Network security (DNS caching and DoS)

CS 161: Computer Security

Prof. Raluca Ada Popa

March 1, 2018

Slides adapted from David Wagner

DNS Overview

- DNS translates www.google.com to 74.125.25.99
- It’s a performance-critical distributed database.
- DNS security is critical for the web.

- Analogy: If you don’t know the answer to a question, ask a friend for help (who may in turn refer you to a friend of theirs, and so on).

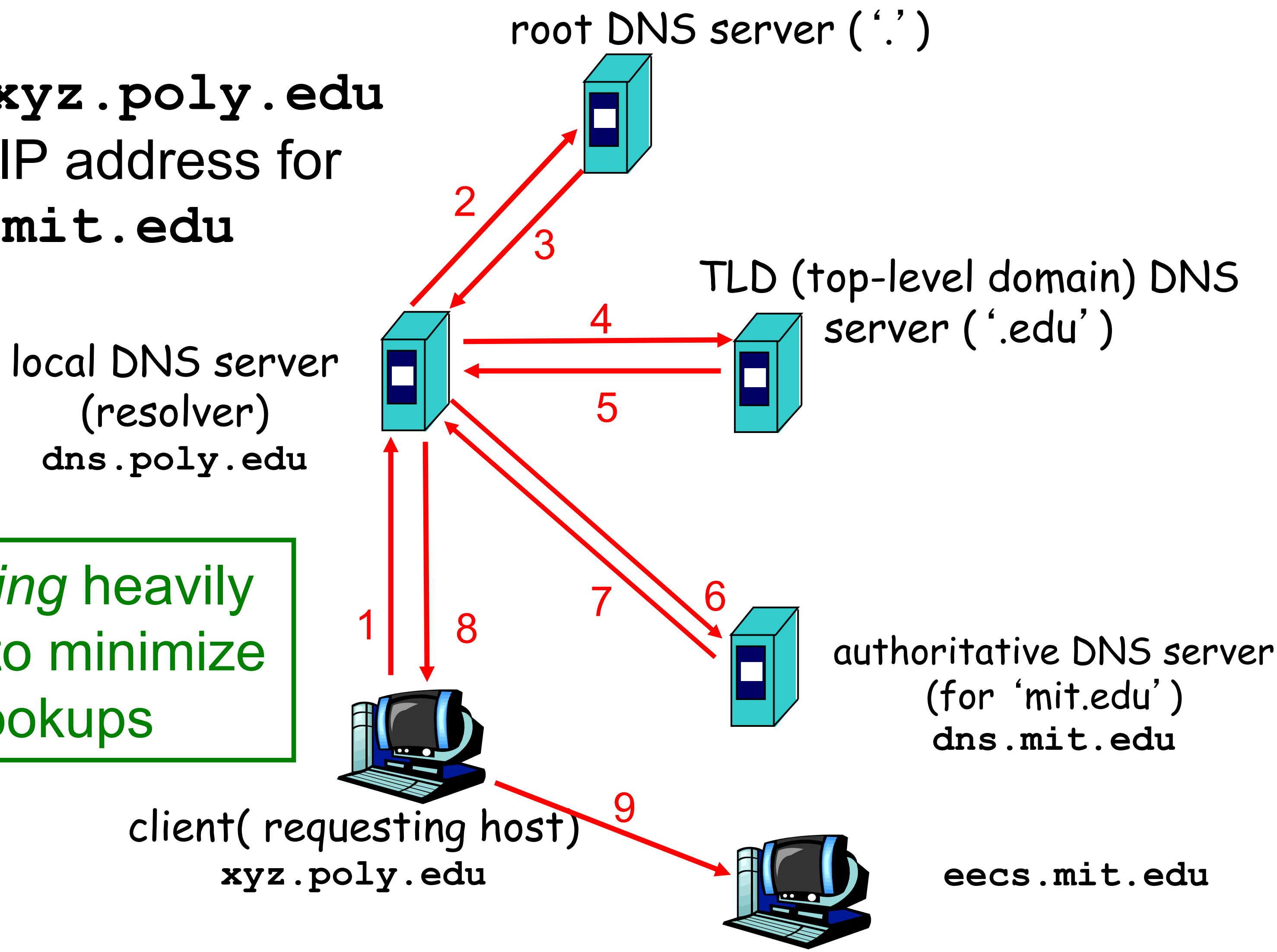
DNS Overview

- DNS translates www.google.com to 74.125.25.99
- It’s a performance-critical distributed database.
- DNS security is critical for the web.

- Analogy: If you don’t know the answer to a question, ask a friend for help (who may in turn refer you to a friend of theirs, and so on).
- Security risks: friend might be malicious, communication channel to friend might be insecure, friend might be well-intentioned but misinformed

DNS Lookups via a *Resolver*

Host at **`xyz.poly.edu`** wants IP address for **`eecs.mit.edu`**



Caching heavily used to minimize lookups

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

Security risk #2: on-path attacker

- If attacker can eavesdrop on our traffic… we’re hosed.
- Why? We’ll see why.

Security risk #3: off-path attacker

- If attacker can’t eavesdrop on our traffic, can he inject spoofed DNS responses?
- Yes. This case is especially interesting, so we’ll look at it in detail.

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/TCP, can spoof responses
- 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     IN      NS      BITSY.mit.edu.
mit.edu.                  11088     IN      NS      W20NS.mit.edu.
mit.edu.                                            RAWB.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.           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
;; 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 SECTIO
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-APl
;; 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 (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 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.

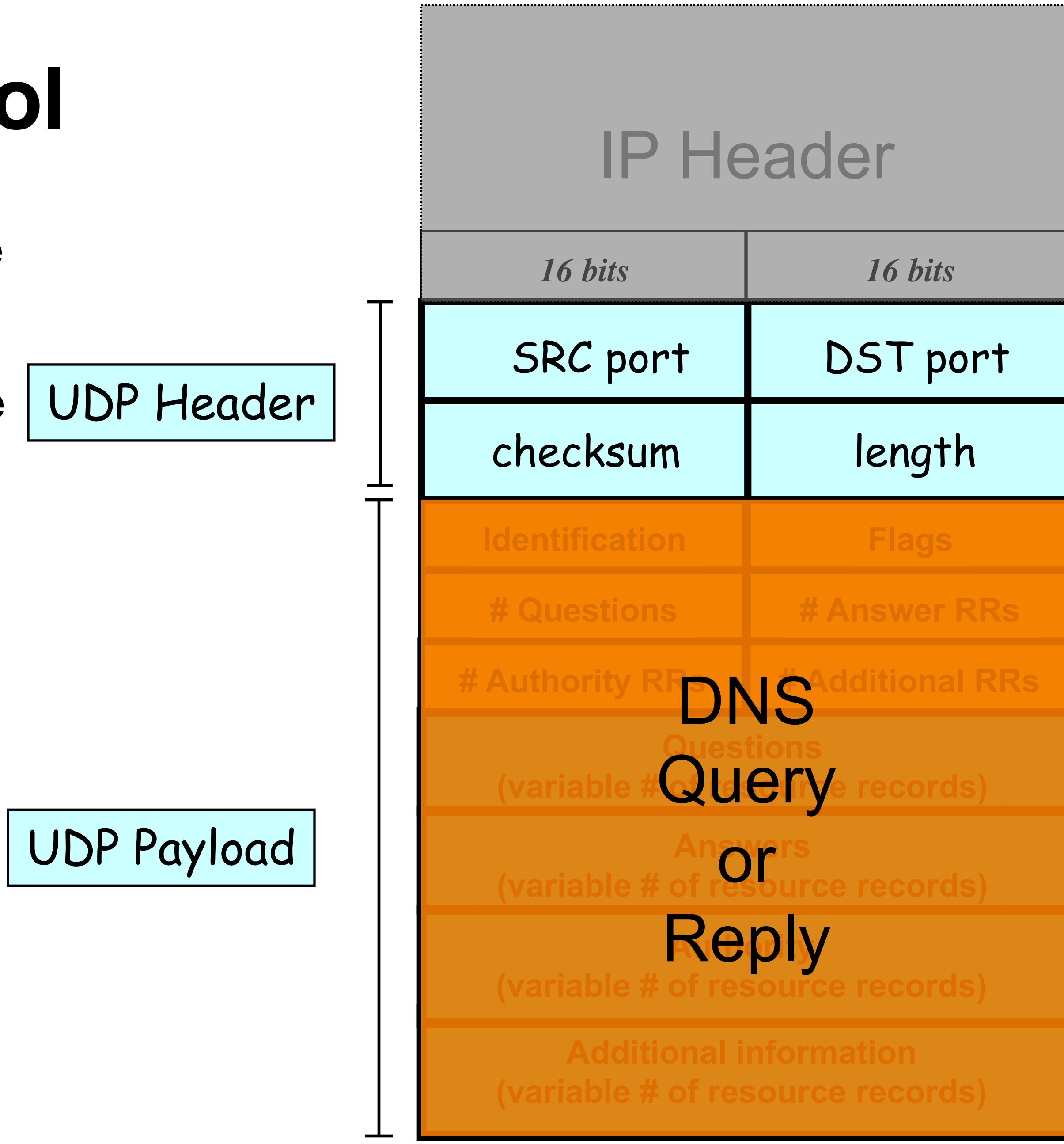
Here, it tells us the IP addresses for the hostnames of the name servers. We add these to our cache.

DNS Protocol

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

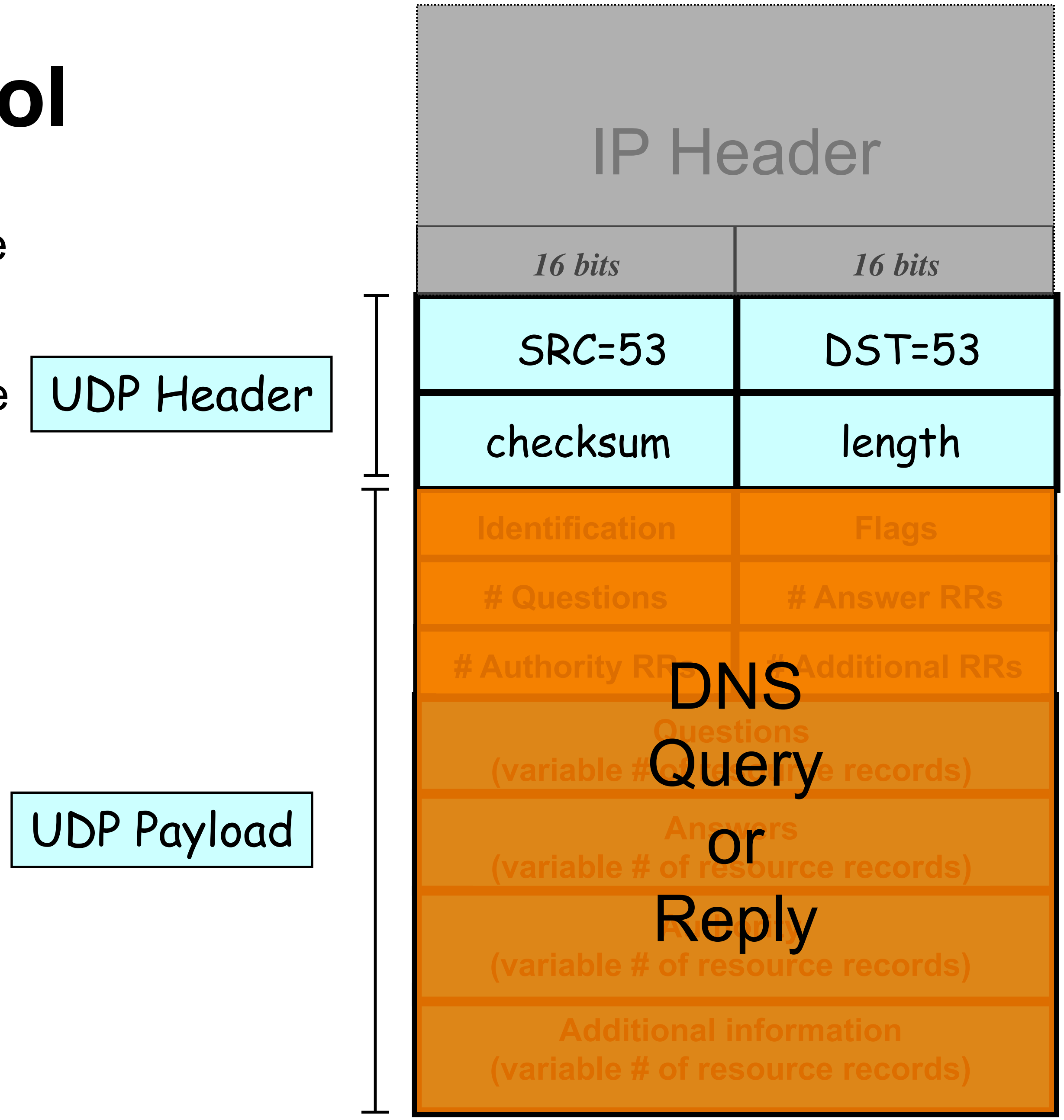
Primarily uses UDP for its transport protocol, which is what we'll assume

Frequently, both clients and servers use port 53



DNS Protocol

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



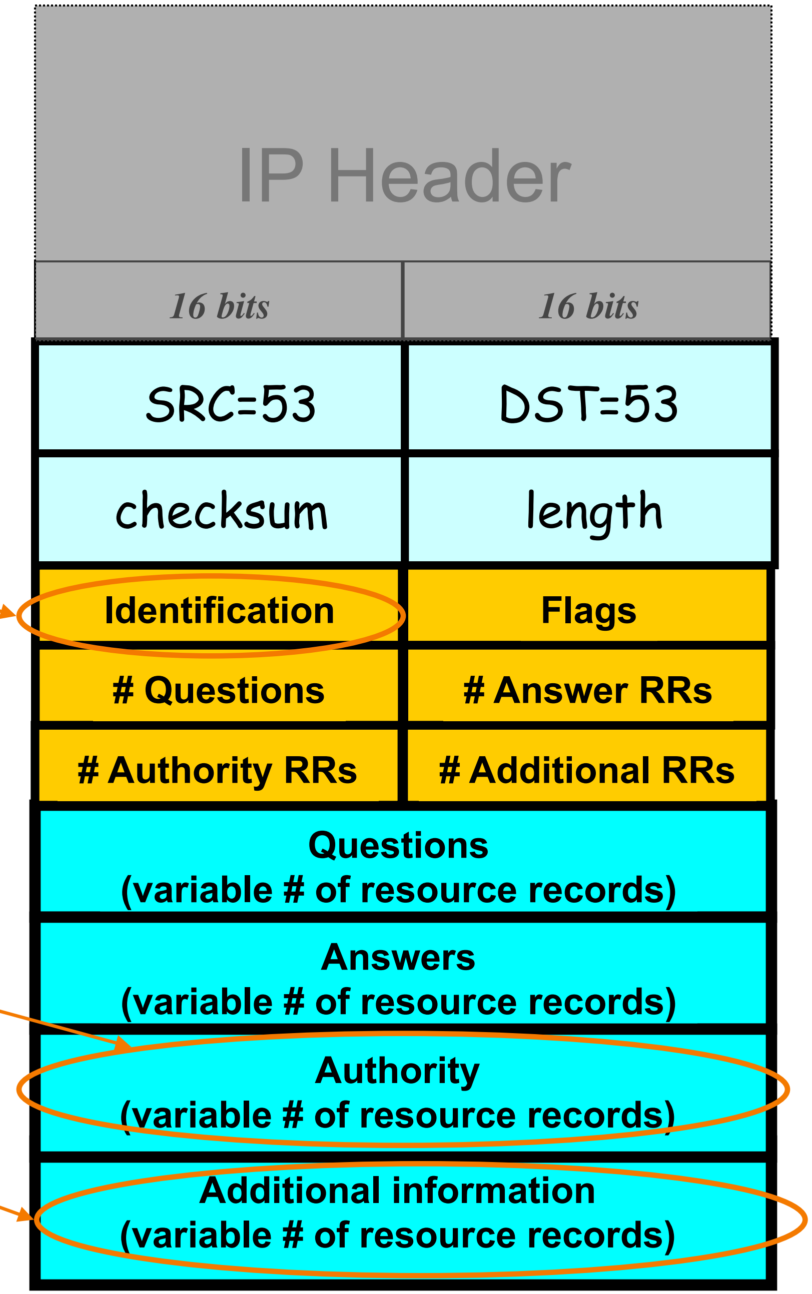
- Primarily uses UDP for its transport protocol, which is what we'll assume

- Frequently, both clients and servers use port 53

DNS Protocol, 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)*



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                                   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

```
; ; <<>> 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 DNS ***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.

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).

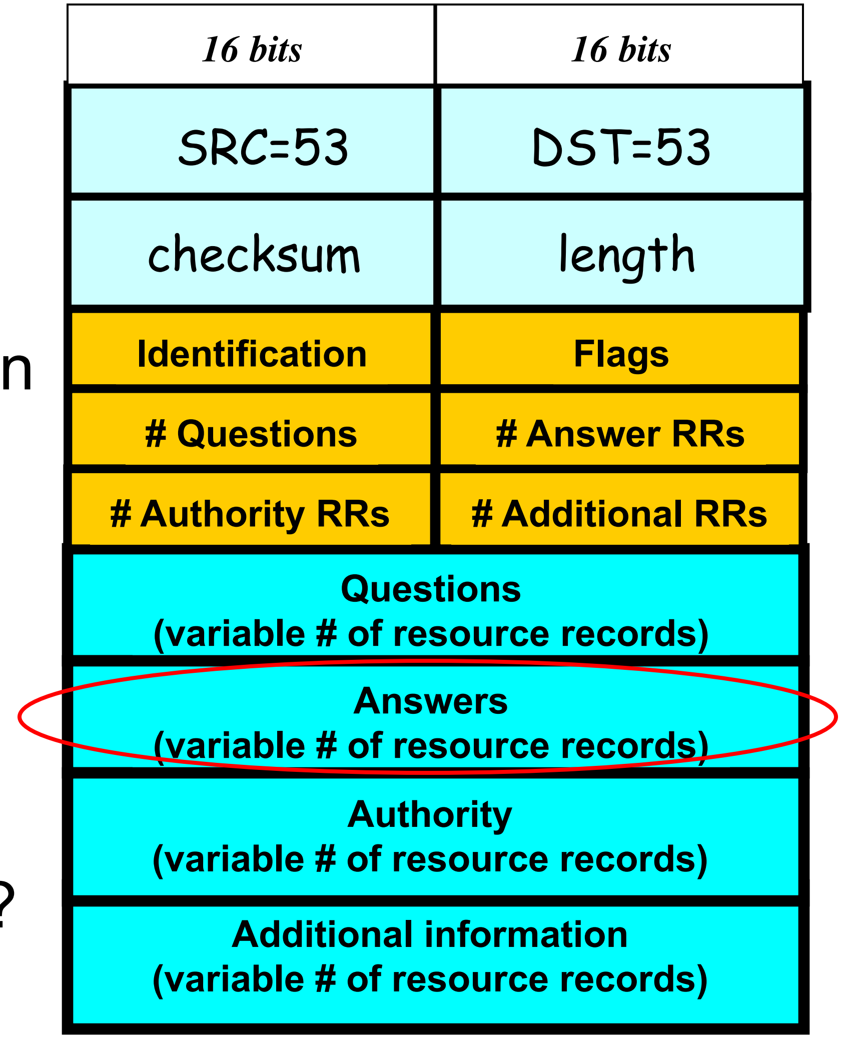
Blind spoofing

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

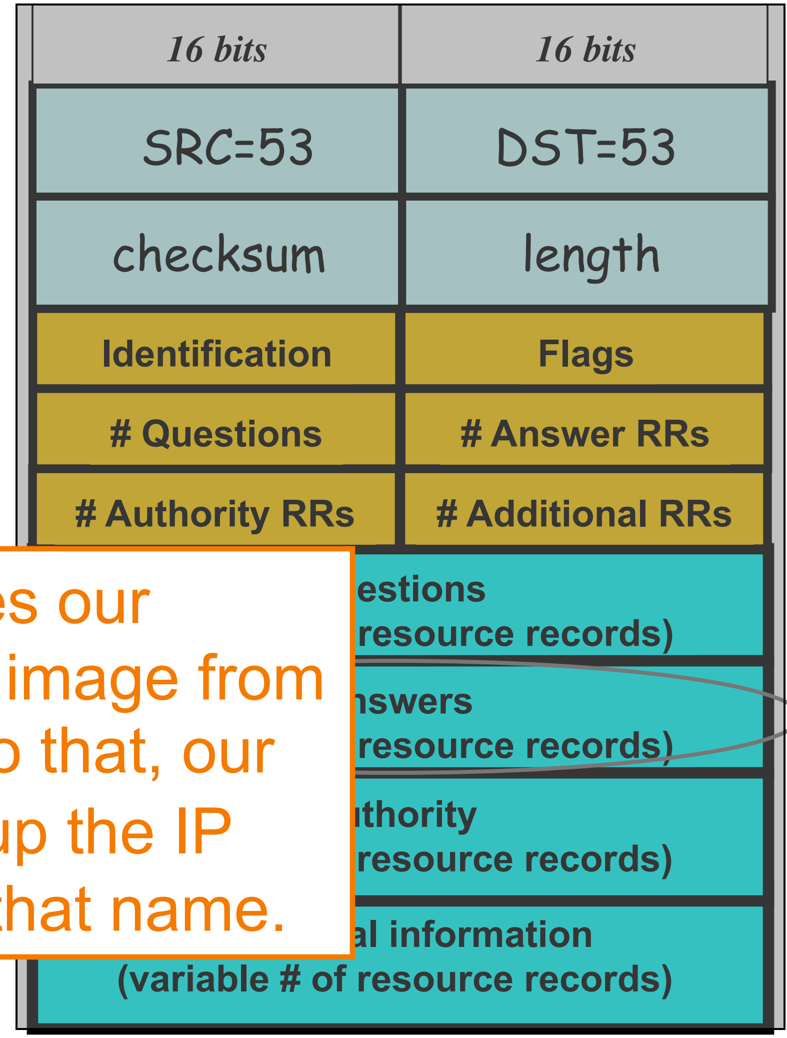
- How can such a **remote** attacker even know we are looking up `mail.google.com`?

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 c 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

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?

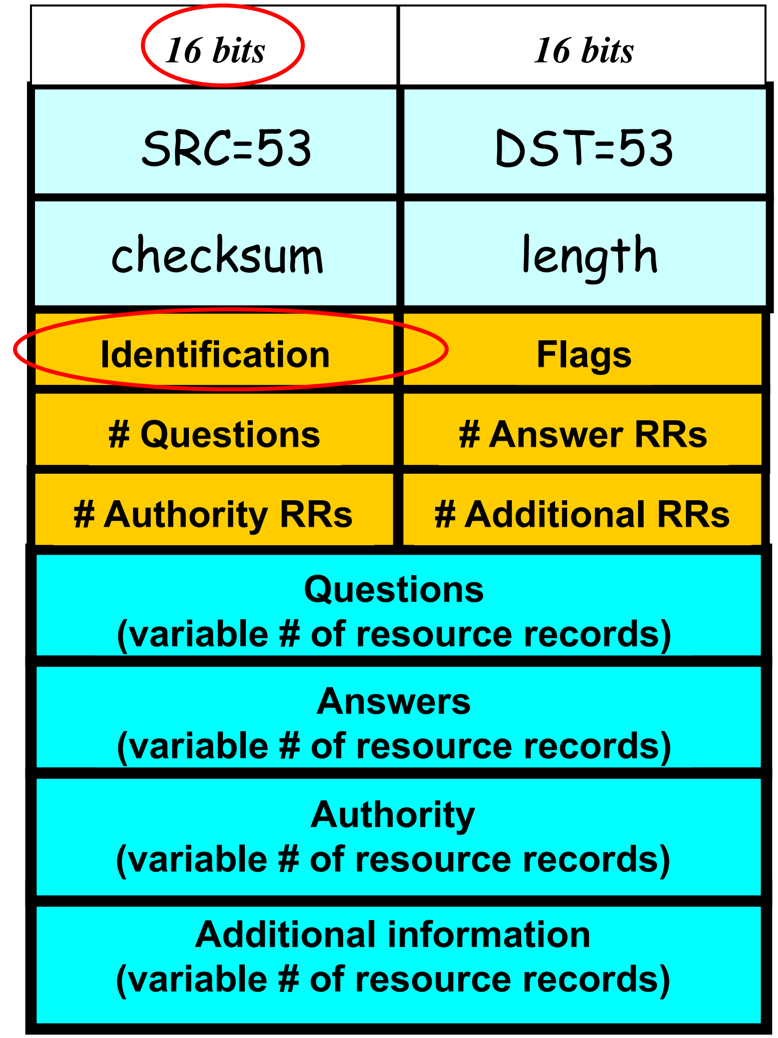
Originally, identification field incremented by 1 for each request. How does attacker guess it?

16 bits	*16 bits*
SRC=53	DST=53
checksum	length
Identification	**Flags**
# Questions	**# Answer RRs**
# Authority RRs	**# Additional RRs**
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

`<img src="http://badguy.com" …>` They observe ID k here

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

Summary of DNS Security Issues

- DNS threats highlight:
  - Attackers can attack opportunistically rather than eavesdropping
    - o 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
    - o 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

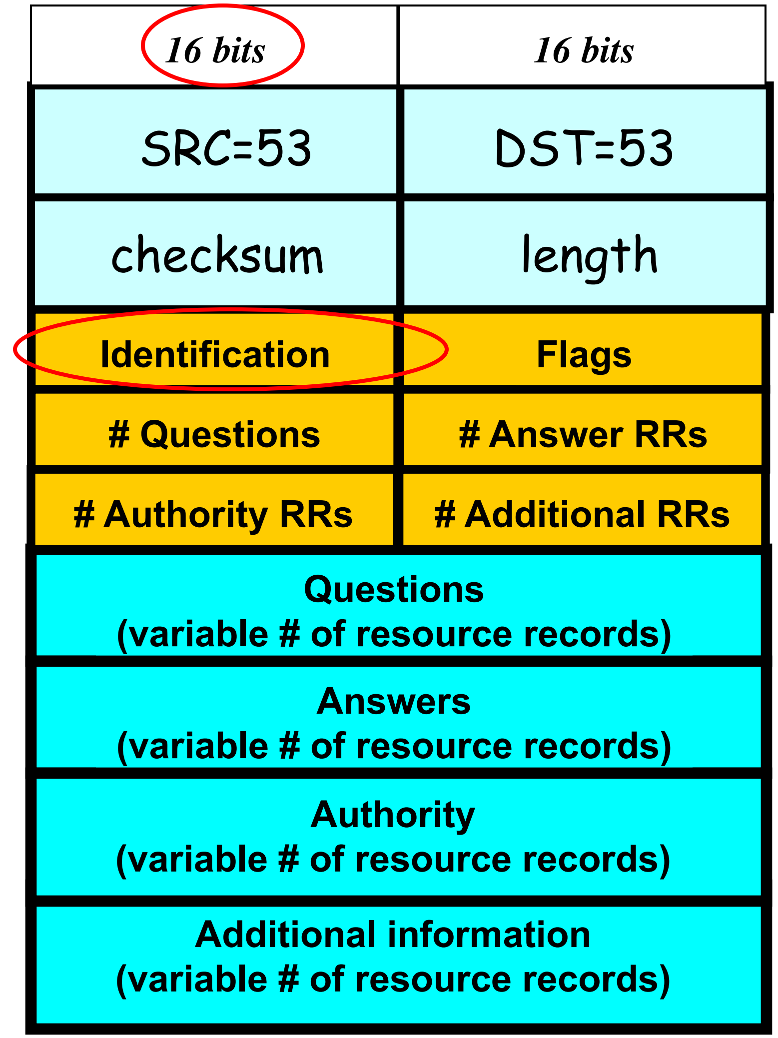
- (Note, these tend to be pessimistic … but prudent)
- Attackers can interact with our systems without particular notice
  - *Probing* (poking at systems) may go unnoticed …
  - … even if highly repetitive, leading to crashes, and *easy to detect*
- It's easy for attackers to know general information 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 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:
  - 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.

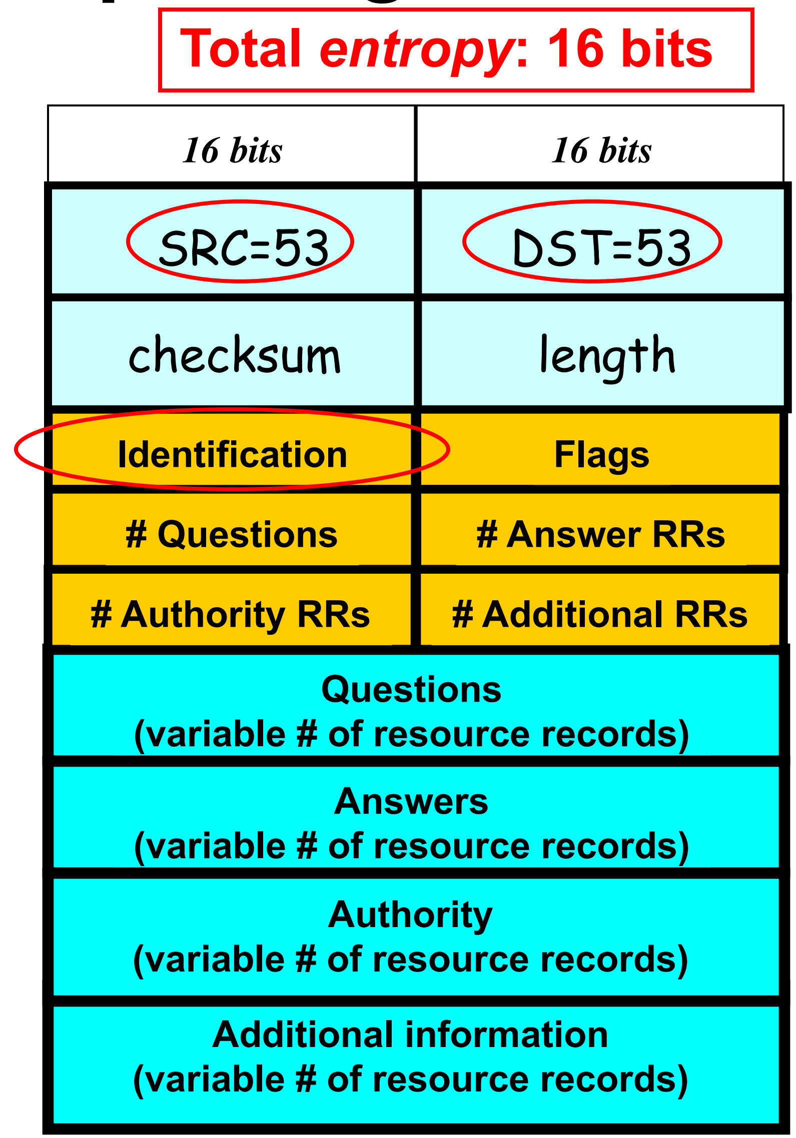
Where can we get more entropy? (*Without* requiring a protocol change.)

16 bits	*16 bits*
SRC=53	DST=53
checksum	length
Identification	**Flags**
# Questions	**# Answer RRs**
# Authority RRs	**# Additional RRs**
Questions **(variable # of resource records)**	
Answers **(variable # of resource records)**	
Authority **(variable # of resource records)**	
Additional information **(variable # of resource records)**	

Defending Against Blind Spoofing

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

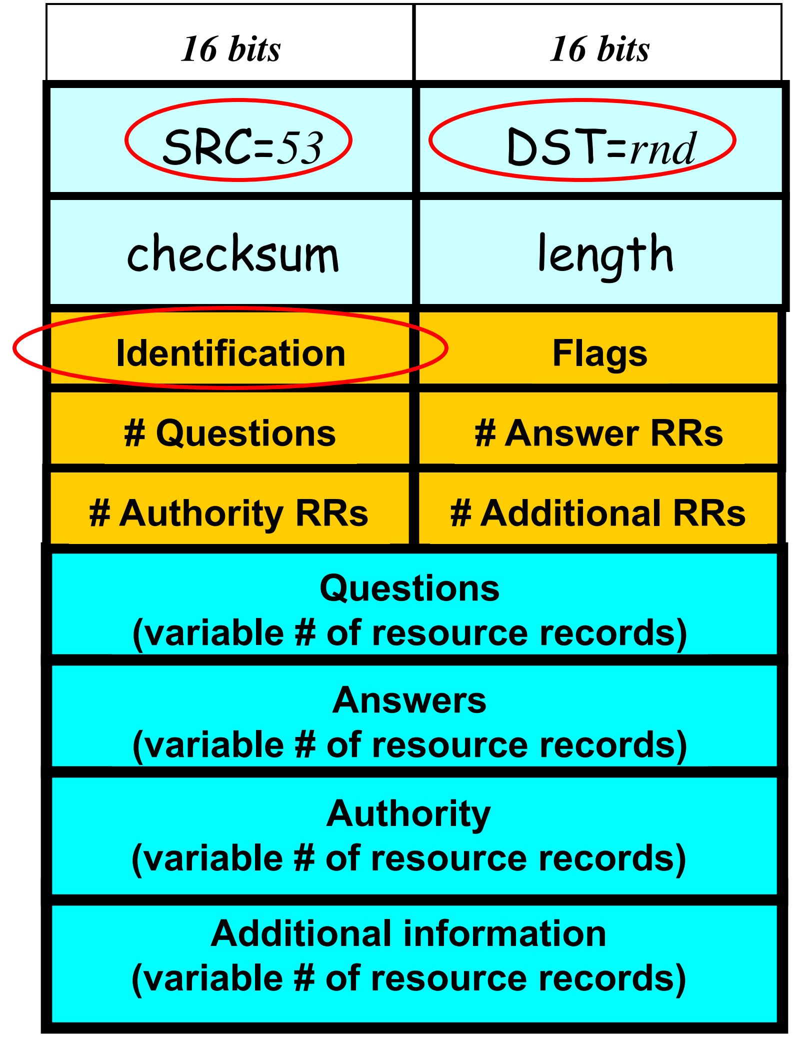
On a request, DST port = 53. SRC port usually also 53 – but not fundamental, just convenient.



Defending Against Blind Spoofing

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

Total *entropy*: ? bits

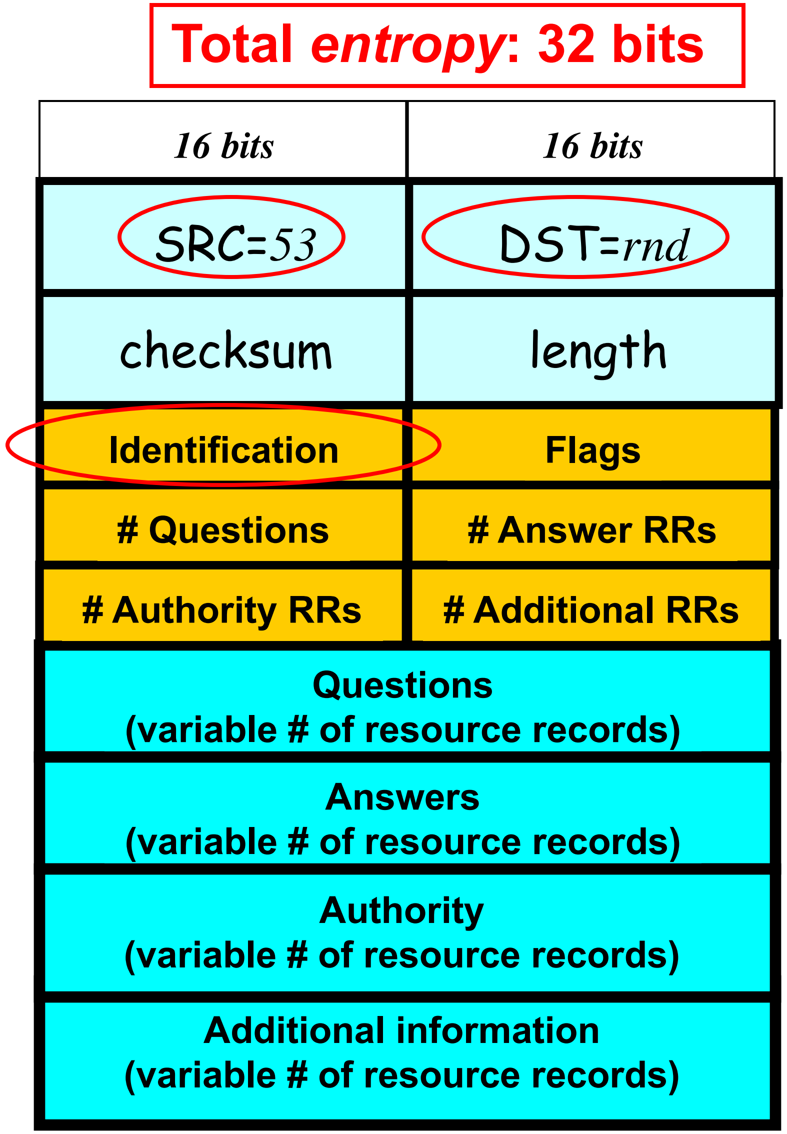


Defending Against Blind Spoofing

"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.



Lessons learned

- Special risks of caching and distributed systems where information is spread across many machines

- Security risks: friend (cache) might be malicious
- Communication channel to friend (cache) might be insecure
- Friend (cache) might be well-intentioned but misinformed

Denial-of-Service (DoS)

Attacks on Availability

- Denial-of-Service (DoS): *preventing legitimate users from using a computing service*
- Distributed Denial-of-Service *(DDoS)* occurs when a server is flooded with traffic from many different devices
- We do though need to consider our threat model …
  - What might motivate a DoS attack?

Botnets Beat Spartan Laser on *Halo 3*

By Kevin Poulsen | February 4, 2009 | 12:13 pm | Categories: Cybarmageddon!

What's the most powerful weapon you can wield when playing *Halo 3* online?

I know. You can control the entire map with a battle rifle and a couple of sticky grenades. But that teeny-bopper you just pwned has you beat with the tiny botnet he leased with his allowance money.

Motivations for DoS

- Showing off / entertainment / ego
- Competitive advantage
  - Maybe commercial, maybe just to win
- Vendetta / denial-of-money
- Extortion
- Political statements
- Impair defenses
- Espionage
- Warfare

There are dozens of underground forums where members advertise their ability to execute debilitating "distributed denial-of-service" or DDoS attacks for a price. DDoS attack services tend to charge the same prices, and the average rate for taking a Web site offline is surprisingly affordable: about \$5 to \$10 per hour; \$40 to \$50 per day; \$350-\$400 a week; and upwards of \$1,200 per month.

An ad for a DDoS attack service.

Of course, it pays to read the fine print before you enter into any contract. Most DDoS services charge varying rates depending on the complexity of the target's infrastructure, and how much lead time the attack service is given to size up the mark. Still, buying in bulk always helps: One service advertised on several fraud forums offered discounts for regular and wholesale customers.

Extortion via DDoS on the rise

By Denise Pappalardo and Ellen Messmer, Network World, 05/16/05

Criminals are increasingly targeting corporations with distributed denial-of-service attacks designed not to disrupt business networks but to extort thousands of dollars from the companies.

Ivan Maksakov, Alexander Petrov and Denis Stepanov were accused of receiving $4 million from firms that they threatened with cyberattacks.

The trio concentrated on U.K. Internet gambling sites, according to the prosecution. One bookmaker, which refused to pay a demand for $10,000, was attacked and brought offline--which reportedly cost it more than $200,000 a day in lost business.

NOV 06

8

DDoS makes a phishing e-mail look real

Posted by Munir Kotadia @ 12:00 0 comments

Just as Internet users learn that clicking on a link in an e-mail purporting to come from their bank is a bad idea, phishers seem to be developing a new tactic -- launch a DDoS attack on the Web site of the company whose customers they are targeting and then send e-mails "explaining" the outage and offering an "alternative" URL.

November 17th, 2008

Anti fraud site hit by a DDoS attack

Posted by Dancho Danchev @ 4:01 pm

Categories: Botnets, Denial of Service (DoS), Hackers, Malware, Pen testing...

Tags: Security, Cybercrime, DDoS, Fraud, Bobbear...

9 TalkBacks ADD YOUR OPINION | SHARE | PRINT | E-MAIL | WORTHWHILE? +2 4 VOTES

The popular British anti-fraud site **Bobbear.co.uk** is currently under a DDoS attack (distributed denial of service attack) , originally launched last Wednesday, and is continuing to hit the site with 3/4 million hits daily from hundreds of thousands of malware infected hosts mostly based in Asia and Eastern Europe, according to the site's owner. Targeted DDoS attacks against anti-fraud and volunteer cybercrime fighting communities clearly indicate the impact these communities have on the revenue stream of scammers, and with Bobbear attracting such a high profile underground attention, the site is indeed doing a very good job.

December 8, 2010, 4:18 PM

‘Operation Payback’ Attacks Fell Visa.com

By ROBERT MACKEY

A message posted on Twitter by a group of Internet activists announcing the start of an attack on Visa’s Web site, in retaliation for the company’s actions against WikiLeaks.

Last Updated | 6:54 p.m. A group of Internet activists took credit for crashing the Visa.com Web site on Wednesday afternoon, hours after they launched a similar attack on MasterCard. The cyber attacks, by activists who call themselves Anonymous, are aimed at punishing companies that have acted to stop the flow of donations to WikiLeaks in recent days.

The group explained that its distributed denial of service attacks — in which they essentially flood Web sites site with traffic to slow them down or knock them offline — were part of a broader effort called Operation Payback, which

---

9. In the past year, has your site been subjected to a denial of service attack, meaning an attacker prevented or attempted to prevent access to your site altogether?

#	Answer	Bar	Response	%
1	yes		21	62%
2	no		8	24%
3	not sure		5	15%
	Total		34	

Row over Korean election DDoS attack heats up

Ruling party staffer accused of disrupting Seoul mayoral by-election

By John Leyden • Get more from this author

Posted in Security, 7th December 2011 09:23 GMT

A political scandal is brewing in Korea over alleged denial of service attacks against the National Election Commission (NEC) website.

Police have arrested the 27-year-old personal assistant of ruling Grand National Party politician Choi Gu-sik over the alleged cyber-assault, which disrupted a Seoul mayoral by-election back in October.

However, security experts said that they doubt the suspect, identified only by his surname "Gong", had the technical expertise or resources needed to pull off the sophisticated attack.

Row over Korean election DDoS attack heats up

Ruling party staffer accused of disrupting Seoul mayoral by-election

By **John Leyden** • **Get more from this author**

Posted in Security, 7th December 2011 09:23 GMT

A political scandal is brewing in Korea over alleged denial of service attacks against the National Election Commission (NEC) website.

Police have arrested the 27-year-old personal assistant of ruling Grand National Party politician Choi Gu-sik over the alleged cyber-assault, which disrupted a Seoul mayoral by-election back in October.

However, security experts said that they doubt the suspect, identified only by his surname "Gong", had the technical expertise or resources needed to pull off the sophisticated attack.

Gong continues to protest his innocence, a factor that has led opposition politicians to speculate that he is covering up for higher-ranking officials who ordered the attack.

Democratic Party politician Baek Won-woo told *The HankYoreh*: “We need to determine quickly and precisely whether there was someone up the line who ordered the attack, and whether there was compensation.” ®

Russia accused of unleashing cyberwar to disable Estonia

· Parliament, ministries, banks, media targeted
· Nato experts sent in to strengthen defences

Ian Traynor in Brussels
The Guardian, Thursday 17 May 2007
Article history

Bronze Soldier, the Soviet war memorial removed from Tallinn. Nisametdinov/AP

A three-week wave of massive cyber-attacks on the small Baltic country of Estonia, the first known incidence of such an assault on a state, is causing alarm across the western alliance, with Nato urgently examining the offensive and its implications.

August 11th, 2008

Coordinated Russia vs Georgia cyber attack in progress

Posted by Dancho Danchev @ 4:23 pm

Categories: Black Hat, Botnets, Denial of Service (DoS), Governments, Hackers...
Tags: Security, Cyber Warfare, DDoS, Georgia, South Osetia...

62 TalkBacks
ADD YOUR OPINION | SHARE | PRINT | E-MAIL | WORTHWHILE? +18 24 VOTES

In the wake of the Russian-Georgian conflict, a week worth of speculations around Russian Internet forums have finally materialized into a coordinated cyber attack against Georgia's Internet infrastructure. The attacks have already managed to compromise several government web sites, with continuing DDoS attacks against numerous other Georgian government sites, prompting the government to switch to hosting locations to the U.S, with Georgia's Ministry of Foreign Affairs undertaking a desperate step in order to disseminate real-time information by moving to a Blogspot account

Posted on Tuesday, August 12th, 2008 | Bookmark on del.icio.us

Georgia DDoS Attacks - A Quick Summary of Observations

by Jose Nazario

The clashes between Russia and Georgia over the region of South Ossetia have been shadowed by attacks on the Internet. As we noted in July, the Georgia presidential website fell victim to attack during a war of words. A number of DDoS attacks have

> Raw statistics of the attack traffic paint a pretty intense picture. We can discern that the attacks would cause injury to almost any common website.

Average peak bits per second per attack	211.66 Mbps
Largest attack, peak bits per second	814.33 Mbps
Average attack duration	2 hours 15 minutes
Longest attack duration	6 hour

ATLAS Peak Monitored Attack Sizes Month-By-Month (January 2009-Present)

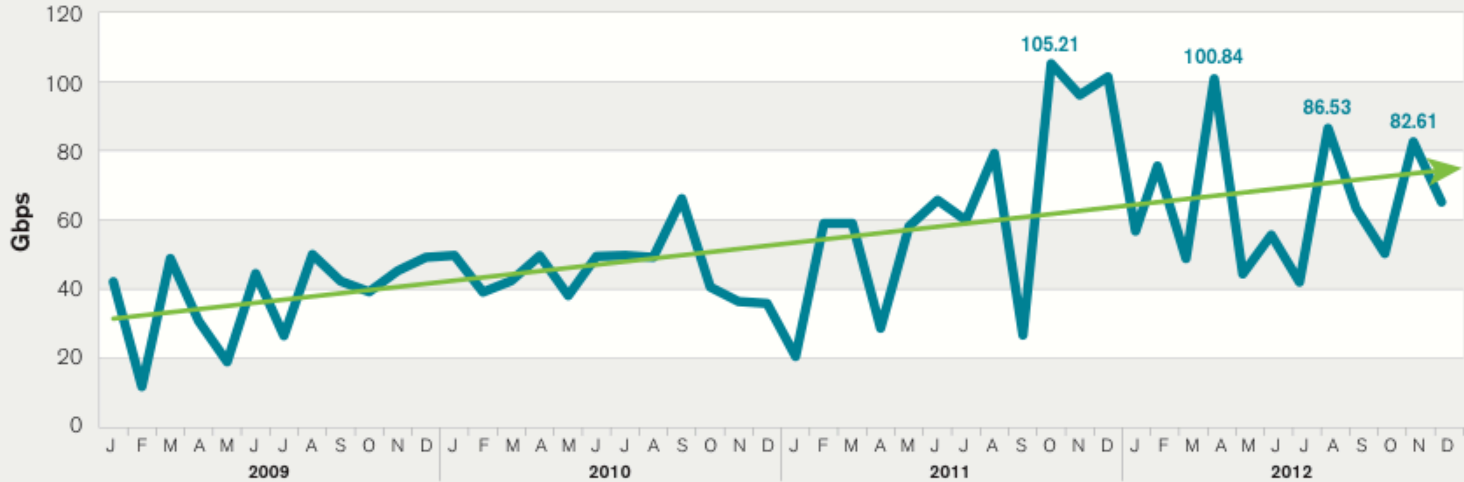


Figure 17 *Source: Arbor Networks, Inc.*

ATLAS Average Monitored Attack Sizes Month-By-Month (January 2009-Present)

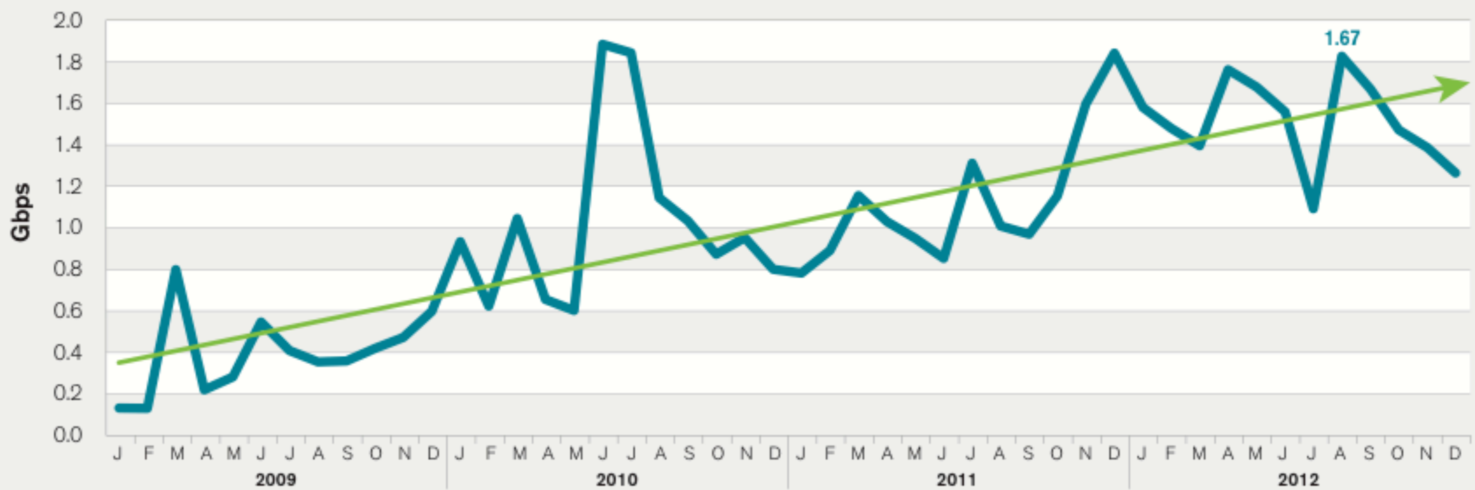


Figure 18 *Source: Arbor Networks, Inc.*

Most Significant Operational Threats

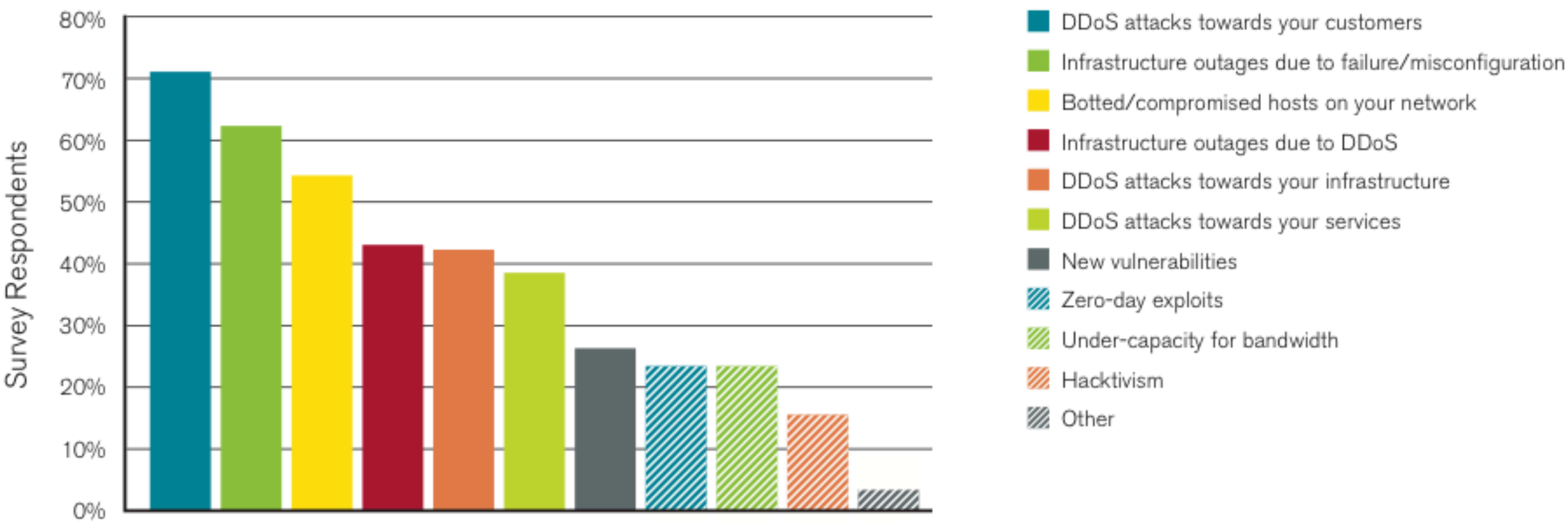


Figure 6 *Source: Arbor Networks, Inc.*

GITHUB ATTACK PERPETRATED BY CHINA'S GREAT CANNON TRAFFIC INJECTION TOOL

by **Brian Donohue** April 10, 2015 , 1:06 pm

Chinese attackers used the Great Firewall's offensive sister-system, named the Great Cannon, to launch a recent series of distributed denial of service attacks targeting the anti-censorship site, GreatFire.org, and the code repository, Github, which was hosting content from the former.

Attacks on Availability

- Deny service via a **program flaw** (“`*NULL`”)
  - E.g., supply an input that crashes a server
  - E.g., fool a system into shutting down

- Deny service via **resource exhaustion** (“`while(1);`”)
  - E.g., consume CPU, memory, disk, network

- Network-level DoS vs application-level DoS

DoS & Operating Systems

- How could you DoS a multi-user Unix system on which you have a login?

DoS & Operating Systems

- How could you DoS a multi-user Unix system on which you have a login?
  - ```
    char buf[1024];
    int f = open("/tmp/junk");
    while (1) write(f, buf, sizeof(buf));
    ```
    - Gobble up all the disk space!
  - `while (1) fork();`
    - Create a zillion processes!
  - Create zillions of files, keep opening, reading, writing, deleting
    - Thrash the disk
  - … doubtless many more
- Defenses?
    ```

DoS & Operating Systems

- How could you DoS a multi-user Unix system on which you have a login?
  - ```
    char buf[1024];
    int f = open("/tmp/junk");
    while (1) write(f, buf, sizeof(buf));
    ```
    - Gobble up all the disk space!
  - `while (1) fork();`
    - Create a zillion processes!
  - Create zillions of files, keep opening, reading, writing, deleting
    - Thrash the disk
  - … doubtless many more
- Defenses?
  - Isolate users / impose quotas
    ```

Network-level DoS

- Can exhaust network resources by
  - Flooding with lots of packets (brute-force)
  - DDoS: flood with packets from many sources
  - Amplification: Abuse patsies who will amplify your traffic for you

DoS & Networks

- How could you DoS a target's Internet access?
  - Send a zillion packets at them
  - Internet lacks isolation between traffic of different users!
- What resources does attacker need to pull this off?
  - At least as much sending capacity ("bandwidth") as the bottleneck link of the target's Internet connection
    - Attacker sends maximum-sized packets
  - **Or**: overwhelm the rate at which the bottleneck router can process packets
    - Attacker sends minimum-sized packets!
      - (in order to maximize the packet arrival rate)

Defending Against Network DoS

- Suppose an attacker has access to a beefy system with high-speed Internet access (a “big pipe”).
- They pump out packets towards the target at a very high rate.
- What might the target do to defend against the onslaught?
  - Install a network filter to discard any packets that arrive with attacker’s IP address as their source
    - E.g., `drop * 66.31.1.37:* -> *:*`
    - Or it can leverage *any other pattern* in the flooding traffic that’s not in benign traffic
  - Attacker’s IP address = means of *identifying* misbehaving user

Filtering Sounds Pretty Easy …

- … but DoS filters can be easily evaded:
  - Make traffic appear as though it's from many hosts
    - Spoof the source address so it can't be used to filter
      - Just pick a random 32-bit number of each packet sent
    - How does a defender filter this?
      - They don't!
      - Best they can hope for is that operators around the world implement anti-spoofing mechanisms (today about 75% do)
  - Use many hosts to send traffic rather than just one
    - Distributed Denial-of-Service = **DDoS** ("dee-doss")
    - Requires defender to install complex filters
    - How many hosts is "enough" for the attacker?
      - Today they are very cheap to acquire … :-(

It’s Not A “Level Playing Field”

- When defending resources from exhaustion, need to beware of asymmetries, where attackers can consume victim resources with little comparable effort
  - Makes DoS easier to launch
  - Defense costs much more than attack
- Particularly dangerous form of asymmetry: **amplification**
  - Attacker leverages system’s own structure to pump up the load they induce on a resource

Amplification: Network DoS

- One technique for magnifying flood traffic: leverage Internet's *broadcast functionality*

Amplification: Network DoS

- One technique for magnifying flood traffic: leverage Internet's *broadcast functionality*
- How does an attacker exploit this?
  - Send traffic to the broadcast address and spoof it *as though the DoS victim sent it*
  - All of the replies then go to the victim rather than the attacker's machine
  - Each attacker pkt yields dozens of flooding pkts

smurf attack

- Note, this particular threat has been fixed
  - By *changing the Internet standard* to state routers shouldn't forward pkts addressed to broadcast addrs
  - Thus, attacker's spoofs won't make it to target subnet

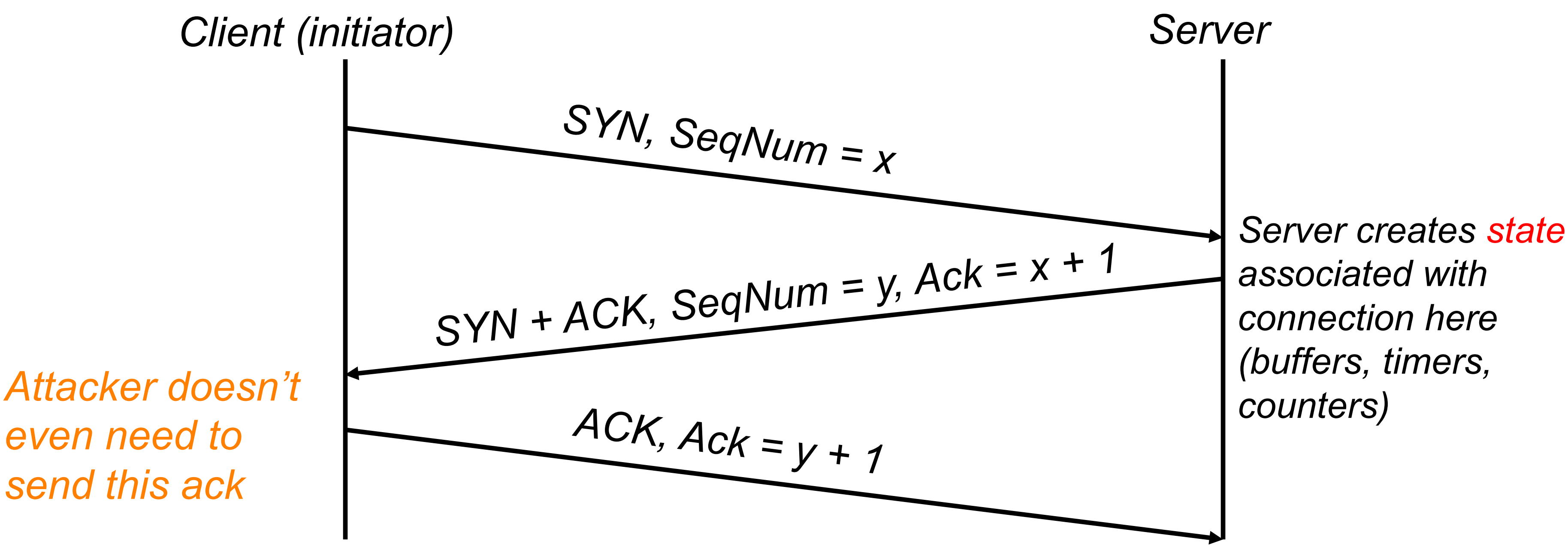
Amplification

- Example of amplification: DNS lookups
  - *Reply is generally much bigger than request*
    - Since it includes a copy of the reply, plus answers etc.

  ⇒ Attacker spoofs DNS request to a patsy DNS server, seemingly from the target
    - Small attacker packet yields large flooding packet
    - Doesn't increase # of packets, but **total volume**
- Note #1: these examples involve blind spoofing
  - So for network-layer flooding, generally only works for UDP-based protocols (can't establish TCP conn.)
- Note #2: victim doesn't see spoofed source addresses
  - Addresses are those of actual intermediary systems

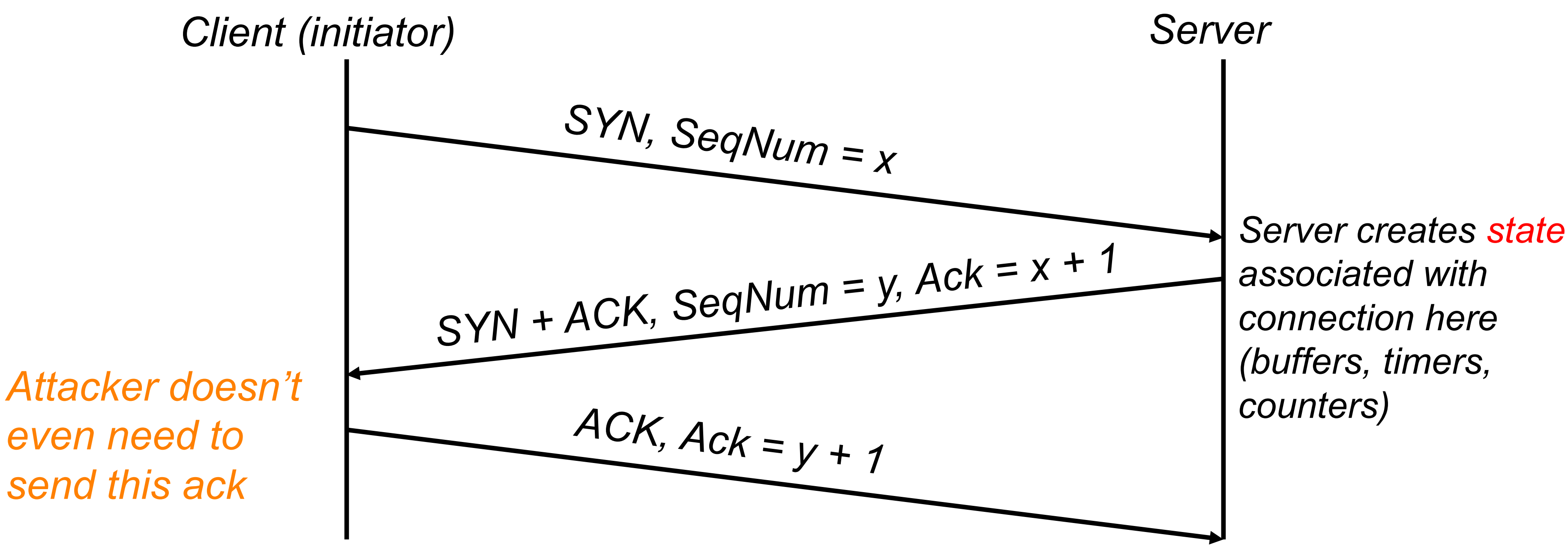
Transport-Level Denial-of-Service

- Recall TCP's 3-way connection establishment handshake
  - Goal: agree on initial sequence numbers



Transport-Level Denial-of-Service

- Recall TCP's 3-way connection establishment handshake
  - Goal: agree on initial sequence numbers
- So a single SYN from an attacker suffices to force the server to *spend some memory*



TCP *SYN Flooding*

- Attacker targets *memory* rather than network capacity
- Every (unique) SYN that the attacker sends burdens the target
- What should target do when it has no more memory for a new connection?
- No good answer!
  - *Refuse* new connection?
    - Legit new users can't access service
  - *Evict* old connections to make room?
    - Legit old users get kicked off

TCP SYN Flooding Defenses

- How can the target defend itself?

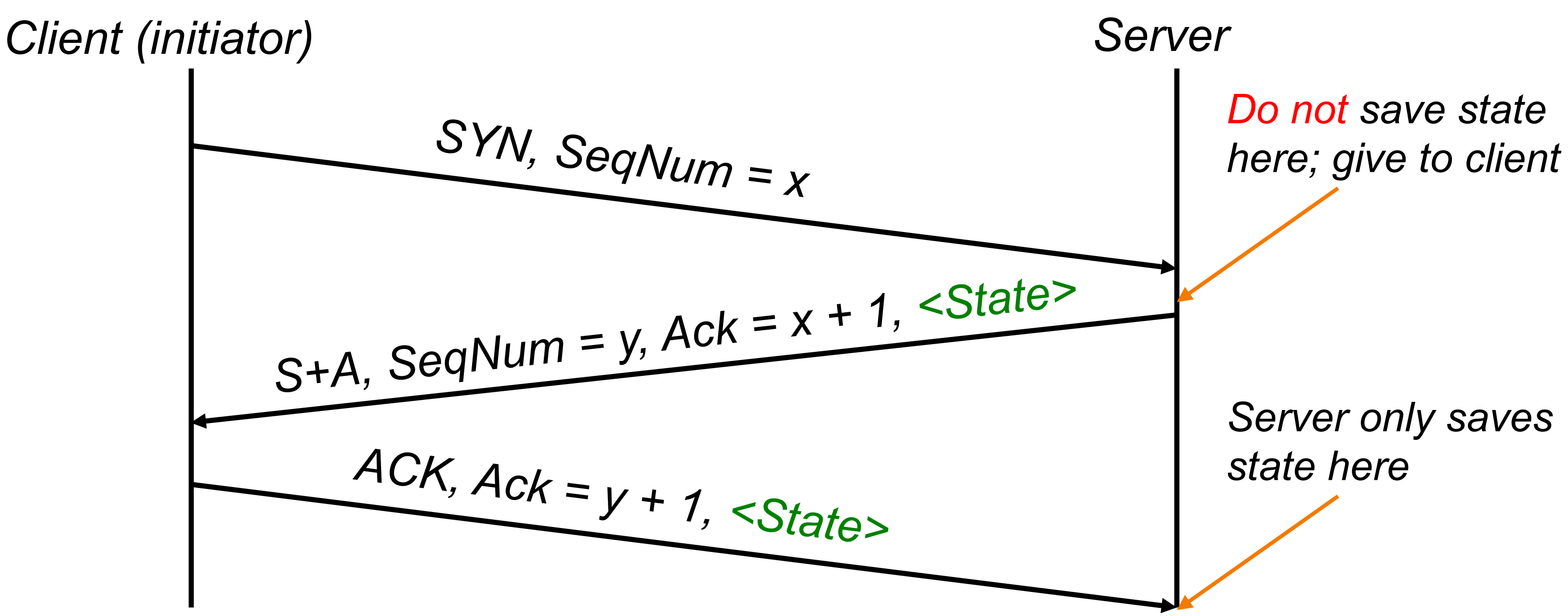
- Approach #1: make sure they have **tons of memory!**
  - How much is enough?
  - Depends on resources attacker can bring to bear (threat model), which might be hard to know

TCP SYN Flooding Defenses

- Approach #2: identify bad actors & refuse their connections
  - Hard because only way to identify them is based on IP address
    - We can't for example require them to send a password because doing so requires we have an established connection!
  - For a public Internet service, who knows which addresses customers might come from?
  - Plus: attacker can spoof addresses since they don't need to complete TCP 3-way handshake
- Approach #3: don't keep state! ("*SYN cookies*"; *only works for spoofed SYN flooding*)

SYN Flooding Defense: *Idealized*

- Server: when SYN arrives, rather than keeping state locally, *send it to the client* …
- Client needs to *return the state* in order to established connection



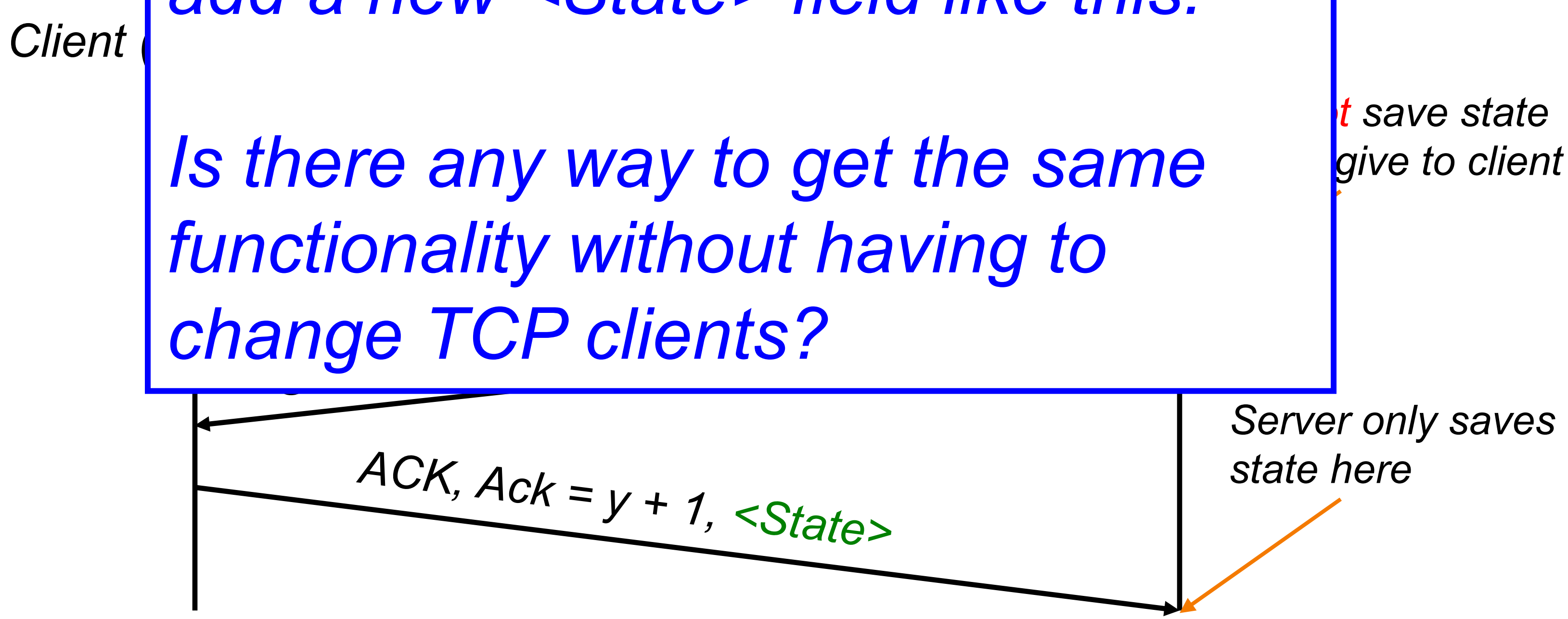
SYN Flooding Defense: *Idealized*

- Server: when SYN arrives, rather than keeping state locally, *send it to the client* …
- Client
  establ

Problem: the world isn't so ideal!

TCP doesn't include an easy way to add a new <State> field like this.

Is there any way to get the same functionality without having to change TCP clients?

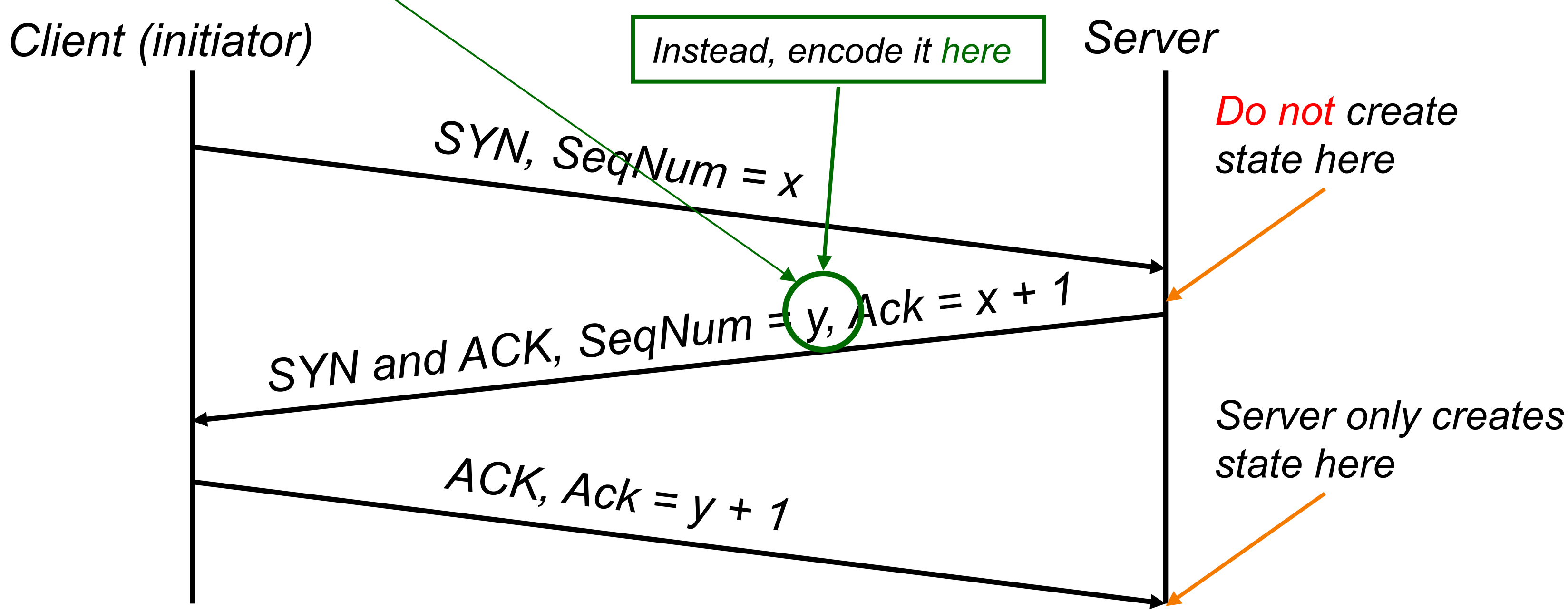


Practical Defense: *SYN Cookies*

- Server: when SYN arrives, encode connection state entirely within SYN-ACK's sequence # y
  - y = *encoding* of necessary state, using server secret

y = T (lower bits of timestamp), lower bits of HMAC(key, T, source port & IP, destination port & IP)]

- When ACK of SYN-ACK arrives, server only creates state ***if*** value of y from it agrees w/ secret



SYN Cookies: Discussion

- Illustrates general strategy: rather than *holding* state, *encode* it so that it is returned when needed
- For SYN cookies, attacker must complete 3-way handshake in order to burden server
  - *Can't use spoofed source addresses*
- Note #1: strategy requires that you have enough bits to encode all the state
  - (This is just barely the case for SYN cookies)
- Note #2: if it's expensive to generate *or check* the cookie, then it's not a win

Application-Layer DoS

- Rather than exhausting network or memory resources, attacker can overwhelm a service's processing capacity
- There are many ways to do so, often at little expense to attacker compared to target (*asymmetry*)

The link sends a request to the web server that requires heavy processing by its "backend database".

Algorithmic complexity attacks

- Attacker can try to trigger worst-case complexity of algorithms / data structures
- Example: You have a hash table.
  Expected time: O(1). Worst-case: O(n).
- Attacker picks inputs that cause table collisions.
  Time per lookup: O(n).
  Total time to do n operations: O(n^2).
- Solution? Use algorithms with good worst-case running time.
  - E.g., universal hash function guarantees that $Pr[h_k(x)=h_k(y)] = 1/2^b$, so hash collisions will be rare.

Application-Layer DoS

- Rather than exhausting network or memory resources, attacker can overwhelm a service's processing capacity
- There are many ways to do so, often at little expense to attacker compared to target (asymmetry)
- Defenses against such attacks?
- Approach #1: Only let legit users issue expensive requests
  - Relies on being able to identify/authenticate them
  - Note: that *this itself might be expensive*!
- Approach #2: Force legit users to "burn" cash
- Approach #3: massive over-provisioning ($$$)

DoS Defense in General Terms

- Defending against program flaws requires:
  - Careful design and coding/testing/review
  - Consideration of behavior of defense mechanisms
    - E.g. buffer overflow detector that when triggered halts execution to prevent code injection $\Rightarrow$ denial-of-service
- Defending resources from exhaustion can be **really** hard. Requires:
  - *Isolation and scheduling mechanisms*
    - Keep adversary's consumption from affecting others
  - *Reliable identification* of different users