Network Attacks, con't

CS 161: Computer Security

Prof. Vern Paxson

TAs: Jethro Beekman, Mobin Javed,
Antonio Lupher, Paul Pearce
& Matthias Vallentin

http://inst.eecs.berkeley.edu/~cs161/

February 12, 2013

Game Plan

- Reminder: Homework #1 due Friday night, 10:00PM
- Goals for today:
  - Clarify assumptions made when analyzing security threats
  - More network attacks
    - DNS: protocol for mapping hostnames to IP addresses
    - DHCP: protocol for bootstrapping Internet access

      (time permitting)

Common Assumptions When Discussing Attacks

- (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, con't

- 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 req'd
  - Computation, network capacity
  - But they are *not* super-powerful (e.g., control entire ISPs)

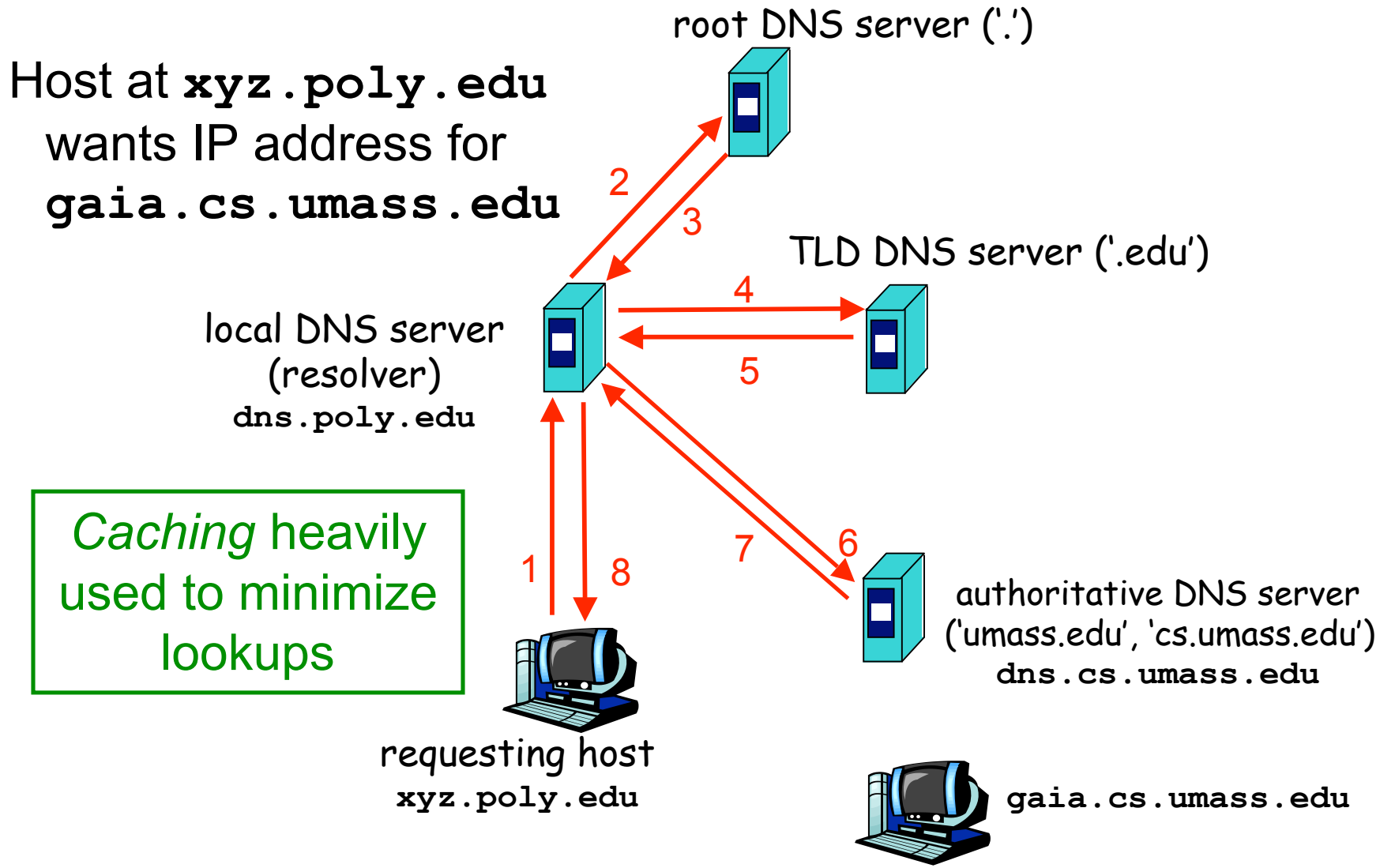
Common Assumptions, con't

- If it helps the attacker in some way, assume they can obtain privileges
  - But if the privilege gives everything away (attack becomes trivial), then we care about unprivileged attacks
- The ability to robustly *detect* that an attack has occurred does not replace desirability of preventing
- Infrastructure machines/systems are well protected (hard to directly take over)
  - So a vulnerability that requires infrastructure compromise is less worrisome than same vulnerability that doesn't

Common Assumptions, con't

- Network routing is hard to alter … other than with physical access near clients (e.g., "coffeeshop")
  - Such access helps fool clients to send to wrong place
  - Can enable *Man-in-the-Middle* (MITM) attacks
- We worry about attackers who are lucky
  - Since often automation/repetition can help "make luck"
- Just because a system does not have apparent value, it may still be a target
- Attackers are undaunted by fear of getting caught

DNS Lookups via a *Resolver*



DNS Threats

- DNS: path-critical for just about everything we do
  - Maps hostnames ⇔ IP addresses
  - Design only **scales** if we can minimize lookup traffic
    - #1 way to do so: caching
    - #2 way to do so: return not only answers to queries, but additional info that will likely be needed shortly
- What if attacker eavesdrops on our DNS queries?
  - Then similar to DHCP, can redirect us w/ misinformation
- Consider attackers who *can't* eavesdrop - but still aim to manipulate us via *how the protocol functions*
- Directly interacting w/ DNS: **`dig`** program on Unix
  - Allows querying of 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.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088
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
```

This is `dig` identifying its version and the query it is attempting to look up

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                           .
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
```

Status values returned from the remote name server queried by `dig`

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
```

Including 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.                                        W20NS.mit.edu.
mit.edu.                                        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 name server echoes back the question that it is answering as the first part of its reply

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-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.                                                edu.
mit.edu.                                                edu.
mit.edu.                                                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
```

In general, a single *Resource Record* (RR) like this includes, left-to-right, a DNS name, a *time-to-live*, a family (`IN` for our purposes - ignore), a type (`A` here), and an associated value

dig eecs.mit.edu A

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

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

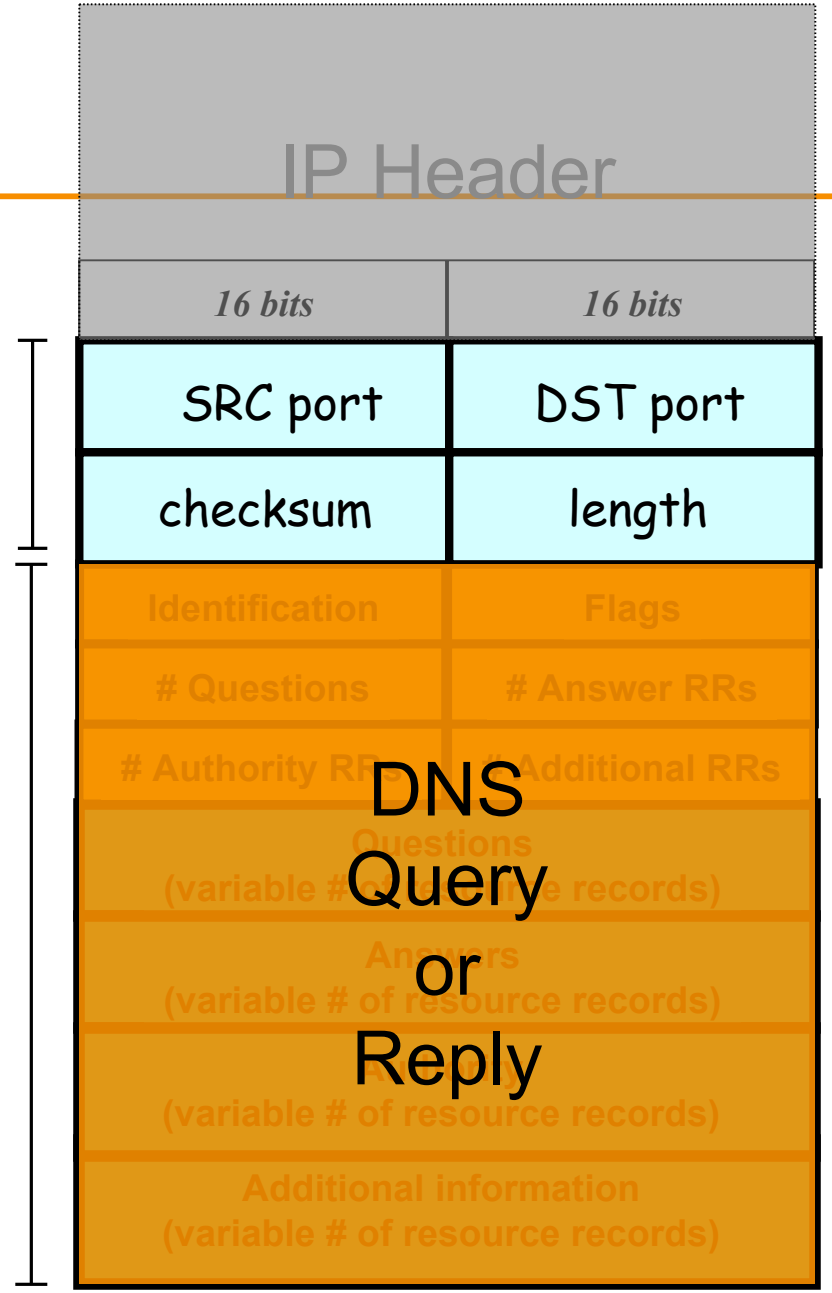
Here, it tells us the IP addresses for the hostnames of the name servers. We add these to our cache.

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

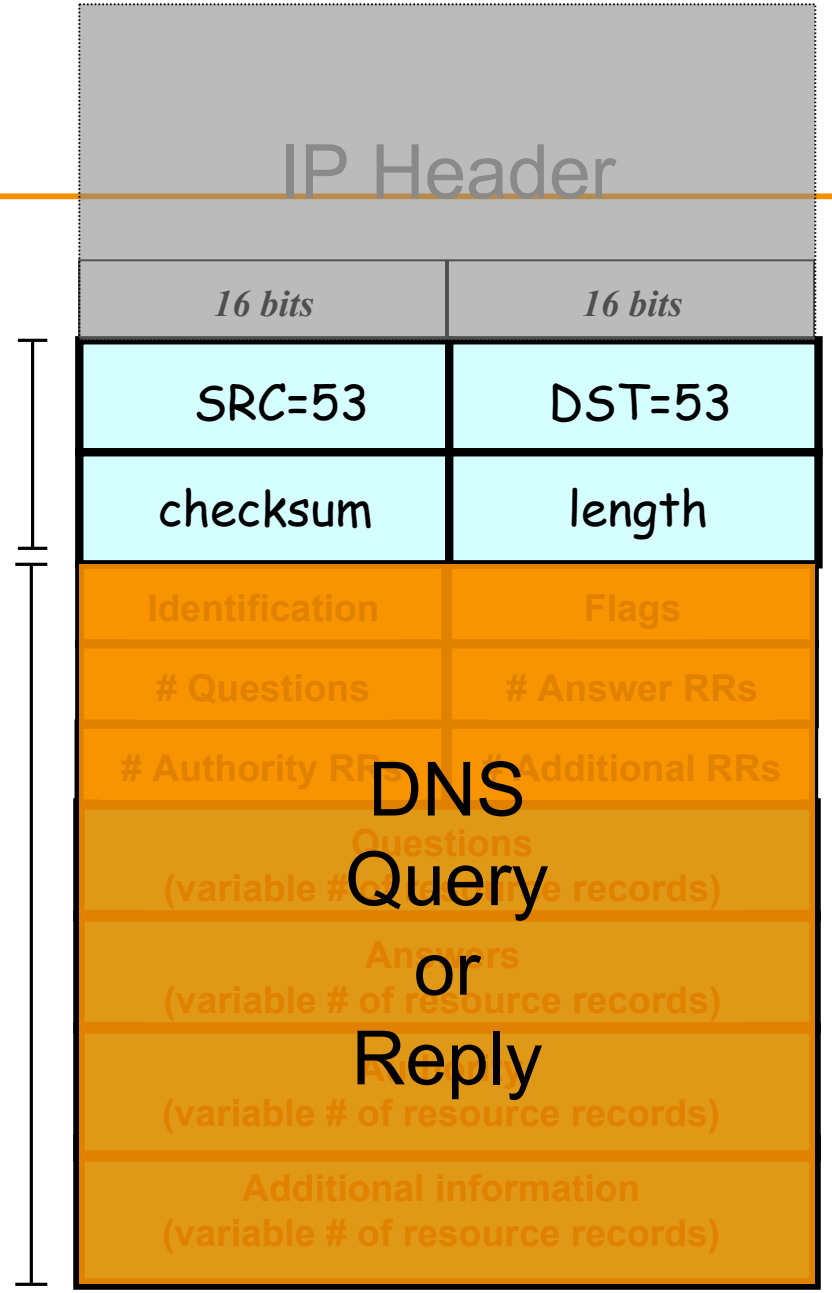


DNS Protocol

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

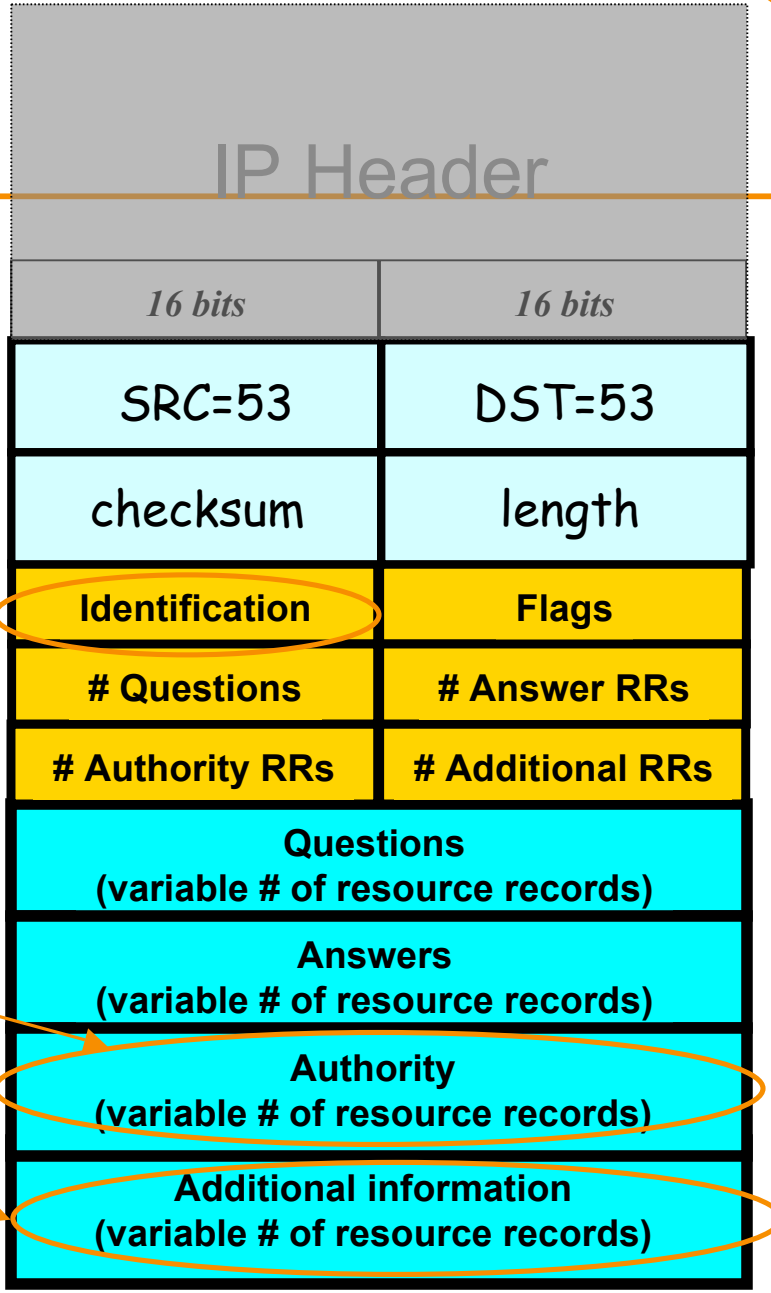
Frequently, both clients and servers use port 53



DNS Protocol, con't

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)*



dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QU                                    901
;; flags: qr rd ra; QUERY:                                    ADDITIONAL: 3

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

;; ANSWER SECTION:
eecs.mit.edu.           21                                    .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 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.

;; 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
```

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

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.                30      IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       30      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.

;; 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.                30      IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       30      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.                                                    6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                30      IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       30      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
```

In this case they chose to make the mapping *disappear* after 30 seconds. They could have made it persist for weeks, or disappear even quicker.

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 ***cache poisoning***?

```
;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                30      IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       30      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.           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.                30      IN      NS      www.berkeley.edu.

;; ADDITIONAL SECTION:
www.berkeley.edu.       30      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.

5 Minute Break

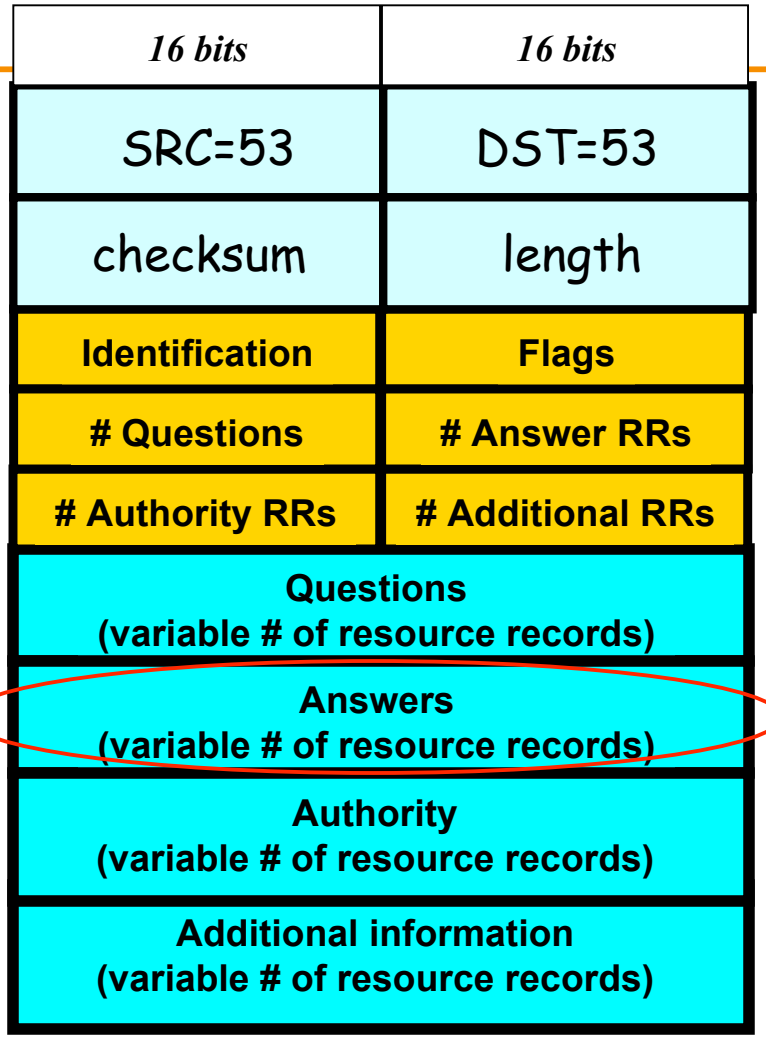
Questions Before We Proceed?

DNS Threats, con't

What about *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`?



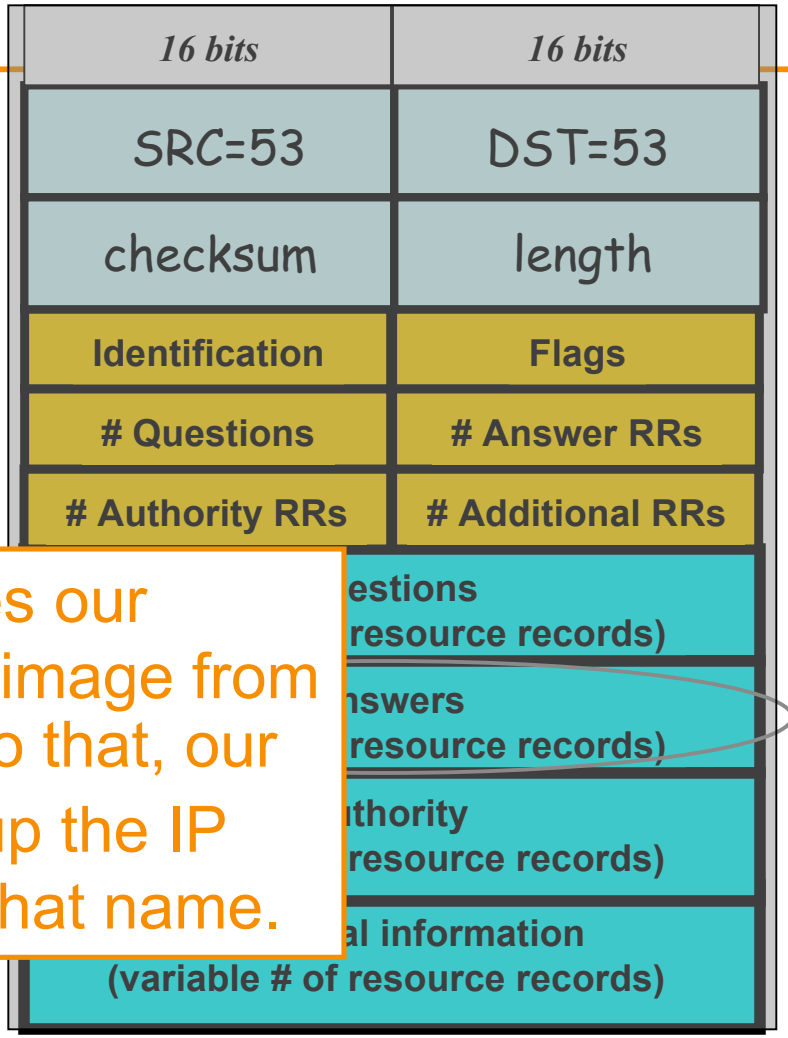
Suppose, e.g., we visit a web page under their control:

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

DNS Threats, con't

What about *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" …> ...
```

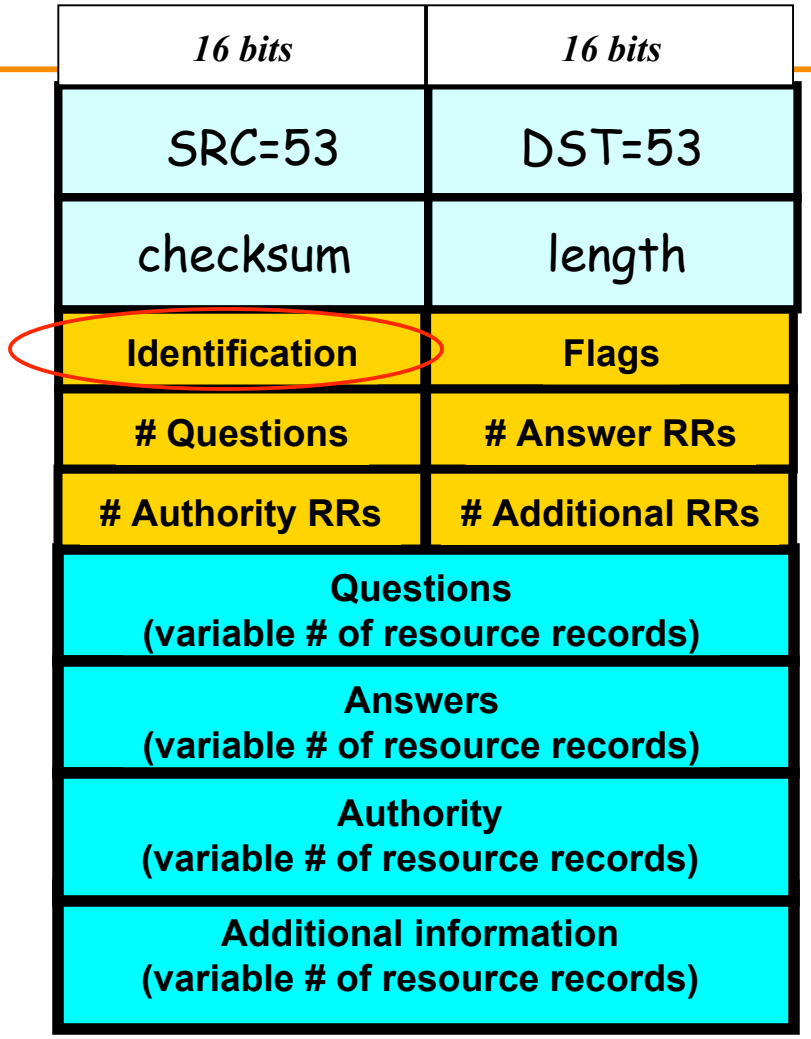
DNS Blind Spoofing, con't

Fix?

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?




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

They observe ID k here

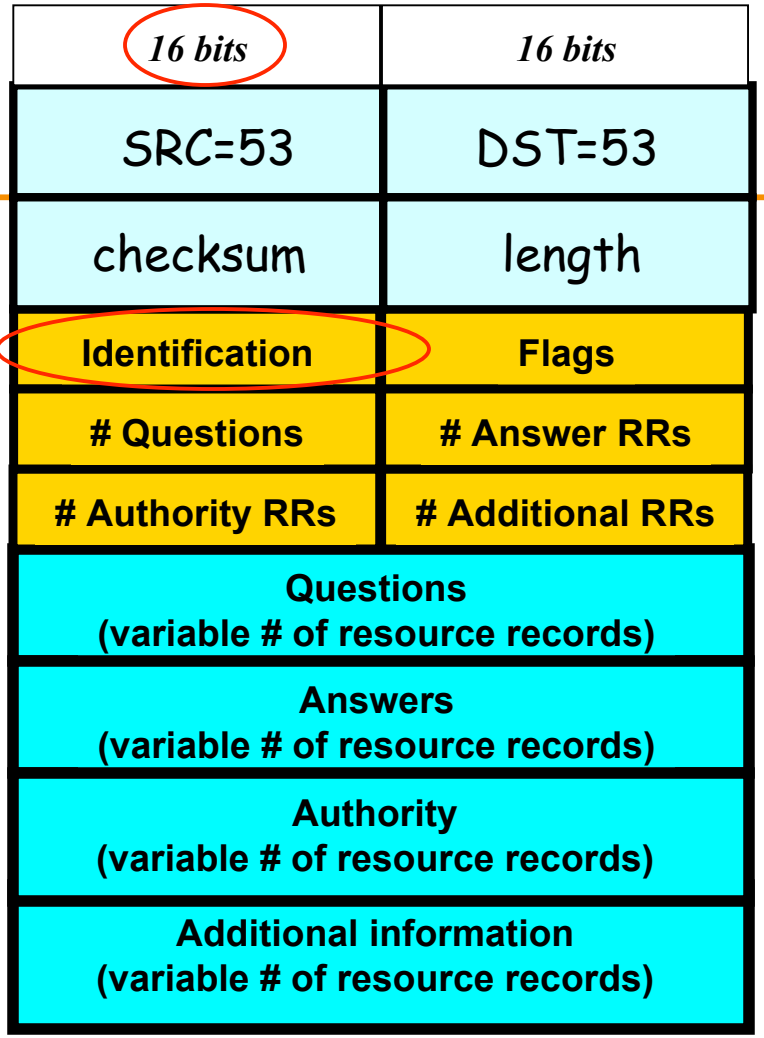
So this will be k+1

DNS Blind Spoofing, con't

Once we randomize the Identification, attacker has a 1/65536 chance of guessing it correctly.
Are we pretty much 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, we're likely safe - phew! ?

DNS Blind Spoofing (Kaminsky 2008)

- Two key ideas:
  - Spoof uses **Additional** field (rather than **Answer**)
  - 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" …>
```

Kaminsky Blind Spoofing, con't

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

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

;; ANSWER SECTION:
randomk.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, con't

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

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

;; ANSWER SECTION:
randomk.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
```

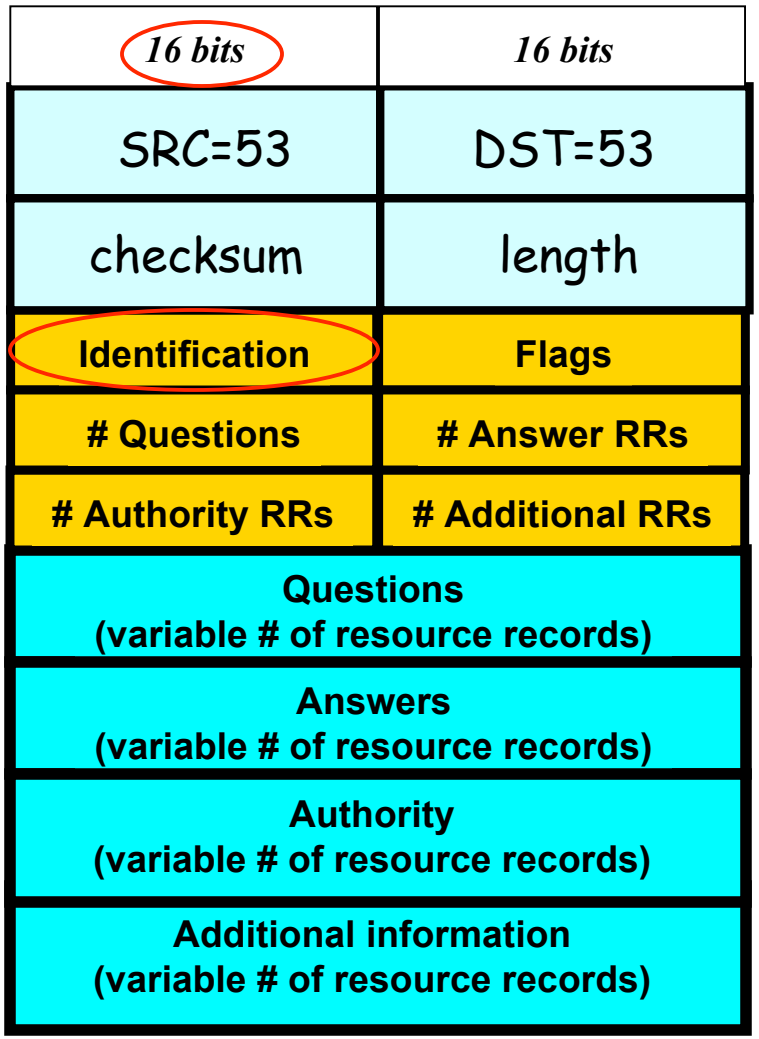
Once they win the race, not only have they poisoned mail.google.com … but also the cached NS record for google.com's name server - so any **future** *X*.google.com lookups *go through the attacker's machine*

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.)

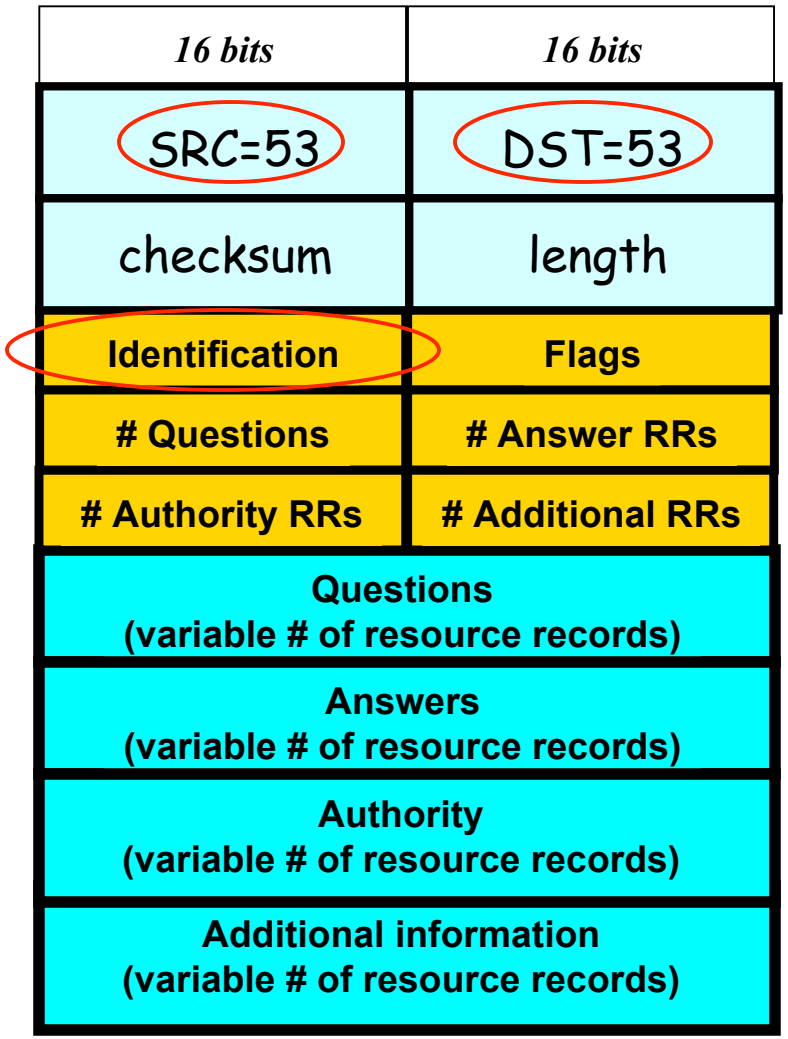


Defending Against Blind Spoofing

Total *entropy*: 16 bits

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

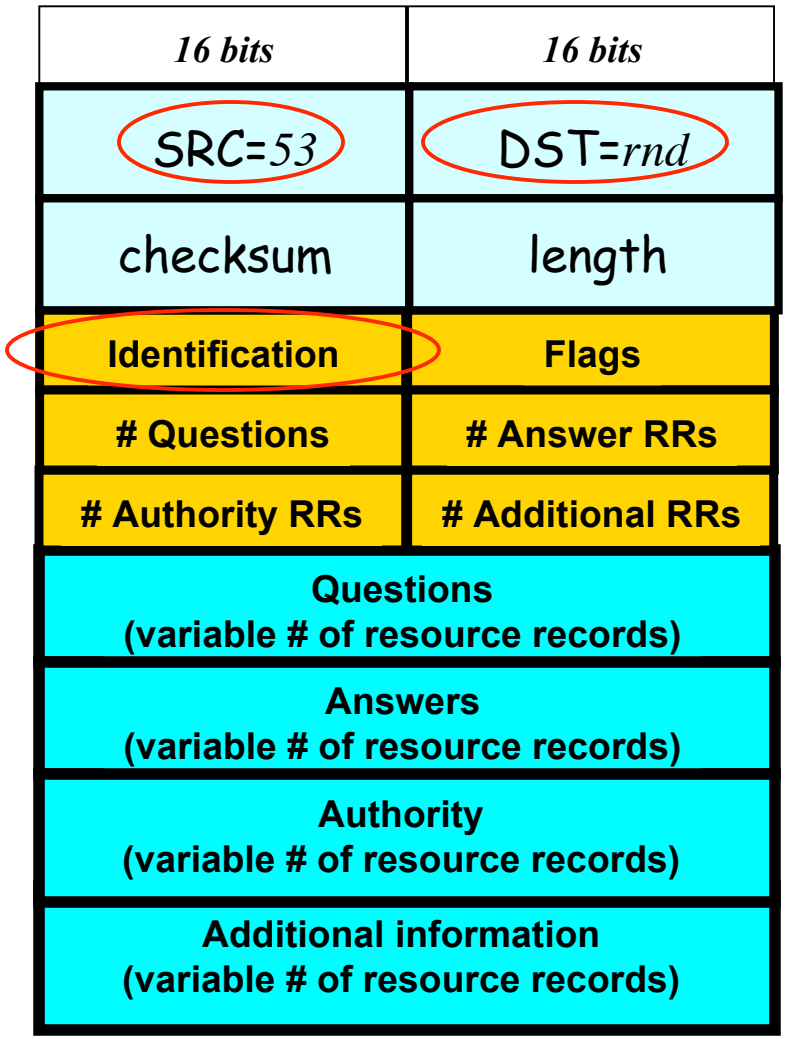
On a request, DST port = 53. SRC port usually also 53 - but not fundamental, just convenient.



Defending Against Blind Spoofing

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

Total *entropy*: ? bits

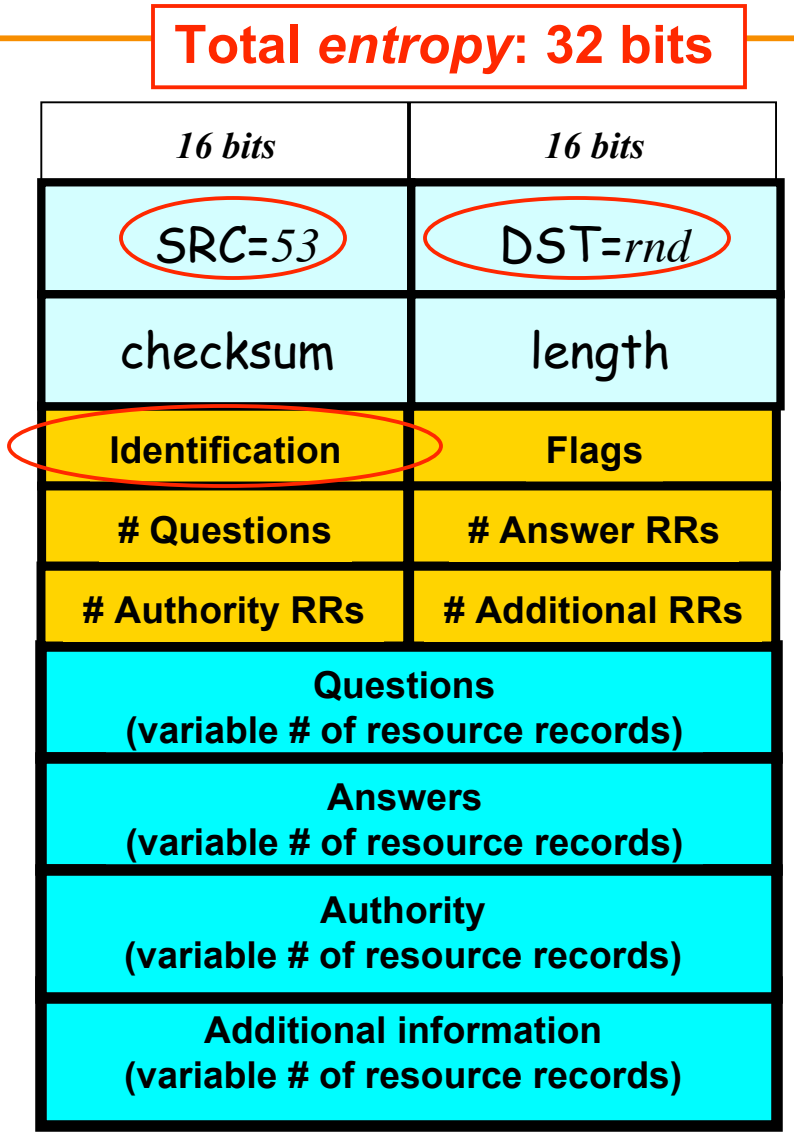


Defending Against Blind Spoofing

"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. (Note: not all resolvers have implemented random source ports!)

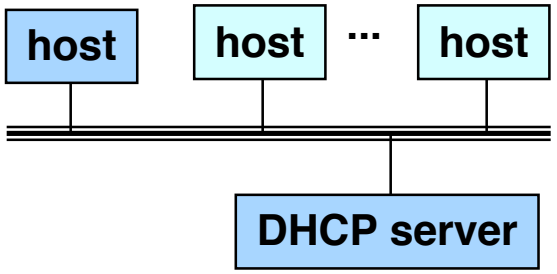


Summary of DNS Security Issues

- 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

Internet Bootstrapping: DHCP

- New host doesn't have an IP address yet
  - So, host doesn't know what source address to use
- Host doesn't know *who to ask* for an IP address
  - So, host doesn't know what destination address to use
- Solution: *shout* to "**discover**" server that can help
  - Broadcast a server-discovery message (layer 2)
  - Server(s) sends a reply offering an address



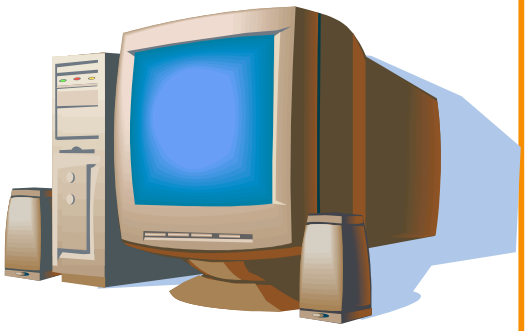
DHCP = Dynamic Host Configuration Protocol

Dynamic Host Configuration Protocol

new client

DHCP discover (broadcast)

DHCP offer



DHCP server

"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

DNS server = system used by client to map hostnames like `gmail.com` to IP addresses like `74.125.224.149`

Gateway router = router that client uses as the first hop for all of its Internet traffic to remote hosts

Dynamic Host Configuration Protocol

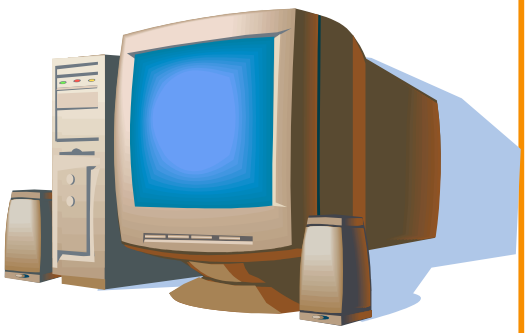
new client

DHCP discover (broadcast)

DHCP offer

DHCP request (broadcast)

DHCP ACK



DHCP server

"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

Dynamic Host Configuration Protocol

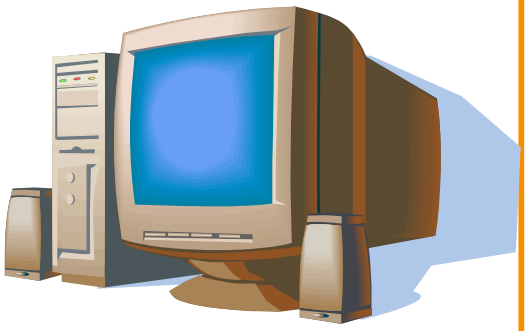
new client

DHCP discover (broadcast)

DHCP offer

DHCP request (broadcast)

DHCP ACK



DHCP server

"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

Threats?

Dynamic Host Configuration Protocol

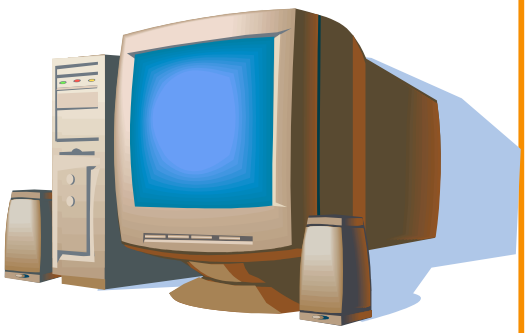
new client

DHCP discover (broadcast)

DHCP offer

DHCP request (broadcast)

DHCP ACK



DHCP server

Local attacker on same subnet can **hear** new host's DHCP request

"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

Dynamic Host Configuration Protocol

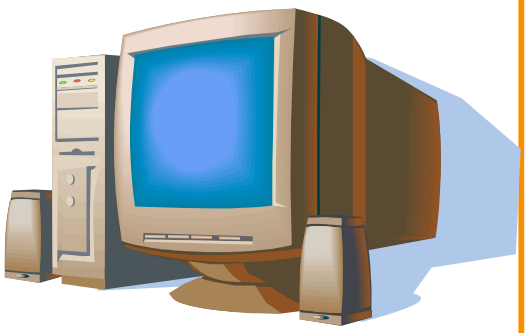
new client

DHCP discover (broadcast)

DHCP offer

DHCP request (broadcast)

DHCP ACK



DHCP server

"**offer**" message includes IP address, DNS server, "gateway router", and how long client can have these ("lease" time)

Attacker can **race** the actual server; if attacker wins, replaces DNS server and/or gateway router

DHCP Threats

- Substitute a fake DNS server
  - Redirect any of a host's lookups to a machine of attacker's choice (e.g., `gmail.com` = `6.6.6.6`)
- Substitute a fake gateway router
  - Intercept all of a host's off-subnet traffic
    - (even if not preceded by a DNS lookup)
  - Relay contents back and forth between host and remote server
    - Modify however attacker chooses
  - This is one type of invisible *Man In The Middle* (MITM)
    - Victim host generally has no way of knowing it's happening! :-(
    - (Can't necessarily alarm on peculiarity of receiving multiple DHCP replies, since that can happen benignly)
- How can we fix this? ***Hard***

Summary of DHCP Security Issues

- DHCP threats highlight:
  - Broadcast protocols inherently at risk of **local** attacker spoofing
    - Attacker knows exactly when to try it …
    - … and can see the victim's messages
  - When initializing, systems are particularly vulnerable because they can *lack a trusted foundation* to build upon
  - Tension between wiring in trust vs. flexibility/convenience
  - MITM attacks insidious because no indicators they're occurring