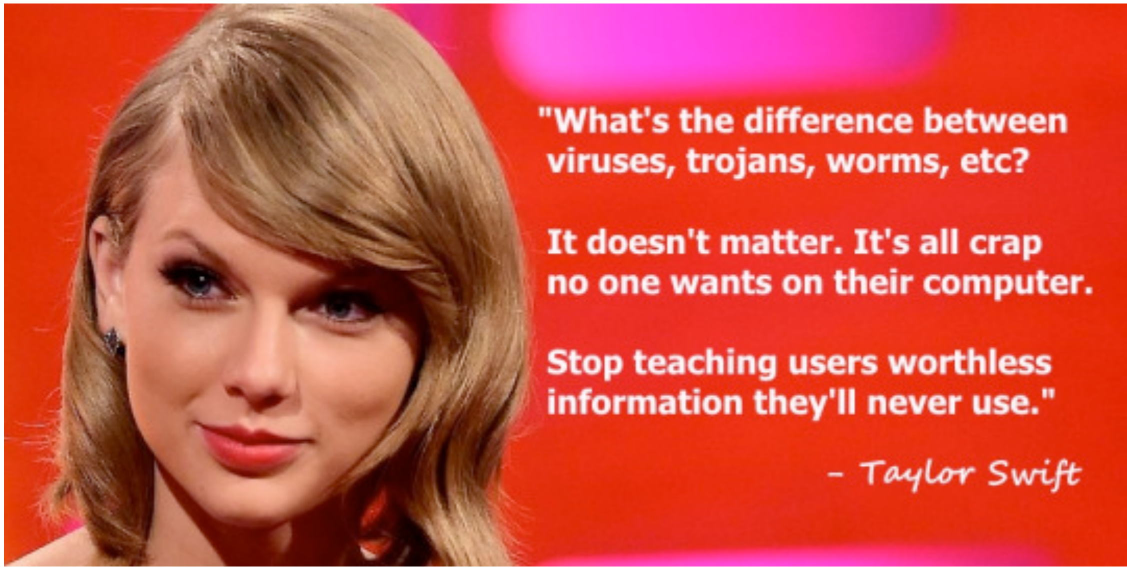
Network Security 6

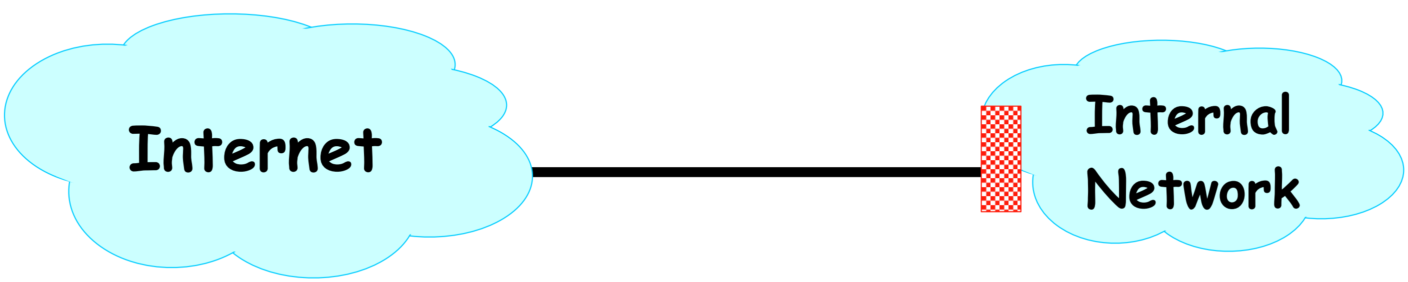


Controlling Networks … On The Cheap

- 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 all the services that are running
  - 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 thousands of hosts
    - Where in everyday experience do we see such chokepoints?



Selecting a Security Policy

- Firewall enforces an (access control) policy:
  - Who is allowed to talk to whom, accessing what service?
- Distinguish between inbound & outbound connections
  - Inbound: attempts by external users to connect to services on internal machines
  - Outbound: internal users to external services
  - Why? Because fits with a common threat model. There are thousands of internal users (and we've vetted them). There are billions of outsiders.
- 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

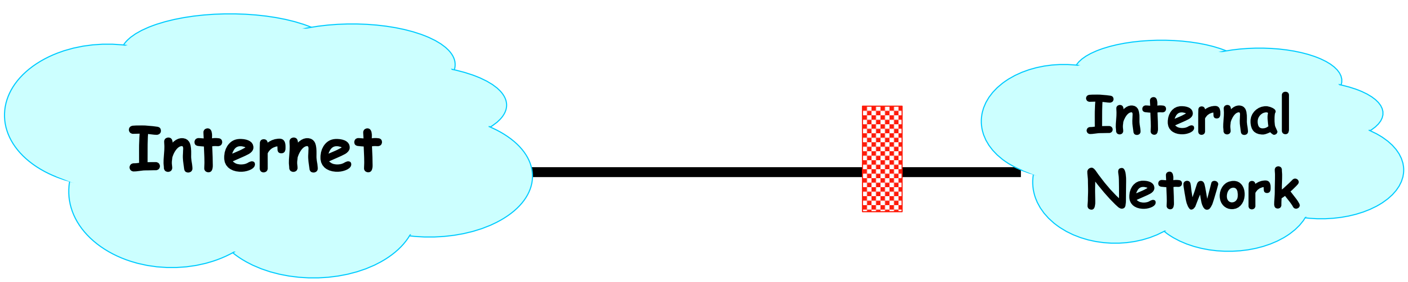
In general, use Default Deny

A Dumb Policy: Deny All Inbound connections...

- The simplest packet filters are ***stateless***
  - They examine only individual packets to make a decision
- But even the simplest policy can be hard to implement
  - Deny All Inbound is the default policy on your home connection
- Allow:
  - Any outbound packet
  - Any inbound packet that is a reply... OOPS
- We can fake it for TCP with some ugly hacks
  - Allow all outbound TCP
  - Allow all inbound TCP that does not have both the SYN flag set and the ACK flag not set
    - May still allow an attacker to play some interesting games
- We can't even fake this for UDP!

Stateful Packet Filter

- Stateful packet filter is a router that checks each packet against security rules and decides to forward or drop it
  - Firewall keeps track of all connections (inbound/outbound)
  - Each rule specifies which connections are allowed/denied (access control policy)
  - A packet is forwarded if it is part of an allowed connection



Example Rule

- `allow tcp connection 4.5.5.4:* -> 3.1.1.2:80`
  - Firewall should permit TCP connection that's:
    - Initiated by host with Internet address 4.5.5.4 and
    - Connecting to port 80 of host with IP address 3.1.1.2
  - Firewall should permit any packet associated with this connection
- Thus, firewall keeps a table of (allowed) active connections. When firewall sees a packet, it checks whether it is part of one of those active connections. If yes, forward it; if no, check to see if rule should create a new allowed connection

Example Rule

- `allow tcp connection *:*/int -> 3.1.1.2:80/ext`
  - Firewall should permit TCP connection that's:
    - Initiated by host with any internal host and
    - Connecting to port 80 of host with IP address 3.1.1.2 on external Internet
  - Firewall should permit any packet associated with this connection
  - The /int indicates the network interface.
- This is "Allow all outgoing web requests"

Example Ruleset

- **`allow tcp connection *:*/int -> *:*/ext`**
- **`allow tcp connection *:*/ext -> 1.2.2.3:80/int`**
  - Firewall should permit outbound TCP connections (i.e., those that are initiated by internal hosts)
  - Firewall should permit inbound TCP connection to our public webserver at IP address 1.2.2.3

Stateful Filtering

- Suppose you want to allow inbound connection to a FTP server, but block any attempts to login as “root”. How would you build a stateful packet filter to do that? In particular, what state would it keep, for each connection?

State Kept

- No state – just drop any packet with root in them

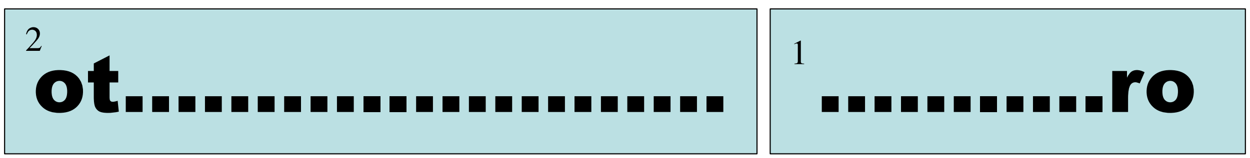
- Is it a FTP connection?
- Where in FTP state (e.g. command, what command)
- Src ip addr, dst ip addr, src port, dst port
- Inbound/outbound connection
- Keep piece of login command until it's completed – only first 5 bytes of username

Beware!

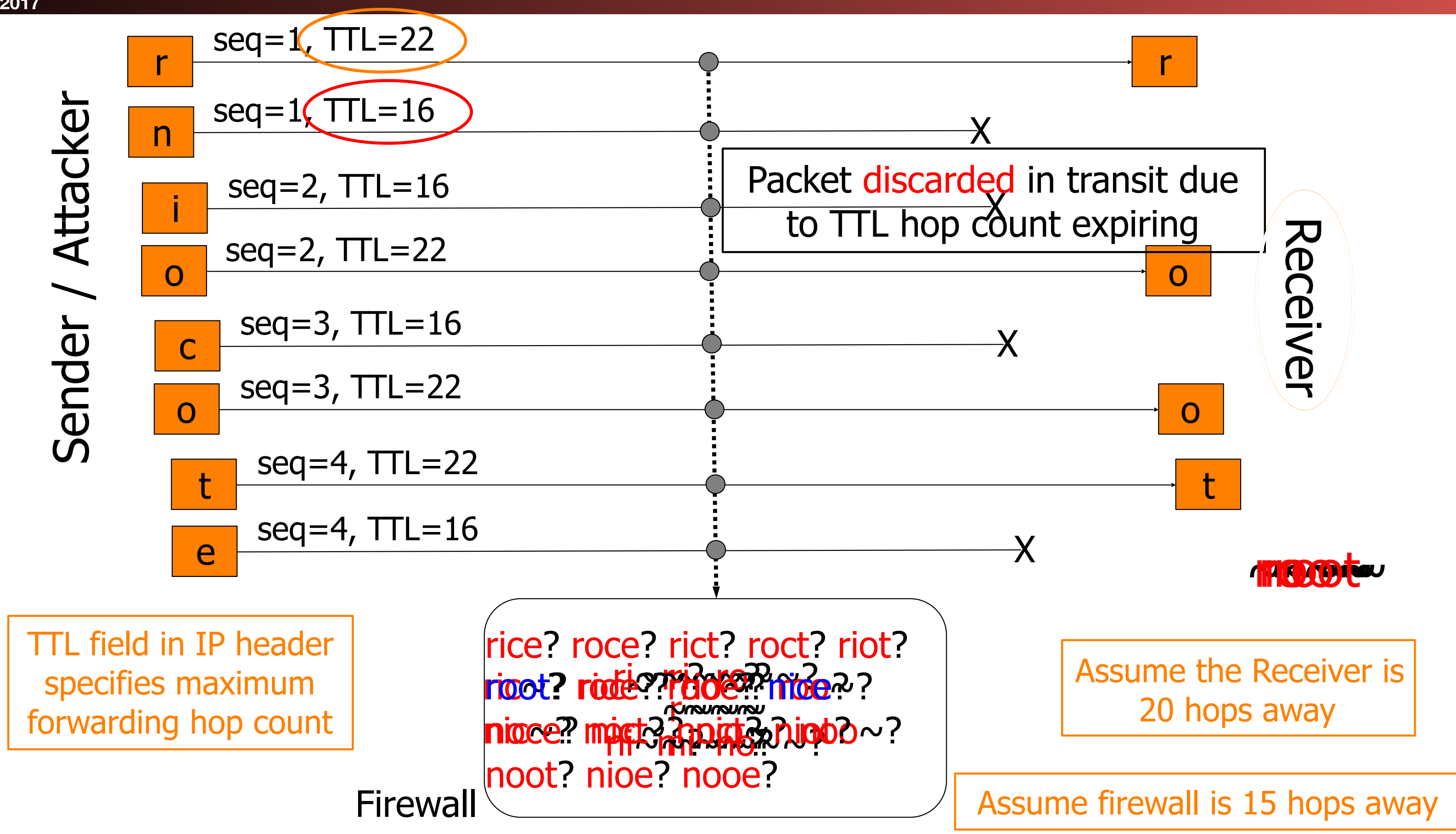
- Sender might be malicious and trying to sneak through firewall
- “root” might span packet boundaries

Beware!

- Packets might be re-ordered



Beware!

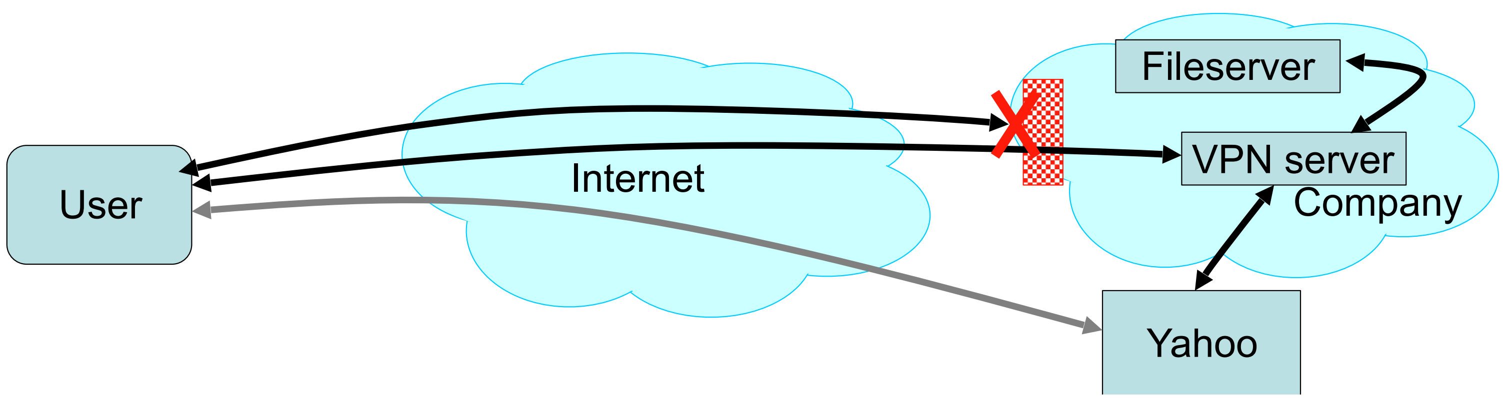

Sender / Attacker
r
seq=1, TTL=22
n
seq=1, TTL=16
i
seq=2, TTL=16
o
seq=2, TTL=22
c
seq=3, TTL=16
o
seq=3, TTL=22
t
seq=4, TTL=22
e
seq=4, TTL=16
X
Packet discarded in transit due to TTL hop count expiring
Receiver
r
o
o
t
TTL field in IP header specifies maximum forwarding hop count
Firewall
rice? roce? rict? roct? riot?
noot? nioe? nooe?
Assume the Receiver is 20 hops away
Assume firewall is 15 hops away

Other Kinds of Firewalls

- Application-level firewall
  - Firewall acts as a proxy. TCP connection from client to firewall, which then makes a second TCP connection from firewall to server.
  - Only modest benefits over stateful packet filter.

Secure External Access to Inside Machines

- Often need to provide secure remote access to a network protected by a firewall
  - Remote access, telecommuting, branch offices, …
- Create secure channel (Virtual Private Network, or VPN) to tunnel traffic from outside host/network to inside network
  - Provides Authentication, Confidentiality, Integrity
  - However, also raises perimeter issues
  - (Try it yourself at http://www.net.berkeley.edu/vpn/)



Why Have Firewalls Been Successful?

- Central control – easy administration and update
  - Single point of control: update one config to change security policies
  - Potentially allows rapid response
- Easy to deploy – transparent to end users
  - Easy incremental/total deployment to protect 1000's
- Addresses an important problem
  - Security vulnerabilities in network services are rampant
  - Easier to use firewall than to directly secure code …

Firewall Disadvantages

- Functionality loss – less connectivity, less risk
  - May reduce network's usefulness
  - Some applications don't work with firewalls
    - Two peer-to-peer users behind different firewalls
- The malicious insider problem
  - Assume insiders are trusted
    - Malicious insider (or anyone gaining control of internal machine) can wreak havoc
- Firewalls establish a security perimeter
  - Like Eskimo Pies: "hard crunchy exterior, soft creamy center"
  - Threat from travelers with laptops, cell phones, …

Pivoting...

- Thus the goal of the attacker is to "pivot" through the system
  - Start running on a single victim system
    - EG, using a channel that goes from the victim to the attacker's server over port 443: an encrypted web connection
- From there, you can now exploit internal systems directly
  - Bypassing the primary firewall
- That is the problem: A ***single*** breach of the perimeter by an attacker and you can no longer make ***any*** assertions about subsequent internal state

Takeaways on Firewalls

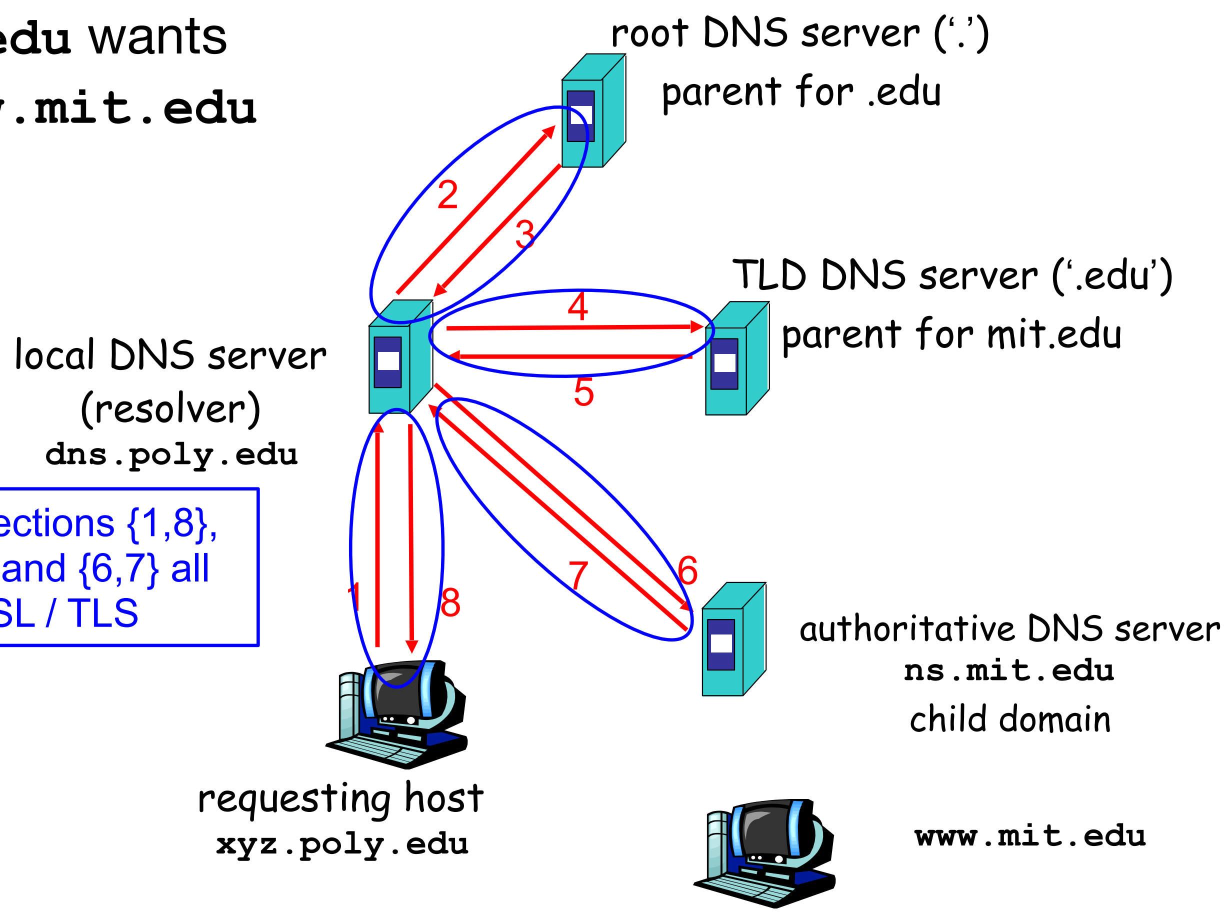
- Firewalls: Reference monitors and access control all over again, but at the network level
- Attack surface reduction
- Centralized control

A Warning:
I'm Giving ***Unfiltered*** DNSSEC

- Why?
  - Because it is a well thought through cryptographic protocol designed to solve a real world data integrity problem
  - It is a real world PKI with some very unique trust properties:
    - A constrained ***path of trust*** along ***established business relationships***.
  - It is important to appreciate the real world of what it takes to build a secure system
  - I've worked with it for far too much for my own sanity...
  - And I'm a cruel bastard

Hypothetical: Securing DNS Using SSL/TLS

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



Idea: connections {1,8}, {2,3}, {4,5} and {6,7} all run over SSL / TLS

But This Doesn't Work

- TLS provides ***channel*** integrity, but we need ***data*** integrity
- TLS in this scheme is not ***end to end***
  - In particular, the recursive resolver is a ***known adversary:***
    - "NXDOMAIN wildcarding": a "helpful" page when you give a typo
    - Malicious MitM of targeted schemes for profit
- TLS in this scheme is ***painfully slow***:
  - DNS lookups are 1 RTT, this is 3 RTTs!
- And ***confidentiality*** is of little benefit:
  - We use DNS to contact hosts:
    Keeping the DNS secret doesn't actually disguise who you talk to!

DNS security:
If the Attacker sees the traffic...

- All bets are off:
  - DNS offers NO protection against an on-path or in-path adversary
    - Attacker sees the request, sends the reply, and the reply is accepted!
- The recursive resolver is the most common in-path adversary!
  - It is implicitly trusted
  - Yet ***often abuses*** the trust
- And this scheme keeps the resolver as the in-path adversary

So Instead Let's Make DNS a PKI and records certificates

- **`www.berkeley.edu`** is already trusting the DNS authorities for **`berkeley.edu`**, **`.edu`**, and . (the root)
  - Since **`www.berkeley.edu`** is in bailiwick for all these servers and you end up having to contact all of them to get an answer.
- So let's start signing things:
  - . will sign .edu's key
  - .edu will sign Berkeley's key
  - Berkeley's key will sign the record
- DNSSEC: DNS Security Extensions
  - A heirarchical, distributed trust system to validate the mappings of names to values

Enter DNSSEC (DNS Security Extensions)

- An extension to the DNS protocol to enable cryptographic authentication of DNS records
  - Designed to prove the value of an answer, ***or that there is no answer***!
  - A restricted path of trust
    - Unlike the HTTPS CA (Certificate Authority) system where your browser trusts every CA to speak for every site
- With backwards compatibility:
  - Authority servers don't need to support DNSSEC
    - But clients should know that the domain is not secured
  - Recursive and stub resolvers that don't support DNSSEC must not receive DNSSEC information

Reminder: DNS Message Structure

- DNS messages:
  - A fixed header: Transaction ID, flags, etc...
  - 1 question: Asking for a name and type
  - 0-N answers: The set of answers
  - 0-N authority: ("glue records"): Information about the authority servers and/or ownership of the domain
  - 0-N additional: ("glue records"): Information about the authority server's IP addresses
    - Glue records are needed for the resolution process but aren't the answer to the question

Reminder:
DNS Resource Records and RRSETs

- DNS records (Resource Records) can be one of various types
  - Name TYPE TTL Value
- Groups of records of the same name and type form RRSETs:
  - E.g. all the nameservers for a given domain.
  - All the records in the RRSET have the same name, type, and TTL

The First New Type: OPT

- DNS contains some old limits:
  - Only 8 total flag bits, and messages are limited to 512B
- DNSSEC messages are much bigger
- DNSSEC needs two additional flags
  - DO: Want DNSSEC information
  - CD: Don't check DNSSEC information
- EDNS0 (Extension Mechanisms for DNS) adds the OPT resource record
  - Sent in the ***request*** and reply in the additional section
    - Uses CLASS field to specify how large a UDP reply can be handled
    - Uses TTL field to add 16 flag bits
      - Only flag bit currently used is DO
  - Used to signal to the authority that the client desires DNSSEC information

EDNS0 in action

- A query using **`dig +bufsize=1024`** uses EDNS0

```
nweaver% dig +norecurse +bufsize=1024 slashdot.org @a.root-servers.net

; <<>> DiG 9.8.3-P1 <<>> +bufsize=1024 slashdot.org @a.root-servers.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13419
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 6, ADDITIONAL: 13

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096

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

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

The second new type, a certificate: RRSIG

- A signature over an RRSET (not just a single answer): Multiple fields
  - Type: The DNS type which this is the RRSIG for
  - Algorithm: IANA assigned identifier telling the encryption algorithm
  - Labels: Number of segments in the DNS name
  - Original TTL: The TTL for the record delivered by the authority
  - Signature Expiration
  - Signature Inception
    - Both in seconds since January 1, 1970
  - Key tag: What key was used (roughly. Its a checksum on the key bits)
  - Signer's name
  - Signature

So an **RRSIG** in action
(The **NS** entries for **isc.org.**)

- Type of the record its an RRSIG for
- Algorithm #5: RSA/SHA-1
- 2 labels in the name
- 7200s initial TTL
- Valid 2013-04-15-23:32:55 to 2013-05-15-23:32:53
- Key tag 50012
- Key belongs to isc.org.
- And lots of cryptogarbage...

```
nweaver% dig +dnssec NS isc.org @8.8.8.8
...

;; ANSWER SECTION:
isc.org.                4282    IN      NS      ns.isc.afilias-nst.info.
isc.org.                4282    IN      NS      sfba.sns-pb.isc.org.
isc.org.                4282    IN      NS      ord.sns-pb.isc.org.
isc.org.                4282    IN      NS      ams.sns-pb.isc.org.
isc.org.                4282    IN      RRSIG   NS 5 2 7200 20130515233253
20130415233253 50012 isc.org. HUXmb89gB4pVehWRcuSkJg020gw2d8QMhTrcu1ZD7nKomXHQFupXl5vT
iq5VUREGBQtnT7FEdPEJlCiJeogbAmqt3F1V5kBfdxZLe/EzYZgvSGWq sy/VHI5d+t6/
```

How Do We Know What Key To Use Part 1: **DNSKEY**

- The **DNSKEY** record stores key information
  - 16 bits of flags
  - Protocol identifier (always 3)
  - Algorithm identifier
  - And then the key itself
- The keys are split into multiple roles
  - The Key Signing Key (KSK) is used only to sign the **DNSKEY** RRSET
  - The Zone Signing Key (ZSK) is used to sign everything else
- The client has hardwired in one key for .
  - This is the root's KSK (Key Signing Key)

The **DNSKEY** for .

- The first is the root's ZSK
- The second is the root's KSK
- The RRSIG is signed using the KSK
  - Now the client can verify that the ZSK is correct

```
nweaver% dig +norecurse +dnssec DNSKEY . @a.root-servers.net
...
;; ANSWER SECTION:
.                       172800  IN      DNSKEY  256 3 8 AwEAAc5byZvwmHUlCQt7WSeAr3OZ2ao4x0Yj/
3UcbtFzQ0T67N7CpYmN qFmfvXxksS1/E+mtT0axFVDjiJjtklUsyqIm9ZlWGZKU3GZqI9Sfp1Bj
Qkhi+yLa4m4y4z2N28rxWXsWHCY740PREnmUtgXRdthwABYaB2WPum3y RGxNCP1/
.                       172800  IN      DNSKEY  257 3 8
AwEAAagAIKlVZrpC6Ia7gEzahOR+9W29euxhJhVVLOyQbSEW0O8gcCjF FVQUTf6v58fLjwBd0YI0EzrAcQqBGCzh/
RStIoO8g0NfnfL2MTJRkxoX bfDaUeVPQuYEhg37NZWAJQ9VnMVDxP/VHL496M/QZxkjf5/Efucp2gaD
X6RS6CXpoY68LsvPVjR0ZSwzz1apAzvN9dlzEheX7ICJBBtuA6G3LQpz
W5hOA2hzCTMjJPJ8LbqF6dsV6DoBQzgul0sGIcGOYl7OyQdXfZ57relS
Qageu+ipAdTTJ25AsRTAoub8ONGcLmqrAmRLKBP1dfwhYB4N7knNnulq QxA+Uk1ihz0=
.                       172800  IN      RRSIG   DNSKEY 8 0 172800 20130425235959 20130411000000
19036 . {Cryptographic Goop}
```

But how do we know what key to use part 2? `DS`

- The **`DS`** (Delegated Signer) record is relatively simple
  - The key tag
  - The algorithm identifier
  - The hash function used
  - The hash of the signer's name and the KSK
- The parent signs **`DS`** (Delegated Signer) records for the child's keys
  - So for the DS for .org is provided by the root
  - This is returned with the NS RRSET by the parent
    - And the RRSIG is signed by the parent, not the child

The **DS** for **org.**

- The two DS records are for the same key
  - Just with different hash functions, SHA-256 and SHA-1
- The RRSIG is signed using the ZSK not the KSK
  - And covers both DS records

```
nweaver% nweaver% dig +norecurse +dnssec www.isc.org @a.root-servers.net
...
;; AUTHORITY SECTION:
org.                    172800  IN      NS      d0.org.afilias-nst.org.
...
org.                    172800  IN      NS      a0.org.afilias-nst.info.
org.                    86400   IN      DS      21366 7 2
96EEB2FFD9B00CD4694E78278B5EFDAB0A80446567B69F634DA078F0 D90F01BA
org.                    86400   IN      DS      21366 7 1 E6C1716CFB6BDC84E84CE1AB5510DAC69173B5B2
org.                    86400   IN      RRSIG   DS 8 1 86400 20130423000000 20130415230000 20580 .
{Cryptographic Goop}
```

Putting It All Together To Lookup `www.isc.org`

`? A www.isc.org`

User's ISP's Recursive Resolver

`? A www.isc.org`

Name	Type	Value	TTL	Valid
.	DNSKEY	{cryptogoop}	N/A	Yes

. Authority Server (the "root")

```
? A www.isc.org
Answers:
Authority:
org. NS a0.afilias-nst.info
org. IN DS 21366 7 2 {cryptogoop}
org. IN DS 21366 7 1 {cryptogoop}
org. IN RRSIG DS 8 1 86400 20130423000000
20130415230000 20580 . {cryptogoop}
Additional:
a0.afilias-nst.info A 199.19.56.1
```

Putting It All Together To Lookup `www.isc.org`

.
Authority Server
(the "root")

```
? DNSKEY .
Answers:
. IN DNSKEY  257 3 8 {cryptogoop}
. IN DNSKEY 256 3 8 {cryptogoop}
. IN RRSIG DNSKEY 8 0 172800 20130425235959
20130411000000 19036 . {cryptogoop}
Authority:
Additional:
```

User's ISP's Recursive Resolver

? DNSKEY .

Name	Type	Value	TTL	Valid
org.	NS	a0.afilia-nst.info		No
a0.afilias-nst.info	A	199.19.56.1	86400	No
org.	DS	{cryptogoop}	86400	No
org.	DS	{cryptogoop}	86400	No
org.	RRSIG	DS {goop}	86400	No
.	DNSKEY	{cryptogoop}	N/A	Yes

Putting It All Together To Lookup `www.isc.org`

.
Authority Server
(the "root")

User's ISP's
Recursive Resolver

Name	Type	Value	TTL	Valid
org.	NS	a0.afilia-nst.info		No
a0.afilias-nst.info	A	199.19.56.1	86400	No
org.	DS	{cryptogoop}	86400	No
org.	DS	{cryptogoop}	86400	No
org.	RRSIG	DS {goop}	86400	No
.	DNSKEY	{cryptogoop}	172800	Yes
.	RRSIG	DNSKEY {goop}	172800	Yes
.	DNSKEY	{cryptogoop}	N/A	Yes

Putting It All Together To Lookup `www.isc.org`

User's ISP's Recursive Resolver

? A www.isc.org

Name	Type	Value	TTL	Valid
org.	NS	a0.afilia-nst.info		No
a0.afilias-nst.info	A	199.19.56.1	86400	No
org.	DS	{cryptogoop}	86400	Yes
org.	DS	{cryptogoop}	86400	Yes
org.	RRSIG	DS {goop}	86400	Yes
.	DNSKEY	{cryptogoop}	172800	Yes
.	RRSIG	DNSKEY {goop}	172800	Yes
.	DNSKEY	{cryptogoop}	N/A	Yes

org. Authority Server

```
? A www.isc.org
Answers:
Authority:
isc.org. NS sfba.sns-pb.isc.org.
isc.org. DS {cryptogoop}
isc.org. RRSIG DS {cryptogoop}
Additional:
sfba.sns-pb.isc.org.     A 199.6.1.30
```

Putting It All Together To Lookup `www.isc.org`

User's ISP's
Recursive Resolver

Name	Type	Value	TTL	Valid
org.	NS	a0.afilia-nst.info		No
a0.afilias-nst.info	A	199.19.56.1	86400	No
org.	DS	{cryptogoop}	86400	Yes
org.	DS	{cryptogoop}	86400	Yes
org.	RRSIG	DS {goop}	86400	Yes
.	DNSKEY	{cryptogoop}	172800	Yes
.	RRSIG	DNSKEY {goop}	172800	Yes
isc.org.	DS	{cryptogoop}	86400	No
isc.org.	DS	{cryptogoop}	86400	No
isc.org.	RRSIG	DS {goop}	86400	No
isc.org.	NS	sfbay.sns-pb.isc.org	86400	No
sfbay.sns-pb.isc.org	A	149.20.64.3	86400	No
.	DNSKEY	{cryptogoop}	N/A	Yes

And so on...

- The process ends up requiring:
  - Ask the root for **`www.isc.org`** and the **`DNSKEY`** for **`.`**
  - Ask **`org`** for **`www.isc.org`** and the **`DNSKEY`** for **`org.`**
  - Ask **`isc.org`** for **`www.isc.org`** and the DNSKEY for **`isc.org`**
- Dig commands
  - `dig +dnssec +norecurse www.isc.org @a.root-servers.net`
  - `dig +dnssec +norecurse DNSKEY . @a.root-servers.net`
  - `dig +dnssec +norecurse www.isc.org @199.19.56.1`
  - `dig +dnssec +norecurse DNSKEY org. @199.19.56.1`
  - `dig +dnssec +norecurse www.isc.org @149.20.64.3`
  - `dig +dnssec +norecurse DNSKEY isc.org. @149.20.64.3`

So why such a baroque structure?

- Goal is end-to-end data ***integrity***
  - Even authorized intermediaries such as the recursive resolver don't need to be trusted
  - Don't benefit (much) from confidentiality since DNS is used to contact hosts
- Signature generation can be done all offline
  - Attacker must compromise the signature generation system, not just the authority nameserver
    - Allows other authority servers to be simply mirrors
- Validation can happen at either the recursive resolver or the client
  - The DNSKEYs cache very well
    - So most subsequent lookups will not need to do these lookups
- Constrained path of trust
  - For a given name, can enumerate the trusted entities

Another reason: Latency

- The DNS community is obsessed with latency
  - Thus the refusal to simply switch to TCP for all DNS traffic
- A recursive resolver may
  - Automatically fetch the **DNSKEY** record with a parallel request
  - While waiting for a child's response, validate the parent's **DS** record
    - Generally the validation should be the same time or faster so we can do this in parallel
  - Result: Only two signature validations of latency added even on uncached requests and no additional network latency
    - One for the **DNSKEY** to get the ZSK
    - One for the final RRSET
- A stub resolver looking up foo.example.com:
  - In parallel fetch **DS** and **DNSKEY** for foo.example.com, example.com, .com, and the DNSKEY for .

Two additional complications

- ***NOERROR***:
  - The name exists but there is no record of that given type for that name
  - For DNSSEC, prove that there is no ds record
    - Says the subdomain doesn't sign with DNSSEC
- ***NXDOMAIN***:
  - The name does not exist
- **`NSEC`** (Provable denial of existence), a record with just two fields
  - Next domain name
    - The next valid name in the domain
  - Valid types for this name
    - In a bitmap for efficiency

NSEC in action

- Name is valid so **NOERROR** but no answers
- Single **NSEC** record for **www.isc.org**:
  - No names exist between **www.isc.org** and **www-dev.isc.org**
  - **www.isc.org** only has an **A**, **AAAA**, **RRSIG**, and **NSEC** record

```
nweaver% dig +dnssec TXT www.isc.org @8.8.8.8
...
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20430
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 1
...
;; QUESTION SECTION:
;www.isc.org.                     IN      TXT

;; AUTHORITY SECTION:
...
www.isc.org.            3600      IN      NSEC     www-dev.isc.org. A AAAA RRSIG NSEC
www.isc.org.            3600      IN      RRSIG    NSEC {RRSIG DATA}
```

The Use of **NSEC**

- Proof that a name exists but no type exists for that name
  - Critical for “This subdomain doesn’t support DNSSEC”:
    Return an **NSEC** record with the authority stating “There is no **DS** record”
- Proof that a name does not exist
  - It falls between the two **NSEC** names
  - Plus an **NSEC** saying “there is no wildcard”
- Allows trivial domain enumeration
  - Attacker just starts at the beginning and walks through the NSEC records
    - Some consider this bad...

So **NSEC3**

- Rather than having the name, use a ***hash*** of the name
  - Hash Algorithm
  - Flags
- Iterations of the hash algorithm
- Salt (optional)
- The next name
- The RRTYPEs for this name
  - Otherwise acts like **NSEC**, just in a different space

```
nweaver% dig +dnssec TXT org @199.19.57.1
...
;; AUTHORITY SECTION:
...
h9p7u7tr2u91d0v0ljs9l1gidnp90u3h.org. 86400 IN NSEC3 1 1 1 D399EAAB
    H9Q3IMI6H6CIJ4708DK5A3HMJLEIQ0PF NS SOA RRSIG DNSKEY NSEC3PARAM
h9p7u7tr2u91d0v0ljs9l1gidnp90u3h.org. 86400 IN RRSIG NSEC3 {RRSIG}
```

Comments on NSEC3

- It doesn't ***really*** prevent enumeration
  - You get a hash-space enumeration instead, but since people chose reasonable names...
  - An attacker can just do a brute-force attack to find out what names exist and don't exist
- The salt is mostly pointless!
  - Since the ***whole*** name is hashed, **`foo.example.com`** and **`foo.example.org`** will have different hashes anyway
- The only way to really prevent enumeration is to dynamically sign values
  - But that defeats the purpose of DNSSEC's offline signature generation

So what can *possibly* go wrong?

- Screwups on the authority side...
  - Too many ways to count...
    - But comcast is keeping track of it:
      Follow @comcastdns on twitter
- The validator can't access DNSSEC records
- The validator can't process DNSSEC records correctly

Authority Side Screwups...

- Its quite common to screw up
- Tell your registrar you support DNSSEC when you don't
  - Took down HBO Go's launch for Comcast users and those using Google Public DNS
- Rotate your key but present old signatures
- Forget that your signatures expire

And The Recursive Resolver Must Not Be Trusted!

- Most deployments validate at the recursive resolver, not the client
  - Notably Google Public DNS and Comcast
- This provides very little practical security:
  - The recursive resolver has proven to be the biggest threat in DNS
  - And this doesn't protect you between the recursive resolver and your system
- But causes a lot of headaches
  - Comcast or Google invariably get blamed when a zone screws up
  - Fortunately this is getting less common...

DNSSEC transport

- A validating client must be able to fetch the DNSSEC related records
  - It may be through the recursive resolver
  - It may be by contacting arbitrary DNS servers on the Internet
- One of these two must work or the client ***can not validate*** DNSSEC
  - This acts to limit DNSSEC's real use:
    Signing other types such as cryptographic fingerprints (e.g. DANE)

Probe the Root
To Check For DNSSEC Transport

- Can the client get DNSSEC data from the Internet?
  - Probe every root with DO for:
    - DS for .com with RRSIG
    - DNSKEY for . with RRSIG
    - NSEC for an invalid TLD with RRSIG
- Serves two purposes:
  - Some networks have one or more bad root mirrors
    - Notably one Chinese educational network has root mirrors for all but 3 that don't support DNSSEC
  - If no information can be retrieved
    - Proxy which strips out DNSSEC information and/or can't handle DO

DNSSEC Root Transport: Results We've Seen In The Wild

- Bad news at Starbucks: Hotspot gateways often proxy all DNS and can't handle DO-enabled traffic
  - And then have DNS resolvers that can't handle DNSSEC requests!
- Confirmed the Chinese educational network "Bad root mirror" problem

Implications of “No DNSSEC at Starbucks”

- DNSSEC failure depends on the usage.
- For name->address bindings:
  - If the recursive resolver practices proper port randomization:
    - No problem. The same “attackers” who can manipulate your DNS could do anything they want at the proxy that’s controlling your DNS traffic
  - Else:
    - Problem. Network is not secure
- For name->key bindings:
  - Unless the resolver supports it directly, you are Out of Luck
    - DNSSEC information must have an alternate channel if you want to use it to transmit keys instead of just IPs

In fact, my preferred DNSSEC policy For Client Validation

- For name->address mappings
  - Any existing APIs that don't provide DNSSEC status
  - If valid: use
  - If invalid OR no complete DNSSEC chain:
    - Begin an iterative fetch with the most precise DNSSEC-validated data
    - Use the result without question
- For name->data mappings
  - An API which returns DNSSEC status
  - If valid: Use
  - If invalid: Return DNSSEC failure status
    - Up to the application

And That's The Real Thing...

- DNSSEC in all its *emm* glory.
- OPT records to say "I want DNSSEC"
- RRSIG records are certificates
- DNSKEY records hold public keys
- DS records hold key fingerprints
    - Used by the parent to tell the child's keys
- NSEC/NSEC3 records to prove that a name doesn't exist or there is no record of that type