

# **Nowhere To Hide: Using Passive DNS To Find Spammer Infrastructure**

M<sup>3</sup>AAWG Paris, France

12:30-14:30, October 24<sup>th</sup>, 2016

Forum ABC Room, Level 0

Joe St Sauver, Ph.D. (stsauver@fsi.io)

M3AAWG Sr. Technical Advisor

Scientist, Farsight Security, Inc.

<https://www.stsauver.com/joe/nowheretohide/>

## – Social Media Posting **Allowed** –

Twitter, Facebook, LinkedIn, other social media posts are **permitted** during this session, PROVIDED THAT YOU...

- **DO NOT post the private link provided for attendees to use to obtain a free account for training purposes**
- Only post comments made by the speaker
- Do not post comments or questions from the audience (but you can share the speaker's responses to questions)
- Do not post the name, position or company of other meeting attendees
- Do not post conversations with attendees
- M<sup>3</sup>AAWG is not a deliverability conference; we are
  - An industry working group meeting
  - An anti-abuse conference, or
  - A gathering of security experts
- All of the M<sup>3</sup>AAWG Membership, Trademarks and Logo guidelines apply  
<https://www.m3aawg.org/members/how-promote-m3aawg#TrademarkGuidelines>

# Welcome To The Paris Passive DNS Training!

- Let me begin by thanking **Anna, Chris & Udeme** for the invitation to do passive DNS training for you here in Paris.
- I also want to thank all of **you** for making the time to sit in on today's session – I know that you're in "La Ville Lumière" with lots of tempting sites to see and wonderful bars/restaurants to enjoy, so I'll do my best to make sure your training time is well spent.
- We're going to begin by talking about **passive DNS**, and then, after that, if we still have time and there's interest, we'll go back and provide some optional backfill about "**regular DNS**" (normally I'd reverse those sections, but I suspect that many of you already have a working fluency about DNS)
- There will be a sign-in sheet going around if you need a **certificate of attendance** for your files back home.
- Please also take the time to fill out a **session evaluation**.

# A Disclaimer About The IPs and Domains Used Today

- To illustrate the tools and techniques we'll be talking about today, I'll be showing examples that involve various IP addresses, netblocks, ASNs, and domains. **Unless stated to the contrary, I do NOT mean to imply that these are bad (or good!) IPs, domains, etc. (I'll often use colleges) These are JUST EXAMPLES.**
- You, however, when practicing with passive DNS, may want to try investigating domains that you believe to be suspicious. Some of those sites may even be involved with **malware**.
- **Please do NOT investigate any sites that may be involved with malware while here at M3AAWG** (we don't want anyone getting infected, or spreading an infection to others). If you do choose to investigate such sites elsewhere, you do so at your own risk.

# My Odd Slide Style

- Let me also get one other thing "out of the way" up front: as you've seen by now, I produce **detailed slides**. Some people don't get why I use this style, so I now routinely try to explain this
- I've tried the more-typical 3-4 bullets/slide (with ~15 slides for an hour long talk), but I find myself getting **sidetracked, rambling/running over**, or I end up **missing/skipping stuff**.
- I also deal with **complex issues**, and I HATE to be misquoted.
- My slide style prevents a lot of those problems, and means that **you don't need to try to take notes**.
- That said, I'm **not going to read my slides word-for-word for you**. You don't need to try to do so, either, although they are a sort of "closed captioning" if you're deaf or hard-of-hearing.
- I also write detailed slides to help people **looking at them after the fact**, and to facilitate indexing by **web search engines**.

# Today's Session

- **I've prepared some material I'm planning to go over, but I encourage you to ask questions as we go along, should questions arise** – I've explicitly left some time for you to do so.
- I will also suggest some **exercises** you can try. The exercises are optional -- you can do them or skip them as you may like, but I think people learn more effectively when they actually try stuff.
- **Farsight is providing all attendees at today's session with complimentary temporary access to DNSDB for use as part of this training.** If you don't want to use DNSDB, you're welcome to use another passive DNS system you may have access to, instead.
- There are unavoidable differences between various passive DNS implementations and we can't document everything, but the techniques and approaches I'll show should broadly generalize. We want you to learn skills that will help you in your work.

# Some Entities Offering Passive DNS Services

- Farsight Security, Inc.'s DNSDB (see <https://dnsdb.info/> )
  - DNSDB is a commercial product, but **individual** law enforcement officers (LEOs), vetted academic researchers, and vetted-but-unfunded "Internet superheroes" can request free (grant) access from Farsight.
- Some other passive DNS implementations include:
  - Florian Weimer's BFK, [http://www.bfk.de/bfk\\_dnslogger.html](http://www.bfk.de/bfk_dnslogger.html)
  - CERT.at/Aconet Passive DNS (inquire: [kaplan@cert.at](mailto:kaplan@cert.at) or [lendl@cert.at](mailto:lendl@cert.at))
  - CIRCL Passive DNS, <http://www.circl.lu/services/passive-dns/>
  - <http://passivedns.mnemonic.no/search/>
  - <https://www.opendns.com/enterprise-security/resources/data-sheets/investigate/>
  - <https://www.cs.auckland.ac.nz/research/groups/sde/dhdb-index.php>
  - VirusTotal, <https://www.virustotal.com/#search>
  - 360.cn Passive DNS, <https://www.passivedns.cn/help/>
- If I missed any other passive DNS sites, please drop me a note...

This slide intentionally omitted – it contained details about how in person attendees could get a DNSDB account for use during the training



# **PART I. BASIC PASSIVE DNS VIA YOUR WEB BROWSER**

## **1) Introduction**

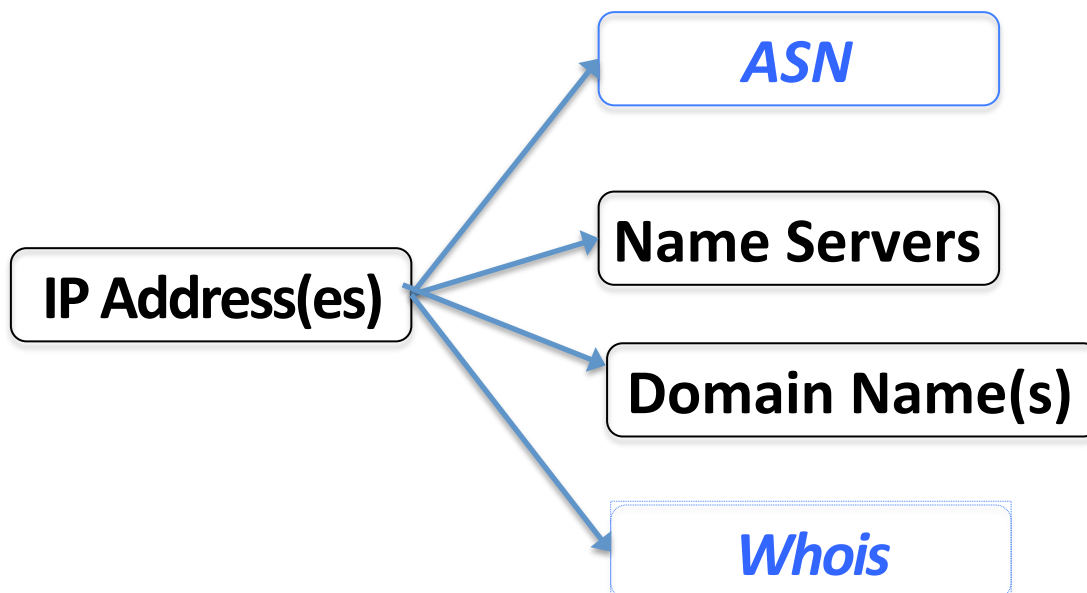
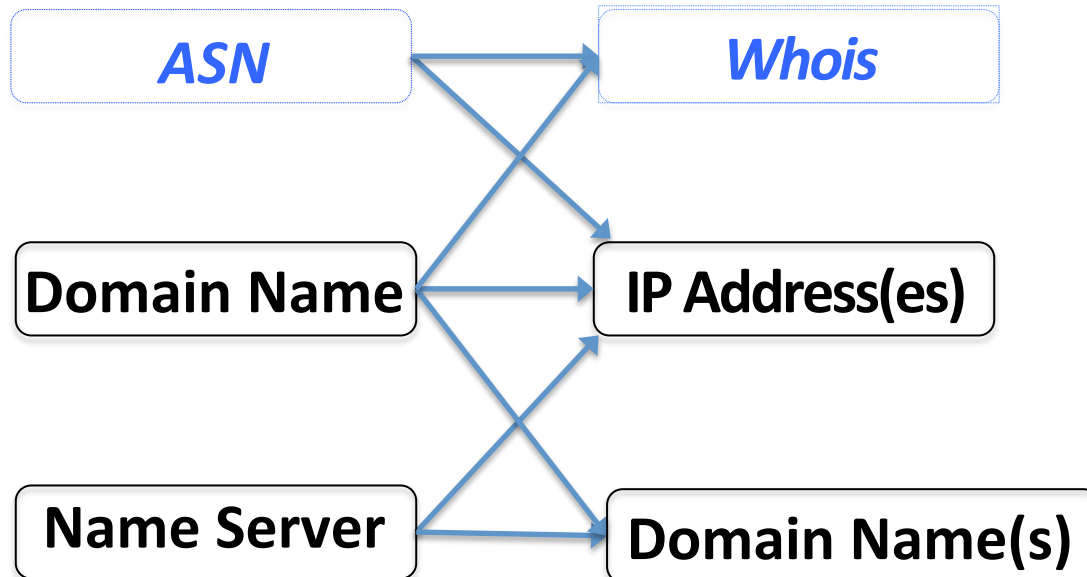
# What's DNS? (A Quick One Slide Intro)

- DNS is one of the Internet's core protocols. DNS maps domain names (such as `www.cnn.com`) to numeric IP addresses (such as `151.101.20.73`), and vice versa.
- We use the Domain Name System all the time without even thinking about it, in part because DNS seems to "just magically work," and symbolic domain names are **much** easier to remember than all-numeric IP addresses.
- Domain names can also be very convenient for server admins. For example, if a server administrator needs to move her web server to a new provider, she can do so without having to manually tell each user her site's new IP address. She can just change the DNS for her site, and then her users will automatically go to the right place when they next try to visit.

# Spammers and Other Cyber Criminals ALSO Use DNS

- DNS is convenient for everyone, good guys and bad guys alike.
- A typical spammer might have many web sites, perhaps different ones for each different affiliate sender, or different ones for each spamadvertised item. DNS make it easy for the spammer to manage those sites, and to share their limited pool of IP addresses.
- If a cyber criminal has a "bit of a set back," and ends up kicked off a hosting provider they've been using, they can just update their DNS when they find a new rock to hide under. Clearly, that's a convenience for the bad guy.
- However, DNS can also work AGAINST cyber criminals...

**Key idea: If we can discover one initial bad guy site, we may be able to use passive DNS to "PIVOT" and find related sites also being used by that bad guy, or OTHER bad guys.**



## Pivoting:

Using "Initial Clues" To Find "Related Resources"

Key.....

-- **Passive DNS Based**

-- **NOT *Passive DNS-Based pivot attributes***

# What's The Point of Pivoting via Passive DNS?

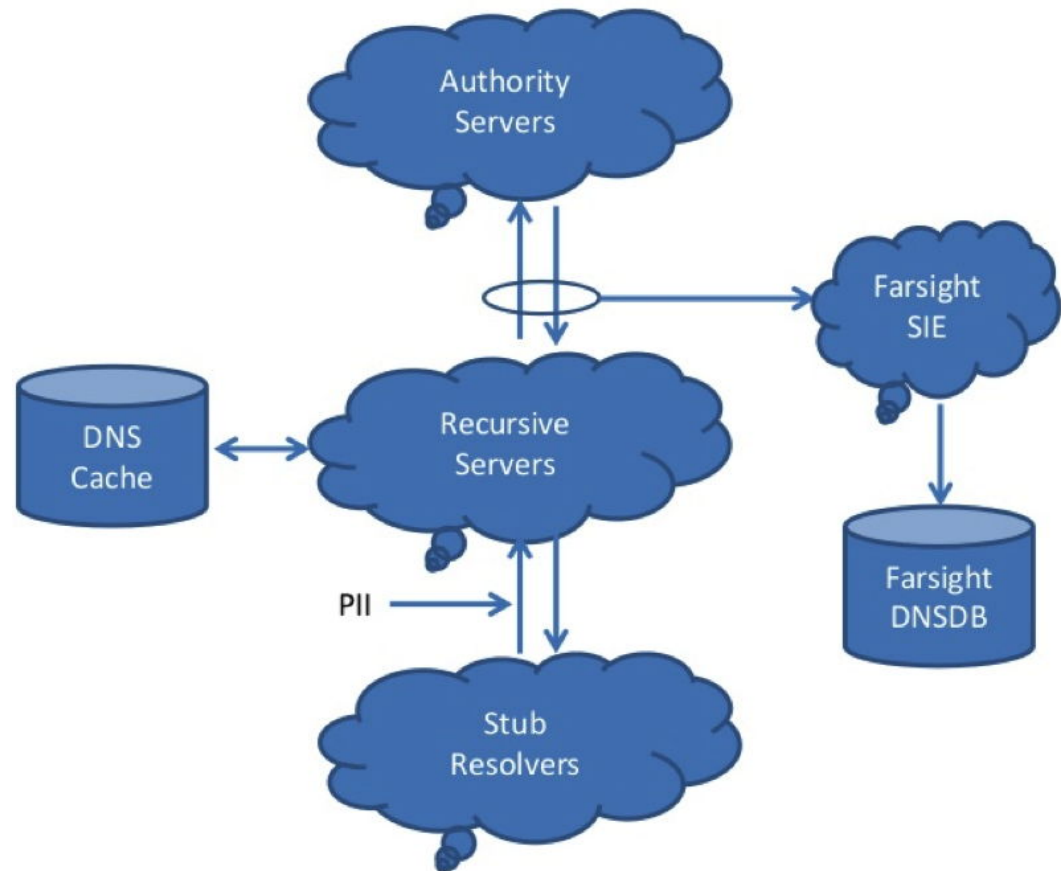
- **By exploring shared "evil" IPs (or CIDR netblocks)**
  - We can identify compromised machines that may need remediation,
  - It avoids leaving the miscreant with a foundation from which to recover
  - You maximize your chances of successfully chasing financial payment details and other business records
  - You may even discover additional unknown criminal "lines of business"
- **Identifying related bad domains**
  - Cyber criminals tend to be using more and more domain names, so there's no choice but to scale up right along with them.
  - More domains seized = more "news worthy" law enforcement actions, and more incentive for the good guys to spend their limited cycles on *this* case, not some other alternative ones
  - You wouldn't want to end up with an incomplete takedown/seizure (you know, potentially spawning online remarks such as "Hah hah hah, they seized a dozen of my domains, but they missed two thousand other ones I also have, so I didn't even really notice it")

# One Small Problem: Regular DNS Isn't Designed To Facilitate Hunting Spammers/Cybercriminals

- Hunting down bad guys was never a DNS design goal.
- It shouldn't be surprising, therefore, that normally we can't...
  - Find all the fully qualified domain names under a base domain
  - Find all the domains that use a specific name server
  - Given the IP used by one fully qualified domain name, find all other domain names that are also on that same IP address
  - Given a net block, find all the domains in that network range
  - Given a domain name, see if it has resolved to multiple different IPs over time, and if so, what were those IPs?
- These queries are all examples of the sort of things that "regular DNS" was just **NOT** set up to do...
- Fortunately, passive DNS *\*can\** handle those sort of queries.
- **By using passive DNS, we can make it harder for bad guys to hide**

# Where Does Passive DNS Data Come From?

- At least in Farsight's case, our passive DNS data is collected by passively monitoring **DNS cache miss traffic above large recursive resolvers – actual DNS queries and responses.**
- That primary data is also augmented with information from zone file access programs, as may be available.



# An Aside About Passive DNS and Privacy

- Farsight Security and I both care a great deal about user privacy, and we hope that you do, too.
- At the same time, we want and need to ensure that spammers and cyber criminals can be held accountable. Law breakers must not be free to perpetrate their online crimes with impunity.
- Passive DNS, collected properly, comes from **above large shared recursive resolvers. Queries appear to originate from the recursive resolver, not from any individual user.** As a result, no personally identifiable information gets collected or stored.
- Because of this architecture, passive DNS does NOT raise the sort of pervasive monitoring concerns that are associated with things like bulk metadata collection and traditional traffic analytic methods, as discussed in "The Enduring Challenges of Traffic Analysis," <https://www.stsauver.com/joe/dublin-traffic-analysis/>



## **2) The Simple Web Interface to DNSDB**

# Accessing DNSDB

- DNSDB can be accessed multiple different ways, including:
  - 1) Via a simple **web interface**. We'll talk about that first.
  - 2) Via a versatile **RESTful API** (see [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer) if you're curious about REST)
    - Often the API is accessed via a Unix command-line interface tool. We'll show you the Python DNSDB API client in the next section.
    - You can also access it via a Splunk plugin, convenient if you "live in Splunk"
    - From your own C language (or other programming language) applications, programmatically. We'll even show you skeleton libcurl code for doing that
- There are some other mechanisms that you can use to access DNSDB, too, but we'll forego talking about them today.
- Let's begin with the simple web interface.

# Limitations/Advantages of the Web Interface

## Limitations:

- The web interface is primarily meant for casual/occasional usage, and as such, is intentionally kept simple.
- The web interface returns at most 10,000 results (attempting to render larger tables of results can make some web browsers sluggish)
- The web interface currently lacks the ability to do time fencing or sorting of query results. It also doesn't offer special output formats (such as JSON-format output).

## Advantages:

- The web interface is simple to use
- The web interface works on pretty much any system with a browser, with no software installation required

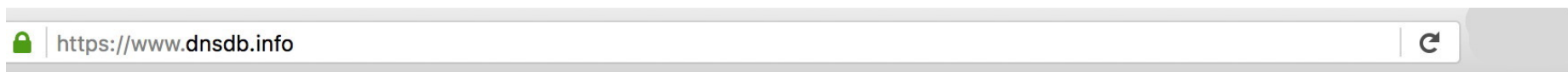
# Prerequisites For Using The DNSDB Web Interface

- In order to be able to use the web interface, you'll need a **DNSDB username** and **password**. If you requested one via the link provided earlier, that information should be waiting for you in your inbox.
- The web interface to DNSDB is at **<https://www.dnsdb.info/>**

You should be able to access that web site from Firefox, Chrome, Safari or any other popular web browser – just type that address into your web browser's address bar and hit return.

- Please try logging on to your test account now.

# The DNSDB Web Client's Opening Screen



[Home](#) [Login](#) [Apply](#) [Help](#)

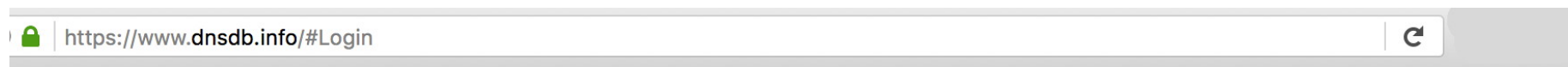
## Welcome to DNSDB

**DNSDB** is a database that stores and indexes both the passive DNS data available via Farsight Security's [Security Information Exchange](#) as well as the authoritative DNS data that various zone operators make available. DNSDB makes it easy to search for individual **DNS RRsets** and provides additional metadata for search results such as **first seen** and **last seen** timestamps as well as the **DNS bailiwick** associated with an RRset. DNSDB also has the ability to perform **inverse** or **rdata** searches.

Access to DNSDB is only allowed for authorized users. Please [apply for an account](#) if you are interested in obtaining access.

© 2010-2016 [Farsight Security, Inc.](#)  
Contact us by sending email to [dnsdb@farsightsecurity.com](mailto:dnsdb@farsightsecurity.com).

# Click "Login" on the Blue Menu Bar, Then Login...



Home Login Apply Help

## DNSDB Login

Username:



Password:

Login

© 2010-2016 **Farsight Security, Inc.**

Contact us by sending email to [dnsdb@farsightsecurity.com](mailto:dnsdb@farsightsecurity.com).

# This Is The Opening Search Screen You Should Then See

 <https://www.dnsdb.info/#Search> 

[Home](#) [Logout](#) [Search](#) [Account](#) [Help](#)

## DNSDB Search

**Search mode:** ☒ RRset ☐ Rdata

**Record type:** ☒  ☐

**Domain name:**

**Bailiwick:**

**Search**

**Reset**

© 2010-2016 [Farsight Security, Inc.](#)  
Contact us by sending email to [dnsdb@farsightsecurity.com](mailto:dnsdb@farsightsecurity.com).

# Make a Simple Sample Query...

## What IPv4 Addresses Have Been Used By ieee.org?

https://www.dnsdb.info/#Search

Home Logout Search Account Help

### DNSDB Search

Search mode: ☒ RRset ☐ Rdata

Record type:

☒ A

Domain name:

ieee.org

Bailiwick:

Search

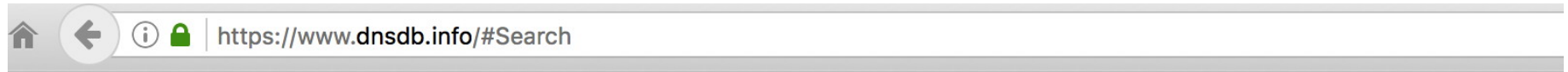
Reset

© 2010-2016 **Farsight Security, Inc.**

Contact us by sending email to [dnsdb@farsightsecurity.com](mailto:dnsdb@farsightsecurity.com).



# Results For That Query



## DNSDB Search

Search mode: ☒ RRset ☐ Rdata

Record type: ☒ A ☐

Domain name:

Bailiwick:

Search

Reset

  RRset results for **ieee.org/A** 

**Returned 3 RRsets in 0.05 seconds.**

```
bailiwick  ieee.org.
count      2527214
first seen  2010-06-24 04:11:02 -0000
last seen   2016-06-27 22:08:58 -0000
ieee.org. A 140.98.193.141
```

# Some Tips For Working With The Web Interface

- If you make multiple queries, results from later queries will get added to the **bottom** of the output screen. Scroll down to see them. [*Very helpful* to know if you think "nothing's happening" :-) ]
- To retain results from the web interface, copy and paste the results into a text file and save that file.
- If there are query results you no longer need, remove them from your output window by clicking the red X icon
- If there are query results you'd like to just *temporarily hide*, hit the green arrow icon. To restore those results, hit that green arrow icon again.

# Making Correct Choices In The Web Interface

- The parameters you set in the web interface control what gets searched for and found in DNSDB.
- The web form will change somewhat as you ask DNSDB to do different sorts of searches.
- If you pay attention to what DNSDB wants (e.g., "domain name" vs. "IP address or network") you shouldn't have too much trouble.
- If you do make a mistake, no big deal, just take a closer look and try it again...

# Sample Mistake: Mistakenly Putting In An IP Where A Domain Name Is Needed...

https://www.dnsdb.info/#Search

Home Logout Search Account Help

## DNSDB Search

**Search mode:** ☒ RRset ☐ Rdata

**Record type:** ☒ A ☐

**Domain name:**

**Bailiwick:**

Error... 128.223.32.35 isn't a domain name! That's an IP address

✖ No RRset results for **128.223.32.35/A**.

No big deal, just nothing found for the wrong query....

© 2010-2016 **Farsight Security, Inc.**  
Contact us by sending email to [dnsdb@farsightsecurity.com](mailto:dnsdb@farsightsecurity.com).

# Trying It Again, After Correcting The Query Settings

https://www.dnsdb.info/#Search

Home Logout Search Account Help

## DNSDB Search

**Search mode:** ☐ RRset ☒ Rdata

**Record type:** ☒ A ☐

**Record data:**

**Input mode:** ☐ Name ☒ IP or network ☐ Raw hex

✗ No RRset results for **128.223.32.35/A**.

✓ ✗ Rdata results for **A/128.223.32.35**

**Returned 16 RRs in 0.22 seconds.**

phloem.uoregon.edu.	A	128.223.32.35
cs.usac.edu.at	A	128.223.32.35

Worked, this time...

# Avoiding DNSDB Errors In The First Place

- The settings you pick in the web interface control what gets searched in the DNSDB data. Records in the DNSDB database are conceptually similar to regular DNS records:

<u><b>RRSET NAME</b></u>	<u><b>RECORD TYPE</b></u>	<u><b>RDATA</b></u>
ieee.org.	A	140.98.193.141
<b>"LEFT HAND SIDE"</b>		<b>"RIGHT HAND SIDE"</b>

- A **RRset** (Resource Record Set) name search looks for the specified domain name on the **left hand side** and returns any associated IPs
- If you wanted to see all known hosts (or "fully qualified domain names") that live "under" a domain, you'd do an **rrset** search for **\*.domain**
- To see the hosts that have been seen using an IP address, do an **rdata** ("right hand side") search for that **IP**
- Want the domains using a nameserver? Do an **Rdata name** search

# RRset searches: find left hand side matches

https://www.dnsdb.info/#Search

### DNSDB Search

"search the left hand side of DNSDB records..."

Search mode: ☒ RRset ☐ Rdata

Record type: ☒ ANY ☐

Domain name: \*.uoregon.edu

Bailiwick:

Search Reset

RRset results for \*.uoregon.edu/ANY

Returned 10000 RRsets in 3.11 seconds.

bailiwick	.
count	2256
first seen in zone file	2010-04-13 18:39:17 -0000
last seen in zone file	2016-06-28 20:11:15 -0000
phloem.uoregon.edu.	A 128.223.32.35
bailiwick	.
count	2256
first seen in zone file	2010-04-13 18:39:17 -0000
last seen in zone file	2016-06-28 20:11:15 -0000
phloem.uoregon.edu.	AAAA 2001:468:d01:20::80df:2023
bailiwick	uoregon.edu.

# Rdata searches: find right hand side matches

## DNSDB Search

"search the right hand side of DNSDB records..."

**Search mode:** ☐ RRset ☒ Rdata

**Record type:** ☒ ANY ☐

**Record data:**

**Input mode:** ☒ Name ☐ IP or network ☐ Raw hex

🟢 🚫 Rdata results for **ANY**/\*.uoregon.edu

Returned 10000 RRs in 95.66 seconds.

uoregon.biz.	NS	phloem.uoregon.edu.
uoregon.biz.	NS	ruminant.uoregon.edu.
coglink.com.	NS	dns.cs.uoregon.edu.
sabeink.com.	NS	dns.cs.uoregon.edu.
aerocenter.com.	NS	dns.cs.uoregon.edu.
bsdcommerce.com.	NS	dns.cs.uoregon.edu.
lnxcommerce.com.	NS	dns.cs.uoregon.edu.
solcommerce.com.	NS	dns.cs.uoregon.edu.
project-hifi.com.	NS	dns.cs.uoregon.edu.
warsawcenter.com.	NS	dns.cs.uoregon.edu.
astutesupport.com.	NS	dns.cs.uoregon.edu.
simpsonstimber.com.	NS	dns.cs.uoregon.edu.
thesingingkettle.com.	NS	dns.cs.uoregon.edu.
pattiandbobgreene.com.	NS	dns.cs.uoregon.edu.
simpsoninvestment.com.	NS	dns.cs.uoregon.edu.
bluemoonconsulting.com.	NS	dns.cs.uoregon.edu.



# A Brief DNS Record-Type "Cheat Sheet"

- If you don't choose a DNS record type, you'll see ALL record types by default. If you're not sure, just leave the record type set to ALL, and see what looks relevant to your needs (learn by playing with the tool).
- Too many results? Pick a record type to reduce unwanted "noise:"
  - "A" → name to **IPv4** address records ("regular DNS records")
  - "AAAA" → name to **IPv6** address records
  - "CNAME" → **aliases** for other names
  - "NS" → **name server** records
  - "TXT" → text records (often used for creative purposes)
  - "MX" → **mail exchanger** records
  - "SOA" → Start Of Authority records
  - "PTR" → pointer (or "inverse address") records
  - "SRV" → (relatively uncommon) server records
  - "RRSIG" → one (of several types) of DNSSEC records

# What About the "Bailiwick" Field In The Interface?

- **Just ignore the bailiwick field for now...** everything should work fine if you just leave it blank.
- If you're an enthusiast (or just overcome with curiosity):
  - If you find you're getting too many results, or puzzling results, you can set the bailiwick to be equal to the base domain that you're querying. This may reduce the number of records you're shown.

For example, if you're doing an rdata search for `www.example.com`, you could try setting the bailiwick to be equal to `example.com`

- If you'd like to read more about bailiwicks, see section 2.4 of [https://archive.farsightsecurity.com/Passive\\_DNS/passive-dns-architecture.pdf](https://archive.farsightsecurity.com/Passive_DNS/passive-dns-architecture.pdf)

# "How Do I Get More Than 10,000 Results?"

- **You can't** in the simple web interface. If we were to allow you to try, it would make at least some web browsers go completely catatonic as the browser struggled to render such a huge table.
- That said, even 10,000 results is far more than many web-based passive DNS implementations allow.
- If you need more than 10,000 results, try the command-line API client, which we'll be talking about in the next section. It will let you retrieve up to a million results (1,000,000) per query.

# Exercises

**These exercises are optional, but why not give them a try, eh?**

**a)** Using DNSDB, look up **your company's** main web site (this is an RRset query, Record type=A, Domain name=www.example.com (or whatever)). ***Find the most recent IP address it has used.***

**b)** Now click on that IP. **Are there any other hosts sharing that IP address?** (you may need to scroll down to see!)

**c)** **Let's now try doing a search for your company's nameservers.** Close the previous results (click the little red X stop signs above the results). Go back to the search box. Do an RRset query, Record type="NS", Domain name=example.com (or whatever). Do **NOT** include the www as part of the domain name you provide...

Look for one of the most recent name servers used by the domain and make a note of it. Now go on to exercise d) on the next slide.

## Exercises (continued #1)

**d) Let's see what other domains also use that name server.**

Do an **Rdata** (not **RRset**) search. Set the record type= NS.

Record data= the name of the **name server** you found in part c).

Input mode= Name.

How MANY domains appear to be using that name server?

**e) What hosts are known to be part of the europa.eu domain?**

RRset search.

Record type= ANY

Record data= \*.europa.eu (note the leading asterisk followed by a dot and then the domain name)

Bailiwick (optional)= europa.eu

**f) What hosts are in 147.67.0.0/17 ?** (Rdata search, record type=A,

Record data=147.67.0.0/17 , Input mode="IP or network").

Do you see *www.clubenglishhp.com*? Does it look out of place?

When was that host last seen by DNSDB? (how can you find out?) <sup>37</sup>

## Exercises (continued #2)

**g)** Different passive DNS systems may have slightly different interfaces, but you should be able to "cross walk" what you've learned without too much effort.

For instance, try some of the preceding exercises on

[http://www.bfk.de/bfk\\_dnslogger\\_en.html](http://www.bfk.de/bfk_dnslogger_en.html)

Try searching for europa.eu (see the next screen)

Note that some queries may NOT be available from some passive DNS implementations.

There may also be differences in the number (or format) of the results returned.

# europa.eu from BFK Passive DNS

www.bfk.de/bfk\_dnslogger\_en.html?query=europa.eu



The server returned the following data:

Name	Typ	Daten
europa.eu	A	147.67.119.2
europa.eu	A	147.67.119.20
europa.eu	A	147.67.119.102
europa.eu	A	147.67.136.2
europa.eu	A	147.67.136.20
europa.eu	A	147.67.136.102
europa.eu	NS	ns1.be.colt.net
europa.eu	NS	ns1.bt.net
europa.eu	NS	ns1bru.europa.eu
europa.eu	NS	ns1lux.europa.eu
europa.eu	NS	ns2bru.europa.eu
europa.eu	NS	ns2eu.bt.net
europa.eu	NS	ns2lux.europa.eu
europa.eu	NS	ns3bru.europa.eu
europa.eu	NS	ns3lux.europa.eu
europa.eu	AAAA	2a01:7080:14:100::361:1
europa.eu	AAAA	2a01:7080:14:100::362:1
europa.eu	AAAA	2a01:7080:24:100::361:1
europa.eu	AAAA	2a01:7080:24:100::362:1

# End of Part I

- Assuming you successfully completed the exercises on the preceding few slides, **congratulations**, you're now able to do basic passive DNS tasks!
- **If you're NOT a technical person and you're itching to hit a museum or see the Eiffel tower, you can "escape" now, and maybe try doing some further queries with your temporary access later tonight or in a few days. You should have access for about 15 days.**
- On the other hand, we'd encourage you to stay here and keep going! We've got more to share with you!



## **PART II. PASSIVE DNS API/Command Line Interface**

### **3) Installing the Command Line Interface (CLI) Python Client**

# Limitations And Advantages of Using The DNSDB API

## Limitations:

- The API Command Line Client works best if you're comfortable working at the **Unix shell prompt**
- **You will need to install software** (but just once!) to use the API Command Line Client from the command line

## Advantages:

- The API interface is easy to script if you're comfortable using Unix pipes and redirection
- The API can be asked to return up to 1,000,000 results/query
- The API can do time fencing and sorting of query results
- The API offers JSON output format, in addition to traditional plain text output

# Prerequisites For Accessing DNSDB Via The API

- You'll need your DNSDB API key. You were sent one along with your DNSDB username and password, so check your email!
- **If at all possible, access the API from a Linux, BSD or Mac OS system.**
- **If you MUST run under Windows:**
  - You can try running the DNSDB command line client in a Unix virtual machine on top of Windows
  - Virtualbox is one free virtual machine environment you can try for this purpose. This is a relatively large download, so you may not want to download it right now, but directions are available in the MS Word document "installing-virtualbox-m3aawg.docx" in the <https://www.stsauver.com/joe/nowheretohide/> directory

# Python DNSDB API Command Line Client

- There are actually multiple DNSDB command line clients available.
- **For this training, we're going to use the Python DNSDB client known as "dnsdb\_query.py"**
- To get that software (and instructions for its installation), see <https://github.com/dnsdb/dnsdb-query>
- You will need to be comfortable installing software, or get help from someone who is, to install dnsdb\_query.py
- If you're not comfortable installing that software, you can also just watch while we go through this section.

# Prerequisites

(per <https://github.com/dnsdb/dnsdb-query>)

- Linux, BSD, OS X
  - Curl *[see <https://curl.haxx.se/> ]*
  - Python 2.7.x *[see <https://www.python.org/> ]*
  - Farsight DNSDB API key *[see your email, if you signed up for a free DNSDB training account]*
- 
- Curl and python 2.7.x often come pre-installed on many Unix-ish systems, so we will not address installing them further here. If you do need to install them, they will usually be available from your operating system's package management system.

# Installing dnsdb\_query.py per <https://github.com/dnsdb/dnsdb-query>

*You will only need to do this ONCE:*

*1) Create a directory*

```
mkdir ~/dnsdb
```

*2) Download the software:*

```
curl https://codeload.github.com/dnsdb/dnsdb-query/  
tar.gz/debian/0.2-1 -o ~/dnsdb/0.2-1.tar.gz
```

(continues on next slide)

# Installing dnsdb\_query.py (continued)

*3) Extract the software*

```
tar xzvf ~/dnsdb/0.2-1.tar.gz -C ~/dnsdb/  
--strip-components=1
```

*4) Create an API key file [note the dot before dnsdb-query.conf!]*

```
nano ~/.dnsdb-query.conf
```

*5) Cut and paste the following, substituting your API Key where shown [there should be quotes shown around the API key in the file!]*

```
APIKEY="putYourAPIkeyhere"
```

## Installing dnsdb\_query.py (continued #2)

*6) Test the software:*

```
python dnsdb/dnsdb_query.py -i 104.244.13.104
```

```
...
```

```
www.farsightsecurity.com. IN A 104.244.13.104
```

*7) Recommended: Copy dnsdb\_query.py to a directory in your path*

If dnsdb\_query.py is copied to a directory that's in your Unix system's default path and is executable, you will then simply be able to say:

```
dnsdb_query.py -i 104.244.13.104
```



# Debugging Errors Trying To Run `dnsdb_query.py`

- If your API key is wrong, you may see:

```
$ dnsdb_query.py -r yahoo.com
```

```
HTTP Error 403: forbidden
```

- Seeing a silent "error" with no error message/feedback?

```
$ dnsdb_query.py -r stsauver.com
```

```
$
```

Do you have a `.dnsdb-query.conf` file in your home directory?  
(note the leading dot!) Does it contains an `APIKEY` line, with your actual key where it says *YourLongAlphaNumericAPIKeyHere* ?

```
APIKEY="YourLongAlphaNumericAPIKeyHere"
```

# Learning A Little About The Command Line Client

- Some (comparatively terse) information about the DNSDB API is available online at <https://api.dnsdb.info/>
- You can also read the command line synopsis...

## \$ **dnsdb\_query.py --help**

Usage: dnsdb\_query.py [options]

### Options:

- |                                       |   |
|---------------------------------------|---|
| -h, --help                            | show this help message and exit           |
| -c CONFIG, --config=CONFIG            | config file                               |
| -r RRSET, --rrset=RRSET               |   |
|                                       | rrset <ONAME>[/<RRTYPE>[/BAILIWICK]]      |
| -n RDATA_NAME, --rdataname=RDATA_NAME |   |
|                                       | rdata name <NAME>[/<RRTYPE>]              |
| -i RDATA_IP, --rdataip=RDATA_IP       |   |
|                                       | rdata ip <IPADDRESS IPRANGE IPNETWORK>    |
| -s SORT, --sort=SORT                  | sort key                                  |
| -R, --reverse                         | reverse sort                              |
| -j, --json                            | output in JSON format                     |
| -l LIMIT, --limit=LIMIT               | limit number of results                   |
| --before=BEFORE                       | only output results seen before this time |
| --after=AFTER                         | only output results seen after this time  |

Time formats are: "%Y-%m-%d", "%Y-%m-%d %H:%M:%S", "%d" (UNIX timestamp),  
"%-d" (Relative time in seconds), BIND format (e.g. 1w1h, (w)EEK, (d)ay,  
(h)our, (m)inute, (s)econd)

# Learning By Examples and Mucking Around A Little

- The best way to learn to use passive DNS in API mode is probably by seeing some examples, and then playing with the tools a bit yourself.
- The DNSDB API basically has three modes in which it can be used:

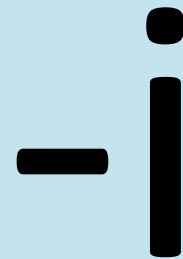
-i

-r

-n

- Let's start with -i, our only option if we're searching for an IP address or a CIDR netblock.

## 4) Investigating An IP Address or CIDR Block



## If Your Starting Point Is An IP Address

**You'll want to make a "dash i" query (that's the ONLY option that's available for searching IP addresses, network ranges, etc.).**

You may find IPs of interest in your mail server logs, firewall logs, email message headers, etc.

```
$ dnsdb_query.py -i 199.48.133.170  
3sea.ru. IN A 199.48.133.170  
mail.3sea.ru. IN A 199.48.133.170  
mail.it-solver.ru. IN A 199.48.133.170  
[etc]
```

# Why Would You Want To Search For An IP Address In Passive DNS?

- Some reasons include...
- You have an IP address, and you want to know what domains have been seen on that IP address. Is there anything unexpected? Do the domain names found look suspicious? (I33t speak, names that appear confusingly similar to popular phishing targets, names that mention prohibited activities such as carding, etc.)
- Ultimately, you'll typically want to "pivot" from that IP address to related domain names (start with a clue (that's an IP address) and follow that lead to the domain names that use that IP)
- **Searching passive DNS by IP address is a fundamental skill....**

# Time Fencing: Show Results From Just the Last 90 Days

Operational security analysts often don't care about historical results. They're intensely focused on what's happening NOW, or what's just happened in the immediate past.

If that's true for you, maybe limit what you get shown to the last thirty or ninety days...

```
$ dnsdb_query.py -i 199.48.133.170 --after=90d
```



## A Few Other Ways To Specify Times....

```
$ dnsdb_query.py -i 199.48.133.170 --after=2015-08-22
$ dnsdb_query.py -i 128.223.0.0/16 --before=2016-01-01
--after=2015-01-01
$ dnsdb_query.py -i 199.48.133.170 --after="2015-08-22
14:36:10"
$ dnsdb_query.py -i 199.48.133.170 --before=2013-01-22
$ dnsdb_query.py -i 128.223.0.0/16 --after="-3600"
```

*You Can Even Use Raw Unix epoch seconds...*

```
$ dnsdb_query.py -i 199.48.133.170 --after=1467074500
```

*Basics of working with Unix epoch second dates (on a Mac)...*

```
$ date +%s                                ← if you want the current time in seconds
1477022891
$ date -j -f "%b %d %Y %T" "Aug 28 1991 00:00:00" "+%s"
683362800
$ date -r 683362800
Wed Aug 28 00:00:00 PDT 1991
```

# What If You Want To See What's In A **CIDR Netblock**?

Sometimes you want to see what's in an entire net block, not just what's related to a single IP address:

```
$ dnsdb_query.py -i 199.48.128.0/21 > temp.txt
```

```
$ wc -l temp.txt
```

```
10000 temp.txt
```

← max of 10,000 results by default

If you need more results, increase the max results limit...

```
$ dnsdb_query.py -l 1000000 -i 199.48.128.0/21 >  
temp.txt
```

```
$ wc -l temp.txt
```

```
22205 temp.txt
```

```
$ more temp.txt
```

```
rootbsd.daleco.biz. IN A 199.48.129.182
```

```
[etc]
```

# What If I'm Interested in IPv6 Addresses or Blocks?

```
$ dnsdb_query.py -i 2607:f010:2e8:228:0:ff:fe00:152
```

```
gateway.lb.it.ucla.edu. IN AAAA
```

```
2607:f010:2e8:228:0:ff:fe00:152
```

```
gateway-v6.lb.it.ucla.edu. IN AAAA
```

```
2607:f010:2e8:228:0:ff:fe00:152
```

```
$ dnsdb_query.py -i 2607:F010::/32 > temp.txt
```

```
$ wc -l temp.txt
```

```
932 temp.txt
```

```
$ more temp.txt
```

```
ip-list.emileaben.com. IN AAAA 2607:f010::1
```

```
secure.math.ucla.edu. IN AAAA 2607:f010:2a8:8fe4::44
```

```
secure.math.ucla.edu. IN AAAA 2607:f010:2a8:8fe4::45
```

```
webmail.math.ucla.edu. IN AAAA 2607:f010:2a8:8fe4::211
```

```
webmail2.math.ucla.edu. IN AAAA 2607:f010:2a8:8fe4::211
```

```
[etc]
```

# How About An Arbitrary Network IP Range?

```
$ dnsdb_query.py -i 199.48.128.33-199.48.128.43
ns1.whittingtonpark.org. IN A 199.48.128.42
ns2.whittingtonpark.org. IN A 199.48.128.42
ns1.musicman.com. IN A 199.48.128.34
[etc]
```

*But that output is unsorted! What if we want it sorted by IP?*

```
$ dnsdb_query.py -i 199.48.128.33-199.48.128.43 --sort
rdata
ns1.musicman.com. IN A 199.48.128.34
drjohnngreenphotos.com. IN A 199.48.128.34
persianenglishtranslations.com. IN A 199.48.128.34
ilcr.x.rootbsd.net. IN A 199.48.128.34
[etc]
```

## Some -i Sorting Notes

*"valid sort keys are count, rdata, rrname, rrtype, zone\_time\_first, zone\_time\_last"*

To reverse the sort order, specify -R (or --reverse). No ability to sort by multiple keys (e.g., you can't sort by IP and then rrname within IP)

But try:

```
$ dnsdb_query.py -i 199.48.128.33-199.48.128.43 | grep  
-v ";;" | awk '{print $3 " " $1 " " $2}' | sort -u | awk  
'{print $2 " " $3 " " $1}'
```

```
drjohnngreenphotos.com. A 199.48.128.34  
ilcr.x.rootbsd.net. A 199.48.128.34  
ns1.musicman.com. A 199.48.128.34  
persianenglishtranslations.com. A 199.48.128.34  
[etc]
```

## Some Of The Sort-Related Jargon...

**count**=number of times DNSDB has seen this unique RRset

**rdata**=right hand side of the record (e.g., the IP info for "A" records)  
For "ucla.edu. IN A 128.97.27.37", the rdata is 127.97.27.37

**rrname**=left hand side of the record (e.g., the domain name for "A" records). For "ucla.edu. IN A 128.97.27.37", the rrname is ucla.edu

**rrtype**=is this an "A" record? a "AAAA"? an "NS" record? a "CNAME"? an "MX" record? an "SOA"? For "ucla.edu. IN A 128.97.27.37" the rrtype is "A"

**zone\_time\_first, zone\_time\_last**=for data gleaned from ICANN zone files, when was the first time the record was seen? when was the last time it was seen?

## **5) Searching Domain Names Seen On The LEFT Side ("rrnames")**

**-r**

# If You've Got A Domain Name and Want To Know The IPs It Has Used, Think "I'll make a -r query"

```
$ dnsdb_query.py -r farsightsecurity.com/A --sort  
time_last
```

```
;; bailiwick: farsightsecurity.com.  
;; count: 628  
;; first seen: 2013-07-17 22:08:50 -0000  
;; last seen: 2013-09-25 15:47:47 -0000  
farsightsecurity.com. IN A 149.20.4.207
```

```
;; bailiwick: farsightsecurity.com.  
;; count: 6,350  
;; first seen: 2013-09-25 15:37:03 -0000  
;; last seen: 2015-04-01 06:17:25 -0000  
farsightsecurity.com. IN A 66.160.140.81  
[etc]
```

```
["valid sort keys are bailiwick, count, rdata, rrname,  
rrtype, time_first, time_last"]
```



# Selecting Just A Specific DNS Record Type

- On the preceding slide, we specified:

```
$ dnsdb_query.py -r farsightsecurity.com/A --sort  
time_last
```

That returned only "A" (name to IPv4 address) records. Other options:

- AAAA → name to IPv6 address records
- CNAME → aliases for other names
- NS → name server records
- TXT → txt records
- SOA → start of authority records
- MX → mail exchanger records
- PTR → pointer (or "inverse address" records)
- SRV → server records
- RRSIG → one of several types of DNSSEC records

# Find All \*.farsightsecurity.com Hostnames With "AAAA" Records, Seen Since 1-May-2016?

```
$ dnsdb_query.py -r \*.farsightsecurity.com/AAAA --  
after=2016-05-01 --sort=count --reverse
```

← Note use of a backslash to avoid shell expansion of the asterisk

```
;; bailiwick: farsightsecurity.com.  
;;          count: 17,106  
;; first seen: 2015-04-01 13:05:08 -0000  
;; last seen: 2016-06-27 14:15:53 -0000  
dl.farsightsecurity.com. IN AAAA 2620:11c:f004::105
```

```
;; bailiwick: farsightsecurity.com.  
;;          count: 7,903  
;; first seen: 2015-04-09 13:31:11 -0000  
;; last seen: 2016-06-27 09:16:20 -0000  
www.farsightsecurity.com. IN AAAA 2620:11c:f004::104  
[etc]
```

# Show Me All Domains Containing **\*paypal\*** ...

**You CAN'T search for \*paypal\*** (at least not via the DNSDB API; you CAN do this via direct access to raw DNSDB files via DNSDB Export).

You CAN do a left hand wildcard **OR** a right hand wildcard, but **not both** at the same time. You also can't do an embedded ("middle") wildcard search (foo\*info)

## **WORKS:**

```
$ dnsdb_query.py -r _dkim._domainkey.\*/TXT > temp.txt
```

[shows DKIM TXT records]

[right hand queries can be potentially time consuming to run!]

## **WORKS:**

```
$ dnsdb_query.py -r \*.va > temp2.txt
```

[this command shows all domains in the Vatican's ccTLD]

[do not try this for larger domains, remember: <=1,000,000 results!]

# More -r "Wildcarding" Notes

- Wildcarding happens for an entire dot-delimited label, not just **part** of dot-delimited labels. If we're interested in stuff related to "stsauver"....

```
$ dnsdb_query.py -r *tsauver.com
```

```
HTTP Error 404: Not Found
```

```
$ dnsdb_query.py -r stsauve*
```

```
HTTP Error 404: Not Found
```

**VS....**

```
$ dnsdb_query.py -r *.stsauver.com > temp.txt
```

```
$ wc -l temp.txt
```

```
118 temp.txt
```

- Wildcards CAN match one **OR MORE** entire labels (e.g., a wildcard will find subdomains (multiple labels) rather than just matching within a single label):

```
$ dnsdb_query.py -r \*.uoregon.edu/A | grep
```

```
www\.cs\.uoregon.edu | uniq
```

```
www.cs.uoregon.edu. IN A 128.223.4.25
```

# IDNs? Yes -- In Punycode Format

- `$ dnsdb_query.py -r www.bbc.*` does NOT find `www.bbc.在线` (display-format simplified Chinese Internationalized Domain Name TLD meaning "online")
- Convert the display format label `www.bbc.在线` to punycode via the converter that's at <http://mct.verisign-grs.com/> , then try:

```
$ dnsdb_query.py -r www.bbc.xn--3ds443g
```

the punycode'd version of that domain also isn't found...

- We do have SOME punycode'd domains for that TLD, however:

```
$ dnsdb_query.py -r \*.xn--3ds443g | wc -l
62277
```

- FWIW, a list of all IDNs (actually, all top level domains) can be found at [https://en.wikipedia.org/wiki/List\\_of\\_Internet\\_top-level\\_domains](https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains)

**6) Match Domain Names Seen On The RIGHT Side ("rdata"), Typically Used To Find Domains Sharing The Same Name Server Or Domains Sharing a Common Mail Server**

**-n**

## If You've Got a Name Server's FQDN, Think -n

```
$ dnsdb_query.py -n phloem.uoregon.edu/NS > temp.txt
$ wc -l temp.txt
    1575 temp.txt
$ more temp.txt
uoregon.biz. IN NS phloem.uoregon.edu.
maoz.com. IN NS phloem.uoregon.edu.
bogus.com. IN NS phloem.uoregon.edu.
jhome.com. IN NS phloem.uoregon.edu.
o-gig.com. IN NS phloem.uoregon.edu.
otsys.com. IN NS phloem.uoregon.edu.
flyeug.com. IN NS phloem.uoregon.edu.
[etc]
```

What will -n return *besides* NS records? You may see SOA's, PTR's, CNAME's, MX's – IF you don't limit the records returned to just /NS's

# List Just Effective 2<sup>nd</sup>-Level Domains From a NS Search (Also Omitting Any .arpa. Domains)

```
$ dnsdb_query.py -n phloem.uoregon.edu/NS | awk '{print $1}' | 2nd-level-dom | grep -v "\.arpa\." | sort -u > temp.txt
```

```
$ more temp.txt
```

```
1-4-5.net.  
3bcomm.gq.  
55tours.tj.  
aboutlanegov.com.  
ac.ci.  
ac.mz.  
agd.gov.jm.  
aha-intl.org.  
ahastudyabroad.com.  
[etc]
```



# The Little 2nd-level-dom Script

```
#!/usr/bin/perl
use strict;
use warnings;
use IO::Socket::SSL::PublicSuffix;
my $pslfile = 'your_path_here/effective_tld_names.dat';
my $ps = IO::Socket::SSL::PublicSuffix->from_file($pslfile);
my $line;
foreach $line (<>) {
    chomp($line);
    my $root_domain = $ps->public_suffix($line,1);
    printf( "%s.\n", $root_domain );
}
```

The required data file? See <https://publicsuffix.org/list/>

# Testing the 2nd-Level Domain Script....

```
$ echo "www.bbc.co.uk" | 2nd-level-dom  
bbc.co.uk
```

```
$ echo "www.cs.uoregon.edu" | 2nd-level-dom  
uoregon.edu
```

```
$ echo "www.springfield.k12.or.us" | 2nd-level-dom  
springfield.k12.or.us
```

WHY would you want to do this sort of domain name reduction?  
Imagine a large file with thousands of wildcarded domains... you may not care about the random leading gibberish, you just want the base (effective 2<sup>nd</sup>-level) domains...

# Why Do You Call Them Effective 2<sup>nd</sup> Level Domains?

For "normal" top level domains, such as dot com, new domains are registered immediately under the top level domain. For instance, example.com is a typical hypothetical 2<sup>nd</sup> level domain.

Other pseudo TLDs, such as dot co dot uk, may actually see many domains registered as what look like "third" level domains (e.g., because .co.uk "uses up" the first and second level domains). Because of the effective 2<sup>nd</sup> level domain concept, "co.uk" gets treated as a single "chunk," as if it were its own TLD.

Some domains may even have longer effective TLDs, such as k12.or.us, leading to effective 2<sup>nd</sup> level domains such as springfield.k12.or.us (the three domains, k12.or.us, gets treated as a single "chunk")

See the list at [https://publicsuffix.org/list/public\\_suffix\\_list.dat](https://publicsuffix.org/list/public_suffix_list.dat)

## Another Example:

# Finding Domains That Share a Common Mail Server

```
$ dnsdb_query.py -n mx.berkeley.edu/MX > temp.txt
```

```
$ wc -l temp.txt  
292 temp.txt
```

```
$ more temp.txt  
athletics.calbears.com. IN MX 5 mx.berkeley.edu.  
engineeringpathway.com. IN MX 10 mx.berkeley.edu.  
engineeringpathway.com. IN MX 15 mx.berkeley.edu.  
berkeley.edu. IN MX 10 mx.berkeley.edu.  
berkeley.edu. IN MX 15 mx.berkeley.edu.  
ce.berkeley.edu. IN MX 10 mx.berkeley.edu.  
ce.berkeley.edu. IN MX 15 mx.berkeley.edu.  
cs.berkeley.edu. IN MX 5 mx.berkeley.edu.  
[etc]
```

## **7) Formatting CLI Query Output**

# Choice of Output Formats for **dnsdb\_query.py**

- The default output format for dnsdb\_query.py is **plain text**
- However, at your option, you can also request **JSON output**:

```
$ dnsdb_query.py -r stsauver.com/A -j
```

```
{"count": 3617, "time_first": 1407639108, "rrtype": "A",  
"rrname": "stsauver.com.", "bailiwick": "stsauver.com.",  
"rdata": ["199.48.133.170"], "time_last": 1467065919}  
{"count": 5, "time_first": 1321883198, "rrtype": "A", "rrname":  
"stsauver.com.", "bailiwick": "stsauver.com.", "rdata":  
["209.151.96.70"], "time_last": 1385778016}
```

- **JSON output can be "pretty printed" using jq**  
See <https://github.com/stedolan/jq> for information on jq  
Tips on using <https://stedolan.github.io/jq/>

# Sample jq-formatted Output From dnsdb\_query.py

```
$ dnsdb_query.py -r stsauver.com/A -j | jq '.'
```

```
{  
  "count": 3617,  
  "time_first": 1407639108,  
  "rrtype": "A",  
  "rrname": "stsauver.com.",  
  "bailiwick": "stsauver.com.",  
  "rdata": [  
    "199.48.133.170"  
  ],  
  "time_last": 1467065919  
}  
[etc]
```

# Outputting Just One Selected Field With jq

```
$ dnsdb_query.py -r farsightsecurity.com/A -j |  
jq .rdata | tr -d '[]" ' | grep -v "^$"  
66.160.140.81  
104.244.13.104  
149.20.4.207
```

Translating...

-- **jq .rdata**

*Select the rdata values from JSON output*

-- **tr -d '[]" '**

*Delete any square brackets, double quotes or spaces*

-- **grep -v "^\$"**

*Delete any blank lines*



# Reversing Domains For Ease of Sorting

```
$ dnsdb_query.py -l 1000000 -i 104.140.106.223 | awk '{print $1}' |  
2nd-level-dom | reverse-domain-names | sort -u > temp.txt
```

```
$ wc -l temp.txt
```

```
704 temp.txt
```

```
$ more temp.txt
```

```
date.15cmw
```

```
date.4a7pb
```

```
date.bhj6k
```

```
date.d19rq
```

```
date.d4ors
```

```
date.fqgd0
```

```
date.g69vs
```

```
date.gctgn
```

```
[etc]
```

# The Little reverse-domain-names Script

```
#!/usr/bin/perl

my @lines = <>;
chomp @lines;

@lines =
    map { join ".", reverse split /\./ }
    sort
    @lines;

print "$_\n" for @lines;
```

## 8) Querying DNSDB At Scale: Querying For The Domains In All Prefixes Announced by an Autonomous System

***Gentle reminder:*** You've only got a modest 100 queries/day quota, so be careful you don't exhaust ALL your queries with just one or two runs of this sort. Each time you do a `dbstdb_query.py` command, that "counts" against your limited quota, and some ASNs may have scores or even hundreds of prefixes!

# What If We Want To Look At All Prefixes For An ASN?

Assume we want to look at the domains found in passive DNS for "all prefixes associated with the ASN that's routing 104.140.106.223"

Begin by mapping the IP to ASN (see the script on the next page):

```
$ ip2asn 104.140.106.223  
62904 104.140.106.223
```

Now go to Hurricane Electric's BGP data site and check it for AS62904 to find the full list of associated prefixes...

```
http://bgp.he.net/AS62904#_prefixes  
http://bgp.he.net/AS62904#_prefixes6
```

We'll just do the IPv4 ones for now (feel free to do the IPv6 ones as an exercise)

# The ip2asn Script (As Used On The Previous Page)

```
#!/bin/sh
```

```
origip=`echo $1`  
revip=`echo $1 | sed 's/\([0-9]*\) \. \([0-9]*\) \. \([0-9]*\) \. \([0-9]*\) /\4.\3.\2.\1/'`  
listing=`host -w -t txt $  
{revip}.asn.routeviews.org 2>/dev/null | tail  
-1`  
listing2=`echo ${listing} | awk '{print $4}' |  
sed 's/"/"/g'`  
echo "${listing2} ${origip}"
```

# Processing the HE BGP Data To Build The Queries...

Copy the prefixes from [http://bgp.he.net/AS62904#\\_prefixes](http://bgp.he.net/AS62904#_prefixes) and paste them into the temporary file called temp.txt

*Massage temp.txt, keeping just the first column...*

```
$ awk '{print $1}' < temp.txt | sort -u > temp2.txt
```

*Reformat the resulting file with vi (or your favorite editor)*

```
$ vi temp2.txt
```

*[manually dd (e.g., delete) any lines other than CIDR prefixes, then add on.....]*

```
:1,$s/^/dnsdb_query.py -l 1000000 -i /
```

```
:1,$s/$/ >> temp3.txt ; sleep 2/
```

```
:1
```

*[x out (delete) one of the > signs on the first line]*

```
:wq
```

*This should leave you with a file that looks like the following...*

```
dnsdb_query.py -l 1000000 -i 104.140.104.0/22 > temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.140.140.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.140.184.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.140.216.0/24 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.140.223.0/24 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.140.252.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.140.68.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.140.80.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.193.40.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.206.104.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.206.116.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.206.144.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.206.156.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.206.172.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.206.176.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 104.206.178.0/23 >> temp3.txt ; sleep 2
[...]
dnsdb_query.py -l 1000000 -i 50.3.240.0/22 >> temp3.txt ; sleep 2
dnsdb_query.py -l 1000000 -i 75.75.227.0/24 >> temp3.txt
```

## Now Process Those Queries...

**\$ time sh -x temp2.txt**

real 7m50.108s

← less than 8 minutes wall clock time

user 1m45.125s

sys 0m13.536s

**\$ wc -l temp3.txt**

2990981 temp3.txt

← nearly 3 million total results

**\$ more temp3.txt**

ns1.finalhost.biz. IN A 104.140.104.100

ns1.betterhosting.biz. IN A 104.140.104.100

ns2.finalhost.biz. IN A 104.140.104.101

ns2.betterhosting.biz. IN A 104.140.104.101

ns1.jindalbullion.biz. IN A 104.140.106.228

ns2.jindalbullion.biz. IN A 104.140.106.228

ns1.servicioscloudperu.com. IN A 104.140.104.20

[etc]



# Simplify The Results, Output-ing The Largest Results

```
$ time awk '{print $1}' < temp3.txt | 2nd-level-dom | sort | uniq -c  
| sort -nr > temp4.txt
```

```
real    1m46.661s
```

```
user    1m50.297s
```

```
sys 0m1.216s
```

```
$ wc -l temp4.txt
```

```
84031 temp4.txt
```

```
$ more temp4.txt
```

```
277476 seguards.su.
```

```
86621 bai.su.
```

```
78603 xstats.su.
```

```
75081 westats.cc.
```

```
72534 sxo.su.
```

```
57789 sge.su.
```

```
[etc]
```

## **9) Administrivia and Debugging**

# Out of Queries?

Most DNSDB accounts have a limited quota of queries (unless you have a DNSDB account that's been configured to have an unlimited number of queries). You can check to see how many queries you've got left with code such as:

```
$ curl --header "X-API-Key: PutYourLongNumericAPIKeyHere" \
https://api.dnsdb.info/lookup/rate_limit
```

```
{
  "rate": {
    "reset": "n/a",
    "limit": "unlimited",
    "remaining": "n/a"
  }
}
```

# Some dnsdb\_query.py Errors

- `$ dnsdb_query.py foo.com`

Usage: `dnsdb_query.py [options]`

[followed by the full command line syntax summary]

You forgot to specify `-i`, `-r`, or `-n` or otherwise mis-entered something – the `dnsdb_query.py` client can't figure out what you're trying to do

- `$ dnsdb_query.py -i stsauver.com`

HTTP Error 400: Bad Request

In this example, you supplied a domain name as a `-i` parameter. The `-i` parameter needs an IP address, address range, or CIDR prefix, not a domain name.

## dnsdb\_query.py Errors (continued)

- `$ dnsdb_query.py -r 128.223.32.35`  
HTTP Error 404: Not Found

You supplied an IP address to a -r parameter, which searches the left hand side. **There are no IP addresses on the left hand side, and cannot be.** A -r parameter needs to be given a **domain name**

"Q. But Joe! What about in-addr's? They're on the left side, right?"

A. Yet, but in-addr's are actually *names*, not a purely numeric IP address...

# Searching For An in-addr.arpa

```
$ dnsdb_query.py -r 35.32.223.128.in-addr.arpa
;; bailiwick: 223.128.in-addr.arpa.
;;          count: 180,166
;; first seen: 2010-06-24 14:40:38 -0000
;; last seen: 2016-06-28 01:18:28 -0000
35.32.223.128.in-addr.arpa. IN PTR phloem.uoregon.edu.
```

*Contrast that result with with:*

```
$ dnsdb_query.py -i 128.223.32.35
phloem.uoregon.edu. IN A 128.223.32.35
c.ns.usac.edu.gt.  IN A 128.223.32.35
[etc]
```

*Note: some pDNS imputed IP RRs may not be trustworthy, and in-addr.s are definitely able to be used in misleading ways.*

## **10) Farsight's Splunk Plugin and Passive DNS**

## How Does Splunk "Fit?"

- Splunk is a very popular log management tool, terrific for digging into syslog data and similar data sources. Multiple versions of Splunk are available, including a free version, and trial versions of the Enterprise and Cloud versions.
- ***The Farsight Splunk plugin requires use of the Enterprise version of Splunk.*** To get a **free 60 day trial of Splunk Enterprise** (registration required; limited to indexing 500MB of data per day):

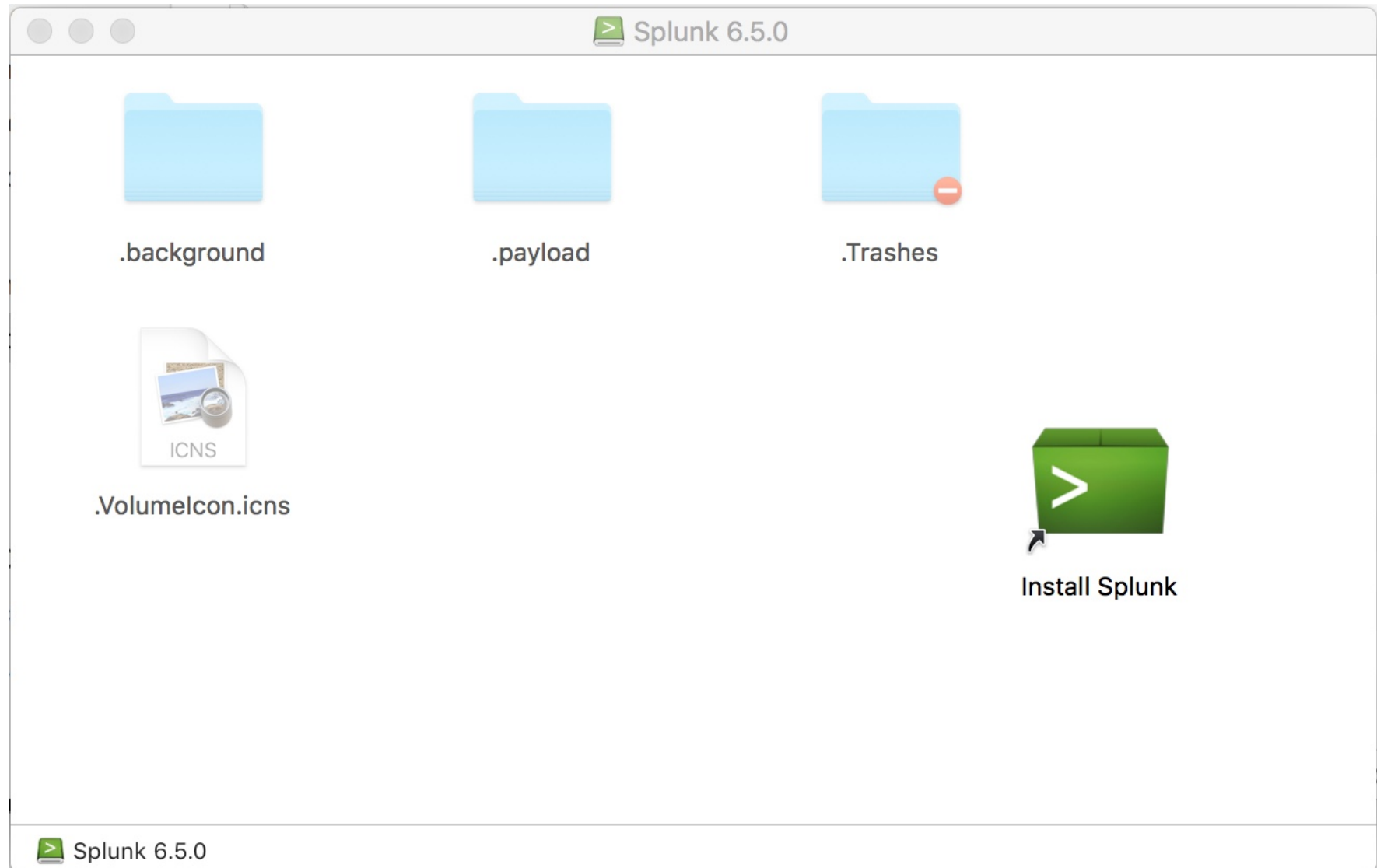
[https://www.splunk.com/en\\_us/products/splunk-enterprise.html](https://www.splunk.com/en_us/products/splunk-enterprise.html)

You can even install and run the trial version of Enterprise Splunk and the Farsight DNSDB plugin on your Mac!

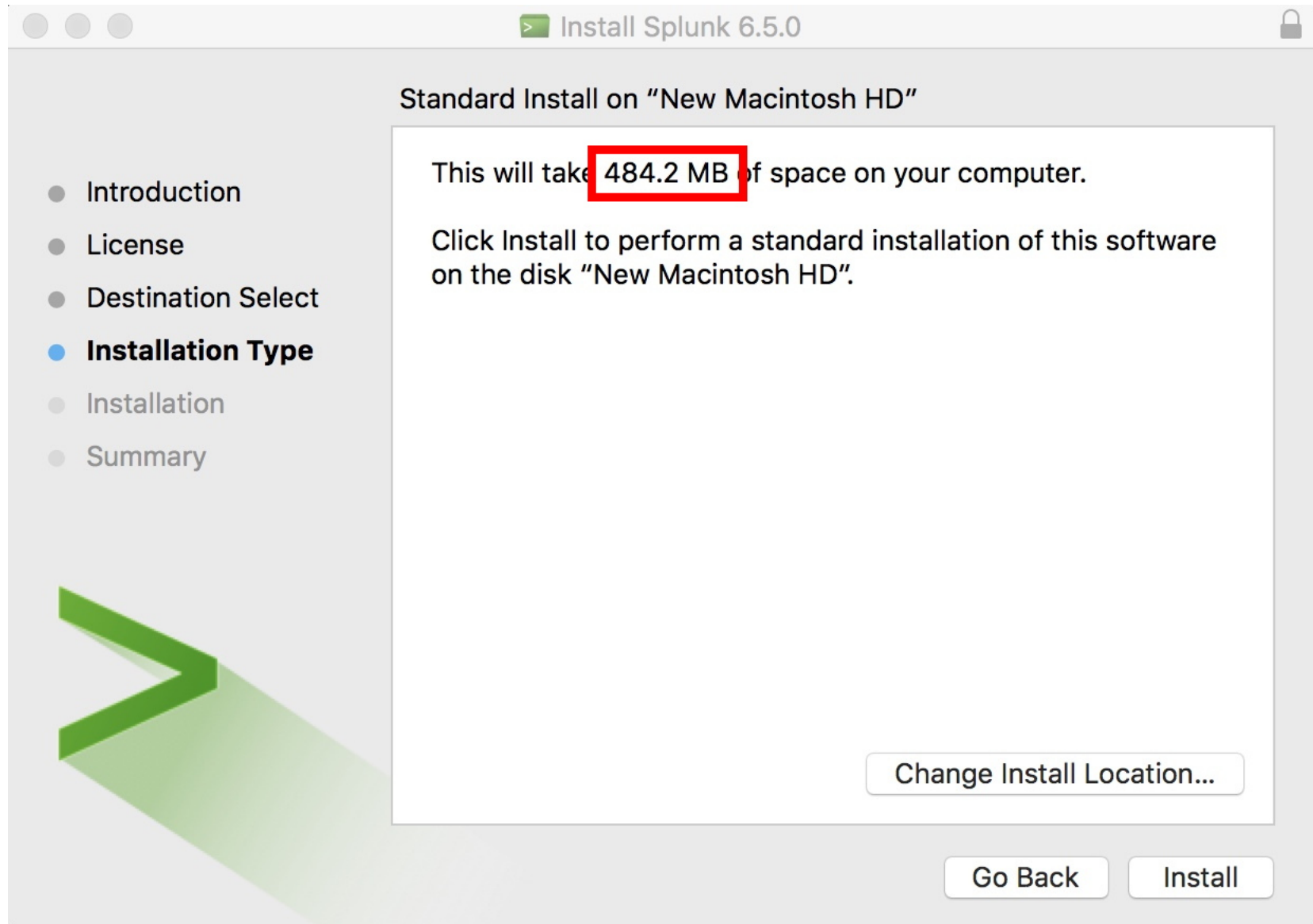
- The installation is pretty straightforward.



# Installation Is Straightforward on the Mac



# Installation (2)



# Once You've Got Splunk Installed, Launch It From Your Desktop

localhost:8000/en-US/account/logout



Search

splunk>enterprise

admin

password

Sign in

First time signing in?

You have been logged out. Log in to return to the system.

# The Farsight Security Splunk Plugin

- To get the plugin, see:

<https://www.farsightsecurity.com/FarsightDNSDBforSplunk/>

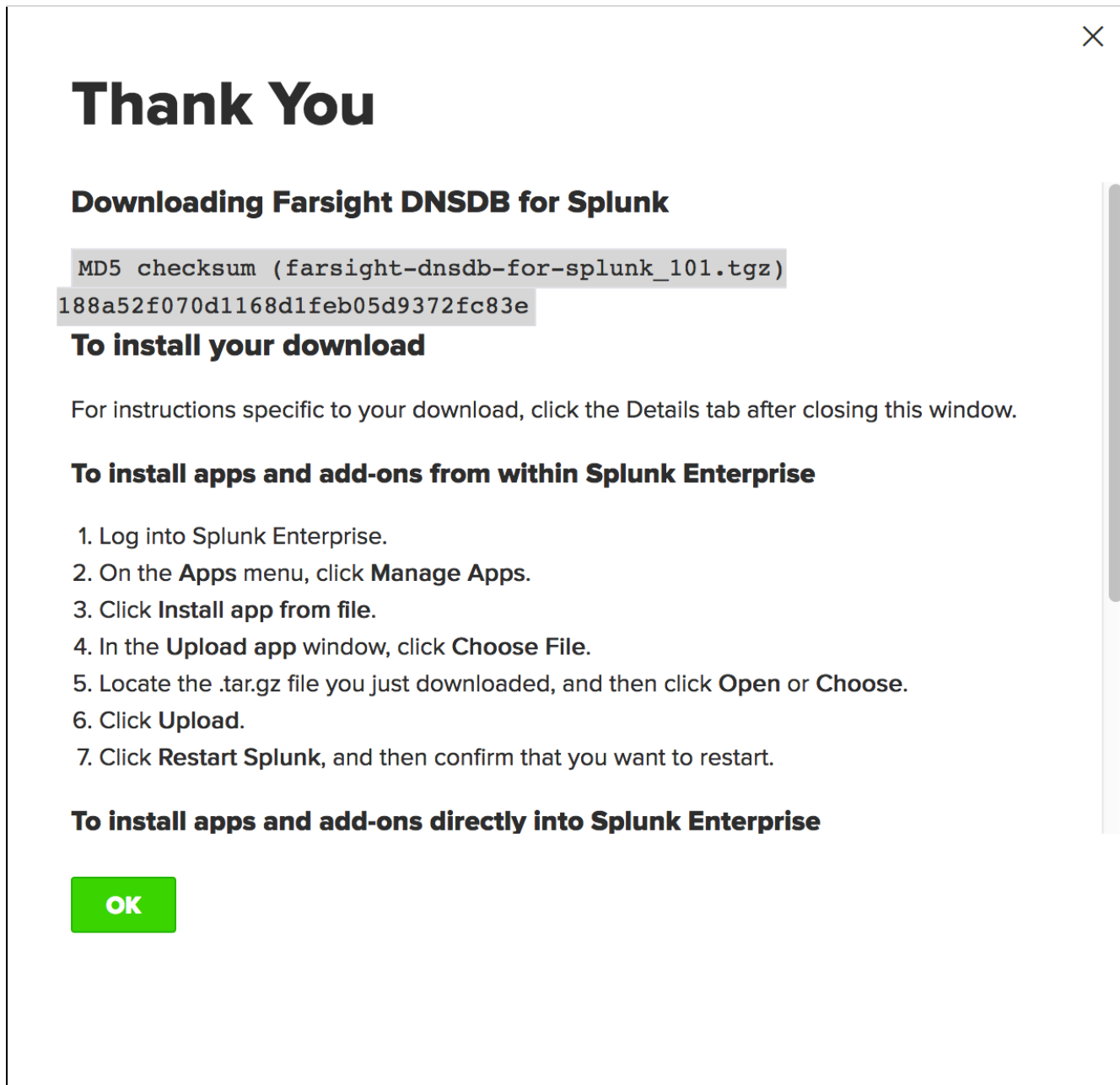
<https://splunkbase.splunk.com/app/3050/>

<https://www.farsightsecurity.com/splunk/>

**FarsightSplunkAppUserGuide.pdf** ← *must-have guide!*

- Note that you will need a Farsight API key to use the DNSDB plugin for Splunk. Your regular DNSDB API key will work fine for this purpose.

# What You See After You Download the Splunk Plugin



# Confirming The Integrity Of That Download On A Mac

It's always good to confirm the integrity of your downloads, including the Farsight Splunk plugin:

```
$ /sbin/md5 farsight-dnsdb-for-splunk_101.tgz  
MD5 (farsight-dnsdb-for-splunk_101.tgz) =  
    188a52f070d1168d1feb05d9372fc83e
```

The quoted checksum matches the highlighted value on the welcome message shown on the preceding screen, so you're good to install as described on the preceding screen.

# The Splunk DNSDB *Ad Hoc* Query Interface In Action

The screenshot displays the Splunk DNSDB Ad Hoc Query Interface. The top navigation bar includes the Splunk logo, app name 'Farsight DNSDB for Splu...', user 'Joe St Sauver', and various utility links like Messages, Settings, Activity, and Help. A search bar is also present.

The main interface features a search section with the following filters:

- Select a time range:** All time
- Select RRTYPE:** Any
- OR Add Custom RRTYPE:** ANY
- Enter an IP or Domain Name:** www.cornell.edu

A green 'Submit' button and a 'Hide Filters' link are located to the right of the filters.

The results are divided into two sections:

### DNSDB RDATA Results

RRTYPE	RData	RRName	Time First	Time Last	Zone Time First	Zone Time Last	rdata_tok	Count
CNAME	<a href="#">www.cornell.edu.</a>	<a href="#">as26.http.sasm3.net.</a>	03/21/13 17:04:09	08/06/16 04:51:33	N/A	N/A	set	70

### DNSDB RRSET Results

RRName	Time First	RData	Time Last	RRTYPE	bailiwick	Zone Time First	Zone Time Last	rrset_tok	Count
<a href="#">www.cornell.edu.</a>	N/A	<a href="#">cfprod2.cit.cornell.edu.</a>	01/11/11 02:01:33	CNAME	cornell.edu.	N/A	N/A	set	366764
<a href="#">www.cornell.edu.</a>	N/A	<a href="#">pineapple.cit.cornell.edu.</a>	01/08/11 17:04:08	CNAME	cornell.edu.	N/A	N/A	set	0
<a href="#">www.cornell.edu.</a>	N/A	<a href="#">wwwcornelledu-ssl.cit.cornell.edu.</a>	06/28/15 03:02:37	CNAME	cornell.edu.	N/A	N/A	set	1116031
<a href="#">www.cornell.edu.</a>	N/A	<a href="#">lb-cornelledu-univcom-prod.cit.cornell.edu.</a>	10/03/16 17:20:07	CNAME	cornell.edu.	N/A	N/A	set	2410629

# Using The DNSDB Splunk Connector More Aggressively

- While you can use the Farsight plugin for Splunk as just another web interface for making manual *ad hoc* DNSDB queries, **the Splunk plugin really shines as an automated way to enhance large databases you import into Splunk.**
- **Note:** enhancing large datasets can generate a large number of DNSDB queries. **Because your training query quota is very modest, we are NOT going to show you how to use Splunk this way today.**



## **11) Making Programmatic Passive DNS Queries With libcurl**

# libcurl

- To make RESTful TLS-protected queries against the DNSDB API from your own code, you'll want a library to handle the TLS "heavy lifting" work.
- I chose libcurl, the API version of the command line web client we all know and love. See <https://curl.haxx.se/libcurl/>
- Installation instructions are available at <https://curl.haxx.se/docs/install.html>
- Documentation is available at <https://curl.haxx.se/libcurl/>
- If you want to check out other alternatives, take a look at <https://curl.haxx.se/libcurl/competitors.html>

# A Few Quick libcurl Notes

- Libcurl is under **active** development.

The version I installed and used for the following sample was:

```
$ curl-config --version
```

```
libcurl 7.50.3
```

*[released on Sept 14, 2016]*

- Release history? See <https://curl.haxx.se/docs/releases.html>
- Lots of work on-going, see <https://curl.haxx.se/changes.html>

Always use a current version of libcurl

Be sure that your copy of openssl is fully up-to-date, too

# Skeleton libcurl DNSDB Query C Language Code

```
$ cat sample.c
```

```
#include <stdlib.h>
#include <string.h>
#include <curl/curl.h>
```

```
int main (void)
{
    CURL *curl;
    CURLcode res;
    char mydomain[1024], tempstring[1024],
    fullcommand[1024];
```

```
/* initialize curl. call this once and only once. */
curl_global_init(CURL_GLOBAL_ALL);
```

## Sample libcurl code (2)

```
while (scanf("%s",mydomain) == 1)
{
    /* build the command we want to pass to curl */
    /* all our commands use the same basic RESTFUL API
endpoint... */
    strcpy(fullcommand,"https://api.dnsdb.info/lookup/rrset/
name/");

    /* now tack on the domain */
    strcpy(tempstring,mydomain);
    strcat(fullcommand,tempstring);

    /* just give me a token dozen results */
    strcpy(tempstring,"?limit=12");
    strcat(fullcommand,tempstring);

    /* get curl ready for action */
    curl = curl_easy_init();
```

## Sample libcurl code (3)

```
/* pass the API key */
struct curl_slist *chunk = NULL;
chunk = curl_slist_append(chunk, "X-API-Key:
YourActualAPIKeyHere");
res = curl_easy_setopt(curl, CURLOPT_HTTPHEADER, chunk);

/* do the actual curl command */
curl_easy_setopt(curl, CURLOPT_URL, fullcommand);
res = curl_easy_perform(curl);

if(res != CURLE_OK) { return(EXIT_FAILURE); }
curl_easy_cleanup(curl);
}

curl_global_cleanup();
return (EXIT_SUCCESS);
}
```

# Sample Run

```
$ cat test-domains.txt
```

```
www.stsauver.com
```

```
[etc]
```

```
$ gcc -Wall -O3 -o sample sample.c -I/usr/local/include  
-L/usr/local/lib/ -lcurl
```

```
$ ./sample < test-domains.txt > output.txt
```

```
$ less output.txt
```

```
;; bailiwick: stsauver.com.
```

```
;; count: 1
```

```
;; first seen: 2011-04-30 04:19:39 -0000
```

```
;; last seen: 2011-04-30 04:19:39 -0000
```

```
www.stsauver.com. IN A 209.151.96.70
```

```
[etc]
```

# Notes About That Sample API Invocation

- That's just a proof of concept/illustration, it is NOT meant as production grade code (it would need to be a lot more paranoid about the way it handles some things, including doing extensive data sanitization and error checking). You've been warned!
- You can (and should!) do better, after getting the sample code to actually run, assuming you're a programmer interested in doing so.



# Exercises

**a)** Get the `dnsdb_query.py` client installed and running. Successfully make a couple of test queries, perhaps:

```
$ dnsdb_query.py -r m3aawg.org/A --after=30d
$ dnsdb_query.py -i 67.192.153.75 --after=30d
```

**b)** What 2<sup>nd</sup>-level domains share the same netblock as `m3aawg.org`? (note: this requires you to install the little 2nd-level-dom script):

```
$ dnsdb_query.py -i 67.192.153.64-67.192.153.95
--after=30d | awk '{print $1}' | 2nd-level-dom |
sort -u > temp.txt
```

**c)** Find some IPs from messages in your spam folder. Try using the passive DNS API to investigate them. If you have no spam message of your own to "mine", perhaps see:

```
-- https://www.spamhaus.org/sbl/latest/
-- http://www.senderbase.org/static/spam/#tab=2
```

## **This Ends Part II**

- This is another opportunity for you to "escape," whether because your head is full, or because you don't need a review of DNS.
- Again, however, we'd encourage you to consider staying for the rest of what we'll cover if you can.

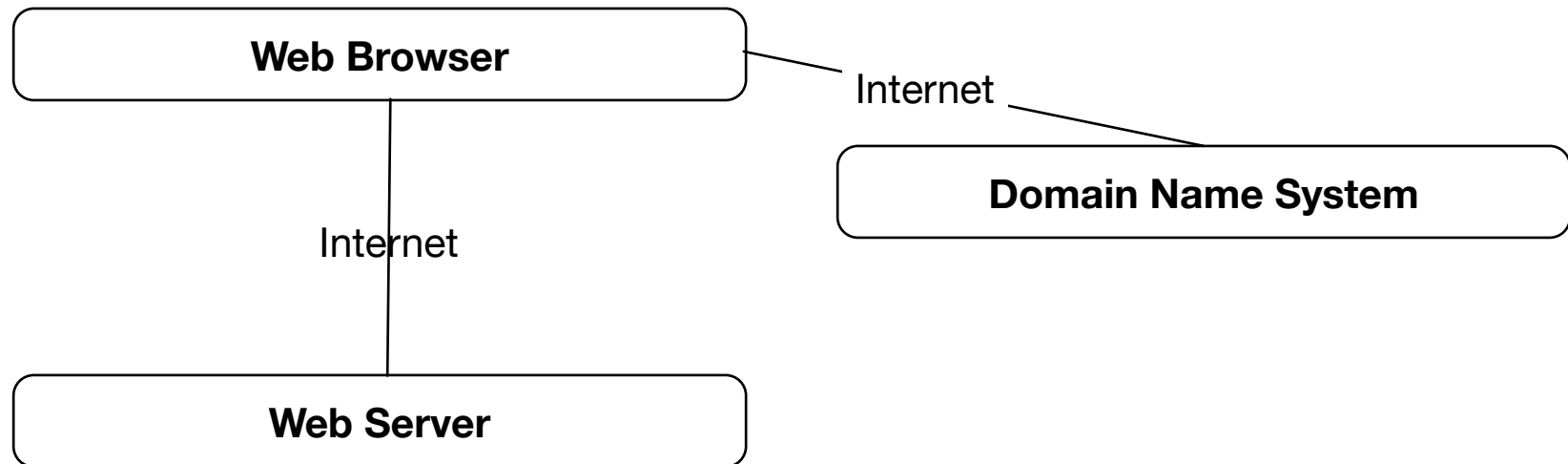
# **PART III. DNS Backfill**

## **1) Introduction**

# Referring to Computers on the Internet

- Internet users refer to Internet sites by their ***domain names***.  
A domain name that refers to a *specific* site is normally referred to as a "**Fully Qualified Domain Name**" or "**FQDN**." The FQDN of a typical web server might be **www.farsightsecurity.com**
- FQDN's normally resolve (or "get translated into") numeric IP addresses.
- FQDNs can also be set up as an alias for another domain name.
- And of course, typo'd/broken FQDNs may not resolve at all.
- Notionally, we can represent this process as shown on the following slide...

# Notional Diagram Showing Web Browser Relying On Domain Name System For the Information Needed To Reach A Web Server



## The Description In The Preceding Section Was Highly Simplified. For Example, Caching Was Omitted

- If we drill down, there's a lot more that's going on. A lot of what happens is designed to ensure that unnecessary work is avoided, and performance remains good even as DNS usage scales up.
- **Caching** (or "saving recently-received results") is a big part of that strategy.
- Pretty much all parts of the DNS ecosystem strive to **remember** what they've recently seen so they don't need to repeatedly ask – time after time after time – for answers to the same questions.
- This is key for DNS use by highly popular sites such as Amazon, Google, Facebook, Twitter, YouTube, online advertising sites, etc.
- Depending on how often things might need to change, answers may only be cached for some **few seconds**, while in other cases, answers may be made to persist for **hours, days or even weeks** – each domain administrator can pick what they think is best.

# Time-To-Live Value (TTLs)

- Cache durations are controlled by Time-To-Live values ("TTLs"). TTLs are specified in seconds. You'll see them (among lots of other places) in the output from the Unix "dig" command:

```
$ dig www.farsightsecurity.com
[...]
www.farsightsecurity.com. 3589 IN A 104.244.13.104
[snip]
```

These TTLs "decrement," or "cook down." Rechecking a bit later:

```
$ dig www.farsightsecurity.com
[...]
www.farsightsecurity.com. 3541 IN A 104.244.13.104
[snip]
```

When the TTL hits 0, the saved data will be discarded from the cache.

## Should TTLs Be Long or Short, and Why?

- **Long TTLs** (e.g., tens of thousands of seconds) mean:
  - Fewer queries for the name servers to service, which translates to lower name server load
  - If a remote name server does go down, local name servers that have cached values may not even notice
  - On the other hand, if you need to **change** the address that a server is using, you'll need to wait while a long TTL "cooks down" and expires so that the new value can get discovered.
- **Short TTLs** (hundreds of seconds) mean:
  - The name servers will see more requests
  - Resolution may be slightly slower (as some stuff that could have been efficiently cached gets re-looked-up)
  - You've got reduced breathing room in an outage, BUT more flexibility if you need to make an "emergency change" (as in cases where you may be dealing with a DDoS attack)



## UNsuccessful Resolutions

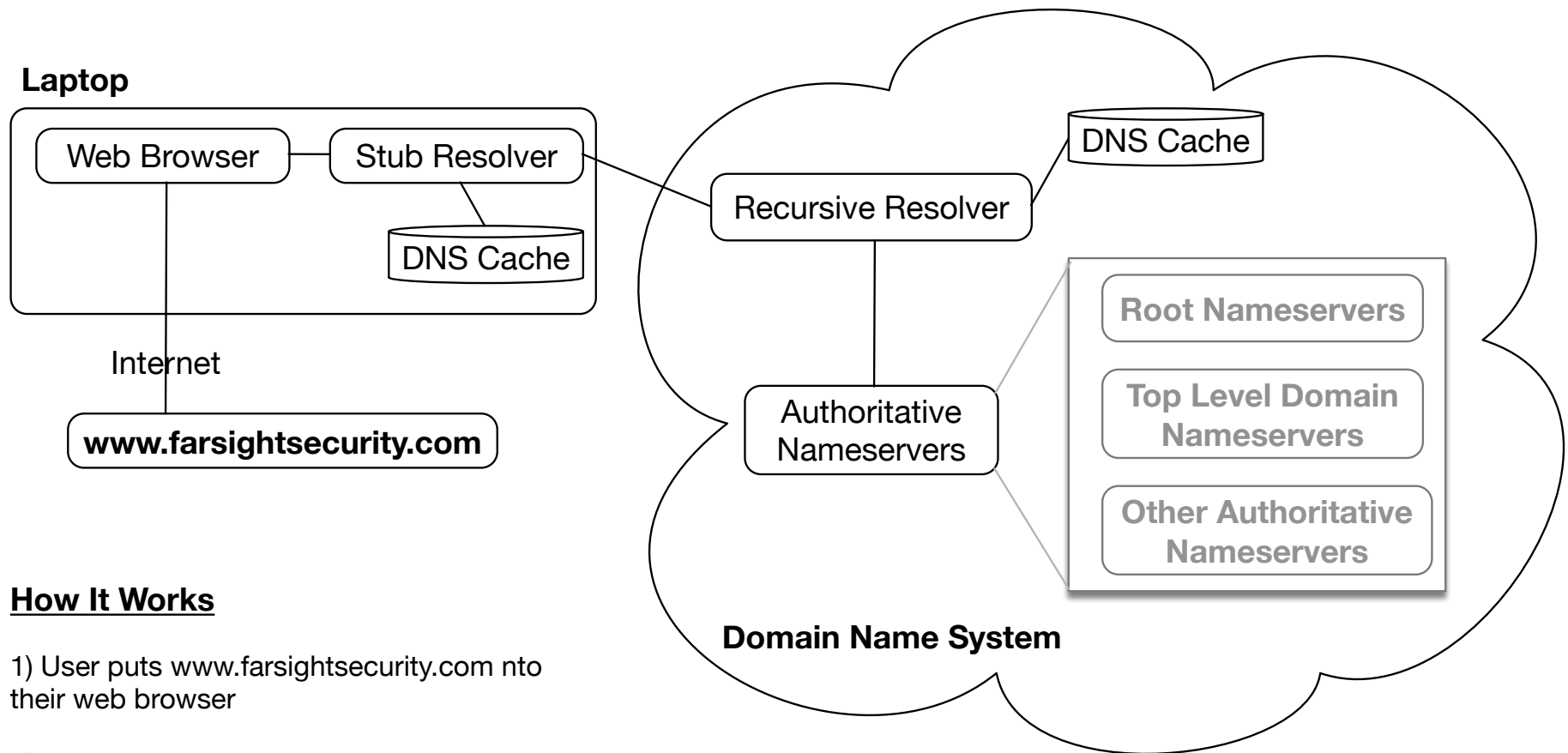
- What if a domain name CANNOT be successfully resolved? Should that NEGATIVE information be remembered, too?
- Why sure! What's the point of asking:  
"Do you know how to resolve asodaosdjasodjasodasd.com? No? Okay." and then potentially immediately repeating that question again, and again, and again, and again....
- The idea of remembering the fact that a domain name could NOT be resolved is customarily known as a "negative TTL."
- Negative TTLs, like most DNS things, are defined in RFCs. RFCs are "Requests for Comments," Internet Engineering Task Force technical standards. (Name notwithstanding, by the time they're published, no additional audience input is actually being solicited.)
- Anyhow, see <https://tools.ietf.org/html/rfc2308> for details about negative TTLs.

# Distributed Name Servers, Not One Central One

- Another thing that was simplified in the earlier diagram was the way name servers were represented. The Domain Name System was just shown in that diagram as one "blob."
- In fact, there is no single central unified "god" name server that is omniscient about all the domain names on the Internet.
- Instead, there's a distributed set of name servers that work together to resolve domain names "collaboratively" as required
- Resolution begins with the root of the domain name tree ("."). After that, authoritative name servers provides information about a domain name, OR pointers ("referrals") to a name server that will get you closer to completing resolution of your domain name.

# "Huh?"

- Start with the stub resolver (a very simple name server) running on the user's laptop.
- If the stub resolver hasn't recently cached the answer to a particular DNS query, the stub resolver asks the user's ISP's recursive resolver for help. Does the ISP's recursive resolver know how to resolve that name?
- If the ISP's recursive resolver doesn't have the required bit of information cached, the recursive resolver begins asking a series of questions of authoritative name servers in order to find the ultimate answer.
- See the diagram on the following slide.



## How It Works

1) User puts `www.farsightsecurity.com` into their web browser

2) Browser automatically asks caching stub resolver (running on the laptop), “What’s the IP address for `www.farsightsecurity.com`?”

3) If stub resolver already knows, it replies with that IP address. If it doesn’t know, it asks the recursive resolver. The recursive resolver is usually run by the Service Provider for use by all the SP’s customers.

4) If the recursive resolver already knows, it replies with that IP address. If it doesn’t know, it asks the authoritative name servers.

5) When the recursive resolver receives the IP address answer to its query from the authoritative name servers, it tells the stub resolver, which tells the web browser. The caching recursive resolver also remembers the returned IP address (for a while), in case the same question gets asked again relatively soon.

# DNS Resolution Is Iterative, and Recursive

- In the worst case, a name server is brand new and knows nothing but the address of the DNS root servers ("dot"). It would begin by asking the root name servers, "Hey, what's the address of the name servers that know about dot com?"
- After it learned the address of the dot com name servers, it would ask one of those name servers, "OK! Now let me ask you: what's the address of the name servers that know about the domain farsightsecurity.com?"
- After it learned the address of those, it would finally ask one of them, "Hey, what's the address of www.farsightsecurity.com?"
- This process is shown diagrammatically in the following graphic.

What's the IP address of  
**www.farsightsecurity.com**?  
I'll ask the recursive resolver!

➔ Recursive Resolver



*First step...*

I know the (static) address  
of the *root name servers*.  
Let me ask one of them  
for an address of one of  
the authoritative servers  
for dot com...

**The Authoritative  
Server Says...**

k.root-servers.net  
(at the bootstrap  
address 193.0.14.129)  
says "one of the name  
servers for dot com is  
e.gtld-servers.net  
at 192.12.94.30"

*Second step...*

Now that I know the address  
for a *dot com name server*,  
let me ask it for the address  
of one of the authoritative  
servers for farsightsecurity.com

e.gtld-servers.net  
(at the IP address  
192.12.94.30) says "one  
of the name servers for  
farsightsecurity.com is  
ns5.dnsmadeeasy.com at  
208.94.148.13"

*Final step...*

Now that I know the  
address for one of the  
*farsightsecurity.com name  
servers*, let me ask it for  
the address of the  
**www.farsightsecurity.com**

ns5.dnsmadeeasy.com  
(at the IP address  
208.94.148.13) says  
**"the address for  
www.farsightsecurity.com  
is 104.244.13.104"**

## Redundant Authoritative Name Servers

- There's still a bit more complexity that has been omitted from the diagrams you just saw, and that relates to redundancy: name servers are "critical infrastructure" and you always want them to work. That normally means deploying "more than one of them," even if we didn't show that on the preceding diagrams.
- This means that there will be multiple root servers, multiple top level domain servers, and multiple authoritative name servers.
- By deploying multiple name servers, even if one (or more) servers is down or unreachable, one of the others can respond, instead.
- **Q:** "But how do they stay synchronized?" **A:** "One server acts as the "master" for each zone. The slave servers periodically check that master server, and if necessary, downloads updated data."
- You can see the presence of redundant name servers in the output of the Unix "dig +trace" command on the next slides...

# How Do We Resolve `www.farsightsecurity.com`? *(showing redundant name servers at each stage)*

```
$ dig +trace www.farsightsecurity.com
```

```
[...]
```

```
.           491686  IN   NS   j.root-servers.net.           [13 root servers]
.           491686  IN   NS   m.root-servers.net.
.           491686  IN   NS   k.root-servers.net.
.           491686  IN   NS   i.root-servers.net.
.           491686  IN   NS   b.root-servers.net.
.           491686  IN   NS   c.root-servers.net.
.           491686  IN   NS   l.root-servers.net.
.           491686  IN   NS   e.root-servers.net.
.           491686  IN   NS   f.root-servers.net.
.           491686  IN   NS   d.root-servers.net.
.           491686  IN   NS   g.root-servers.net.
.           491686  IN   NS   a.root-servers.net.
.           491686  IN   NS   h.root-servers.net.
```

```
;; Received 508 bytes from 75.75.75.75#53(75.75.75.75) in 221 ms
```

```
[continued]
```



## (continued #1)

com.	172800	IN	NS	f.gtld-servers.net.	<i>[13 gTLD servers]</i>
com.	172800	IN	NS	l.gtld-servers.net.	
com.	172800	IN	NS	c.gtld-servers.net.	
com.	172800	IN	NS	h.gtld-servers.net.	
com.	172800	IN	NS	k.gtld-servers.net.	
com.	172800	IN	NS	m.gtld-servers.net.	
com.	172800	IN	NS	b.gtld-servers.net.	
com.	172800	IN	NS	j.gtld-servers.net.	
com.	172800	IN	NS	g.gtld-servers.net.	
com.	172800	IN	NS	a.gtld-servers.net.	
com.	172800	IN	NS	i.gtld-servers.net.	
com.	172800	IN	NS	e.gtld-servers.net.	
com.	172800	IN	NS	d.gtld-servers.net.	

;; Received 502 bytes from 192.112.36.4#53(192.112.36.4) in 279 ms

## (continued #2)

```
farsightsecurity.com. 172800 IN NS ns5.dnsmadeeasy.com. [3 servers]
farsightsecurity.com. 172800 IN NS ns6.dnsmadeeasy.com.
farsightsecurity.com. 172800 IN NS ns7.dnsmadeeasy.com.
;; Received 184 bytes from 192.43.172.30#53(192.43.172.30) in 85 ms
```

```
www.farsightsecurity.com. 3600 IN A 104.244.13.104
farsightsecurity.com. 3600 IN NS ns7.dnsmadeeasy.com. [3 servers]
farsightsecurity.com. 3600 IN NS ns5.dnsmadeeasy.com.
farsightsecurity.com. 3600 IN NS ns6.dnsmadeeasy.com.
;; Received 124 bytes from 208.94.148.13#53(208.94.148.13) in 17 ms
```

# No Matter What Gets Shown, There Are Actually More Than 13 Root/gTLD Name Servers

- While the `dig +trace` command we just showed mentioned 13 root name servers and 13 gTLD name servers, there are actually *\*many\** more than that. This is accomplished via **"anycast."**
- Anycast involves announcing the *\*same\** IP address block at multiple sites. For example, ICANN advertises the "L" root server IP address block from 158 sites (as of the time this was written), and the University of Maryland announces the "D" root server IP address block from 106 sites (as of the time this was written)
- Thanks to the magic of the Internet's wide area routing system ("BGP"), you always automatically end up using the "closest" instance for any given name server.

# Look at Just The Locations Where The "L" Root Lives

www.root-servers.org

Search

Locations: Sites: 158

- Abidjan, Cote d'Ivoire
- Al Muharraq, Bahrain
- Amman, Jordan
- Anchorage, United States
- Ankara, Turkey
- Apia, Samoa
- Asuncion, Paraguay
- Atlanta, United States
- Baku, Azerbaijan
- Bangkok, Thailand
- Beijing, People's Republic of China
- Beirut, Lebanon
- Belem, Brazil
- Belgrade, Serbia
- Belo Horizonte, Brazil
- Blantyre, Malawi
- Bogota, Colombia
- Bouake, Cote d'Ivoire
- Brasilia, Brazil
- Bratislava, Slovakia
- Brisbane, Australia
- Brussels, Belgium
- Callao, Peru
- Campinas, Brazil
- Cape Town, South Africa
- Chicago, United States
- Christchurch, New Zealand
- Cochabamba, Bolivia
- Copenhagen, Denmark
- Curitiba, Brazil
- Dakar, Senegal
- Dammam, Saudi Arabia
- Dar es Salaam, Tanzania
- Denver, United States
- Dortmund, Germany
- Dubai, United Arab Emirates
- Dundee, United Kingdom
- Dusseldorf, Germany
- El Prat de Llobregat, Spain
- Ezeiza, Argentina
- Florence, Italy
- Florianopolis, Brazil
- Fortaleza, Brazil
- Geneva, Switzerland
- Haarlemmermeer, Netherlands
- Hagatna, Guam
- Hamburg, Germany
- Heraklion, Greece
- Honiara, Solomon Islands
- Honolulu, United States
- Incheon, South Korea
- Islamabad-Rawalpindi, Pakistan
- Istanbul, Turkey
- Jakarta, Indonesia
- Jeddah, Saudi Arabia
- Johannesburg, South Africa
- Kalamazoo-Battle Creek, United States
- Kharkiv, Ukraine
- Kiev, Ukraine
- Kolkata, India
- Kolonia, Federated States of Micronesia
- Kuwait City, Kuwait
- Lahore, Pakistan
- Lawrence, United States
- Leeds-Bradford, England
- London, United Kingdom
- Londrina, Brazil
- Los Angeles, United States
- Lyon, France
- Mahe, Seychelles
- Maiquetia, Venezuela
- Malmo, Sweden
- Mandalay, Myanmar
- Mangere, New Zealand
- Marseille, France
- Mascot, Australia
- Melbourne, Australia
- Metro Manila, Philippines
- Miami, United States
- Minsk, Belarus
- Mississauga, Canada
- Monterrey, Mexico
- Montevideo, Uruguay
- Moscow, Russia
- Mumbai, India
- Muscat, Oman
- Nadi, Fiji
- Nairobi, Kenya
- Natal, Brazil
- Noumea, New Caledonia
- Odessa, Ukraine
- Ottawa, Canada
- Papeete, French Polynesia
- Paris, France
- Paris-Orly, France
- Perth, Australia

[Green=IPv6 enabled; Brown=IPv4 only]

## 2) Domain Name Structure and Nomenclature

*"Look, I know there's a lot of jargon but  
some of these really are self-explanatory."*

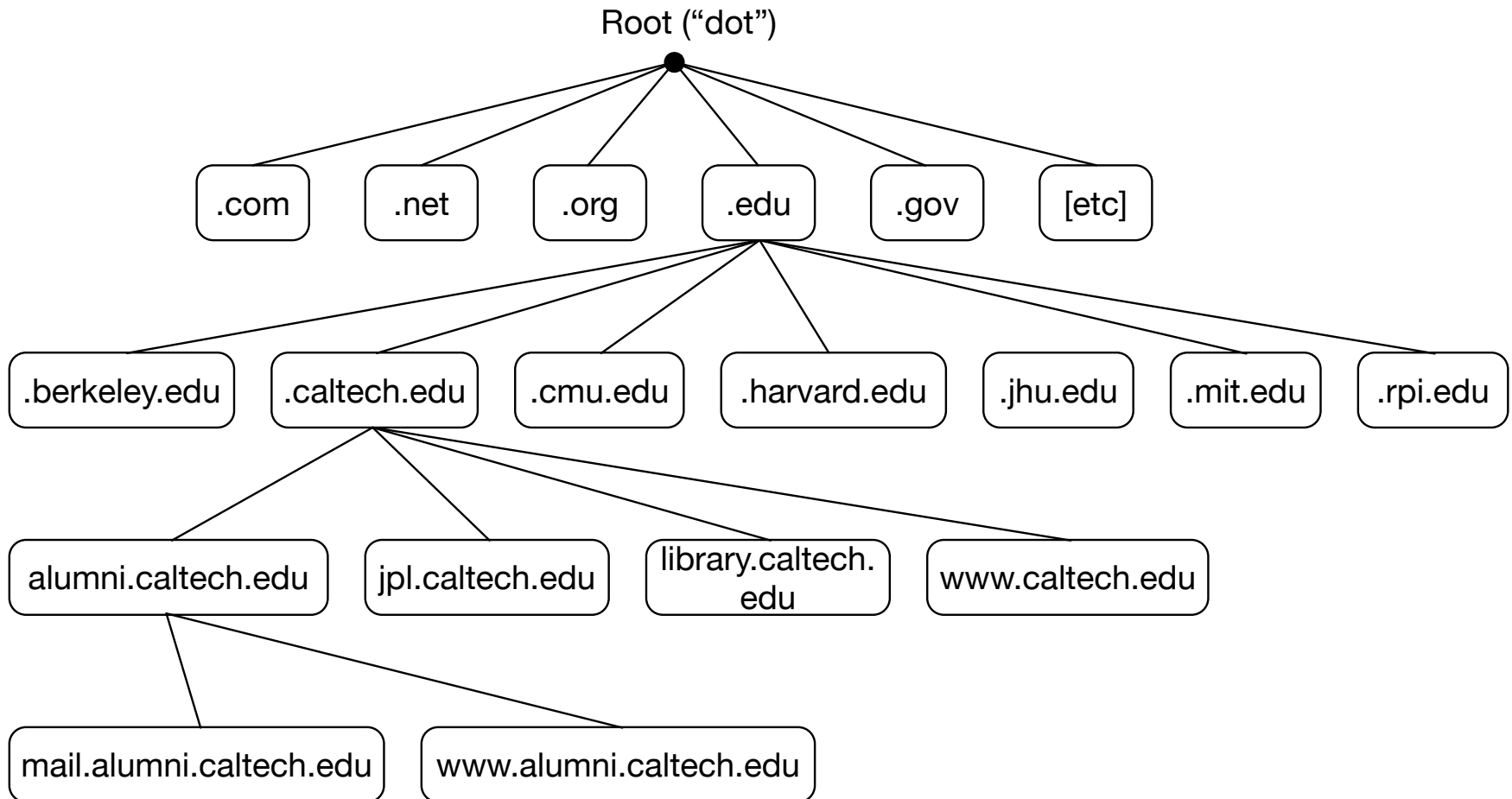
Dan Rydell (a character in the movie "Sports Night"),  
in response to being asked if throwing a  
"perfect game" is "good"

Like sports, DNS also has a lot of jargon,  
but unfortunately very little of it is self-explanatory

# Fully Qualified Domain Name Nomenclature

- Decomposing a FQDN, such as `www.alumni.caltech.edu`, we see:
  - `edu` is a *Top Level Domain* (TLD)
  - `caltech.edu` is the *2<sup>nd</sup> Level Domain* (what a domain registrant normally registers with a registrar)
  - `alumni.caltech.edu` is a *subdomain* of `caltech.edu`
  - `www.alumni.caltech.edu` is the *FQDN* (or "hostname")
- There's also an implicit/unwritten "dot" at the right side of `www.alumni.caltech.edu` – that's the "root" of the entire DNS. It is normally not shown or specified when domain names are used (although you will see it in DNSDB output)
- See the following diagram.

# Hierarchical Domains



# Subdomains

- In a large organization, some units may be (more-or-less) autonomous, and might want to manage their own domain names. For example, a university's engineering school and the alumni association might both want to run their own DNS.
- In that case, the school's central DNS admin team might create and delegate two subdomains of the institution's domain, e.g., `engineering.example.edu`, and `alumni.example.edu`
- Departmental DNS people might then create FQDN's under that subdomain, or delegate further sub-sub-domains, perhaps:
  - `chemical.engineering.example.edu`
  - `civil.engineering.example.edu`
  - `electrical.engineering.example.edu`
  - `mechanical.engineering.example.edu`
- And going one step further, a FQDN (hostname) might look like `ilovemanualtransmissions.mechanical.engineering.example.edu`

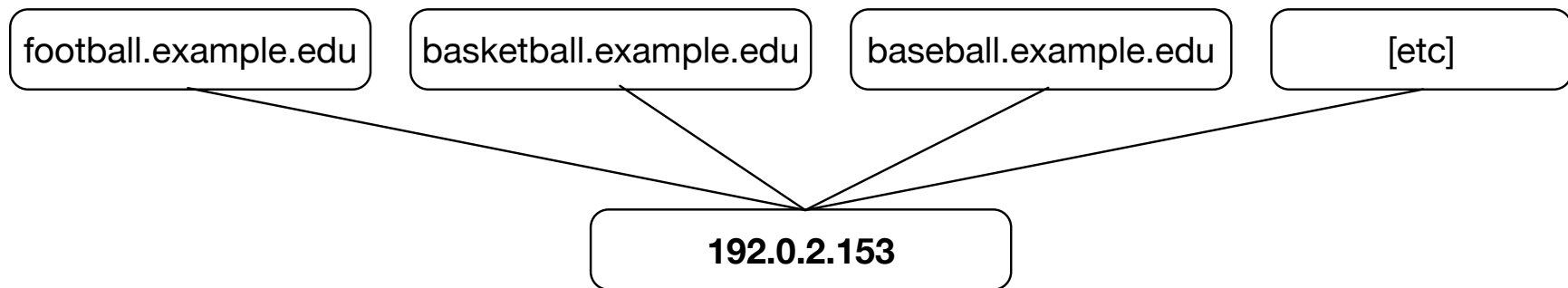


# How Do I Know If A Label Is A "Hostname"?

- When you see a label such as "alumni.caltech.edu", you can't tell just by looking at it whether that's a hostname in it's own right, or just a subdomain (with other FQDNs appearing below it), or both.
- Caltech.edu (all by itself) might be/is a perfectly OK hostname.
- You also can't tell (just by looking at it) whether there's a server "behind" any given name, or whether all servers are only assigned *below* that label, or, for that matter, **what** a server's being used for.
- Domain name labels can sometimes be used in misleading ways:
  - www.something will *often* be a web server, but it doesn't *have* to be
  - mail.something will *often* be a mail server, but it doesn't *have* to be
  - ns1.something will *often* be a name server, but it doesn't *have* to be

# More Than One FQDN May Point At One IP

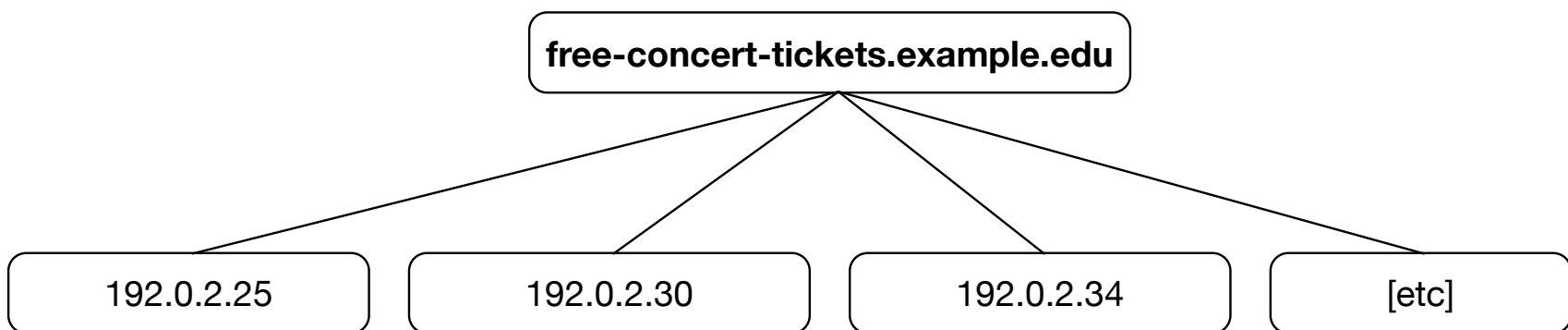
- For example, imagine a big web server.
- It might host hundreds or even thousands of different smaller web sites all serviced from a single IP address. For example, maybe a site has a **set** of athletic web sites all handled by a big shared web server:



[Note: 192.0.2.153 is an IP from a special block of IP's reserved for use in documentation, see <https://tools.ietf.org/html/rfc5735> ]

# A Single FQDN May Point At More Than One IP

- On the other hand, imagine a **really** busy web server. A single site might have so much (potentially crushing!) traffic that it needs multiple physical servers to handle it all.
- The domain name system makes it easy to accommodate that load: just point the busy web server name at a set of servers, each on a different IP address. Traffic gets sent to IPs on a round-robin basis, much the way a commercial load balancer (such as an F5 box) often is used....



# **A Domain Name Can Point At Another Domain Name Rather Than At An IP Address**

- Domain names that point at other domain names are normally called "CNAMES." You can think of these as additional aliases that can be used in lieu of a host's primary name. For example:

senate.uoregon.edu. IN CNAME faprod.uoregon.edu.  
parking.uoregon.edu. IN CNAME faprod.uoregon.edu.  
committees.uoregon.edu. IN CNAME faprod.uoregon.edu.  
facilities.uoregon.edu. IN CNAME faprod.uoregon.edu.  
courseevals.uoregon.edu. IN CNAME faprod.uoregon.edu.  
riskmanagement.uoregon.edu. IN CNAME faprod.uoregon.edu.  
[etc]

- If faprod.uoregon.edu ever needed to move to a new IP address, you could just change it, not each and every one of these sites...

# IP Addresses Can (Sometimes) Also Be Mapped Back To A Domain Name

- Just as you can map a domain name to an IP address, you can also (sometimes) map an IP address back to a domain name:
- Normal domain name to IP DNS resolution might look like:  
drupal-cluster5.uoregon.edu → 128.223.142.244
- Reverse (IP to domain name) DNS resolution might look like:  
244.142.223.128.in-addr.arpa →  
drupal-cluster5.uoregon.edu
- These are normally called "inverse address" or "pointer" records. Does the following help you to see why?

```
$ dig -x 128.223.142.244
```

```
[...]
```

```
244.142.223.128.in-addr.arpa. 86400 IN PTR  
drupal-cluster5.uoregon.edu.
```

# Generic TLDs (Regular Domain Names)

- gTLDs are managed by ICANN, the Internet Corporation for Assigned Names and Numbers. **Traditional generic TLDs** include .com, .net, .org, .edu, .gov, .mil, .int, .biz, .info, .mobi, .name, .jobs, .me, .xxx, .pro, .aero, .jobs, .asia, .museum, .tel, and .travel.
- Some gTLDs are **unrestricted** (e.g., anyone can register a domain in them), such as .com, .net, .org, and .info
- Other gTLDs **are limited to just particular communities**. For example: .edu's are now limited to just accredited US colleges and universities; .gov is restricted to just American government agencies; .mil is limited to just the US Army, Navy, Air Force, Marines, and Coast Guard; .int is just for international organizations (such as NATO)
- gTLDs are most common, but there are also many other types of top level domains.

# ccTLDs

- ccTLDs are two letter "Country Code" top level domains such as:

.ar Argentina	.au Australia	.be Belgium
.br Brazil	.ca Canada	.ch Switzerland
.co Colombia	.cn China	.de Germany
.es Spain	.eu European Union	.fr France
.in India	.it Italy	.nl Netherlands
.pl Poland	.ru Russia	.tk Tokelau
.uk United Kingdom	.us United States	etc.

- Most ccTLD abbreviations are self explanatory, but there are some noteworthy exceptions such as ch= Switzerland (Cantons of Helvetica), de=Deutschland, dz = Algeria (Dzayer in Berber), etc.
- There are 254 ccTLDs in total, see the country code TLDs listed as part of <https://icannwiki.com/CcTLD>

# Weird ccTLD Factoids

- So-called "open" ccTLDs have been "disconnected" from the countries they're named after. For example, .tk was sold by the Tokelau government and is now largely used for free domain registrations (and not by the 1,400 or so people of Tokelau)
- Some ccTLDs register new 2<sup>nd</sup>-level domains differently than .com, .net, etc., do. For example, most commercial domains in the uk get registered under .co.uk, not directly under .uk (we call .co.uk and similarly situated domains "effective TLDs")
- Some legacy ccTLDs still exist even if the associated country no longer exist. For example: .su (Soviet Union)
- ccTLDs are not subject to ICANN contractual requirements. This means that they are run as the individual ccTLD operators deem appropriate. In some cases, this translates to things like no requirement for publicly available whois service (whois normally tells you who owns/controls a given domain)



# IDNs (Internationalized Domain Names)

- Most TLDs use "normal" ("Roman" or "Latin") letters, numbers and/or hyphens, and can be up to 63 characters long.
- IDNs were created to meet the needs of those using languages with other scripts, such as:
  - Arabic
  - Chinese
  - Cyrillic (Russian)
  - Greek
  - Hangul (Korean)
  - Hebrew
  - Indian (Bangla, Devanagari, Gujarati, Gurmukhi, Tamil, Telugu)
  - Katakana (Japanese), etc.
- IDNs get represented two ways: as a "U label" (presentation format, using the international character set, such as 中信), and as an "A label" (the ASCII-encoded form, such as xn--fiq64b)

# Internationalized Domain Names (cont)

- This is now a valid domain name: はじめよう.みんな  
Plug it into your browser, and you'll be taken to that web site.  
[Anyone want to attempt pronunciation of that domain name?]
- That same name in Punycode ASCII format gets written:  
xn--p8j9a0d9c9a.xn--q9jyb4c

*Note:* Punycode-format IDNs always begin "xn—"

- Need to manually convert between the two? See  
<http://idna-converter.com/>
- More resources relating to IDN's:  
<https://www.icann.org/resources/pages/more-2012-05-08-en>

# New gTLDs

- Recently ICANN has begun the creation of over 1,300 new gTLDs, such as .xyz, .red, .faith, .london, stream, etc., see 1,188 listed at <http://newgtlds.icann.org/en/program-status/delegated-strings> for the new gTLDs delegated to-date
- Some of these new gTLDs are restricted to designated communities, others are open and new domains can be registered by anyone (sometimes for as little as \$0.49/domain, as can be seen at <https://tld-list.com/> (click on "Cheapest Register" to sort by domain name cost))
- Trademark holders can use the new Trademark Clearinghouse to protect their marks against possible dilution by registrants in the new gTLDs, see <http://newgtlds.icann.org/en/about/trademark-clearinghouse>.
- *Question:* has **your** company protected **its own** marks via the Trademark Clearinghouse?

# **We've Been Talking About Names, What About IPs?**

- Even though domain names are important, and we like talking about domain names, computers and computer networks also need IP addresses.
- Just as there's a lot of jargon around domain names, there's a lot of jargon around IP addresses, too.

### **3) IPv4 and IPv6 Addresses**

# The Critical Role of IP Addresses

- While we often refer to Internet resources according to their domain names, **each server/workstation/laptop/tablet/smartphone/etc. requires an IP address to connect to the Internet.**
- A host may have a traditional ("IPv4") address, a much longer "new-fangled" IPv6 address, or both.
- We'll talk about IPv4 addresses a little first.

# IPv4 Addresses and Prefixes

- Traditional ("IPv4") IP addresses, consist of four integers separated by dots (that's why these are also often known as "dotted quads"). Each of the four values can have a value from 0 to 255. For example: 104.244.13.104
- Ranges of IPv4 addresses are expressed in a variety of forms, most simply by stating a starting and ending address. For example: 104.244.12.0 - 104.244.15.255
- IPv4 address ranges can also be expressed as "CIDR prefixes." CIDR prefixes consist of a starting address and a "mask length" or "prefix length" indicative of the size of the block (see the table in a few pages). For example, 104.244.12.0 - 104.244.15.255 can also be represented in CIDR notation as 104.244.12.0/22

# Classless (CIDR) Addressing: Much More Flexibility...

- Some commonly seen CIDR lengths – note the pattern (each longer mask has ½ the number of addresses of the preceding)

[...]		/23	512 IPv4 addresses
/14	262,144 IPv4 addresses	/24	256 IPv4 addresses
/15	131,072 IPv4 addresses	/25	128 IPv4 addresses
/16	65,536 IPv4 addresses	/26	64 IPv4 addresses
/17	32,768 IPv4 addresses	/27	32 IPv4 addresses
/18	16,384 IPv4 addresses	/28	16 IPv4 addresses
/19	8,192 IPv4 addresses	/29	8 IPv4 addresses
/20	4,096 IPv4 addresses	/30	4 IPv4 addresses
/21	2,048 IPv4 addresses	/31	2 IPv4 addresses
/22	1,024 IPv4 addresses	/32	1 IPv4 address



# Whois

- Whois tells you "who is" responsible for a given number or name. You can query whois for domain names, IP addresses, and ASNs.
- You can access whois information from the command line. In a Unix terminal window, you can enter:

```
$ whois farsightsecurity.com
```

- You can also query whois over the web from a variety of 3<sup>rd</sup> party web gateways, such as <https://dnsquery.org/whois/>
- Some domain whois data may be hidden behind proxy/private registrations, obfuscating the true owner of the domain name.

# Types of IPv4 Addresses

- **Globally routable IP addresses:** these addresses are usable worldwide, and will be unique worldwide. These are "regular" or "normal" IPv4 addresses. We can further subdivide regular IPv4 addresses into:
  - **Provider independent (PI) globally routable IP addresses:** these addresses can be used with any single provider, or with any combination of providers.
  - **Provider assigned (PA) globally routable IP addresses:** these addresses are provided for your use by your upstream provider. If you leave that provider, you "can't take them with you." If you leave, you'll need to "renumber out of that block" into a new block of address space, returning your previous block for some other customer to eventually use.

Do NOT attempt to have some third party ISP advertise PA address space on your behalf (the third party ISP normally won't do it, anyhow).

# RFC1918 Addresses for Private Intranets

- RFC 1918 address space consists of three IP address ranges:  
10.0.0.0-10.255.255.255 (10/8 prefix)  
172.16.0.0-172.31.255.255 (172.16/12 prefix)  
192.168.0.0-192.168.255.255 (192.168/16 prefix)
- You can use these addresses however you like – **within your own site**. However, you CANNOT route those addresses to the Internet as a whole. To understand why, realize that thousands of sites might have networks using addresses from the address range 10.0.0.0-10.255.255.255 – how would the Internet know which one of those was the "right one?"
- RFC1918 addresses are perfect, however, for things like networked printers that should only be used by local people.

# Network Address Translation (NAT)

- You may already be using RFC1918 addresses on your own home network. A common scenario is that you:
  - Receive a publicly routable address from Comcast, CenturyLink, etc.
  - You then use a wireless access point/"router" to connect all of your family's devices to the Internet, sharing that one public IP.
  - This is done through use of RFC 1918 address space within your home, with those private addresses automatically "translated" to the single public IP address you received from your ISP.
- Sounds like magic, right? It sort of is, but, unfortunately, it does have some downsides:
  - It is difficult (although not impossible) to run servers from behind a NAT'd IP address (most broadband providers ban home servers, anyhow)
  - NAT boxes, because they need to rewrite IP addresses, need to be "protocol aware" – this means that H.323 video conferencing may have trouble traversing some NAT boxes
  - An ISP can't tell which host behind a NAT box might be botted, if bad traffic is seen getting emitted from a shared IP

# Localhost/Loopback Address

- 127.0.0.1 is a special IPv4 "loopback" address that refers to the host itself.

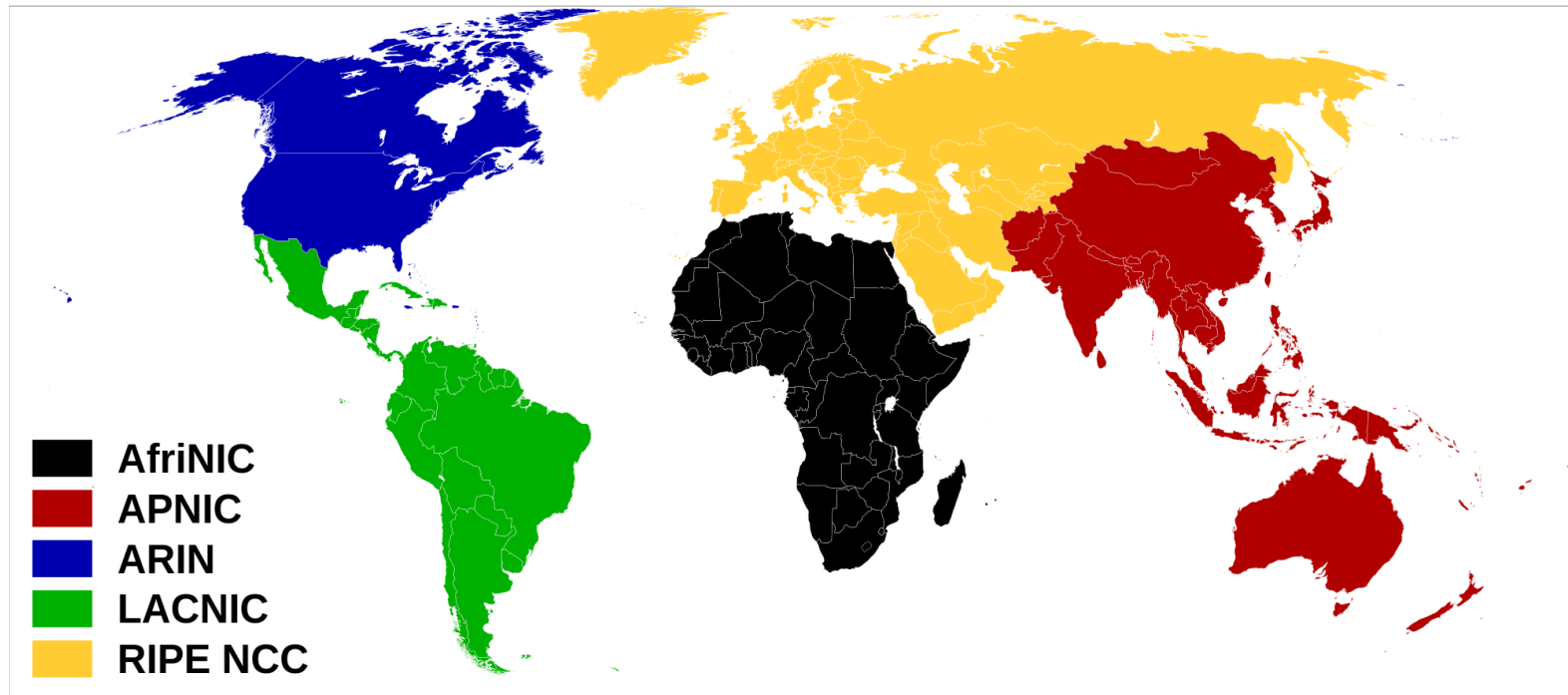
The host can talk to its own 127.0.0.1 (and other addresses in 127.0.0.0/8), but no one is able to talk to someone else's 127.0.0.0 addresses.

# Assigning IPs to Systems: Static vs Dynamic

- Servers in a department probably need "static" IP addresses. These are IPs that have been manually/permanently assigned for use by that particular computer. They will normally be associated with a meaningful public hostname, such as finance.example.edu
- User laptops, tablets, smart phones, etc. will probably be automatically given **temporary, or "dynamically assigned," IP addresses from a site's DHCP server**. These address "leases" might last for just a few hours, but can be renewed if needed for a longer period. DHCP names are seldom very exciting (perhaps looking like dhcp-250.example.edu or dyn-110-72.something.edu)
- Samantha Smith's laptop might use dhcp-250.example.edu this morning, and dhcp-18.example.edu this afternoon, while Jimmy Johnson's iPad might use dhcp-250.example.edu this afternoon.
- DHCP allows IPs to be easily assigned, and allows a relatively small pool of addresses to be shared by a large(r) pool of users.

# Where Do Sites Request Blocks of IP Addresses?

- Small customers (such as an individual or a new startup business) normally get small blocks of IP address space from their upstream Internet Service Provider.
- Larger sites (such as an established national or regional ISP that is multi-homed (connected to two or more upstream providers using BGP)) normally request provider-independent address space from their regional registry.



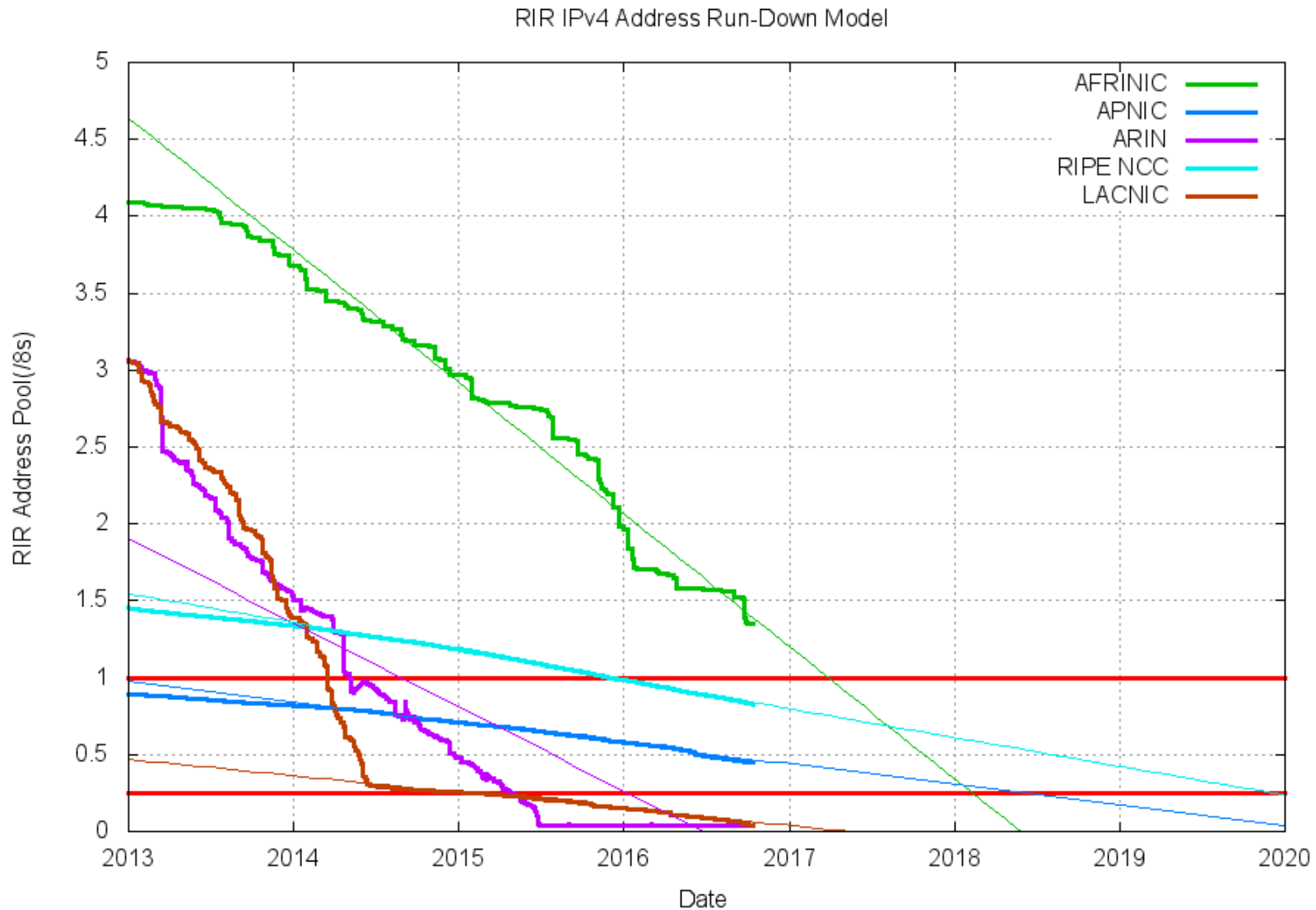
[https://upload.wikimedia.org/wikipedia/commons/thumb/9/95/Regional\\_Internet\\_Registries\\_world\\_map.svg/2000px-Regional\\_Internet\\_Registries\\_world\\_map.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/9/95/Regional_Internet_Registries_world_map.svg/2000px-Regional_Internet_Registries_world_map.svg.png)

# **Sadly, The World's Largely Run Out of Unassigned IPv4 Addresses**

- IPv4 addresses are just 32 bits long. That means that there's only a maximum of  $2^{32}=4,294,967,296$  potential addresses available.
- There's actually not that many potential addresses, and in fact they've run out fast. See the graph on the following page.



# IPv4 Address Run Out By Regional Registry



Source: <http://www.potaroo.net/tools/ipv4/>

# Fortunately We Have Lots of IPv6 Addresses

- IPv6 addresses are a little different. For example, IPv6 addresses are written a little differently than IPv4 addresses.
- IPv6 addresses are normally written as a series of 32 hexadecimal letters, broken into colon-separated 4-character chunks.
- One such sample IPv6 address: 2620:11c:f004::104
- When writing IPv6 addresses, **leading zeros *within a chunk*** can be omitted (as was done in the "11c" and "104" chunks above). If there's a **run of zeros** in an IPv6 address, those can be replaced, in **one location** only, with **two successive colons**.
- **Verbose form:** 2620:011c:f004:0000:0000:0000:0000:0104

# "What's That About Hexadecimal Numbers Again?"

- Decimal (base 10) numbers are written using the digits 0 - 9. Hexadecimal numbers (base 16) are written using 0 - 9 **PLUS** the letters A (=10), B (=11), C (=12), D (=13), E (=14), F (=15)

Place	Decimal:	Hexadecimal:
1 <sup>st</sup>	$10^0=1's$	$16^0=1's$
2 <sup>nd</sup>	$10^1=10's$	$16^1=16's$
3 <sup>rd</sup>	$10^2=100's$	$16^2=256's$
4 <sup>th</sup>	$10^3=1,000's$	$16^3=4096's$

- Convert 5178 (decimal) to hexadecimal:**  
5178/4096=1 with 1082 left over  
1082/256=4 with 58 left over  
58/16=3 with 10 left over  
10/1=10 (written as "A") → 143A (hexadecimal)

# "So How Many IPv6 Addresses Can I Get?"

- The Internet may be just about out of IPv4 addresses, but IPv6 addresses are available in barge-size quantities. Then again, we use IPv6 addresses a little more liberally, too.
- In IPv4 world, a typical subnet was a /24, or 256 addresses.
- In the IPv6 world, a normal subnet is an IPv6 /64, or  $2^{64} = 18,446,744,073,709,551,616$  addresses. That's many more addresses than all the addresses in the IPv4 Internet, worldwide.
- Each IPv6 subnet gets that many addresses even though a typical subnet might only have a handful (or perhaps a few hundred) of those addresses in use. This seems horribly weird and wasteful, but don't worry, IPv6 is intended to be used this way.
- Mentally think "an IPv6 /64 represents a host, or maybe a small number of hosts," and you'll be okay. If you think "an IPv6 /64 is 18,446,744,073,709,551,616 addresses," you'll need therapy.

# Normal IPv6 Allocations and IPv6 CIDR Sizes

- A single local network normally uses a /64.
- A smaller end site needing multiple subnets normally gets an IPv6 /56. How many IPv6 /64's can that site deploy? (see table)
- A provider (such as a small local ISP) might gets an IPv6 /48; how many small end sites can they give a /56 to? (see table)
- If you use up what you've initially received, you can request more [more about IPv6 allocations: <https://tools.ietf.org/html/rfc6177> ]

<b>Prefix</b>	<b># of IPv6 48's</b>	<b># of IPv6 /56's</b>	<b># of IPv6 /64's</b>
/36	4,096	1,048,576	268,435,456
/40	256	65,536	16,777,216
/44	16	4,096	1,048,576
/48	1	256	65,536
/56		1	256
/64			1

# Types of IPv6 Addresses

- There are a wide number of different types of IPv6 addresses, in part because there were many attempts to map IPv4 addresses "automagically" over into IPv6 through gateway services.  
**A summary of those types can be seen at [https://www.ripe.net/participate/member-support/new-lir/ipv6\\_reference\\_card.pdf](https://www.ripe.net/participate/member-support/new-lir/ipv6_reference_card.pdf)**
- The most commonly seen types of native IPv6 addresses?
  - Globally routeable unicast addresses
  - Unique Local Addresses, the IPv6 equivalent of RFC1918 IPv4 private address space (all from fc00::/7)
  - Link Local Addresses, most commonly seen when looking at IPv6-enabled hosts that don't have a globally routeable address or ULA address (seeing just an fe80 address is often a sign that you don't actually have IPv6 internet connectivity)

## **4) DNS: Connecting Domain Names to IPs**

# Connecting Names to IPs and IPs to Names

- We've talked a little about names and a little about IPs.
- Now let's come back to talking a little about how DNS connects the two together.



# Resource Record Format

- DNS relationships are defined in "resource records."
- A sample DNS resource record looks like this:

```
www.farsightsecurity.com. 3600 IN A 104.244.13.104
```

- The first field is the domain name.
- The second field is the TTL, or time to live (which we've already introduced in our earlier caching discussion).
- The third field, the "class code," will almost always be "IN" ("Internet"), and can normally be ignored.
- The fourth field is the DNS record type. In this example, we have an "A" record (a normal IPv4 name-to-IP address record)
- The fifth and final field is RDATA, in this case, an IPv4 address.

# DNS Record Types

- As we've previously mentioned, here are many different types of DNS records:
  - A: Normal domain name → IPv4 address record
  - AAAA: Normal domain name → IPv6 address records
  - CNAME: Domain name → different domain name
  - NS: Name Server records (tells what name servers to use for a domain)
  - MX: Mail eXchanger records (where should I send email for this domain?)
  - TXT: Text records, capable of carrying arbitrary text
  - SOA: Start Of Authority records, defines the TTL and other zone parameters
  - SRV: Server pointer records, explaining where to find a service (IP address and port information)
  - And there are others, but only a comparative handful get widely used.

# Record Types Seen In A Day's Worth of DNS Data

Observations	% of Obs	Record Type & Code
16,964,386	57.27%	A (1)
9,460,957	31.94%	SOA (6)
1,745,213	5.89%	CNAME (5)
714,677	2.41%	NS (2)
259,468	0.88%	PTR (12)
204,785	0.69%	MX (15)
149,771	0.51%	TXT (16)
100,424	0.34%	AAAA (28)
18,140	0.06%	NULL (10)
2,393	0.01%	SRV (33)
440	<0.01%	SPF (99)
77	<0.01%	WKS (11)
59	<0.01%	<UNKNOWN>(1169)
7	<0.01%	DNAME (39)
4	<0.01%	LOC (29)
3	<0.01%	HINFO (13)
1	<0.01%	<UNKNOWN>(4652)
1	<0.01%	<UNKNOWN>(4097)
1	<0.01%	RP (17)
<b>29,620,807</b>	<b>100.00%</b>	

# A Less Commonly Talked-About Record Type: SOA

- `$ dig farsightsecurity.com SOA +short`  
`fsi.io. hostmaster.fsi.io. 2016101202 7200 3600`  
`604800 3600`  
Decoded...
- Primary master server for the zone? That's at the FQDN fsi.io
- Contact address? hostmaster@fsi.io (sub an @ sign for the first .)
- Serial Number/date gets updated whenever the zone is changed
- Refresh: secondary servers are told to wait this long between checks to see if primary has updated (7200 seconds in this case)
- Retry: secondary can't talk to primary? try again in 3600 seconds
- Expire: secondary can't talk to primary? treat cached zone data as being still valid for a week (604800 seconds)
- Negative caching time: if you get an NXDOMAIN for names in this domain, remember that negative response for 3600 seconds
- See also dig's `+multiline` option for SOA records

# Wildcard DNS

- Normally DNS servers only send out answers for specific FQDNs.
- However, some name servers will get configured to support wildcarding, in which case the name server will answer for ANY pattern matching the wildcard
- For example, assume you're a marketer, and you've uniquely tagged a hidden tracking URL in each message you send. E.G.,  
`xfasd-ttsuoa.campaign7.example.com`  
`hruak-adhdwr.campaign7.example.com`  
`[etc]`
- You want your name server to respond to each such address when resolved. Using DNS wildcarding, you could tell a name server to return a response for `*.campaign7.example.com`

# Zone Transfers

- If you wanted to get a copy of all the records in a zone, such as all the hosts defined under uoregon.edu, you'd do a "zone transfer." Zone transfers can be full (AXFR) or incremental (IXFR).
- Because a full listing of all hosts in a domain would be hugely useful if you were a hacker/cracker planning an attack, normally zone transfers are only allowed for a small set of explicitly authorized parties who have a legitimate need for such access. (Sometimes zone files transfers are accidentally/unintentionally allowed as a result of operator error/configuration problems, see the next slide)
- By the way, you now also understand why providers of passive DNS normally like to carefully vet their users – having access to passive DNS is like being able to get copies of a site's zone file.

# Example of a Zone Transfer Leaking: .kp

## North Korea .kp TLD Zone Data

On Sept 19, 2016 at approximately 10:00PM (PDT), one of North Korea's top level nameservers was accidentally configured to allow global DNS zone transfers. This allows anyone who performs an `AXFR` (zone transfer) request to the country's `ns2.kptc.kp` nameserver to get a copy of the nation's top level DNS data. This was detected by the [TLDR Project](#) - an effort to attempt zone transfers against all top level domain (TLD) nameservers every three hours and keep a running Github repo with the resulting data. This data gives us a better picture of North Korea's domains and top level DNS.

[Click here for the commit showing this incident.](#)

~~As of the time of this writing, zone transfers are still enabled for multiple `.kp` top level domains via `ns2.kptc.kp`.~~

**Update:** North Korea has now patched this issue, however this project will continue to scan the Internet for future slip-ups just like this one.

# DNS Response Codes

- When a DNS query is made, it may succeed or it may fail. The status code that's returned is called as a "DNS response code."
- `$ dig google.com`  
[...]  
;; ->>HEADER<<- opcode: QUERY, **status: NOERROR** [etc]  
google.com. 180 IN A 216.58.193.110
- **NOERROR == normal successful completion status code**
- `$ dig asasdjasjnasfjnasfnkafs.com`  
[...]  
;; ->>HEADER<<- opcode: QUERY, **status: NXDOMAIN** [etc]
- **NXDOMAIN == domain does not exist**
- There are other (relatively obscure) additional error codes, too



**Thanks For the Chance to Talk Today!**

Are There Any Questions?

Please remember to fill out your  
session evaluation!

**THANK YOU for Participating In  
M3AAWG's Paris Training Program**