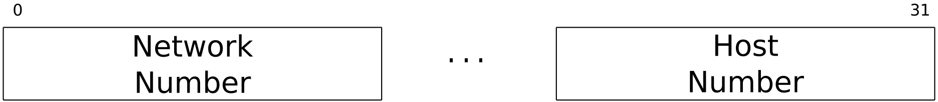
IP Addresses; DNS

Obtaining—and Stealing—IP Addresses

IP Addresses

- How are they structured?
- Where do they come from?
- How do hosts acquire IP addresses?
- How does a host learn another host's IP address?

The Structure of an IP Address



- IP addresses are separated into a *network number* and a *host number* within that network
- The boundary varies, often even within an organization
- Columbia University owns 128.59/16: a 16-bit network number, i.e., addresses 128.59.0.0 – 128.59.255.255
- (More on this next class)

IP Address Allocation

- IP addresses are allocated hierarchically
- Major ISPs and enterprises can acquire their own large blocks of addresses
- They then suballocate within their blocks to their customers (for ISPs) or constituent units (for enterprises)
- Anyone can use `10/8`, `172.16/12`, `192.168/16` internally
- Example: the CS department has (among other networks) `128.59.32.0/21`
- Organizations hand out host numbers within their address block

IP Address Assignment

- Host number `0` or all `0`s means “no address assigned”
- Host number `1...1` is *broadcast*: send to all hosts on this network
- A network number of all 1s is also broadcast
- (N.B.: the link layer has to map that to the MAC broadcast address)
- Other addresses are assigned either statically or via the *Dynamic Host Configuration Protocol* (DHCP)

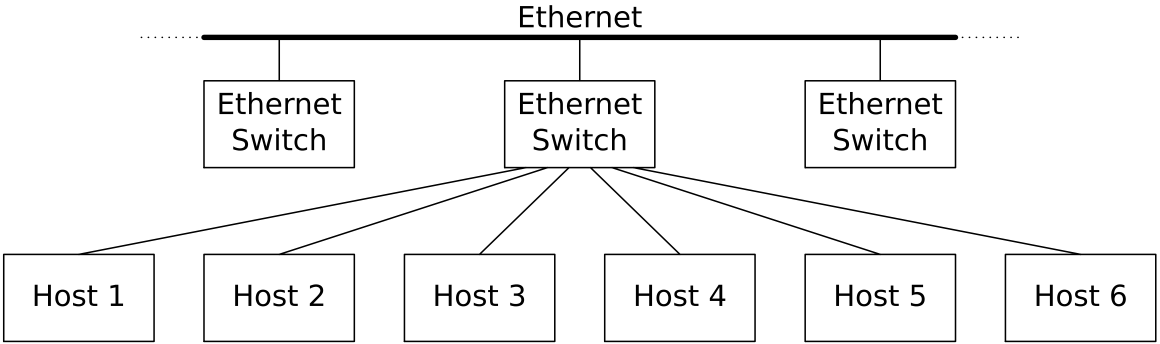
DHCP

- At boot time, hosts send broadcast messages requesting an IP address
- The request can specify a hostname or a MAC address, depending on local policy (the latter is far more common)
- Permanent addresses can be configured per host, or an address pool can be used, or both
- At Columbia, registered hosts have permanent addresses; our phones, laptops, etc., are given addresses from a pool
- DHCP-assigned addresses carry a *lease time*
- Hosts *renew* DHCP leases before they expire

Security Issues in IP Address Assignment

- Any host can simply start using another IP address
- Any host can issue a DHCP request using another host's MAC address and/or name
- ☞ It's even possible for a program to change the host's MAC address on most platforms—the hardware only sets the default
- We can only rarely prevent the problem—but we can often detect it

Ethernet Switches



Ethernet Switches

- Ethernet *switches*—layer 1/2 devices—learn which MAC addresses are on each port by looking at source MAC addresses on packets
- This is dynamic—computers can move, e.g., if there are different WiFi access points on different switch ports
- Enterprise-grade switches (i.e., not the cheap desktop ones I use in my apartment) can log which MAC addresses have shown up on which ports
- Some cheat and look at layer 3: they record—and sometimes filter—which *IP addresses* are on each port
- Look at your logs!
- Note that filtering requires a static topology—not always the case

Other Ways to Detect Address Theft

- If a host sees two different ARP replies for the same IP address, it can yell
- If a host sees an ARP request for its own IP address, it can yell
- If a host receives a TCP packet for a connection that is purportedly its own but does not exist, it can log that, and yell if it sees to many of them
- Besides, it not only can, it should send a RST packet
- (For reasons I don't want to get into, switches will sometimes (but rarely) forward a packet into "incorrect" ports based on MAC address)
- Besides—if you use end-to-end encryption, the other party should detect a bad certificate

The Domain Name System

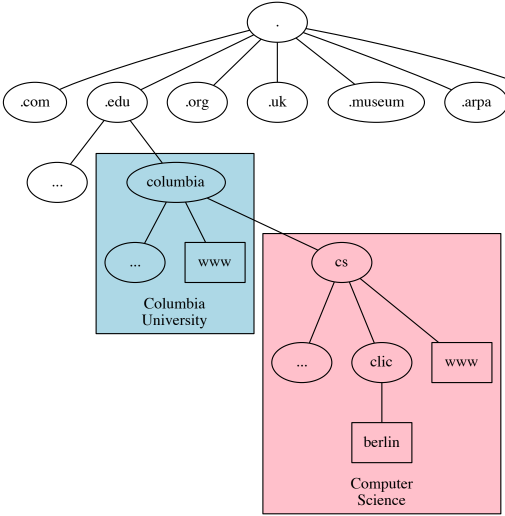
Learning IP Addresses

- We normally connect to hosts by hostname, not IP address
- `www.cs.columbia.edu` is much easier to remember than `128.59.11.206`
- The answer is the *Domain Name System* (DNS), a distributed, approximately correct database

Organization of the DNS

- The DNS is tree-structured and organized into *zones*
- Zones are administrative boundaries, not tree levels
- Thus, `columbia.edu` and `cs.columbia.edu` are separate zones, but `clic.cs.columbia.edu` is not a zone, even though it has subnodes
- Each zone adminstrator controls the content of that zone

The DNS


.
.com
.edu
.org
.uk
.museum
.arpa
...
...
columbia
...
www
Columbia University
cs
...
clic
www
berlin
Computer Science

DNS Properties

- Designed for many different namespaces, not just the Internet (but that's not really used)
- Many different types of records possible at each node
- Use for address lookup, email handling, and more

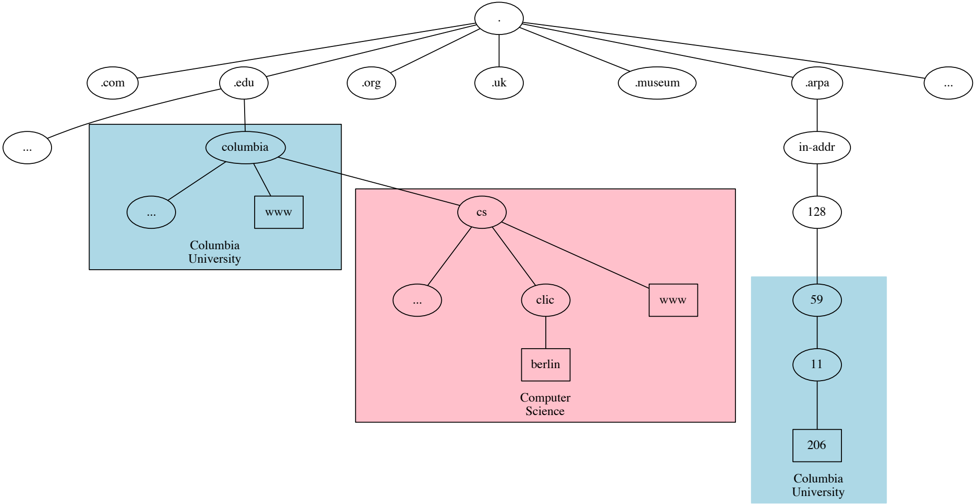
DNS Entry for Facebook.com

```
facebook.com.  3600 IN MX   10 smtpin.vvv.facebook.com.
facebook.com. 86400 IN NS   d.ns.facebook.com.
facebook.com. 86400 IN NS   c.ns.facebook.com.
facebook.com. 86400 IN NS   b.ns.facebook.com.
facebook.com. 86400 IN NS   a.ns.facebook.com.
facebook.com.  3600 IN CAA  0 issue "digicert.com"
facebook.com. 86400 IN TXT  "v=spf1 redirect=_spf.facebook.com"
facebook.com.  7200 IN TXT  "google-site-verification=A2WZWCNQHrGV_TWwKh6KHY90tY0SHZo_RnyMJoDaG0s"
facebook.com.  7200 IN TXT  "google-site-verification=wdH5DTJTc9AYNwVunSVFeK0hYDGUIEOGb-RReU6pJlY"
facebook.com.  3600 IN SOA  a.ns.facebook.com. dns.facebook.com. 1605582192 14400 1800 604800 300
facebook.com.   300 IN A    157.240.205.35
facebook.com.   300 IN AAAA 2a03:2880:f113:81:face:b00c:0:25de
```

Looking Up Hostnames

- Suppose we have an IP address and want to look up the hostname
- We have no idea what part of the name tree to look in!
- Instead, there's a separate tree, the `in-addr.arpa` tree, which maps IP addresses to hostnames
- Remember that IP addresses are allocated hierarchically
- Convert, e.g., `128.59.11.206` to `206.11.59.128.in-addr.arpa` to match the tree structure
- Note: this diagram is a bit oversimplified, in that it assumes that delegation happens only at byte boundaries

The `in-addr.arpa` Tree



DNS Name Resolution

DNS Name Resolution

- The DNS tree seems straightforward enough—but how it's used is rather complex
- (Well, the reality of the tree is far more complex than I've indicated, but. . .)
- Queries have to start from the root—but we can't have every Internet-connected computer banging on "the" root nameserver all the time
- Solution: caching and multiple levels of queriers

Looking up www.cs.columbia.edu

- Every actual DNS querier has, pre-configured, the IP addresses of 13 root servers
- (Why 13? DNS uses UDP, with a maximum packet size of 512 bytes, and that's all that will fit. . .)
- The querier asks a root server for www.cs.columbia.edu's address
- That server replies, in effect, "Here's a server for .edu; ask it"
- The query is repeated; the .edu server says, "Here's a server for columbia.edu; ask it"
- Etc.

Caching

- Every DNS record carries a *time to live* (TTL) field
- This Facebook A (address) record may be retained for five minutes

  `facebook.com. 300 IN A 157.240.205.35`

- NS records—name server records, which tell who can answer queries for a zone—last longer

  `facebook.com. 86400 IN NS d.ns.facebook.com.`

- But life is more complex still...

A DNS Query

```
$ dig www.columbia.edu
```

☞ *We want an "A" (IPv4 address) record*

```
;; QUESTION SECTION:
;www.columbia.edu.              IN      A
```

☞ *The answer has two levels of alias*

```
;; ANSWER SECTION:
www.columbia.edu.       3600    IN      CNAME   www.a.columbia.edu.
www.a.columbia.edu.     3600    IN      CNAME   www.wwwr53.cc.columbia.edu.
www.wwwr53.cc.columbia.edu. 60  IN      A       128.59.105.24
```

☞ *Identify the authoritative name servers for this zone*

```
;; AUTHORITY SECTION:
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1000.awsdns-61.net.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-508.awsdns-63.com.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1308.awsdns-35.org.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1721.awsdns-23.co.uk.
```

☞ *Pass along IPv4 and IPv6 addresses for one of them*

```
;; ADDITIONAL SECTION:
ns-1721.awsdns-23.co.uk. 172800 IN      A       205.251.198.185
ns-1721.awsdns-23.co.uk. 172800 IN      AAAA    2600:9000:5306:b900::1
```

A DNS Query

```
$ dig www.columbia.edu
```

☞ *We want an "A" (IPv4 address) record*

```
;; QUESTION SECTION:
;www.columbia.edu.              IN      A
```

☞ *The answer has two levels of alias*

```
;; ANSWER SECTION:
www.columbia.edu.       3600    IN      CNAME   www.a.columbia.edu.
www.a.columbia.edu.     3600    IN      CNAME   www.wwwr53.cc.columbia.edu.
www.wwwr53.cc.columbia.edu. 60  IN      A       128.59.105.24
```

☞ *Identify the authoritative name servers for this zone*

```
;; AUTHORITY SECTION:
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1000.awsdns-61.net.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-508.awsdns-63.com.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1308.awsdns-35.org.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1721.awsdns-23.co.uk.
```

☞ *Pass along IPv4 and IPv6 addresses for one of them*

```
;; ADDITIONAL SECTION:
ns-1721.awsdns-23.co.uk. 172800 IN      A       205.251.198.185
ns-1721.awsdns-23.co.uk. 172800 IN      AAAA    2600:9000:5306:b900::1
```

A DNS Query

```
$ dig www.columbia.edu
```

☞ *We want an "A" (IPv4 address) record*

```
;; QUESTION SECTION:
;www.columbia.edu.              IN      A
```

☞ *The answer has two levels of alias*

```
;; ANSWER SECTION:
www.columbia.edu.       3600    IN      CNAME   www.a.columbia.edu.
www.a.columbia.edu.     3600    IN      CNAME   www.wwwr53.cc.columbia.edu.
www.wwwr53.cc.columbia.edu. 60  IN      A       128.59.105.24
```

☞ *Identify the authoritative name servers for this zone*

```
;; AUTHORITY SECTION:
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1000.awsdns-61.net.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-508.awsdns-63.com.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1308.awsdns-35.org.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1721.awsdns-23.co.uk.
```

☞ *Pass along IPv4 and IPv6 addresses for one of them*

```
;; ADDITIONAL SECTION:
ns-1721.awsdns-23.co.uk. 172800 IN      A       205.251.198.185
ns-1721.awsdns-23.co.uk. 172800 IN      AAAA    2600:9000:5306:b900::1
```

A DNS Query

```
$ dig www.columbia.edu
```

☞ *We want an "A" (IPv4 address) record*

```
;; QUESTION SECTION:
;www.columbia.edu.              IN      A
```

☞ *The answer has two levels of alias*

```
;; ANSWER SECTION:
www.columbia.edu.       3600    IN      CNAME   www.a.columbia.edu.
www.a.columbia.edu.     3600    IN      CNAME   www.wwwr53.cc.columbia.edu.
www.wwwr53.cc.columbia.edu. 60  IN      A       128.59.105.24
```

☞ *Identify the authoritative name servers for this zone*

```
;; AUTHORITY SECTION:
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1000.awsdns-61.net.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-508.awsdns-63.com.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1308.awsdns-35.org.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1721.awsdns-23.co.uk.
```

☞ *Pass along IPv4 and IPv6 addresses for one of them*

```
;; ADDITIONAL SECTION:
ns-1721.awsdns-23.co.uk. 172800 IN      A       205.251.198.185
ns-1721.awsdns-23.co.uk. 172800 IN      AAAA    2600:9000:5306:b900::1
```

A DNS Query

```
$ dig www.columbia.edu
```

☞ *We want an "A" (IPv4 address) record*

```
;; QUESTION SECTION:
;www.columbia.edu.              IN      A
```

☞ *The answer has two levels of alias*

```
;; ANSWER SECTION:
www.columbia.edu.       3600    IN      CNAME   www.a.columbia.edu.
www.a.columbia.edu.     3600    IN      CNAME   www.wwwr53.cc.columbia.edu.
www.wwwr53.cc.columbia.edu. 60  IN      A       128.59.105.24
```

☞ *Identify the authoritative name servers for this zone*

```
;; AUTHORITY SECTION:
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1000.awsdns-61.net.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-508.awsdns-63.com.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1308.awsdns-35.org.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1721.awsdns-23.co.uk.
```

☞ *Pass along IPv4 and IPv6 addresses for one of them*

```
;; ADDITIONAL SECTION:
ns-1721.awsdns-23.co.uk. 172800 IN      A       205.251.198.185
ns-1721.awsdns-23.co.uk. 172800 IN      AAAA    2600:9000:5306:b900::1
```

A DNS Query

```
$ dig www.columbia.edu
```

☞ *We want an "A" (IPv4 address) record*

```
;; QUESTION SECTION:
;www.columbia.edu.              IN      A
```

☞ *The answer has two levels of alias*

```
;; ANSWER SECTION:
www.columbia.edu.       3600    IN      CNAME   www.a.columbia.edu.
www.a.columbia.edu.     3600    IN      CNAME   www.wwwr53.cc.columbia.edu.
www.wwwr53.cc.columbia.edu. 60  IN      A       128.59.105.24
```

☞ *Identify the authoritative name servers for this zone*

```
;; AUTHORITY SECTION:
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1000.awsdns-61.net.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-508.awsdns-63.com.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1308.awsdns-35.org.
wwwr53.cc.columbia.edu. 3600    IN      NS      ns-1721.awsdns-23.co.uk.
```

☞ *Pass along IPv4 and IPv6 addresses for one of them*

```
;; ADDITIONAL SECTION:
ns-1721.awsdns-23.co.uk. 172800 IN      A       205.251.198.185
ns-1721.awsdns-23.co.uk. 172800 IN      AAAA    2600:9000:5306:b900::1
```

Queriers

- End-nodes—phones, laptops, etc.—typically run *stub resolvers*
- A stub resolver does nothing but send a query to a smarter node, typically run by the ISP or organization
- These smarter nodes—*caching resolvers*—do the detailed queries and cache results, per the TTLs
- Caching works better when many clients share the data—*many* people want to look up `www.google.com`, `www.facebook.com`, etc.
- The IP address of the local caching resolver is typically passed to hosts by the DHCP server
- Answers to DNS queries can thus be *authoritative*, if they're from the actual zone server, or *non-authoritative*, from the local caching resolver
- End-nodes can (usually) run their own caching resolver, but few do

DNS Insecurity

DNS Insecurity

- There are a number of security problems with the DNS—and they have nothing to do with our usual network threat models
- In addition, there are network security and privacy issues

Cache Contamination

- Suppose a caching resolver is evil
- You send a query to it, but get the wrong answer
- Suppose the query was encrypted—you'd get the wrong answer securely!
- The *information* was wrong

DNS Cache Contamination

- DNS responses can contain “additional information” as well as the answer
- (That’s intended for things like NS responses: don’t just give the nameserver’s DNS name, give its IP address, too
- Basic attack:
  - Induce a caching resolver to send a query to your authoritative name server
  - Include, along with the intended answer, extra—and malicious—records
  - When the victim asks for one of those records,they get the wrong answer
  - (Several known variations on this attack)
- Note well: the caching resolver is itself an innocent victim

Address-to-Name (Inverse) Issues

- Someone connects to your computer; you want to log that
- You know the IP address, so you convert it to a name
- But—the address-to-name tree isn't connected to the name-to-address tree!
- What if the answers don't match?

Forward and Reverse Queries

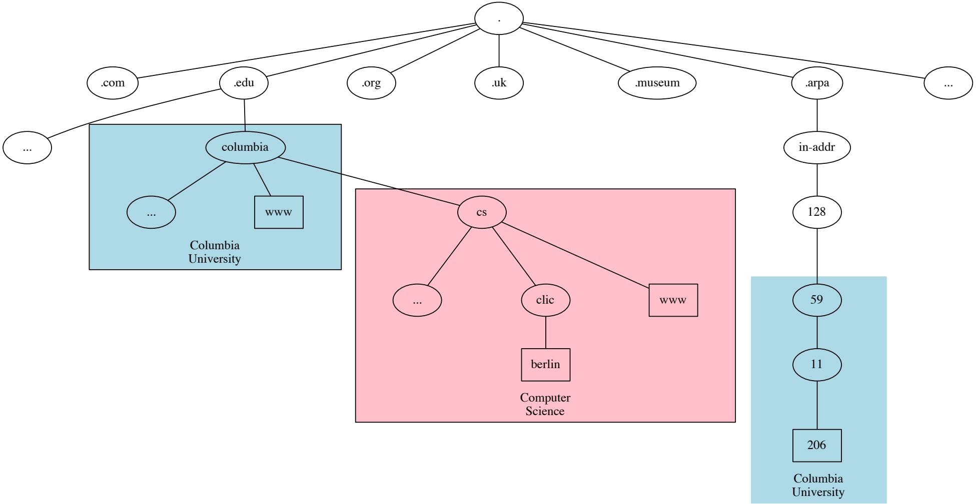
```
$ host www.google.com
www.google.com has address 172.217.6.196
$ host 172.217.6.196
196.6.217.172.in-addr.arpa domain name pointer lga25s54-in-f4.1e100.net.
196.6.217.172.in-addr.arpa domain name pointer lga25s54-in-f196.1e100.net.
```

We even see that close to home:

```
$ host www.cs.columbia.edu
www.cs.columbia.edu is an alias for webcluster.cs.columbia.edu.
webcluster.cs.columbia.edu has address 128.59.11.206
$ host 128.59.11.206
206.11.59.128.in-addr.arpa domain name pointer webcluster.cs.columbia.edu.
```

And if hostnames are used for access control?

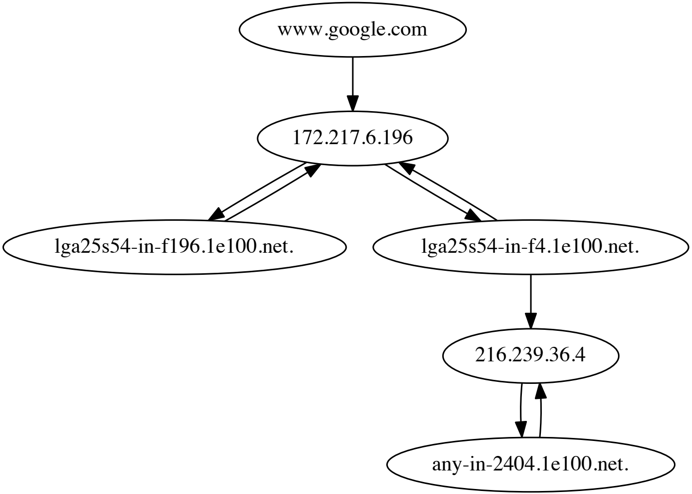
The in-addr.arpa Tree



It Can Be Very Confusing

```
$ host www.google.com
www.google.com has address 172.217.6.196
$ host 172.217.6.196
196.6.217.172.in-addr.arpa domain name pointer lga25s54-in-f196.1e100.net.
196.6.217.172.in-addr.arpa domain name pointer lga25s54-in-f4.1e100.net.
$ host lga25s54-in-f196.1e100.net.
lga25s54-in-f196.1e100.net has address 172.217.6.196
$ host lga25s54-in-f4.1e100.net.
lga25s54-in-f4.1e100.net has address 216.239.36.4
lga25s54-in-f4.1e100.net has address 172.217.6.196
$ host 216.239.36.4
4.36.239.216.in-addr.arpa domain name pointer any-in-2404.1e100.net.
$ host any-in-2404.1e100.net.
any-in-2404.1e100.net has address 216.239.36.4
```

www.google.com



Local Differences—and Local Evil

Sometimes, you get different answers depending on where you are—Google, for example, wants to direct you to the appropriate regional data center

Columbia
```
$ host www.google.com
www.google.com has address 172.217.6.196
```

Seattle, WA
```
$ host www.google.com
www.google.com has address 172.217.14.196
```

Ashburn, VA
```
$ host www.google.com
www.google.com has address 172.217.12.228
```

London, UK
```
$ host www.google.com
www.google.com has address 216.58.204.4
```

But this can also limit the visibility of an attack—you might not see the same results as someone in another locale

Privacy Issues

- Generally, you ask your ISP's caching resolver for answers
- But this means that your ISP knows every host you want to visit
- Encryption doesn't help—they're the endpoint!
- And if you run your own caching resolver, the root, `.com`, etc., know where you're going

Securing the DNS

Securing the DNS

- We’ve seen several different problems
- The solutions are different, too

Privacy Issues

- Suppose you don't trust your ISP
- Google and Cloudflare run public DNS caching resolvers
- You can even do DNS queries over TLS (the default for Firefox)
- But—do you trust Google or CloudFlare with your data? With ISP DNS, granularity of potential tracking is per-household; with DNS over TLS, it can be per-computer
- And how do you know that they haven't been deceived about correctness?

Inverse Mapping

- There can’t be a perfect solution—there have to be two different trees
- Best possible: do the inverse query, do the forward query with the result, and if they don’t match log both
- *Never* use hostnames for access control

Protecting the *Information*

- Encryption doesn't protect against malicious or confused resolvers
- The problem: the *information* is bad, not the transmission
- We have to protect the records, even if they're cached

Protecting the *Information*

- Encryption doesn't protect against malicious or confused resolvers
- The problem: the *information* is bad, not the transmission
- We have to protect the records, even if they're cached
- Solution: signed DNS records, via *DNSSEC*

DNSSEC

- The details are exceedingly complex—DNSSEC was retrofitted to a structure not designed for digital signatures
- Basic notion: all records for a name (the `RRset`) are signed with a private *zone-signing key*
- The parent zone signs the public zone-signing key
- That goes all the way up to the root
- Again, the details are *exceedingly* complex

What DNSSEC Gets Us

- Protects against (some) attacks on DNS records
- Allows for secure storage of other public keys in the DNS for, e.g., DKIM
- Possibly lets us replace the need for external CAs

Issues

- An attacker can strip the signature indications and just return ordinary, unsigned records
- ☞ After all, DNSSEC is optional
- Returning authoritative negative answers (“this host does not exist”) is hard and/or privacy-violating
- DNS registrars and registries for the top-level zones may not be secure enough to authoritatively sign records
- Responses are far too large for a single UDP packet—must use TCP, with all the overhead of the three-way handshake and the connection close sequence
- Many more. . .

DNS Security

- DNS is an essential part of the Internet's infrastructure
- But it dates to 1984, when the net was a very different place
- There are no credible alternatives, even though it's far more complex today than the original design
- The security issues are an important reason why we need end-to-end encryption—it's far easier to launch these DNS attacks than it is to tap a fiber

Questions?

(Male and female northern cardinals (probabably mates), Morningside Park, April 21, 2018)