ Parent						

Table 4: Expected Resolver Behavior

Experiments: We configure the authoritative name servers for our test domain (`AnonDomain.TLD`) as a *disjoint* NSSet, as in §4.1. We configure the parent zone with [ns1,ns3].`AnonDomain.TLD`, and the child with [ns2,ns4].`AnonDomain.TLD`

Each experiment includes the four tests described in Table 4 (i–iv), in which we vary query types and query sequence. In (i), we ask the resolver for an `A` record of a subdomain in our test zone. In test (ii), we ask for the `NS` record of the zone. In (iii) we send first an `A` query followed by an `NS` query, to understand if resolvers use non-authoritative cached `NS` information to answer to the following query violating (§5.4.1 of RFC2181 [5]). In (iv) we invert this order to understand if authoritative record are overwritten by non-authoritative ones in the cache.

We dump the cache of the resolver after each query, and show which records are in cache and received by our client (we clear the cache after each query). Table 4 shows the expected NS usage by the resolvers, if they conform to the RFCs.

5.1 Results

We evaluate five resolver vendors and multiple versions. In total, we found that out of 22 resolvers/vendors evaluated, 13 conform to the RFCs. Next, we report the non-confirming resolver vendors/versions.

For experiment (i), in which we query for `A` records, we found that BIND packaged for Ubuntu did not conform to the standards: it caches only information from the parent and does not override it with information from the authoritative section provided by the child (which comes as additional section). This, in turn, could explain part of results of parent centricity observed in §4.

For experiment (i) and (iii), if we compile the latest *BIND* from source it also does not behave as expected: it sends the parent an explicit `NS` query in case of minimal responses enabled on the authoritative name server before performing the `A` query. This is not a bad behavior, *i.e.,* it does not violate RFCs, instead it tries to retrieve more authoritative information. However, either if the name server information retrieved and used in the following query is the one provided by the child, *BIND* caches the data from the parent. This behavior of BIND could be one explanation of the small number of child-centric resolvers shown in §4 with Minimal Responses.

We are in the process of notifying BIND developers about this issue.

For experiment (iii), *PowerDNS* packaged for `CentOS 6` and `Ubuntu Xenial`, and Windows (all) use the cached non-authoritative information to answer the *NS* query in the test, not conforming to RFC2181.

PowerDNS notification We reached out to the developers of *PowerDNS*, who have confirmed the behavior. They do not maintain older versions anymore and the fix will not be backported due to the low severity of the problem. Our suggestion to the package maintainers of the distributions is to update the software to a newer version of the software.

6 Conclusions and Recommendations

Given a domain name, its NSSet in the parent and child DNS zones should be consistent [11]. This is the first study that shows, across the `.com`, `.net` and `org` zones (50% of the DNS namespace), that roughly 7.8% (13M) domains do not conform to that. We also show that DNS resolvers in the wild differ in behavior in returning information from the parent or child.

Inconsistency in parent and child NSSets have consequences for the operation of the DNS, such as improper load balancing among the name servers, increased resolution latency and unresponsive name servers. We strongly advise operators to verify their zones and follow RFC1034. To automate this process, we advise zone operators to consider supporting CSYNC DNS records (RFC7477) or other automated consistency checks, so the synchronization can be done in an automated fashion.

Finally, we also recommend that resolver vendors conform to the authoritative information ranking in RFC2181 (taking into account the recommendations to mitigate the Kaminsky attack as specified in RFC5452), and when possible, to *explicitly* ask for the child's NS records, similarly to what is done in DNSSEC, where signed records are only available at the child (§5).

References

1. Almond, C.: Cname at the apex of a zone. https://www.isc.org/blogs/cname-at-the-apex-of-a-zone/
2. Consortium, I.S.: Bind: Berkeley internet name domain. https://www.isc.org/bind/
3. CZ.NIC: Knot resolver. https://www.knot-resolver.cz
4. DENIC AG: Statistics of .de domains (Oct 22 2019), https://www.denic.de/en/know-how/statistics/l
5. Elz, R., Bush, R.: Clarifications to the DNS Specification. RFC 2181, IETF (Jul 1997), http://tools.ietf.org/rfc/rfc2181.txt
6. Hardaker, W.: Child-to-Parent Synchronization in DNS. RFC 7477, IETF (Mar 2015), http://tools.ietf.org/rfc/rfc7477.txt
7. Hoffman, P., Sullivan, A., Fujiwara, K.: DNS Terminology. RFC 8499, IETF (Nov 2018), http://tools.ietf.org/rfc/rfc8499.txt

8. Hubert, A., Mook, R.v.: Measures for Making DNS More Resilient against Forged Answers. RFC 5452, IETF (Jan 2009), http://tools.ietf.org/rfc/rfc5452.txt
9. Kristoff, J.: Dns inconsistency. https://blog.apnic.net/2018/08/29/dns-inconsistency/ (2018)
10. Liu, D., Hao, S., Wang, H.: All your dns records point to us: Understanding the security threats of dangling dns records. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1414–1425. CCS '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978387
11. Mockapetris, P.: Domain names - concepts and facilities. RFC 1034, IETF (Nov 1987), http://tools.ietf.org/rfc/rfc1034.txt
12. Moura, G.C.M., Heidemann, J., Müller, M., de O. Schmidt, R., Davids, M.: When the dike breaks: Dissecting DNS defenses during DDoS. In: Proceedings of the ACM Internet Measurement Conference (Oct 2018). https://doi.org/https://doi.org/10.1145/3278532.3278534
13. Moura, G.C.M., Heidemann, J., de O. Schmidt, R., Hardaker, W.: Cache me if you can: Effects of DNS Time-to-Live (extended). In: Proceedings of the ACM Internet Measurement Conference. p. to appear. ACM, Amsterdam, the Netherlands (Oct 2019). https://doi.org/https://doi.org/10.1145/3355369.3355568
14. Müller, M., Moura, G.C.M., de O. Schmidt, R., Heidemann, J.: Recursives in the wild: Engineering authoritative DNS servers. In: Proceedings of the ACM Internet Measurement Conference. pp. 489–495. London, UK (2017). https://doi.org/https://doi.org/10.1145/3131365.3131366
15. NLNetLabs: Unbound. https://unbound.net/ (Mar 2019)
16. PowerDNS: Powerdns recursor. https://www.powerdns.com/recursor.html
17. van Rijswijk-Deij, R., Sperotto, A., Pras, A.: DNSSEC and Its Potential for DDoS Attacks: a comprehensive measurement study. In: Proceedings of the 2014 ACM Conference on Internet Measurement Conference. pp. 449–460. IMC, ACM (Nov 2014)
18. RIPE NCC Staff: RIPE Atlas: A Global Internet Measurement Network. Internet Protocol Journal (IPJ) **18**(3), 2–26 (Sep 2015)
19. RIPE Network Coordination Centre: RIPE Atlas. https://atlas.ripe.net (2015)
20. Root Zone file: Root (Feb 2019), http://www.internic.net/domain/root.zone
21. van Rijswijk-Deij, R., Jonker, M., Sperotto, A., Pras, A.: A high-performance, scalable infrastructure for large-scale active dns measurements. IEEE Journal on Selected Areas in Communications **34**(6), 1877–1888 (June 2016). https://doi.org/10.1109/JSAC.2016.2558918

The Forgotten Side of DNS: Orphan and Abandoned Records

Anonymous Author(s)

ABSTRACT

DNS Zone administration is a very complex task and can easily result in misconfigurations and vulnerabilities. Orphan records are one of these misconfigurations, in which a glue record for a delegation that does not exist anymore has been forgotten in the zone file. The goal of this paper is to analyze this misconfiguration reproducing and extending a previous work by Kalafut *et al.* Our study extends on this work by also introducing a new type of glue record misconfiguration, which we refer to as *abandoned records*. Our results highlight how the situation has changed, not always for the better, compared to a decade ago.

1 INTRODUCTION

The Domain Name System (DNS) [5] is one of the most important and complex infrastructures of the Internet. The primary task of DNS is to perform the conversion between human-readable hostnames and IP addresses, but nowadays DNS is also used for other services (*e.g.,* email, VoIP, etc...).

The DNS is a hierarchical, distributed database. The "." zone is the root of the DNS tree and is usually referred as *root* zone. The root zone provides information related to the nameservers that manage .com, .net, and all the other top-level domains (TLDs). TLDs are managed by different entities, *e.g.,* country-code TLDs (ccTLDs) are typically managed by national country registries, gTLDs are delegated by ICANN. The administration of a domain is a therefore complex task and involves multiple parties (registry, registrant, and registrar) and can be subject to misconfigurations and errors. Such misconfigurations can lead to reachability and security issues.

The goal of this paper is to analyze a specific misconfiguration, defined as the *orphan record* in a TLD. This work reproduces and extends the analysis performed by Kalafut *et al.* [4] in 2010 and is motivated by the main question: *A decade after the original analysis, what does the orphan records phenomenon look like now?* Compared to [4], we characterize the *orphan records* phenomenon through a significantly larger dataset and over a wider time window. We also introduce a related, but not considered before in literature, type of misconfiguration that we refer to as *abandoned records*, which could prelude to the creation of new orphan records.

IMC '19, October 21-October 23, 2019, Amsterdam, The Netherlands
.

2 BACKGROUND

Zone files and glue records. A zone is a portion of the DNS managed by a single entity. A zone file describes all the records related to a zone. The zone file format is defined in *RFC1034* and *RFC1035* [2, 6]. Resource Records (RRs) are defined in a textual form inside the zone file. Since the role of a TLD in the resolution process is to refer a recursive resolver to the authoritative server, the only RR types in a TLD zone, with exception of Start of Authority (`SOA`) RRs and DNSSEC-related RRs, are `NS` records (delegations) and `A/AAAA` records. `A/AAAA` records are known as *glue records* (defined in *RFC1033* [1]). If the `NS` record for a domain points to a record that is inside the domain (also called "in-bailiwick"), that name is included in the zone as a glue record to enable the resolution process to continue. Consider for example: `example.com NS ns1.example.com`. To resolve `example.com`, we need to resolve `ns1.example.com`, but this implies resolving the `example.com` delegation. Defining the glue record (`A/AAAA`) for `ns1.example.com` in the parent zone file breaks this loop.

Orphan and abandoned records. Glue records are supposed to be removed after a delegation is removed or changed. However, earlier work indicates that this does not always happen in practice [4]. In this paper, we define an **orphan record** as a former glue record for which the related domain no longer exists in the zone (the delegation has been removed). We also define an **abandoned record** as a former glue record for which the related domain still exists in the zone but the delegation no longer requires the glue record. Under normal operation, abandoned records do not show up in DNS resolving, as there is no longer relation with the domain the record was for. However, it is still questionable if they should remain at all in a TLD zone file. Finally, we define **junk records**, as the union of orphan and abandoned records. Generally, junk records are caused by bad administrative practices of TLD registry operators or registrars.

Related work. Our main goal is to revisit the work of Kalafut *et al.* [4]. Their work provides an overview of the problem of orphan records, an analysis of the spread of the problem and a characterization of orphan record usage, including lifetime and hosted resources. The study is performed using zone files for different TLDs and malware URL feeds. The duration of the analysis is 31 days. The work of Liang *et al.* [3], albeit not focusing on junk records specifically, proposes a method for keeping DNS records locked in the cache of

example.com	86400	IN	NS	ns.external.org
ns1.example.com	3600	IN	A	1.2.3.4
ns1.expired1.com	3600	IN	A	3.2.5.4
ns1.expired2.com	3600	IN	A	8.4.5.6
active.com	86400	IN	NS	ns1.expired2.com
good.com	86400	IN	NS	ns1.good.com
ns1.good.com	3600	IN	A	1.2.3.5

Table 1: Example Zone File
■ Algorithm 1 (O) ■ Algorithm 2 (A)

open DNS resolvers after the domain expires. The authors define these records as ghost records and prove that by performing queries against open resolvers and by crafting an ad-hoc response in the controlled authoritative nameserver it is possible to refresh the value of TTL for the record in the cache of the open resolver even if the domain does not exist anymore in the parent zone.

3 METHODOLOGY AND DATASET

Methodology. We develop two algorithms to respectively identify orphan and abandoned records inside zone files. These algorithms are based on the principle that in the zone file the only A records available are glue records.[1]

Algorithm 1 identifies orphan records and it is similar to the one described in [4]. The algorithm first collects all the domains in A records available inside the zone file. Then it trims the domains to the second level domain (SLD). Finally, it looks for the SLDs that do not have any associated NS record.

Algorithm 2 identifies abandoned records. The algorithm collects the list of domains in the A records available in the zone file, but instead of looking for the associated SLD, it joins this list of domain names with the list of NS values (the nameservers). The A records that do not appear as nameserver of any domains are marked as *junk records* (either orphan or abandoned records). Finally, algorithm 2 subtracts from the obtained list of junk records the orphan records recovered with algorithm 1 and returns only the abandoned records.

Table 1 provides an example of records retrieved by the two algorithms. Algorithm 1 identifies `ns1.expired1.com` and `ns1.expired2.com` as orphans since no NS records exist for `expired1.com` or `expired2.com`. Algorithm 2 marks `ns1.example.com` and `ns1.expired1.com` as junk records. Algorithm 2 filters out `ns1.expired1.com` since it is an orphan (no NS referring to it), while it returns `ns1.example.com` as an abandoned (a delegation for `example.com` exists, but points elsewhere).

[1]To be able to reproduce the work in [4], we do not consider AAAA records.

Dataset. The TLDs we chose for this study are `.aero`, `.asia`, `.biz`, `.ca`, `.com`, `.fi`, `.info`, `.mobi`, `.name`, `.net`, `.nu`, `.org`, `.ru`, `.se` and `.us`. We also include 1184 new gTLDs introduced by ICANN. In this paper, we will refer to these TLDs as CZDS. The zone files are collected on a daily basis. The combined zones cover a period of 25 months, from April 2017 to May 2019 (760 days). The collected data contains on average 3,283,404 unique A records per day, 199,249,769 unique domains and 1,317,987 unique in-bailiwick domains. However, a zone file might occasionally not be collected (*e.g.,* due to contract renewal processes). This means that our results are a lower-bound for the orphan and abandoned records problem. Table 2 lists the effective start and end dates for each TLD in our dataset and the percentage of days covered in the range. We use Apache Spark [7], an open source cluster computing framework, to perform our analysis.

4 RESULTS

4.1 Orphan records distribution

Kalafut *et al.* [4] identify `.info` as the TLD with the highest percentage of orphan records. Our analysis shows that 10 years later, the number of orphan records in this TLD is still rising (Table 3). Out of an average of 169,946 A records per day, in the period of analysis, an average 24.9% of these are orphan records, with a maximum of 36% and a minimum of 17.3%. Comparing these results to [4] we find an increase of 6.1% in the average and of 17.2% in the worst case.

The `.mobi` TLD shows a similar trend. With an average of 6,855 A records per day, the mean percentage of orphan records is 22.7% with a peak of 37.5%. Compared to [4], the number of orphan records doubled, with an increase of 12% in the total number of records. We note that the number of orphan records for `.mobi` has decreased over the last year, but found no evidence that this is due to a targeted cleanup action. A remarkable difference with [4] is that `.com` and `.net` no longer contain any orphan records. Our dataset does not allow us to pinpoint the moment when these records disappeared. We suspect however that some automatic cleaning mechanism has been introduced since 2010.

For `.asia` and `.org`, we find more records compared to [4] with an almost constant trend. For `.fi` and `.nu`, we find no orphan records. For `.us` and `.ca` and `.ru`, we find fewer records compared to [4], which underestimates the number of records for these TLDs, since they did not have access to zone files for these TLDs. The number of records is almost constant during the period of analysis.

4.2 Abandoned record distribution

The analysis of abandoned records shows different results compared to the orphan record analysis. The TLD with the highest percentage of abandoned records is `.org`, with a

TLD	Coverage	Start-Date	End-Date	# Domains	TLD	Coverage	Start-Date	End-Date	# Domains
info	98.3%	2017-04-01	2019-04-30	5521578	ca	94.3%	2017-04-01	2019-04-30	2664476
mobi	96.2%	2017-04-01	2019-04-30	464127	fi	97.5%	2017-04-01	2019-04-30	444198
asia	94.4%	2018-11-20	2019-04-30	255153	aero	94.4%	2018-11-20	2019-04-30	35062
org	99.9%	2017-04-01	2019-04-30	10319958	biz	93.8%	2018-11-20	2019-04-30	2063492
com	96.0%	2017-04-01	2019-04-30	132510044	name	94.4%	2018-11-20	2019-04-30	129792
net	98.6%	2017-04-01	2019-04-30	14000292	nu	99.3%	2017-04-01	2019-04-30	387985
us	99.4%	2018-11-19	2019-04-30	2044937	se	99.3%	2017-04-01	2019-04-30	1616161
ru	99.1%	2017-06-17	2019-04-30	4978742	CZDS	99.9%	2017-04-01	2019-04-30	22068927

Table 2: Overview of datasets used in this work

TLD	A Record	# Orphan	Min	Max	Mean	Prev	# Abandoned	Min	Max	Avg	Sum
.info	169946	43687	17.3%	36.0%	24.9%	18.8%	68619	35.7%	47.9%	40.8%	65.8%
.mobi	6855	1602	5.5%	37.5%	22.7%	10.7%	2949	35.7%	53.7%	43.6%	66.3%
.asia	6122	1140	17.8%	19.9%	18.6%	7.5%	3097	49.2%	52.1%	50.6%	69.2%
.org	364568	21929	4.4%	7.5%	6.0%	3.7%	227161	60.8%	63.6%	62.3%	68.3%
.com	1873668	0	0.0%	0.0%	0.0%	0.4%	0	0.0%	0.0%	0.0%	0.0%
.net	303387	0	0.0%	0.0%	0.0%	0.2%	0	0.0%	0.0%	0.0%	0.0%
.us	26042	1869	6.8%	8.0%	7.2%	3931*	0	0.0%	0.0%	0.0%	7.2%
.ru	79492	54	0.0%	0.1%	0.1%	1801*	0	0.0%	0.0%	0.0%	0.1%
.ca	23537	980	3.9%	4.5%	4.2%	1368*	0	0.0%	0.0%	0.0%	4.2%
.fi	3908	0	0%	0%	0%	N/A	0	0.0%	0.0%	0.0%	0.0%
.aero	626	22	3.0%	4.3%	3.6%	N/A	326	50.6%	53.4%	52.1%	55.7%
.biz	22958	6	0.0%	0.0%	0.0%	N/A	0	0.0%	0.0%	0.0%	0.0%
.name	1820	50	2.2%	3.0%	2.7%	N/A	0	0.0%	0.0%	0.0%	2.7%
.nu	2184	0	0.0%	0.0%	0.0%	N/A	0	0.0%	0.0%	0.0%	0.0%
.se	20053	48	0.2%	0.3%	0.2%	N/A	0	0.0%	0.0%	0.0%	0.2%
CZDS	395706	17433	0.5%	23.9%	5.1%	N/A	83189	3.0%	30.9%	21.3%	26.4%

Kalafut *et al.* report the number of orphans instead of the percentage for these TLDs [4].

Table 3: Orphan records for each TLD

mean of 62.3% abandoned records and a peak of 63.6%. The `.info` TLD performs better in percentage terms but worse in absolute numbers compared to the orphans. Considering the percentage of orphan records and abandoned records together, `.mobi`, `.info`, `.asia` and `.org` show a percentage of junk records around 65%, casting doubts about the management of these zones. For `.com`, `.net`, `.us`, `.ru`, `.ca`, `.fi`, `.biz`, `.name`, `.nu` and `.se`, we find no abandoned records.

4.3 IP address and domain distribution

We now investigate how many domains and how many IP addresses are related to junk records. The distribution of the IP addresses related to the orphan and abandoned records shows 38% of records that point to a single IP address (physical machine) for orphans and 67% for abandoned records. Moreover, 88% of orphans and 92% of abandoned records refer to a single or to two IP addresses, with an average of 2.45 and 1.83 orphan and abandoned records respectively per IP. Compared to [4], the number of orphans per IP decreased (from an average of 3.2 orphans per IP in [4]). Since `.com` and `.net` dominated the number of orphans in [4], we assume that the cleansing of these zones is reflected in this decrease. There are also some peculiar cases. For example, in `.info`, 1,754 orphan records point to CloudFlare's public resolver `1.1.1.1`. This is the result of a misconfiguration of NS resolvers (circular dependency), which causes unreachability.

We also analyze domain distributions. In 87% of the cases, we find two orphan records for a single SLD. This result is consistent with the common configuration practice of DNS, in which administrators set up two authoritative nameservers, thus two `A` glue records for a domain. In 7% of the cases, we find one orphan record for a single SLD. For abandoned records, in 57% of the cases, we find two abandoned records for a single SLD and in 30% of the cases, we find one.

4.4 Lifetime of records

Fig. 1 shows the lifetime CDF of orphan and abandoned records. The lifetime is defined as the continuous time elapsed

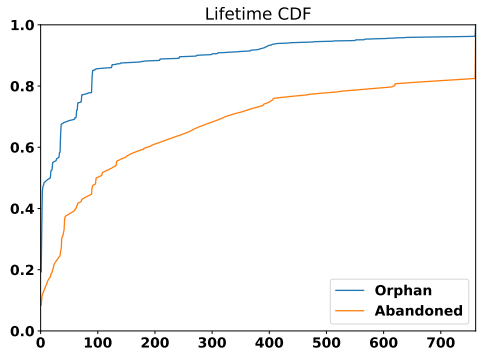


Figure 1: Lifetime of orphan and abandoned records

between the first and the last time that we spot a record as an orphan or abandoned in our analysis. The plot contains only data from `.info`, `.mobi`, `.org`, `.ca`, `.se`, CZDS. We decided to not include other TLDs because we can not calculate the lifetime of their records over the entire period of analysis (since the zone collection started later in time). The shape of the CDF is similar among the different TLDs, with some exceptions explained later. Compared to [4], the number of orphan records that live at most 1 day is 19%, which is higher compared to 12% in that study. [4] indicates that only 2% of the records last longer than their measurement duration (31 days). In our case, 43.4% of domain last at least 31 days, and 4% of records survive for more than 760 days, that is our time frame for the analysis. Around 86% of records survive for a maximum of 100 days. Therefore, we can confirm that these types of records became a long-term misconfiguration, which persistently affects the TLD zones.

The results for abandoned records are worse than the orphan ones. Only 8% of records live one day or less and 28% of records live more than 760 days. This difference has various root causes. One is that `.ca` and `.se` have no abandoned records. Another cause is that abandoned records in `.org` and new gTLDs live longer than orphan records compared to `.mobi` and `.info`, in which the lifetime curve is similar. Other interesting results come from the analysis of single TLDs. For `.ca` over 50% of records live more than 760 days. For new gTLDs after 100 days, only 2% of records are still present. Also in this case, our analysis suggests that abandoned records are a long-term misconfiguration.

4.5 NS records reference

We find 39,683 NS records that refer to orphan records, either in the same zone or in other zones. In Table 4, we report the reference matrix for each TLD. For typographical reasons, we exclude empty and not relevant columns. The most referenced orphan records are in `.ca` and are referenced ~10K times in the `.ca` and ~6K times in `.com`. A domain that sets NS records to orphan records run into a serious vulnerability. An attacker can register the orphan record and take control of the domain performing an NS hijacking. This matrix also helps us understand that the removal of an orphan record could have an impact on other domains in other TLDs, and for this reason should be analyzed carefully.

Orphan records											
Ref by	ru	org	asia	CZDS	se	name	mobi	us	ca	info	Sum
aero	0	0	0	0	0	0	0	36	4	0	40
asia	0	0	26	23	0	0	0	0	0	11	60
biz	0	17	12	93	4	0	2	62	101	96	387
ca	0	16	0	5	0	0	0	2	10320	24	10367
com	0	1337	41	276	34	2	19	3923	6223	1111	12966
CZDS	3	194	11	404	0	0	1	44	208	292	1157
fi	0	0	0	0	0	0	0	0	0	38	38
info	0	101	5	126	4	1	7	291	219	453	1207
mobi	0	8	4	99	0	0	30	7	149	5	302
name	0	0	0	28	2	68	0	0	9	0	107
net	0	277	10	139	19	0	3	1566	874	136	3024
nu	0	14	0	0	18	0	0	0	1	0	33
org	0	288	6	133	14	2	3	4695	1038	160	6339
ru	43	26	0	3	0	0	24	0	0	14	110
se	0	0	0	0	110	0	0	0	0	2	112
us	0	33	5	22	0	0	0	3287	67	20	3434
Sum	46	2311	120	1351	205	73	89	13913	19213	2362	39683

Table 4: NS records pointing to orphan records

5 CONCLUSIONS

This study was motivated by the idea of looking at the state of orphan misconfiguration after a decade from the first study by Kalafut *et al.* [4]. We discovered that for some TLDs (*e.g.*, `.com` and `.net`) the number of orphan records fell to zero or to values near to zero, which means that operators have introduced mechanisms for cleaning their zone files. Unfortunately, these best practices are not adopted by all TLDs. For some TLDs, the number of orphan records increased in 10 years. Also, in the new gTLDs introduced after [4], this type of records is present, indicating that this misconfiguration is widely spread among TLDs. We also discover and analyze another misconfiguration: the abandoned record. Our analysis shows that this misconfiguration is broader than the orphan misconfiguration. Even if these records are not resolved, common sense would suggest they should be removed, as they potentially represent the initial stage of orphan creation. We also recognize by looking at reference matrix that the removal of these records from the zone file may not be a simple operation, since it can incur the risk of breaking other domains.

REFERENCES

[1] 1987. Domain Administrators Operations Guide. RFC 1033. (Nov. 1987). https://doi.org/10.17487/RFC1033

[2] 1987. Domain names - concepts and facilities. RFC 1034. (Nov. 1987). https://doi.org/10.17487/RFC1034

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

[4] Andrew J Kalafut, Minaxi Gupta, Christopher A Cole, Lei Chen, and Nathan E Myers. 2010. An empirical study of orphan DNS servers in the internet. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 308–314.

[5] J. Klensin. 2003. *Role of the Domain Name System (DNS)*. RFC 3467. RFC Editor.

[6] P. Mockapetris. 1987. Domain names - implementation and specification. Internet Requests for Comments. (November 1987). http://www.rfc-editor.org/rfc/rfc1035.txt http://www.rfc-editor.org/rfc/rfc1035.txt.

[7] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1863103.1863113