Denial of Service and Firewalls

CS 161 Fall 2022 - Lecture 22

Last Time: DNSSEC

- DNSSEC: An extension of the DNS protocol that ensures integrity on the results
  - Provides object security (unlike DNS over TLS, which would provide channel security)
  - Uses signatures to cryptographically verify records
  - Uses a hierarchical public key infrastructure to delegate trust from the trust anchor (root)
- DNSSEC Implementation
  - Each name server replies with its public key (**`DNSKEY`** type)
  - When delegating trust, each name server signs the public key of the next name server (**`DS`** and **`RRSIG`** types)
  - When providing a final answer, the name server signs the final answer (**`RRSIG`** type)
  - Zones are split into key-signing keys and zone-signing keys
  - NSEC signs a message saying no domains exist alphabetically between two records

Today: Denial of Service and Firewalls

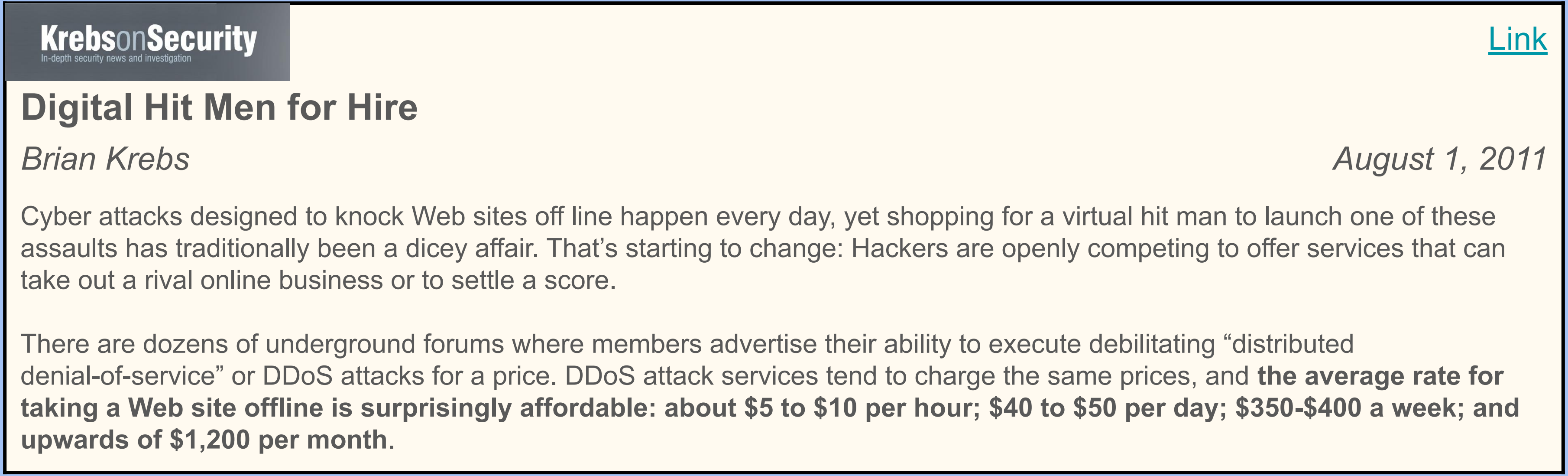
- Denial of service
    - Availability
    - Application-level DoS
        - Algorithmic complexity attacks
    - Network-level DoS
        - Distributed DoS (DDoS)
        - Amplified DoS
    - SYN flooding
        - SYN cookies
    - Defenses
- Firewalls
    - Packet filters
        - Stateless/stateful packet filters
    - Proxy firewalls

Denial of Service (DoS)

Availability and Denial of Service (DoS)

- **Availability**: Making a service on the network available for legitimate users
- **Denial of service** (**DoS**): An attack that disrupts availability of a service, making it unavailable for legitimate users
    - Reasons for a DoS attack
        - Competitors might DoS each other to benefit their own services
        - Criminals might DoS services unless the services pay a ransom
        - People might DoS services to make a political statement
        - Entities might DoS each other as part of warfare tactics
        - Some people might DoS for fun or revenge (e.g. online games)

DoS in the News



KrebsonSecurity
In-depth security news and investigation

Link

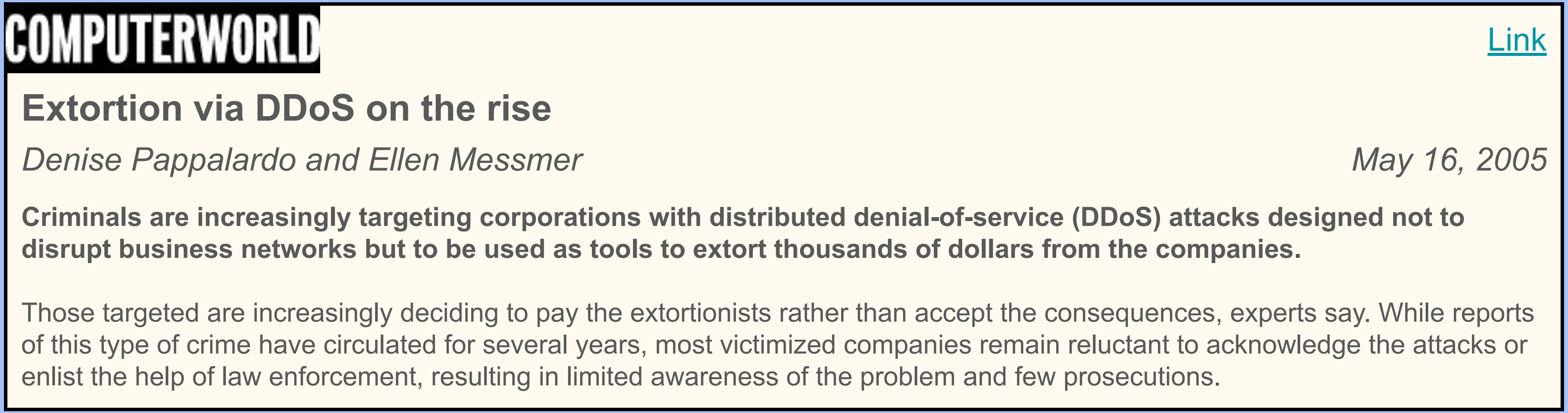
Digital Hit Men for Hire

Brian Krebs *August 1, 2011*

Cyber attacks designed to knock Web sites off line happen every day, yet shopping for a virtual hit man to launch one of these assaults has traditionally been a dicey affair. That’s starting to change: Hackers are openly competing to offer services that can take out a rival online business or to settle a score.

There are dozens of underground forums where members advertise their ability to execute debilitating “distributed denial-of-service” or DDoS attacks for a price. DDoS attack services tend to charge the same prices, and **the average rate for taking a Web site offline is surprisingly affordable: about $5 to $10 per hour; $40 to $50 per day; $350-$400 a week; and upwards of $1,200 per month**.

DoS in the News



COMPUTERWORLD

Link

Extortion via DDoS on the rise

Denise Pappalardo and Ellen Messmer *May 16, 2005*

Criminals are increasingly targeting corporations with distributed denial-of-service (DDoS) attacks designed not to disrupt business networks but to be used as tools to extort thousands of dollars from the companies.

Those targeted are increasingly deciding to pay the extortionists rather than accept the consequences, experts say. While reports of this type of crime have circulated for several years, most victimized companies remain reluctant to acknowledge the attacks or enlist the help of law enforcement, resulting in limited awareness of the problem and few prosecutions.

DoS in the News

ZDNet

Link

DDoS makes a phishing e-mail look real

Munir Kotadia

November 8, 2006

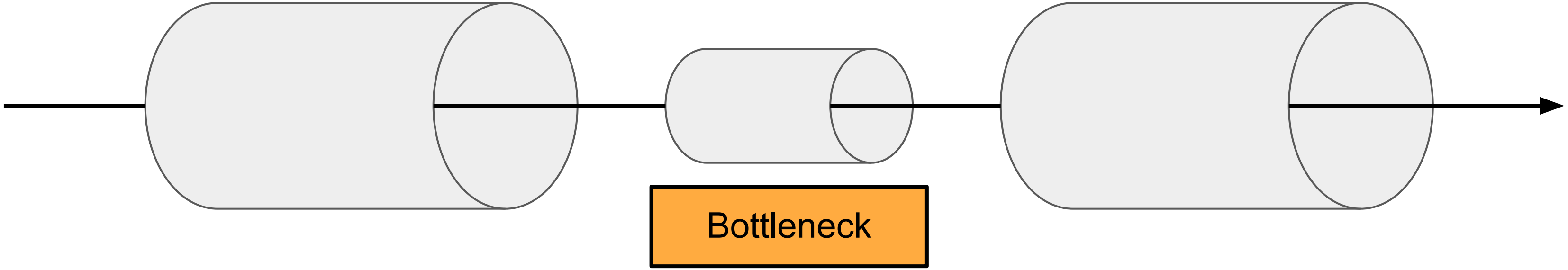
Just as Internet users learn that clicking on a link in an e-mail purporting to come from their bank is a bad idea, phishers seem to be developing a new tactic -- launch a DDoS attack on the Web site of the company whose customers they are targeting and then send e-mails "explaining" the outage and offering an "alternative" URL.

DoS Attacks: Strategies

- Exploiting program flaws
  - Software vulnerabilities can cause a service to go offline
  - Example: Exploit a buffer overflow to execute a shutdown command to the system
  - Example: Exploit a SQL injection vulnerability to delete the database
- Resource exhaustion
  - Everything on the network has limited resources
  - The attacker consumes all the limited resources so legitimate users can’t use them

DoS Attacks: Strategies

- Bottlenecks
  - Different parts of the system might have different resource limits
  - The attacker only needs to exhaust the **bottleneck**: the part of the system with the least resources



DoS Targets

- **Application-level DoS**: Target the high-level application running on the host
- **Network-level DoS**: Target network protocols to affect the host’s Internet

Application-Level DoS

Application-Level DoS

- Target the resources that the application uses
- Exploit features of the application itself
- Some attacks rely on **asymmetry**: A small amount of input from the attack results in a large amount of consumed resources!

Resource Consumption

- Idea: Force the server to consume all its resources

```
int fd = open('/tmp/junk');
char buf[4096]
while (1) { write(fd, buf, 4096) };
```

Exhausts filesystem space

```
while (1) { malloc(1000000000); }
```

Exhausts RAM

```
while (1) { fork(); }
```

Exhausts processing threads

```
while (1) {
    int fd = open(random_file());
    write(fd, "abcde", 5);
    close(fd);
}
```

Exhausts disk I/O operations

Algorithmic Complexity Attacks

- Consider an application that runs a sort on user-chosen data
    - What if the attacker intentionally chooses inputs that cause the worst-time runtime to occur?
- **Algorithmic complexity attack**: Supplying inputs that trigger worst-case complexity of algorithms and data structures
    - Defense: Choose algorithms with good worst-case running times
        - Mergesort is safer than quicksort against DoS!

	Expected runtime	Worst-case runtime
Mergesort	O(*n* log *n*)	O(*n* log *n*)
Quicksort	O(*n* log *n*)	$O(n^2)$

Application-Level DoS: Defenses

- **Identification**: Step 0 of any defense
  - You must be able to distinguish requests from different users before you can do anything else!
  - Requires some method to identify/authenticate users
  - Authenticating users might be expensive and itself vulnerable to DoS
- **Isolation**: Ensure that one user’s actions do not affect another user’s experience
- **Quotas**: Ensure that users can only access a certain proportion resources
  - Example: Only trusted users can execute expensive requests
  - Example: Limit each user to 4 GB of RAM and 2 CPU cores

Application-Level DoS: Defenses

- **Proof-of-work**: Force users to spend some resources to issue a request
  - Idea: Make a DoS attack more expensive for the attacker, who now needs to spend resources
  - Example: Add a CAPTCHA, which the attacker will now have to solve (or pay for solving services)
- **Overprovisioning**: Allocate a huge amount of resources
  - Can cost the server a lot of money!
  - Depends on your threat model
  - Often the most effective defense (“security is economics”)
  - **Content delivery network** (CDN): A service that allocates a huge amount of resources for you
    - Example of a CDN: Cloudflare
    - Cloudflare runs your service for you with a huge amount of resources

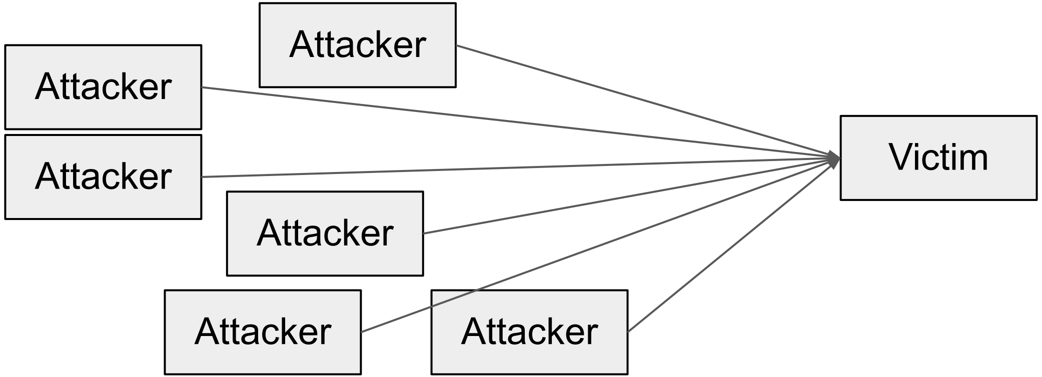
Network-Level DoS

Network-Level DoS

- Approaches target network protocols to affect the victim’s Internet access
  - Example: Send a huge amount of packets to the victim
- Overwhelm the victim’s **bandwidth** (amount of data it can upload/download in a given time)
  - Example: The server can only upload/download 10 MB/s. The attacker sends the server 20 MB/s.
    - Lots of maximum-sized packets
- Overwhelm the victim’s **packet processing capacity**
  - Example: The server can process 10 packets/second. The attacker sends the server 20 packets/second.
    - Lots of minimum-sized packets

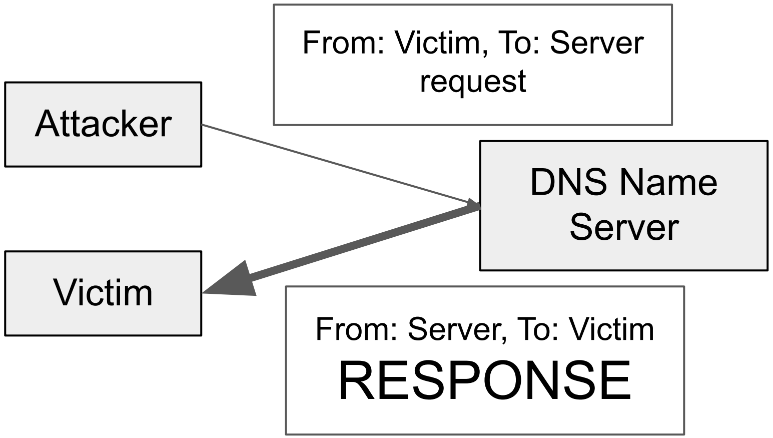
Distributed Denial-of-Service (DDoS)

- **Distributed denial-of-service** (**DDoS**): Use multiple systems to overwhelm the target system
  - Controlling many systems gives the attacker a huge amount of bandwidth
  - Sending packets from many sources makes it hard for packet filters to distinguish DDoS traffic from normal traffic
  - **Botnet**: A collection of compromised computers controlled by one attacker
    - The attacker can tell all the computers on the botnet to flood a given target



Amplified Denial-of-Service

- **Amplified denial-of-service**: Use an amplifier to overwhelm the target more effectively
  - Idea: Some services send a large response when sent a small request
  - Spoofing a small request that appears to come from the victim results in a large amount of data sent to the victim
  - Example: DNS amplification
    - Requests contain only the question
    - Responses contain answer records, authority records, and additional records



Amplified Denial-of-Service

- Benefits:
  - The attacker's identity is concealed because the packets come from the amplification server
  - The attacker is able to overwhelm more bandwidth with relatively little bandwidth
    - Amplification servers often have massive bandwidths to support large numbers of users
- Drawbacks:
  - Requires blind spoofing capability
    - Cannot work over TCP, since TCP spoofing is assumed to be hard, only UDP protocols

Network-Level DoS: Defenses

- **Packet filter**: Discard any packets that are part of the DoS attack
  - Discard packets where the source IP is the attacker's IP address
  - Find some pattern in the content of the DoS packets to distinguish DoS packets from legitimate packets
  - The packet filter must be before the bottleneck
- Subverting packet filters
  - Spoof DoS packets so that packets look like they're coming from many IP addresses
    - Packet filters can't use IP addresses to filter packets anymore!
    - Hard to defend against
    - Rely on anti-spoofing mechanisms on the network
  - Distributed DoS actually send packets from many IP addresses
    - Packet filters need to be much more sophisticated to defend against DDoS attacks

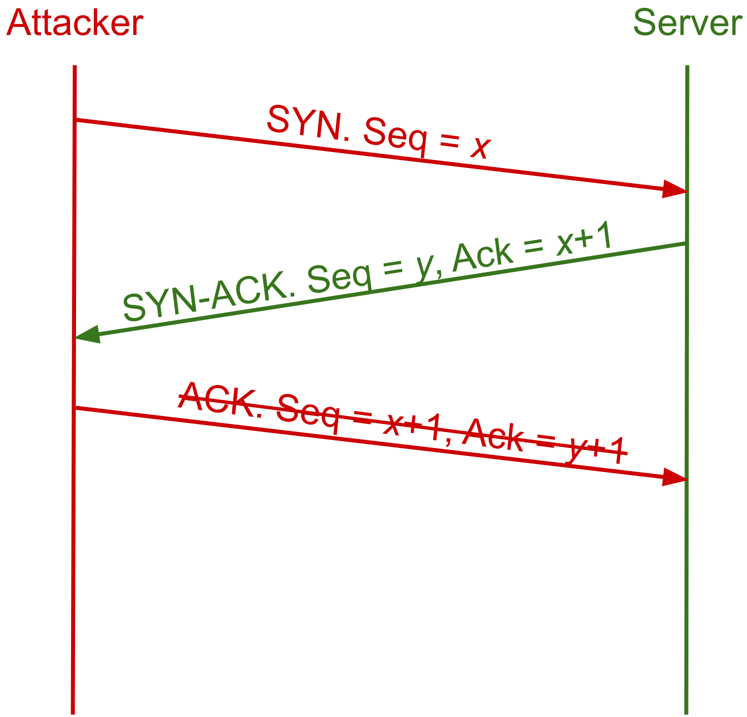
Network-Level DoS: Defenses

- **Overprovisioning**: Purchase enough networking bandwidth and equipment to make it harder for attackers to overwhelm the network
  - Again, depends on your threat model

SYN Flooding and SYN Cookies

SYN Flooding

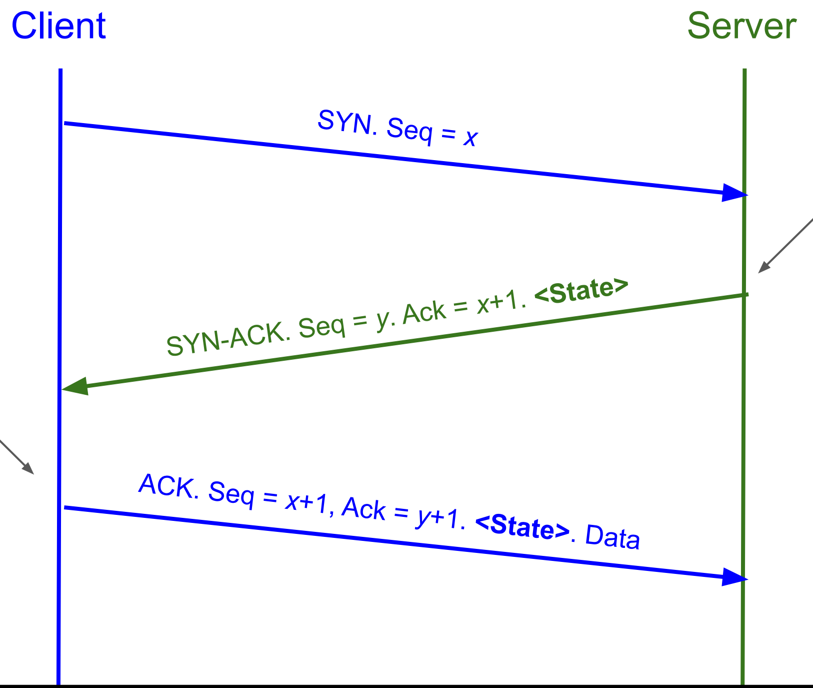
- A type of DoS that exploits many TCP connections
  - Each connection established by the server needs to allocate some memory
    - Used to store sequence numbers, ACK numbers, buffered data, etc.
  - Idea: Establish many connections with the server, causing it to consume a lot of memory
- TCP state is allocated upon receiving a SYN
  - The attacker only needs to send the SYN, so the attacker doesn’t its own consume resources!


Attacker
Server
SYN. Seq = *x*
SYN-ACK. Seq = *y*, Ack = *x*+1
~~ACK. Seq = *x*+1, Ack = *y*+1~~

SYN Flooding: Defenses

- **Overprovisioning**: Ensure the server has a lot of memory
  - Can be expensive and depends on your threat model
- **Filtering**: Ensure that only legitimate connections will create state
  - Same problems as standard packet filtering for network-level DoS attacks
  - Hard to distinguish legitimate traffic so early in the connection
  - Attacker can spoof source address since they only need to send the SYN, not the ACK
- **SYN cookies**: Don’t store state!
  - Relies on the client to store the server’s state
  - The client returns the state to the server in the ACK packet of the handshake

Idealized SYN Cookies


Client
Server
SYN. Seq = *x*
SYN-ACK. Seq = *y*. Ack = *x*+1. **<State>**
ACK. Seq = *x*+1, Ack = *y*+1. **<State>**. Data

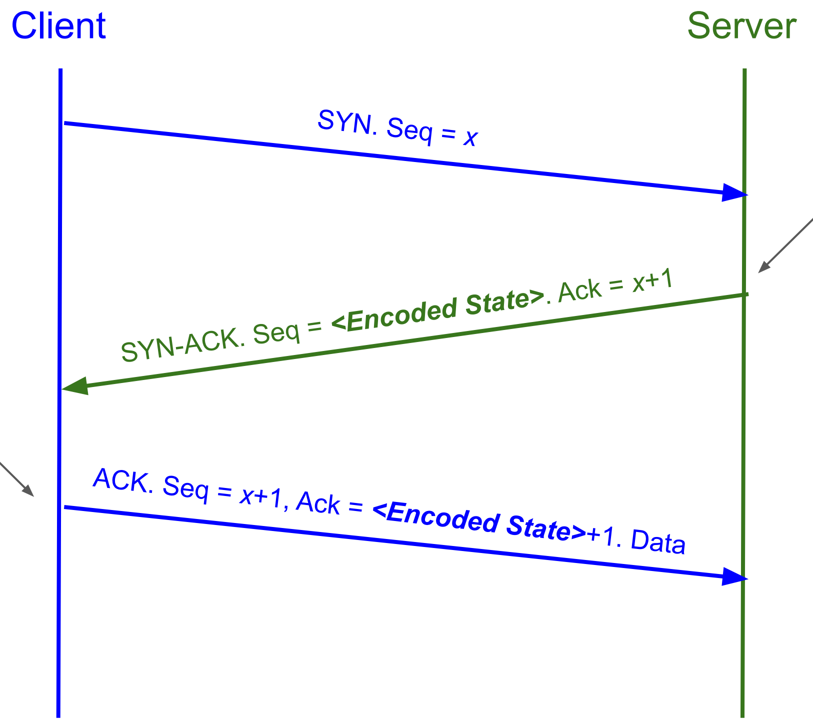
The server generates state for the client but *doesn't save it*, sending it to the client instead encoded with a secret

The client stores the state on behalf of the server and returns it in the ACK packet

Now that the handshake is complete, only now does the server allocate state for the connection, after checking the cookie against the secret

Issue: TCP doesn't have a mechanism to store state! What field of the SYN-ACK packet could we store data in?

Practical SYN Cookies


Client
Server
SYN. Seq = *x*
SYN-ACK. Seq = ***<Encoded State>***. Ack = x+1
ACK. Seq = *x*+1, Ack = ***<Encoded State>***+1. Data

The server generates state for the client but *doesn't save it*, encoding it in the sequence number with a secret

The client remembers the sequence number and returns it in the ACK number

Now that the handshake is complete, only now does the server allocate state for the connection, after checking the cookie against the secret

Practical SYN Cookies

- Observation: The server doesn’t create state until the handshake is completed, so the attacker can’t spoof source addresses
    - Filtering becomes easier with SYN cookies
- We can generalize this: Instead of holding state in the server, encode it with a secret and send it to the client, who will return it when it is next needed
    - Requires enough bits to encode the state
    - We must make sure that checking the state against the secret is inexpensive, or this becomes another DoS vector!

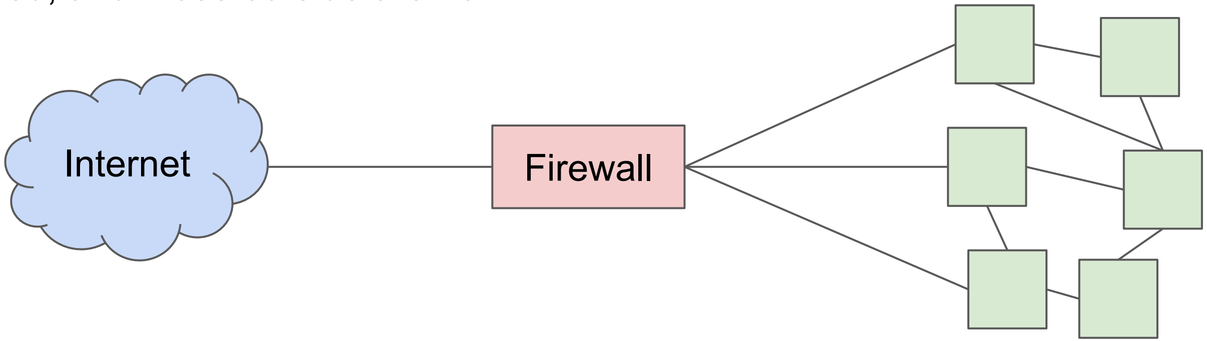
Firewalls

Motivation: Scalable Defenses

- How do you protect a set of systems against external attack?
  - Example: A company network with many servers and employee computers
- Observation: More network services = more risk
  - Each network connection creates more opportunities for attacks (greater attack surface)
  - Turning off all network services is often infeasible (print services, SSH services, etc.)
- Observation: More networked machines = more risk
  - What if you have to secure hundreds of systems?
  - What if the systems have different hardware, operating systems, and users?
  - What if there are some systems in the network that you aren’t aware of?
- Instead of securing individual machines, we want to secure the entire network!

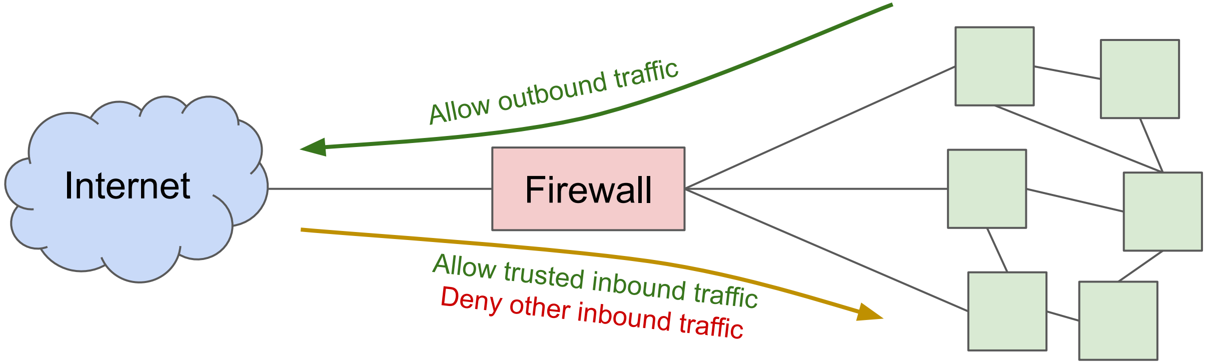
Firewalls and Security Policies

- Idea: Add a single point of access in and out of the network, with a monitor
  - "Ensure complete mediation"
  - Any traffic that could affect vulnerable systems must pass through the firewall
- Network access is controlled by a **policy**
  - Defines what traffic is allowed to exit the network (**outbound policy**)
  - Defines what traffic is allowed to enter the network (**inbound policy**)
  - Policy model based on our threat model: We usually assume users "inside" the network are trusted, and those outside are not



Firewalls and Security Policies

- What's the policy of a standard home network?
  - Outbound policy: Allow outbound traffic
    - Users inside the network can connect to any service
  - Inbound policy: Only some traffic is able to enter the network
    - Allow inbound traffic in response an outbound connection
    - Allow inbound traffic to certain, trusted services (e.g. SSH)
    - Deny all other inbound traffic



Default Security Policies?

- How should we handle traffic that isn’t explicitly allowed or denied?
  - **Default-allow policy**: Allow all traffic, but deny those on a specified **denylist**
    - As problems arise, add them to the denylist
  - **Default-deny policy**: Deny all traffic, but allow those on a specified **allowlist**
    - As needs arise (or users complain), add them to the allowlist?
- Which default policy is best?
  - Default-allow is more flexible, but flaws are vulnerabilities and can be catastrophic
  - Default-deny is more conservative, but flaws are less painful
  - Default-deny is generally accepted to be the best default policy (“consider fail-safe defaults”)

Stateless Packet Filters

- Firewalls are often **packet filters**, which inspect network packets and chooses what to do with them
  - Option #1: Allow the packet to pass through the firewall, forwarding it onwards
  - Option #2: Deny the packet from passing through the firewall, dropping it
- Stateless packet filters
  - Packet filters that have no history
  - All decisions must be made using only the information in the packet itself
  - Can have trouble implementing complex policies that require knowledge of history

Stateless Packet Filters

- Consider implementing the typical home network policy from earlier:
  - Allow outbound traffic
  - Allow inbound traffic in response to an outbound connection
  - Deny all other inbound traffic
- Issue: How do we know what inbound traffic is in response to an outbound connection?
  - TCP: Can be implemented with a hack
    - Allow inbound traffic with the ACK flag set
    - Deny inbound traffic without an ACK flag set
    - If the internal computer sees an ACK packet without having formed a connection, it will ignore it or send a RST
  - UDP: Impossible to implement
    - UDP “connections” are typically implemented at the application layer, so we can’t inspect much

Stateful Packet Filters

- A better idea: Keep state in the implementation of the packet filter
  - The filter keeps track of inbound/outbound connections
    - Notice: All connections have packets going in both directions, so a stateless filter could not do this
  - Rules define what connections are allowed or denied
  - Ultimately, packets are still either forwarded or dropped
- Example rules:
  - **`allow tcp connection 4.5.5.4:* -> 3.1.1.2:80`**
    - Allow connections from **`4.5.5.4`** to **`3.1.1.2`** with destination port 80
  - **`allow tcp connection *:*/int -> *:80/ext`**
    - Allow outbound connections with destination port 80
  - **`allow tcp connection *:*/int -> *:*/ext`**
    - Allow all outbound connections
  - **`allow tcp connection *:*/ext -> 1.2.2.3:80`**
    - Allow inbound connections to **`1.2.2.3`** with destination port 80

Stateful Packet Filters

- Stateful packet filters can also track the state of well-known applications
  - Example: Decoding and tracking HTTP requests/responses
  - Example: Tracking the files sent in an FTP (File Transfer Protocol) connection

State in an FTP Rule

- Consider this rule: “Allow all inbound FTP connections, except those logging in as `root`”
- What state does the packet filter have to track?
  - Source IP, destination IP, source port, destination port, etc.
  - Whether this is an FTP connection or not
  - Status of the FTP connection (what command is executed)
  - Username
    - Or just the first 5 bytes of the username…
    - Otherwise, the attacker could send a really long username and DoS the firewall
- **Takeaway**: To keep track of applications, firewalls must be smart about how they store state

Subverting Packet Filters

- Consider a simple example: Deny all connections containing the string **`root`**
  - Deny packets that contain the sequence of bytes **`r`**, **`o`**, **`o`**, and **`t`**
  - Allow all other packets

From: A	To: B
Seq = 4	
`Hello world`	

✓

From: C	To: D
Seq = 2	
`Log in`	

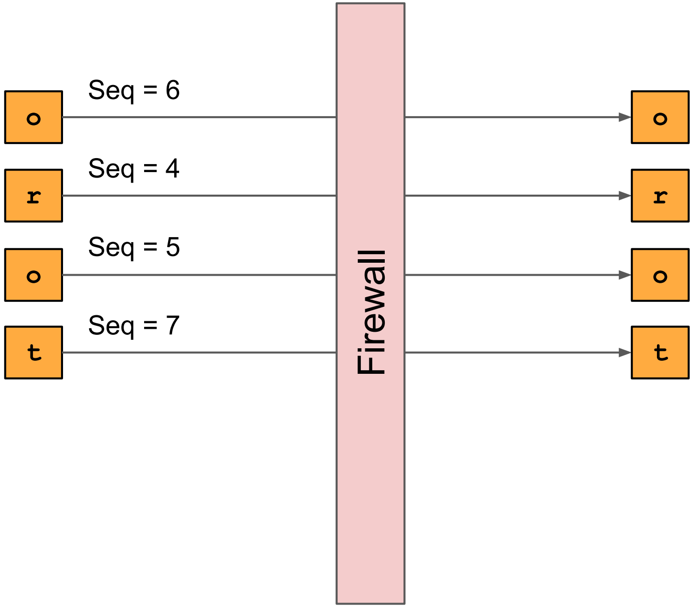
✓

From: C	To: D
Seq = 8	
` as root`	

✗

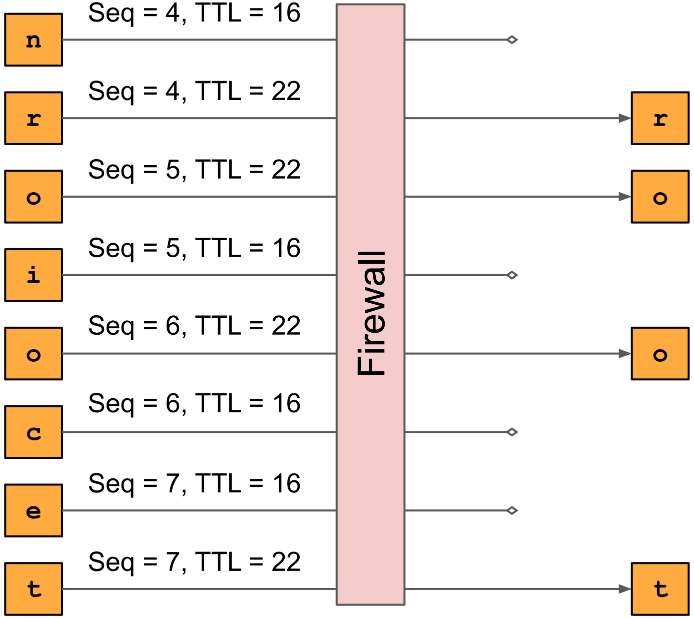
Subverting Packet Filters

- Recall TCP
  - Messages are split into packets before being sent
  - Packets can arrive out of order: The application will use sequence numbers to reorder packets
- Attack: Split the word `root` across packets
  - No single packet contains `root`, so the firewall won't stop any of these packets
- Attack: Send the split packets out of order
  - Now the firewall has to reconstruct TCP connections to detect the `root` message



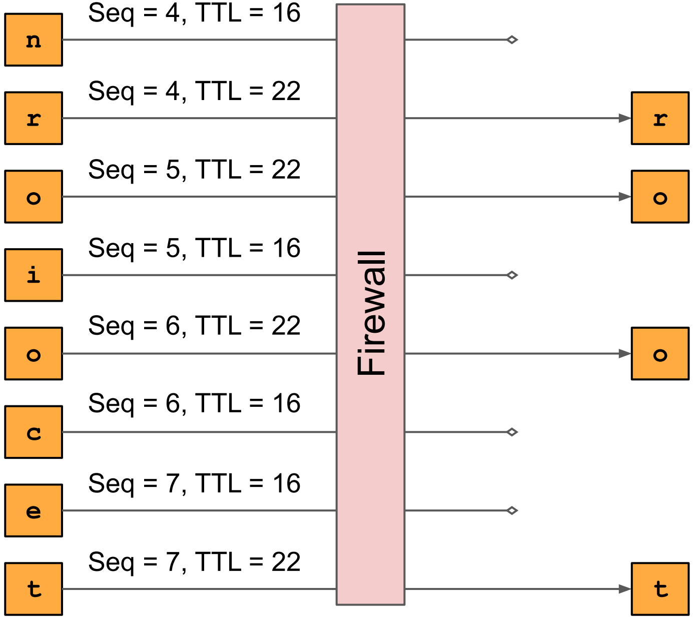
Subverting Packet Filters

- IP packets have a time-to-live (TTL)
  - The number of hops a packet may take before the packet is dropped
- The attacker can easily find how many hops away a given server is
  - Technique: Send ping packets with increasing TTLs until the server responds
- If the destination takes more hops than the firewall, the attacker can exploit this
  - Send multiple packets with the same sequence number, setting the TTLs on the dummy packets so that they are dropped before the reach the destination



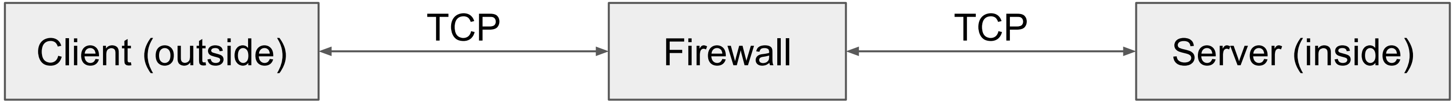
Subverting Packet Filters

- Difficult for the stateful packet filter to defend against
  - ○ TTLs for different packets naturally vary, since packets may take different routes
  - ○ Storing all possible combinations takes exponential space
  - ○ Hard to predict which packets will reach the destination and which won’t



Other Types of Firewalls

- **Proxy firewall**: Instead of forwarding packets, form two TCP connections: One with the source, and one with the destination
  - The firewall is really just a MITM, so it can easily spoof the addresses of the end hosts
  - Avoids problems with packets, since the firewall has direct access to the TCP byte streams
- **Application proxy firewall**: Certain protocols allow for proxying at the application layer
  - Example: HTTP proxies will make an HTTP request on behalf of the user then return the HTTP response to the client



Alternatives to Allowing Firewall Traffic

- **Virtual private network** (**VPN**): A set of protocols that allows direct access to an internal network via an external connection
    - Creates an encrypted tunnel to allow internal network traffic to be sent securely over the Internet
    - Intuition: The encrypted tunnel is an emulated Ethernet cable that allows you to connect “inside” the network
    - The firewall allows VPN traffic, which allows arbitrary traffic to be tunneled inside

Firewall Pros and Cons

- Pros
    - Centralized management of security policies (single point of control)
    - Transparent operation to end users
    - Mitigates security vulnerabilities on end hosts (e.g. block anything that looks like shellcode)
- Cons
    - Reduced network connectivity
        - Some applications don’t work well inside a firewall
    - Vulnerability to “insiders”
        - Employees could be bribed or threatened
        - Devices are often brought from into the network outside (e.g. cell phones, laptops)
        - Once one device is compromised, attackers can quickly spread through the network
        - Could be mitigated by layering firewalls for more sensitive devices

Summary: Denial of Service

- **Availability**: Making sure users are able to use a service
  - DoS attacks availability of services
- **Application-level DoS**: Attacks the high-level applications
  - Algorithmic complexity attacks: Attack using inputs that cause the worst-case runtime of an algorithm
  - Defense: Identification, isolation, and quotas
  - Defense: Proof of work
- **Network-level DoS**: Attacks the network of a service
  - Typically floods either the network bandwidth or the packet processing capacity
  - Distributed DoS: Use multiple computers to flood a network at the same time
  - Amplified DoS: Use an amplifier to turn a small input into a large output, spoofing packets so the reply goes to the victim
  - Defense: Packet filtering
- All DoS attacks can be defended against by overprovisioning

Summary: SYN Cookies

- **SYN flooding**: A type of DoS that causes a server to allocate state for unfinished TCP connections, upon receiving a SYN packet
    - **SYN cookies**: Instead of allocating state when receiving a SYN, send the state back to the client in the sequence number
    - The client returns the state back to the server, which it only then allocates state for

Summary: Firewalls

- **Firewalls**: Defend many devices by defending the network
  - **Security policies** dictate how traffic on the network is handled
- **Packet filters**: Choose to either forward or drop packets
  - **Stateless packet filters**: Choose depending on the packet only
  - **Stateful packet filters**: Choose depending on the packet and the history of the connection
  - Attackers can subvert packet filters by splitting key payloads or exploiting the TTL
- **Proxy firewalls**: Create a connection with both sides instead of forwarding packets