TsuNAME: exploiting misconfiguration and vulnerability to DDoS DNS

Giovane C. M. Moura (1) Sebastian Castro (2) John Heidemann (3) Wes Hardaker (3)
1: SIDN Labs 2: InternetNZ 3: USC/ISI

ABSTRACT

The Internet's Domain Name System (DNS) is a part of every web request and e-mail exchange, so DNS failures can be catastrophic, taking out major websites and services. This paper identifies TsuNAME, a vulnerability where some recursive resolvers can greatly amplify queries, potentially resulting in a denial-of-service to DNS services. TsuNAME is caused by cyclical dependencies in DNS records. A recursive resolver repeatedly follows these cycles, coupled with insufficient caching and application-level retries greatly amplify an initial query, stressing authoritative servers. Although issues with cyclic dependencies are not new, the scale of amplification has not previously been understood. We document real-world events in `.nz` (a country-level domain), where two misconfigured domains resulted in a 50% increase on overall traffic. We reproduce and document root causes of this event through experiments, and demostrate a 500× amplification factor. In response to our disclosure, several DNS software vendors have documented their mitigations, including Google public DNS and Cisco OpenDNS. For operators of authoritative DNS services we have developed and released `CycleHunter`, an open-source tool that detects cyclic dependencies and prevents attacks. We use `CycleHunter` to evaluate roughly 184 million domain names in 7 large, top-level domains (TLDs), finding 44 cyclic dependent NS records used by 1.4k domain names. The TsuNAME vulnerability is weaponizable, since an adversary can easily create cycles to attack the infrastructure of a parent domains. Documenting this threat and its solutions is an important step to ensuring it is fully addressed.

ACM Reference Format:
Giovane C. M. Moura, Sebastian Castro, John Heidemann. 2021. TsuNAME: exploiting misconfiguration and vulnerability to DDoS DNS . In *ACM Internet Measurement Conference (IMC '21), November 2–4, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3487552.3487824

1 INTRODUCTION

The Internet's Domain Name System (DNS) [27] provides one of the core services of the Internet, by mapping hosts names, applications, and services to IP addresses and other information. Every web page visit requires a series of DNS queries, and large failures of the DNS have severe consequences that make even large websites and other Internet infrastructure fail. For example, the Oct. 2016 denial-of-service (DoS) attack against Dyn [5] made many prominent websites such as Twitter, Spotify, and Netflix unreachable to many of their customers [40]. Another DoS against Amazon's DNS service affected large number of services [61] in Oct. 2019.

The DNS can be seen as a hierarchical and distributed database, where DNS *records* [28] are stored in and distributed from *authoritative servers* [18] (for instance, the Root DNS servers [51] distribute records from the Root DNS zone [52]). As such, all information about an end domain name in the DNS are served by *authoritative servers* for that domain. This information is typically retrieved by *recursive resolvers* [18], which answer questions originally posed by users and their applications. Resolvers are typically operated by a user's ISP, or alternatively public DNS resolvers operated by Google [15], Cloudflare [1], Quad9 [43], Cisco OpenDNS [38], and others.

The configuration of authoritative servers and their records is prone to several types of errors [2, 25, 27, 39, 55]. Here we are concerned about *loops* where records required for resolution point at each other. *Cyclic dependencies* occur when resolving a name requires resolution of another name, that in turn refers back to the first [39]. Loops can involve CNAMEs (§ 3.6.2, [27]) or NS records (§ 2, [25]). For example, if the NS record for `example.org` points to `example.com` and vice versa, then an attempt to resolve any name within either domain will fail because the IP address for both servers cannot be confirmed.

The first contribution of this paper is to report that, in the wild, cyclic dependencies can result in a query cascade that greatly increases traffic to authoritative servers. We call this amplification *TsuNAME* (inspired by the destructive potential of a tsunami) and describe the several factors that contribute to it in §2. TsuNAME amplifcation has happened multiple times in the real world. §3 shows an event on 2020-02-01 at `.nz`, where a *configuration error* (not an intentional attack) in two domains each having cyclic nameservers (NS records). While normally these domains result in only a few queries to `.nz`'s authoritative DNS servers, the misconfiguration resulted in 50% increase in aggregate traffic volume (from 800M to 1.2B daily queries, the shaded area in Figure 1). While these servers handled this increase in load, this large amplification shows the risk a malicious attack could pose. Others have seen greater increases: §6 shows a European ccTLD that experienced a 10× increase in traffic due to TsuNAME.

These accidental events raise the question of what a motivated attacker could do to exploit this problem. An intentional attack could leverage multiple cycles to amplify moderate client traffic to overwhelm authoritative servers. In addition, since DNS providers often host multiple domains on shared infrastructure, other services could suffer collateral DoS damage (we discuss this threat model in Appendix F). This threat poses a great concern for any domains

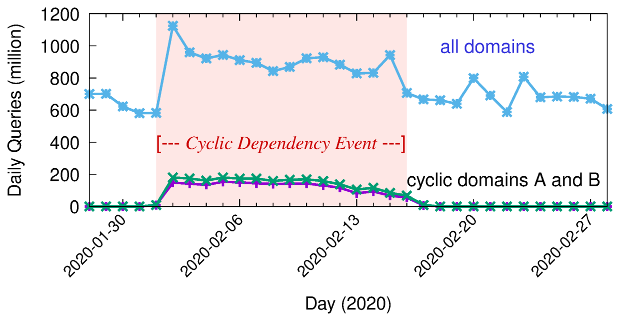


Figure 1: Daily queries to .nz for all domains (top) and the cyclic dependent domains (bottom two lines).

and registrations points, such as TLDs and ccTLDs, who often host domains that provide essential services to their users, such as government websites, banking, and online shopping.

Our second contribution is to demonstrate this threat in controlled conditions in §4. We *emulate* a TsuNAME event by setting up multiple cyclic dependent domain names under our control on our servers (so as to not harm others) and measure the consequence, reaching an amplification factor of more than 500×. Google operates Google Public DNS (GDNS), a large, popular public resolver service [15] and makes up 8% of *all* queries sent to `.nz` [30]. We show that GDNS was responsible for the bulk of queries, but we also found other vulnerable resolvers in 260 Autonomous Systems (ASes). Following responsible disclosure practices, we notified Google, Cisco OpenDNS, and other TLD and resolver operators (§6.) We worked with Google about GDNS (§4.5) and Cisco about OpenDNS, both of which have since been fixed.

Our third contribution is to develop `CycleHunter`, a tool that finds cyclic dependencies in DNS zone files (§5). This tool allows *authoritative* server operators (such as ccTLD operators) to identify and mitigate cyclic dependencies, *preemptively* protecting their authoritative servers from possible TsuNAME attacks. We use `CycleHunter` to evaluate the Root DNS zone and 7 other TLDs (∼185M domain names altogether), and found cyclic dependent domains in half of these zones. We made `CycleHunter` publicly available on GitHub and we thank the various contributors that have helped improve the tool. We have carefully disclosed our findings with the relevant DNS communities and include their contributions and feedback (§6) and discuss ethics around our disclosure and data release policies in §7.

2 THE TSUNAME PROBLEM

The Cause: The fundamental problem in TsuNAME is a cyclic dependency that amplifies a traffic sent to authoritative servers. When two zones have name server (NS) records that point at each other, a recursive resolver trying to resolve a name in that zone will loop between the two, trying to break the cycle. These queries mean a single query from a client to result in recursive resolvers placing many queries against the authoritative servers, perhaps overwhelming authoritative server capacity. Amplification in TsuNAME amplification is this end-to-end effect, with several contributing factors described below—a more complex process than traditional DNS amplification where a short query directly creates a large reply [23].

The potential of DNS cycles was documented in the initial specification in 1987 [27] and was called out as an implementation risk in 1993 [25]. Nevertheless, cyclic dependencies were noted as a continuing risk in 2004 [39], and they cause huge traffic volumes even in 2020 (§3). The contribution of our paper is not to document cyclic dependencies as a *new* problem, but to show that their impact can be huge even today, and that impact can be amplified by modern recursive resolver architectures.

DNS cycles cause a problem when an end-user's query results in excessive traffic to authoritative DNS servers. However, the amount of traffic depends on the interaction of a number of components of the DNS system:

(1) The injection rate of client queries to the cycle.
(2) Stub resolvers query retries and parallel queries [62].
(3) The number of independent caches in recursive resolver services [44, 53].
(4) If recursive resolvers cache negative replies.
(5) If recursive resolvers send additional requests for an unresolved pending request.
(6) Recursive resolver limits on the number of queries made when responding to an incoming request.

We careful example components (3) to (6) in this paper, since as infrastructure, they are easier to mitigate.

The Threat: *The risk of TsuNAME is that it is "weaponizable".* The authoritative servers of *any zone with third-party registration* are at risk. An adversary can register two or more domains, later reconfigure them to create a cyclic dependency, then inject client traffic from a botnet. TsuNAME causes recursive resolvers amplify injected traffic; in some prior events by more than 500×. Registrating domains without cycles is easy, cheap, and benign. The adversary can then activate the attack by reconfiguring to create a cycle and triggering traffic from a botnet—none of these steps are visible to the authoritative operator until the attack takes effect.

Although we (§3) and others (§6) encountered the problem by accident, we want to prevent its intentional use. We expand on this threat in Appendix F.

Recommendation: Our recommendation is the same as 1987 [27] (and repeated in 1993 [25]): resolver implementations *must* "bound the amount of work. . . so a request can't get into an infinite loop or start a chain reaction. . . *even if someone has incorrectly configured some data*".

We add one prescription and one recommendation to this guideline: *all recursive resolvers must implement negative caching, so they do not repeatedly retry a cyclic question. Negative caching* means recording (and replying) replies that are errors, replying immediately to a query from a new client rather than repeating the amplification. This design choice would have mitigated the problems that we report on below, and has been deployed by Google Public DNS (§4.5 and other vendors §5.3).

We also recommend that operators of parallel resolution systems to share caches between resolvers, and we strongly encourage recursive resolvers to avoid making duplicate requests for the same content when one request is still being resolved. (Authoritative servers typically already "cache" results in memory; additional caching does not reduce their network traffic.)

We encourage zone operators to scan for cycles using our tool to detect accidental cycles (§5).

While cycles were recognized as a problem more than 30 years ago, the problem is pressing *today* because *evolution of the system has amplified query rates*: the "happy eyeballs" algorithm doubles queries [62] and large public resolvers often use parallelism across many machines without shared caches [44]. Recommendations from 1987 and 1993 emphasize the role of the single recursive resolver without considering the interactions in today's DNS ecosystem.

3 TSUNAME'S IMPACT IN .NZ

On 2020-02-01, two domains in .nz (DomainA and DomainB) had their NS records misconfigured to be cyclically dependent. DomainA's NS records were set to ns[1,2].DomainB.nz, while DomainB's NS records pointed to ns[1,2].DomainA.nz. This misconfiguration increased the query volume to .nz's authoritative servers by 50%. Figure 1 shows this problem, as measured at the authoritative servers of .nz. The .nz operators manually fixed this misconfiguration more than two weeks later on 2020-02-17, returning query volumes to normal.

We describe this problem here, reproduce it in §4, and discuss detection in §5. While this traffic increase was large, .nz servers are overprovisioned and handle other bursts (such as those on 2020-02-20 and -23 in Figure 1), but a malicious attack could create much greater traffic volumes using more domains and more initial traffic.

3.1 Query sources

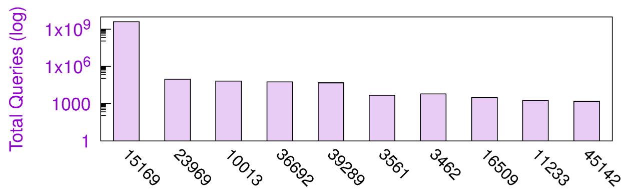
During the sixteen-day period of this TsuNAME event (2020-02-[01–17]), there were 4.07B combined queries for DomainA and DomainB, with a daily average of 269M. Figure 2a and Table 11 show the top 10 ASes by query volume during the event period. The overwhelming majority (99.99%) of all traffic originated from Google (AS15169), with only 324k queries originating from 579 other ASes. Queries from Google outnumbered the other sources by 4 orders of magnitude.

For comparison, Figure 2b shows the top 10 ASes for both domains during the "normal" periods when there was no cyclic dependency, spanning over the 16 days before and after the TsuNAME period (2020-01-[24–30] and 2020-02-[18–28]). During this "normal" period, Google sent no more than 100k daily queries for both DomainA and DomainB. During the TsuNAME period, however, Google's query volume multiplied 5453× (Figure 2c). No other AS had an traffic growth greater than 100× in the same period.

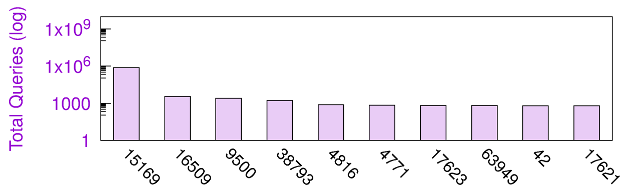
3.2 Scrutinizing Google queries

To understand *Why was Google responsible for so many queries?* we turn to: *How long and how many times should resolvers retry when resolving domains with cyclic dependencies?* And *how aggressive should they be when finding answers?*

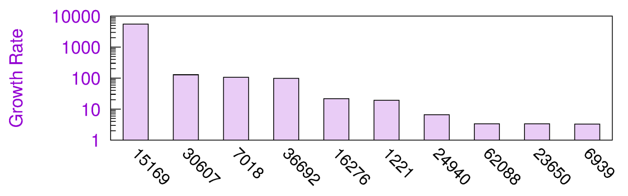
Previous research has shown resolvers will *hammer* unresponsive authoritative servers [34], generating up to 8× typical query rates, depending on the DNS records' time-to-live (TTL) value. But in the case of cyclic dependencies, authoritative servers are *responsive* and resolvers *bounce* from one authoritative server to another, asking the same sequence of questions repeatedly. This difference



(a) TsuNAME period



(b) Normal period



(c) Growth rate

Figure 2: Top 10 ASes querying for Domains A and B, for .nz. TsuNAME period: Feb. 1–17, normal period: Jan. 24–30, Feb. 18–28, 2020.

allows recursive resolvers to generate even more traffic than with unresponsive servers.

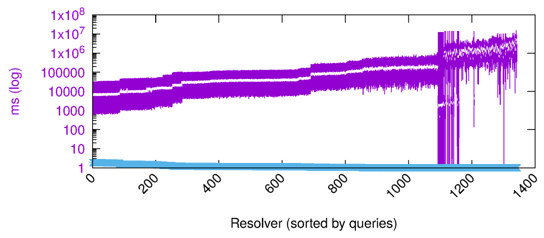
Given that Google was responsible for virtually all queries during the .nz TsuNAME event for the cyclic dependent domains (§3.1), we isolate and study the queries from Google. Table 1 shows the breakdown of the query names and types from Google during the .nz event. We see that most queries to .nz are for A and AAAA records for each of the two domain's own NS records (NS records store the authoritative server names of a domain, while A [27] and AAAA records [57] store each server's IPv4 and IPv6 addresses, respectively.) With a cyclic dependency, these queries can never be resolved, since each authoritative server refers resolvers to the other. The NS records for the zones, however, were readily available within the parent .nz zone – which explains the lower volume of queries compared to the A/AAAA requires.

3.2.1 *Interquery interval.*

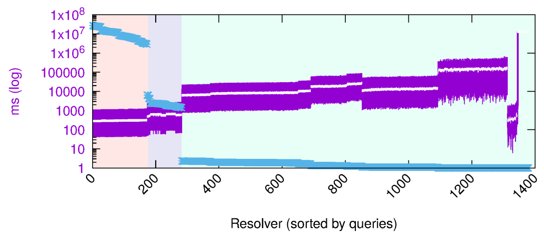
How frequently did GDNS resolvers send .nz queries for these domains during the TsuNAME event? We estimate this query rate by measuring the inter-query interval for queries with the same query name and type from GDNS to the .nz authoritative servers on one day (2020-02-06).

Query Name	Query Type	Queries(v4)	Queries(v6)
DomainA.nz	NS	13.0M	10.9M
DomainB.nz	NS	4.3M	3.0M
ns1.DomainA.nz	A	266.1M	281.3M
	AAAA	266.2M	281.4M
ns2.DomainA.nz	A	266.1M	281.2M
	AAAA	266.1M	281.4M
ns1.DomainB.nz	A	222.6M	237.9M
	AAAA	222.5M	237.7M
ns2.DomainB.nz	A	222.5M	237.7M
	AAAA	222.3M	237.5M

Table 1: Google queries during the `.nz` event



(a) NS queries for `DomainA.nz`



(b) A queries for `ns1.DomainA.nz`

Figure 3: Google (AS15169) resolvers on 2020-02-06, during `.nz` TsuNAME event: time in between queries.

Figure 3 shows the results (for space constraints, we show only results for the queries highlighted in the green rows of Table 1). We start with the NS queries to `DomainA.nz`. Figure 3a shows individual resolvers on the x axis, and the number of queries they sent on the right y axis. We see that all resolvers send fewer than 10k queries. On the left y axis, we show the interval inter-quartile range (IQR) of the time between queries (with the white line showing the median value in ms). Given that the TTL value of these records is 86400 s (1 day), we should not see (assuming low packet loss) any resolver sending more than one query on this date (except for multi-cache or anycast-based resolvers [34, 44]). (While both domains in Figure 3 show some recursive resolvers query frequently, we expect the recursives see different offered load from clients, and we sort them differently, so the curves differ in detail.)

As shown in Table 1, the bulk of queries is for A and AAAA records of the authoritative servers of `DomainA` and `DomainB` . Figure 3b shows the results for A records of `ns1.DomainA.nz`. We see three categories of resolvers, according to their query volume, which we highlight in different colors. The first group – *heavy hammers* – sent 162-186k queries on this day, one every 300 ms. The second group – *moderate hammers* – sent 75-95k daily queries, one every 590 ms – roughly double the rate of the heavy hammers. The last group, which covers most of the addresses – is less aggressive: they sent up to 10k daily queries each. Given they are more numerous than the other group, their aggregated contribution matters. (Appendix A shows the results for AAAA records, which are similar to Figure 3b).

This heterogeneity in Google's resolver behavior is surprising. We notified Google and were able to work with them on the issue, and they both confirmed and fixed their Public DNS service on 2020-02-05. We discuss this in detail in §4.5.

	Zones	
	`sub.verfwinkel.net`	`sub.cachetest.net`
NS	`ns.sub.cachetest.net`	`ns.sub.verfwinkel.net`
TTL	1s	1s

Table 2: New domain experiment setup.

4 REPRODUCING TSUNAME

To understand TsuNAME we next recreate the problem on the Internet through controlled experiments (§4.1), the role of clients (§4.2), multiple-step cycles (§4.3), and recursives (§4.4 and §4.5).

4.1 Controlled Experiments on a New Domain

To determine the *lower* bound on traffic to authoritative servers experience during a TsuNAME event, we carry out a controlled experiment from RIPE Atlas [47] to a new domain under our control. Since this is a new domain, we know there is no caching or prior query history.

Setup: We configure two third-level domains with cyclic dependencies (Table 2). We use third-level domains since TsuNAME traffic goes to the *parent* of the cyclic domains. A third-level domain allows us to isolate traffic in our (second-level-domain) authoritative servers, protecting top-level domains that are widely shared. If we create a cycle in a second-level domain like `example.org`, then traffic goes to the widely used `.org` authoritative servers.

We ran our own authoritative servers using BIND9 [22], popular open-source software for authoritative DNS service. We ran on Linux VMs located in AWS EC2 in Frankfurt, Germany.

To minimize caching effects, we set the TTL for every record in the zone to 1 s (Table 2). Short TTLs maximize the chances of cache misses and avoid hiding misbehavior with caches.

Vantage points (VPs): we use ~10k RIPE Atlas probes [47, 48] as VPs. RIPE Atlas provides more than 11k active devices (probes or VMs), distributed over 6740 global ASes (as of Jan. 2021). Atlas supports custom queries of standard types (ICMP, DNS, HTTP, etc.), and they archive and provide public access to research results, including our experiments [46].

We configure each probe to query *once* for an A record for `PID.sub.verfwinkel.net.`, where PID is the probe unique identifier [49]. Unique queries reduced the risk of accidentally warming up the resolver's caches with queries from other VPs. The query is sent to each probe's local resolver, as can be seen in Figure 4.

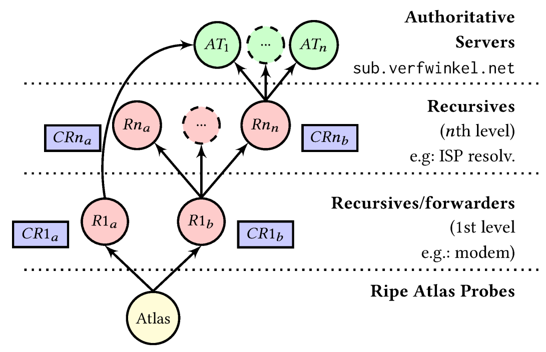


Figure 4: Relationship between Atlas probes (yellow), recursive resolvers (red) with their caches (blue), and authoritative servers (green).

As one probe may have multiple resolvers, we identify a VP using the combination of a probe's unique ID and the last resolver's IP address.

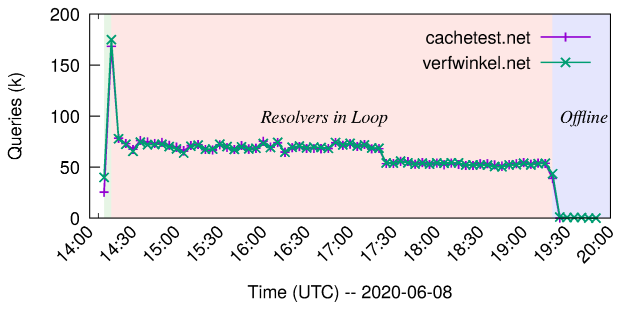
Results: Table 3 shows the results for this measurement ("new domain" column). We see ~9.7k Atlas probes placing queries through 16.8k recursive resolvers (the vantage points). With a few additional forwarders, they send 18,715 queries to their first level recursives (archived at Ripe Atlas and accessible at [46]), which are mostly answered to the probes with SERVFAIL status codes [27] or simply timing out – both statuses signal domain name resolution issues to the client.

Heavy traffic growth on the authoritative server side: Authoritative servers see queries from ~11k IPs addresses belonging to n_{th}-level servers in ~2.6k ASes (traffic into top green circles in Figure 4). As each Atlas probe query its recursive resolvers some forward their queries or ask multi-level resolvers [34, 44, 53]. Our authoritative servers see only the *final* resolver in the chain. In total, these resolvers send ~ 8M queries to both authoritative servers, a 435× *amplification factor* compared to the 18k queries at the client-side. We believe this amplification contributes to the .nz event.

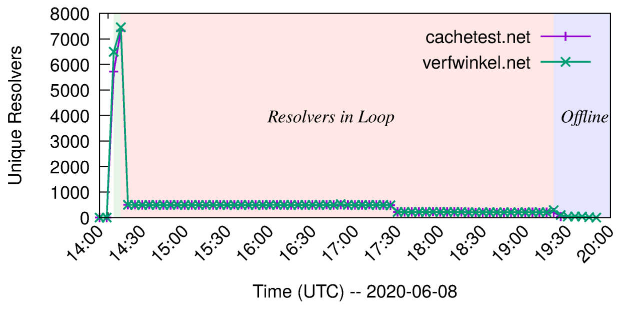
Identifying problematic resolvers. Figure 5 shows the timeseries of both queries and resolvers we observe at our authoritative servers (each line shows a different authoritative server, one per domain). We identify three phases in this measurement. First *warm-up* (the narrow, green shaded area, x < 14:30 UTC) when the VPs send the queries we have configured. We see more than 150k queries (Figure 5a) to each authoritative server, from roughly 7.5k resolvers (Figure 5b). In this initial rush each recursive repeats queries trying to resolve the cycle. Many have a query limit and break out after this rush.

After this initial rush, there are no new client queries, and at 14:30 we enter the *looping* phase, when authoritative servers keep on receiving queries from some *problematic* resolvers that do not have a query limit (the salmon-colored area, "Resolvers in Loop"). For the next hour a few resolvers (574 from 37 ASes) seem to loop indefinitely (Table 4).

Finally, in the *offline* phase, after 19:30 UTC, we stop our authoritative servers, but keep observing incoming queries. As our servers become unresponsive, even problematic resolvers cannot



(a) Queries



(b) Resolvers

Figure 5: New domain measurement: queries and unique resolvers timeseries (5min bins)

obtain answers and stop looping. Without our manual intervention (stopping our servers), one may wonder when (or if!) these loops would stop. We show in §5.2 that these loops may last for *weeks*.

Other ASes also affected: Figure 6 shows the histogram of queries per source ASes. We see than the AS with most traffic (Google, 15169) is responsible for 60% of the queries. Google here has a far more modest fraction than on the .nz event (§3). We see looping traffic from other ASes as well, such as AS200050 (ITSvision) and AS30844 (Liquid Telecom). In fact, we found in this experiment that 37 ASes were vulnerable to TsuNAME (Table 4). Although posing a much lower query volume than Google, old resolver software represents an ongoing risk.

How often do the problematic resolvers loop? For each problematic resolver, we compute the interval between queries for the query name and query type, for each authoritative server, as in §3.2. Figure 7 shows the top 50 resolvers that send queries to one of the authoritative servers for A records of `ns.sub.cachetest.net`. We see a large variation in behavior. The resolver with most queries (at $x = 1$) sends a query every 13 ms, with 858k queries during the looping phase. Resolvers ranked 7 to 19 loop every second. Finally, resolvers ranked 20 to 50 all belong to Google and behave similarly, sending queries with a median interarrival of 2.9 s. Taken together Google resolvers are responsible for most of the queries (see Figure 6), but they are not the most aggressive individually.

Recurrent queries: New queries from clients ("recurrent queries") make the problem worse. We run another experiment using RIPE

	Atlas VPs						**Sinkholed**
Measurement	New Domain		Recurrent		TripleDep	CNAME	NA
Frequency	once		1800s		Once	Once	NA
Qname	`$PID.sub.verfwinkel.net.`		`$PID-R.vuur.verfwinkel.net.`		`$PID-R.jupiter.essedarius.net.`	`minuano.essedarius.net.`	Multiple
Query Type	A		A		A	A	Multiple
Date	2020-06-08		2020-06-[09,10]		2021-04-13	2021-04-13	2020-11-18
Duration	6h		17h		3h	3h	22h
	Client Side						
Atlas Probes	9724		9382		9622	9646	NA
VPs	16892		13030		13504	17638	NA
Queries	18715		727778		13579	17736	NA
Responses	18715		727778		13579	17736	NA
SERVFAIL	12585		482470		13437	8841	NA
Timeout	5969		238864		0	0	NA
REFUSED	103		4240		114	97	NA
FORMERR	28		1084		0	13	NA
NOERROR	22		95		15	8791	NA
NXDOMAIN	8		162		0	7	NA
NO ANSWER						11507	
	Authoritative Server Side						
	ns1	ns2	ns1	ns2	4 combined	4 combined	4 combined
Querying IPs	11195	11572	16127	16328	17272	10730	41433
ASes	2587	2611	2408	2446	2554	2412	3615
Queries	4064870	4080446	35546101	36917334	5349129	156356	110282443
Responses	4064801	4070035	35546090	36917300	5349129	156356	110282443

Table 3: TsuNAME Emulation Experiments. Datasets: [46].

.

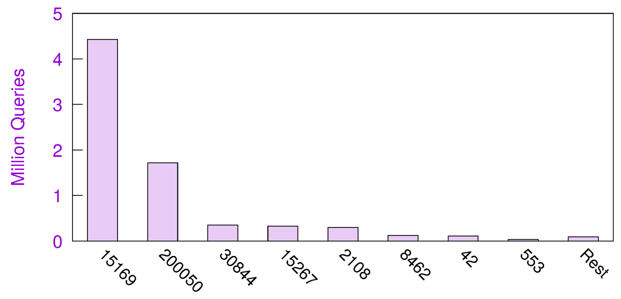


Figure 6: New domain: queries per AS with problematic resolvers

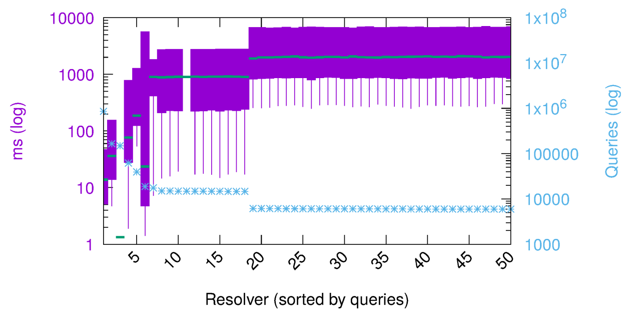


Figure 7: New domain: IQR and queries for A records of `ns.sub.cachetest.net`

	Queries	**Resolvers**	**ASes**
New domain	7.5M	574	37
Recurrent	30.6M	1423	192
Sinkhole	18.1M	2652	127
Unique	56.2M	3696	261

Table 4: Problematic Resolvers found on experiments

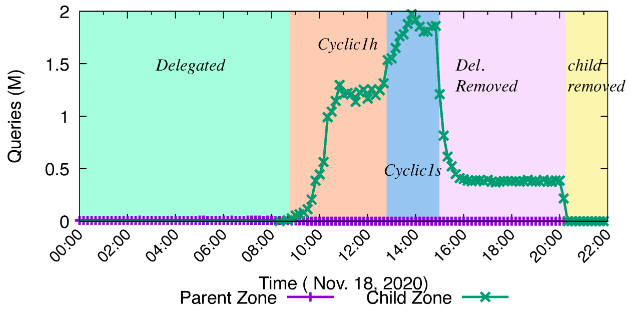
Atlas, but with probes sending new queries every 10 minutes ("Recurrent" column in Table 3). Details are in Appendix B, but fresh queries from clients recapitulate the initial influx of queries for recursive resolvers that do not have a negative cache for the cycle. We find a total of 1423 vulnerable resolvers, from 192 ASes (Table 4), far more than with the New Domain experiment.

4.2 Broader Clients and Recursives

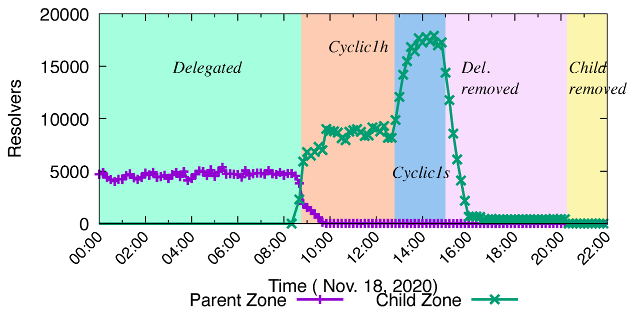
Experiments with RIPE Atlas see a specific set of locations and recursive resolvers. To reproduce this experiment with a broader set of clients and recursives, we carry out a similar experiment with a *sinkholed domain.*

Botnets are compromised end-user computers that are controlled by the botmaster through a command-and-control (C&C) server at some DNS name. A sinkholed domain is a C&C server that has been taken over by the authorities from the botmaster. Since end-user computers cannot be de-infected, the sinkhole domain continues to receive traffic. Typically, such traffic is discarded, but we were given approval to use this traffic to evaluate cyclic domains.

Our sinkholed domain (`bad-domain.nl`) has two subdomains under it (`{a,b}.`**`bad-domain`**`.nl`) that are still frequently queried, even



(a) Queries



(b) Unique Resolvers

Figure 8: Sinkhole experiment timeseries (10min bin)

after more than 8 years since it has been sinkholed. We can configure these domains with a cycle to see how the botnet and the recursives it uses react. We then compare the results to our Atlas experiments. (We make no changes to the bots, nor cause any harm to them. The only traffic they send is their ongoing DNS queries.)

Our experiment is divided into multiple phases, as shown in Table 5: the *Pre*, *Delegated*, *Cyclic dependent*, *TTL influence*, and the *Delegation removed* phases.

Pre phase: Before (and after) our experiment the sinkhole answers DNS queries with "localhost" with a 1 h TTL, causing bots to talk to themselves when searching for C&C. With this typical sinkhole operation the bots do no harm to themselves or others.

Delegated phase: We next change the NS records of the sinkholed domain to our experiment's authoritative servers. We can now observe bot queries to determine the baseline query traffic and number of bots. Our servers continue to answer with the localhost address, as in the Pre phase.

We run in Delegated for over 8 hours to ensure any cached DNS records learn about this delegation. We see 344k queries from 29k unique resolver IP addresses during this entire phase (Table 6). We see 10k queries from ~5k resolvers every 10 minutes (Figure 8), defining our baseline query rate.

Cyclic dependent phase: We next introduce a cyclic dependency between our two domains by deploying cyclic NS records (auth.sub2.essedarius.net ↔ auth.sub2.verfwinkel.net), at 08:48 UTC. This configuration reproduces the setup that took place during .nz event (§3). We leave this configuration active for 4 h, four times longer than record's TTL.

Traffic Growth: After adding the cyclic dependency, we see that, around 11:00 UTC, the authoritative servers experience surge in traffic, peaking at 1.2M queries, 120× the original query rate of 10k queries per ten minutes. The new average query rate is 2581 queries per second (q/s), up from 10.8 q/s, a 239× increase. This experiment *confirms large query amplification (239×) due to TsuNAME* with a population independent from RIPE Atlas.

In addition to growth in query rate, we see *more unique resolvers* in Cyclic dependent phase. In Figure 8b, we see an average of 5k unique resolver IPs in the Pre and Delegated Phases, and ~9k at 11:00 UTC, almost twice as many. Prior work has seen similar use of new recursives during DDoS attacks (Appendix F in [34, 35]). We show here a similar diversity is discovered from cyclic dependencies as clients or intermediate recursives try others when initial queries timeout or fail. Analyzing the entire phase, we see that there were 33.9k active resolvers, from 29.2k in the Delegated phase (Table 6).

TTL influence phase: Next we evaluate the role of this TTL value, by reducing the NS record TTL from 1 h to 1 s while keeping the cyclic dependency in place (Table 6). TTL largely disables caching in resolvers (most resolvers respect non-zero TTLs [34]).

A smaller TTL further increases in query volume, from an average of 2.5kq/s with a 1 h TTL to to 5.6kq/ with a 1 s TTL, as shown in Table 6. Relative to the 10q/s baseline in the Delegated phase, this is 526× the traffic. We speculate that this short TTL causes more rapid use of new resolvers, making the problem even worse.

Delegation removed phase: Finally, we stop the experiment at 20:18 UTC, terminating our authoritative servers and reverting the NS records back to those from the Pre phase. After this, all clients send their queries to the original servers, which we do not monitor. We cover its details in Appendix C.

4.2.1 Resolvers amplification factor: This experiment allows us to estimate the amplification factor for each resolver: how many queries does it send during Cyclic dependent phase compared to Delegated phase. In total, we see ~ 22k resolvers querying for A records during both the Pre and Cyclic dependent phases, and ~ 14k for AAAA records.

Figure 9 shows the CDF of both A and AAAA traffic growth. We see that most resolvers indeed send more queries during the Cyclic dependent and TTL influence phases (comparing the median value of AAAA records, a 1 s TTL results in 100× amplification, while 1 h TTL is only about 4×). For both A and AAAA records, roughly 20% of resolvers have at least a 100x growth (x>100) during Cyclic dependent phase and a 40% during TTL influence phase. Finally, while the amplification factor varies by 3 orders of magnitude, even a few resolvers with large amplification (more than 10) can stress authoritative servers.

4.2.2 Identifying problematic resolvers. Finally, we look for problematic resolvers in this data—the resolvers that sent too many queries. Given we cannot control user query frequency, we are unable to simply use the IQR of received queries like with RIPE Atlas (§4.1). Instead, we use the traffic growth observed by the *matching resolvers*.

We start first by singling out resolvers with a normalized amplification factor of at least 100 – an arbitrarily chosen number.

	Parent Zone (`bad-domain.nl`)				*Child Zone* (`a.bad-domain.nl`)			
Phase	**A**	**AAAA**	**NS**	**TTL**	**A**	**AAAA**	**NS**	**TTL**
Pre	127.0.0.1	::1	–	3600	–	–	–	–
Delegated	–	–	`ns-414.awsdns-51.com`	3600	127.0.0.1	::1		3600
Cyclic1h	–	–	`ns-414.awsdns-51.com`	3600	–	–	`auth.sub2.essedarius.net`	3600
Cyclic1s	–	–	`ns-414.awsdns-51.com`	3600	–	–	`auth.sub2.essedarius.net`	1
Del. removed	127.0.0.1	::1	–	3600	–	–	`auth.sub2.essedarius.net`	1
Child removed	127.0.0.1	::1	–	3600	–	–	–	–

Table 5: Sinkhole experiment: zone file for `a.bad-domain.nl`, at parent and child delegation

	Delegated	**Cyclic1h**	**Cyclic1s**	**Del. removed**
Queries	344222	37173567	45054434	27386057
Queries/s	10.8	2581.4	5688.6	1444.4
Resolvers	29280	33943	33679	18920
∩ Del. A	–	22541	22239	18566
∩ Del.AAAA	–	14125	14411	15696
ASes	2490	2620	2785	1228
Duration	8h48min	4h	2h12min	5h16min

Table 6: Phase classification and characteristics

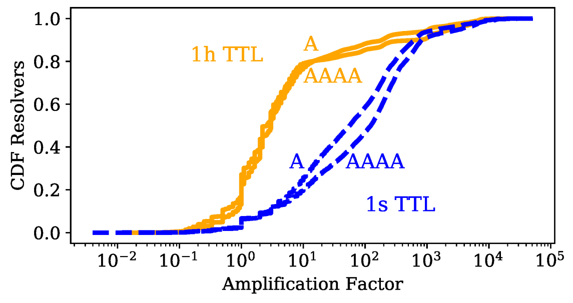


Figure 9: CDF resolvers by amplification factor

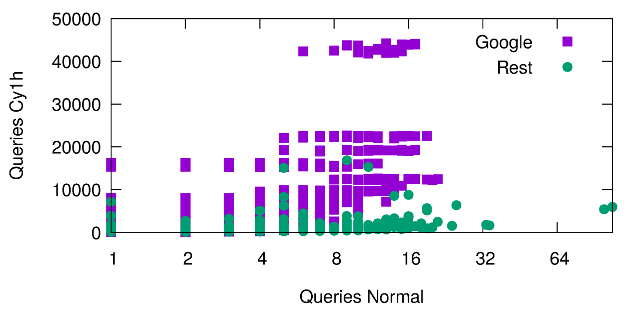


Figure 10: Resolvers with at least 100x traffic growth (A queries)

Authoritative		Ripe Atlas	
IPs	11931	IPs	7317
not querying $PID	4642	Private	2924
querying $PID	7289	Public	4393
> 1 $PID	1558		
== 1 $PID	5731		
Matching Resolver IPs			1256
Problematic Resolvers			574
intersection (∩)			4

Table 7: Singling out looping resolvers

Out of the 22451 active resolvers during both the Pre and Cyclic dependent phases, 2653 and 2076 resolvers sent at least 100x more A and AAAA records respectively (Figure 9).

Figure 10 shows the matching resolvers and the total queries sent for A queries (we show the AAAA queries in Appendix A) We see that both the most aggressive and the most resolvers are from Google: only Google sent more than 20k queries per resolver during Cyclic dependent phase. However, we also see in Table 4 that 2652 resolvers from 127 ASes were also vulnerable to TsuNAME.

4.3 Longer cycles and CNAME cycles

Most of our experiment focuses on cycles of two NS records, but there can be cycles of more NS records ($A \rightarrow B \rightarrow C \rightarrow A$) and cycles using CNAMEs. We carried out experiments on these cases. We found multiple-hop NS cycles further increase the amplification factor (possibly linearly), but CNAME cycles do not. Due to space constraints, details are in Appendix D.

4.4 Finding Problematic Recursives

Events (§3) and experiments (§4.1 and §4.2) show the problem of TsuNAME, but they treat the recursives as an opaque box. We next open that box up to find *problematic recursives*.

To identify problematic resolvers we want to find *one-hop clients*—cases where a user queries a recursive resolvers that queries our authoritative servers. These clients allow us to identify the amplification from a single user query, and to demonstrate looping recursives.

We walk authoritative queries back to clients by examining queries from thousands of RIPE Atlas probes through their recursive resolvers (Figure 4). We ask each RIPE Atlas probe to query our new domain. (These queries are not affected by caching or other users). We see 7,317 unique resolver IPs across 9,724 probes (Table 7). Of these, 4,393 resolvers use routable IP addresses, and the rest are in private networks [45], presumably behind a NAT.

At the authoritative servers, we see queries arrive from 11,931 unique IP addresses (Table 7). Some of these sources are shared by multiple clients. To avoid inflating our query counts we identify and discard shared recursive resolvers. We can distinguish unique from shared resolvers since queries include the resolvers ID (each is for `$PID.sub.verfwinkel.net`, where $PID is the Atlas probe's ID). We find 5,731 non-shared recursives, each serving a single Atlas probe (== 1 $PID).

Resolver	Queries	Median Δt	Duration
r1	34351	901 ms	3.17h
r2	2783	6095 ms	5.13h
r3	833	60821 ms	5.10h
r4	775	61680 ms	5.10h

Table 8: Confirmed looping resolvers

		RIPE Atlas Side	
#	Time	Query/Type	Resolver
1	14:14:57	`19817.sub.verfwinkel.net/A`	IP1
2	14:15:03	`19817.sub.verfwinkel.net/A`	IP2
3	14:15:09	`19817.sub.verfwinkel.net/A`	IP3
		Authoritative Server Side from IP2	
4	14:15:07	`19817.sub.verfwinkel.net/A`	IP2
5	14:15:10	`ns.sub.cachetest.net/A`	IP2
6	14:15:10	`ns.sub.verfwinkel.net/A`	IP2
		Remaining queries	
Median Δt		Query/Type	Total
901ms		`ns.sub.verfwinkel.net/A`	17905
901ms		`ns.sub.cachetest.net/A`	18169

Table 9: Query sequence for Probe 19817

We can now find one-hop clients (client to recursive to authoritative, without forwarders or multi-level recursives). We identify one-hop clients as Atlas VPs where their local resolver's IP address matches an IP address seen in traffic to our authoritative servers—the cells highlighted in Table 7. This process may miss recursive resolvers that serve DNS traffic on multiple network interfaces. We intersect these lists to identify 1,256 matching IP addresses.

We compare these recursives against our list of 574 problematic resolvers (from controlled experiments §4.1). We find that 4 one-hop clients use problematic resolvers. Table 8 lists these anonymized recursive resolvers and shows how many queries and their query interarrival rate from our prior experiment. We see that the most prolific, $r1$, queried every second for more than 3 hours, while the others queried every 6 or 60 s.

Resolver query history: We next examine query sequence of these problematic recursives to understand when they start looping. We begin with $r1$ from AS553 (BelWue, Germany), which sent 36k queries. Table 9 shows its query history. First, at the Atlas side, we see that this Atlas probe is configured with 3 different resolvers (IP1–IP3, omitted for privacy), and it sends one query per resolver. No other queries are issued from Atlas after that.

We see that these resolvers produce different results at the authoritative side. IP1 sends only 9 queries to our servers, and IP3 sent 1 query only. But IP2, on the other hand, sends 36,075 queries! After queries #5 and #6, this resolver repeats these queries every 900 ms for more than 3 h (Table 8). Even without new client queries, this recursive resolver seems stuck in an infinite cycle. We contacted this operator and they reported that this recursive ran Windows 2008R2 and was marked to be phased out.

We have shown that we can identify specific problematic resolvers, and proven that at least one specific resolver will cycle indefinitely. While the operator is resolving this case, it shows TsuNAME is a problem today. We examine $r2$ (Table 8) is in Appendix H (omitted here due to space).

4.5 Revisiting Google Public DNS

We have seen that Google Public DNS was responsible for most queries during both the `.nz` event (§3) and our experiments (§4). Given how much traffic they handle, their role is not surprising. Oddly, however, they show diverse behavior: some GDNS resolvers seem to loop while others do not (Figure 3). We reached out to them and other operators (§6), following responsible disclosure guidelines. Our interaction with Google helped us understand how the different components of the DNS system come together to make TsuNAME amplification problematic.

Based on our input, Google engineers reproduced the problem. They could not reproduce continuous looping, but did show an amplification factor of 10 from retries. While a powerful amplification, this behavior conforms to limitations required by [25, 27].

But if GDNS does not loop by itself, why does it contribute large traffic volumes during the `.nz` event and in our experiments? Google engineers found it was a combination of two factors: first, retries occur *outside* Google's recursive resolvers: GDNS clients themselves can retry (for example, $R1_b$ in Figure 4). Secondly, Google's recursive resolver system did not cache the cyclic failure. Thus every new external query would force Google's resolvers to re-prove the cycle, amplifying each external query by ten. This *interaction* of DNS system components results in large traffic volumes.

This interaction also explains why we see only some Google IP addresses during the `.nz` event (Figure 3): the clients retrying queries produced much more traffic than those where clients do minimal retries.

Google then fixed GDNS by implemeting negative caching of cyclic dependent records. After Google reported their fix to us, we confirmed they have mitigated the they TsuNAME vulnerability in their service. When we repeated our experiments (§4.1 and Appendix B), wee see a much lower query volume (Appendix E). We also thank Google for a bug bounty for reporting this problem; we donated it to Wikipedia.

5 DETECTING CYCLIC DEPENDENCIES

TsuNAME attacks are intrinsically asymmetrical: the victims (authoritative server operators) are different companies than the amplifiers (vulnerable resolver operators). We discuss this threat model in greater detail in Appendix F.

Next consider the side of the authoritative server operator, and work on *preventing* TsuNAME attacks by detecting and removing cyclic dependencies from their zones. We present `CycleHunter`, a tool that we developed that *proactively* detects cyclic dependencies in zone files, allowing operators to identify them *before* any vulnerable resolvers do. We make `CycleHunter` publicly available at http://tsuname.io and [10].

`CycleHunter` uses active DNS measurements to detect cyclic dependencies, given many NS records in a DNS zone are typically out-of-zone (out-of-bailiwick) [55]. As such, it requires external zone knowledge which can only be done if an operator has every necessary zone file in possession (a condition we do not assume).

5.1 `CycleHunter`

`CycleHunter` begins with a bulk copy of the zone, processing it (Figure 11):

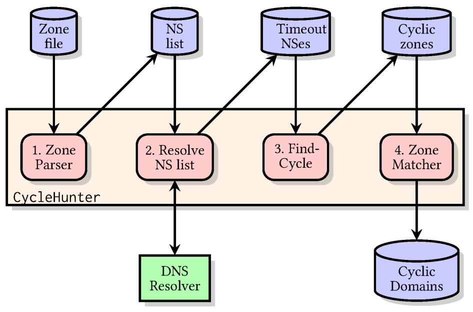


Figure 11: `CycleHunter` workflow

1. Zone Parser: the first module reads a DNS zone file, such as the `.org` zone and extracts records of interest. Zone files contain both delegations and other data records (A, AAAA, NS, SOA, DNSSEC, and so forth). The Zone Parser module extracts the NS records into the "NS List" (Figure 11). `CycleHunter's` goal is to determine *which* of these NS records are cyclically dependent, and what domain names in the zone file use them. Given that many domain names use the same authoritative servers [3, 24], this step significantly reduces the search space. For instance, the `.com` zone has 151M domain names, but only 2.19M unique NS records (Table 10).

2. Resolve NS list: this module tries to resolve every NS record in the NS list. `CycleHunter` uses the local computer's configured resolver (BIND 9 in our experiments). `CycleHunter` retrieves the start-of-authority (SOA) record [27] required of each zone, for each NS in the NS list. Successful resolution of the domain's SOA proves that domain is *resolvable* and is not cyclically dependent. We are interested in domains that *fail* this test, as these reveal which domains have cyclically dependent NS records that are not resolvable.

3. Find Cycle: Since NS records fail for cycles, but also other reasons (missing data, missing or "lame" servers [26], packet loss, etc.), we next identify cycles.

To detect cycles, it first records each NS's Authority information [27] (an Authority object in the `NS list`). For example, suppose that `ns0.wikimedia.org` was in the NS list (Figure 12). This module will create an `Authority` object for this NS record, which will include its parent zone `wikimedia.org` and its NS records (`wikimedia.org: [ns1,ns2].example,com`). It does that by querying the parent authoritative servers instead of the unresponsive, possibly cyclic, NS records. For the `ns0.wikimedia.org` example, it retrieves the authority data for `wikimedia.org` directly from `.org`'s authoritative servers.

To detect cycles, we examine each NS record in the NS list, identifying the parent zone and its NS records. We get this information from the parent zone of the NS record. For example, if `ns0.wikimedia.org` is in the NS list, we would check NS records for `wikimedia.org`, perhaps finding `ns1.example.com` and ns2.example .com. This query to the parent often succeeds even if the ns0.wiki media.org is cyclic.

Then the *Find Cycle* module determines what *zones* this Authority zone depends on by analyzing its own NS records. In our fictional Figure 12 example, we see that `wikimedia.org` depends on `example.com` so we also need to create an authority object for `example.com` and determine what zones it depends on. The last step consists of comparing these two authority objects: the `example.com`

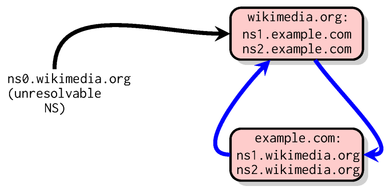


Figure 12: `CycleHunter` Cyclic Dependency Detector

zone	Size	NSSet	Cyclic	Affect.	Date
.com	151445463	2199652	21	1233	2020-12-05
.net	13444518	708837	6	17	2020-12-10
.org	10797217	540819	13	121	2020-12-10
.nl	6072961	79619	4	64	2020-12-03
.se	1655434	27540	0	0	2020-12-10
.nz	718254	35738	0	0	2021-01-11
.nu	274018	10519	0	0	2020-12-10
Root	1506	115	0	0	2020-12-04
Total	184409371	3602839	44	1435	

Table 10: `CycleHunter`: evaluated DNS Zones

NS records that depend on `wikimedia.org`, which in turn depends on `example.com`, confirming a cyclic dependency between `wikimedia.org` and `example.com`.

Note that although we start with one zone, we track all records in other zones that are referenced.

`CycleHunter` can also detect other dependency types. For example, if the `wikimedia.org` zone has an in-zone NS record (ns3.wikime dia.org), but with an unresponsive or lame or missing glue record, `CycleHunter` will classify this zone as cyclic dependent with an in-zone NS record ("`fullDepWithInZone`").

Zone Matcher: the last `CycleHunter` module identifies which domain names in the DNS zone are requiring cyclic dependent NS records identified by the *Find Cycle* module. For example, ns0.wi kipedia.org could be the authoritative server for both `dog.org` and `cat.org`.

Performance: `CycleHunter` is a concurrent and asynchronous application that allows a user configurable number of threads/workers. However, the largest performance bottleneck is in the *resolver* used in Step 2. As such, we recommend operators use a high performance resolver for faster results. To ensure accuracy, `CycleHunter` should always start after the resolver's cache has been cleared in order to retrieve the most up to date DNS records. With 8 CPU cores we can evaluate 10M domains per hour, and this work can parallelizes with more cores.

Limitations: Currently `CycleHunter` detects cycles of length two. We can detect longer cycles (see §4.3) and plan to extend it to cover them. Fortunately, mitigations at other stages (see §2) help for cycles that are not discovered.

5.2 Evaluation of DNS Zones

We use `CycleHunter` to evaluate the DNS Root zone and 7 TLDs which are either public [20, 21] or available via ICANN CZDS [19]. Table 10 shows the number of domains (size) and the total number of NS records (NSset) per zone.

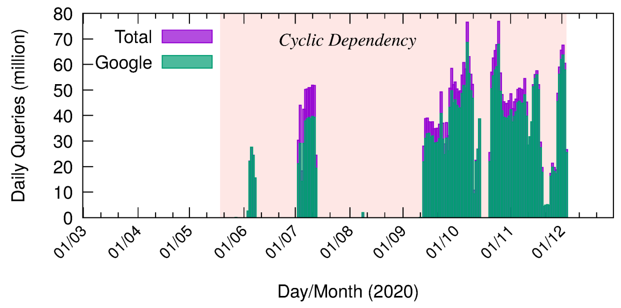


Figure 13: Query timeseries for `.nl` domain with cyclic dependency found with `CycleHunter`

From the evaluated 184M child domains, we obtained 3.6M distinct NS records. After processing, `CycleHunter` finds 44 cyclic-dependent NS records ("Cyclic" in Table 10). We manually verified each of these cases, confirming the cycles. In total, we found 1435 unreachable domain names employing cyclic dependent domains.

This fairly a small number of domains most likely suffer simply configuration errors that render them unresolvable. However, as we cover in Appendix F, adversaries could exploit the TsuNAME vulnerability to incur damage.

5.2.1 Analyzing the `.nl` Cyclic Domains: The 6M domains in `.nl` use 79k NS records (Table 10). `CycleHunter` identified 6 cyclic dependent NSes in the `.nl` zone: 3 were `.nl` domain names, and 2 were within `.com` and 1 within `.net`. Final analysis showed 64 domains that employed these cyclic 6 NS records.

Of the 3 `.nl` domain names, two were our test domains. These were not publicized and receive no more than 1k queries daily. The remaining domain `bugged-example.nl`, however, is an old domain, registered in 2004.

With access to `.nl` authoritative DNS traffic, we use the open-source DNS analytics package ENTRADA [54, 64] to determine the daily queries this cyclic domain received. Figure 13 shows the results. Very few queries were seen until mid-June (<300 daily). However, on May 19, the domain owner changed the NS records of the domain to be cyclic dependent – probably due to human error as was the case of `.nz` (§3). After June 4th, we observe a significant amount more queries: 2.2M, reaching 27M on June 8th. After that point, we see three intervals of large query volumes, each averaging 42M daily queries to this domain. The first interval (July 3rd– July 13th), lasts for 10 days, the second for over a month (Sep. 13th – Oct. 15th), and the last interval covers 43 days (Oct. 21st – Dec. 3rd).

Figure 13 shows also that most of these queries come from Google. To fix that, we notified the domain owner on Dec. 4th, and they quickly fixed their NS settings, which after that, the number of queries reduced to 300 daily (we did this analysis prior to GDNS being repaired – §4.5).

Simply having cyclic dependencies alone does not trigger large volume of queries – our two test cyclic domains have not experienced large query volumes – this only happens in combination with vulnerable resolvers.

5.3 Resolver software evaluation

Google repaired GDNS, significantly reducing the potential for harm. However, as shown in Table 4, Google was not the only affected resolver operator.

To determine how vulnerable current resolver software is to TsuNAME, we create two tests: 1. determine if a resolver loops, given a cyclic dependency (like $r1$ in §4.4) and 2. if the resolver caches these loops (Appendix I).

With a test zone with cyclic dependency (§4), we evaluate popular DNS resolvers on an AWS EC2 (Fra) VM: Unbound (v 1.6.7) [36], BIND (v 9.11.3) [22], and KnotDNS (v 5.1.3) [11]. We find that none of these loop in the presence of cyclic dependent domains, and hence are not vulnerable.

We also evaluated multiple public resolvers, finding that Quad9 [43] and Quad1 [1] were not vulnerable. However, we did find that Cisco's OpenDNS [38] was vulnerable and notified the operators. They fixed the issue on 2021-04-13.

6 RESPONSIBLE DISCLOSURE

We wish to protect zone operators from DDoS attacks using TsuNAME. The problem can be solved by modifying and deploying recursive resolvers (§2), but it takes considerable time. However, operators of an authoritative DNS service can at least protect their servers by ensuring that no sub-domain's of theirs have cyclic dependencies. In the long run, both resolver and domain fixes should be in place, since either new recursive software or new domains with cyclic dependencies can always recreate the problem.

To address these problems, we notified and worked with Google, whose public DNS service represents the largest recursive resolver traffic we see. After notification, they solved this problem in their GDNS (§4.5). We followed best practices for these disclosures, allowing operators at least 90 days to address problems. This threshold is consistent with `cert.org`'s 45-day notification policy [9] and Google Project Zero's 90-day policy [16]. We also notified Cisco OpenDNS, another large public DNS provider, which also fixed their resolver software.

In addition to Google we also notified operators of the other ASes that generated the greatest amount of recursive traffic in our experiments from §4. Three of these ten responded to us. Of those, two reported running old recursive resolver software. One was using a PowerDNS resolver (3.6.2-2, from 2014 [41]), and the other was using Windows 2008R2(§4.4). Both planned to update these resolvers.

6.1 Private and Public Disclosure

We made early, limited disclosure to specific operators, and broader but limited disclosure to the operator and vendor community in February 2021 [29].

Developer reaction. Upon our private disclose, Google and Cisco have fixed their public DNS software. Moreover, the developers of three popular resolver software – BIND, Unbound, and PowerDNS have released public statements about how they mitigate or planned to mitigate TsuNAME– with PowerDNS and Unbound explicitly deploying negative caching of cyclic dependent records [6, 37, 42].

Operator reaction: After our February disclosure, two ccTLD operators reported to us that they experienced TsuNAME events. A European ccTLD kindly shared their experience with us. Around

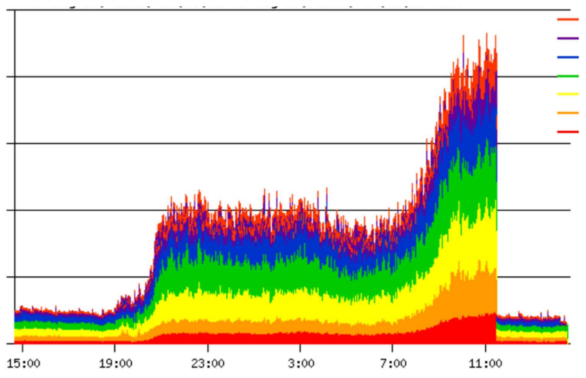


Figure 14: TsuNAME event at an Anonymous EU-based ccTLD operator. Each color represents the traffic to one of the ccTLD authoritative servers.

19:00 UTC one day in 2019, two domains in their zones were misconfigured with cyclic dependencies. Given these domain names were particularly popular in their country, they created 10x their normal traffic. Figure 14 shows a timeseries of queries from this event (y axis anonymized by the operator, each color denoting one authoritative server). The ccTLD operator contacted the domain owner, who fixed it around 11:00 UTC on the following day. Similarly to the .nz event, after fixing an immediate drop is seen in the traffic.

A second large anycast operator confirmed that Google had at least on one occasion sent a surge of queries to their authoritative servers, following a cyclic dependency misconfiguration. Their experience was *worse* than ours: they received a surge in UDP traffic, to which they responded with response rate limiting (RRL [56]). However, because these queries were from legitimate clients (not spoofed), the clients then retried with TCP, further amplifying the problem.

Public Disclosure: On 2021-05-06 we public disclosed TsuNAME with a website, security advisory, and technical report. This paper documents the problem in more detail and enables peer review of the analysis.

7 ETHICS AND PRIVACY

Our paper poses three ethical concerns: avoiding negative consequences in our evaluation, the availability of our collected data, and proper disclosure to impacted operators.

Our evaluation to identify, reproduce, and confirm solutions to TsuNAME placed traffic on both networks and servers. We designed our experiments to minimize this impact. First, our experiments in §4 use third-level domain names, isolating the amplification to authoritative servers under our control. This step ensures that the authoritative servers for TLDs are not affected by the extra traffic from our experiments. The .nz event (§3) shows that individual resolvers send *safe* query volumes (fewer than 10 q/s per resolver), supporting that our experiments will not harm end users or their resolvers. In addition, we monitored the experiment for potential surges. Aggregate traffic during the experiment can harm the authoritative servers, but we monitor and can manage any self-inflicted threat. We see reached at most 5.6k q/s in the sinkhole experiment, well within our capacity.

Secondly, we limit the duration of these experiments (at most 11 hours) to reduce the traffic generated from the tested recursive resolvers to a limited period of repeated queries. Our goal is to reproduce the problem, confirm which resolvers are affected and how they behave – not to evaluate how long they will loop. Finally, we design our experiment to weigh experiment benefit against costs (value-sensitive design [14, 17]). Our benefits are understanding potential harm, allowing us to notify operators, avert future problems by removing the vulnerability. These benefits greatly exceed the carefully managed, modest experimental risks. Changes made by Google and OpenDNS show evidence of our success.

Our sinkhole experiment (§4.2) temporarily introduced cyclic dependencies in the domains, preventing bots from receiving answers related to their C&C sinkholed domain. (We obtained permission from the operator to run this experiment). This change does not impact bots or regular users, but it may cause vulnerable resolvers to query more often than normal, but no more than 1 q/s – a very safe value that is very unlikely to disrupt any resolver's operation. Given our goal is to evaluate the extent of vulnerable resolvers, we reverted the domain to its original configuration after the experiment ended.

Although we often release datasets publicly, in this case we are unable to do so. Disclosure of IP address of vulnerable resolvers and cyclic zones could enable abuse by others, and provide little lasting benefit if problems are resolved. We have provided information privately to operators and developers, with at least 60 days prior to the public disclosure (Appendix G).

8 RELATED WORK

DNS misconfiguration: There have long been reports of many types of DNS misconfiguration [2, 8, 12, 39, 60]. Our work focuses on the problem of cyclic dependencies, first described as loops on RFC1536 [25], and the term was coined by Pappas *et al.* [39]. Our work goes beyond that work to show how this misconfiguration interacts in real networks resulting in some recursive resolvers greatly amplifying traffic, and we provide the first scan for these cyclic dependencies.

DDoS attacks against DNS: Given their role in DNS resolution, loss of authoritative DNS servers due to denial-of-service can have catastrophic consequences. The Mirai Attack against DYN harmed many prominent web services due to loss of DNS [40]. The root DNS has suffered a number of DDoS attacks [32] although no prior attack has affected enough root DNS instances to cause a user-visible outage, in part due to heavy caching [33].

DDoS attacks using DNS: The DNS protocol itself has been exploited to carry out attacks on others. With spoofed queries, DNSSEC provides a vector for attack amplification [23, 59]. Long CNAME chains have also been exploited to carry amplification attacks against authoritative servers [7]: by carefully crafting CNAME chains where each element points to the same authoritative servers, an attacker can amplify query volumes. We analyze short CNAME loops and show that resolvers can detect them.

Shared infrastructure and collateral damage: Previous studies have shown where shared infrastructure in DNS [3, 24] has increased the odds of collateral damage: once a zone is attacked, other zones hosted in the same infrastructure may suffer together – for example, this happened to `.nl` when parts of the Root DNS was attacked [32]. Similar behavior was observed when the Dyn DNS provider was attacked and multiple zones hosted by it had connectivity problems [40]. Using the DNS to protect origin servers behind CDNs has been thwarted by simultaneous connecting through IP addresses only and by using carefully crafted URLs [58].

Recursive and Public Resolvers: Studies have examined the recursive DNS resolution infrastructure, and inferred the internals of Google's public DNS service, both as a subject of study [53], to understand end-user behavior [13], and to better understand caches [44]. Google DNS has been found to be one of the most popular public DNS resolvers [30]. In the same study, the authors showed one of the benefits of centralization: when Google adopted the privacy protecting query-name minimization technique, it benefited many users at the same time. Our study shows another negative side of this coin, showing that when something breaks in large DNS providers, it can be exploited to cause significant harm.

9 CONCLUSIONS

The risk of DNS cycles has been documented for more than 30 years, but this threat has been underestimated. Large traffic increases at `.nz` (§3) prompted us to carefully investigate this problem through controlled experiments (§4). Although the exact amplification factor varies, we showed factors ranging from 120× and 526×, explaining two weeks of 50% growth at `.nz` and a 10× increase in a European ccTLD.

Our contribution is to document the cause and threat of the problem (§2), and to disclose this information to the vendor and operation community (§6). We also provide a tool to detect existing cyclic dependencies (§5). We hope that these steps will defuse this problem, and we are happy that multiple vendors have confirmed current software mitigates the problem (and identified old versions at risk), and important operators such as Google Public DNS have taken steps to manage the challenge.

Acknowledgments

We thank the anonymous IMC reviewers and particularly the shepherd for their time detailed suggestions.

We would like to thank Marco Davids for his suppport and help in this research. We also would like to thank Puneet Sood, Warren Kumari, Brian Dickson, Barry Greene, Keith Mitchell, Denesh Bhabuta, Benno Overeinder, Dario Ciccarone, Ralph Weber and the DNS-OARC community for their feedback and support in handling this vulnerability disclosure. We thank the Anonymous European ccTLD operator for kindly sharing their experience and their traffic graph when they experienced a TsuNAME event (Figure 14).

We thank all contributors to `CycleHunter` on Github: https://github.com/SIDN/CycleHunter/graphs/contributors

We also thank the Ripe Atlas measurement platform [47, 48], which we used extensively in the characterization of this vulnerability in [31].

The work of Giovane C. M. Moura is partially funded by the European Union's Horizon 2020 CONCORDIA project (Grant Agreement 830927) and by the Plannning for Anycast as Anti-DDoS research project (PAADDoS), under the research programme Cyber Security (project number 628.001.029), which is funded by the Dutch Research Council (NWO).

Wes Hardaker and John Heidemann are partially supporte partially supported by the project "CICI: RSARC: DDoS Defense In Depth for DNS" (NSF OAC-1739034) and "DNS, Identity, and Internet Naming for Experimentation and Research (DIINER)" (NSF CNS-1925737).

REFERENCES

[1] 1.1.1.1. 2018. The Internet's Fastest, Privacy-First DNS Resolver. https://1.1.1.1/. https://1.1.1.1/
[2] Gautam Akiwate, Mattijs Jonker, Raffaele Sommese, Ian Foster, Geoffrey M. Voelker, Stefan Savage, and KC Claffy. 2020. Unresolved Issues: Prevalence, Persistence, and Perils of Lame Delegations. In *Proceedings of the ACM Internet Measurement Conference* (Virtual Event, USA) *(IMC '20)*. Association for Computing Machinery, New York, NY, USA, 281–294. https://doi.org/10.1145/3419394.3423623
[3] Mark Allman. 2018. Comments on DNS Robustness. In *Proceedings of the Internet Measurement Conference 2018* (Boston, MA, USA) *(IMC '18)*. Association for Computing Machinery, New York, NY, USA, 84–90. https://doi.org/10.1145/3278532.3278541
[4] M. Andrews. 1998. *Negative Caching of DNS Queries (DNS NCACHE)*. RFC 2308. IETF. http://tools.ietf.org/rfc/rfc2308.txt
[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *Proceedings of the 26th USENIX Security Symposium*. USENIX, Vancouver, BC, Canada, 1093–1110. https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf
[6] ISC BIND. 2021. TsuNAME DNS Vulnerability and BIND 9. https://www.isc.org/blogs/2021_tsuname_vulnerability/.
[7] Jonas Bushart and Christian Rossow. 2018. DNS Unchained: Amplified Application-Layer DoS Attacks Against DNS Authoritatives. In *Research in Attacks, Intrusions, and Defenses*, Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis (Eds.). Springer International Publishing, Cham, 139–160.
[8] Sebastian Castro, Duane Wessels, Marina Fomenkov, and Kimberly Claffy. 2008. A day at the root of the Internet. *ACM Computer Communication Review* 38, 5 (Oct. 2008), 41–46. https://doi.org/10.1145/1452335.1452341
[9] cert.gov. 2021. Vulnerability Disclosure Policy. https://vuls.cert.org/confluence/display/Wiki/Vulnerability+Disclosure+Policy.
[10] CycleHunter. 2021. GitHub - SIDN/CycleHunter: Python software that reads zone files, extract NS records, and detect cyclic dependencies. https://github.com/SIDN/CycleHunter.
[11] CZ-NIC. 2021. Knot DNS. https://www.knot-dns.cz/
[12] Peter B. Danzig, Katia Obraczka, and Anant Kumar. 1992. An Analysis of Wide-Area Name Server Traffic: A study of the Domain Name System. In *Proceedings of the ACM SIGCOMM Conference* (johnh: folder: networking/dns). ACM, Baltimore, Mayrland, USA, 281–292. https://doi.org/10.1145/144191.144301
[13] Wouter B. De Vries, Roland Van Rijswijk-Deij, Pieter Tjerk De Boer, and Aiko Pras. 2018. Passive Observations of a Large DNS Service: 2.5 Years in the Life of Google. In *2018 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, United States. https://doi.org/10.23919/TMA.2018.8506536
[14] Batya Friedman, David G. Hendry, and Alan Borning. 2017. A Survey of Value Sensitive Design Methods. *Foundations and Trends® in Human–Computer Interaction* 11, 2 (2017), 63–125. https://doi.org/10.1561/1100000015
[15] Google. 2020. Public DNS. https://developers.google.com/speed/public-dns/. https://developers.google.com/speed/public-dns/
[16] Google Project Zero. 2021. Vulnerability Disclosure FAQ. https://googleprojectzero.blogspot.com/p/vulnerability-disclosure-faq.html.
[17] Kenneth Einar Himma, Herman T Tavani, et al. 2008. *The handbook of information and computer ethics*. Wiley Online Library.
[18] P. Hoffman, A. Sullivan, and K. Fujiwara. 2018. *DNS Terminology*. RFC 8499. IETF. http://tools.ietf.org/rfc/rfc8499.txt
[19] ICANN. 2020. Centralized Zone Data Service. https://czds.icann.org/.
[20] Internet Assigned Numbers Authority (IANA). 2020. Root Files. https://www.iana.org/domains/root/files.
[21] Internetstiftelsen. 2020. Zone Data. https://zonedata.iis.se/.
[22] ISC . 2021. BIND 9 . https://www.isc.org/bind/.
[23] Georgios Kambourakis, Tassos Moschos, Dimitris Geneiatakis, and Stefanos Gritzalis. 2007. A Fair Solution to DNS Amplification Attacks. In *Proceedings of the Second IEEE International Workshop on Digital Forensics and Incident Analysis (WDFIA)*. IEEE, 38–47. https://doi.org/10.1109/WDFIA.2007.4299371

[24] Aqsa Kashaf, Vyas Sekar, and Yuvraj Agarwal. 2020. Analyzing Third Party Service Dependencies in Modern Web Services: Have We Learned from the Mirai-Dyn Incident?. In *Proceedings of the ACM Internet Measurement Conference* (Virtual Event, USA) *(IMC '20)*. Association for Computing Machinery, New York, NY, USA, 634–647. https://doi.org/10.1145/3419394.3423664

[25] A. Kumar, J. Postel, C. Neuman, P. Danzig, and S. Miller. 1993. *Common DNS Implementation Errors and Suggested Fixes.* RFC 1536. IETF. http://tools.ietf.org/rfc/rfc1536.txt

[26] M. Larson and P. Barber. 2006. *Observed DNS Resolution Misbehavior.* RFC 4697. IETF. http://tools.ietf.org/rfc/rfc4697.txt

[27] P.V. Mockapetris. 1987. *Domain names - concepts and facilities.* RFC 1034. IETF. http://tools.ietf.org/rfc/rfc1034.txt

[28] P.V. Mockapetris. 1987. *Domain names - implementation and specification.* RFC 1035. IETF. http://tools.ietf.org/rfc/rfc1035.txt

[29] Giovane C. M. Moura. 2021. OARC Members Only Session: Vulnerability Disclosure (DDoS). https://indico.dns-oarc.net/event/37/contributions/821/. https://indico.dns-oarc.net/event/37/contributions/821/

[30] Giovane C. M. Moura, Sebastian Castro, Wes Hardaker, Maarten Wullink, and Cristian Hesselman. 2020. Clouding up the Internet: How Centralized is DNS Traffic Becoming?. In *Proceedings of the ACM Internet Measurement Conference* (Virtual Event, USA) *(IMC '20)*. Association for Computing Machinery, New York, NY, USA, 42–49.

[31] Giovane C. M. Moura, Sebastian Castro, John Heidemann, and Wes Hardaker. 2021. *tsuNAME: exploiting misconfiguration and vulnerability to DDoS DNS.* Technical Report 2021-01. SIDN Labs. https://tsuname.io/tech_report.pdf. https://doi.org/paper.pdf

[32] Giovane C. M. Moura, Ricardo de O. Schmidt, John Heidemann, Wouter B. de Vries, Moritz Müller, Lan Wei, and Christian Hesselman. 2016. Anycast vs. DDoS: Evaluating the November 2015 Root DNS Event. In *Proceedings of the ACM Internet Measurement Conference.* ACM, Santa Monica, California, USA, 255–270. https://doi.org/10.1145/2987443.2987446

[33] Giovane C. M. Moura, John Heidemann, Ricardo de O. Schmidt, and Wes Hardaker. 2019. Cache Me If You Can: Effects of DNS Time-to-Live. In *Proceedings of the ACM Internet Measurement Conference.* ACM, Amsterdam, the Netherlands, 101–115. https://doi.org/10.1145/3355369.3355568

[34] Giovane C. M. Moura, John Heidemann, Moritz Müller, Ricardo de O. Schmidt, and Marco Davids. 2018. When the Dike Breaks: Dissecting DNS Defenses During DDoS. In *Proceedings of the ACM Internet Measurement Conference.* ACM, Boston, MA, USA, 8–21. https://doi.org/10.1145/3278532.3278534

[35] Giovane C. M. Moura, John Heidemann, Moritz Müller, Ricardo de O. Schmidt, and Marco Davids. 2018. *When the Dike Breaks: Dissecting DNS Defenses During DDoS (extended).* Technical Report ISI-TR-725. USC/Information Sciences Institute. https://www.isi.edu/%7ejohnh/PAPERS/Moura18a.html

[36] NL Netlabs. 2021. UNBOUND. https://www.nlnetlabs.nl/projects/unbound/about/.

[37] NLnetLabs. 2021. tsuNAME vulnerability and Unbound. https://nlnetlabs.nl/news/2021/May/10/tsuname-vulnerability-and-unbound/.

[38] OpenDNS. 2021. Setup Guide: OpenDNS. https://www.opendns.com/. https://www.opendns.com/

[39] Vasileios Pappas, Zhiguo Xu, Songwu Lu, Daniel Massey, Andreas Terzis, and Lixia Zhang. 2004. Impact of Configuration Errors on DNS Robustness. *SIGCOMM Comput. Commun. Rev.* 34, 4 (Aug. 2004), 319–330. https://doi.org/10.1145/1030194.1015503

[40] Nicole Perlroth. 2016. Hackers Used New Weapons to Disrupt Major Websites Across U.S. *New York Times* (Oct. 22 2016), A1. http://www.nytimes.com/2016/10/22/business/internet-problems-attack.html

[41] PowerDNS. 2021. Changelogs for all pre 4.0 releases. https://doc.powerdns.com/recursor/changelog/pre-4.0.html.

[42] PowerDNS. 2021. TsuNAME vulnerability and PowerDNS Recursor. https://blog.powerdns.com/2021/05/10/tsuname-vulnerability-and-powerdns-recursor/.

[43] Quad9. 2018. Quad9 | Internet Security & Privacy In a Few Easy Steps. https://quad9.net.

[44] Audrey Randall, Enze Liu, Gautam Akiwate, Ramakrishna Padmanabhan, Geoffrey M. Voelker, Stefan Savage, and Aaron Schulman. 2020. Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers. In *Proceedings of the ACM Internet Measurement Conference* (Virtual Event, USA) *(IMC '20)*. Association for Computing Machinery, New York, NY, USA, 50–64. https://doi.org/10.1145/3419394.3423640

[45] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. 1996. *Address Allocation for Private Internets.* RFC 1918. IETF. http://tools.ietf.org/rfc/rfc1918.txt

[46] RIPE NCC. 2021. RIPE Atlas Measurement IDS. https://atlas.ripe.net/measurements/ID. , where ID is the experiment ID: New Domain:25666966, Recurrent:25683316 , One-off-AfterGoogle: 29078085, RecurrentAfterGoogle: 29099244, probe52196:29491104, TripeDep:29559226, CNAME: 29560025.

[47] RIPE NCC Staff. 2015. RIPE Atlas: A Global Internet Measurement Network. *Internet Protocol Journal (IPJ)* 18, 3 (Sep 2015), 2–26.

[48] RIPE Network Coordination Centre. 2020. RIPE Atlas. https://atlas.ripe.net.

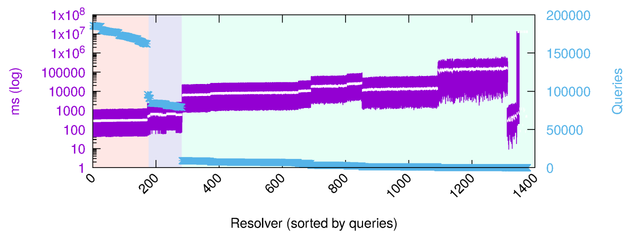


Figure 15: Google (AS15169) resolvers on 2020-02-06, during .nz TsuNAME event: time in between queries for AAAA queries

[49] RIPE Network Coordination Centre. 2020. RIPE Atlas - Raw data structure documentations,https://atlas.ripe.net/docs/data_struct/.

[50] Root Server Operators. 2015. Events of 2015-11-30. http://root-servers.org/news/events-of-20151130.txt.

[51] Root Server Operators. 2020. Root DNS. http://root-servers.org/.

[52] Root Zone file. 2020. Root. http://www.internic.net/domain/root.zone.

[53] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. 2013. On measuring the client-side DNS infrastructure. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference.* ACM, 77–90.

[54] SIDN Labs. 2020. ENTRADA - DNS Big Data Analytics. https://entrada.sidnlabs.nl/.

[55] Raffaele Sommese, Leandro Bertholdo, Gautam Akiwate, Mattijs Jonker, van Rijswijk-Deij, Roland, Alberto Dainotti, KC Claffy, and Anna Sperotto. 2020. MAnycast2—Using Anycast to Measure Anycast. In *Proceedings of the ACM Internet Measurement Conference.* ACM, Pittsburgh, PA, USA. https://doi.org/10.1145/3419394.3423646

[56] Suzanne Goldlust. 2018. Using the Response Rate Limiting Feature. https://kb.isc.org/docs/aa-00994.

[57] S. Thomson, C. Huitema, V. Ksinant, and M. Souissi. 2003. *DNS Extensions to Support IP Version 6.* RFC 3596. IETF. http://tools.ietf.org/rfc/rfc3596.txt

[58] Sipat Triukose, Zakaria Al-Qudah, and Michael Rabinovich. 2009. Content Delivery Networks: Protection or Threat?. In *Computer Security – ESORICS 2009*, Michael Backes and Peng Ning (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 371–389.

[59] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. 2014. DNSSEC and Its Potential for DDoS Attacks: a comprehensive measurement study. In *Proceedings of the 2014 ACM Conference on Internet Measurement Conference (IMC).* ACM, 449–460.

[60] Duane Wessels and Marina Fomenkov. 2003. Wow, That's a Lot of Packets. In *Proceedings of the Passive and Active Measurement Workshop.* https://www.caida.org/publications/papers/2003/dnspackets/wessels-pam2003.pdf

[61] Chris Williams. 2019. Bezos DDoS'd: Amazon Web Services' DNS systems knackered by hours-long cyber-attack. https://www.theregister.co.uk/2019/10/22/aws_dns_ddos/.

[62] D. Wing and A. Yourtchenko. 2012. *Happy Eyeballs: Success with Dual-Stack Hosts.* RFC 6555. IETF. http://tools.ietf.org/rfc/rfc6555.txt

[63] S. Woolf and D. Conrad. 2007. *Requirements for a Mechanism Identifying a Name Server Instance.* RFC 4892. IETF. http://tools.ietf.org/rfc/rfc4892.txt

[64] Maarten Wullink, Giovane CM Moura, Moritz Müller, and Cristian Hesselman. 2016. ENTRADA: A high-performance network traffic data streaming warehouse. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP.* IEEE, 913–918.

A EXTRA TABLE AND FIGURES

Figure 15 shows the AAAA queries for Google during the .nz event.

Figure 16 show a timeseries of daily queries per AS during the .nz event.

Figure 17 shows the resolvers with at least 100x traffic growth, for AAAA queries, in the sinkhole experiment.

Table 11 shows the list of top ASes during the .nz event.

B INFLUENCE OF RECURRENT QUERIES

In the §4.1 we establish the New domain for problematic resolvers using a one-off configuration: we saw the volume of queries is time

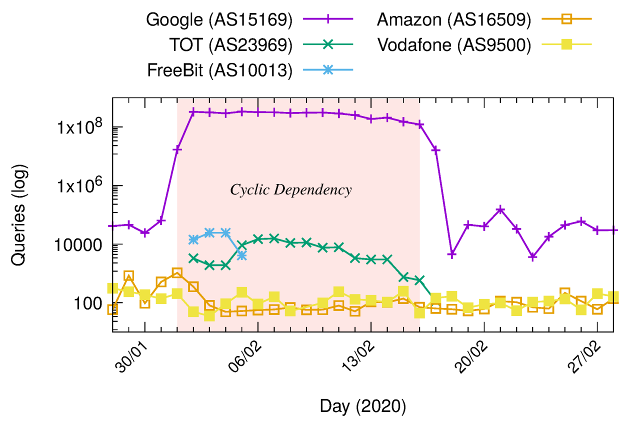


Figure 16: .nz event: queries per AS

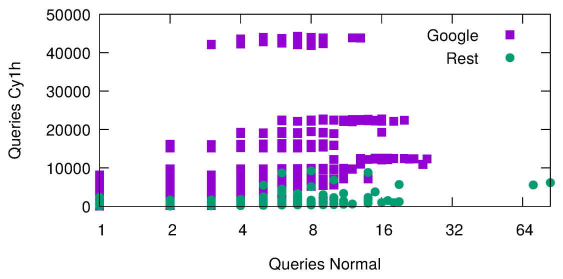


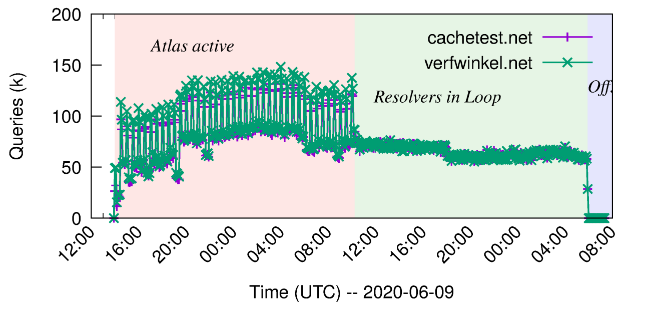
Figure 17: Resolvers with at least 100x traffic growth (AAAA queries)

AS Number	AS name	Country
15169	Google	US
23969	TOT Public Company Limited	Thailand
10013	FreeBit	Japan
36692	Cisco OpenDNS	US
39289	MediaSeti	Russia
3561	CENTURYLINK	US
3452	University of Alabama	US
16509	Amazon, Inc	US
11233	Gorge Networks	US
45142	Loxley Wireless	Thailand
200050	ITSVision	France
30844	Liquid Telecom	UK
15267	702 communications	US

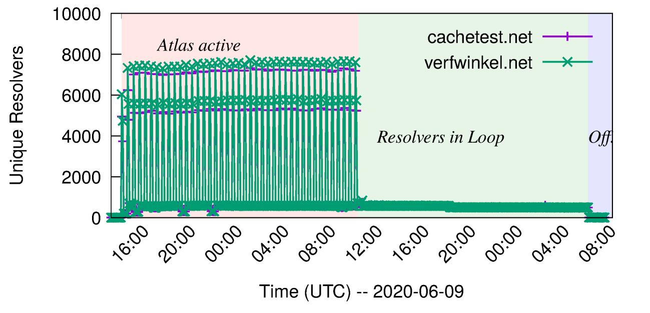
Table 11: List of top ASes per volume of queries during experiments and .nz event.

dependent, and that roughly 574 resolvers (out of 11k, thus 5.1%) are *problematic*.

To determine the influence of recurrent queries – similar to what happened with TsuNAME– we now set up an experiment in which we configure VPs to repeat queries every 10mins, as shown in Table 3 (recurrent column). To avoid warming up caches, we configure Atlas probes to query unique query names, with random values (*R* in qname).



(a) Queries



(b) Resolvers

Figure 18: Recurrent: queries and unique resolvers timeseries (5min bins)

On the client side, we see 13k VPs in Table 3, which issued ~727k queries for this measurement. Most of them were answered as SERVFAIL, similar to the one-off measurement.

On the authoritative server side (ns1 and ns2 in Table 3), we see a different story: altogether, the authoritative servers received ~70M queries over the period – an amplification factor of 99x compared to the queries sent by Atlas VPs.

Influence of recurrent queries: Figure 18 shows the timeseries of both queries and unique resolvers reaching our authoritative servers. We see a large oscillation during the period in which Atlas is active – anywhere from 1k to 8k are active at any moment (Figure 18b). The reasons for that are twofold: some resolvers are indeed in loop here, but also our is configure to send new queries every 30min (Table 3). Similarly to §4.1, once Atlas stops sending queries, we still see a portion of resolvers staying in loop. In fact, for this measurement, we find 1423 resolvers from 192 ASes (Table 4) that are in the loop mode.

Figure 19 shows the top 10 ASes sending queries our authoritative servers when atlas stopped, *i.e.,* they should not have sent any queries. We see that Google did roughly 60% of the queries again, but similarly to the one-off measurement, other ASes have the same issue.

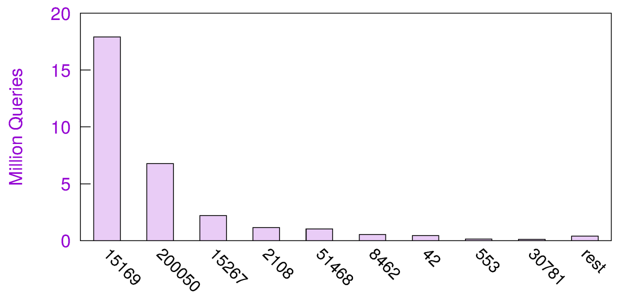


Figure 19: Recurrent experiment: queries per AS with problematic resolvers

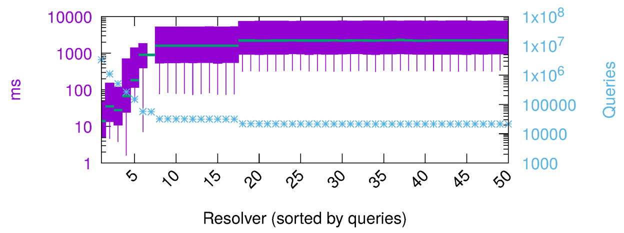


Figure 20: Recurrent: IQR and queries for A records of `ns.vuur.cachetest.net`

Time in-between queries: Figure 20 shows the IQR and queries for the top 50 resolvers in terms of query volume, that send A records queries for a server for a qname in cyclic dependency. Compared with the one-off case (Figure 7), we see a very similar pattern, except for the volume of queries, which is larger, given that the measurement was kept running for longer than the one-off.

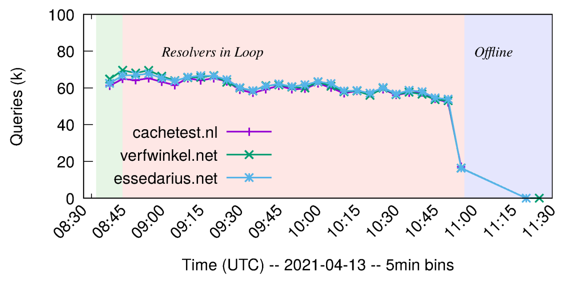
The conclusion we can draw from this experiment is that more client incoming queries will amplify even further the number of queries experienced by authoritative servers.

C STOPPING THE SINKHOLE EXPERIMENT

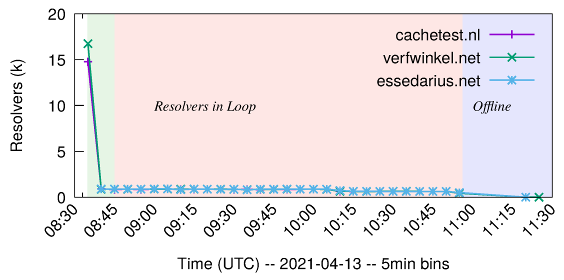
We stop the sinkhole experiment in two steps: We do this in two steps: first, we return these domains to their original, sinkholed NS records – as in the Pre phase. We do that by changing the NS records in the parent DNS zone (`.nl`). In theory, this should redirect all clients to the newly configured NS records. We see that *most* of the resolvers do that, but we still keep on receiving queries on the "old" server (AWS route 53) – the latter are referred to as child-centric resolvers [55], as they trust the information of the child zone delegation over the parent. We fully stop the experiment at 20:18 UTC, by removing the zones from AWS Route 53 (Delegation removed phase). After that point, all clients of this domain query the NS records of the Pre phase, which we do not monitor.

D LONGER AND CNAME CYCLES

In this section we investigate other types of loops, namely longer cycles and CNAME-based loops.



(a) Queries



(b) Resolvers

Figure 21: TripleDep measurement: Queries and unique resolvers querying authoritative servers (5min bins)

D.1 Triple cyclic dependency

First, in measurement TripleDep (Table 3), we configure a triple cyclic dependency (Table 12) to determine if resolvers would also be vulnerable it, and what would be the impact compared to regular cyclic dependencies.

Similarly to the New Domain experiment, we only configure the probes to send 1 query per vantage point. We see in in Figure 21 the timeseries of queries and resolvers we see. Compared with the New domain experiment (Figure 5), we see that the query rates *reduce very litte* after Atlas stop sending queries (> 8:45). We see, however, that only a fraction of resolvers remain active after Atlas stops sending queries (Figure 21b).

Figure 22 shows the top 10 ASes for this experiment. We see that it is similar to the ASes from the New Domain Experiment (Figure 6) – except for the fact that GDNS (AS15169) has been fixed in the meantime, reducing the number of queries.

Figure 21 shows the timeseries results. We see that, differenlty from the normal cyclic dependent domains, a triple cyclic dependency query volume does not reduce as fast as the previous ones. So making longer cycles will make the problem even worse.

D.2 CNAME cycles

We also ran an experiment with loops done with CNAME records [28], which are like 'aliases' for a domain. We configured the experiment CNAME in Table 3, in which we configure cyclic CNAMES: `minuano.essedarius.net` ↔ `tramontana.verfwinkel.net`. Figure 23 shows the timeseries of queries. We see that most resolvers detect the cycle, and do not begin to loop.

	Zones		
Zone	jupiter.essedarius.net	mars.verfwinkel.net	vulcan.cachetest.nl
NS	ns1.mars.verfwinkel.net	ns1.vulcan.cachetest.nl	ns1.jupiter.essedarius.net
TTL	1s	1s	1s

Table 12: Triple cyclic dependency configuration

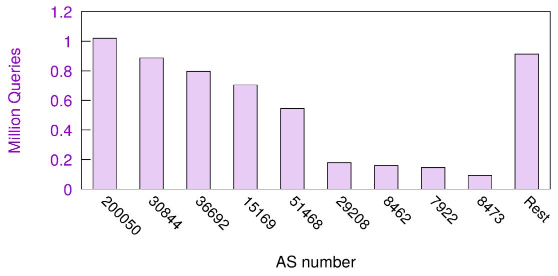
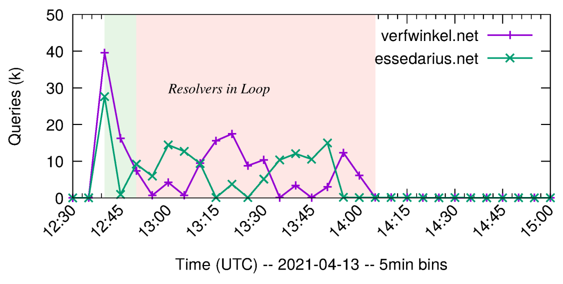
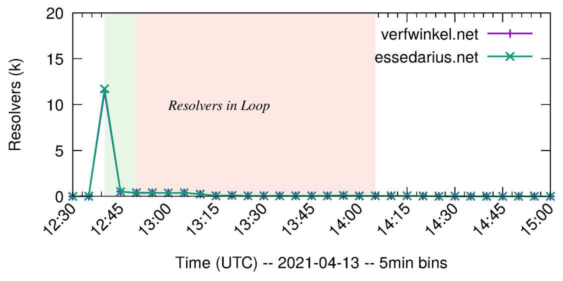


Figure 22: Top 10 ASes for TripleDep experiment



(a) Queries



(b) Resolvers

Figure 23: CNAME measurement: Queries and unique resolvers querying authoritative servers (5min bins)

E IMPACT OF GOOGLE PUBLIC DNS MITIGATION

We worked together with Google in helping to understand the `CycleHunter` vulnerability. We determine that Google would not loop by itself, it was its client population that would. Given Google did not cache cyclic dependent records, queries from looping clients were amplified and sent to authoritative servers.

	Zones	
	platypus.essedarius.net	liger.verfwinkel.net
NS	ns1.liger.verfwinkel.net	ns3.platypus.essedarius.net
TTL	60s	60s

Table 13: Cyclic dependency for new one-off measurement.

Then, on Feb. 3rd, 2021, Google mitigated this vulnerability on their Public DNS services, but implementing a cyclic dependent detector and caching such records, so once it was cached, it would not pass along any client queries – blocking the effects of looping downstream resolvers.

In this section, we reproduce the measurements made with RIPE Atlas to determine the impact of Google's mitigation.

E.1 Repeating lower-bound experiment

In §4.1, we configure ~ 10k Atlas probes to send 1 query only to each of their local resolvers, in order to measure the lower-bound of amplification. In this section, we repeat that experiment in order to determine how much of a problem is stil is after Google's mitigation.

Table 13 shows the cyclic dependency we configured – third level domains not used before. We delete the record on Wed 10 Feb 2021 08:30 UTC, after keeping the cyclic dependency active for 12h and 30min.

Figure 24 shows the results. We see that altogether, after no more user queries, the authoritative servers receiving roughly 88k queries/5min combined (both zones, in Figure 24a). Previously, this value in Figure 5 was around 135k queries/5min. That is a 35% reduction in the total query volume. We also found 1560 problematic resolvers (Figure 24b), which are unique IP addresses sending queries *after* the initial round of queries from the Atlas probes – resolvers in the Cyclic dependent phase. This number is, however, *larger* than the one from Figure 5 – but they generate fewer queries.

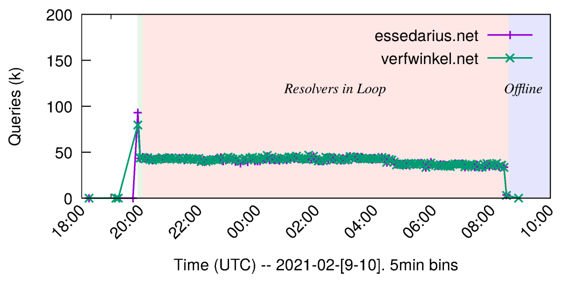
Table 14 shows the details of this measurement. We see that 18.5k VPs sent 18.6k queries, which resulted in 12.3M queries at the authoritative server, in the 12.5h that this measurement lasted.

Figure 25 show the top10 ASes for this experiment. Compared with before the mitigation (Figure 6), Google significantly reduce its volume of queries, from roughly 4.5M to 400k (90%), even if this measurement lasted for 12.5h instead of 6.

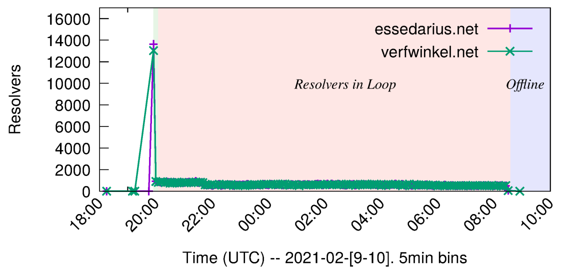
E.2 Repeating recurrent queries measurement

Next, we set out to repeat the measurement with recurrent queries from Appendix B, after Google's mitigation. Table 15 shows the cyclically dependent zones we configured.

Figure 26 shows the timeseries for this experiment. We see in Figure 26a that when Atlas is active, each zone authoritative servers receives roughly 90k queries/5min. That is already fewer queries than Figure 18a, in which it peaked to almost 150k queries/5min.



(a) Queries



(b) Resolvers

Figure 24: Ripe Atlas: Queries and unique resolvers querying authoritative servers (5min bins)

Measurement	**One-Off-AfterGoogle**
Frequency	One-off
Qname	`$P-$r.platypus.essedarius.net.`
Query Type	A
Date	2021-02-09
Duration	12,5h
Client Side	
Atlas Probes	9594
VPs	18539
Queries	18655
Responses	17051
SERVFAIL	13054
Timeout	3865
REFUSED	100
FORMERR	10
NOERROR	22
NXDOMAIN	32
Authoritative Server Side	
Querying IPs	14657
ASes	2581
Queries	12386591
Responses	12386591

Table 14: TsuNAME Emulation Experiments after Google's mitigation. Datasets: [46].

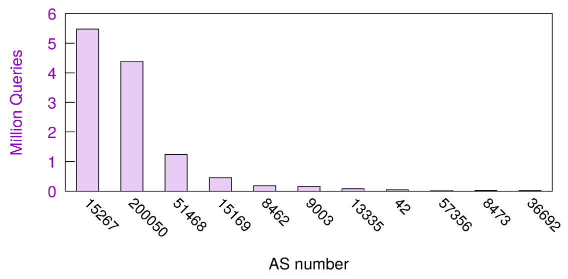


Figure 25: One-off-AfterGoogle: top 10 ASes by query volume with problematic resolvers

	Zones	
	`gioia.essedarius.net`	`infinita.verfwinkel.net`
NS	`ns3.infinita.verfwinkel.net`	`ns5.gioia.essedarius.net`
TTL	60s	60s

Table 15: Cyclic dependency for new one-off measurement.

Measurement	**Recurrent-AfterGoogle**
Frequency	One-off
Qname	`$P-$r.gioia.essedarius.net.`
Query Type	A
Date	2021-02-12
Duration	16h
Client Side	
Atlas Probes	9962
VPs	17884
Queries	451997
Responses	451997
SERVFAIL	446941
Timeout	0
REFUSED	3456
FORMERR	604
NOERROR	816
NXDOMAIN	180
Authoritative Server Side	
Querying IPs	21865
ASes	2560
Queries	21257464
Responses	21257464

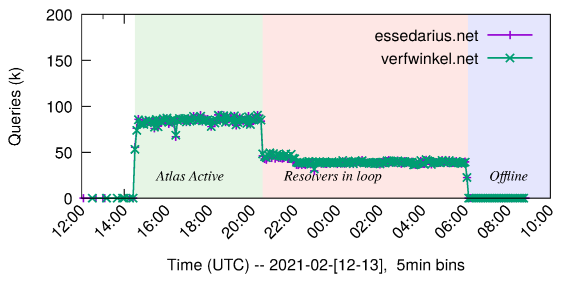
Table 16: TsuNAME Recurrent Experiments after Google's mitigation. Datasets: [46].

When Atlas stops sending queries, the authoritative servers receive roughly 50k queries/5min, also fewer than previously (~75k).

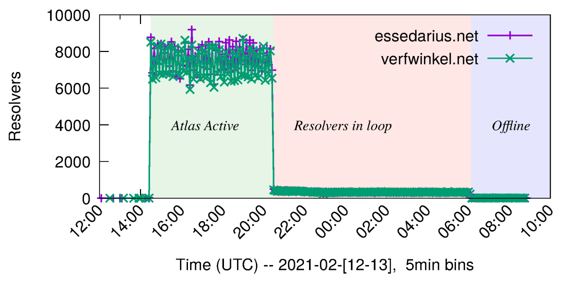
We see that Google now is the 8th position in Figure 27, sending 430k queries out of the 21M – 2.02% of the total, as shown in Table 16.

F THREAT MODEL

The TsuNAME threat model involves using *DNS reflection* to carry out a denial-of-service attack. Instead of attacking these servers



(a) Queries



(b) Resolvers

Figure 26: Ripe Atlas: Queries and unique resolvers querying authoritative servers (5min bins)

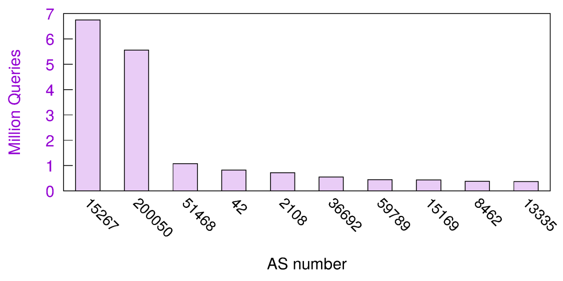


Figure 27: RecurrentAfterGoogle: top 10 ASes by query volume with problematic resolvers

directly, the attack could use cyclic dependent domains and vulnerable resolvers to keep a continuous stream of queries to the designated targets. None of our experiments fully exploited this possibility for ethical reasons; next we discuss how a well motivated could attack could as well do it.

For this to happen, an attacker needs (i) to have domains under a given zone (or take control over them, *e.g.,* by stealing registrant or registrar credentials), (ii) misconfigure them with cyclic dependent NS records, and (iii) induce vulnerable resolvers to carry out queries.

The first and second part are not difficult – most TLDs such as `.org` and `.es` have an open registration policy, so anyone can register domains and misconfigure them. For example, say an attacker has 500 `.fr` and 500 `.ca` domain names under its disposable: it could configure each of them with NS records pointing to each other, as Figure 12.

The last step consists in inducing vulnerable resolvers to query for these domains, so they can enter start looping and unleash a large volume of queries. It will be the parent authoritative servers of the cyclic NS records that will be receiving all the queries (in this case, `.ca` and `.fr` authoritative servers).

The last step involves in finding vulnerable resolvers – our experiments show that there are 4k resolves from 261 ASes vulnerable to TsuNAME, but that is a lower-bound estimative, given we have not covered most resolvers on the Internet (we were limited by the view of our vantage points). Luckily, Google has fixed GDNS after our notification, but there are still other vulnerable resolvers out there, including OpenDNS. One could only think of the possible damage that can be done if an attacker decides to employ a large botnet to send frequent queries, such as the Mirai botnet [5].

Alternatively, hijacking *only one* popular domain and misconfiguring its NS records would also suffice, as in the case with the anonymous European ccTLD (Figure 14). In this way, it is likely that vulnerable resolvers would be automatically found by the regular stream of user queries.

Once resolvers start looping, the effect on the authoritative servers will depend on the attack size versus the authoritative servers's capacity, and there is a large variation among TLDs when it comes to capacity, given there is large variation in the number of authoritative servers and anycast instances per ccTLD.

Most TLDs are likely to suffer at least partial unavailability if faced with 100s of thousands of queries per second. Once down, the consequences can be catastrophic: in case of country-code TLD, most official services, banks, online shopping and others would become unreachable.

Collateral damage: an attack against a particular TLD may have impact a series of others, given they may share parts of the same infrastructure [3, 24], by using the same authoritative DNS providers. When Dyn DNS was attacked, multiple DNS zones were affected. When some of the Root DNS servers were attack in 2015 [50], parts of the Netherlands' `.nl` ccTLD was also affected [32].

G PRIVATE AND PUBLIC DISCLOSURE TIMELINE

Table 17 shows the dates of the public and private disclosures we performed.

Date	Type	Group
2021-02-05	Private Disclosure	OARC34
2021-02-22	Private Disclosure	APTLD
2021-02-23	Private Disclosure	CENTR
2021-03-04	Private Disclosure	LACTLD
2021-02-18–2021-05-05	Private Disclosure	Private
2021-05-06	Public Disclosure	OARC35
2021-05-06	Public Disclosure	https://tsuname.io

Table 17: TsuNAME disclosure timeline

H ANOTHER LOOPING PROBE

		RIPE Atlas Side	
#	**Time**	**Query/Type**	**Resolver**
1	14:14:57	`52196.sub.verfwinkel.net/A`	192.168.88.1
2	14:15:01	`52196.sub.verfwinkel.net/A`	208.67.222.123
3	14:15:02	`52196.sub.verfwinkel.net/A`	208.67.220.123
		Authoritative Server Side	
4	14:15:11	`52196.sub.verfwinkel.net/A`	IP4
5	14:15:11	`ns.sub.cachetest.net/A`	IP4
6	14:15:13	`ns.sub.verfwinkel.net/A`	IP4
7	14:15:14	`ns.sub.verfwinkel.net/A`	IP4
8	14:15:14	`ns.sub.verfwinkel.net/A`	IP4
9	14:15:14	`52196.sub.verfwinkel.net/A`	IP4
		Remaining queries	
Median Δt		**Query/Type**	**Total**
37ms		`ns.sub.verfwinkel.net/A`	169462
36ms		`ns.sub.cachetest.net/A`	169871

Table 18: Query sequence for Probe 52196 during Low bound measurement

Consider probe 52916, from the new domain measurement. Table 18 shows the query history for this probe. At the Atlas side, we see that this probe has sent 3 queries – one per resolver it was configured with. The first query goes to a private IP address, likely a local resolver. Queries 2 and 3 go to OpenDNS, Cisco's public resolver service.

At the authoritative server side, however, we see queries from only one IP address (anonymized as IPv4), which belongs to the the same AS number as the probe (AS15267). Query #4 it is the first query we see on the authoritative server related to this probe, so we map this IP to the probe (as in §4.4). After that, it asks for the A records of the authoritative servers, and it asks again at 14:15:14 for the `52196.sub.verfwinkel.net/A` domain.

Then, the resolver begins to loop: it sends 169k queries for each dependent NS record (as queries #5 and #6), every 37ms, even in the absence of new Ripe Atlas queries. Given this probe use two Open DNS resolvers and a private a private IP address space, we do know if the looping occurs at this private resolver, if it is a forwarder, or at the last level resolver (Figure 4). We tried to identify this local resolver by issuing `chaos TXT queries` to determine their software version [63], but this feature was not implemented (see measurement probe52196 in [46]). Still, the effect is send a large volume of queries to our authoritative servers.

I RESOLVERS DEV/OPS RECOMMENDATIONS

To mitigate the traffic surge from resolvers to authoritative servers caused by the TsuNAME vulnerability, resolver developers MUST instrument their code to both detect cyclically dependent NS records (so loops can be avoided), and cache them likewise (so no further user queries generate new queries to the targeted authoritative servers).

For example, in the Listing 1 and Listing 2 examples, that would involve in detecting that #3 delegation NSes are *unresolvable*, and caching it as that (possibly as SERVFAIL [27]). Then, any subsequent queries to these delegations will notice that there is no resolvable NS record for this zone, and will be answered as SERVFAIL from the cache, reducing the volume of queries to authoritative servers.

```
1 essedarius.net.     1   IN  NS  ns1.example.nl.
  essedarius.net.     1   IN  NS  ns2.example.nl.
```

Listing 1: DNS Zone file: `essedarius.net`

```
  example.nl.     1   IN  NS  ns3.essedarius.net.
2 example.nl.     1   IN  NS  ns4.essedarius.net.
```

Listing 2: DNS Zone file: `example.nl`

Caching, but for how long? The caching duration is inversely proportional to the volume of queries that are forwarded to authoritative servers. Resolver developers must choose this caching value carefully.

RFC2308 [4] states that a SERVFAIL response may be cached for no longer than 5 minutes. That may be reasonable for this case, given that mean-time-to-repair such cyclically dependent records is at least minutes.

Alternatively, a resolver developer may employ a more adaptive TTL method. For example, it may start with 5 minutes, and perform some linear back-off to a larger value, possibly controlled by e.g. negative TTL on the parent zone and/or RFC-specified hard limit, such as 1 hour or 4 hours.

I.1 Testing your resolver software

To test your resolver software, set up cyclically dependent delegations, as shown in Listing 1 and Listing 2. We **strongly recommend creating third-level domain names** (as in our examples) instead of second-level (*e.g.,* `example.nl`) given that cyclically dependent second-level domains will stress authoritative servers of their respective TLDs.

After creating these cyclically dependent delegations, we suggest the following tests:

I.1.1 Test 1: Loop Detection.

(1) Clean the cache of your resolver
(2) Monitor the traffic between the resolver and the Internet
(3) Send ONE query to your for a domain under the misconfigured delegation. For example, `dig A` random.platypus.essedarius.net.
 - Compute how many queries are then send to the parent authoritative servers of both misconfigured zones (lines #1 and #2 of Listing 1 and Listing 2)
 - Determine if your resolver loops indefinitely, or if eventually stop sending queries to the authoritative servers. You may need to monitor for various minutes or hours.

Please notice that the resolver may send a `SERVFAIL` response to your client, but it may remain looping, sending non-stop queries to the authoritative servers.

For a reference, you may want to check `Unbound`'s source code, which includes various cycle detections, as described in their changelog[1].

I.1.2 Test 2: Caching Cyclic Records and Amplification.

(1) Clean the cache of your resolver
(2) Monitor the traffic between the resolver and the Internet
(3) Send ONE query to your for a domain under the misconfigured delegation. For example, `dig A` random.platypus.essedarius.net.

[1]https://github.com/NLnetLabs/unbound/blob/master/doc/Changelog

(4) Then, send this query multiple times every 5 s (or other short interval)
   - Compute how many queries are then send to the parent authoritative servers of both misconfigured zones (lines #1 and #2 of Listing 1 and Listing 2)
   - Determine if the new, recurrent queries from your client (`dig` in this case) cause your resolver to send many more queries to reach the authoritative servers, or if they are answered from cache.

If new user queries (`dig`) lead to more queries to the authoritative servers, you resolver is then vulnerable to TsuNAME.