Technical Report

DomainTools

The DEF CON Recon Village Subdomain Enumeration Challenge:

A Retrospective

Joe St Sauver, Ph.D.
joe@domaintools.com
Distinguished Scientist

Executive Summary

At the 2023 DEF CON, the Recon Village ran the "ReconAacharya Subdomain Enumeration Challenge." Subdomain enumeration is a terrific subject for a cybersecurity competition, since it is often a key first step in attack surface assessment. This report is a post-hoc discussion of the DomainTools-affiliated team's submission to that challenge. In addition to describing our solution, we also describe an alternative approach that can yield two orders of magnitude more results than our team's winning submission at DEF CON 2023.

Organizers provided participants with 14,917 starting "seed" domains. Participants were tasked with finding as many subdomains (aka "fully qualified domain names" or "FQDNs") based upon those seeds as possible during a 72-hour time window. Valid FQDNs would earn a point; invalid FQDNs would lose a point, and if a participant submitted more than 1,000 bogus FQDNs, they could be disqualified. Discovered FQDNs had to satisfy constraints. For example, only "A", "CNAME", "MX", or "TXT" records would count. The FQDNs needed to be based on the supplied "seed" domains. The FQDNs needed to be resolvable via Google's 8.8.8.8 public resolver. Wildcard domains couldn't be exploited. FQDNs for submission had to be listed one per line in files no larger than 25MB, and no more than 100 files could be submitted for scoring.

DEF CON's 2023 winner was a team of current and former DomainTools employees. The DomainTools-affiliated team (DT) chose to use an open source tool called `subfinder` that leveraged DomainTools Farsight DNSDB Standard Search (DNSDB), winning with a submission of a little over 4.1 million FQDNs. While the DomainTools team prevailed, their submission of 4.1 million FQDNs for the 14,917 "seed" domains translated to an average of just 275 FQDNs per seed domain, and an average discovery rate of only around 30 FQDNs/second. Given that DNSDB can return up to a million results for a single domain query, the submission, while sufficient to win this challenge, appeared to leave room for improvement.

In particular, one big opportunity for improvement would be to use DNSDB Flexible Search instead of DNSDB Standard Search. Each Flexible Search query can return up to a million results per query, and customers can run up to ten DNSDB API queries in parallel. Each Flexible Search query result consists solely of a unique query and type (RRname and RRtype) combination, making DNSDB Flexible Search perfectly suited to FQDN enumeration.

Time fencing is a fundamental "knob" controlling how Flexible Search performs by limiting results to observations during a given time window. By varying the time fence used, we can balance how many FQDNs we find against our risk of finding now-unresolvable domains AND our script's run time. Sample run times for time fences of a week, month, or quarter range from 30 minutes to just over two hours, returning a million+ results in each case:

DNSDB Flexible Search (10 Streams)		
7 day time fence (run time: 29:52)	**30 day time fence** (run time: 50:00)	**90 day time fence** (run time: 121:01)
104,274,176 raw FQDNs	213,455,787 raw FQDNs	368,166,151 raw FQDNs

Those raw results can then be validated using active queries. To complete validation in a timely fashion, a parallel resolver such as massdns must be used. Quoting that site, "Without special configuration, MassDNS is capable of resolving over 350,000 names per second using publicly available resolvers."

However, recall that submitted files can only be 25MB in size, and we can upload no more than 100 of those. Taking the shortest FQDNs we found, our final result (100 files no larger than 25MB) ==> **89,280,343 subdomains**.

Some of the "seed" domains for the contest were potentially sensitive, such as domestic or international gov/mil domains, critical infrastructure domains, financial institution domains, security company domains, etc. We recommend voluntarily refraining from enumerating any believed-to-be-sensitive seed domains ("filtering") for two primary reasons: first, the domains discovered and submitted for the challenge would be publicly released, including to potential adversaries, and second, sensitive domains are disproportionately likely to be actively monitored/aggressively protected (e.g., access rate-limited via tarpitting).

The truly difficult part of the challenge is identifying and excluding so-called "deep wildcards," should you need to do so. Regular wildcards are registrable domains (effective 2nd-level domains) that will resolve literally any hostname. "Deep" wildcards involve wildcarded more-specific FQDNs. "Deep" wildcards can only be identified by active probing: does a synthetic name constructed by adding a random string as a hostname successfully resolve? Conveniently, the contest organizers stated, notwithstanding their initially-specified rules, that "Wildcards were handled at domain level, and have been removed. However, wildcards at a nested level were not removed at the time of processing, as nested wildcard subdomains are very unpredictable and difficult to identify."

We nonetheless demonstrate an approach to doing so, reducing the number of discovered domains to **29,408,813**. We finish by recommending that wildcard domains be routinely tracked and reported from time of discovery given the risks they represent.

	2023 DEF CON-Winning Run Using "subfinder" with DNSDB (365- day time fence) (baseline comparator)	**DNSDB Flexible Search (10 Streams) 90-day time fence**
Raw FQDNs	(not tracked)	368,166,151
FQDNs After filtering	4,060,439	206,142,295
Submittable (or submitted) FQDNs	4,060,439	89,280,343
Likely true unique subdomains	n/a	29,408,813

Table of Contents

1. Introduction

During DEF CON in Las Vegas in August 2023, on-site DomainTools staff and alumni participated in the "Recon Village Aacharya Subdomain Enumeration Contest." The idea behind that contest? Given a set of "seed" domain names, how many subdomains (aka "Fully Qualified Domain Names" or "FQDNs") could you identify "under" those domains during the allotted time? For more on the exact parameters of this event, see:

- https://github.com/ReconVillage/reconaacharya/blob/main/domains.txt
- https://www.reconvillage.org/post/the-recon-aacharya-contest-community-data-release
- https://www.domaintools.com/resources/blog/hunting-subdomains-at-def-con-31/
- https://riskreboot.substack.com/p/snatching-defeat-from-the-jaws-of

The DomainTools-affiliated team consisted of the following individuals (in alphabetical order by last name):

- Dan Fernandez (DomainTools alumnus)
- Steven Hallman (Solutions Engineer)
- Tim Helming (Security Evangelist)
- Sean McNee (VP of Research and Data)
- Dan Nunes (VP of Product), and
- Daniel Schwalbe (Chief Information Security Officer)

The DomainTools-affiliated team won the contest (and received a cool PS5 gaming console as a prize)! Even with the excitement from the win, we know there's room for improvement, particularly away from the pressure of an ongoing event and with more time and resources. Can we identify "lessons-learned" for next time? The contest also explicitly states "winners are expected to share their techniques and demonstrate them. Writeups are encouraged, allowing the community to learn from their approaches and insights."

All of the contest participants – organizers and teams (DomainTools-affiliated team and non-DomainTools teams alike) – did a great job, and we don't mean in any way to diminish anyone's efforts by producing and sharing this retrospective.

2. The Challenge in a Nutshell

The challenge provided participants with a list of 14,917 base domains chosen by the contest organizers. Participants were to identify as many subdomains (aka FQDNs) based on those base domains as possible from 10AM PDT on 11th Aug 2023, through 11:59 PM PDT on 12th Aug 2023 (roughly 1 day and 14 hours). Only subdomains with an RRtype of A, CNAME, MX, or TXT record type would count.[1]

Subdomains had to be resolvable via Google's 8.8.8.8 intentionally-open recursive resolver. Any stale or bogus entries that failed to actively resolve would be penalized: "Each valid subdomain adds 1 point to your score, while each invalid subdomain deducts 1 point". Symmetric scoring (+1 for a good FQDN, -1 for a bad FQDN)

[1] The author is always a bit sad to see AAAA (IPv6) records overlooked

might tempt some people to skip validating the domains they'd submit altogether, but another contest rule provided that "Additionally, submitting an excessive number of invalid subdomains (e.g., +1,000) may also result in a ban." The stated limit of just 1,000 invalid subdomains is actually quite strict – 1,000 invalid domains for even 4,000,000 results would be just a 0.025% failure rate! That's really pretty strict if actually enforced.[2]

Wildcard domains (domains that can resolve an infinite number of domains for random subdomains) might also needed to be carefully managed ("Exploiting wildcard subdomains is strictly prohibited.")[3]

Participants could upload a maximum of 100 files of results, with each file limited to 25MB. If we assume domain names are perhaps 11-16 octets long, and "25MB" actually means 25,000,000 bytes (e.g., "decimal" megabytes), that implies just 156,250,000[4] to perhaps 227,272,727[5] domains might be able to be submitted. A distinct possibility: contestants might discover more results than could be submitted.[6]

There was also some ambiguity around what it meant to "successfully resolve" a FQDN. Some names can be queried and will return a "NOERROR" (RCODE=0) response, even if zero answer records are provided. For example:

```
$ dig ipv6.l.google.com A

; <<>> DiG 9.17.21 <<>> ipv6.l.google.com A
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7279
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1
[snip]
```

Following discussions with participating colleagues, the consensus was that a "successful" resolution actually required the DNS to return at least one answer record, not merely returning "NOERROR." We will not consider whether a domain actually has accessible services running on the host that resolves (i.e., no connecting to the hosts themselves).

[2] This ban rule did not appear to have actually been enforced: all teams listed on the final challenge scoreboard had >1,000 invalid subdomains.
[3] This rule also does not appear to have been enforced. When contacted about the wildcard rule after the contest, the contest administrators stated that "Wildcards were handled at domain level, and have been removed. However, wildcards at nested level were not removed at the time of processing, as nested wildcards subdomains are very unpredictable and difficult to identify."
[4] 100*25,000,000/16=156,250,000
[5] 100*25,000,000/11=227,272,727
[6] That might mean that doing things like selecting short domains, thereby maximizing the "number of domains per MB," might become important.

Given all the above constraints, our fundamental question becomes:

Even though the company-affiliated team won with 4.1 million entries, that number still seemed low for 14,917 "seed" domains.

Could we have found more domains in the allotted time, and if so, how?

First, let's talk a little about the approach the DomainTools-affiliated team used this year.

3. Using the `subfinder` Tool to Find Subdomains

The challenge explicitly mentioned (and the DomainTools-affiliated team elected to use) a free/open source tool called "`subfinder`" ("Fast passive subdomain enumeration tool)." The tool is quite powerful, scraping results from up to 42 free and paid sources (provided you have your own key), but had some limitations specific to DNSDB, including:

- The tool performed DNSDB API version 1[7] Standard Search queries. API Version 1 queries don't explicitly address truncated or incomplete queries. Taken together with short timeouts hardcoded into the tool, some results discovered in DNSDB API version 1 may not have been successfully retrieved and tallied. Given the limitations of API Version 1 calls, we simply can't know.

  DNSDB API Version 1 also doesn't support DNSDB API Flexible Search. DNSDB Standard Search returns full record sets (RRsets), and may potentially "use up" results returning multiple variants of a single DNS query (RRname). We'll discuss this in detail further below.

- The subfinder tool did not initially "time fence"[8] the results returned. Without time fencing, DNSDB results may be from any time all the way back to June 2010. Historical results have a heightened likelihood of not resolving currently.

  The DomainTools team modified the tool to use a 365-day time fence. That one year period is a reasonable choice for a DNSDB time fence, but potentially not an optimal one. Is it possible that a shorter time fence (7-day, 30-day, or 90-day time fence, etc.) might have been better? Or should the time fence have been even longer?

- The subfinder tool attempted to request one trillion (1,000,000,000,000) results per DNSDB API query. The default number of results returned by DNSDB API is normally 10,000 per query and the maximum is one million (1,000,000) results per query. Fortunately, the DNSDB API server intelligently interprets requests for more than a million results as "please give me as many results as possible," so ultimately this proved to be a "benign error."

[7] https://www.domaintools.com/resources/user-guides/farsight-dnsdb-api-version-1-documentation/
[8] https://www.domaintools.com/resources/blog/farsights-dnsdb-time-fencing-a-post-attack-time-machine/

- In some cases, domains may have more than a million results available. When more than a million results are available for a domain, and you'd like an answer that's as inclusive as possible, the standard recommendation is to make additional "offset" queries to "page forward" and retrieve additional results. Using three more offset queries (each for up to a million additional results) could have significantly expanded the set of returned results.

DomainTools has since contributed an update to `subfinder` to address many of these shortcomings, where possible, given the design of the tool.

Let's now set the `subfinder` tool aside, and focus on a complete *"tabula rasa"* approach to the problem. See the following outline:

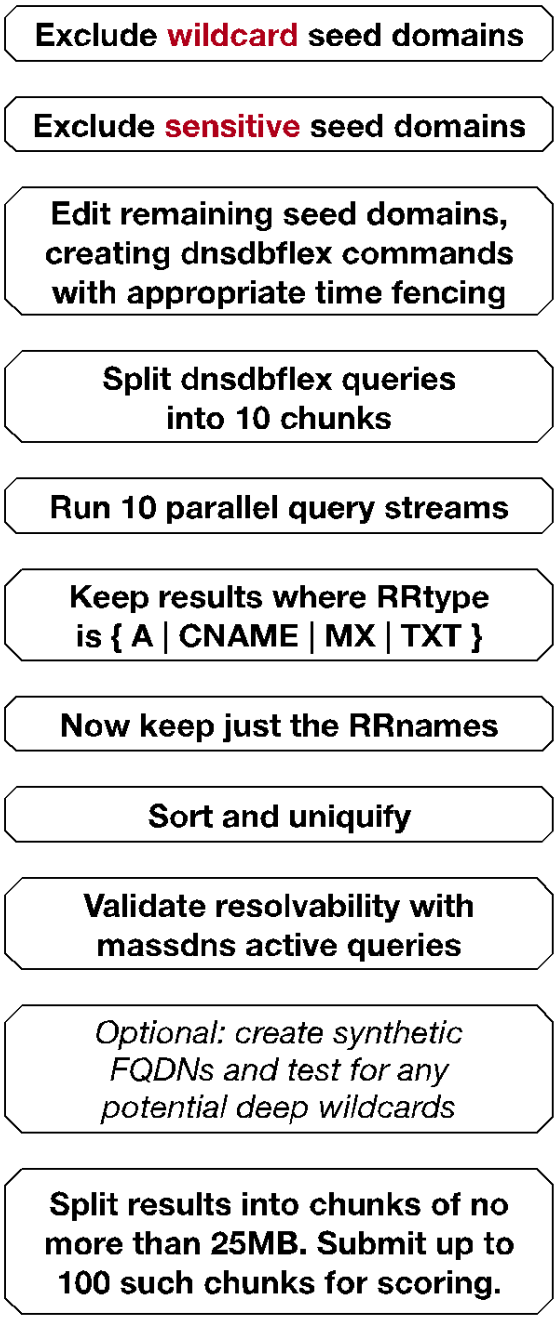


Figure 1. Alternative Approach to the Subdomain Enumeration Challenge

4. "Seed" Domains to Refrain from Enumerating

The contest rules did not explicitly require contestants to enumerate all provided "seed" domains – as far as we can tell, you had the option to "pick and choose" a subset of domains and RRtypes (from the set A, CNAME, MX, TXT) you'd like to focus on and you could also choose the order in which you'd like process those domains and RRtypes.

Passive DNS queries are "safe" in the sense that when you make them, you don't interact with live DNS operations, you're just querying a database. We could safely query any/all of the contest supplied seed domains if we restricted ourselves to just making passive DNS queries. Active queries, on the other hand, as the contest required, might actually impact some seed sites or resolver operators.

Contest organizers were also candid about their plans to publicly disclose the results submitted by participants. This might also have implications for some of the seed sites, publicly exposing FQDNs that they might have assumed were not publicly known/knowable.

So, what domains could be considered for voluntarily omitting, whether for pragmatic reasons or as a matter of professional discretion/ethical self-restraint?

- **Some seed domains might be wildcards.** Making a single active probe of all 14,917 base domains (using a randomly chosen test hostname) allowed us to quickly identify 28 base domains that will act as wildcards for any/all names. These were: `aax.com`, `adda247.com`, `airmap.com`, `allcommonstories.com`, `archdaily.com`, `asapp.com`, `azurefd.us`, `bt1207ka.top`, `chefishoani.com`, `dergipark.org.tr`, `emule.org.cn`, `idencys.nl`, `javgo.net`, `klickly.com`, `lansweeper.com`, `my.com`, `njav.tv`, `okdiario.com`, `pfxcloud.com`, `premierhero.com`, `skrbtla.top`, `technical-service.net`, `vidhub.top`, `vlibras.gov.br`, `wuqianka.top`, `x666x.me`, `yhvod.net`, and `yourbestjournal.com`. Under the terms of the contest, these should be excluded from enumeration.

- Another more subtle wildcard-related question: many domains may have a wildcard for MX purposes – including `marriott.com`, `lincoln.com`, `honda.com`, `huntington.com`, `coca-cola.com`, `ford.com`, and `honeywell.com`. e.g., testing a random synthetic hostname, an A/CNAME record might not resolve, but the MX might:

  ```
  $ dig nvmofvmosvd.telekom.de
  [NXDOMAIN]
  ```

  BUT...

  ```
  $ dig nvmofvmosvd.telekom.de mx
  [...]
  nvmofvmosvd.telekom.de.    3600  IN    MX    100 mailin23.telekom.de.
  nvmofvmosvd.telekom.de.    3600  IN    MX    100 mailin43.telekom.de.
  ```

```
nvmofvmosvd.telekom.de.    3600  IN    MX    100 mailin33.telekom.de.
nvmofvmosvd.telekom.de.    3600  IN    MX    100 mailin13.telekom.de.
```

We can't (and won't!) know about these RRtype-specific wildcards unless we explicitly test each seed domain or each discovered FQDN for all four RRtypes of interest

- **Some domains found in passive DNS may use "tarpitting" (simply "hanging" or "timing out") if actively queried.** We recommend skipping those recalcitrant domains when you find them – there are plenty of other domains you can focus on, instead. Speed counts during a time-limited challenge, and tarpitting hurts throughput.

- **Gov/mil domains, and the foreign equivalents thereof, are another obvious category of domains we might exclude.** The contest operators stated that they intended to publicly share the results of the contest, and they've done what they've promised to do. We wouldn't want to equip any potential adversary with free reconnaissance details relating to potentially sensitive governmental domains. We will also note that in some cases it can be all-to-easy to accidentally include some governmental domains even when we don't mean to do so. For example, consider `gc.ca`, as used by the Government of Canada – it's easy to overlook that domain because it doesn't use a typical "gov" or "mil" label.

- **We also wanted to exclude life/safety-related domains such as medical centers and critical infrastructure (power grid operators, gas line operators, airlines, etc.)** – we wouldn't want to inadvertently interfere with the operation of critical facilities or hinder the treatment of someone who's sick or having a medical emergency just to try to win a contest! This resulted in exclusion of sites such as `clevelandclinic.org`, `mayoclinic.org`, `delta.com`, `united.com`, etc.

- Probing **banks and other financial institutions** might result in their security teams getting paged or in other protective measures being deployed. Those considerations resulted in excluding domains such as `barclays.co.uk`, `capitalone.com`, `discovercard.com`, `mastercard.com`, `paypal.com`, `visa.com`, `westernunion.com`, and so on.

- **Cyber infrastructure-related organizations** (such as `icann.org`, `apnic.net`, and `ripe.net`) are still another example of domains that deserve special "exemption from enumeration status" given the significance of their role in the Internet ecosystem.

- Any **cybersecurity companies** we noticed were also excluded (you can think of this as "professional courtesy" if you like). We don't mind including our own `domaintools.com` which happened to be in the list, however.

- **Adult-only domains** (such as pr0n, gambling, and similar "NSFW" sites) were candidates for exclusion, too. The content of those sites might not be legal in all jurisdictions or for all audiences.

We apologize in advance for any "special" domains of the above sorts that we may have accidentally overlooked.

5. Farsight DNSDB Standard Search vs. Flexible Search

Per our outlined plan, we'll use Farsight DNSDB, now part of DomainTools, to find the FQDNs associated with our list of seed domains. The DNSDB API has two potentially relevant search modes: DNSDB Standard Search and DNSDB Flexible Search.

DNSDB Standard Search

DNSDB Standard Search is the best-known DNSDB search mode. It can perform queries using left hand whole-label wildcards, e.g., you could search for things like `*.example.com`. Superficially, DNSDB Standard Search feels like it would be perfect for this challenge.

However, consider what DNSDB Standard Search returns for results: It tracks and returns full "RRsets" – five-tuples consisting of an RRname, RRtype, Rdata, Bailiwick, and source (sensor or zone file) for each result. Each unique RRset returned counts against the maximum of a million results that can be retrieved for a given pattern of interest in a single query.

For example, if we make a standard search in the DNSDB API for `www.whitman.edu`, we find three results, differing solely in their Rdata IP address values. These get tracked and reported separately in DNSDB API Standard Search:

```
$ dnsdbq -r www.whitman.edu -s -k first

   ;; record times: 2010-06-24 13:49:41 .. 2014-03-27 16:40:25 (~3y ~277d)
   ;; count: 940920; bailiwick: whitman.edu.
   www.whitman.edu.  A  199.89.174.13

   ;; record times: 2014-03-26 21:27:00 .. 2022-04-18 17:13:39 (~8y ~24d)
   ;; count: 1616593; bailiwick: whitman.edu.
   www.whitman.edu.  A  199.89.174.11

   ;; record times: 2022-04-18 17:23:31 .. 2023-09-07 21:22:17 (~1y ~142d)
   ;; count: 215887; bailiwick: whitman.edu.
   www.whitman.edu.  A  216.176.184.235
```

Three results yielding just one name? Not a material issue. However, sometimes you may run into single RRnames that have millions of unique RRsets for just a single fully qualified domain name and RRtype. For example, consider the FQDN:

`abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn.`

We'll make an initial query for up to a million results from over a ninety day period:

```
$ dnsdbq -r abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn/A
-l0 -A90d -j
-T datefix >
abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn.jsonl
dnsdbq [2023-09-07 23:25:36]: Database API limit: Result limit reached
```

Decoding that command:

- We're using **dnsdbq**, our command line client (see https://github.com/dnsdb/dnsdbq):
- We're searching RRnames (**-r**) (DNS queries) for the exact domain name specified
- We're only interested in "A" records (**/A**)
- We want up to a million results (**-l0** ("dash ell zero"))
- We're only interested in results that were seen sometime in the last 90 days (**-A90d**)
- We want output in JSON Lines format (**-j**)
- We want datetimes shown in human format (rather than Un*x seconds): **-T datefix**
- We want our results to be piped into the specified output file (**> filename**)

Since DNSDB told the user that the result limit had been reached, we know that there might still be **more results** if we were to make an additional offset query, "paging forward" through our results pool. Doing so entails using the -O (dash capital O) option, appending the output (>>) onto our first tranche of results:

```
$ dnsdbq -r abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn/A
-l0 -A90d -j
-T datefix -O1000000 >>
abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn.jsonl
dnsdbq [2023-09-07 23:26:21]: Database API limit: Result limit reached
$ dnsdbq -r abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn/A
-l0 -A90d -j
-T datefix -O2000000 >>
abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn.jsonl
dnsdbq [2023-09-07 23:26:54]: Database API limit: Result limit reached
$ dnsdbq -r abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn/A
-l0 -A90d -j
-T datefix -O3000000 >>
abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn.jsonl
dnsdbq [2023-09-07 23:28:01]: Database API limit: Result limit reached
```

After running that full set of four queries, we can confirm that we now have 4,000,000 results:

```
$ wc -l abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn.jsonl
4000000 abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn.jsonl
```

So, what the heck is going on? How can there be 4,000,000 unique RRsets associated with just one FQDN? Answer: each of the RRsets separately tracks a unique combination of Rdata IP address values. Let's look at the data more closely:

```
$ more abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn.jsonl
{"count":1,"time_first":"2023-06-20 18:57:19",
"time_last":"2023-06-20 18:57:19",
"rrname":"abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn.",
"rrtype":"A","bailiwick":"cn-north-1.elb.amazonaws.com.cn.","rdata":["43
.196.4.90",
"52.80.6.198","52.80.21.148","52.80.28.126","52.80.38.86","52.81.119.56",
"71.131.255.137","140.179.153.199"]}
{"count":1,"time_first":"2023-06-20 19:15:14",
"time_last":"2023-06-20 19:15:14",
"rrname":"abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn.",
"rrtype":"A","bailiwick":"cn-north-1.elb.amazonaws.com.cn.","rdata":["43
.196.4.90",
"52.80.6.198","52.80.21.148","52.80.28.126","52.80.164.52","54.223.107.44
",
"54.223.135.134","71.131.232.164"]}
```

[etc, etc, etc for four million results]

We can identify the unique IP addresses that make up that pool using `jq` (see https://jqlang.github.io/jq/) and `sort -u`:

```
$ jq -r '.rdata[]' <
abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn.jsonl | sort -u
> all-the-ips.txt
$ wc -l all-the-ips.txt
375 all-the-ips.txt
```

So, we now know the full pool of IPs is only 375 IPs in size. But can we really end up with over 4,000,000 unique combinations of 8 IPs from a pool of just 375 IP addresses? Through the magic of combinatorial math, yes. Given 375 objects, choosing 8 without replacement is 8,996,714,454,741,750 order-insensitive combinations. That's a lot of combinations.

In this example, we've "run through" or "used up" four million queries just returning combinations of 375 IP addresses for one RRNAME! The DEF CON subdomain enumeration contest does not care about unique RRsets, just if unique RRnames actually return results. In this case, we didn't need four million results, we only needed one.

DNSDB Flexible Search

Fortunately, there is DNSDB Flexible Search. As its name implies, it affords greater flexibility with how searches can be represented and submitted to DNSDB. It can do keyword searches and regular expression searches.

So how does Flexible Search help in this case? Well, Flexible Search only returns unique (RRname, RRtype) combinations, exactly what we need for this contest! Normally "more detail is better," but in this case, the reduced information tracked and returned by DNSDB Flexible Search means that it can return more unique (RRname, RRtype) combinations than from an equivalent DNSDB Standard Search.

To see what this all looks like, let's repeat our `amazonaws.com.cn` example domain using our `dnsdbflex` tool (https://github.com/farsightsec/dnsdbflex). Instead of finding four million RRsets, we get one result for the RRname:

```
$ dnsdbflex --regex
"^abj-clb-spider-1728190622\.cn-north-1\.elb\.amazonaws\.com\.cn\.$"

{"rrname":"abj-clb-spider-1728190622.cn-north-1.elb.amazonaws.com.cn.",
"rrtype":"A"}
```

Now let's try using Flexible Search with a **wildcard pattern**.[9] For example, let's try searching `*.custhelp.com`, one of the contest's supplied domains, over a 90 day time window for up to one million results:

```
$ dnsdbflex --regex "\.custhelp\.com\.$" -l0 -A90d > custhelp.com.jsonl
```

We formally write our regular expression with backslashed dots (so they're treated as literal periods), the formal trailing dot that's on every DNSDB domain name, and a right hand anchor just in case this string might otherwise end up found in the middle of some other name.

When we run that command, we find 22,601 results, largely comprised of RRtypes relevant to our challenge:

```
$ wc -l custhelp.com.jsonl
22601 custhelp.com.jsonl

$ jq -r '.rrtype' < custhelp.com.jsonl | sort | uniq -c | sort -nr
  22347 A
    118 CNAME
     68 AAAA
     40 TXT
     28 MX
```

The contest isn't interested in AAAA's, so let's just keep unique RRnames with "A," "CNAME," "TXT," or "MX" RRtypes:

```
$ egrep '("A"|"CNAME"|"TXT"|"MX")' custhelp.com.jsonl | jq -r '.rrname' |
sort -u > wanted-results.txt

$ wc -l wanted-results.txt
22483 wanted-results.txt
```

We now have an approach that seems perfectly targeted to find precisely the sort of names we're after.

[9] These are wildcard DNSDB queries, not wildcard domain names. For details on the formatting of these expressions, see https://www.domaintools.com/resources/user-guides/dnsdb-farsight-compatible-regular-expressions-fcre-reference-guide/

6. Enumerating Our Seed Domains with Flexible Search

Now that we've identified an appropriate approach, let's try it with our seed domains. We've excluded the domains known to be wildcards, but otherwise left the rest of the domains (14,890) in our test set since we're just making passive queries at this point.

We ran a sequential set of queries with a little script that looked like:

```
date
dnsdbflex --regex "\.0123movie.net\.$" -l0 -A90d > 0123movie.net.jsonl
dnsdbflex --regex "\.0800-8051.nl\.$" -l0 -A90d > 0800-8051.nl.jsonl
dnsdbflex --regex "\.0900-8844.nl\.$" -l0 -A90d > 0900-8844.nl.jsonl
dnsdbflex --regex "\.09008844.nl\.$" -l0 -A90d > 09008844.nl.jsonl
dnsdbflex --regex "\.10010.com\.$" -l0 -A90d > 10010.com.jsonl
[...]
dnsdbflex --regex "\.zztongyun.com\.$" -l0 -A90d > zztongyun.com.jsonl
dnsdbflex --regex "\.zzu.edu.cn\.$" -l0 -A90d > zzu.edu.cn.jsonl
dnsdbflex --regex "\.zzzmh.cn\.$" -l0 -A90d > zzzmh.cn.jsonl
date
```

Running that script sequentially took 13 hours, 10 minutes, and 50 seconds, with these clock times:

```
Fri 08 Sep 2023 06:30:55 AM UTC
Fri 08 Sep 2023 07:41:45 PM UTC
```

Those runs found nearly 400 million unfiltered results, potentially including deep wildcards:

```
$ wc -l *.jsonl
[...]
399,857,969 total
```

How many results did we see for each of our runs? There was a wide range of results – from zero results per query to over 2.8 million results/query. We can use a little Python3 Seaborn graphics program to build a cumulative distribution histogram (Figure 2):

```
$ cat distplot.py
import seaborn as sn
import matplotlib.pyplot as plt
import pandas as pd

pd.option_context('mode.use_inf_as_na', True)

df = pd.read_csv("sorted-output-line-counts")

# handle the log(0) problem
df["results_returned_by_dnsdbflex"] += 1

fig, ax = plt.subplots(figsize=(7, 4))

sn.histplot(df, x="results_returned_by_dnsdbflex", cumulative=True,
```

```
    log_scale=10, fill=False, bins=100)
plt.title("Results per run returned by dnsdbflex")
plt.xlabel("number of dnsdbflex run with this count")
plt.xlabel("log10( (observations returned per run)+1 )\n")
plt.ylabel("cumulative observations out of 14890")
plt.savefig('distro.pdf')
```

Figure 2. Cumulative Distribution Showing the Number of Runs Returning N or Fewer Results

You might be surprised to see `dnsdbflex` return more than a million results (e.g., there are bars going "past" 10^6 on the X axis) since a million is the largest number of results we can ask to receive. The explanation for this is simple: one million is the maximum number of unique RRnames we can receive for any single RRtype for a Flexible Search query, but since an RRname can have multiple RRtypes in the results, that can sometimes result in more results than expected when all RRtypes are considered.

Let's look at the results from doing one Flexible Search query, in this case for `pages.dev.jsonl` (our results for `*.pages.dev`).

That single run returned 2,814,313 results. Looking at the distribution of RRtypes, we see that there are multiple RRtypes, but each RRtype has less than 1,000,000:

```
$ jq -r '.rrtype' < pages.dev.jsonl | sort | uniq -c
 968166 A
 589608 AAAA
      1 CNAME
    565 HINFO
```

```
 15976 HTTPS
     3 MX
652085 NS
587087 SOA
   822 TXT
```

So how many results had a million or more records? 209 of our "seed" domains returned at least a million results from Flexible Search, see Appendix B for the results culled from the top of these results:

```
$ wc -l *.jsonl | sed 's/\.jsonl$//' | sort -nr > sorted-output-line-counts
```

What about "the other end" of the distribution? What about seed domains that returned zero hits for our time fenced queries? We had 560 of those. These domains might have "real" FQDNs that we just didn't happen to see during our 90-day window, but including them would increase the chance that we'd waste time on them for no effect. We'd suggest ignoring them. See Appendix C.

7. Post Processing Our Flexible Search Results

Now that we have 399,857,969 total results, we need to post-process those and keep the relevant bits. We will remove the following

- Domains we want to voluntarily exclude
- Domains that returned zero hits
- Inapplicable RRtypes, everything beyond "A", "CNAME", "MX" and "TXT"
- The "deep wildcards", the toughest challenge to tackle

Because we named each of our output files after the respective domains, removing our to-voluntarily-exclude domains is easy enough -- just get a list of all the files-we-should-exclude, and use `grep -v` to exclude names matching anything in the exclusion list:

```
$ ls -1 *.jsonl > result_files.txt
$ grep -v -f excluded.txt result_files.txt > results_wo_exclusions.txt

$ wc -l results_wo_exclusions.txt
13897 results_wo_exclusions.txt                ← down 1,020 from 14,917
```

We can do a similar exclusion for empty results files:

```
$ find . -name "*.jsonl" -size 0c -print | sed 's/\.\///' > zero-files.txt
$ wc -l zero-files.txt
560 zero-files.txt
$ grep -v -f zero-files.txt results_wo_exclusions.txt > results_reduced.txt

$ wc -l results_reduced.txt
13380 results_reduced.txt                      ← down 517 from 13,897
```

Now we just want to keep hits that have an RRtype that's one of (`"A", "CNAME", "MX", "TXT"`). Note that the following command uses a slightly tricky bit of bash syntax[10] that matches and dumps results from a list of files:

[10] https://www.gnu.org/software/bash/manual/bash.html#Command-Substitution

```
$ time egrep --no-filename '("A"|"CNAME"|"MX"|"TXT")' $(< results_reduced.txt)
> results_with_good_rrtypes.jsonl

real    7m36.913s
user    0m55.082s
sys     0m19.628s

$ wc -l results_with_good_rrtypes.jsonl
348,489,095 results_with_good_rrtypes.jsonl
```

The above command saves just the records with the relevant RRtypes in `results_reduced.txt`.

> **A Brief Digression Around Alternatives to egrep:** Because this is a time sensitive challenge, minutes can count. Spending ~7.5 minutes on the preceding egrep (as bolded above) may seem like "too long of a time." There's some truth to that. We can reduce our elapsed time by over 4 minutes (~50%) by simply substituting ripgrep for egrep (see https://github.com/BurntSushi/ripgrep):
>
> ```
> $ time rg --no-filename '("A"|"CNAME"|"MX"|"TXT")' $(< reduced.txt) >
> results_with_good_rrtypes-RG.jsonl
>
> real  3m32.748s
> user  1m32.549s
> sys   0m28.486s
> ```
>
> The number of results returned by ripgrep is the same as it was for egrep:
>
> ```
> $ wc -l results_with_good_rrtypes-RG.jsonl
> 348,489,095 results_with_good_rrtypes-RG.jsonl
> ```
>
> This is still nearly 350 million results

Coming back to our data reduction pipeline, the next step is to extract just the relevant field we need from those records. We'll use `jq` for that. While not particularly speedy, it processes the output line-by-line and does not attempt to read the whole JSON Lines files into memory, so it is well-suited to our large dataset:

```
$ time jq -r '.rrname' < results_with_good_rrtypes.jsonl | sed 's/\.$//' >
just_the_rrnames

real    10m45.168s
user    12m48.551s
sys     0m36.748s
```

> **A Brief Digression Around Alternatives to jq and sed**: As with replacing egrep, we can reduce the time for the above routine by writing a small Python3 program that leverages the fast orjson JSON library (https://pypi.org/project/orjson/):

```
import sys
import orjson

for line in sys.stdin:
    line = line.rstrip()
    results=orjson.loads(line)
    results2=orjson.dumps(results["rrname"]).decode('utf-8')

print(results2.lstrip('"').rstrip('".'))
```

Running this specialized code instead of jq and sed, we see a 3+ minute improvement in throughput vs the original red time above:

```
$ time test_orjson.py < results_with_good_rrtypes.jsonl >
just_the_rrnames_orjson

real  7m29.194s
user  6m52.880s
sys   0m17.874s
```

We now want to "uniquify" our results (e.g., eliminating duplicate RRnames). We can't do that until we've sorted the RRnames. Because we're dealing with ~350 million results, we'll do this process "one step at a time" rather than as a single consolidated Un*x pipeline, and we'll be careful to specify `LC_ALL=C` to minimize overhead for our `sort` command:

```
$ time LC_ALL=C sort -T . < just_the_rrnames > just_the_rrnames_sorted

real    2m21.635s
user    4m32.993s
sys     0m38.350s

$ time uniq < just_the_rrnames_sorted > just_the_rrnames_sorted_and_uniqued

real    1m0.441s
user    0m32.325s
sys     0m11.948s

$ wc -l just_the_rrnames_sorted_and_uniqued
346,241,655 just_the_rrnames_sorted_and_uniqued
```

<-- We've now removed 2,247,440 duplicates

So in total, we've removed 53,616,314 results that were of the wrong RRtype or were present multiple times.

The whole preceding operation was painfully sequential. It doesn't need to be. We can run up to ten concurrent DNSDB API queries with our API key. We can also better optimize our time fencing.

8. Optimizing Time Fencing and Parallelization

By default, DNSDB will return results going back to June 2010, which is great for identifying historic low-traffic (or low-public-traffic) subdomains. However, the further back we go, the greater the chance that we'll receive historical but "no-longer-resolvable" FQDNs (i.e. hosts that have been renamed or taken off of the network).

On the other hand, if we severely curtail our time window, we increase the chance that a passive DNS sensor won't happen to observe legitimate FQDNs that actually do exist for our "seed" domains. We will test three time fencing windows (7, 30, and 90 days) to illustrate this point. Changing our time fencing might also impact `dnsdbflex` run times.

Speaking of run times, we also want to demonstrate the impact of running jobs concurrently. Recall that our sequential approach took over 13 hours to run. Could we reduce that by running in parallel, or would some phenomena we failed to fully understand interfere with accelerating our runs that way? There's no way to know for sure except by testing it! These time fencing runs provide a perfect "excuse" for us to do so.

We began by using an editor to add our `dnsdbflex` commands to the list of domains to be checked. The domains were then sorted and split into ten groups using the Un*x `split` command, resulting in ten roughly equal-size scripts full of `dnsdbflex` commands:

```
$ wc -l xa[abcdefghij]
  1417 xaa
  1374 xab
  1386 xac
  1398 xad
  1406 xae
  1378 xaf
  1393 xag
  1388 xah
  1380 xai
  1397 xaj
13917 total
```

Those files looked something like:

```
$ cat xaa
date
dnsdbflex --regex "\.0123movie.net\.$" -l0 -A90d > 0123movie.net.jsonl
dnsdbflex --regex "\.0800-8051.nl\.$" -l0 -A90d > 0800-8051.nl.jsonl
dnsdbflex --regex "\.0900-8844.nl\.$" -l0 -A90d > 0900-8844.nl.jsonl
[...]
dnsdbflex --regex "\.bbcgoodfood.com\.$" -l0 -A90d > bbcgoodfood.com.jsonl
dnsdbflex --regex "\.bbci.co.uk\.$" -l0 -A90d > bbci.co.uk.jsonl
dnsdbflex --regex "\.bbcollab.com\.$" -l0 -A90d > bbcollab.com.jsonl
date

* * *
```

```
$ cat xaj
date
dnsdbflex --regex "\.upcbroadband.com\.$" -l0 -A90d > upcbroadband.com.jsonl
dnsdbflex --regex "\.upenn.edu\.$" -l0 -A90d > upenn.edu.jsonl
dnsdbflex --regex "\.upgrade.com\.$" -l0 -A90d > upgrade.com.jsonl
[...]
dnsdbflex --regex "\.zztongyun.com\.$" -l0 -A90d > zztongyun.com.jsonl
dnsdbflex --regex "\.zzu.edu.cn\.$" -l0 -A90d > zzu.edu.cn.jsonl
dnsdbflex --regex "\.zzzmh.cn\.$" -l0 -A90d > zzzmh.cn.jsonl
date
```

We then did three sets of runs:

- Ten parallel `dnsdbflex` processes with a 90-day time window, all ten streams started at Fri 08 Sep 2023 08:11:01 PM UTC. The first of the ten jobs finished at Fri 08 Sep 2023 09:43:14 PM UTC (time to first job completion: 1 hour 32 minutes and 13 seconds), while the last of the ten jobs finished at Fri 08 Sep 2023 10:12:02 PM UTC (total elapsed time 2 hours 1 minute and 1 second**).** Results found: 368,166,151 total

- Ten parallel `dnsdbflex` processes, with a 30-day time window, all ten streams started at Fri 08 Sep 2023 10:25:19 PM UTC. The first of the ten jobs finished at Fri 08 Sep 2023 10:57:24 PM UTC (time to first job completion: 32 minutes and 5 seconds), while the last of the ten jobs finished at Fri 08 Sep 2023 11:15:19 PM UTC (total elapsed time: 50 minutes and 0 seconds). Results found: 213,455,787 total

- Ten parallel `dnsdbflex` processes with a 7-day time window, all ten streams started at Fri 08 Sep 2023 11:41:59 PM UTC. The first of the ten jobs finished at Sat 09 Sep 2023 12:00:28 AM UTC (time to first job completion: 18 minutes and 29 seconds), while the last of the ten jobs finished at Sat 09 Sep 2023 12:11:51 AM UTC (total elapsed time: 29 minutes and 52 seconds). Results found: 104,274,176 total

DNSDB Flexible Search (10 Streams)		
7 day time fence (run time: 29:52)	**30 day time fence** (run time: 50:00)	**90 day time fence** (run time: 121:01)
104,274,176 raw FQDNs	213,455,787 raw FQDNs	368,166,151 raw FQDNs

As shown in the table,

- A 7-day time fence discovered over 104 million raw FQDNs in half an hour;
- At 30 days, we discovered over 213 million raw FQDNs in fifty minutes; and
- At 90 days, we discovered over 368 million raw FQDNs in just a couple of hours.

Clearly running in parallel dramatically reduces our run times, and shortening our time fence both reduced the number of results found and the time it took to find them.

Given the fact that participants were limited to uploading a maximum of 100 files of results, with each file 25MB or smaller in size, a 30-day time fence (rather than the 365-day time fence actually used) may have yielded a near-optimal number of submittable results, depending on the number of domains that proved to actually be deep wildcards or currently unresolvable.

9. Actively Validating the Resolvability of Our Results; Final Total Number of Results

New FQDNs are continually being created and deleted. The subdomain enumeration challenge had a miniscule allowance for non-resolvable or otherwise bogus domains – potentially as few as 1,000! It was essential to validate that all submitted domains would actively resolve. Doing that process conventionally (e.g., sequentially) would take far too long.

Fortunately, we can use massdns to resolve a set of domains using numerous concurrent query streams spread across multiple public resolvers. For example, testing massdns on a file of 84,575,861 domains using the default (10,000 concurrent sessions) completed in less than three hours, returning 5.4GB of results:

```
$ massdns -r resolvers.txt -t A -o Sm -w massdns.out domains-to-test.txt
Concurrency: 10000
Processed queries: 84575861
Received packets: 114799104
Progress: 100.00% (02 h 50 min 15 sec / 02 h 50 min 15 sec)
Current incoming rate: 119 pps, average: 11238 pps
Current success rate: 2 pps, average: 8229 pps
Finished total: 84575861, success: 84068307 (99.40%)
Mismatched domains: 18650719 (16.26%), IDs: 299 (0.00%)
Failures: 0: 33.57%, 1: 25.56%, 2: 15.91%, 3: 9.55%, 4: 5.69%, 5: 3.37%, 6:
2.00%, 7: 1.21%, 8: 0.74%, 9: 0.47%, 10: 0.30%, 11: 0.20%, 12: 0.14%, 13:
0.10%, 14: 0.08%, 15: 0.06%, 16: 0.05%, 17: 0.04%, 18: 0.03%, 19: 0.03%, 20:
0.02%, 21: 0.02%, 22: 0.02%, 23: 0.02%, 24: 0.02%, 25: 0.01%, 26: 0.01%, 27:
0.01%, 28: 0.01%, 29: 0.01%, 30: 0.01%, 31: 0.01%, 32: 0.01%, 33: 0.01%, 34:
0.01%, 35: 0.01%, 36: 0.01%, 37: 0.01%, 38: 0.01%, 39: 0.01%, 40: 0.01%, 41:
0.01%, 42: 0.01%, 43: 0.01%, 44: 0.01%, 45: 0.01%, 46: 0.01%, 47: 0.01%, 48:
0.00%, 49: 0.00%, 50: 0.60%,
Response: | Success:               | Total:
OK:       |     74400561 ( 88.50%) |     87428543 ( 76.21%)
NXDOMAIN: |      9667746 ( 11.50%) |     11019179 (  9.61%)
SERVFAIL: |            0 (  0.00%) |     14087422 ( 12.28%)
REFUSED:  |            0 (  0.00%) |      2158629 (  1.88%)
FORMERR:  |            0 (  0.00%) |        22675 (  0.02%)


$ ls -l massdns.out
[snip] 5,704,102,674 Oct 2 02:06 massdns.out

$ wc -l massdns.out
89,741,059 massdns.out
```

The careful reader will have noticed that we have more lines of results than we started with – that's because while some queries failed to resolve, other queries actively resolved to "numerous" results. For example, if you

use `dig` to resolve `all.ranker.app.eu-west-1.prod.cloud.netflix.net`, you'll find that it resolves to literally thousands of IPs:

```
all.ranker.app.eu-west-1.prod.cloud.netflix.net. A 100.86.163.209
all.ranker.app.eu-west-1.prod.cloud.netflix.net. A 100.83.145.226
all.ranker.app.eu-west-1.prod.cloud.netflix.net. A 100.83.82.239
all.ranker.app.eu-west-1.prod.cloud.netflix.net. A 100.83.136.112
all.ranker.app.eu-west-1.prod.cloud.netflix.net. A 100.83.136.86
[etc., etc., etc.]
```

We can now try a larger `massdns` run. Actively resolving the full set of results from DNSDB Flexible Search with a 90-day window using a concurrency of 30,000 takes around 19 hours:

```
$ massdns -r resolvers.txt -t A -o Sm -s 30000 -c 20 -w massdns-rezults-out.txt
rezults-sorted-uniqued.txt

Concurrency: 30000
Processed queries: 346241655
Received packets: 223350478
Progress: 100.00% (19 h 11 min 58 sec / 19 h 11 min 58 sec)
Current incoming rate: 56 pps, average: 3231 pps
Current success rate: 0 pps, average: 3174 pps
Finished total: 346241655, success: 219415590 (63.37%)
Mismatched domains: 433395 (0.19%), IDs: 0 (0.00%)
Failures: 0: 9.16%, 1: 4.89%, 2: 4.15%, 3: 3.87%, 4: 3.88%, 5: 3.80%, 6: 3.49%,
7: 2.92%, 8: 2.77%, 9: 2.80%, 10: 2.95%, 11: 2.70%, 12: 2.24%, 13: 2.12%, 14:
2.14%, 15: 2.34%, 16: 2.13%, 17: 1.74%, 18: 1.63%, 19: 1.64%, 20: 36.63%,
Response: | Success:               | Total:
OK:       |    204043256 ( 92.99%) |    204234640 ( 91.44%)
NXDOMAIN: |     15372334 (  7.01%) |     15392889 (  6.89%)
SERVFAIL: |            0 (  0.00%) |      2026384 (  0.91%)
REFUSED:  |            0 (  0.00%) |      1696565 (  0.76%)
FORMERR:  |            0 (  0.00%) |            0 (  0.00%)
```

The results from the run include the RRtype and Rdata found, so we need to extract just the RRnames, sort, uniquify and split the results into files of 25MB (as required by the organizers):

```
$ awk '{print $1}' < massdns-rezults-out.txt > just-rrnames.txt
$ grep -v "wildcard" just-rrnames.txt > just-rrnames-no-wildcards.txt
$ wc -l just-rrnames.txt just-rrnames-no-wildcards.txt
  325955018 just-rrnames.txt
  325420844 just-rrnames-no-wildcards.txt     <-- a difference of 534,174
$ LC_ALL=C sort -T . < just-rrnames-no-wildcards.txt > just-rrnames-sorted.txt
$ uniq < just-rrnames-sorted.txt > just-rrnames-sorted-uniqued.txt
$ wc -l just-rrnames-sorted-uniqued.txt
206,142,295 just-rrnames-sorted-uniqued.txt
$ ls -l just-rrnames-sorted-uniqued.txt
[...] 8573089973 Oct  3 23:23 just-rrnames-sorted-uniqued.txt
$ split -C 25M -a 3 -d just-rrnames-sorted-uniqued.txt
```

The problem? We end up with over 3x too many files! We understand the organizers had a limited time to verify the competition results, so the limitation of 100 25MB files comes into play. Many of our discovered RRnames are apparently far longer than we expected:

```
$ ls -lh x* | awk '{print $9 " " $5}'
x000 25M
x001 25M
x002 25M
[...]
x326 25M
x327 966K
```

In order to maximize the number of submissions we could make within the competition's parameters, let's sort our results by length, selecting those that are shortest:

```
$ awk '{print length($0) " " $0}' just-rrnames-sorted-uniqued.txt >
rrnames-with-lengths.txt
$ LC_ALL=C sort -T . -n rrnames-with-lengths.txt >
rrnames-with-lengths-sorted.txt
$ awk '{print $2}' < rrnames-with-lengths-sorted.txt >
rrnames-sorted-by-length.txt
$ split -C 25M -a 3 -d rrnames-sorted-by-length.txt
```

Keeping just the first hundred 25 MB splits from the sorted parent file (x000 to x099), that leaves us with:

```
$ wc -l x*
   1644691 x000    <-- shortest FQDNs
   1432112 x001
   1348541 x002
   [...]
    722520 x097
    708497 x098
    708497 x099    <-- slightly longer FQDNs
89,280,343 total
```

The bottom line: while we identified 206M active subdomains, we would have only been able to find and report 89,280,343 verified FQDNs using DNSDB Flexible Search and `massdns`. This is substantially larger than the 4,060,439 FQDNs the team actually submitted.

The bottom line:

While we identified 206M active subdomains, we would have only been able to find and report 89,280,343 verified FQDNs using DNSDB Flexible Search and `massdns`. This is substantially larger than the 4,060,439 FQDNs the team actually submitted.

10. A Caveat with Respect to Relying on the Supplied massdns resolvers.txt File

DNS queries are normally handled by recursive resolvers operated by your ISP (or your company/school). One reason `massdns` is so fast is that it spreads its resolution work out over many resolvers, rather than just relying on a small set of default resolvers. For the convenience of its users, massdns supplies a list of over 8,000 resolvers. Many users will simply take and use that list at face value, notwithstanding this note in the repo:

> *The repository includes the file `resolvers.txt` consisting of a filtered subset of the resolvers provided by the subbrute project. Please note that the usage of MassDNS may cause a significant load on the used resolvers and result in abuse complaints being sent to your ISP. Also note that the provided resolvers are not guaranteed to be trustworthy. The resolver list is currently outdated with a large share of resolvers being dysfunctional.*

The subbrute project list of resolvers consists of 2014 entries, and was last updated eight years ago. There is no guarantee that all of these resolvers in either of those locations will consistently map domains reliably – they might do so, or they might actually:

- Intentionally block/filter some categories of queries
- Run catch-all records of their own, supplying alternatives responses instead of NXDOMAINs, or
- Send you to dangerous malware-dropping IPs.

Trusting those IPs is kin to hitching alone at night, taking a drink from someone you don't know at a party, or asking a random stranger how you should invest your retirement savings – all risky behaviors with potentially undesirable consequences.

To at least get a sense of where the servers in the `massdns resolvers.txt` lived, we processed that list via Team Cymru's IP to ASN mapping service. ASNs with 25 or more resolvers were seen from:

```
COUNT    ASN    ASN Name
  516  13335 Cloudflare
  309  23393 NuCDN
  233   7922 Comcast Cable
  219   4766 Korea Telecom (KR)
  167   3549 Level3
  152   9318 SK Broadband (KR)
  137   3356 Level3
  123  23089 Hotwire
  118   3215 Orange (FR)
  110    209 CenturyLink
  107  22773 Cox
   82   3462 Hinet (TW)
   81  16276 OVH (FR)
   75  23969 TOT (TH)
   72  34939 NextDNS
   70   9299 PLDT (PH)
   67  51167 Contabo (DE)
```

```
 63   7018 AT&T
 61   5617 Orange Polska (PL)
 47  12389 Rostelecom (RU)
 46 272106 Telnet Peru (PE)
 42     NA (non-mapable)
 39  16509 Amazon
 36 262982 Nardi & Cano (BR)
 36  13489 EPM Telecom (CO)
 36   1221 Telstra (AU)
 35   8075 Microsoft
 32   4788 tm.com.my (MY)
 32  15557 SFR (FR)
 30    701 Verizon
 30  45899 VNPT (VN)
 29  36994 Vodacom (ZA)
 29  24940 Hetzner (DE)
 29  20115 Charter
 28 266935 Centrosulnet (BR)
 28  14061 DigitalOcean
 27   7470 True Internet (TH)
 25   4713 NTT (JP)
 25   3352 Telefonica (ES)
 25  30722 Vodafone (IT)
```

There are other lists of available resolvers you may want to review, including https://github.com/trickest/resolvers – checking the trickest combined `resolvers.txt` (40,839 entries), ASNs with 100 or more entries were as follows:

```
COUNT    ASN    ASN Name
 4591   13335  Cloudflare
 3588  397213  Neustar
 2935  397218  Neustar
 2731  397224  Neustar
 2558  397215  Neustar
 2557  397220  Neustar
 2462  397238  Neustar
 2274  397235  Neustar
 1683  397231  Neustar
 1645    7922  Comcast
  730   34939  NextDNS
  688   22773  Cox
  661  397225  Neustar
  661  397219  Neustar
  653  397233  Neustar
  610    4766  Korea Telecom (KR)
  570  397232  Neustar
  483    3462  Hinet (TW)
  440   12389  Rostelecom (RU)
  382   23393  NuCDN
  361   16276  OVH (FR)
  342    9299  PLDT (PH)
  291   21342  Akamai (NL)
```

```
287    9318  SK Broadband (KR)
268   17488  Hathway (IN)
244    3215  Orange (FR)
218     209  CenturyLink
206   51167  Contabo (DE)
205    5617  Orange Polska (PL)
191    3549  Level3
189    7018  AT&T
176   10036  DLIVE (KR)
150   13489  EPM Telecom (CO)
147    3356  Level3
139   12874  Fastweb (IT)
133    3320  Deutsche Telekom (DE)
131    4713  NTT (JP)
128    6724  Strato (DE)
124   45899  VNPT (VN)
124   45102  Alibaba (CN)
121   24940  Hetzner (DE)
116    3216  Vimpelcom (RU)
111    3786  DACOM (KR)
106   14061  DigitalOcean
103   36994  Vodacom (ZA)
102    8359  MTS (RU)
101   23089  Hotwire
101   12479  Orange Espagne (ES)
101   10796  Charter
100    8560  IONOS (DE)
100    3352  Telefonica (ES)
```

Note that the competition specifically mentioned using Google's 8.8.8.8 resolver, but for the interest of performance and not expecting the results to be significantly different among the set of resolvers we chose, we opted to use this list from trickest.

11. Finding and Removing Deep Wildcards

As we've previously mentioned, regular wildcard domains will resolve any hostname part you choose to use. The contest strictly forbade "exploiting" them, yet actually includes some wildcard domains in the initial seed domain set, perhaps as a bit of a tripwire/boobytrap? For example, the contest list of "seed" domains includes `aax.com`. If we probe the live DNS for a couple of random test hostname on top of that domain, we see:

```
$ dig afnapiufniaufaof.aax.com +short
aax.com.
24.199.73.23
$ dig dont-let-joe-pick-the-pizza-toppings.aax.com +short
aax.com.
24.199.73.23
```

Those two completely random hostnames successfully resolve – in fact `<anything>.aax.com` will successfully resolve since `aax.com` is a wildcarded 2nd-level domain. For the contest, we should not report any results for it, due to the risk of being disqualified, even though some `aax.com` subdomains are actually "real." Out of an abundance of caution, we've skipped enumerating this domain and other wildcarded effective 2nd-level domains completely.

Other domains MAY be associated with "deep wildcards." That is, the registrable domain may not be a wildcard, but some subdomain under the registrable domain may be. For example, `force.com` is not a wildcard at the registrable domain level, but one if its subdomains is:

```
$ dig snvosimdosdv.force.com +short
[nothing returned]
```

But `ap14.force.com` (a subdomain of force.com) is a "deep" wildcard since `<anything>.ap14.force.com` will resolve:

```
$ dig he-always-orders-a-meat-lovers-pizza.ap14.force.com +short
ap14.force.com.
ap14-syd.force.com.
ap14-syd.syd.r.force.com.
13.210.4.196
13.210.6.44
13.210.4.106
```

There's no way to tell "just by looking" whether a FQDN is or isn't a wildcard. You'll need to actively probe the domain with a random hostname to find out. Normally random test subdomains won't resolve if a domain isn't a wildcard, but will resolve if a domain is a wildcard.

For example, consider the domain `municipalite.saint-paul-de-la-croix.qc.ca`. That's a real Canadian municipal website:

However, this FQDN is currently also a "deep wildcard." Virtually anything prepended to that FQDN will resolve:

```
$ dig leaves-have-fallen.municipalite.saint-paul-de-la-croix.qc.ca
[...]
leaves-have-fallen.municipalite.saint-paul-de-la-croix.qc.ca. 14400 IN A
108.163.144.50
[...]
```

If we remove one label from that name and try checking the now-reduced name, the now-reduced name is not a wildcard. Here it returns NXDOMAIN for a test query:

```
$ dig and-frost-is-around-the-corner.saint-paul-de-la-croix.qc.ca
[...]
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 35800
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1
[...]
```

If we remove yet another label and try probing `qc.ca` with a random domain, it's also not a wildcard, nor is the bare TLD (`ca`). The only wildcard is the FQDN. This is an example of a "deep wildcard" that we can only detect by active probing. We went from a "most specific" name to a "most general" name, and testing allowed us to quickly confirm that the FQDN was a "deep" wildcard.

Other times you may want to work "in the opposite direction" (e.g., from the TLD on out). For example, any domain in the `ph`, `ws`, or `vg` TLDs will resolve because there is a TLD-wide wildcard covering those domains:

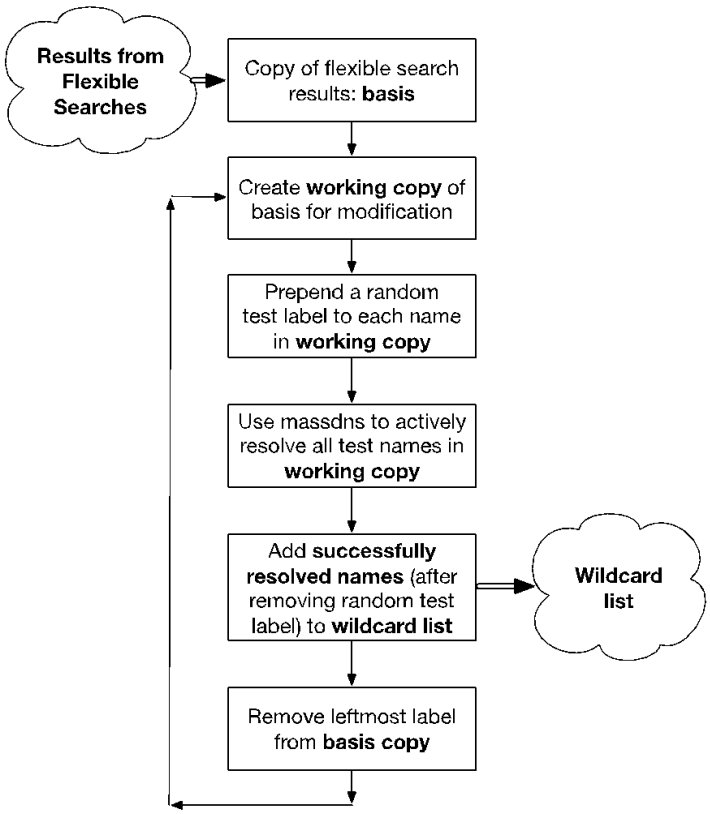
```
$ dig it-never-snows-in-ph.does-it.ph
[...]
it-never-snows-in-ph.does-it.ph. 86400 IN A   45.79.222.138
```

Now some good news: we may not need to worry about handling "deep wildcard" domains. When asked if "deep wildcards" need to be eliminated from submitted domains the administrators stated that "Wildcards were handled at domain level, and have been removed. However, wildcards at nested level were not removed at the

time of processing, as nested wildcard subdomains are very unpredictable and difficult to identify." Given that clarification, we assume that any subdomains surviving validation, whether potentially “deep wildcards” or not, are acceptable and will "count."

If deep wildcards do in fact need to be identified and removed, the process for doing so would be what's shown on the following diagram (Figure 3):

Figure 3. Conceptual Process of Building List of "Deep" Wildcards



Besides copying files, we need two new commands to run that algorithm:

- A command to trim the leftmost label off each name (`trim-one.py`, see Appendix D).
- A way to tack on a randomly-generated label for testing purposes (`generate-wildcard-probes.py`, see Appendix E).

A sample run illustrating the process from the above diagram is shown in Appendix F.

Assuming you've followed the process shown in Appendix F and have extracted all wildcard domains, you can then strip the random prepended label, sorting and uniquifying the wildcard domains.

```
$ cat wildcards-that-resolve-*.txt > rollup.txt
$ awk '{print $1}' < rollup.txt > rollup2.txt
$ ./trim-one.py < rollup2.txt > rollup3.txt
$ LC_ALL=C sort -T . -u < rollup3.txt > all-wildcards-from-processing.txt
```

```
$ wc -l all-wildcards-from-processing.txt
170,410,917 all-wildcards-from-processing.txt
```

That's a lot of deep wildcards! In some cases, it is virtually certain that we have multiple more-specific deep wildcards under a more general wildcard. Before we use that list to remove wildcards from our Flexible Search results, let's remove "redundant wildcards."

To understand what we mean by "redundant wildcard," imagine we have a deep wildcard domain such as:

> **hostname.example.com**

If that's a deep wildcard, we might also have related wildcards "based on top of that name," e.g.:

> blahblah.**hostname.example.com**
> whatever.**hostname.example.com**
> somestuff.foobar.**hostname.example.com**
> moreandmoreandmore.blahblah.**hostname.example.com**

If we're already matching and removing `<anything>.hostname.example.com`, we usually won't also need to remove more specific wildcards based "on top" of that more general wildcard. So let's make our lives easier and remove those redundant deep wildcards. To get the names in the right hierarchical order to do, we first need to reverse the wildcard patterns by label. We'll use `rev-dom-large.py` (Appendix G) to do that:

```
$ rev-dom-large.py < all-wildcards-from-processing.txt >
all-wildcards-from-processing-reversed.txt
```

We'll now sort our names.

When performing this in a Linux environment, we must use the **`LANG=en_US.ascii`** and **`-n`** sort options to get the sort order we need. To understand the issue here, note that the Linux sort command actually ignores some characters when sorting, and "inconveniently orders" other punctuation characters (particularly dot, dash, and underbar) by default. The closest-to-optimum sorting order we can get will be by above listed options with `sort`:

```
$ LANG=en_US.ascii sort -n -T . < all-wildcards-from-processing-reversed.txt >
all-wildcards-from-processing-reversed-sorted.txt
```

We'll then use another piece of custom Python3 code, `remove-unneeded-wildcards.py`, (Appendix H) to eliminate redundant wildcards.

```
$ ./remove-unneeded-wildcards.py <
all-wildcards-from-processing-reversed-sorted.txt > first-past-removing.txt

$ wc -l all-wildcards-from-processing-reversed-sorted.txt first-past-removing.txt
 170,410,917 all-wildcards-from-processing-reversed-sorted.txt
   4,605,094 first-past-removing.txt
```

We now have a reduced wildcard file that's far smaller than it was (170,410,917 / 4,605,094 = 37.0). This will really speed up the process of removing wildcards from our results!

Next we need to remove those wildcards from our set of `206,142,295` raw matches. Conceptually, we could try using `grep -v -f` (or the rip grep equivalent), but memory constraints could mean that neither will handle even our now-dramatically-reduced number of matches. So, we'll employ yet one more custom piece of Python3 code, `match-and-drop.py` (Appendix I), instead.

```
$ ./match-and-drop.py < names-to-test.txt > left-after-wildcard-removal.txt
$ wc -l names-to-test.txt left-after-wildcard-removal.txt

 206,142,295 names-to-test.txt
  29,408,813 left-after-wildcard-removal.txt
```

This was another very successful reduction: `206,142,295 / 29,408,813 = 7.00`

We still have 29.4 million names left. Where do we find them? That is, are they all in just a few effective 2nd-level domains? Let's check. `2nd-level-dom-large` reduces FQDNs to just effective 2nd-level domains using the Public Suffix List. A copy of the `2nd-level-dom-large` script can be seen in Appendix J.

```
$ 2nd-level-dom-large < left-after-wildcard-removal.txt >
left-after-wildcard-removal-2ld.txt
$ sort left-after-wildcard-removal-2ld.txt >
left-after-wildcard-removal-2ld-sorted.txt
$ uniq -c < left-after-wildcard-removal-2ld-sorted.txt | sort -nr >
left-after-wildcard-removal-2ld-sorted-uniq.txt

$ more left-after-wildcard-removal-2ld-sorted-uniq.txt
 787227 telekom.hu
 786459 wsp.com
 783232 spectrum.com
 776231 orange.pl
 774486 hostedemail.com
 773331 vodafonedsl.it
 762102 wd2go.com
 757827 pushy.io
 754426 rogers.com
 737416 115.com
 730292 your-server.de
 691274 xiaoeknow.com
 605338 mesh.ad.jp
 595443 telefonica.de
 542484 blogger.com
 444801 hp.com
 426079 lagou.com
 406084 oraclecloud.com
 400615 telenor.se
 354219 linodeusercontent.com
 353321 sakura.ne.jp
 349089 zaq.ne.jp
```

```
 326558 cnt-grms.ec
 303808 dynamics.com
 271458 16clouds.com
 [etc]

$ wc -l left-after-wildcard-removal-2ld-sorted-uniq.txt
1765777 left-after-wildcard-removal-2ld-sorted-uniq.txt
```

That may seem like a surprisingly-large number of effective 2nd-level domains to see – after all, we started with just 14,917 "seed" domains, right? And we even excluded some of those! The key is taking a closer look at some of the initial "seed" domains – some of those weren't delegation points, they were effective 2nd level domains in their own right, representing requests for entire hierarchies of domains. For example, consider `.jp`. The seed domains specified there included multiple domains from the public suffix list:

```
ac.jp
ad.jp
go.jp
ne.jp
or.jp
sakura.ne.jp
```

Each of those effective TLDs might have many registerable domains under them.

We saw a total of 97 effective top level domains included among the original list of 14,917 "seed" domains, see Appendix K. Some might question the appropriateness of those "seed" domains, much as if someone had proposed an entire gTLD or ccTLD as an initial "seed."

12. Challenge Administrator Evaluation of Contestant-Submitted Results

As interesting as it has been to think about ways to find and optimize contestant responses to this contest, it is also worth considering the evaluation process from the perspective of the challenge administrators. If contestants are under time pressure, the administrators would likely be under even more time pressure:

- Contestants may wait to upload their results until the last possible instant
- Contestants want to know "who won" as soon as possible

The challenge evaluators needed to ensure that:

- All team-submitted domains are based on one of the assigned "seed" domains
- Domains resolved for at least one of {"A" record, "CNAME", "MX", or "TXT"} using 8.8.8.8
- All team-submitted domains were not DNS wildcards (although apparently not strictly enforced)
- No more than 1,000 bad domains were submitted
- No more than 100 files of domains, each larger than 25MB were submitted

Sampling to the Rescue? Normally when faced with constraints of this sort, an evaluator's mind might turn to sampling: they wouldn't need to exhaustively look at every domain to get a fairly-accurate estimate of a population's characteristics, they could just draw a random sample from each contestant's submission, and test those to get an estimate for the population as a whole. This was not employed for this event.

Precompiled Results? Alternatively, since evaluators have the advantage of knowing the "seed domains" they were going to assign in advance, they could potentially spend as much (or as little) time as they liked accumulating acceptable responses for the domains they planned to assign. Assuming they're able to get an exhaustive (or nearly-exhaustive) list of acceptable responses, the evaluator's responsibilities then devolve to:

- Tallying up hits they anticipated seeing, and
- Looking at the remainder, evaluating any heretofore-unknown FQDNs that contestants may have uncovered

One factor to consider here: FQDNs found in advance might actually "go stale" on the evaluators in the interval between when they're found prior to the challenge and when they're subsequently used for evaluation.

Strict Compliance with Contest Rules? Evaluators might also have elected to emphasize strict compliance with contest rules, selectively scanning a submission for evidence of obvious wildcards or non-resolvable domains. If one only needed to find 1,000 "bad" domains to disqualify a team's entire submission, this might be a highly efficient way of excluding submissions from consideration, albeit one has the potential to leave participants feeling disgruntled. Again, this does not appear to have been the approach employed, and is perhaps based on a miscalculation about the overall number of domains that would be submitted. A percentage-based disqualifier might have been a preferable approach.

13. Conclusion and Recommendations For "Deep Wildcards" Moving Forward

This has been a surprisingly interesting and challenging problem to consider, with many subtleties we hadn't anticipated when we began our review.

	2023 DEF CON-Winning Run Using "subfinder" with DNSDB (365- day time fence) (baseline comparator)	**DNSDB Flexible Search (10 Streams) 90-day time fence**
Raw FQDNs	(not tracked)	368,166,151
FQDNs After filtering	4,060,439	206,142,295
Submittable (or submitted) FQDNs	4,060,439	89,280,343
Likely true unique subdomains	n/a	29,408,813

If we weren't constrained in terms of the number of results submitted, and if we didn't need to remove wildcards, our 90-day time fenced Flexible Search runs returned 368,166,151 raw FQDNs. With the insight that nearly a hundred of the assigned “seeds” were actually effective TLDs, we could have expanded those effective TLDs and found still more results.

We have developed a new appreciation for mass parallel active resolution, as it is a challenging problem in its own right. We find ourselves wondering about whether the set of resolvers delivered in `resolvers.txt` with `massdns` require closer scrutiny, and some of the other widely-available lists of open resolvers raise other interesting questions as we begin to look more closely at them.

We more fully appreciated the "submit no more than 100 files, each no more than 25MB in size" constraint. When it was applied, even picking our submissions selectively, we could only have sent in 89,280,343 FQDNs.

Removing "deep" wildcards is by far the most interesting and uniquely challenging aspect of this challenge. We demonstrated an approach that worked for this, taking the number of domains down to 29,408,813 FQDNs.

After realizing that 97 of our initial seed domains could potentially have been expanded out to thousands of individually queryable registerable domains, we might find still more FQDNs via Flexible Search, but we're comfortable leaving this project where it currently lies.

Seeing the extent to which deep wildcards exist makes us wonder if the time has come to individually track the wildcard status of each FQDN in passive DNS. Rather than needing to make bulk probes of publicly accessible resolvers, passive DNS services could organically track the wildcard status of each (RRname, RRtype) combination from the time the (RRname, RRtype) is first seen. We’ll need to investigate this further.

Wildcards are potentially very useful constructs, but they're also unquestionably very dangerous. They can be leveraged to generate uncacheable floods of reflective traffic. They can be used to spoof reputation-impairing domains such as CSAM-related FQDNs, terrorism-supporting FQDNs, or phishing-related FQDNs, even if the underlying infrastructure has nothing substantive to do with any of those areas. Wildcards need to be carefully tracked as potential rogue weapons of (cyber) mass destruction.

We would like to thank the organizers of the "ReconAcharya Subdomain Enumeration Challenge" at the 2023 DEF CON Recon Village. This truly was a terrific subject for a cybersecurity competition highlighting the importance and challenges with understanding subdomains in DNS. Thanks also to the members of the DomainTools-affiliated team and all other teams who participated in this challenge.

Appendix A. Voluntarily-Excluded Domains

18f.gov
1clickvpn.net
1stkissmanga.io
1xbet.com
1x-bet.in
360buy.com
360buyimg.com
360totalsecurity.com
46to.com
4kporn.xxx
91porn.com
91porny.com
aax.com
accenture.com
acl.gov
adda247.com
admanmedia.com
afafb.com
afip.gob.ar
afirm.mil
af.mil
afpims.mil
africom.mil
agid.gov.it
agingstats.gov
ah.mil
aids.gov
ai.mil
aips-services.com
airmap.com
alfabank.ru
alipay.com
alipay.com.cn
allcommonstories.com
allmovieshub.day
allporncomic.com
amaro.mil
analdin.xxx
analvids.com
anses.gob.ar
antispamcloud.com
anysex.com
ap.gov.in
apnic.net
appcard.com
apps.mil
apta.gov.cn
archdaily.com
areeba.com.gn
argentina.gob.ar
arlingtoncemetery.mil
army.mil
asapp.com
asbca.mil
ascgov.com
asextranet.com
ashamedstep.com
asianporn.li
asnbank.nl
ato.gov.au
aviationweather.gov
avira.com
avira-update.com
avira-vpn.com
axasecurity.com
axisbank.co.in
axisbank.com
azurefd.us
b9good.com
babybus.com
babycenter.com
banamex.com
bananarepublic.com
bancainternet.com.ar
banco.bradesco
bancobrasil.com.br
bancodevenezuela.com
bancoestado.cl
bancogalicia.com.ar
bancointer.com.br
bancolombia.com
bancopan.com.br
bancosantander.es
bankbazaar.com
banking.nationwide.co.uk
bankmandiri.co.id
bankmellat.ir
bankofamerica.com
bankofchina.com
bankrate.com
banks-sadler.com
banquepopulaire.fr
barclaycardus.com
barclays.co.uk
beijing.gov.cn
bestialitysextaboo.net
bestjavporn.com
bigbank.at
bigbank.de
bigbank.ee
bigbank.eu
bigbank.fi
bigbank.lt
bigbank.lv
bigbank.nl
bigbank.se
bigcdn.cc
bigcommerce.com
bihar.gov.in
bime.io
biosecuritykwetsbaarheidsanalyse.nl
biosecurityselfscan.nl
biosecurityvulnerabilityscan.nl
biosecurityzelfscan.nl
bldrdoc.gov
blue-extra.com
bom.gov.au
breitbart.com
bruteprotect.com
bt1207ka.top
budget.mil
bureaubiosecurity.nl
c6bank.app
cac.mil
ca.gov
caisse-epargne.fr
caixabank.es
caixa.gov.br
camwhoresbay.com
camwhores.tv
capitalone360.com
capitalonebank.com
capitalone.ca
capitalonecards.com
capitalonecareers.co.uk
capitalone.com
capitalone.co.uk
capitalonegslbex.com
capitaloneshopping.com
capitalone.tz
capmed.mil
cap.mil
cardekho.com
cardsmobile.ru
casinohuone.com
cathaypacific.com
ccdi.gov.cn
cdc.gov
centcom.mil
centrumgoodgovernance.nl
cept.gov.in
chapmanganato.com
chefishoani.com
chibabank.co.jp
chick-fil-a.com
childrensplace.com
china-embassy.gov.cn
chinatax.gov.cn
chumbacasino.com
cic.gc.ca
ciima-club.pics
citibankonline.com
clegc-gckey.gc.ca
clevelandclinic.org
clickbank.com
cloudappsecurity.com
cloud.gov
cloud.mil
cms.gov
cnipa.gov.cn
codefi.network
code.gov
code.mil
coinbase.com
coinbase.pro
commbank.com.au
commissiecorporategovernance.nl
commissievanaanbestedingsexperts.nl
company-information.service.gov.uk
config-security.com
consumerfinance.gov
corpgov.nl
court.gov.cn
covidbehaviouralchallenge.nl
cra-arc.gc.ca
crazyporn.xxx
credit-agricole.fr
creditkarma.ca
creditkarma.com
creditonebank.com
crmforce.mil
ctbcbank.com
cto.mil
cuidadodesalud.gov
customs.go.kr
cybercom.mil
cyberghostvpn.com
cyber.mil
cybersecurityalliantie.nl
cybersecuritycouncil.nl
cybersecurityraad.nl
daft.sex
dailycaller.com
danskebank.com
danskebank.dk
darpa.mil
data.gov
dataprev.gov.br
datarotterdam.nl
dating.com
dau.mil
dbankcdn.cn
dbankcdn.com
dbankcdn.ru
dbankcloud.asia
dbankcloud.cn
dbankcloud.com
dbankcloud.ru
dbankedge.cn
dbankedge.net
dc3.mil
dcaa.mil
dcard.tw
dcma.mil
dcoe.mil
dcsa.mil
dds.mil
debank.com
decipherinc.com
defenseculture.mil
defenseinnovationmarketplace.mil
defenselink.mil
defense.mil
delta.com

deomi.mil
deps.mil
dergipark.org.tr
devolksbank.nl
dfas.mil
dhl.com
dhra.mil
dhs.gov
dia.mil
dib.mil
digital.go.jp
digitalgov.gov
dimoc.mil
disa.mil
discovercard.com
dispo.mil
diu.mil
dla.mil
dma.mil
doctorpost.net
dodcui.mil
doded.mil
dodig.mil
dodlive.mil
dodmantech.mil
dod.mil
dodsbirsttr.mil
dodtap.mil
dodtechipedia.mil
donaciondeorganos.gov
doodcdn.co
doodcdn.com
dood.re
doodstream.com
dood.video
dood.wf
dood.yt
dooood.com
dotmovies.xyz
doubledowncasino2.com
doublepimp.com
doubleucasino.com
dpaa.mil
dren.mil
drudgereport.com
drugs.com
dsca.mil
dspo.mil
dss.mil
dtdjzx.gov.cn
dtic.mil
dtra.mil
dtsa.mil
duckdns.org
dude6.com
dwl.dnswl.org
dynoquant.com
ebanksepah.ir
ebay.ca
ebay.com
ebay.com.au
ebay.co.uk
ebay.de
ebay.es
ebay.it
ebaystatic.com
ebay-us.com
eb.mil
ed.gov
e-gov.az
egov.kz
egov-nsdl.com
emb-japan.go.jp
empornium.sx
emule.org.cn
enabiz.gov.tr
encipher.io
encrypted-encrypted-encrypted-encrypted-encrypted-encrypted-encrypted.link
epa.gov
epfindia.gov.in
epolice.ir
erothots.co
ero-video.net
esgrevents.mil
esgr.mil
esi.mil
e-taxes.gov.az
eucom.mil
ewaybillgst.gov.in
exporntoons.net
expressvpn.com
faculteitmilitairewetenschappen.nl
familyisland.games
fazenda.gov.br
fda.gov
fedramp.gov
figpii.com
finchvpn.com
fireeye.com
fitness.gov
flexchange.nl
flexport.com
foodsafety.gov
forebet.com
fortinet.com
foundrygov.com
fpo.xxx
freedomaward.mil
f-secure.com
fuckmoral.com
fuckword.club
fujian.gov.cn
fullporner.com
gaycock4u.com
gayforfans.com
gaymaletube.com
gaytor.rent
gazt.gov.sa
gc.ca
gdatasecurity.de
gd.gov.cn
gdzwfw.gov.cn
geenmedical.com
gem.gov.in
gfbzb.gov.cn
gib.gov.tr
gipdatabank.nl
girlshealth.gov
gizmoxxx.com
glevoloo.com
globovideos.com
goadsexchange.com
gobankingrates.com
go.gov.br
go.id
go.jp
go.kr
goodporn.to
gosi.gov.sa
goszakup.gov.kz
gotanynudes.com
go.th
gouv.fr
gouv.qc.ca
gov4nano.eu
govbox.nl
gov-dns.nl
govdns.nl
govee.com
government.nl
gov.huaweicloud.com
gov-img.site
gov-static.tech
govt.nz
grantsolutions.gov
grindr.io
grindr.mobi
grortalt.xyz
gsa.gov
gst.gov.in
gsxt.gov.cn
gtbank.com
gtloli.gay
gujarat.gov.in
hacc.mil
hackenproof.com
hacker101.com
hackerone.com
hackerone-ext-content.com
hackerone.net
hackerone-user-content.com
hackerwatch.org
hackmd.io
hangzhou.gov.cn
hcard.ch
hci.mil
hdfcbank.com
hdporn92.com
hdporncomics.com
hdsex2.com
healthdata.gov
healthfinder.gov
health.gov
health.mil
healthypeople.gov
hebgwyks.gov.cn
heimdalsecurity.com
henan.gov.cn
hentaihaven.xxx
hentaistube.com
hhs.gov
hiv.gov
hoes.tube
hokuyobank.co.jp
holavpn.net
hometax.go.kr
hopkinsmedicine.org
hot-sex-tube.com
house.gov
hpc.mil
hrgworldwide.com
hrsa.gov
hsex.men
huidating.com
hunan.gov.cn
h-vpn.org
hypnotube.com
iam.gov.sa
ibabyp2p.com
ibanking-services.com
ibl-video.site
icann.org
icicibank.com
idencys.nl
igram.io
imhentai.xxx
immigratieennaturalisatiedienst.nl
immigratie-naturalisatiedienst.nl
incometax.gov.in
incometaxindia.gov.in
indexxx.com
indianrail.gov.in
indiapostgdsonline.gov.in
indiapost.gov.in
indiehackers.com
infomil.nl
inherentresolve.mil
inss.gov.br
instreamvideo.ru
insurekidsnow.gov
invideo.io
iot.mil
iporntv.net
ipvideotalk.com
irs.gov
isbank.com.tr
itnc.mil
itopvpn.com
itsecure.co.in
ivideosmart.com
ixxx.com
jasper.ai
javgo.net
javhdporn.net
javxxx.me
jb.mil
jbsa.mil
jcse.mil
jcs.mil
jcu.mil
jecc.mil

jeugdautoriteit.nl
jeugdconnect.nl
jeugdengezin.nl
jeuxvideo.com
jfcom.mil
jieddo.mil
jsf.mil
jszwfw.gov.cn
jten.mil
jtj.mil
kapitalbank.az
kasikornbank.com
keepersecurity.com
keepersecurity.eu
kemdikbud.go.id
kemenkeu.go.id
kerala.gov.in
keybank.com
klickly.com
kmff40.com
kotobank.jp
kucoin.com
kucoin.plus
lacity.gov
lacnic.net
lansweeper.com
lbank.info
lgyy.cc
lkpp.go.id
lloydsbank.com
lloydsbank.co.uk
lobstertube.com
login.gov
lottery.gov.cn
love4porn.com
maharashtra.gov.in
mail.mil
mailspamprotection.com
maria.casino
mariacasino.com
marinahospital.com
marines.mil
maskvpn.cc
maskvpn.org
mastercardbiz.ca
mastercardbiz.com
mastercard.ch
mastercard.com
mastercard.com.au
mastercard.com.br
mastercard.co.za
mastercard.nl
mastercard.ru
mastercard.us
mat6tube.com
maybank2u.com.my
mayoclinic.org
mcafee-cloud.com
mcafee.com
mcafeewebadvisor.com
mcls.gov.ir
mc.mil
mda.mil
meb.gov.tr
medicaid.gov
medicare.gov
medlineplus.gov
mee.gov.cn
menlosecurity.com
mercantilbanco.com
metc.mil
metoffice.gov.uk
mg.gov.br
mhlw.go.jp
michinokubank.co.jp
miit.gov.cn
milfnut.com
militaryonesource.mil
milsuite.mil
mission-electric.in
mobiliteitsbank.nl
modernisinggovernment.com
modernisinggovernment.net
mod.gov.uk
moe.gov.cn
moe.gov.my
moe.gov.sa
mofa.gov.sa
mofcom.gov.cn
mohrss.gov.cn
moi.gov.qa
monobank.com.ua
mountsinaiconnect.org
mountsinaidoctors.org
mountsinaiheartnp.org
mountsinai.org
move.mil
mp.gov.in
mponline.gov.in
msedge.net
multporn.net
my.com
myfritz.net
my.gov.au
my.gov.az
my.gov.ir
myip.com
myip.la
myip.top
myip.wtf
mymedicare.gov
mysymptoms.mil
nalog.gov.ru
namethatporn.com
nanjing.gov.cn
nasa.gov
nat.gov.tw
nationalguard.mil
navy.mil
nazorgjeugd.nl
ncsc.gov.uk
netvantasecurityportal.com
nga.mil
ng.mil
nhs.uk
nicbr.mil
nic.in
nic.io
nic.ir
nic.ru
nic.uk
nih.gov
nipr.mil
nist.gov
njav.tv
nldigitalgovernment.nl
noaa.gov
norad.mil
nordvpn.com
northcom.mil
norton.com
nortonlifelock.com
notube.io
notvpn.io
npc.gov.cn
nro.mil
nsfc.gov.cn
nsfw.xxx
nsin.mil
nsw.gov.au
nta.go.jp
ntp-fireos.com
nts.go.kr
nubank.com.br
nurxxx.mobi
nyc.gov
ny.gov
nypost.com
offensive-security.com
okdiario.com
onhir.gov
onlinevideoconverter.pro
opa.mil
organdonor.gov
osd.mil
pacom.mil
pa.gov
pajak.go.id
palantirgov.com
pandasecurity.com
parivahan.gov.in
passportindia.gov.in
paypal.com
paypalcorp.com
paypal.here
paypal.me
paypalobjects.com
paytmbank.com
pbc.gov.cn
pbebank.com
pbteen.com
pentagon.mil
petalpay.huaweicloud.com
pfpa.mil
pfxcloud.com
pickmee.in.th
pickmee.my
planalto.gov.br
platformduurzaamovenspoor.nl
pmkisan.gov.in
poddamnthatsfunny.com
porn7.net
pornez.net
pornfind.org
pornhat.com
pornhits.com
pornhub.com
pornhubpremium.com
pornkai.com
pornmd.com
pornone.com
pornovideoshub.com
porntn.com
porntrex.com
portaldasfinancas.gov.pt
postbank.de
pppdutchgovernment.nl
premierhero.com
previdencia.gov.br
prostovpn.org
protonmail.ch
protonmail.com
proton.me
protonvpn.ch
protonvpn.com
q-bankplants.eu
qianxun.com
rabobank.com
rabobank.nl
rajasthan.gov.in
rakuten-card.co.jp
rbcroyalbank.com
rc03.oray.com
rd.go.th
reasonsecurity.com
redd.tube
redtube.com
redwap-xxx.com
regiobank.nl
repi.mil
returnyoutubedislikeapi.com
ripe.net
royalbank.com
rs.gov.br
rspamd.com
ruankao.org.cn
rule34.us
rule34video.com
rule34.xxx
rutube.ru
rwssportdag.nl
saasexch.cc
saasexch.com
samr.gov.cn
santander.com.ar
santander.com.br
santander.com.mx
santander.co.uk
sapr.mil
sat.gob.mx
saude.gov.br
save-insta.com

sberbank.ru
sc.gov.cn
schatkistbankieren.nl
scorecardresearch.com
scotiabank.com
scs.gov.cn
scvps.my.id
sd.gov.cn
search.gov
securebrowser.com
secureinternetbank.com
secureworks.com
securityscorecard.com
securityscorecard.io
securitytrfx.com
securitytxt.org
segurosocial.gov
sep.gob.mx
serpro.gov.br
service.gov.uk
sexbjcam.com
sexkbj.com
sextb.net
sgk.gov.tr
shaanxi.gov.cn
shandong.gov.cn
sh.gov.cn
showmax.app
shufflesex.com
sigar.mil
simplereport.gov
singpass.gov.sg
skrbtla.top
skyhighsecurity.com
smartgovernance.nl
smbc-card.com
snaptube.app
socialsecurity.gov
soc.mil
socom.mil
sofsa.mil
softbank.jp
sophos.com
sosial.gov.az
southcom.mil
southwest.com
spaceforce.mil
spamhaus.net
spamhaus.org
spell.run
sp.gov.br
sportingbet.com
spotify.xxx
ssa.gov
ssc.nic.in
staatsexamensnt2.nl
staatsexamensvo.nl
stanbicbank.co.bw
stanbicbank.co.ke
stanbicbank.com.ci
stanbicbank.com.gh
stanbicbank.co.tz
stanbicbank.co.ug
stanbicbank.co.zm
stanbicbank.co.zw
stanbicibtcbank.com
standardbank.cd
standardbank.co.ao
standardbank.com
standardbank.com.br
standardbank.com.na
standardbank.co.mw
standardbank.co.mz
standardbank.co.sz
standardbank.co.za
standardbank.mu
standardlesothobank.co.ls
starlingbank.com
state.gov
stlukeshospitalnyc.org
stopbullying.gov
stratcom.mil
stroopjemouwop.nl
studeermeteenplan.nl
studentaid.gov
sunat.gob.pe
sunporno.com
swannsecurity.com
swapcard.com
swisscomras.ch
swisstrustdir.ch
swypeconnect.com
synchronycredit.com
tapevents.mil
tax.gov.ir
tax.service.gov.uk
tdbank.com
teamstercardnow.com
technical-service.net
tejaratbank.ir
telangana.gov.in
texas.gov
tfl.gov.uk
theporndude.com
theunioncard.com
thor-pom.com
time.gov
tktube.com
tlsext.com
tn.gov.in
totinternet.net
transbank.cl
trendyporn.com
tricare.mil
tube8.es
tube8.fr
tubebuddy.com
tubeload.co
tubemogul.com
tubesafari.com
tubetraffic.com
turkiye.gov.tr
uidai.gov.in
unc.mil
unibet.casino
unibet.com
unibet.fr
unibet.me
united.com
unitedincome.com
united-security-providers.ch
universal-credit.service.gov.uk
un.org
un-psf2017.com
up.gov.in
urporn.com
usa.gov
usbank.com
uscg.mil
uscis.gov
usda.gov
usembassy.gov
usfk.mil
usgovcloudapi.net
usgovcloudapp.net
usgovtrafficmanager.net
usgs.gov
usmc.mil
ustranscom.mil
usuhs.mil
uyap.gov.tr
va.gov
vcahospitals.com
veeamgov.com
verwijzingsportaalbankgegevens.nl
vf.force.com
vidhd.fun
vidhub.top
vikiporn.com
virginia.gov
visa.com
vlibras.gov.br
vote.gov
wa.gov
warriorcare.mil
watchporn.to
weather.gov
webhd.cc
web.mil
westernunion.at
westernunionbank.com
westernunion.be
westernunion.ca
westernunion.ch
westernunion.com
westernunion.com.au
westernunion.co.nz
westernunion.co.uk
westernunion.de
westernunion.dk
westernunion.ee
westernunion.es
westernunion.fi
westernunion.fr
westernunion.gr
westernunion.ie
westernunion.it
westernunion.lu
westernunion.nl
westernunion.no
westernunion.pl
westernunion.pt
westernunion.se
whatismybrowser.com
whatismyipaddress.com
whatismyip.host
whatismyip.li
whs.mil
wizitales.com
womenshealth.gov
worldbank.org
wqjbldnnceroue.com
wsj.com
wsj.net
wtfismyip.com
wuqianka.top
www.gob.mx
www.gob.pe
www.gov.cn
www.gov.uk
www.nhs.uk
x63a.com
x666x.me
xfantazy.com
xfreehd.com
xhqxmovies.com
xmoviesforyou.com
xnxx.com
xnxx.gold
yadongtube.net
yandex-bank.net
yavtube.com
yeswehack.com
yhvod.net
ylwt33.com
yougov.com
youjizz.com
youngjoygame.com
youngle.site
youporn.com
yourbestjournal.com
youtube.com
youtubekids.com
youtube-nocookie.com
zakupki.gov.ru
zbporn.com
ziraatbank.com.tr
zj.gov.cn
zjzwfw.gov.cn
zoomgov.com
zorgcijfersdatabank.nl
zscalergov.net

Appendix B. Results Returning 1,000,000+ Results from Flexible Search for the 90-Day Time Fenced Period

2814313 pages.dev
2737145 pushy.io
1928841 aips-services.com
1726896 cloudflare.net
1676041 workers.dev
1640861 aiv-cdn.net
1443267 oray.com
1348796 blogger.com
1336175 duckdns.org
1258083 netflix.com
1183833 ibm.com
1144211 or.jp
1131278 garenanow.com
1103292 my.id
1091477 rogers.com
1079824 16clouds.com
1060762 yahoo.com
1060163 google.com
1059926 ac.id
1054081 hubspotemail.net
1046504 rudderstack.com
1031178 aws.dev
1026630 onmicrosoft.com
1025496 ne.jp
1021613 ac.uk
1017682 sohu.com
1015291 a2z.com
1010200 seek.com.au
1008873 apple-dns.net
1008412 your-server.de
1008021 goo.gl
1007757 azure.com
1007373 lifeomic.com
1007367 cisco.com
1006044 telekom.de
1005743 infomaniak.ch
1005286 go.th
1004381 coinbase.pro
1004337 ac.ir
1004190 opera.com
1003652 intuit.com
1003517 cloud.gov
1003087 altervista.org
1003007 qq.com
1002439 stripe.com
1002341 secureserver.net
1002326 adobe.com
1002293 army.mil
1002019 sage.com
1001833 18f.gov
1001500 washington.edu
1001496 akadns.net
1001197 sharepoint.com
1001183 microsoft.com
1001007 opera-mini.net
1000853 amazonaws.com
1000697 comcast.com
1000671 klarna.net
1000639 epam.com
1000553 windows.net
1000534 nasa.gov
1000526 sony.com
1000511 bidswitch.net
1000510 liquidweb.com
1000435 joins.com
1000419 shawcable.net
1000318 fastly.net
1000266 noon.com
1000251 miui.com
1000245 go.jp
1000244 aliyun-inc.com
1000242 telekom.net
1000229 telstra.com
1000227 amazonaws.com.cn
1000214 orange.pl
1000210 ringcentral.com
1000210 amazon.com
1000183 iherb.com
1000168 playstation.net
1000163 ingka.com
1000135 dailymotion.com
1000133 grammarly.com
1000130 zoomus.cn
1000128 salesforce.com
1000122 rokt.com
1000121 wal-mart.com
1000119 infomaniak.com
1000116 upstart.com
1000116 dhl.com
1000115 azure.net
1000109 telefonica.de
1000109 medallia.com
1000107 zynga.com
1000107 mathworks.com
1000103 ikea.com
1000101 fc2.com
1000098 orange.fr
1000098 mgid.com
1000090 tinderops.net
1000080 outfra.xyz
1000078 unity.com
1000074 grabcad.com
1000070 resellerclub.com
1000068 datastax.com
1000059 gitlab.com
1000059 163.com
1000056 semrush.net
1000056 seek.com
1000053 riotgames.com
1000052 force.com
1000050 bigfishgames.com
1000045 gitlab.net
1000045 bbc.co.uk
1000043 alibaba-inc.com
1000040 allianz.de
1000039 telekom.hu
1000039 scopely.io
1000039 epicgames.com
1000037 mastercard.com
1000037 infobip.com
1000037 azurewebsites.net
1000035 atg.se
1000034 nintendo.net
1000031 gravitational.io
1000030 opera.technology
1000030 ikea.net
1000030 32red.com
1000029 myteksi.net
1000028 usda.gov
1000027 fiverr.com
1000026 yandexcloud.net
1000026 proxad.net
1000025 komoot.de
1000024 perfops.net
1000024 docusign.com
1000024 arubanetworks.com
1000023 mercedes-benz.com
1000023 kiwi.ki
1000021 telenor.se
1000020 ebay.com
1000018 printful.com
1000017 tstaging.tools
1000017 indeed.tech
1000015 progressive.com
1000015 dell.com
1000014 viator.com
1000014 shopee.co.id
1000014 messagebird.com
1000013 omtrdc.net
1000012 shipt.com
1000011 superawesome.tv
1000011 secureworks.com
1000011 intercom.com
1000010 wsp.com
1000010 rev.ai
1000010 lalamove.com
1000010 hostedemail.com
1000010 godrej.com
1000009 nba.com
1000008 xiaoeknow.com
1000008 uberinternal.com
1000008 pindrop.io
1000008 picpay.com
1000008 elastic-cloud.com
1000008 dnswl.org
1000008 dashlane.com
1000007 sbb.ch
1000007 milanuncios.com
1000007 idcsmart.com
1000007 apigee.net
1000007 adobe.io
1000006 totvs.com.br
1000006 otm-r.com
1000006 opsgenie.com
1000005 thenorthface.com
1000005 elastic.co
1000004 magento.cloud
1000004 k8s.io
1000004 apple.cn
1000003 wd2go.com
1000003 newrelic.com
1000003 imunify.com
1000003 e2ro.com
1000003 aliyun.com
1000002 vodafonedsl.it
1000002 t-mobile.com
1000002 spectrum.com
1000002 snooguts.net
1000002 oculus.com
1000002 kpn.net
1000002 infojobs.net
1000002 ebaystatic.com
1000001 wolt.com
1000001 trustpilot.com
1000001 startpage.com
1000001 sberdevices.ru
1000001 ok.ru
1000001 hiltonbusinessonline.com
1000001 fedoraproject.org
1000001 discoveryplus.com
1000001 algolia.io
1000000 vidyard.com
1000000 scbs.ch
1000000 remotewd.com
1000000 huobi.com
1000000 classdojo.com
1000000 acorns.com
1000000 2o7.net
1000000 115.com

Appendix C. Flexible Search Queries Returning Zero Results for the 90-Day Time Fenced Period

1813-2013.nl
1stream.eu
1x-bet.in
1xlite-071412.top
321naturelikefurfuroid.com
8x8.com
8x8.spot
ad-delivery.net
adtrace.online
adtraffic.agency
advatravel.com
aehnet.nl
agacelebir.com
agentschap.com
agnoetecluster.uno
alertonlinequiz.nl
alexamericansystems.com
amaro.mil
americanalliant.com
american-systems.com
american-systems.org
americansystems.org
americansystemsuniversity.com
amunfezanttor.com
anahitagirted.uno
analyticssystems.net
analyt.ir
answerforbusiness.nl
anti-racisme.nl
anyaptitude.cc
appboy-images.com
app.mobile
areeba.com.gn
arfbiqgapb.com
arnholdinstitute.org
arsmtp.com
asc.name
ascvpc.com
asextranet.com
ashamedstep.com
aspirit.art
astivysauran.com
asvpc.com
aswpsdkus.com
ausoafab.net
av19.org
b2o6b39taril.com
b9good.com
backonego.xyz
beepinging.org
besmeargleor.com
betotodilea.com
billboard.comcloudkarafka.com
bime.io
blue-extra.ch
blue-extra.com
blue-games.ch
bluegames.ch
blue-gaming.ch
bluegaming.ch
blue-gaming.com
blue-league.ch
blueleague.ch
blue-league.com
blue-play.ch
blue-plus.ch
blue-streaming.ch
bluestreaming.ch
blue-streaming.com
blue-tv.ch
bluetv.ch
blue-xtra.com
blue-zoom.ch
blunksdaler.uno
boffosgemeled.digital
bp.blogspot.com
brandweercn.nl
braze-images.com
broadsimp.site
broadsview.site
buqkrzbrucz.com
burningapril.info
burningmay.info
byairbnb.com
cajunecch.guru
capitalone.tz
casareal.nl
cbssports.com
cdn4image.com
cdn-cookieyes.com
cdngc.net
cdntechone.com
cdnvideo.ru
cedexis.net
cedexis-radar.net
centrumgoodgovernance.nl
characterai.io
chaya.makeup
ciima-club.pics
cilishenqi.top
clicknupload.to
clienttons.com
cloudapp.net
cloudflareok.com
cloudfront.net
cnf-u45p-alitools.com
cobinhood.exchange
commissiedewinter.nl
communicatieplein.nl
coonandeg.xyz
coopcowboys.com
covidbehaviouralchallenge.nl
creatieveindustrieinbeeld.nl
crisis-management.nl
crjpgate.com
crmentjg.com
cryptography.io
cumgadbpovr.com
cynicaltechnology.com.np
czboox.xyz
d2evh2mef3r450.cloudfront.net
datarotterdam.nl
datatechone.com
datatechonert.com
dcewgduq26.net
dcoe.mil
deadelaarisgeland.nl
deaftrapop.nl
deandereoverheid.nl
degeitwordtgemolken.nl
deliveroo-streams.net
deskundigenregister.nl
dfearinglestp.info
dienstuitvoeringonderwijs.nl
digitalaccess.com
dikgames.com
diromalxx.com
dispo.mil
dm20.biz
dm530p.net
dnacdn.net
domeinen.org
dotnxdomain.net
dutchroyalhouse.nl
duurzamelandbouw.nl
dynoquant.com
e5.sk
e67repidwnfu7gcha.com
eggvod.cn
emojicombos.com
emsolutionsinc.com
en16001.nl
eops.nl
erakzeo.cfd
essworld.net
evaluatiecaribischnederland.nl
everynoise.com
expressobutiolem.onion
ezcgojaamg.com
fagawdasv.com
fallguys.com
fanduel.design
fbcdn.net
fcpfth.xyz
fedapush.net
filedownload-csw-lenovo.com
flacsfor.me
flerap.com
fleraprt.com
flexsparen.nl
foliosedunlin.guru
forlifecode23.com
forprimeapeon.com
foxmodeq.com
fpbns.net
fpdjeugd.nl
fpnpmcdn.net
fripth.xyz
gaesataigal.uno
gametu.net
getsthis.com
ghabovethec.info
ginpithed.live
glassdoor.app
glersakr.com
glevoloo.com
globalforumoncyberexpertise.nl
gml-grp.com
gmxvmvptfm.com
godpvqnszo.com
googapis.org
gpuqizoz.com
grab.com
greentransporthub.org
grortalt.xyz
gulsachpyrexia.uno
gu-st.ru
hazanuttriazo.life
healthandhealingny.com
healthandhealingny.org
heil-hitler.nl
hetbegintmetaal.nl
hetfinancieelloket.nl
hexagon-analytics.com
hhbypdoecp.com
highload.to
hinkhimunpra.info
his-ict.nl
hivepinger.com
hiyobi.me
hoogspanningsverbindingen.nl
hoogspanningsverbinding.nl
hotpics.mom
iaadd.cn
iasa2011.nl
icdns.net
id6.me
idisign.ch
iezxmddndn.com
ifconfig.co
igram.io
ihq431.com
iknight.lol
ikuuu.dev
imgot.info

industrialtechnologies2016.eu
inna.cfd
inncreasukedrev.info
innovatiefondsmkb.nl
inspectielandschap.nl
inutomo11.com
iogjhbnoypg.com
iot.mil
ipv4only.arpa
ir3.xyz
itnc.mil
jaavnacsdw.com
jamie3vkiwibfiwucd6vxijskbhpjdyajmzeor4mc4i7yopvpo4p7cyd.onion
jamiewebgbelqfno.onion
jatomayfair.life
jecc.mil
jfcom.mil
jhkkjkj.com
jpgtrk.com
jsmcrptjmp.com
jtj.mil
kabinetsformatie2010.nl
kgfjrb711.com
kit.ag
konylabs.capitalone
krav257.xyz
ku2d3a7pa8mdi.com
labs-semrush.com
lahitapiola.mobile
laowangla.top
laowangta.top
lby2kd27c.com
learndigital.withgoogle.com
lephaush.net
library.lol
limurol.com
lob-assets.com
localize.live
logitechauthorization.com
logitech-channel-marketing.com
logmein-gateway.com
lordanavid2.com
lordserials.org
louchaug.com
lscr.io
luxuryretreats.com
manage.wix.com
manatoki215.net
manga1000.su
mangaraw.ru
matomeantena.com
medregeu2016.nl
megacloud.tv
merequartz.com
mobile.prod
mobisystems.office
mojave.net
monaco.mobile
monsnode.com
mooo.com
mplayeranyd.info
msg-csw-lenovo.com
mtnplay.co.zm
mtnselfcare.co.zm
mtwdmk9ic.com
mvtvbbs.com
myastaro.com
mybettermb.com
mystarbucks.kr
mysymptoms.mil
naanalle.pl
nanouwho.com
nationaleopendagdji.nl
ndpa.nl
nederlandcallcenter.nl
nereserv.com
netherlandsinvestmentagency.nl
newrotatormarch23.bid
newtoki215.com
nextcloud.talk
nfi-academy.nl
nlbijio.nl
nldevelopment.nl
nltr2012.nl
nltradeandinvest.nl
nlww.nl
nolra.cyou
noonoo26.tv
notifpush.com
notix.io
nova-mobile.de
ntvpforever.com
nullpoantenna.com
olaskilling.in
omnatuor.com
omnisnippet1.com
ondernemeninkoeweit.com
onmarshtompor.com
oppodvd.com
optimise.net
orange-directory.com.ua
osiextantly.com
outsimiseara.com
owrkwilxbw.com
paypal.here
pelisflix2.org
pelisplus2.io
pickmee.id
pickmee.in.th
pickmee.my
pickmee.ph
pickmee.sg
pickmee.vn
pisism.com
play-portraits.com
plex.direct
poddamnthatsfunny.com
pogothere.xyz
polarcdn-engine.com
polarcdn-terrax.com
pongponghoney.net
pre-gbtspain.com
prinsjesdag2013.nl
professioninperspective.nl
propeller-tracking.com
proxycheck.link
psdw89.com
publpush.com
pxv.pay
qgxbluhsgad.com
qvdt3feo.com
randstad380kv-zuidring.nl
raspberrypi.org
ratelimited.me
ratemyprofessors.com
ravenjs.com
ravenminer.com
ravm.tv
rawgit.com
rawkuma.com
rawstory.com
raygun.io
rayjump.com
razer.com
razersynapse.com
razerzone.com
razorpay.com
rbc.com
rbcroyalbank.com
rbc.ru
rbs.com
rbs.co.uk
rbsdigital.co.uk
rbxcdn.com
rbx.com
rcrsv.io
rcs.it
rcsmetrics.it
rcsobjects.it
rcvlink.com
rdatasrv.net
rdcnw.net
rdcpix.com
rd.go.th
rdstation.com.br
rdtcdn.com
reactjs.org
reactnative.dev
readcomiconline.li
readspeaker.com
ready4.icu
real.com
real-debrid.com
realestate.com.au
realmebbs.com
realme.com
realme.com.tw
realmemobile.com
realme.net
realmepaysa.com
realmeservice.com
realogy.com
realpage.com
realpython.com
realsee.com
realsrv.com
realtimely.io
realtor.ca
realtor.com
realytics.io
reaperscans.com
reasonsecurity.com
rebuyengine.com
recaptcha.net
recarga.com
recargapay.com.br
rechargecdn.com
rechtspraak.nl
reckoproduction.com
reckostaging.com
reclameaqui.com.br
rec.net
recombee.com
recordedfuture.com
record.pt
recruit.co.jp
recurbate.com
redbox.com
redbull.com
redbullmediahouse.com
redbullmusicacademy.com
redbull.tv
redbutton.de
redcdn.pl
redecanais.cx
redecanais.la
report1.biz
returnyoutubedislikeapi.com
richersitfast.life
rijksoverheid20.nl
rtbrenab.com
rtbrennab.com
rwssportdag.nl
rxeosevsso.com
sarcoidosis.org
save-insta.com
sbc-nms.ch
sbcnms.ch
schatkistbankieren.nl
sciorijk.nl
sda.fyi
seoab.io
showmax.app
sibautomation.com
simplewebanalysis.com
sirianlucet.digital
skypicker.com
slim-onderhoud.nl
slourenrib.top
soda2016.nl
solemik.com
sophosxl.com
sophosxl.net
spark-summit.com
spell.run
spotify.xxx
spritzawapuhi.guru
sstm.moe
staatssecretarisheemskerk.nl
stackpath.dev
starbucks.br

starbucks.com.cn
starbucks.jp
static1.squarespace.com
steadfastseat.com
stly.eu
stonkstime.com
stoppain.org
stormstone.top
supapush.net
supertms.com
surfsharkstatus.com
sweatco.app
swisscombroadcast.ch
swisscombroadcast.com
swisscom-sharespace.com
swisscom-tv.com
swisstrustdir.ch
szwkerncijfers.nl
tenpay.com
thaudray.com
thecloudvantnow.com
thoughgaffer.uno
thoughtmill.com
toegangocw.nl
topcoders.ir
trackcmp.net
tzegilo.com
u062.com
es.com0cf.io
uipath.com
ukcapitalone.capitalone
ulmoyc.com
unikoingold.com
unitedincome.com
untrk.xyz
upbam.org
urbanaffairskerala.org
use-application-dns.net
users.wix.com
usmgny.org
uspb-depeevee.nl
uuribao.org
verkiezingen2013.nl
vic2016.nl
vidhd.fun
viewyentreat.guru
voedselenwarenautoriteit.nl
volleyhivepong.com
vptbeurs2010.nl
waisheph.com
watisdeslimmemeter.nl
watvooreeneikelbenjij.nl
watvoorloserbenjij.nl
waust.at
webinar-windturbines.nl
webtiser.com
webvenadvdesign.com
weerrhoop.cc
wetlz.nl
wewriteyour.report
wijzermetgeldzaken.nl
winnerscircleregistration.com
wisepops.net
wovationtravel.com
wovensur.com
wqjbldnnceroue.com
wuzbhjpvsf.com
xdiwbc.com
xhqxmovies.com
xiaomi.market
xn--4gq62f52gdss.com
xxfreehdvideos.com
xxupdatemovies.com
yandexmetrica.com
yandex.ua
yaurl302.xyz
yhdmp.cc
ymetrica1.com
yourdelivery.de
zdq0g8hnsrqs1so.com
zichtopiederkind.nl
zivvermeet.com
zoompso.com
zoruus.com

Appendix D. trim-one.py

```
import sys

for line in sys.stdin:
    line = line.rstrip()
    line = line.rstrip(".")

    dot_count = line.count(".")
    labels = line.split(".")
    trimmed_line = ".".join(labels[1:])
    print (trimmed_line)
```

Appendix E. generate-wildcard-probes.py

```
import random
import string
import sys
import fileinput

def id_generator(size=8, chars=string.ascii_uppercase + string.digits):
    return ''.join(random.choice(chars) for _ in range(size))

for line in fileinput.input():

    random_string=(id_generator(size=8))
    random_string=random_string.lower()
    print(random_string+"."+line.rstrip())
```

Appendix F. Sample Wildcard Detection Run

Iteration #1:

```
$ cp just-rrnames-sorted-uniqued.txt names-to-test.txt
$ generate-wildcard-probes.py < names-to-test.txt > names-to-test-2.txt
$ massdns -r resolvers.txt -t A -o Sm -s 30000 -c 20 -w
wildcards-that-resolve-2.txt names-to-test-2.txt

Concurrency: 30000
Processed queries: 206142295
Received packets: 256505222
Progress: 100.00% (02 h 27 min 49 sec / 02 h 27 min 49 sec)
Current incoming rate: 421 pps, average: 28920 pps
Current success rate: 0 pps, average: 22925 pps
Finished total: 206142295, success: 203337637 (98.64%)
Mismatched domains: 39955457 (15.59%), IDs: 1281 (0.00%)
Failures: 0: 31.07%, 1: 23.65%, 2: 15.58%, 3: 10.00%, 4: 6.40%, 5: 4.08%, 6:
2.61%, 7: 1.68%, 8: 1.10%, 9: 0.72%, 10: 0.49%, 11: 0.34%, 12: 0.24%, 13:
0.18%, 14: 0.14%, 15: 0.11%, 16: 0.09%, 17: 0.07%, 18: 0.06%, 19: 0.05%, 20:
1.36%,
Response: | Success:               | Total:
OK:       |    161596145 ( 79.47%) |    180864036 ( 70.58%)
NXDOMAIN: |     41741492 ( 20.53%) |     49514774 ( 19.32%)
SERVFAIL: |            0 (  0.00%) |     20712861 (  8.08%)
REFUSED:  |            0 (  0.00%) |      5135240 (  2.00%)
FORMERR:  |            0 (  0.00%) |        29696 (  0.01%)
```

Response OK ==> wildcard (**`wildcards-that-resolve-2.txt`**)

Iteration #2:

Now take a copy of the original data, and strip one label. Having done so, we may have some duplicate names, so we `sort` and `uniq`, then generate our wildcard test names and attempt to resolve them all:

```
$ cp just-rrnames-sorted-uniqued.txt names-to-test.txt
$ trim-one.py < names-to-test.txt > names-to-test-3.txt
$ LC_ALL=C sort -T . < names-to-test-3.txt > names-to-test-4.txt
$ uniq < names-to-test-4.txt > names-to-test-5.txt
$ generate-wildcard-probes.py < names-to-test-5.txt > names-to-test-6.txt
$ massdns -r resolvers.txt -t A -o Sm -s 30000 -c 20 -w
wildcards-that-resolve-3.txt names-to-test-6.txt

Concurrency: 30000
Processed queries: 21617391
Received packets: 26827027
Progress: 100.00% (00 h 16 min 44 sec / 00 h 16 min 44 sec)
Current incoming rate: 581 pps, average: 26730 pps
Current success rate: 0 pps, average: 21066 pps
Finished total: 21617391, success: 21142808 (97.80%)
```

```
Mismatched domains: 4173127 (15.57%), IDs: 101 (0.00%)
Failures: 0: 30.92%, 1: 23.55%, 2: 15.36%, 3: 9.74%, 4: 6.18%, 5: 3.94%, 6:
2.52%, 7: 1.64%, 8: 1.09%, 9: 0.74%, 10: 0.52%, 11: 0.37%, 12: 0.28%, 13:
0.22%, 14: 0.18%, 15: 0.14%, 16: 0.12%, 17: 0.11%, 18: 0.09%, 19: 0.08%, 20:
2.20%,
Response: | Success:                | Total:
OK:       |     15674493 ( 74.14%) |     17419441 ( 65.00%)
NXDOMAIN: |      5468315 ( 25.86%) |      6444997 ( 24.05%)
SERVFAIL: |            0 (  0.00%) |      2408874 (  8.99%)
REFUSED:  |            0 (  0.00%) |       522557 (  1.95%)
FORMERR:  |            0 (  0.00%) |         5168 (  0.02%)
```

Response OK ==> wildcard (**wildcards-that-resolve-3.txt**)

Iteration #3:

```
$ trim-one.py < names-to-test-5.txt > names-to-test-7.txt
$ LC_ALL=C sort -T . < names-to-test-7.txt > names-to-test-8.txt
$ uniq < names-to-test-8.txt > names-to-test-9.txt
$ generate-wildcard-probes.py < names-to-test-9.txt > names-to-test-10.txt
$ massdns -r resolvers.txt -t A -o Sm -s 30000 -c 20 -w
wildcards-that-resolve-4.txt names-to-test-10.txt

Concurrency: 30000
Processed queries: 4087753
Received packets: 5080114
Progress: 100.00% (00 h 02 min 58 sec / 00 h 02 min 58 sec)
Current incoming rate: 417 pps, average: 28457 pps
Current success rate: 0 pps, average: 22695 pps
Finished total: 4087753, success: 4051506 (99.11%)
Mismatched domains: 764824 (15.07%), IDs: 22 (0.00%)
Failures: 0: 30.54%, 1: 24.02%, 2: 15.96%, 3: 10.30%, 4: 6.60%, 5: 4.22%, 6:
2.68%, 7: 1.70%, 8: 1.10%, 9: 0.71%, 10: 0.45%, 11: 0.29%, 12: 0.19%, 13:
0.12%, 14: 0.08%, 15: 0.06%, 16: 0.04%, 17: 0.03%, 18: 0.02%, 19: 0.01%, 20:
0.89%,
Response: | Success:                | Total:
OK:       |      2726003 ( 67.28%) |      3040710 ( 59.91%)
NXDOMAIN: |      1325503 ( 32.72%) |      1563768 ( 30.81%)
SERVFAIL: |            0 (  0.00%) |       375910 (  7.41%)
REFUSED:  |            0 (  0.00%) |        93981 (  1.85%)
FORMERR:  |            0 (  0.00%) |          782 (  0.02%)
```

Response OK ==> wildcard (**wildcards-that-resolve-4.txt**)

Iteration #4:

```
$ trim-one.py < names-to-test-9.txt > names-to-test-11.txt
$ LC_ALL=C sort -T . < names-to-test-11.txt > names-to-test-12.txt
$ uniq < names-to-test-12.txt > names-to-test-13.txt
$ generate-wildcard-probes.py < names-to-test-13.txt > names-to-test-14.txt
$ massdns -r resolvers.txt -t A -o Sm -s 30000 -c 20 -w
wildcards-that-resolve-5.txt names-to-test-14.txt
```

```
Concurrency: 30000
Processed queries: 1326618
Received packets: 1470931
Progress: 100.00% (00 h 02 min 59 sec / 00 h 02 min 59 sec)
Current incoming rate: 621 pps, average: 8193 pps
Current success rate: 0 pps, average: 6533 pps
Finished total: 1326618, success: 1172845 (88.41%)
Mismatched domains: 154225 (10.50%), IDs: 6 (0.00%)
Failures: 0: 11.12%, 1: 11.10%, 2: 9.34%, 3: 7.94%, 4: 6.83%, 5: 5.91%, 6:
5.17%, 7: 4.47%, 8: 3.95%, 9: 3.51%, 10: 3.09%, 11: 2.74%, 12: 2.41%, 13:
2.15%, 14: 1.90%, 15: 1.68%, 16: 1.51%, 17: 1.32%, 18: 1.19%, 19: 1.07%, 20:
11.59%,
Response: | Success:               | Total:
OK:       |       557886 ( 47.57%) |       642970 ( 43.75%)
NXDOMAIN: |       614959 ( 52.43%) |       665519 ( 45.29%)
SERVFAIL: |            0 (  0.00%) |       132306 (  9.00%)
REFUSED:  |            0 (  0.00%) |        28375 (  1.93%)
FORMERR:  |            0 (  0.00%) |          334 (  0.02%)
```

Response OK ==> wildcard (**`wildcards-that-resolve-5.txt`**)

Iteration #5:

```
$ ./trim-one.py < names-to-test-13.txt > names-to-test-15.txt
$ LC_ALL=C sort -T . < names-to-test-15.txt > names-to-test-16.txt
$ uniq < names-to-test-16.txt > names-to-test-17.txt
$ ./generate-wildcard-probes.py < names-to-test-17.txt > names-to-test-18.txt
$ massdns -r resolvers.txt -t A -o Sm -s 30000 -c 20 -w
wildcards-that-resolve-6.txt names-to-test-18.txt

Concurrency: 30000
Processed queries: 474213
Received packets: 586560
Progress: 100.00% (00 h 00 min 55 sec / 00 h 00 min 55 sec)
Current incoming rate: 4148 pps, average: 10666 pps
Current success rate: 1 pps, average: 7872 pps
Finished total: 474213, success: 432898 (91.29%)
Mismatched domains: 91926 (15.69%), IDs: 1 (0.00%)
Failures: 0: 19.30%, 1: 16.24%, 2: 11.65%, 3: 8.51%, 4: 6.36%, 5: 4.91%, 6:
3.89%, 7: 3.20%, 8: 2.73%, 9: 2.30%, 10: 1.99%, 11: 1.78%, 12: 1.53%, 13:
1.36%, 14: 1.21%, 15: 1.07%, 16: 0.96%, 17: 0.85%, 18: 0.75%, 19: 0.69%, 20:
8.71%,
Response: | Success:               | Total:
OK:       |       175870 ( 40.63%) |       204044 ( 34.82%)
NXDOMAIN: |       257028 ( 59.37%) |       284776 ( 48.60%)
SERVFAIL: |            0 (  0.00%) |        85161 ( 14.53%)
REFUSED:  |            0 (  0.00%) |        11783 (  2.01%)
FORMERR:  |            0 (  0.00%) |          236 (  0.04%)
```

Response OK ==> wildcard (**`wildcards-that-resolve-6.txt`**)

[repeat until no remaining labels need to be evaluated]

Appendix G. rev-dom-large.py

```
import fileinput
for line in fileinput.input():
    line = line.rstrip()
    list = line.split(".")
    list.reverse()
    newline = '.'.join(list)
    print(newline)
```

Appendix H. remove-unneeded-wildcards.py

```
import sys
import fileinput

# wildcards we've already seen will be added here
already_seen=set()

def normal_order(list_of_labels):
    local_copy=list_of_labels.copy()
    local_copy.reverse()
    new_order='.' . join(local_copy)
    return new_order

def reversed_order(list_of_labels):
    new_order='.' . join(list_of_labels)
    return new_order

def reversed_order_stripped_one_label(list_of_labels):
    total_labels=len(list_of_labels)
    shorty='.' . join(list_of_labels[:-1])
    return shorty

master_line = sys.stdin.readline().rstrip()
master_list = master_line.split(".")
master_list_length=len(master_list)

# initial wildcard is always non-redundant by default
print (reversed_order(master_list))
already_seen.add(reversed_order(master_list))

for line in fileinput.input():
    line = line.rstrip()
    list = line.split(".")
    list_length=len(list)
    trimed_name = reversed_order_stripped_one_label(list)

    if trimed_name in already_seen:
            continue
    elif list_length < master_list_length:
            print (reversed_order(list))
            already_seen.add(reversed_order(list))
            # may be a new name, update comparators
            master_line=line
            master_list=list.copy()
            master_list_length=list_length
    elif list_length >= master_list_length:
            # is the comparator the same as the master,
            # at least up to the master's length
            for i in range(master_list_length):
                if (master_list[i] != list[i]):
                    master_line=line
```

```
master_list=list.copy()
master_list_length=list_length
print (reversed_order(list))
already_seen.add(reversed_order(list))
break
```

Appendix I. match-and-drop.py

```
import sys

pattern_set=set()

# following patterns are in normal label order, despite filename
wildcards=open("first-past-removing-reversed.txt", "r")
for pattern in wildcards:
    pattern=pattern.rstrip()
    pattern_set.add(pattern)

# patterns to review should also be in normal label order, such as
# names-to-test.txt
for line in sys.stdin:
    wildcard=False
    line = line.rstrip()
    line = line.rstrip(".")
    labels = line.split(".")
    num_of_labels=len(labels)
    for i in range(num_of_labels):
        offset=i+1
        trimmed_line = ".".join(labels[-offset:])
        if trimmed_line in pattern_set:
            wildcard=True
    if not wildcard:
        print(line)
```

Appendix J. 2nd-level-dom-large.pl

```
#!/usr/bin/perl
use strict;
use warnings;
use IO::Socket::SSL::PublicSuffix;

my $pslfile = '/usr/local/share/public_suffix_list.dat';
my $ps = IO::Socket::SSL::PublicSuffix->from_file($pslfile);

while (my $line = <STDIN>) {
        chomp($line);
        my $root_domain = $ps->public_suffix($line,1);
        printf( "%s\n", $root_domain );
}
```

Appendix K. Effective 2nd-level domains included in the challenge "seed" domains

ac.cn
ac.id
ac.in
ac.ir
ac.jp
ac.kr
ac.th
ac.uk
ad.jp
akadns.net
akamaiedge.net
akamaihd.net
akamai.net
akamaized.net
altervista.org
awsglobalaccelerator.com
azurewebsites.net
b.br
blog.br
cloudapp.net
cloudflare-ipfs.com
cloudfront.net
ddnsking.com
ddns.me
ddns.net
duckdns.org
dvrdns.org
dyndns.biz
dyndns.info
dyndns.org
dyndns.tv
dynns.com
dynu.net
edgekey.net
edgesuite.net
elasticbeanstalk.com
eu.org
fastlylb.net
fly.dev
gc.ca
go.gov.br
go.id
go.jp
go.kr
go.th
gouv.fr
govt.nz
gv.at
homeip.net
hopto.org
httpbin.org
inf.br
in.th
in.ua
jus.br
kasserver.com
kiev.ua
ladesk.com
likescandy.com
me.uk
mg.gov.br
mine.nu
mus.br
myds.me
myfritz.net
my.id
myqnapcloud.com
ne.jp
nhs.uk
nic.in
no-ip.org
ondigitalocean.app
or.id
or.jp
or.kr
pages.dev
qc.ca
remotewd.com
rs.gov.br
ru.com
sakura.ne.jp
sch.id
service.gov.uk
spdns.de
sp.gov.br
srv.br
supabase.co
sx.cn
synology.me
sytes.net
tv.br
tx.us
web.id
workers.dev
yandexcloud.net
ynh.fr
zapto.org

In Honor of Oktoberfest:

"Ein Prosit"
The Polka Brothers
https://www.youtube.com/watch?v=WCIEuWzhquU
49K views