

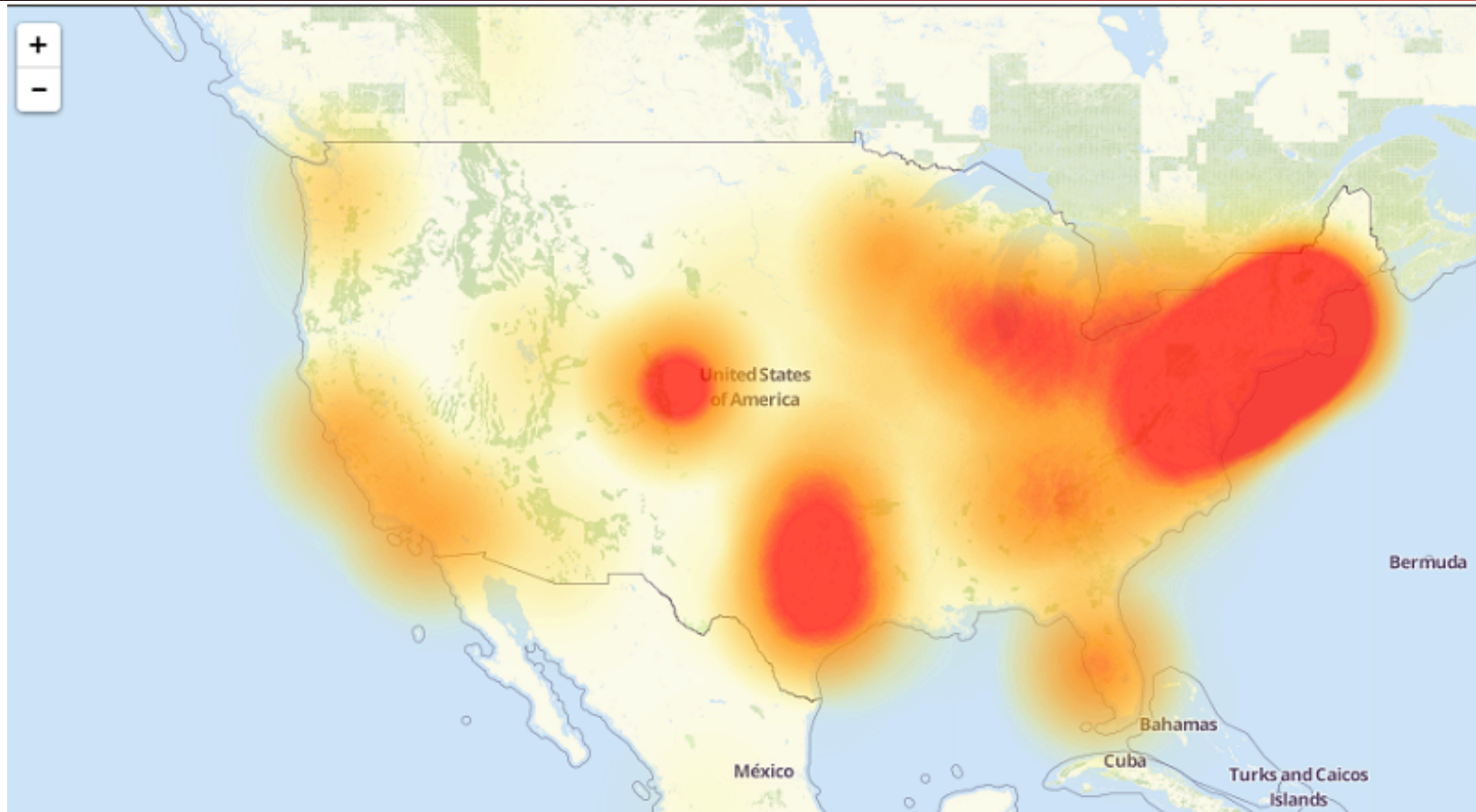
# **Network #5: Denial of Service and Firewalls (Most Slides stolen from Dave Wagner)**

# Theme of This Lecture:

## Why Twitter & Reddit Sucked Last Week

Computer Science 161 Fall 2016

Popa and Weaver



# Theme of This Lecture In Song: 50 Whys to Stop A Server...

Computer Science 161 Fall 2016

Popa and Weaver

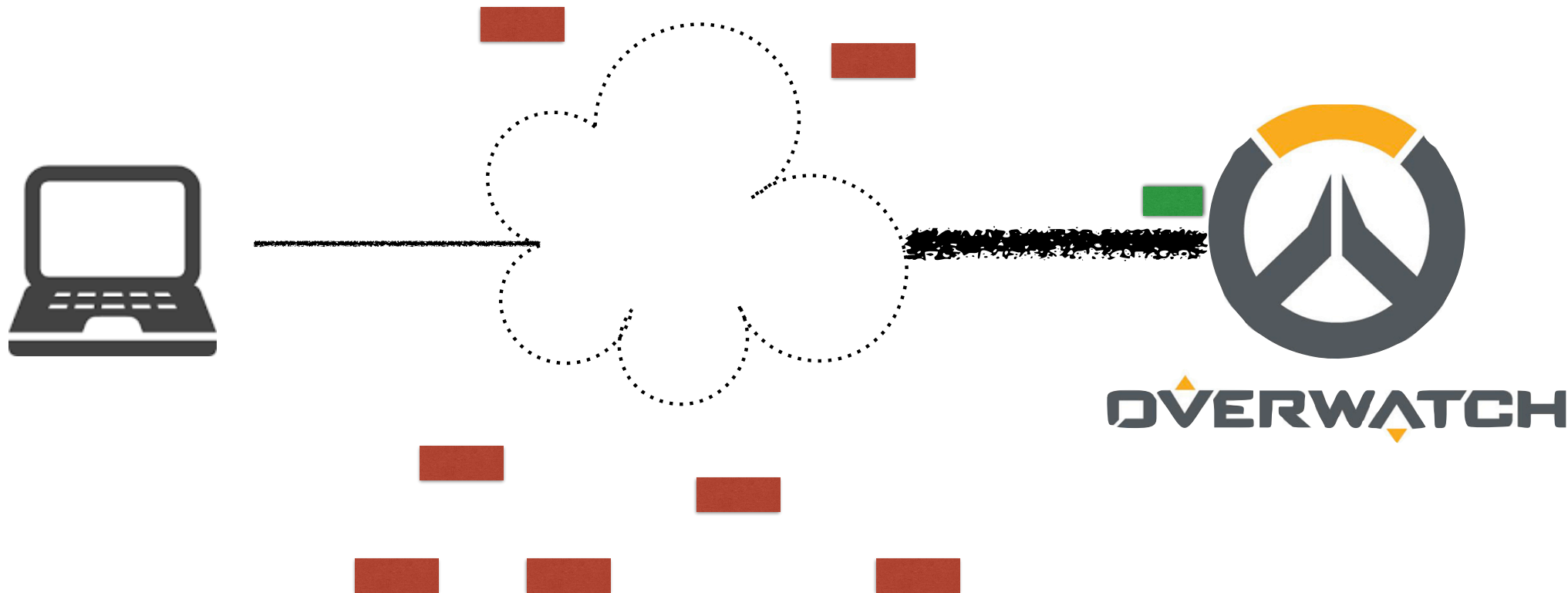
- You are a bad guy...
  - And you want to stop some server from being **available**
- Why? You name it...
  - Because its hard for someone to frag you in an online game if you "boot" him from the network
  - Because people will pay up to stop the attack
  - Because it conveys a political message
  - Get paid for by others



# The Easy DoS on a System: Resource Consumption...

- Bad Dude has an account on your computer...
  - And wants to disrupt your work on Project 2...
- He runs this simple program:
  - while(1):
    - Write random junk to random files
      - (uses disk space, thrashes the disk)
    - Allocate a bunch of RAM and write to it
      - (uses memory)
    - fork()
      - (creates more processes to run)
- Only defense is some form of quota or limits:  
The system itself ***must*** enforce some isolation

# The Network DOS



# Or, another visual explanation...

- <https://twitter.com/kokonoe0825/status/789536739887112192>

# DoS & Networks

- How could you DoS a target's Internet access?
  - Send a zillion packets at them
  - Internet lacks **isolation** between traffic of different users!
- What resources does attacker need to pull this off?
  - At least as much sending capacity (**bandwidth**) as the bottleneck link of the target's Internet connection
    - Attacker sends maximum-sized packets
  - Or: overwhelm the rate at which the bottleneck router can process packets
    - Attacker sends minimum-sized packets!
      - (in order to maximize the packet arrival rate)

# Defending Against Network DoS

- Suppose an attacker has access to a beefy system with high-speed Internet access (a “big pipe”).
- They pump out packets towards the target at a very high rate.
- What might the target do to defend against the onslaught?
  - Install a network filter to discard any packets that arrive with attacker’s IP address as their source
    - E.g., `drop * 66.31.33.7:* -> *:*`
    - Or it can leverage any other pattern in the flooding traffic that’s not in benign traffic
  - Attacker’s IP address = means of identifying misbehaving user



# Filtering Sounds Pretty Easy ...

- ... but DoS filters can be easily evaded:
  - Make traffic appear as though it's from many hosts
    - Spoof the source address so it can't be used to filter
      - Just pick a random 32-bit number of each packet sent
    - How does a defender filter this?
      - They don't!
      - Best they can hope for is that operators around the world implement anti-spoofing mechanisms (today about 75% do)
  - Use many hosts to send traffic rather than just one
    - Distributed Denial-of-Service = DDoS (“dee-doss”)
    - Requires defender to install complex filters
    - How many hosts is “enough” for the attacker?
      - Today they are very cheap to acquire ... :-(

# It's Not A “Level Playing Field”

- When defending resources from exhaustion, need to beware of asymmetries, where attackers can consume victim resources with little comparable effort
  - Makes DoS easier to launch
  - Defense costs much more than attack
- Particularly dangerous form of asymmetry: amplification
  - Attacker leverages system's own structure to pump up the load they induce on a resource

# Amplification

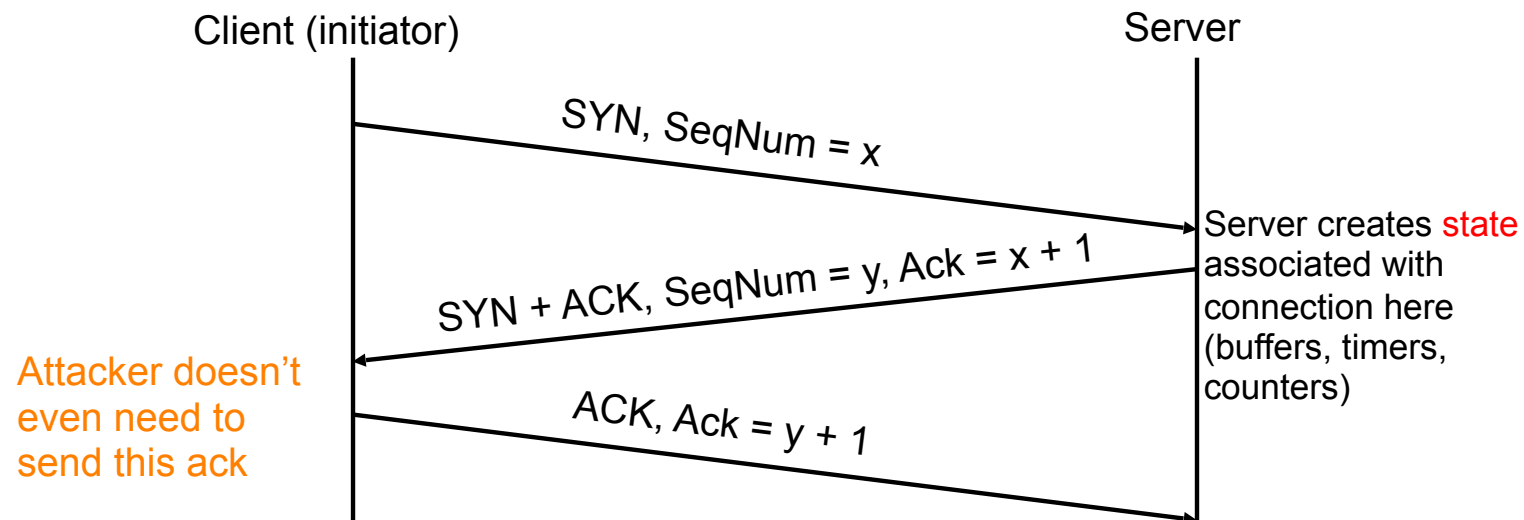
- Example of amplification: DNS lookups
  - Reply is generally much bigger than request
    - Since it includes a copy of the reply, plus answers etc.
  - Attacker spoofs DNS request to a patsy DNS server, seemingly from the target
    - Small attacker packet yields large flooding packet
    - Doesn't increase # of packets, but total volume
- Note #1: these examples involve blind spoofing
  - So for network-layer flooding, generally only works for UDP-based protocols (can't establish a TCP connection)
- Note #2: victim doesn't see spoofed source addresses
  - Addresses are those of actual intermediary systems

# Botnets

- If an attacker can control a *lot* of systems
  - They gain a huge amount of bandwidth
    - Modern DOS attacks approach 1 Terabit-per-second with direct connections
  - And it becomes very hard to filter them out
    - How do you specify 1M machines you want to ignore?
- You control these "bots" in a "botnet"
  - So you can issue commands that cause all these systems to do what you want
- This is what took down dyn DNS (and with it Twitter, Reddit, etc...): A botnet composed primarily of compromised cameras and DVRs

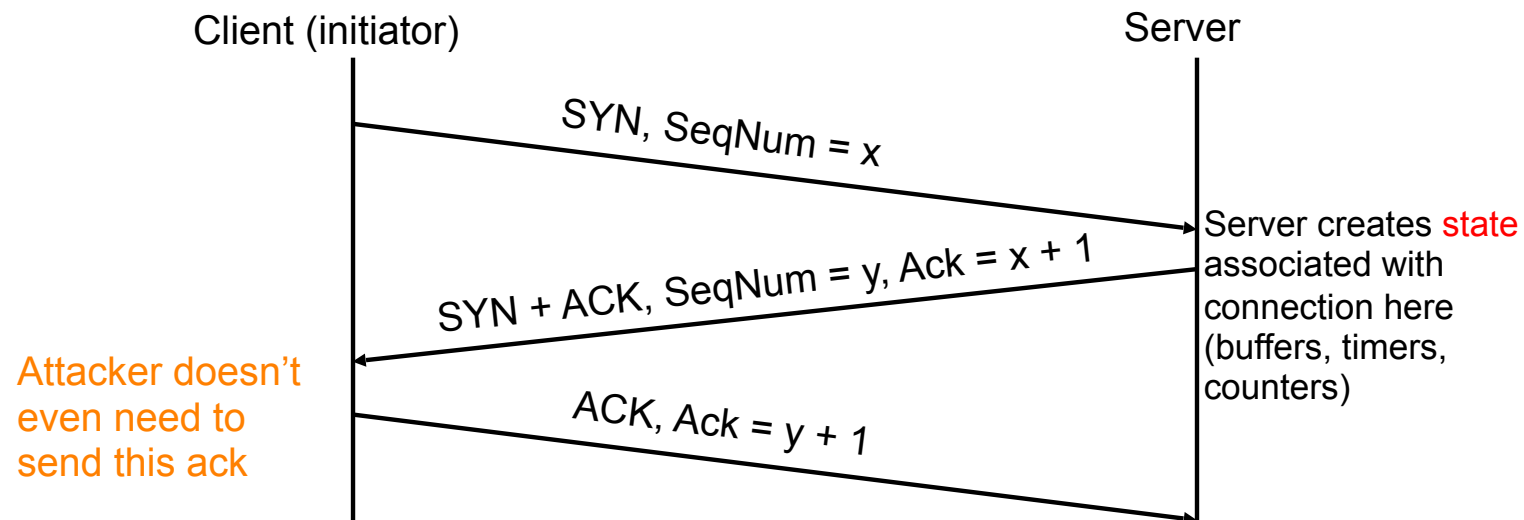
# Transport-Level Denial-of-Service

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



# 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



# 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

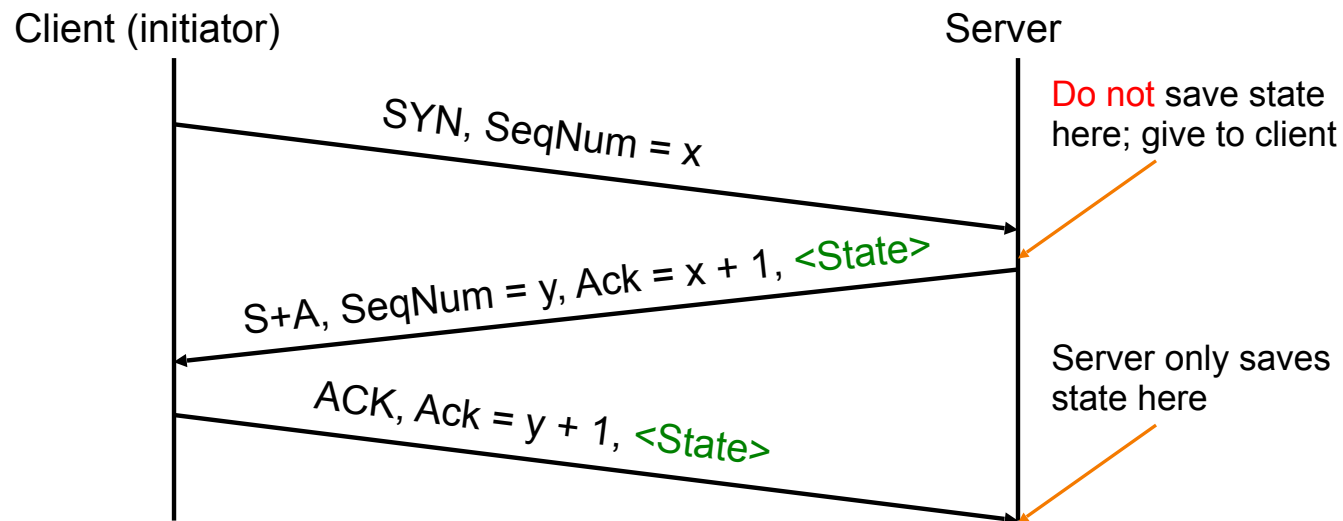


# TCP SYN Flooding Defenses

- 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 establish connection



# SYN Flooding Defense: Idealized

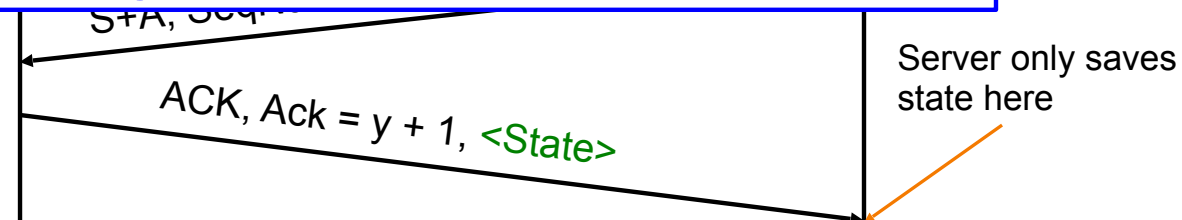
- Server: when SYN arrives, rather than keeping state locally, send it to the client.
- Client needs to establish connection

Problem: the world isn't so ideal!

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

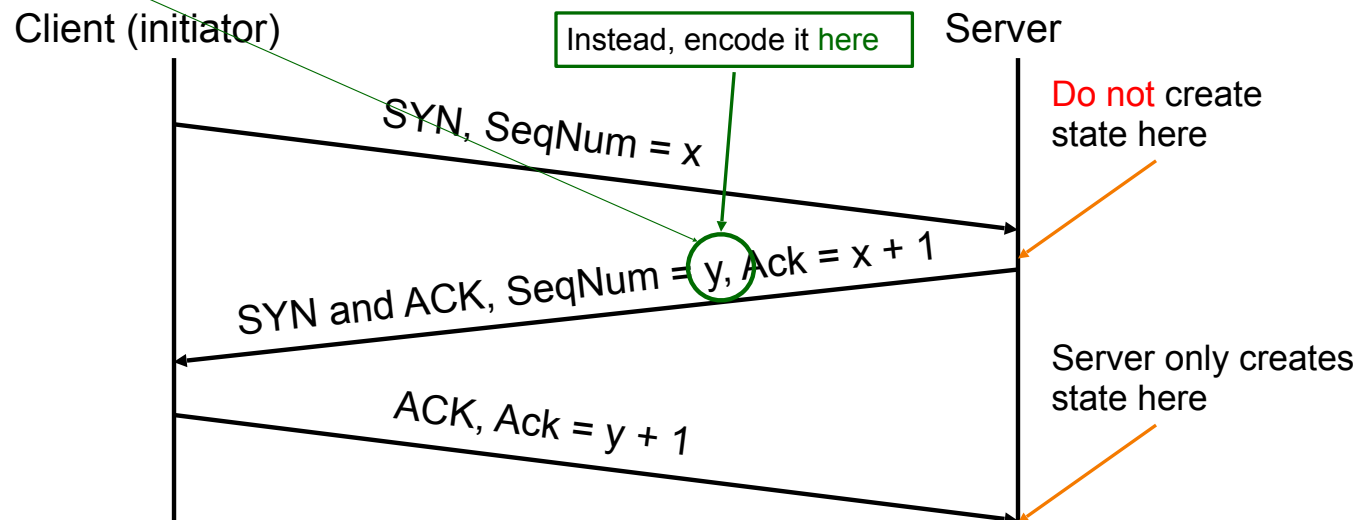
Is there any way to get the same functionality without having to change TCP clients?

Client



# 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
- Note #2: if it's expensive to generate or check the cookie, then it's not a win

# 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)

# 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 or memory overload. We have currently identified several routes for a more sustainable medium-term fix and have developers working on implementation.

It is highly recommended to switch to Parity mining. Use these settings:

# 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  $\Pr[\text{hk}(x)=\text{hk}(y)] = .5^b$ , 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

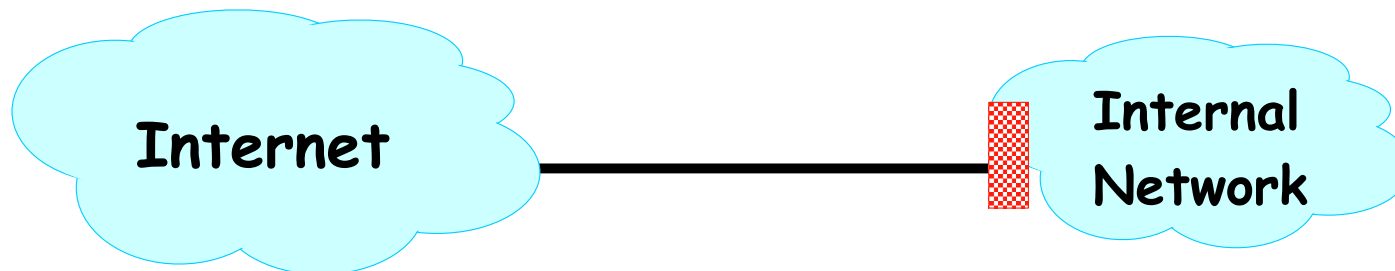
- Defending against program flaws requires:
  - Careful design and coding/testing/review
  - Consideration of behavior of defense mechanisms
    - E.g. buffer overflow detector that when triggered halts execution to prevent code injection ⇒ denial-of-service
- Defending resources from exhaustion can 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 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

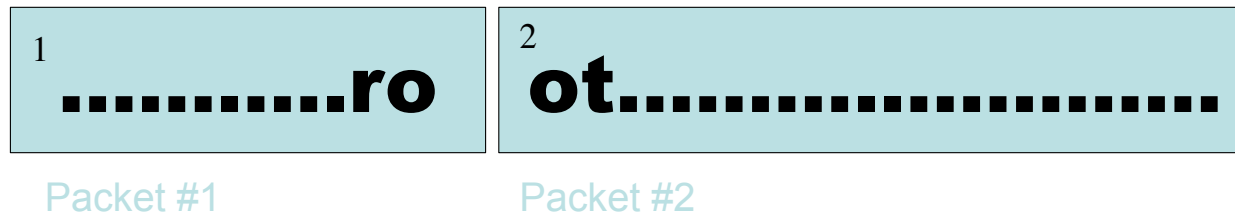
- Suppose you want to allow inbound connection to a FTP server, but block any attempts to login as “root”. How would you build a stateful packet filter to do that? In particular, what state would it keep, for each connection?

# 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

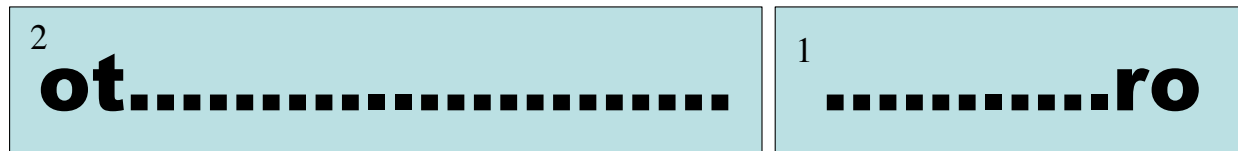
# Beware!

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

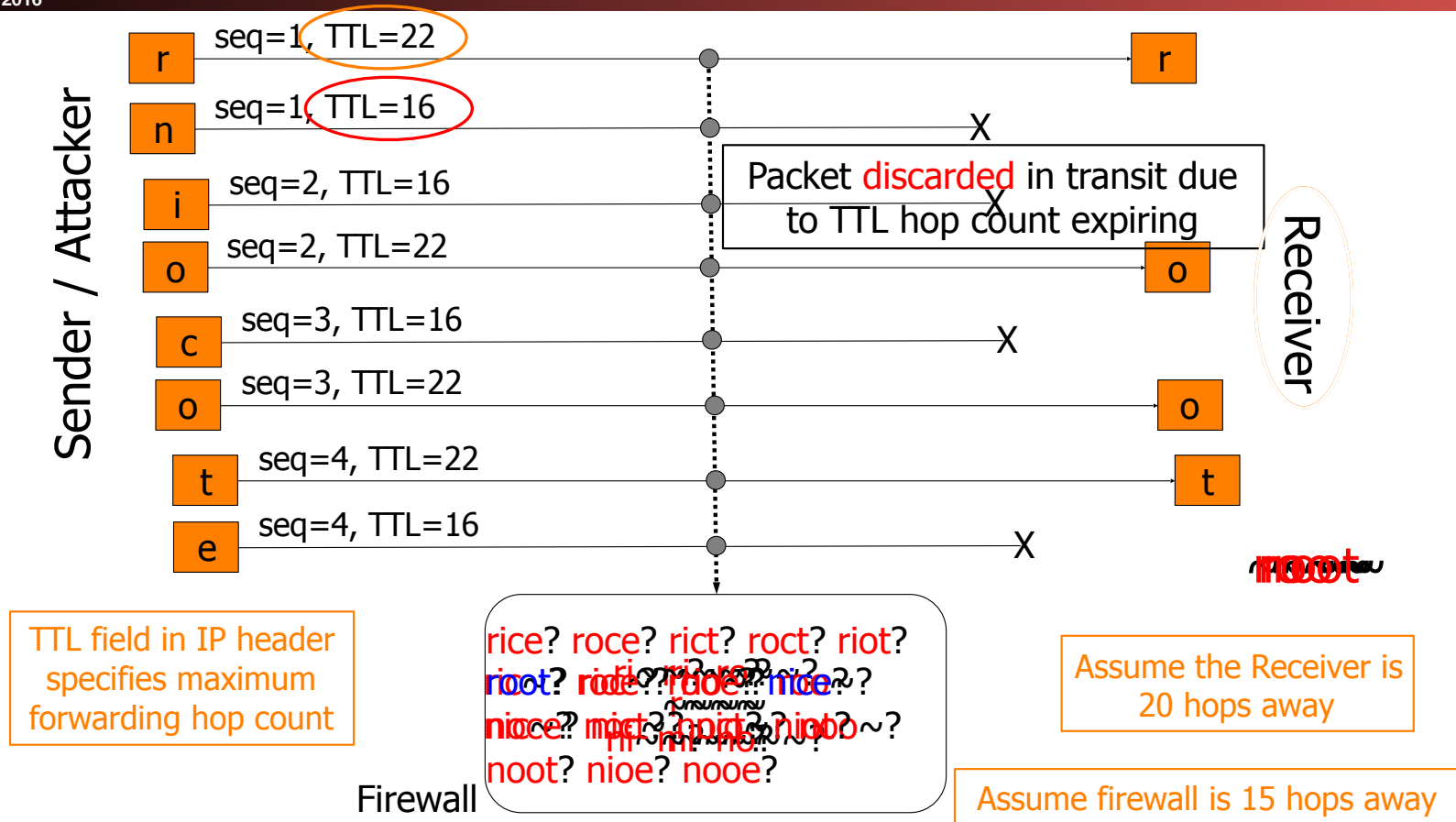


# Beware!

- Packets might be re-ordered



# Beware!



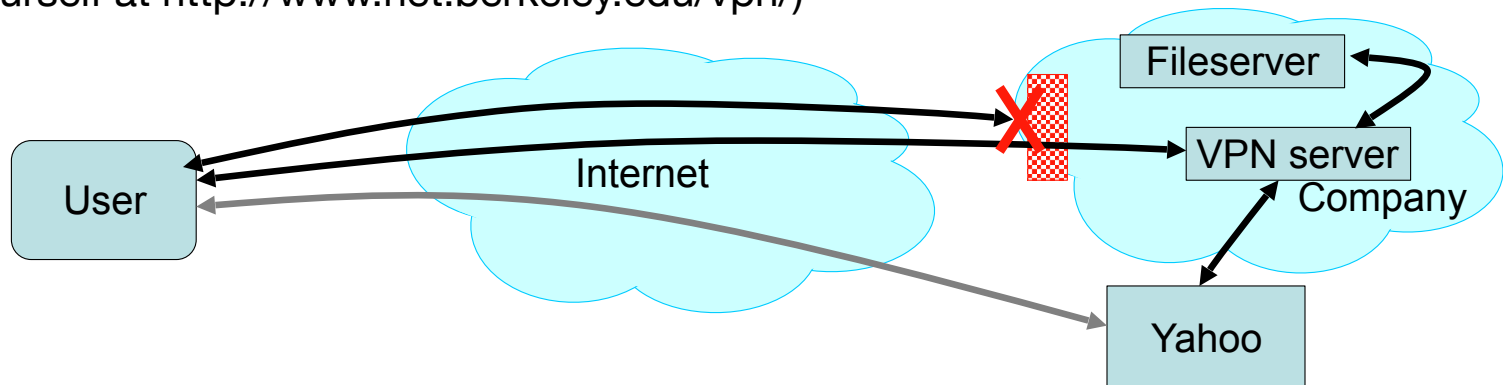


# Other Kinds of Firewalls

- 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