The Net
Part 3
Announcements

• Reminder: Project 2 design doc draft due Tomorrow

• In-person Nick's option Wednesday the 4th
  • Sign-ups and time TBD later this week

• VOTE!
  • If you haven't voted already, vote *in person* or use a *drop box*
Broadcast Protocols
Make The Local Network Worse...

• By default, both DHCP and ARP broadcast requests
  • Sent to all systems on the local area network

• DHCP: Dynamic Host Control Protocol
  • Used to configure all the important network information
    • Including the DNS server:
      If the attacker controls the DNS server they have complete ability to intercept all traffic!
    • Including the Gateway which is where on the LAN a computer sends to:
      If the attacker controls the gateway

• ARP: Address Resolution Protocol
  • "Hey world, what is the Ethernet MAC address of IP X"
  • Used to find both the Gateway's MAC address and other systems on the LAN
2. Configure your connection

Your laptop shouts:

*HEY, ANYBODY, WHAT BASIC CONFIG DO I NEED TO USE?*
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
  - (Note, host does have a separate WiFi MAC address)

- 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

new client

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

DHCP discover (broadcast)

new client

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

Threats?

new client

DHCP discover (broadcast)

DHCP server

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

DHCP offer

DHCP request (broadcast)

DHCP ACK
Dynamic Host Configuration Protocol

A new client sends a DHCP discover message (broadcast) to a DHCP server on the same subnet. The DHCP server responds with a DHCP offer message, specifying the IP address, DNS server, "gateway router", and how long the client can have these ("lease" time) in an offer message. A local attacker on the same subnet can hear the new host's DHCP request and respond with a DHCP ACK message.
Dynamic Host Configuration Protocol

This happens even for WPA2-Enterprise, since request is explicitly sent using broadcast.

“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
  - 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, because we lack a trust anchor
DHCP Conclusion

• 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 and convenience*
  • MITM attacks insidious because no indicators they’re occurring
So How Do We Secure the LAN?

• Option 1: We don't
  • Just assume we can keep bad people out
  • This is how most people run their networks: "Hard on the outside with a goey chewy caramel center"

• Option 2: *smart* switching and active monitoring
The Switch

• Hubs are very inefficient:
  • By broadcasting traffic to all recipients this greatly limits the aggregate network bandwidth

• Instead, most Ethernet uses switches
  • The switch keeps track of which MAC address is seen where

• When a packet comes in:
  • If it is to the broadcast address, send it to all ports
  • If there is no entry in the MAC cache for the destination, broadcast it to all ports
  • If there is an entry, send it just to that port

• Result is vastly improved bandwidth
  • All ports can send or receive at the same time
Smarter Switches: Clean Up the Broadcast Domain

- Modern high-end switches can do even more
  - A large amount of potential packet processing on items of interest
- Basic idea: constrain the broadcast domain
  - Either filter requests so they only go to specific ports
    - Limits other systems from listening
  - Or filter replies
    - Limits other systems from replying
- Locking down the LAN is very important practical security
  - This is real defense in depth:
    - Don't want 'root on random box, pwn whole network'
  - This removes pivots the attacker can try to extend a small foothold into complete network ownership
  - This is why an Enterprise switch may cost $1000s yet provide no more real bandwidth than a $100 Linksys.
Smarter Switches: Virtual Local Area Networks (VLANs)

- Our big expensive switch can connect a lot of things together
  - But really, many are in **different** trust domains:
    - Guest wireless
    - Employee wireless
    - Production desktops
    - File Servers
    - etc...
- Want to isolate the different networks from each other
  - Without actually buying separate switches
VLANs

• An ethernet port can exist in one of two modes:
  • Either on a single VLAN
  • On a trunk containing multiple specified VLANs

• All network traffic in a given VLAN stays only within that VLAN
  • The switch makes sure that this occurs

• When moving to/from a trunk the VLAN tag is added or removed
  • But still enforces that a given trunk can only read/write to specific VLANs
Putting It Together: If I Was In Charge of UC networking...

- I'd isolate networks into 3+ distinct classes
  - The plague pits (AirBears, Dorms, etc)
  - The mildly infected pits (Research)
  - Administration

- Administration would be locked down
  - Separate VLANs
  - Restricted DHCP/system access
  - Isolated from the rest of campus
Addressing on the Layers
On The Internet

• **Ethernet:**
  • Address is 6B MAC address, Identifies a machine on the local LAN

• **IP:**
  • Address is a 4B (IPv4) or 16B (IPv6) address, Identifies a system on the Internet

• **TCP/UDP:**
  • Address is a 2B port number, Identifies a particular listening server/process/activity on the system
    • Both the client and server have to have a port associated with the communication
  • Ports 0-1024 are for privileged services
    • Must be root to accept incoming connections on these ports
    • Any thing can do an outbound request to such a port
  • Port 1025+ are for anybody
    • And high ports are often used ephemerally
UDP: Datagrams on the Internet

• UDP is a protocol built on the Internet Protocol (IP)
• It is an "unreliable, datagram protocol"
  • Messages may or may not be delivered, in any order
  • Messages can be larger than a single packet (but probably shouldn't)
    • IP will fragment these into multiple packets (mostly... Single digit %-age of hosts can't receive fragmented traffic)
• Programs create a socket to send and receive messages
  • Just create a datagram socket for an ephemeral port
  • Bind the socket to a particular port to receive traffic on a specified port
  • Basic recipe for Python:
    https://wiki.python.org/moin/UdpCommunication
DNS Overview

- DNS translates www.google.com to 74.125.25.99
  - Turns a human abstraction into an IP address
  - Can also contain other data
- It’s a performance-critical distributed database.
- DNS security is critical for the web.
  (Same-origin policy assumes DNS is secure.)
  - 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).
  - Based on a notion of hierarchical trust:
    - You trust . for everything, com. for any com, google.com. for everything google...
DNS Lookups via a **Resolver**

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

- **requesting host** `xyz.poly.edu`
- **root DNS server** (`.`)
- **local DNS server** (resolver) `dns.poly.edu`
- **TLD DNS server** (`.edu`)
- **authoritative DNS server** (for `mit.edu`) `dns.mit.edu`
- **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.
- (In fact, they used to be able to fool us about the answer to other queries, too. We’ll come back to that.)
Security risk #2: on-path eavesdropper

- 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?
- 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
      - The "glue records"

- What if attacker eavesdrops on our DNS queries?
  - Then similar to DHCP, ARP, AirPwn etc, 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
Use Unix “dig” utility to look up IP address (“A”) for hostname eecs.mit.edu via DNS

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:
mits.edu. 11088 IN NS BITSY.mits.edu.
mits.edu. 11088 IN NS W20NS.mits.edu.
mits.edu. 11088 IN NS STRAWB.mits.edu.

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

The question we asked the server
dig eecs.mit.edu A

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

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

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

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

"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.                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

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`

```plaintext
; ; <<>> 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
```

“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.       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.
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.

Servers are on port 53 always.

Frequently, clients used to use port 53 but can use any port.

<table>
<thead>
<tr>
<th>SRC port</th>
<th>DST port</th>
</tr>
</thead>
</table>

**UDP Header**
- Checksum
- Length

**UDP Payload**
- Identification
- Flags
- # Questions
- # Answer RRs
- # Authority RRs
- Additional RRs
- Questions (variable # of resource records)
- Answers (variable # of resource records)
- Additional information (variable # of resource records)
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)

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC</td>
<td>16 bits</td>
</tr>
<tr>
<td>DST</td>
<td>16 bits</td>
</tr>
<tr>
<td>checksum</td>
<td>16 bits</td>
</tr>
<tr>
<td>length</td>
<td>16 bits</td>
</tr>
<tr>
<td>Identification</td>
<td>16 bits</td>
</tr>
<tr>
<td>Flags</td>
<td>16 bits</td>
</tr>
<tr>
<td># Questions</td>
<td>variable #</td>
</tr>
<tr>
<td># Answer RRs</td>
<td>variable #</td>
</tr>
<tr>
<td># Authority RRs</td>
<td>variable #</td>
</tr>
<tr>
<td># Additional RRs</td>
<td>variable #</td>
</tr>
<tr>
<td>Questions</td>
<td>variable #</td>
</tr>
<tr>
<td>Answers</td>
<td>variable #</td>
</tr>
<tr>
<td>Authority</td>
<td>variable #</td>
</tr>
<tr>
<td>Additional information</td>
<td>variable #</td>
</tr>
</tbody>
</table>
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 web server?
Let’s look at a flaw in the original DNS design (since fixed)
What could happen if the mit.edu server returns the following to us instead?
`dig eecs.mit.edu A`

`;; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a`
`;; global options: +cmd`
`;; Got answer:`
`;; ->>>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901`
`;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3`
`;; QUESTION SECTION:`
`;eecs.mit.edu.                  IN      A`
`;; ANSWER SECTION:`
`eecs.mit.edu.           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      www.berkeley.edu.`
`;; ADDITIONAL SECTION:`
`www.berkeley.edu.       100000   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

In this case they chose to make the mapping last a long time. They could just as easily make it for just a couple of 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. 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
; ; 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      www.berkeley.edu.
; ; ADDITIONAL SECTION:
www.berkeley.edu.       100000  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.

This is called "Bailiwick checking"
DNS Resource Records and RRSETs

- DNS records (Resource Records) can be one of various types
  - Name TYPE Value
  - Also a “time to live” field: how long in seconds this entry can be cached for
- Addressing:
  - A: IPv4 addresses
  - AAAA: IPv6 addresses
  - CNAME: aliases, “Name X should be name Y”
  - MX: “the mailserver for this name is Y”
- DNS related:
  - NS: “The authority server you should contact is named Y”
  - SOA: “The operator of this domain is Y”
- Other:
  - text records, cryptographic information, etc….
- Groups of records of the same type form RRSETs:
  - E.g. all the nameservers for a given domain.
## The Many Moving Pieces

In a DNS Lookup of **www.isc.org**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

User’s ISP’s: ![Recursive Resolver](image.png)

Authority Server (the “root”)

Answers:

- Authority: org. NS a0.afilias-nst.info
- Additional: a0.afilias-nst.info A 199.19.56.1
The Many Moving Pieces
In a DNS Lookup of www.isc.org

User’s ISP’s  Recursive Resolver

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.</td>
<td>NS</td>
<td>a0.afilias-nst.info</td>
<td>172800</td>
</tr>
<tr>
<td>a0.afilias-nst.info.</td>
<td>A</td>
<td>199.19.56.1</td>
<td>172800</td>
</tr>
</tbody>
</table>

Authority Server

? A www.isc.org
Answers:
Authority:
isc.org. NS ns.isc.afilias-nst.info.
Additional:
sfba.sns-pb.isc.org.  A 199.6.1.30
ns.isc.afilias-nst.info. A 199.254.63.254
The Many Moving Pieces
In a DNS Lookup of www.isc.org

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.</td>
<td>NS</td>
<td>a0.afilias-nst.info</td>
<td>172800</td>
</tr>
<tr>
<td>a0.afilias-nst.info.</td>
<td>A</td>
<td>199.19.56.1</td>
<td>172800</td>
</tr>
<tr>
<td>isc.org.</td>
<td>NS</td>
<td>sfba.sns-pb.isc.org.</td>
<td>86400</td>
</tr>
<tr>
<td>isc.org.</td>
<td>NS</td>
<td>ns.isc.afilias-nst.info</td>
<td>86400</td>
</tr>
<tr>
<td>sfbay.sns-pb.isc.org.</td>
<td>A</td>
<td>199.6.1.30</td>
<td>86400</td>
</tr>
</tbody>
</table>

isc.org.
Authority Server

? A www.isc.org
Answers:
www.isc.org. A 149.20.64.42
Authority:
isc.org. NS ns.isc.afilias-nst.info.
Additional:
sfba.sns-pb.isc.org. A 199.6.1.30
ns.isc.afilias-nst.info. A 199.254.63.254
The Many Moving Pieces
In a DNS Lookup of **www.isc.org**

User’s ISP’s Recursive Resolver

? A **www.isc.org**

**Answers:** **www.isc.org A 149.20.64.42**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.</td>
<td>NS</td>
<td>a0.afilias-nst.info</td>
<td>172800</td>
</tr>
<tr>
<td>a0.afilias-nst.info.</td>
<td>A</td>
<td>199.19.56.1</td>
<td>172800</td>
</tr>
<tr>
<td>isc.org.</td>
<td>NS</td>
<td>sfba.sns-pb.isc.org.</td>
<td>86400</td>
</tr>
<tr>
<td>isc.org.</td>
<td>NS</td>
<td>ns.isc.afilias-net.info.</td>
<td>86400</td>
</tr>
<tr>
<td>sfbay.sns-pb.isc.org.</td>
<td>A</td>
<td>199.6.1.30</td>
<td>86400</td>
</tr>
<tr>
<td><a href="http://www.isc.org">www.isc.org</a></td>
<td>A</td>
<td>149.20.64.42</td>
<td>600</td>
</tr>
</tbody>
</table>
Stepping Through This With **dig**

- Some flags of note:
  - `+norecurse`: Ask directly like a recursive resolver does
  - `+trace`: Act like a recursive resolver without a cache

```
nweaver% dig +norecurse slashdot.org @a.root-servers.net

; <<>> DiG 9.8.3-P1 <<>> +norecurse slashdot.org @a.root-servers.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26444
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 6, ADDITIONAL: 12

;; QUESTION SECTION:
;slashdot.org.                  IN      A

;; AUTHORITY SECTION:
org.                    172800  IN      NS      a0.org.afilias-nst.info.
...                      ...        ...

;; ADDITIONAL SECTION:
a0.org.afilias-nst.info. 172800 IN      A      199.19.56.1
```
So in `dig` parlance

- So if you want to recreate the lookups conducted by the recursive resolver:
  - `dig +norecurse www.isc.org @a.root-servers.net`
  - `dig +norecurse www.isc.org @199.19.56.1`
  - `dig +norecurse www.isc.org @199.6.1.30`
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.
• China does this operationally:
  • dig www.benign.com @www.tsinghua.edu.cn
  • dig www.facebook.com @www.tsinghua.edu.cn
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:

```html
...<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 legitimate server replies?
- How can such an attacker even know we are looking up mail.google.com?

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

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

### DNS Packet Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC</td>
<td>16 bits</td>
</tr>
<tr>
<td>DST</td>
<td>16 bits</td>
</tr>
<tr>
<td>Checksum</td>
<td>16 bits</td>
</tr>
<tr>
<td>Length</td>
<td>16 bits</td>
</tr>
<tr>
<td>Identification</td>
<td># Questions</td>
</tr>
<tr>
<td>Flags</td>
<td># Answer RRs</td>
</tr>
<tr>
<td># Authority RRs</td>
<td># Additional RRs</td>
</tr>
</tbody>
</table>

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.
Blind spoofing

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

How hard is that?

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

<table>
<thead>
<tr>
<th>SRC=53</th>
<th>DST=53</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>Questions (variable # of resource records)</td>
<td></td>
</tr>
<tr>
<td>Answers (variable # of resource records)</td>
<td></td>
</tr>
<tr>
<td>Authority (variable # of resource records)</td>
<td></td>
</tr>
<tr>
<td>Additional information (variable # of resource records)</td>
<td></td>
</tr>
</tbody>
</table>

They observe ID k here
So this will be k+1
DNS Blind Spoofing, cont.

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

Are we 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 inoculated!

<table>
<thead>
<tr>
<th>16 bits</th>
<th>16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC=53</td>
<td>DST=53</td>
</tr>
<tr>
<td>checksum</td>
<td>length</td>
</tr>
</tbody>
</table>

- 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)

Unless attacker can send 1000s of replies before legit arrives, we’re likely safe – phew! ?
Enter Kaminski...

Glue Attacks

- Dan Kaminski noticed something strange, however...
  - Most DNS servers would **cache** the in-bailiwick glue...
  - And then **promote** the glue
  - And will also **update** entries based on glue

- So if you first did this lookup...
  - And then went to query for `a0.org.afilias-nst.info`
  - there would be no other lookup!

```
nweaver% dig +norecurse slashdot.org @a.root-servers.net

; <<>> DiG 9.8.3-P1 <<>> +norecurse slashdot.org @a.root-servers.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26444
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 6, ADDITIONAL: 12

;; QUESTION SECTION:
;slashdot.org.                  IN      A

;; AUTHORITY SECTION:
org.                    172800  IN      NS      a0.org.afilias-nst.info.

;; ADDITIONAL SECTION:
a0.org.afilias-nst.info. 172800 IN      A       199.19.56.1

;; Query time: 128 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Tue Apr 16 09:48:32 2013
;; MSG SIZE  rcvd: 432
```
The Kaminski Attack
In Practice

• Rather than trying to poison www.google.com...

• Instead try to poison a.google.com...
  And state that "www.google.com" is an authority
  And state that "www.google.com A 133.7.133.7"

• If you succeed, great!

• But if you fail, just try again with b.google.com!
  • Turns "Race once per timeout" to "race until win"

• So now the attacker may still have to send lots of packets
  • In the 10s of thousands

• The attacker can keep trying until success
Defending Against Kaminski: Up the Entropy

- Also randomize the UDP source port
  - Adds ~16 bits of entropy
- Observe that most DNS servers just copy the request directly
  - Rather than create a new reply
- So **caMeLcase the Name randomly**
  - Adds only a few bits of entropy however, but it does help
Defend Against Kaminski: Validate Glue

- Don't blindly accept glue records...
  - Well, you *have* to accept them for the purposes of resolving a name
  - But if you are going to cache the glue record...
- Either only use it for the context of a DNS lookup
  - No more promotion
- Or explicitly validate it with another fetch
- Unbound implemented this, bind did not
  - Largely a *political* decision:
    bind's developers are heavily committed to DNSSEC (an upcoming topic)
Oh, and Profiting from Rogue DNS

• Suppose you take over a lot of home routers...
• How do you make money with it?
• Simple: Change their DNS server settings
• Make it point to yours instead of the ISPs
• Now redirect all advertising
• And instead serve up ads for "Vimax" pills...
• Can only do this for unencrypted sites, but...