Network Attacks / Control

CS 161 - Computer Security

Profs. Vern Paxson & David Wagner

TAs: John Bethencourt, Erika Chin, Matthew Finifter, Cynthia Sturton, Joel Weinberger

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

Feb 17, 2010

Focus of Today's Lecture

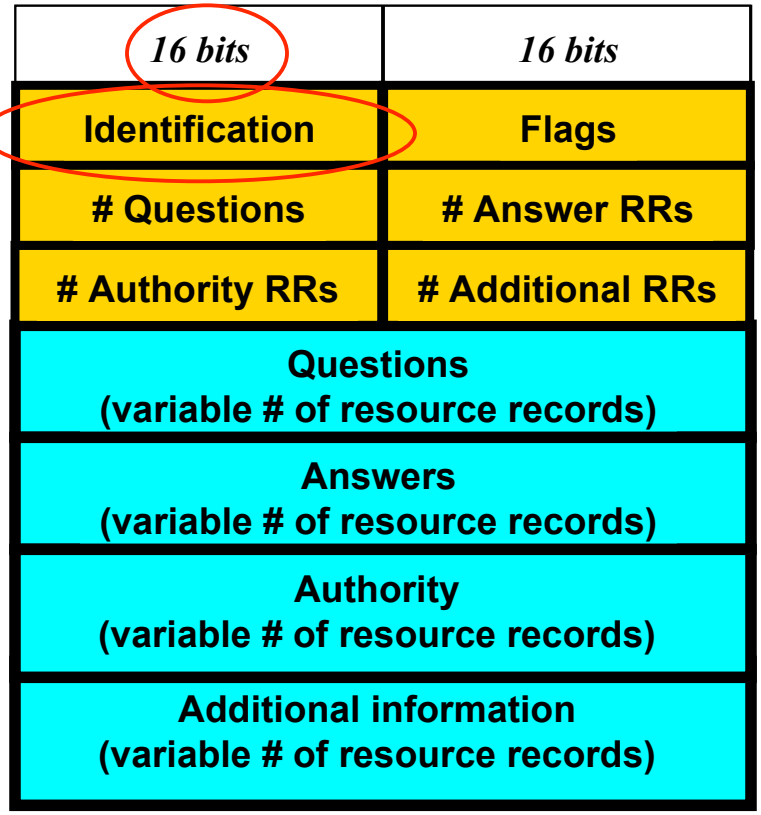
- Finish discussion of DNS attacks
- Begin discussion of approaches for controlling network traffic:
  - Firewalls: restricting allowed communication
  - NATs: Network Address Translators

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 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
```

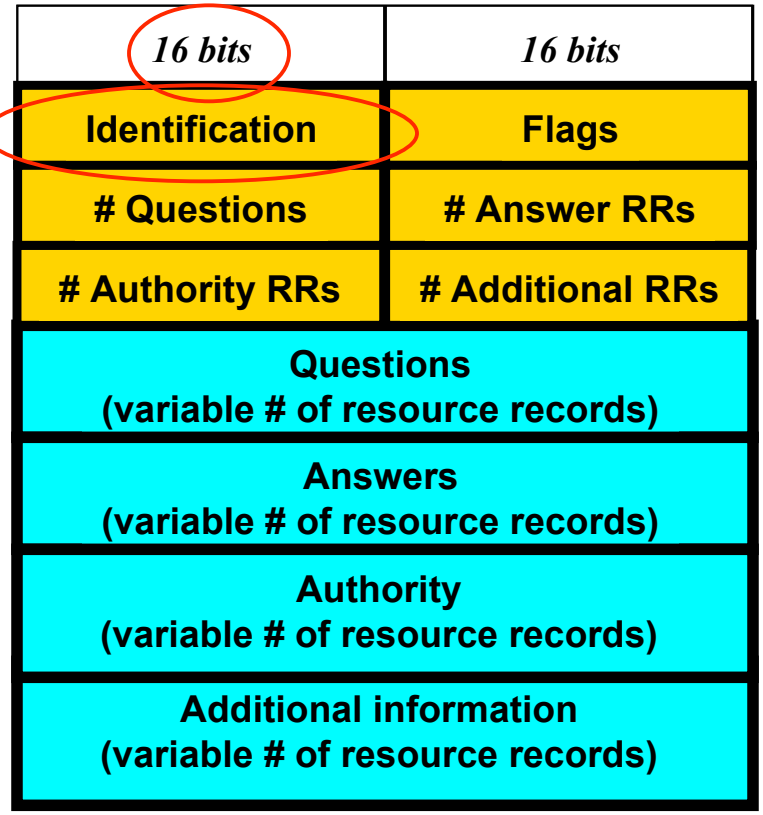
Once they win the race, not only have they poisoned `mail.google.com` … but also the cached `NS` record for `google.com`'s name server - so any **future** *X*`.google.com` lookups *go through the attacker's machine*

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.)



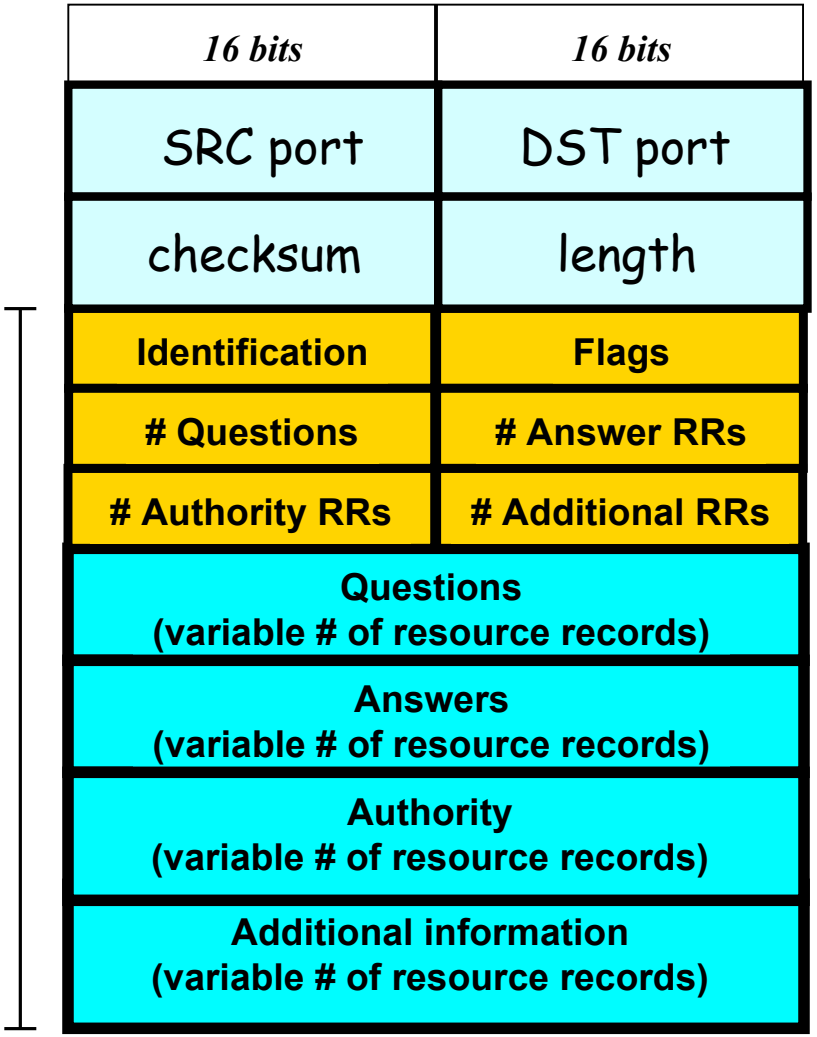
Defending Against Blind Spoofing

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

UDP Payload



Defending Against Blind Spoofing

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

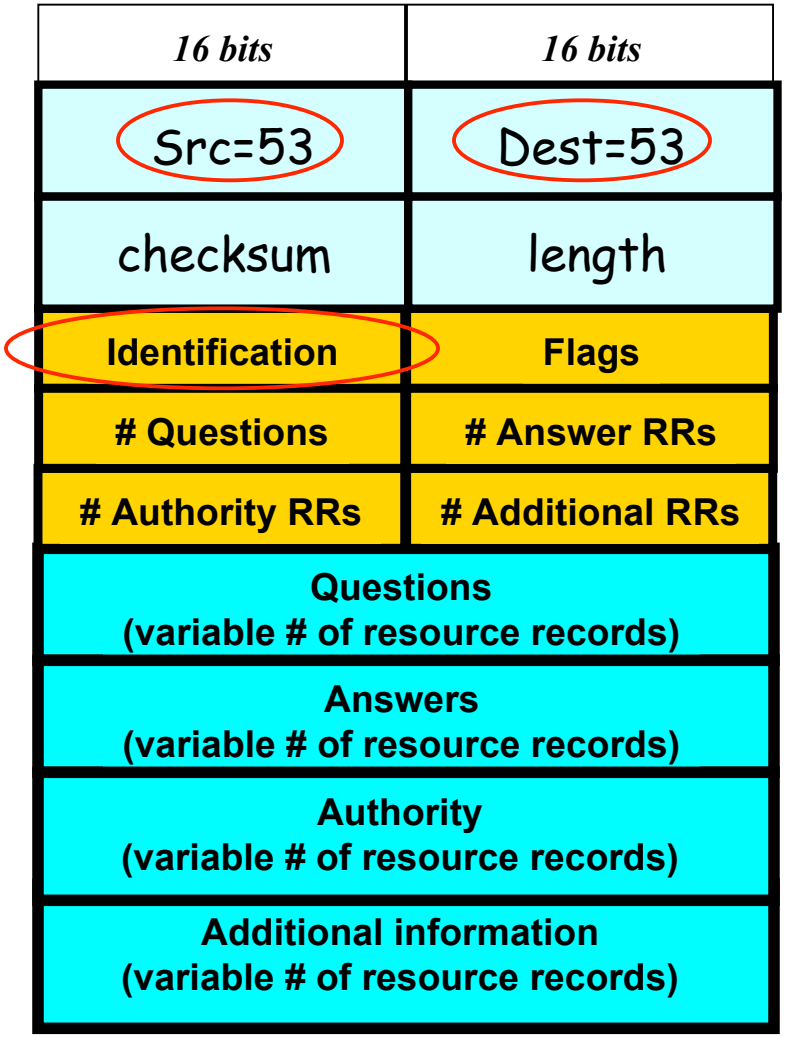
UDP header has:

- 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

Total *entropy*: 16 bits

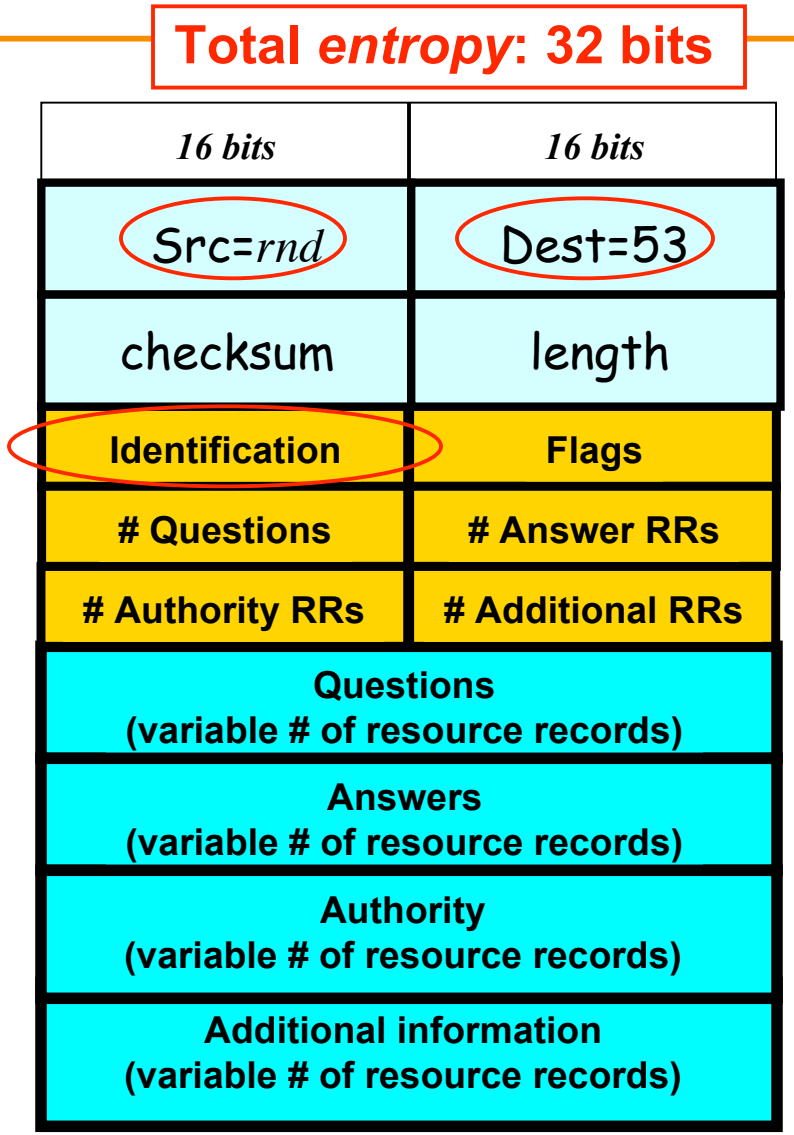


Defending Against Blind Spoofing

"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/DNS Security Issues

- DHCP threats highlight:
  - Broadcast protocols inherently at risk of attacker spoofing
    - Attacker knows exactly when to try it
  - 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 DHCP/DNS Security Issues

- DHCP threats highlight:
  - Broadcast protocols inherently at risk of attacker spoofing
    - Attacker knows exactly when to try it
  - 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
- 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

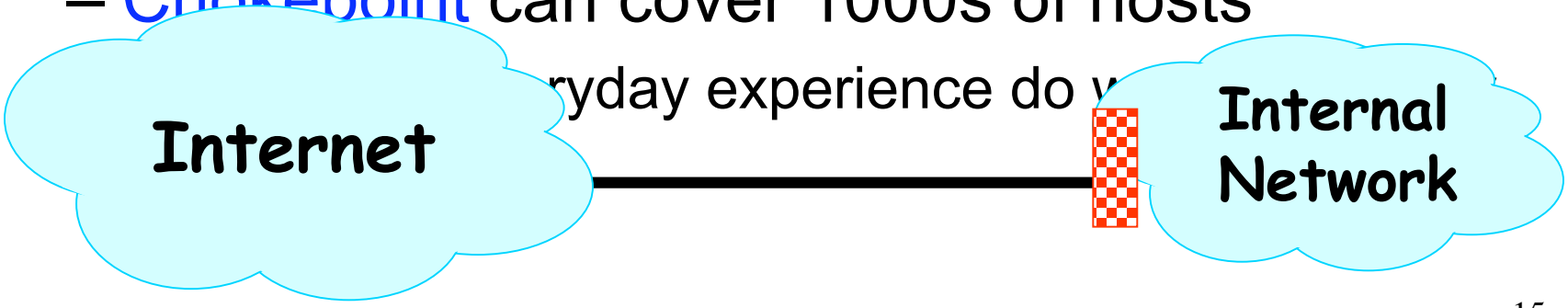
Questions?

Network Control: Firewalls

- Motivation: How do you harden a set of systems against external attack?
  - *Key Observation:*
    - *The more network services your machines run, the greater the risk*
  - Due to larger attack surface
- One approach: on each system, turn off unnecessary network services
  - But you have to know that it's running them
  - And sometimes some trusted remote users still require access
- Plus key question of scaling
  - What happens when you have to secure 100s/1000s of systems?
  - Which may have different OSs, hardware & users
  - Which may in fact not all even be identified

Taming Management Complexity

- Possibly more scalable defense: Reduce risk by blocking *in the network* outsiders from having unwanted access your network services
  - Interpose a **firewall** the traffic to/from the outside must traverse
  - Chokepoint can cover 1000s of hosts



Selecting a Security Policy

- Effectiveness of firewall relies on deciding what policy it should implement:
  - Who is allowed to talk to whom, accessing what service?
- Distinguish between inbound & outbound conns
  - Inbound: attempts by external users to connect to services on internal machines
  - Outbound: internal users to external services
- Conceptually simple *access control policy*:
  - Permit inside users to connect to any service
  - External users restricted:
    - Permit connections to services meant to be externally visible
    - Deny connections to services not meant for external access

How To Treat Traffic Not Mentioned in Policy?

- **Default Allow**: start off permitting external access to services
  - Shut them off as problems recognized
- **Default Deny**: start off permitting just a few known, well-secured services
  - Add more when users complain (and mgt. approves)
- Pros & Cons?
  - Flexibility vs. conservative design
  - Flaws in Default Deny get noticed more quickly / less painfully
- (Which do you think UCB uses?)
  - Default Allow: institute's **mission** thrives on flexibility

In general, use Default Deny

Packet Filters

- Most basic kind of firewall is a *packet filter*
  - Router with list of access control rules
  - Router checks each received packet against security rules to decide to forward or drop it
  - Each rule specifies which packets it applies to based on a packet's header fields
    - Specify source and destination IP addresses, port numbers, and protocol names, or wild cards
    - Each rule specifies the *action* for matching packets: **ALLOW** or **DROP**

    <ACTION> <PROTO> <SRC:PORT> -> <DEST:PORT>
  - First listed rule has precedence

Examples of Packet Filter Rules

```
allow tcp 4.5.5.4:1025 -> 3.1.1.2:80
```

- States that the firewall should **permit** any TCP packet that's:
  - from Internet address 4.5.5.4 **and**
  - using a source port of 1025 **and**
  - destined to port 80 of Internet address 3.1.1.2

```
deny tcp 4.5.5.4:* -> 3.1.1.2:80
```

- States that the firewall should **drop** any TCP packet like the above, regardless of source port

```
deny tcp 4.5.5.4:* -> 3.1.1.2:80
allow tcp 4.5.5.4:1025 -> 3.1.1.2:80
```

- *In this order*, the rules won't allow *any* TCP packets from 4.5.5.4 to port 80 of 3.1.1.2

```
allow tcp 4.5.5.4:1025 -> 3.1.1.2:80
deny tcp 4.5.5.4:* -> 3.1.1.2:80
```

- *In this order*, the rules allow *only* TCP packets from 4.5.5.4 to port 80 of 3.1.1.2 if they come from source port 1025

Expressing Policy with *Rulesets*

- Goal: prevent external access to Windows SMB (TCP port 445)
  - Except for one special external host, 8.4.4.1
- Ruleset:
  - `allow tcp 8.4.4.1:* -> *:445`
  - `drop  tcp *:* -> *:445`
  - `allow  *  *:* -> *:*`
- Problems?
  - No notion of inbound vs outbound connections
    - Drops outbound SMB connections from inside users
  - This is a default-allow policy!!

Expressing Policy with Rulesets, con't

- Want to allow:
  - Inbound mail connections to our mail server (`1.2.3.4:25`)
  - All outbound connections from our network, `1.2.3.0/24`
    - `1.2.3/24` = "any address for which the top 24 bits match `1.2.3.0`"
    - So it ranges from `1.2.3.0`, `1.2.3.1`, …, `1.2.3.255`
  - Nothing else
- Consider this ruleset:

```
allow tcp *:* -> 1.2.3.4:25
allow tcp 1.2.3.0/24:* -> *:*
drop   *  *:* -> *:*
```

- This policy doesn't work …
  - TCP connections are bidirectional
  - 3-way handshake: send `SYN`, receive `SYN+ACK`, send `ACK`, send DATA w/ `ACK` bit set

Problem: Outbound Connections Fail

```
1.allow tcp *:* -> 1.2.3.4:25
2.allow tcp 1.2.3.0/24:* -> *:*
3.drop   *  *:* -> *:*
```

- Inside host opens TCP connection to port 80 on external machine:
  - Initial SYN packet passed through by rule 2
  - SYN+ACK packet coming back is dropped
    - Fails rule 1 (not destined for port 25)
    - Fails rule 2 (source not inside host)
    - Matches rule 3 ⇒ DROP
- Fix?
  - In general, we need to distinguish between 2 kinds of inbound pkts
    - Allow inbound packets associated with an outbound connection
    - Restrict inbound packets associated with an inbound connection
  - How do we tell them apart?
    - Approach #1: remember previous outbound connections (takes **state**)
    - Approach #2: leverage details of how TCP works