Network security (DNS)

CS 161: Computer Security

Prof. Raluca Ada Popa

March 9, 2020

Slides adapted from David Wagner

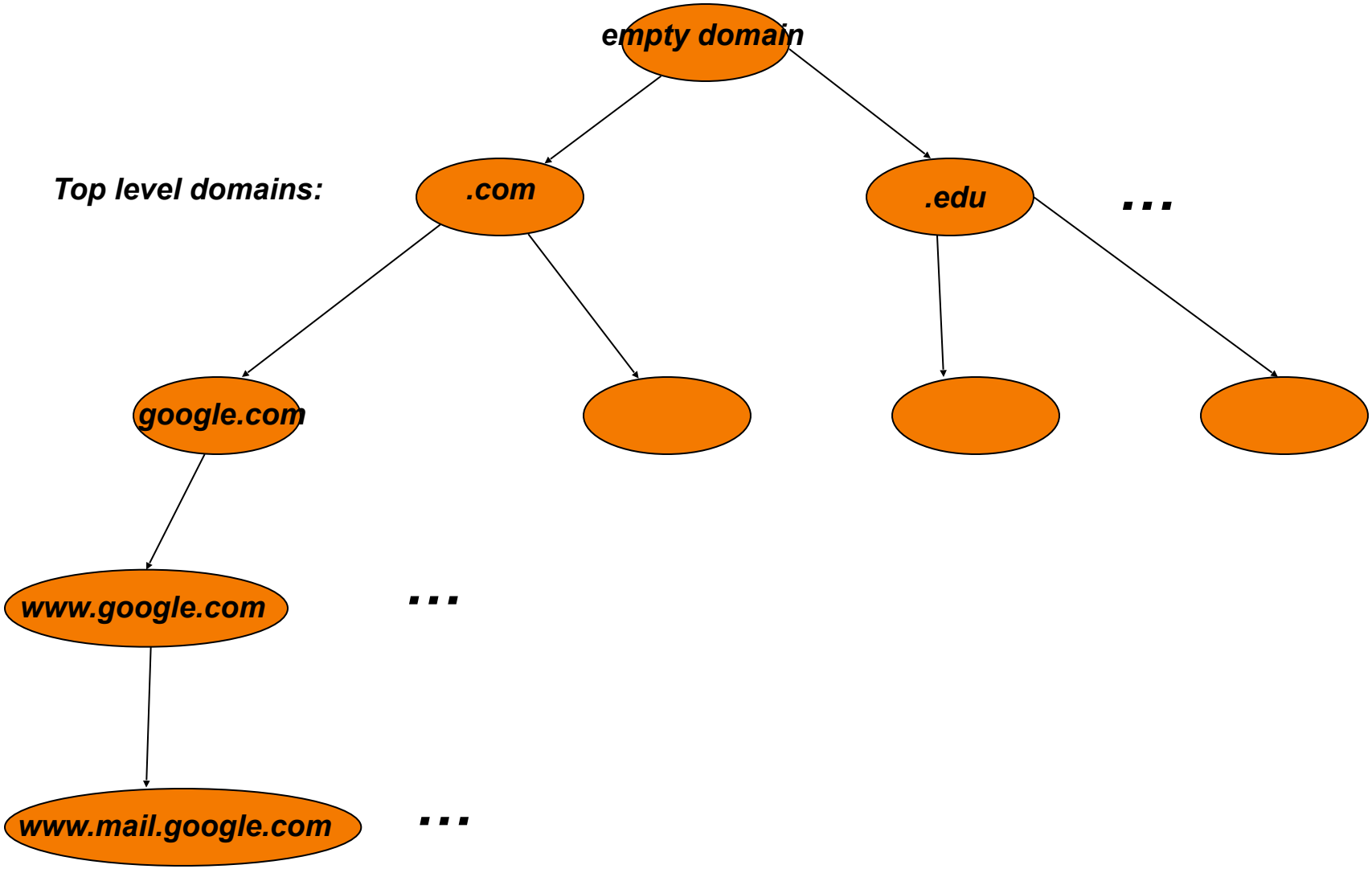
Announcements

- Discussion sections online

Domain names

- Domain names are human friendly names to identify servers or services
  - Arranged hierarchically
  - www.google.com has:
    - .com as TLD (top-level domain) is a subdomain of root
    - google.com as a subdomain of com
    - www.google.com a subdomain of google.com

Hierarchy of domain names



Types of domain names (TLD)

1. Generic TLDs: .com, .edu
2. Country-code TLDs: .au .de .it .us

Creating a domain name

- Domain names are registered and assigned by **domain-name registrars**, accredited by the Internet Corporation for Assigned Names and Numbers (ICANN), same group allocating the IP address space
- Contact the domain-name registrar to register domain space

Cybersquatting or Domain Squatting

- Entities buying a domain in advance of it becoming desirable and later selling to the agency needing it for much more

2013: Microsoft vs. MikeRoweSoft

The boy accepted an Xbox in exchange for the domain name

DNS Overview

- DNS translates www.google.com to 74.125.25.99: **resolves** www.google.com

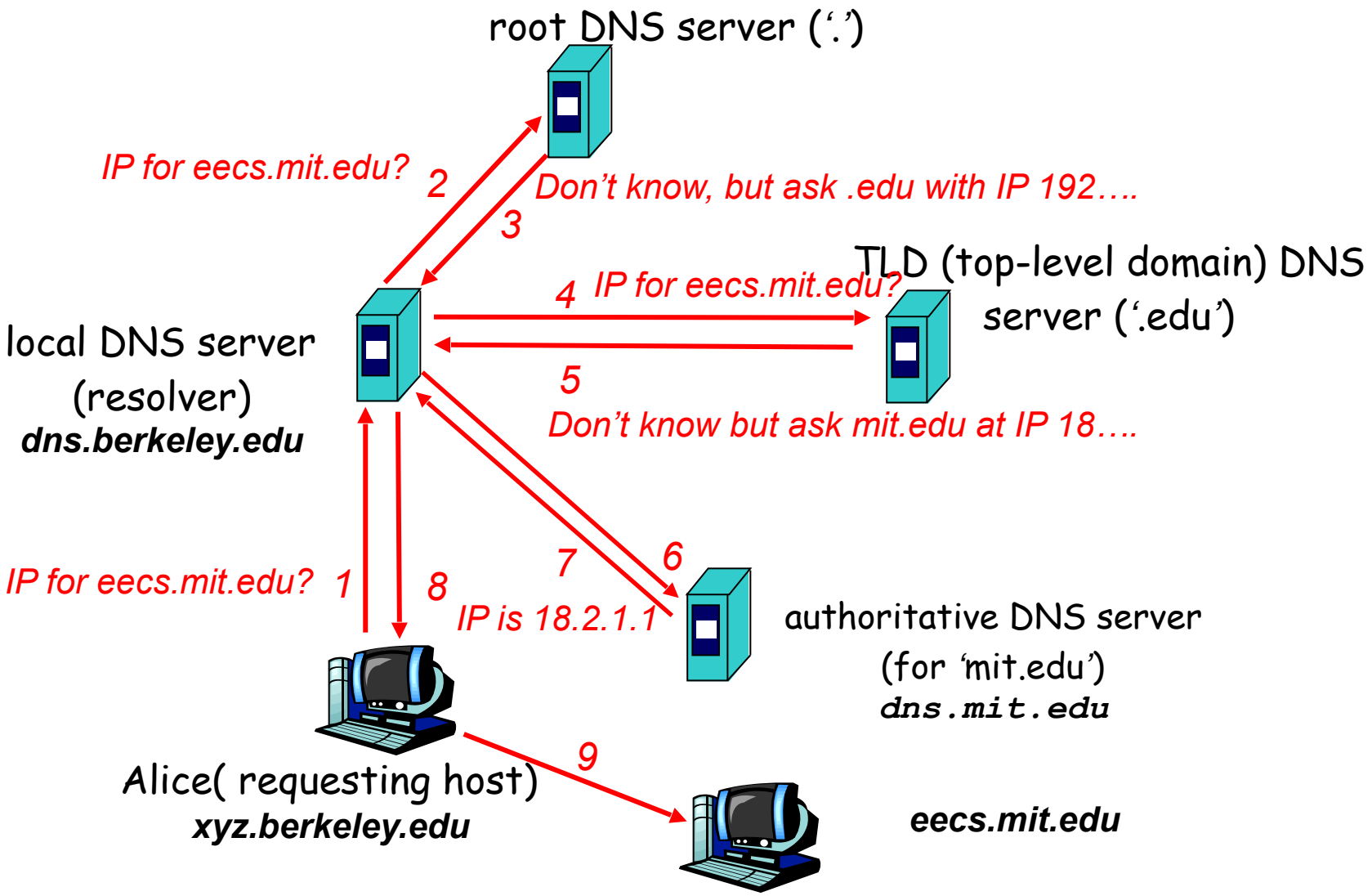
Name servers

- To resolve a domain name, a resolver queries a distributed hierarchy of **DNS servers** also called **name servers**
- At the top level are the root name servers, which resolve TLDs such as .com
  - Store the **authoritative name server** for each TLD (the trusted server for the TLD)
  - Government and commercial organizations run the name servers for TLDs
  - Name server for .com managed by Verisign

A DNS Lookup

1. Alice goes to *eecs.mit.edu* on her browser
2. Her machine contacts a resolver to ask for *eecs.mit.edu*'s IP address
   – The resolver can be a name server for the corporate network of Alice's machine or of her Internet service provider that her machine learned from DHCP
3. The resolver will try to resolve this domain name and return an IP address to Alice's machine

DNS Lookups via a *Resolver*



DNS caching

- Almost all DNS servers (resolver and name servers) cache entries, which improves performance significantly

`dig`

- A program on Unix that allows querying the DNS system
- Dumps each field in DNS responses

dig eecs.mit.edu A

Use Unix "`dig`" utility to look up IP address ("`A`") for hostname `eecs.mit.edu` via DNS

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           2160

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

A 16-bit **transaction identifier** that enables the DNS client (`dig`, in this case) to match up the reply with its original request

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

The question we asked the server

dig eecs.mit.edu A

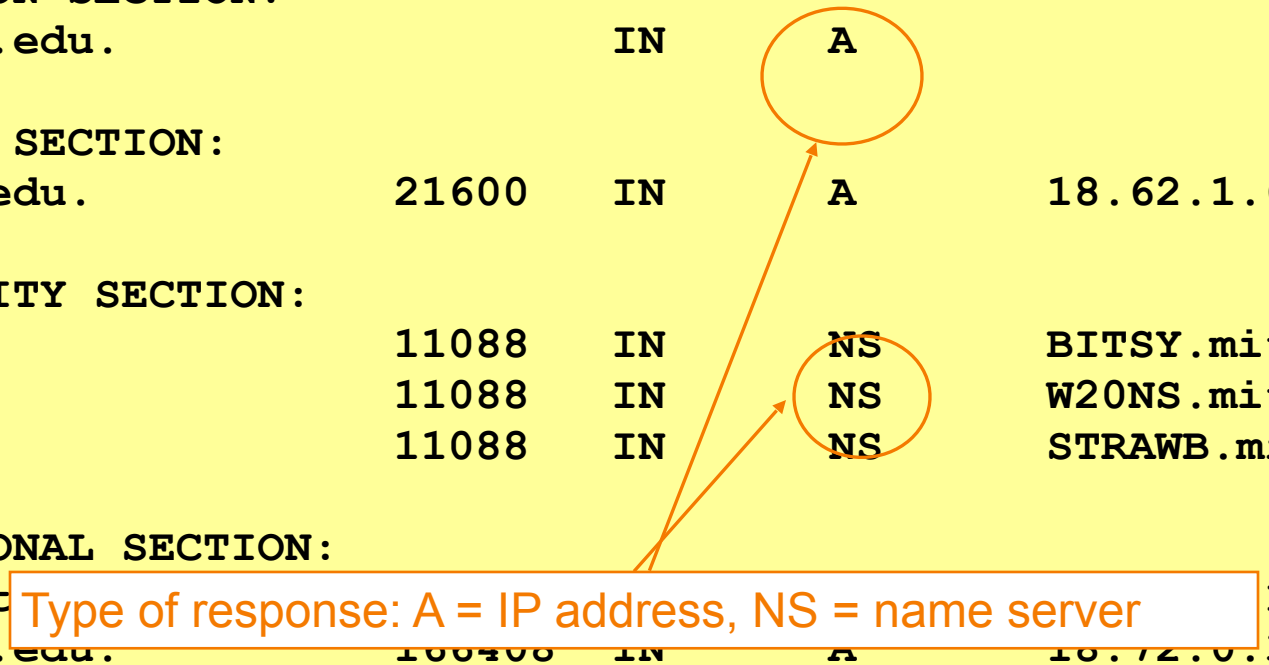
```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit                                                      151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```



dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode
;; flags: qr rd ra; QU                                    IONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

"**Answer**" tells us the IP address associated with `eecs.mit.edu` is `18.62.1.6` and we can *cache* the result for 21,600 seconds

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APP
;; global options: +cm
;; Got answer:
;; ->>HEADER<<- opcode
;; flags: qr rd ra; QU                                          3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

"**Authority**" tells us the *name servers* responsible for the answer. Each RR (resource record) gives the *hostname* of a different name server ("`NS`") for names in `mit.edu.` We should cache each record for 11,088 seconds.

If the "**Answer**" had been empty, then the resolver's next step would be to send the original query to one of these name servers.

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION
eecs.mit.edu.

;; AUTHORITY SECT
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

"**Additional**" provides extra information to save us from making separate lookups for it, or helps with bootstrapping.

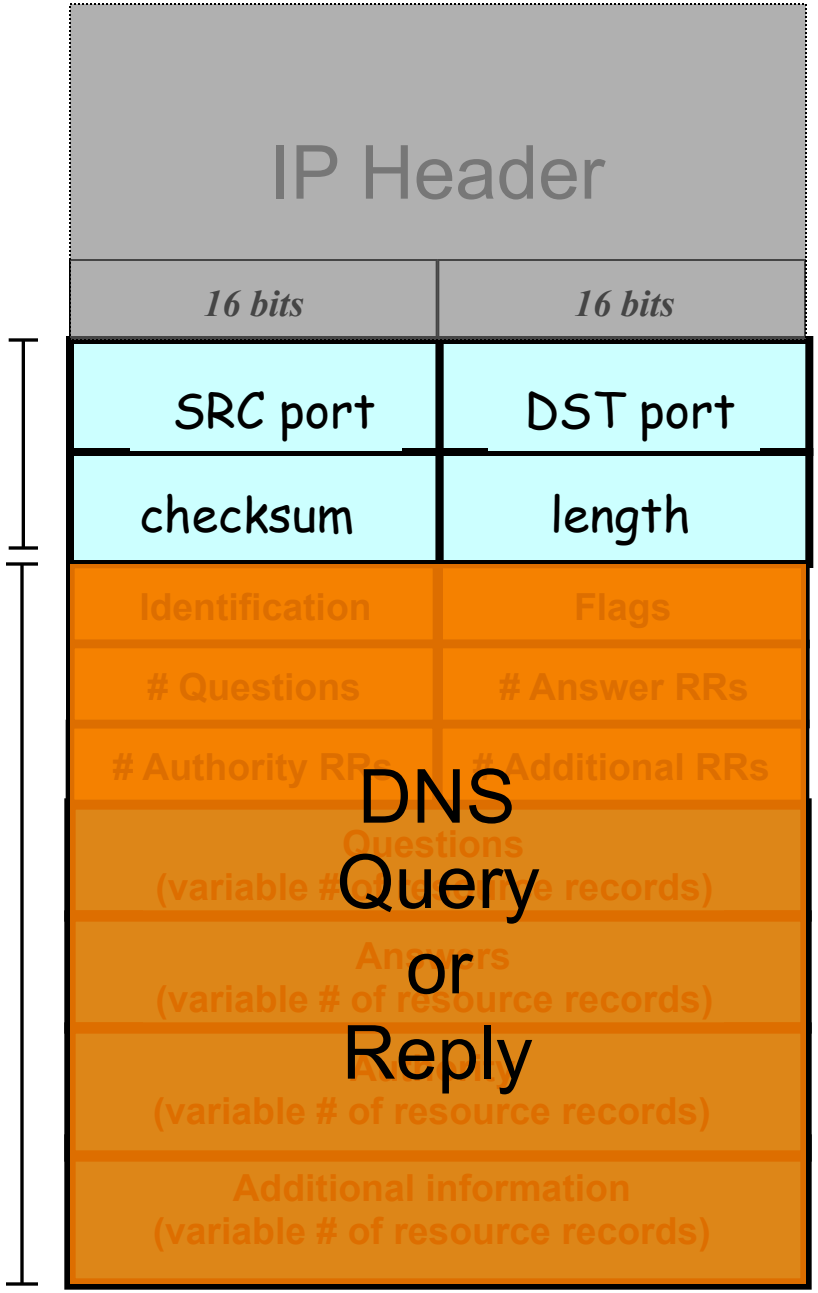
Here, it tells us the IP addresses for the hostnames of the name servers. We add these to our cache.

DNS Protocol

Lightweight exchange of *query* and *reply* messages, both with same message format

Primarily uses UDP for its transport protocol, which is what we'll assume

Frequently, both clients and servers use port 53

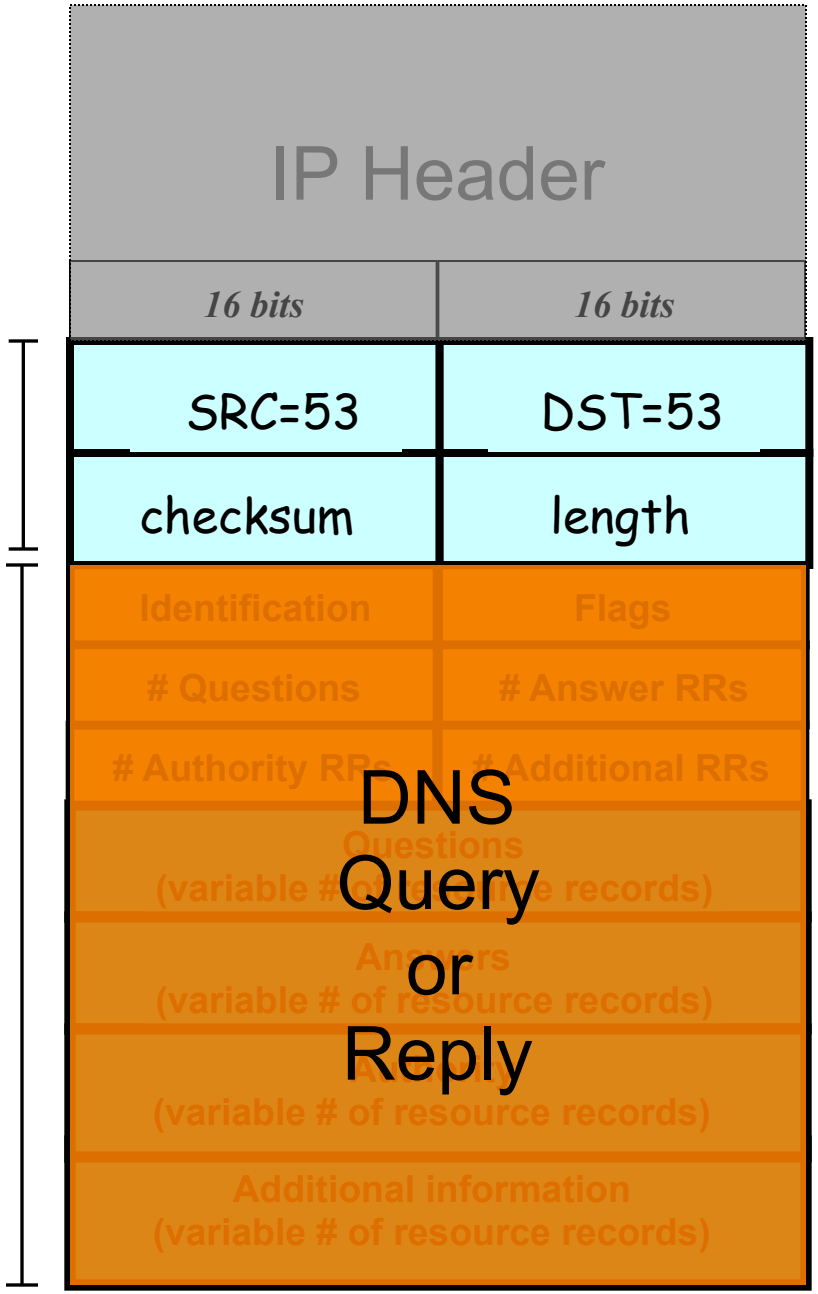


DNS Protocol

Lightweight exchange of *query* and *reply* messages, both with same message format

Primarily uses UDP for its transport protocol, which is what we'll assume

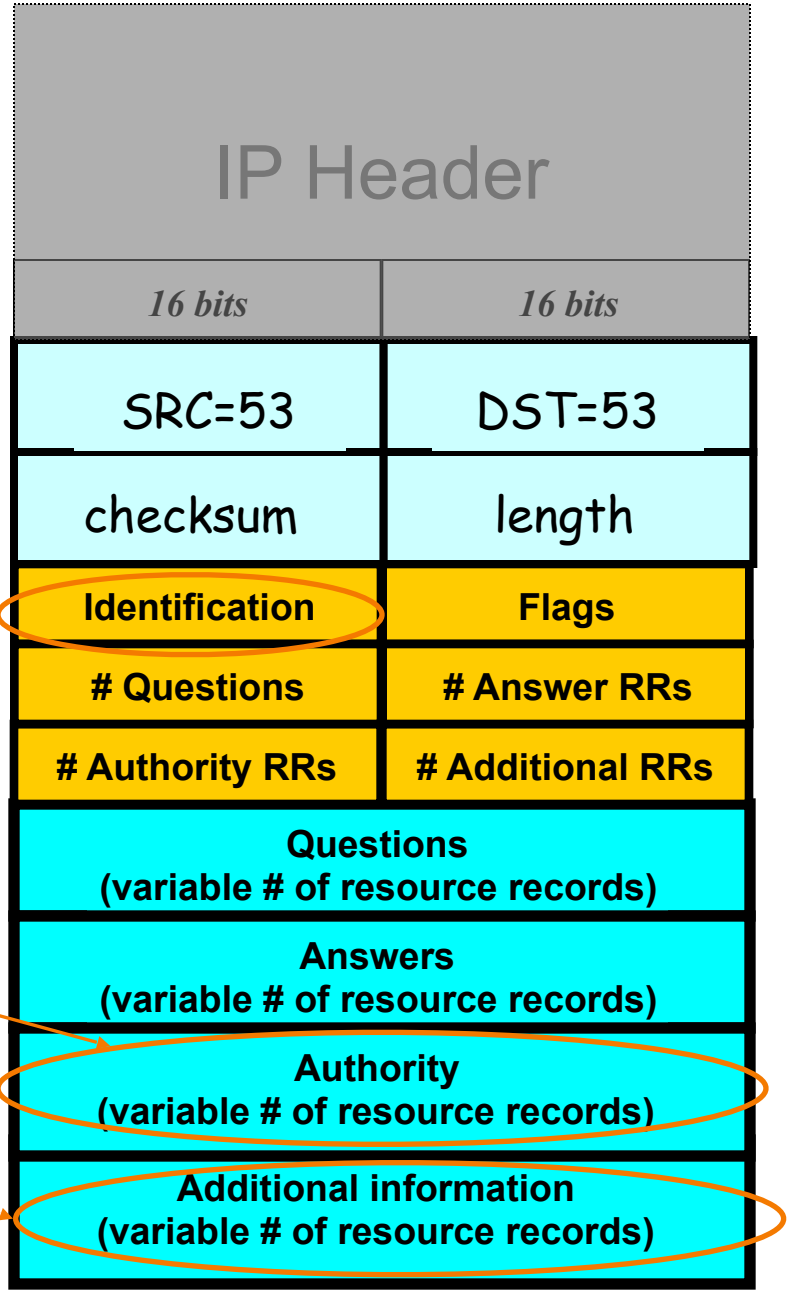
Frequently, both clients and servers use port 53



DNS Protocol, cont.

Message header:

- Identification: 16 bit # for query, reply to query uses same #
- Along with repeating the Question and providing Answer(s), replies can include "**Authority**" (name server responsible for answer) and "**Additional**" (info client is likely to look up soon anyway)
- Each *Resource Record* has a Time To Live (in seconds) for **caching** *(not shown)*



Security risk #1: malicious DNS server

- Of course, if *any* of the DNS servers queried are malicious, they can lie to us and fool us about the answer to our DNS query.
- Any consequence?
  - We talk to the incorrect server

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NO
;; flags: qr rd ra; QUERY: 1, ANSWER: 1,

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

What if the mit.edu server is untrustworthy? Could its operator steal, say, all of our web surfing to berkeley.edu's main web server?

dig eecs.mit.edu A

Let's look at a flaw in the original DNS design (since fixed)

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mi
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                30000   IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       30000   IN      A       18.6.6.6
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

What could happen if the mit.edu server returns the following to us instead?

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                30000   IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       30000   IN      A       18.6.6.6
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

We'd dutifully store in our cache a mapping of `www.berkeley.edu` to an IP address under MIT's control. (It could have been any IP address they wanted, not just one of theirs.)

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.
mit.edu.
mit.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       30000   IN      A       18.6.6.6
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

Later if we need to resolve www.berkeley.edu, we will go to the MIT IP address

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.
```

How do we fix such DNS ***cache poisoning***?

```
;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                30000   IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       30000   IN      A       18.6.6.6
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +c
;; Got answer:
;; ->>HEADER<<- opcod
;; flags: qr rd ra; Q

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                30000   IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       30000   IN      A       18.6.6.6
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

Don't accept **Additional** records unless they're for the domain we're looking up

- E.g., looking up `eecs.mit.edu` ⇒ only accept additional records from `*.mit.edu`

No extra risk in accepting these since server could return them to us directly in an **Answer** anyway.

Security risk #1: malicious DNS server

- Of course, if *any* of the DNS servers queried are malicious, they can lie to us and fool us about the answer to our DNS query…
- and they used to be able to fool us about the answer to other queries, too, using *cache poisoning*. Now fixed (phew).

Security risk #2: on-path eavesdropper

- If attacker can eavesdrop on our traffic… we're hosed.
- Why?

Security risk #2: on-path eavesdropper

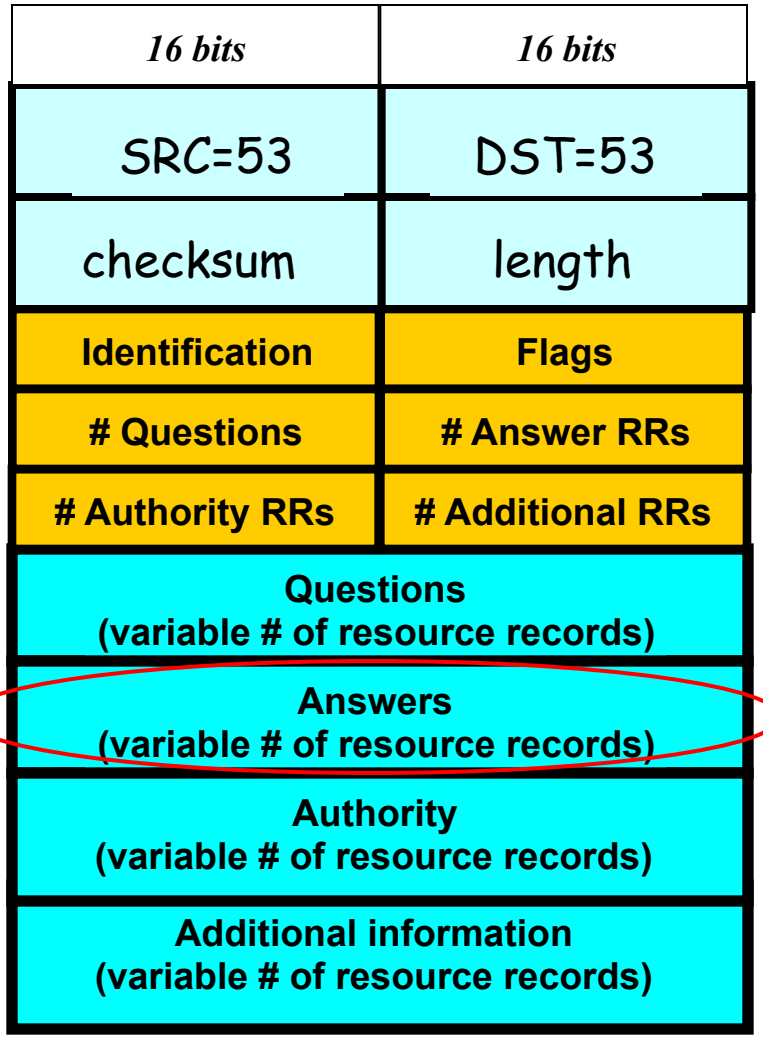
- If attacker can eavesdrop on our traffic…
  we're hosed.
- Why? They can see the query and the 16-bit transaction identifier, and race to send a spoofed response to our query.

Security risk #3: off-path attacker

- If attacker can't eavesdrop on our traffic, can he inject spoofed DNS responses?
- Answer: It used to be possible, via *blind spoofing*. We've since deployed mitigations that makes this harder (but not totally impossible).

Blind spoofing

- Say we look up `mail.google.com`; how can an **off-path** attacker feed us a bogus `A` answer before the legitimate server replies?
- How can such a **remote** attacker even know we are looking up `mail.google.com`?

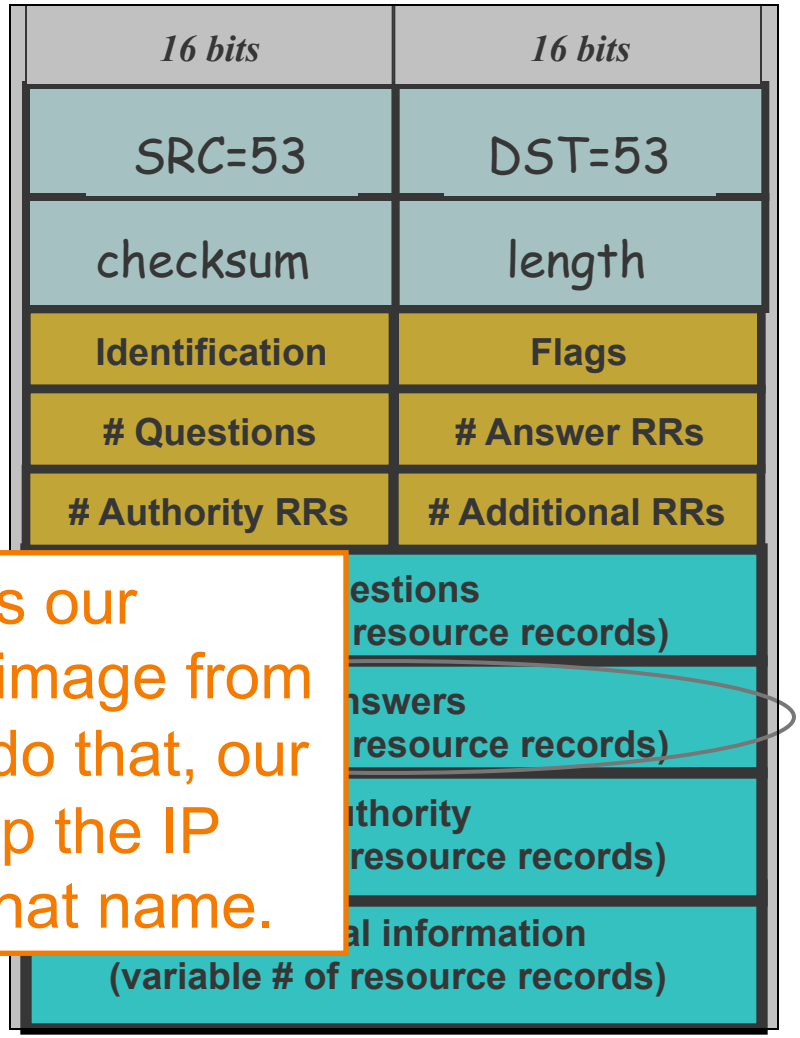


Suppose, e.g., we visit a web page under their control:

```
...<img src="http://mail.google.com" …
```

Blind spoofing

- Say we look up `mail.google.com`; how can an **off-path** attacker feed us a bogus `A` answer before the legitin
- How
even
`mail`
Suppose, e.g., we visit a web page under their control:



This HTML snippet causes our browser to try to fetch an image from `mail.google.com`. To do that, our browser first has to look up the IP address associated with that name.

```
...<img src="http://mail.google.com" …
>
```

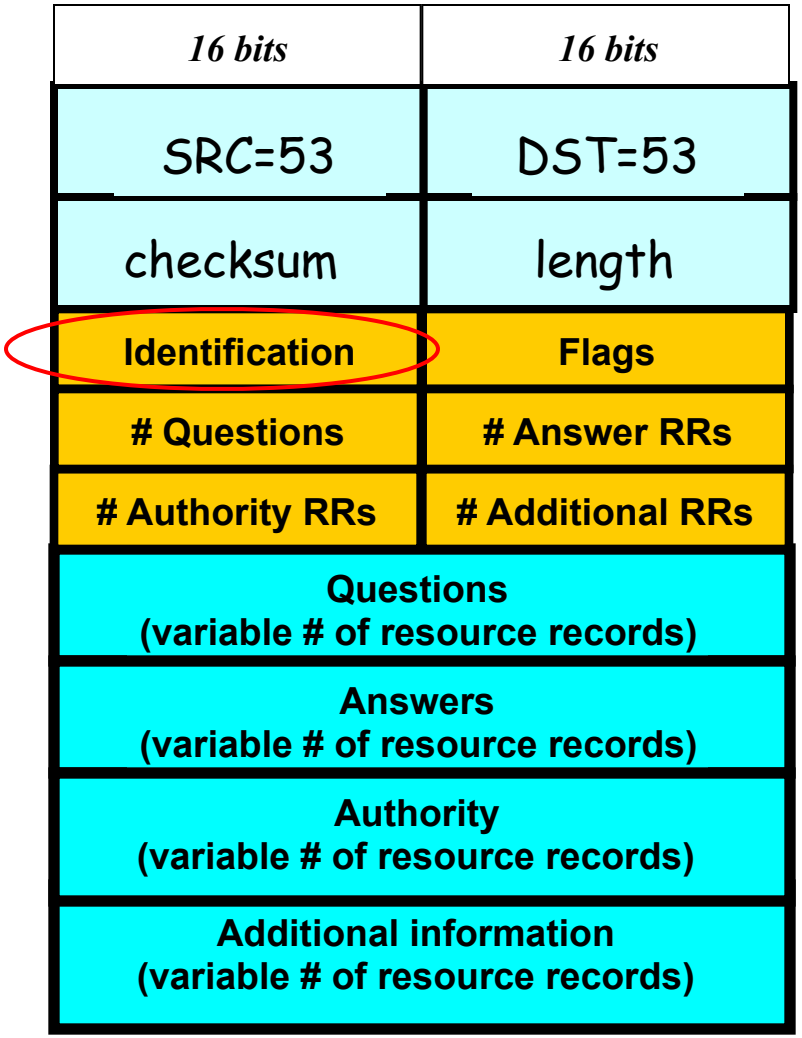
Blind spoofing

Once they know we're looking it up, they just have to guess the Identification field and reply before legit server.

How hard is that?

Originally, identification field incremented by 1 for each request. How does attacker guess it?

Fix?




```
<img src="http://badguy.com" ...>
<img src="http://mail.google.com" ...>
```

They observe ID k here

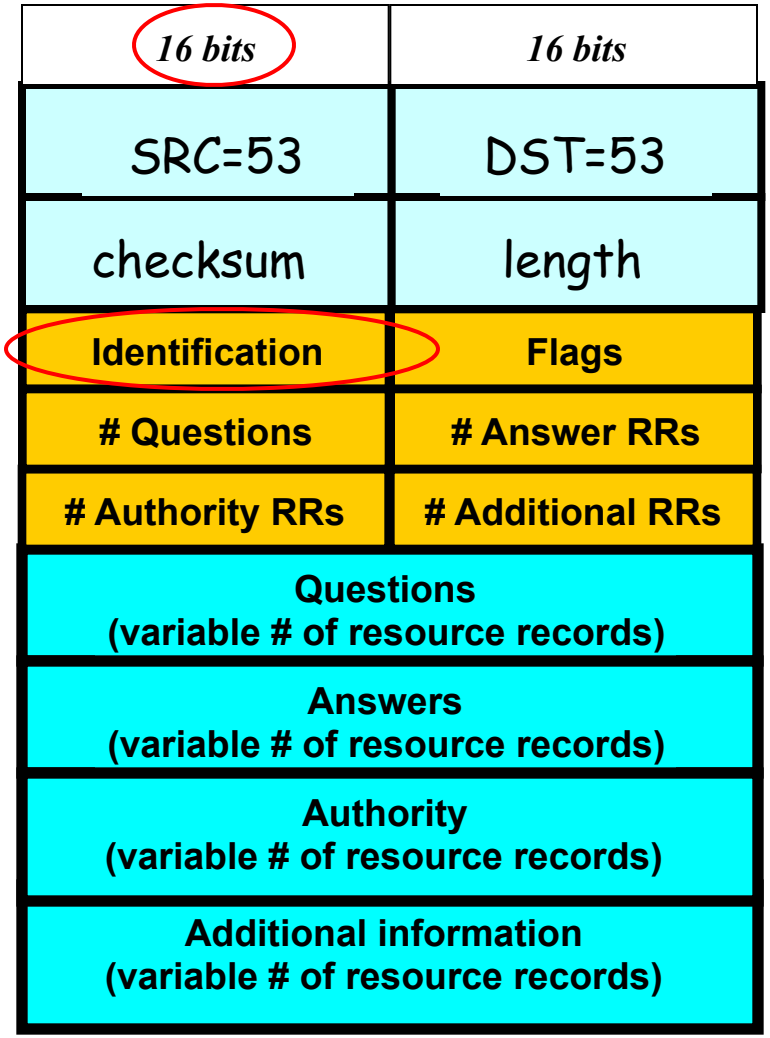
So this will be k+1

DNS Blind Spoofing, cont.

Once we randomize the Identification, attacker has a 1/65536 chance of guessing it correctly.
Are we safe?

Attacker can send *lots* of replies, not just one …

However: once reply from legit server arrives (with correct Identification), it's **cached** and no more opportunity to poison it. Victim is innoculated!



Unless attacker can send 1000s of replies before legit arrives…

Summary of DNS Security Issues

- DNS threats highlight:
  - Attackers can attack opportunistically rather than eavesdropping
    - Cache poisoning only required victim to look up some name under attacker's control (*has been **fixed***)
  - Attackers can often manipulate victims into vulnerable activity
    - E.g., `IMG SRC` in web page to force DNS lookups
  - Crucial for identifiers associated with communication to have sufficient entropy (= a lot of bits of unpredictability)
  - "**Attacks only get better**": threats that appears technically remote can become practical due to unforeseen cleverness

Common Security Assumptions

- (Note, these tend to be pessimistic … but prudent)
- Attackers can interact with our systems without particular notice
  - *Probing* (poking at systems) may go unnoticed …
  - … even if highly repetitive, leading to crashes, and *easy to detect*
- It's easy for attackers to know general information about their targets
  - OS types, software versions, usernames, server ports, IP addresses, usual patterns of activity, administrative procedures

Common Assumptions

- Attackers can obtain access to a copy of a given system to measure and/or determine how it works
- Attackers can make energetic use of automation
  - They can often find clever ways to automate
- Attackers can pull off complicated coordination across a bunch of different elements/systems
- Attackers can bring large resources to bear if needed
  - Computation, network capacity
  - But they are *not* super-powerful (e.g., control entire ISPs)

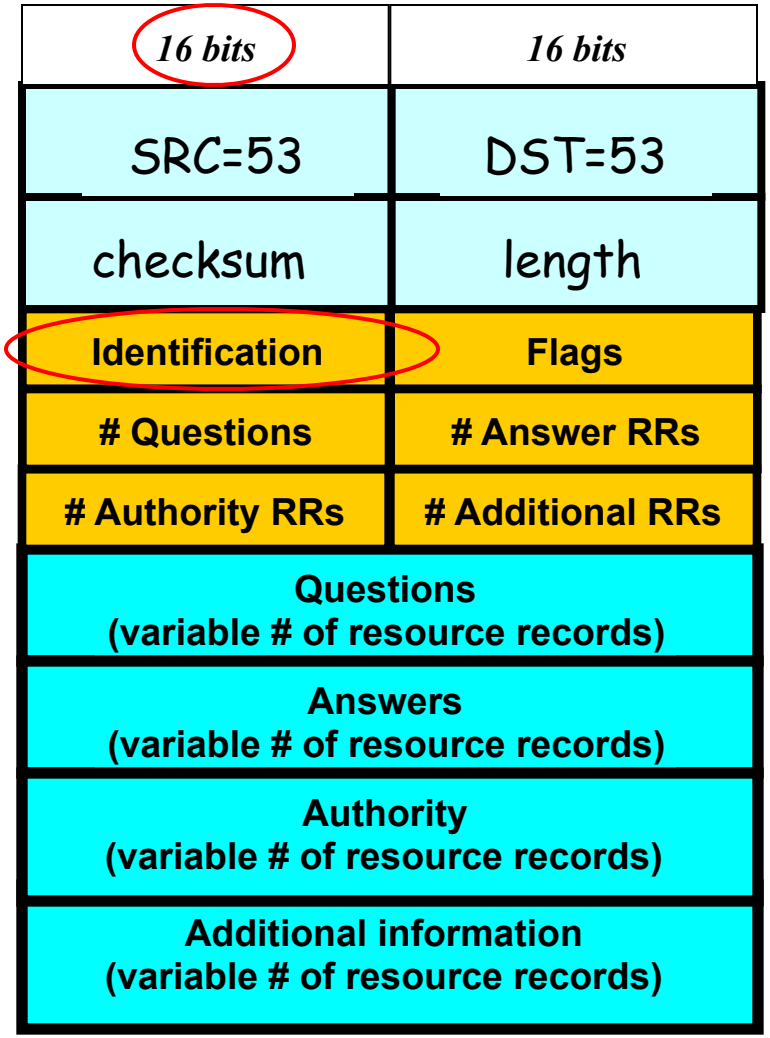
The Kaminsky Blind Spoofing Attack

DNS Blind Spoofing, cont.

Once we randomize the Identification, attacker has a 1/65536 chance of guessing it correctly.
Are we safe?

Attacker can send *lots* of replies, not just one …

However: once reply from legit server arrives (with correct Identification), it's **cached** and no more opportunity to poison it. Victim is innoculated!



Unless attacker can send 1000s of replies before legit arrives…

DNS Blind Spoofing (Kaminsky 2008)

- Two key ideas:
  - Attacker can get around caching of legit replies by generating a series of different name lookups:

```
<img src="http://random1.google.com"
…>
<img src="http://random2.google.com"
…>
<img src="http://random3.google.com"
…>
                    ...
<img src="http://randomN.google.com"
…>
```

  - Trick victim into looking up a domain you don't care about, use **Additional** field to spoof the domain you do care about

Kaminsky Blind Spoofing

For each lookup of *randomk*.google.com, attacker **spoofs** a bunch of records like this, each with a different Identifier

```
;; QUESTION SECTION:
;random7.google.com.              IN      A

;; ANSWER SECTION:
random7.google.com       21600   IN      A       doesn't matter

;; AUTHORITY SECTION:
google.com.              11088   IN      NS      mail.google.com

;; ADDITIONAL SECTION:
mail.google.com          126738  IN      A       6.6.6.6
```

Once they win the race, not only have they poisoned mail.google.com …

Kaminsky Blind Spoofing

For each lookup of *randomk*.google.com, attacker **spoofs** a bunch of records like this, each with a different Identifier

```
;; QUESTION SECTION:
;random7.google.com.              IN      A

;; ANSWER SECTION:
random7.google.com       21600   IN      A       doesn't matter

;; AUTHORITY SECTION:
google.com.              11088   IN      NS      mail.google.com

;; ADDITIONAL SECTION:
mail.google.com          126738  IN      A       6.6.6.6
```

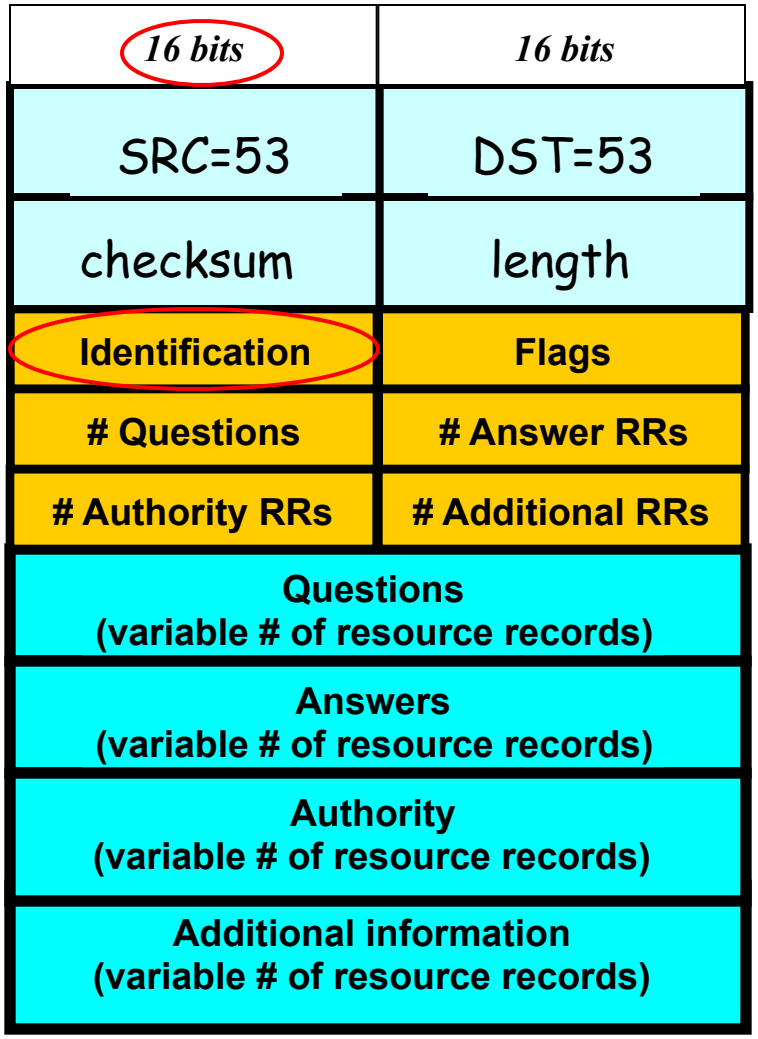
Once they win the race, not only have they poisoned mail.google.com … but also the cached NS record for google.com's name server – so any **future** *X*.google.com lookups *go through the attacker's machine*

Defending Against Blind Spoofing

Central problem: all that tells a client they should accept a response is that it matches the Identification field.

With only 16 bits, it lacks sufficient entropy: even if truly random, the *search space* an attacker must *brute force* is too small.

Where can we get more entropy? (*Without* requiring a protocol change.)

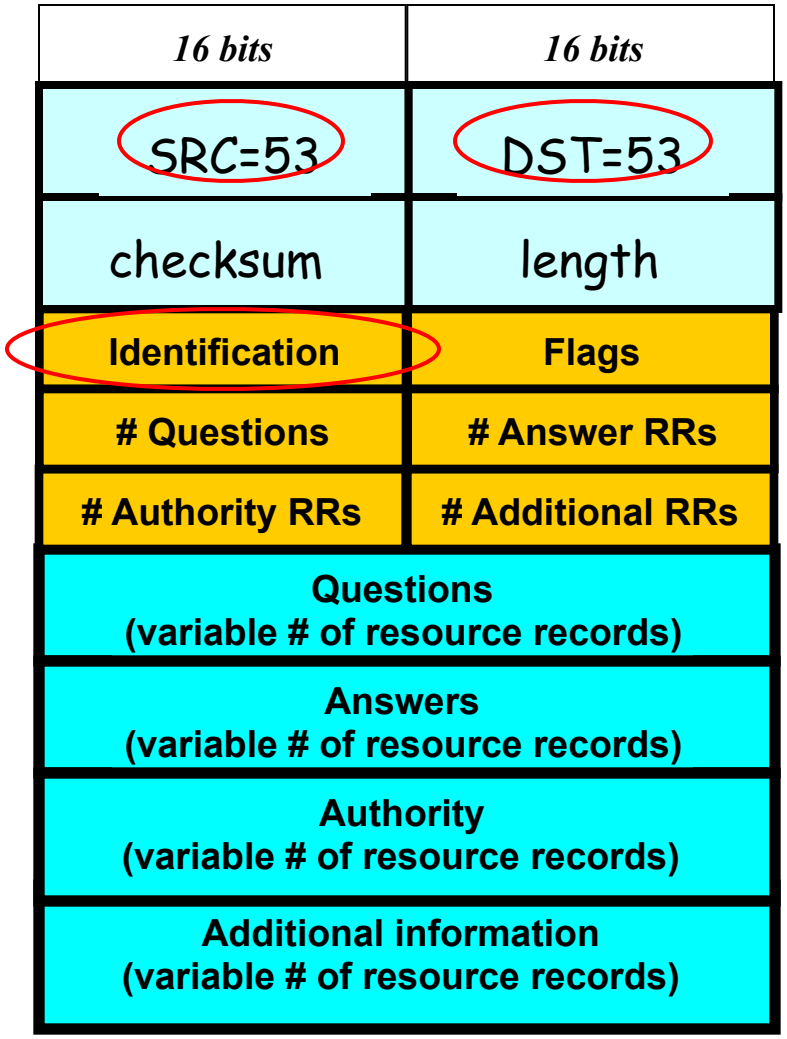


Defending Against Blind Spoofing

For requestor to receive DNS reply, needs both correct Identification and correct ports.

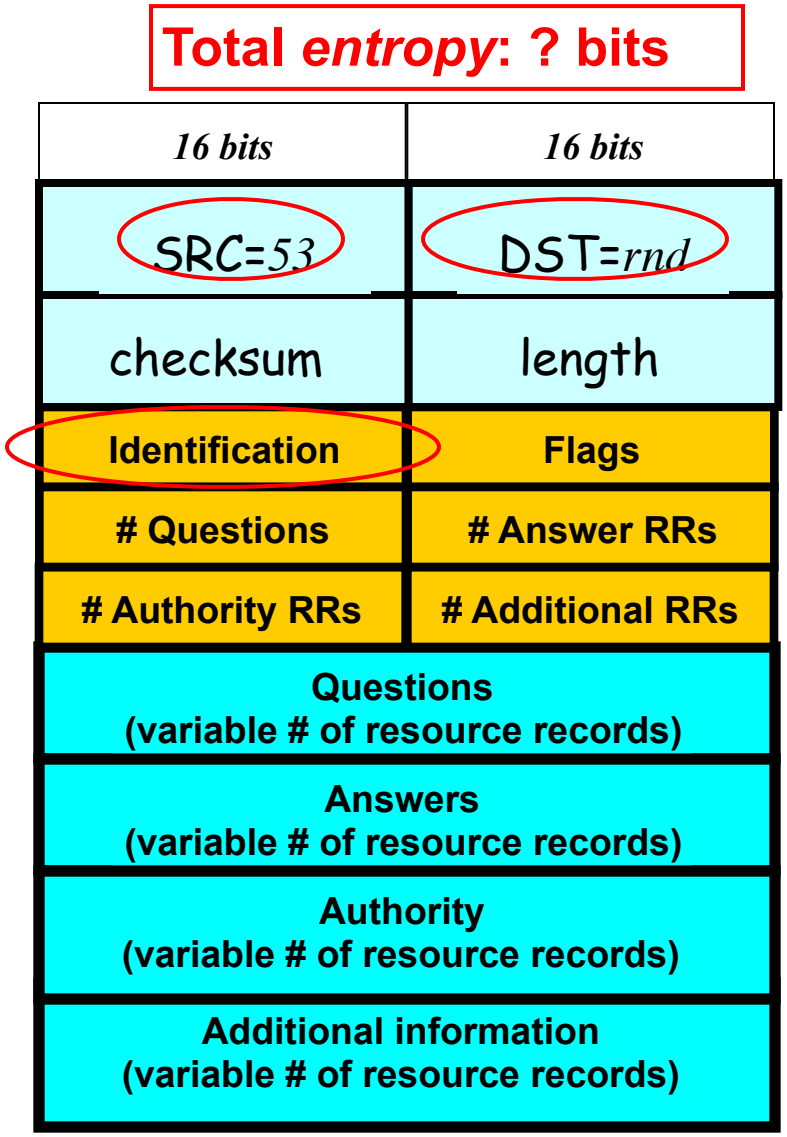
On a request, DST port = 53. SRC port usually also 53 – but not fundamental, just convenient.

Total *entropy*: 16 bits



Defending Against Blind Spoofing

"Fix": client uses random source port ⇒ attacker doesn't know correct dest. port to use in reply



Defending Against Blind Spoofing

"Fix": client uses random source port ⇒ attacker doesn't know correct dest. port to use in reply

32 bits of entropy makes it orders of magnitude harder for attacker to guess all the necessary fields and dupe victim into accepting spoof response.

This is what primarily "secures" DNS against blind spoofing today.