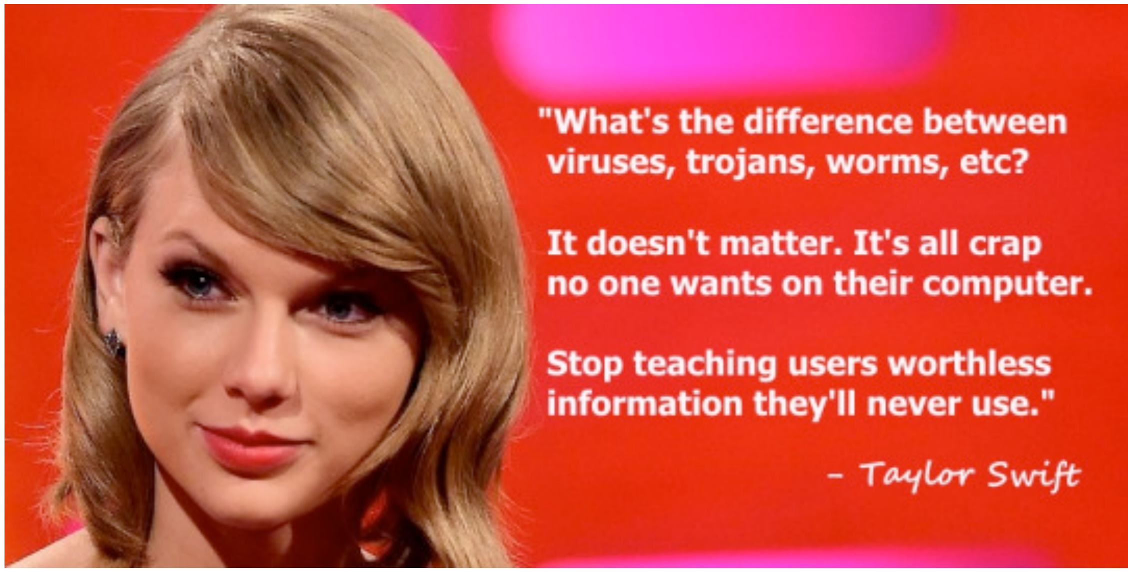
Network Security 7


"What's the difference between viruses, trojans, worms, etc?

It doesn't matter. It's all crap no one wants on their computer.

Stop teaching users worthless information they'll never use."

- Taylor Swift

News of the Day: Facebook/WhatsApp v NSO Group

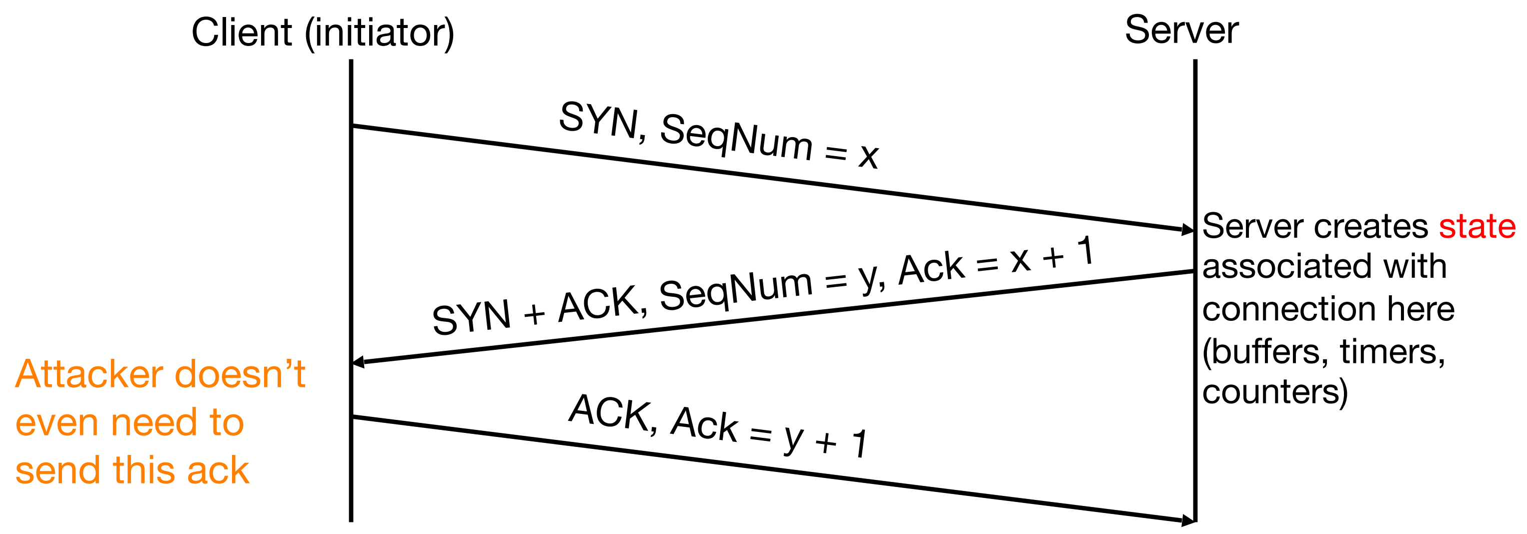
- NSO Group serves the "lawful" hacking market
  - Where "lawful" means "A government official signed our paycheck"
- They recently developed a nasty WhatsApp exploit
  - "Unanswered call" -> Information leakage to break ALSR
  - Then a heap-overflow to exploit
- And their customers used this to target >1400 targets
  - Including >100 journalists, activists, etc...
  - And government officials belonging to US ally governments

Facebook Struck Back

- Filed a lawsuit in federal court
  - Including publicizing a lot of NSO Group internal stuff in the initial filings
- Deleted all the NSO Group employees Facebook accounts
- Notified ***all*** targeted users
  - Based on recorded metadata

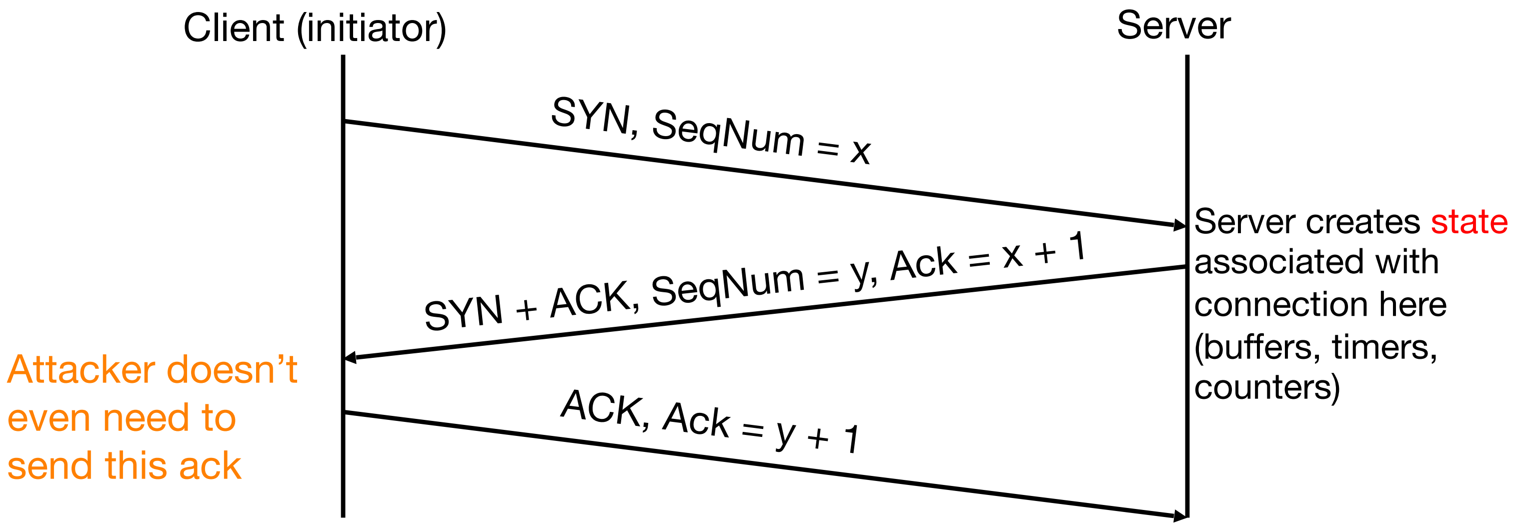
Transport-Level Denial-of-Service

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


Client (initiator)
Server
SYN, SeqNum = x
Server creates state associated with connection here (buffers, timers, counters)
SYN + ACK, SeqNum = y, Ack = x + 1
ACK, Ack = y + 1
Attacker doesn't even need to send this ack

Transport-Level Denial-of-Service

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


Client (initiator)
Server
SYN, SeqNum = x
SYN + ACK, SeqNum = y, Ack = x + 1
ACK, Ack = y + 1
Server creates state associated with connection here (buffers, timers, counters)
Attacker doesn't even need to send this ack

TCP SYN Flooding

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

TCP SYN Flooding Defenses

- How can the target defend itself?

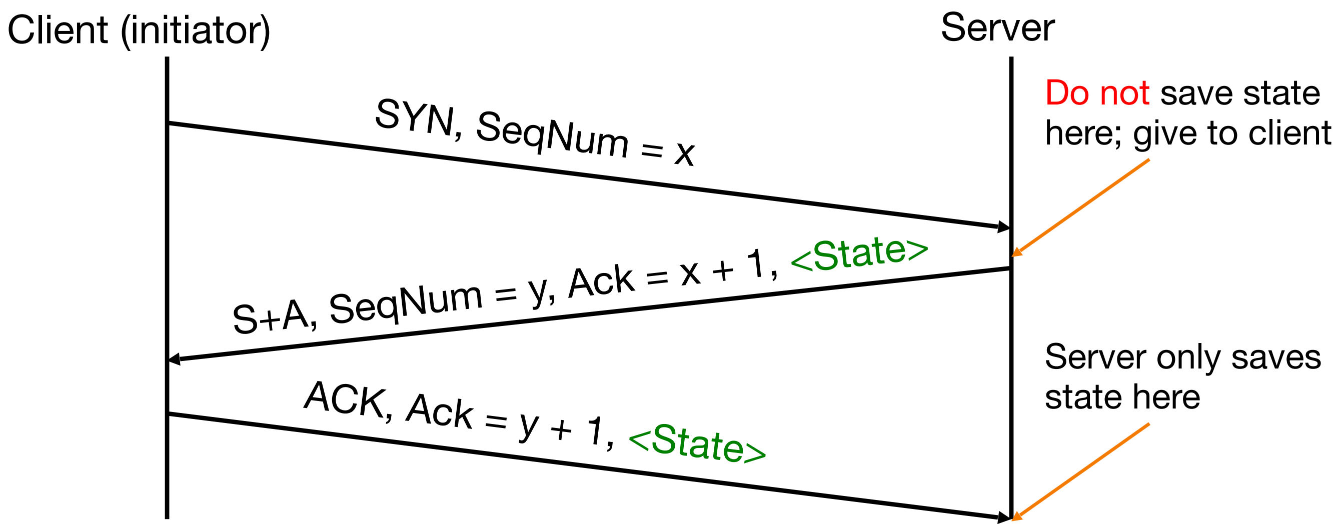
- Approach #1: make sure they have tons of memory!
  - How much is enough?
  - Depends on resources attacker can bring to bear (threat model), which might be hard to know
- Back of the envelope:
  - If we need to hold 10kB for 1 minute: to exhaust 1GB, an attacker needs...
    - 100k packets/minute, or a bit more than 1,000 packets per second

TCP SYN Flooding Defenses

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

SYN Flooding Defense: Idealized

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


Client (initiator)
Server
SYN, SeqNum = x
Do not save state here; give to client
S+A, SeqNum = y, Ack = x + 1, <State>
Server only saves state here
ACK, Ack = y + 1, <State>

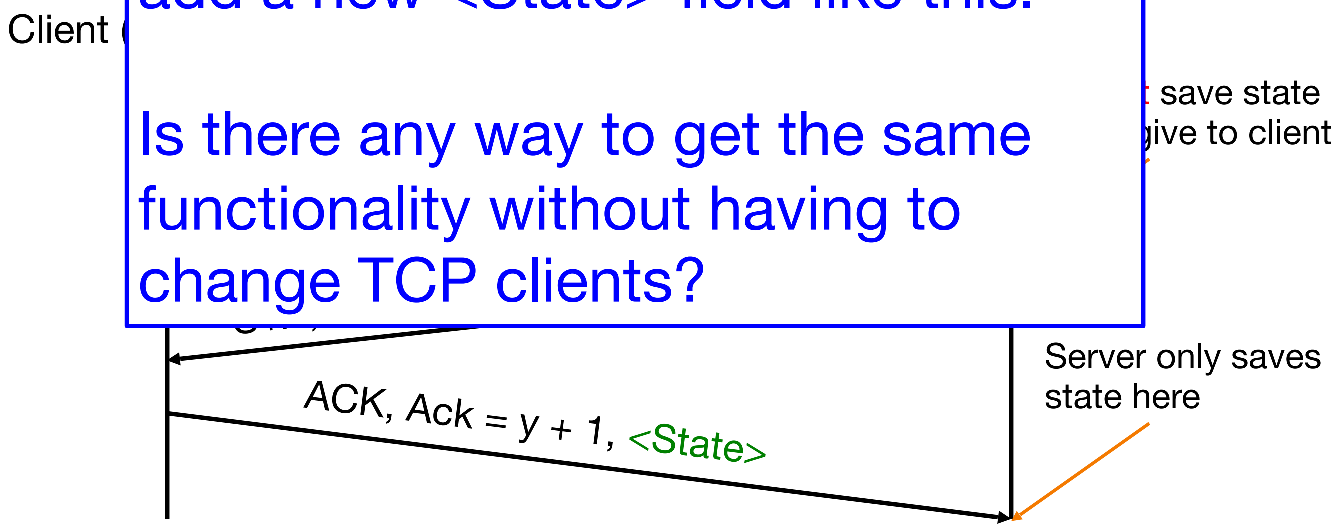
SYN Flooding Defense: Idealized

- Server: when SYN arrives, rather than keeping state locally, send it to the client ..
- Client needs to shed connection

Problem: the world isn't so ideal!

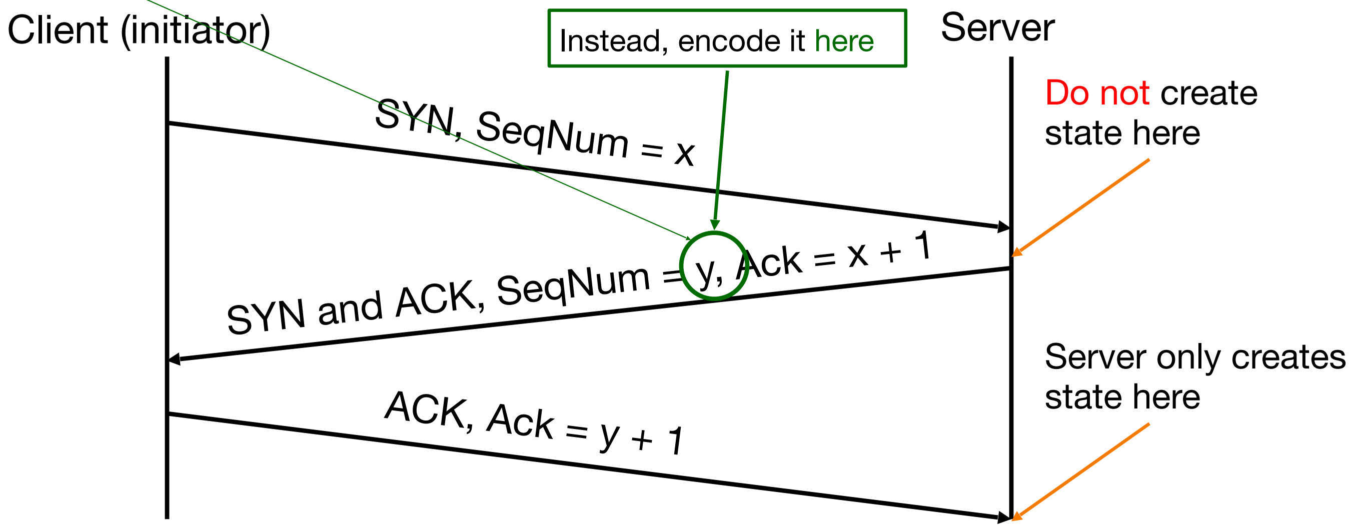
TCP doesn't include an easy way to add a new <State> field like this.

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



Practical Defense: SYN Cookies

- Server: when SYN arrives, encode connection state entirely within SYN-ACK's sequence # y
    - y = encoding of necessary state, using server secret
- When ACK of SYN-ACK arrives, server only creates state if value of y from it agrees w/ secret



SYN Cookies: Discussion

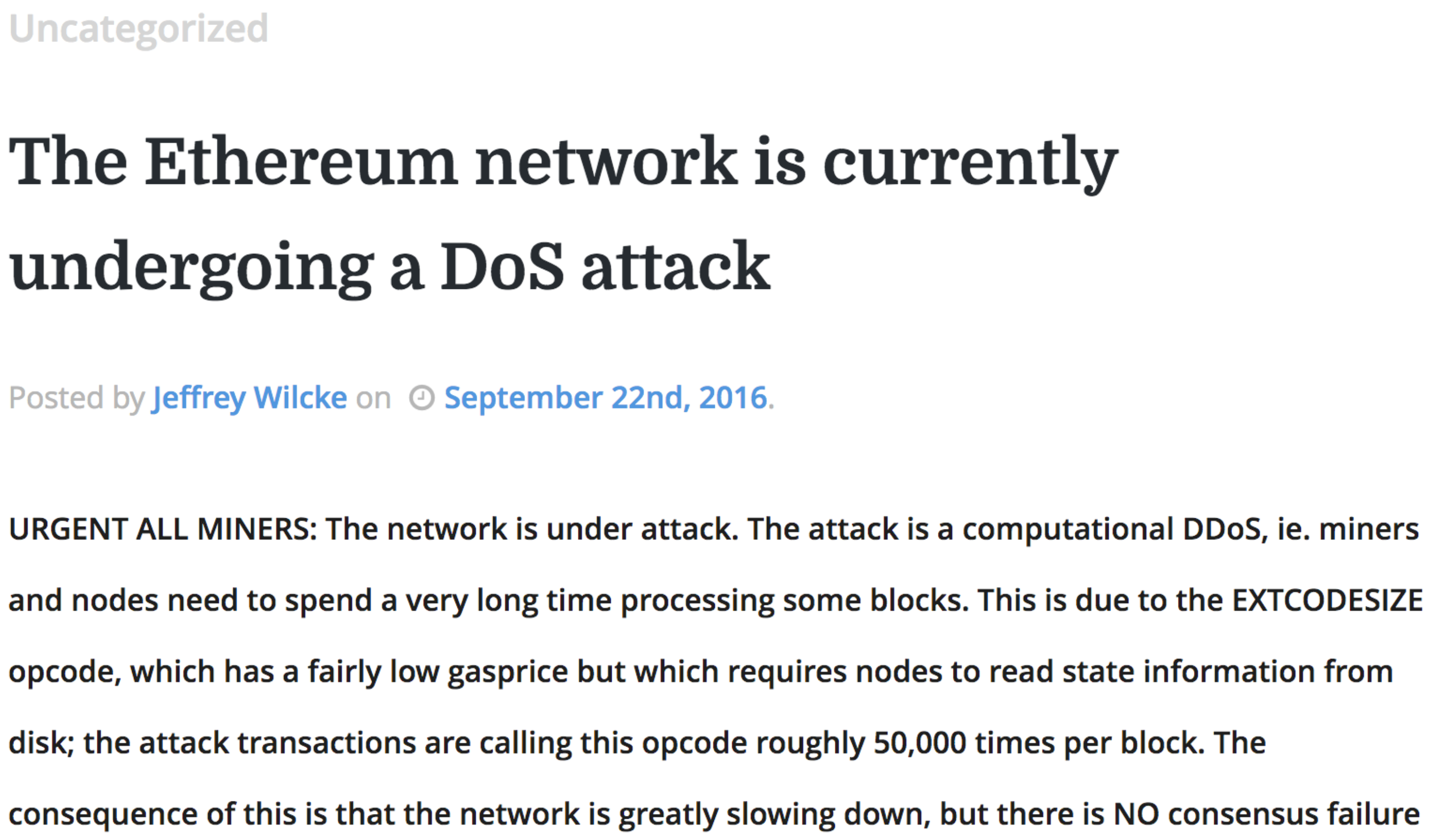
- Illustrates general strategy: rather than holding state, encode it so that it is returned when needed
- For SYN cookies, attacker must complete 3-way handshake in order to burden server
  - Can’t use spoofed source addresses
- Note #1: strategy requires that you have enough bits to encode all the state
  - (This is just barely the case for SYN cookies)
  - You can think of a SYN cookie as a truncated MAC of the sender IP/port/sequence: And really, HMAC is the easiest way to do this!
- Note #2: if it’s expensive to generate or check the cookie, then it’s not a win

And Once Again, HMAC to the rescue...

- HMAC is a great way to force others to store state...
  - Create cookie:
    HMAC(k, data) -> 🍪
  - Check cookie:
    HMAC(k, data) ?= 🍪
- Allow you to force others to store all the data you want that you can then verify later
  - All you need to do is make sure that they know they need to send all the data back to you will the cookie...
    - And you need the cookie to be big ***enough***

Application-Layer DoS

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



Uncategorized

The Ethereum network is currently undergoing a DoS attack

Posted by Jeffrey Wilcke on September 22nd, 2016.

URGENT ALL MINERS: The network is under attack. The attack is a computational DDoS, ie. miners and nodes need to spend a very long time processing some blocks. This is due to the EXTCODESIZE opcode, which has a fairly low gasprice but which requires nodes to read state information from disk; the attack transactions are calling this opcode roughly 50,000 times per block. The consequence of this is that the network is greatly slowing down, but there is NO consensus failure

Algorithmic complexity attacks

- Attacker can try to trigger worst-case complexity of algorithms / data structures
- Example: You have a hash table.
  Expected time: O(1). Worst-case: O(n).
- Attacker picks inputs that cause hash collisions.
  Time per lookup: O(*n*).
  Total time to do *n* operations: O(n^2).
- Solution? Use algorithms with good worst-case running time.
  - E.g., using ***b*** bits of HMAC ensures that $P[h_k(x)=h_k(y)] = .5^b$, so hash collisions will be rare.
    - If the attacker doesn't know the key that is

Application-Layer DoS

- Defenses against such attacks?
- Approach #1: Only let legit users issue expensive requests
  - Relies on being able to identify/authenticate them
  - Note: that this itself might be expensive!
- Approach #2: Force legit users to "burn" cash
  - This is what a captcha really is!
- Approach #3: massive over-provisioning ($$$)
  - Or pay for someone else who massively over provisions for everyone:
    A content delivery network

DoS Defense in General Terms

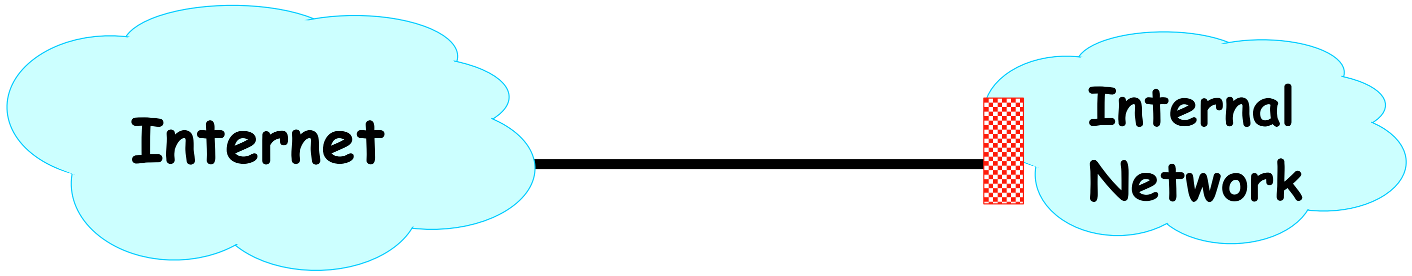
- Defending against program flaws requires:
  - Careful design and coding/testing/review
  - Consideration of behavior of defense mechanisms
    - E.g. buffer overflow detector that when triggered halts execution to prevent code injection ⇒ denial-of-service
- Defending resources from exhaustion can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

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 $\boxed{\text{users complain (and mgt. approves)}}$ ✓
- Pros & Cons?
  - Flexibility vs. conservative design
  - Flaws in Default Deny get noticed more quickly / less painfully

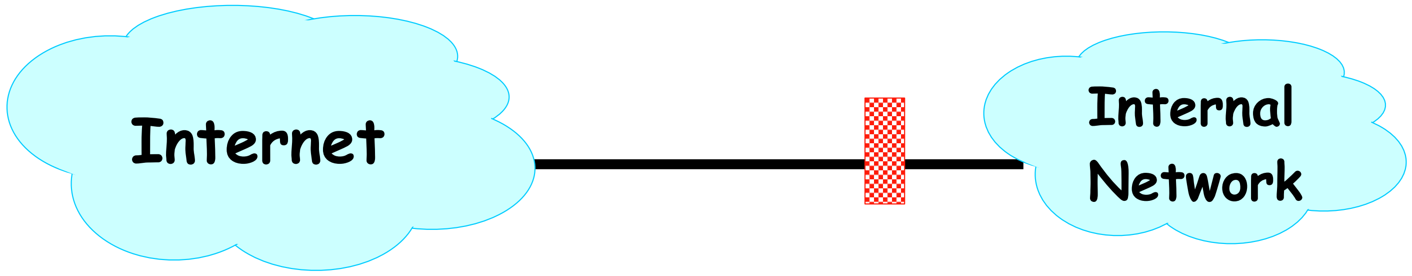
In general, use Default Deny

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?

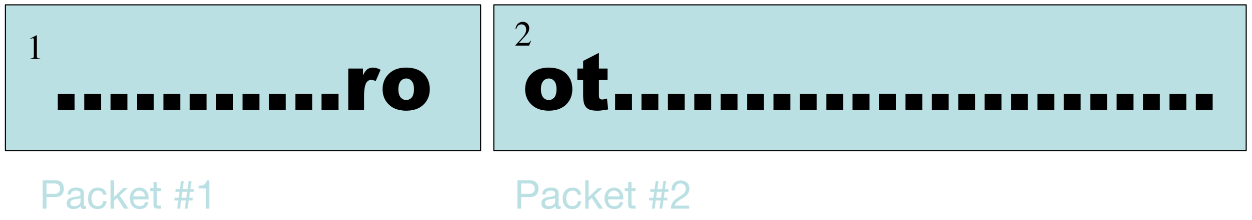
State Kept

- 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

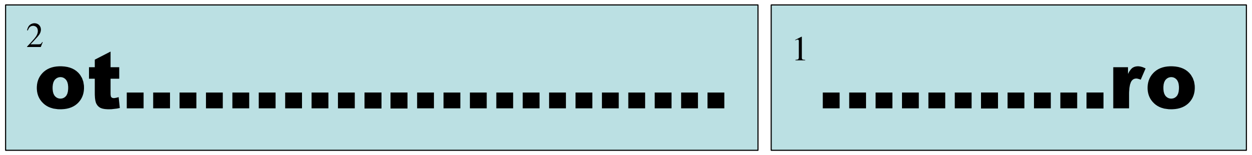
Beware!

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

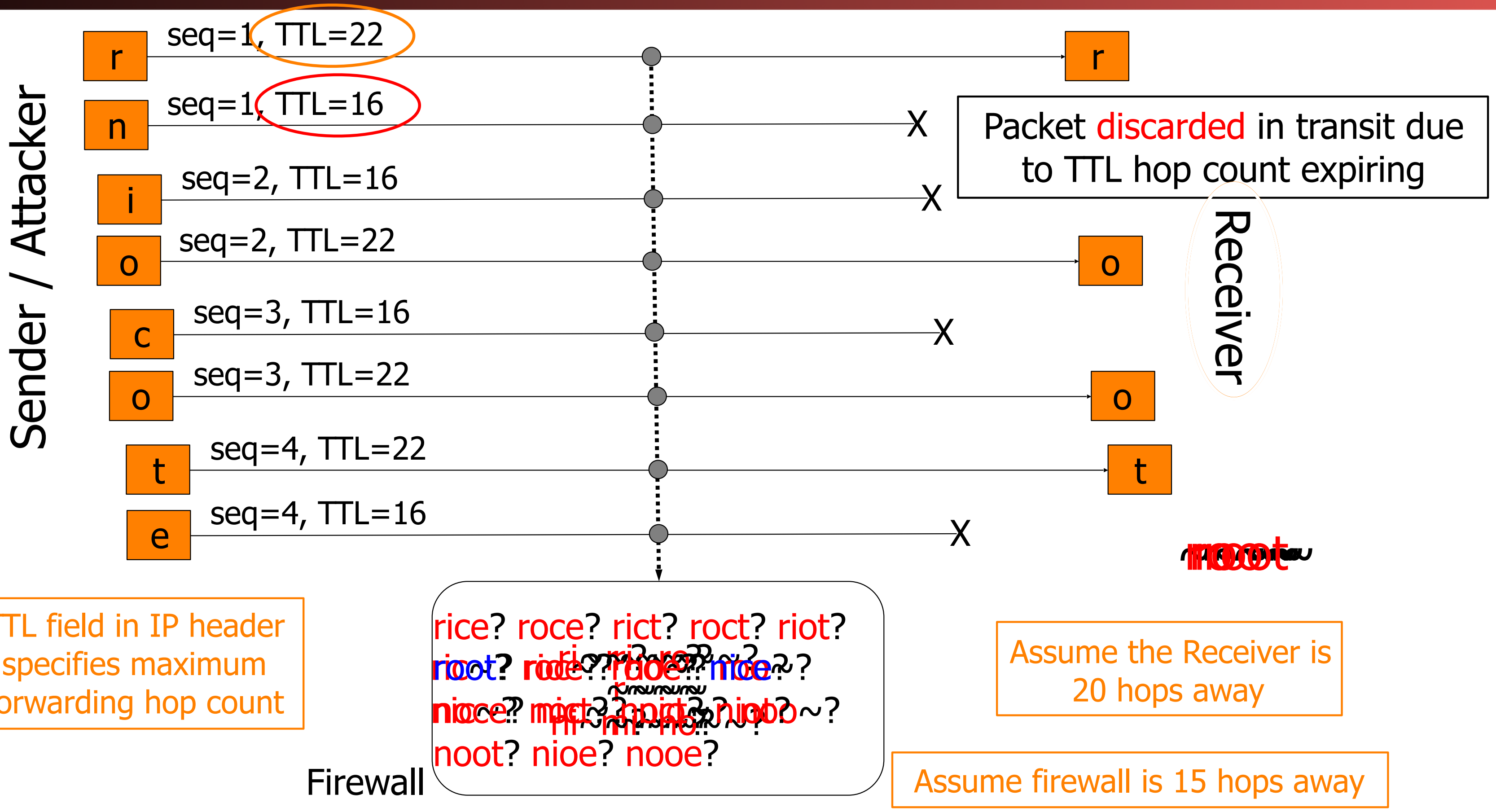


Beware!

- 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
X
X
X
r
o
o
t
Receiver
Packet discarded in transit due to TTL hop count expiring
rice? roce? rict? roct? riot? root? rioe? rooe? rice? nice? noce? nict? noct? niot? noot? nioe? nooe?
Firewall

TTL field in IP header specifies maximum forwarding hop count

Assume the Receiver is 20 hops away

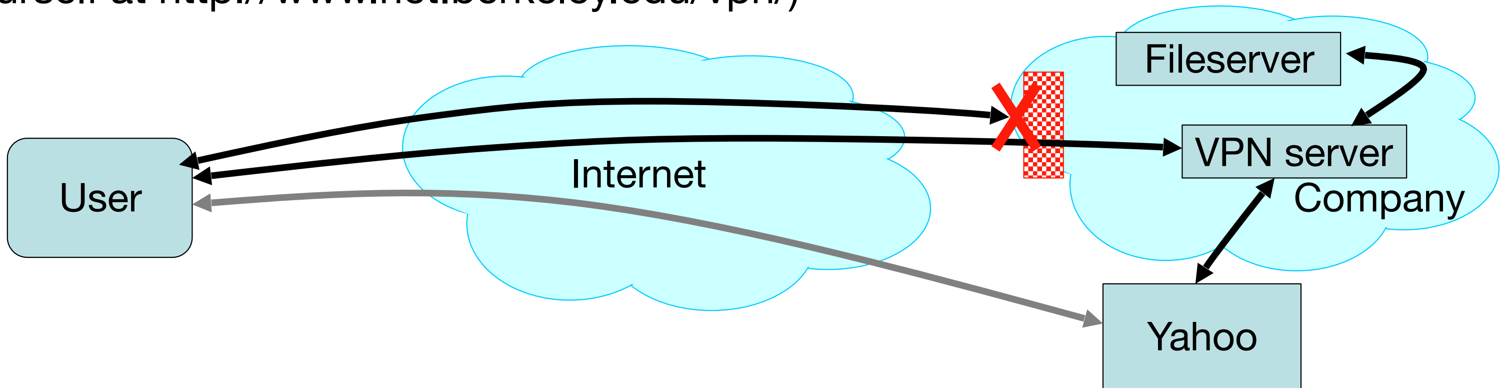
Assume firewall is 15 hops away

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

And the NAT: Network Address Translation...

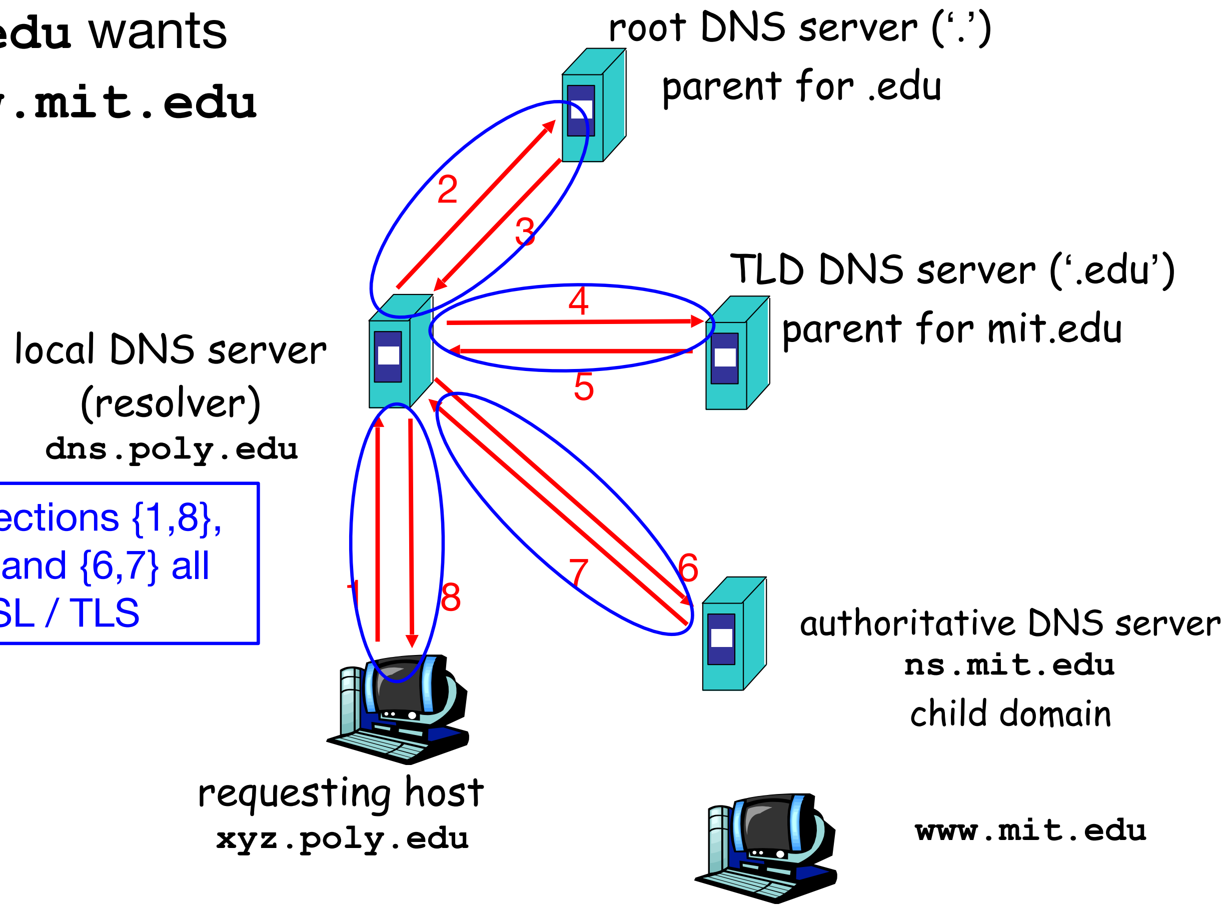
- An ISP might give us just a single IPv4 address
  - As they are expensive...
  - But you do get 2^{64} IPv6 addresses...
- So your "home gateway/home router" implements a NAT
  - Outbount request? Create an entry into a table:
    <in-IP,in-Port,Out-IP,Out-Port,Proto> -> ExteriorPort
- Now on outbound packets
  - Replace in-IP and in-Port with my IP and ExteriorPort
- And on inbound packets
  - Replace my IP and ExteriorPort with in-IP and in-Port
- By default it is a "deny all incoming" firewall...
  - Except these days, your system can ask for a reservation to allow inbound connections

A Warning For Wednesday:
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
- Note: DNSSEC is the cutoff point for MT2:
  Everything up to DNSSEC is fair game...
  DNSSEC won't hit until the final

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