Game Plan

• Reminder: Homework #1 due tomorrow night, 9:59PM

• Goal for today: more network attacks
  – (Clarifications regarding TCP attacks)
  – **DHCP**: protocol for bootstrapping Internet access
  – **DNS**: protocol for mapping hostnames to IP addresses
  – **TCP**: cheating on “fairness” (time permitting)
Blind Spoofing: Attacker’s Viewpoint

Client? (1.2.3.4)

SYN, SeqNum = x

But can’t see this

SYN + ACK, SeqNum = y, Ack = x + 1

ACK, Ack = y + 1

So how do they know what to put here?

Attacker can spoof this

Server (5.6.7.8)

Each host tells its Initial Sequence Number (ISN) to the other host.

(Spec says to pick based on local clock)

Hmm, any way for the attacker to know this?

Sure - make a non-spoofed connection first, and see what server used for ISN y then!

How Do We Fix This?

Use A Random ISN
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
Dynamic Host Configuration Protocol

- DHCP discover (broadcast)
- DHCP offer
- DHCP request (broadcast)
- DHCP ACK

“offer” message includes IP address, DNS server, “gateway router”, and how long client can have these (“lease” time)
Dynamic Host Configuration Protocol

DHCP discover (broadcast)

DHCP offer

DHCP request (broadcast)

DHCP ACK

Threats?

“offer” message includes IP address, DNS server, “gateway router”, and how long client can have these (“lease” time)
Dynamic Host Configuration Protocol

- **DHCP discover** (broadcast)
- **DHCP offer**
- **DHCP request** (broadcast)
- **DHCP ACK**

**new client**

Attacker on same subnet can **hear** new host’s DHCP request

**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

“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 they win, replace 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

• Substitute a fake “gateway”
  – Intercept all of a host’s off-subnet traffic
    o (even if not preceded by a DNS lookup)
  – Relay contents back and forth between host and remote server
    o Modify however attacker chooses

• An invisible Man In The Middle (MITM)
  – Victim host has no way of knowing it’s happening
    o (Can’t necessarily alarm on peculiarity of receiving multiple DHCP replies, since that can happen benignly)

• How can we fix this? Hard
Host at 

requests IP address for 

Local DNS server (resolver) 

dns.poly.edu 

Root DNS server ('.') 

TLD DNS server ('.edu') 

Authoritative DNS server ('umass.edu', 'cs.umass.edu') 

dns.cs.umass.edu 

Caching heavily used to minimize lookups
**DNS Protocol**

**DNS protocol**: *query* and *reply* messages, both with *same message format*

(Mainly uses UDP transport rather than TCP)

**Message header:**

- **Identification**: 16 bit # for query, reply to query uses same #
- Replies can include “Authority” (name server responsible for answer) and “Additional” (info client is likely to look up soon anyway)
- Replies have a **Time To Live** (in seconds) for **caching**

<table>
<thead>
<tr>
<th>16 bits</th>
<th>16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>Flags</td>
</tr>
<tr>
<td># Questions</td>
<td># Answer RRs</td>
</tr>
<tr>
<td># Authority RRs</td>
<td># Additional RRs</td>
</tr>
<tr>
<td>Questions</td>
<td>(variable # of resource records)</td>
</tr>
<tr>
<td>Answers</td>
<td>(variable # of resource records)</td>
</tr>
<tr>
<td>Authority</td>
<td>(variable # of resource records)</td>
</tr>
<tr>
<td>Additional information</td>
<td>(variable # of resource records)</td>
</tr>
</tbody>
</table>
DNS Threats

• DNS: path-critical for just about everything we do
  – Maps hostnames ⇔ IP addresses
  – Design only **scales** if we can minimize lookup traffic
    o #1 way to do so: **caching**
    o #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 protocols function

• Directly interacting w/ DNS: **dig** program on Unix
  – Allows querying of DNS system
  – Dumps each field in DNS responses
Use Unix “dig” utility to look up DNS address (“A”) for hostname eecs.mit.edu

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
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

These are just comments from dig itself with details of the request/response
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
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

Here the server echoes back the question that it is answering.

Typically, mit.edu runs the DNS 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.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 its address 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.

; ; 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 record gives the hostname of a different name server (“NS”) for names in mit.edu. We should cache each record for 11,088 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.

;; 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

"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.
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

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. 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 happens if the mit.edu server returns the following to us instead?
We 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.)
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. 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

How do we fix such cache poisoning?
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

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 an attacker even know we are looking up mail.google.com?

<img src="http://mail.google.com" ...>
DNS Blind Spoofing, con’t

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" ...>  ➞ They observe ID k here
<img src="http://mail.google.com" ...>  ➞ So this will be k+1
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:

```html
<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 returns 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 returns 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
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.)
DNS (primarily) uses UDP for transport rather than TCP.

UDP header has:
- 16-bit Source & Destination ports (identify processes, like w/ TCP)
- 16-bit checksum, 16-bit length

<table>
<thead>
<tr>
<th>16 bits</th>
<th>16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC port</td>
<td>DST port</td>
</tr>
<tr>
<td>checksum</td>
<td>length</td>
</tr>
<tr>
<td>Identification</td>
<td>Flags</td>
</tr>
<tr>
<td># Questions</td>
<td># Answer RRs</td>
</tr>
<tr>
<td># Authority RRs</td>
<td># Additional RRs</td>
</tr>
</tbody>
</table>

- 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

DNS (primarily) uses UDP for transport rather than TCP.

UDP header has:
- 16-bit Source & Destination ports (identify processes, like w/ TCP)
- 16-bit checksum, 16-bit length

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

<table>
<thead>
<tr>
<th>16 bits</th>
<th>16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src=53</td>
<td>Dest=53</td>
</tr>
<tr>
<td>checksum</td>
<td>length</td>
</tr>
<tr>
<td>Identification</td>
<td>Flags</td>
</tr>
<tr>
<td># Questions</td>
<td># Answer RRs</td>
</tr>
<tr>
<td># Authority RRs</td>
<td># Additional RRs</td>
</tr>
</tbody>
</table>

Total entropy: 16 bits

<table>
<thead>
<tr>
<th>Questions</th>
<th>(variable # of resource records)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answers</td>
<td>(variable # of resource records)</td>
</tr>
<tr>
<td>Authority</td>
<td>(variable # of resource records)</td>
</tr>
<tr>
<td>Additional information</td>
<td>(variable # of resource records)</td>
</tr>
</tbody>
</table>
Defending Against Blind Spoofing

“Fix”: use random source port

<table>
<thead>
<tr>
<th></th>
<th>16 bits</th>
<th>16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src=rnd</td>
<td></td>
<td>Dest=53</td>
</tr>
<tr>
<td>checksum</td>
<td></td>
<td>length</td>
</tr>
<tr>
<td>Identification</td>
<td></td>
<td>Flags</td>
</tr>
<tr>
<td># Questions</td>
<td></td>
<td># Answer RRs</td>
</tr>
<tr>
<td># Authority RRs</td>
<td></td>
<td># Additional RRs</td>
</tr>
</tbody>
</table>

- Questions (variable # of resource records)
- Answers (variable # of resource records)
- Authority (variable # of resource records)
- Additional information (variable # of resource records)

Total entropy: ? bits
Defending Against Blind Spoofing

“Fix”: use random source port

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 today. (Note: not all resolvers have implemented random source ports!)
Summary of DHCP Security Issues

• DHCP threats highlight:
  – Broadcast protocols inherently at risk of attacker spoofing
    o Attacker knows exactly when to try it …
    o … 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
Summary of DNS Security Issues

- DNS threats highlight:
  - Attackers can attack opportunistically rather than eavesdropping
    - Cache poisoning only requires victim to look up some name under attacker’s control
  - 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