DNSSEC (continued), Denial of Service, and Firewalls

CS 161 Spring 2022 - Lecture 21

NSEC: Signing Non-Existent Domains

Nonexistent Domains

- The DNSSEC structure works great for domains which exist
  - We have signatures over records stating that they exist
- What if the user queries for a domain that **doesn't** exist?
  - Option #1: Don't authenticate nonexistent domain (NXDOMAIN) responses
    - Issue: If NXDOMAIN responses don't have to be signed, the attacker can still spoof NXDOMAIN responses and cause denial-of-service (DoS)
  - Option #2: Keep the private key in the name server itself, so it signs NXDOMAIN responses
    - Issue: Name servers have access to the private key, which is an issue if they are malicious or hacked
    - Issue: Signing in real time is slow
  - We need a way that can prove that a domain doesn't exist ahead of time

Two Different Cases

- NOERROR without an answer record
  - The name is valid for some other type, but no record exists of that type for that name
- NXDOMAIN (Non eXistent DOMAIN)
  - No records exist for that name
- Need to handle both cases

NSEC: Authenticated Denial of Existence

- Prove nonexistence of a record type (NOERROR)
  - Sign a record stating that no record of a given type exists
  - Useful for proving that a domain doesn't support DNSSEC ("No DS records exist")
- Prove nonexistence of a domain (NXDOMAIN)
  - Provide two adjacent domains alphabetically, so that you know that no domain in the middle exists
  - Example: If I query for **`nonexistent.google.com`**, I can receive a signed NSEC response saying "No domains exist between **`maps.google.com`** and **`one.google.com`**."
  - We can sign all pairs of adjacent records ahead of time and keep them as NSEC records, along with their RRSIGs

Issues with NSEC

- **Domain enumeration**: It is easy for an attacker to find every single subdomain of a domain
  - Start by querying **`a.google.com`**
  - Receive an NSEC record stating that “No domains exist between **`web.google.com`** and **`ap.google.com`**
    - Now we have learned two domain names!
  - Repeat by querying **`apa.google.com`** (alphabetically immediately after **`ap.google.com`**)
  - Receive an NSEC record stating that “No domains exist between **`ap.google.com`** and **`apps.google.com`**”
  - Repeat until you loop back around to the beginning

`web` **`ap`** **`apps`**

NSEC3: Hashed Authenticated Denial of Existence

- Idea: Instead of storing pairs of adjacent domain names, store pairs of adjacent hashes
  - Example: If I query for **`nonexistent.google.com`**, which hashes to **`d48678`**…, I receive a signed NSEC3 saying “There exist no domains which hash to values between **`c612f3`**… and **`d810de`**…

`c612f3` **`d810de`**

Issues with NSEC3

- Domain enumeration is still possible since most people choose short domain names
  - Possible to brute-force through all reasonable domain names!
  - Only prevents attackers from learning long, random domain names, which would make brute-force difficult

Issues with NSEC3

- The only real way to prevent enumeration is online signature generation with the private key
- Easiest solution: NSEC3 “white lies”
  - Take the hash of the name (e.g. **`3RL...S45`**)
  - **Dynamically** create an NSEC3 record saying “No name exists between *H*(name) - 1 and *H*(name) + 1)
    - **`3RL...S44.mydomain.com NSEC3 3RL...S46.mydomain.com`**
  - Sign that **dynamically**
    - Preventing enumeration requires online signatures!
  - Called “white lies” because this doesn’t really use NSEC3 correctly (you’re *supposed* to sign the next domain name that exists)
    - But this is backwards-compatible with NSEC3 validation!

Possibility: NSEC3-only-keys

- This is **not** done but could be...
- DNSKEY records have multiple flag bits (key types), but most are unused
  - Add a new “NSEC3 key,” which indicates that the ZSK only signs NSEC3 white lies
  - The NSEC3 key is stored online; the KSK and other ZSKs are still stored offline
  - Signatures generated by an NSEC3 key are **only** valid for NSEC3 records
- Implications:
  - Adds latency on the first NXDOMAIN from a server for a given name
    - An online signature is required, but these can be cached on the authority server
  - Theft of key has limited damage
    - MITM attacker could only sign NSEC3 white lies if the resolver respects the rules for an NSEC3 key
  - Very limited DoS opportunity
    - An attacker who compromises the NSEC3 key could forge fake NXDOMAIN responses with NSEC3 (but not other DNSSEC responses)

DNSSEC in Practice

Offline Signature Generation

- Offline signatures: The application that computes signatures is separate from the application that serves the signatures
- Benefit: Efficiency
  - Records are signed ahead of time, and the signature is stored and served on request
  - Generating a signature each time a user requests it is slow (and can lead to DoS attacks)
- Benefit: Security
  - An attacker must compromise the signature generation system (e.g. steal the private signing key) to perform an attack
  - If the signature generation system is separate from the name server, compromising the name server is not enough!
  - Redundancy: One secure signature generation system, and many *mirrored* name servers providing the same records and signatures

Efficiency: Parallelization

- Requests can be made in parallel to improve performance
  - Example: Request DNSKEY records from every name server in parallel
- Signatures can be validated in parallel
  - Example: Validate the parent's DS record while waiting for the child's DNSKEY record
- As a result, if network latency is > signature validation latency, the **only** additional latency is the final signature validation!

Implementation Errors

- Implementation errors from the name servers
  - Example: A name server claims to support DNSSEC, even though it doesn't
  - Example: Changing your key but presenting old signatures signed with an old key
  - Example: Present expired signatures
- Implementation errors from the resolvers
  - The resolver can't access DNSSEC records
  - The resolver can't process DNSSEC records correctly

Implementation Errors: Examples

- The launch of HBO Go (a streaming service) was broken for Comcast users and users using Google Public DNS
  - The DNS servers reported that they supported DNSSEC when they didn't
- Google Public DNS and Comcast provide recursive resolvers
  - When a name server messes up, Comcast and Google are often blamed
  - Fortunately, this is getting less common
- An educational network had several mirrors of a name server
  - 3 mirrors supported DNSSEC. All other mirrors didn't support DNSSEC
- Wi-Fi hotspots (e.g. at Starbucks) often proxy DNS
  - Proxy: Receive a DNS request and replace it with its own DNS request
  - The proxy often doesn't support DNSSEC
- The TAs tried to add DNSSEC to cs161.org…
  - And knocked the domain offline for a little while!

Implementation Error: Incomplete Validation

- Most DNSSEC implementations only validate records at the recursive resolver, not the client (stub resolver)
- If the client doesn't validate records, the recursive resolver can poison the cache!
  - Recall: The recursive resolver is the biggest threat in DNS
- If the client doesn't validate records, network attackers can still poison the cache!
  - Example: An on-path attacker between the recursive resolver and the client
- Result: If the client doesn't validate records, DNSSEC provides very little practical security

Security Implication: Authenticating IPs Doesn't Help Much

- Recall: DNS/DNSSEC is used to resolve IP addresses from a domain name
  - Adversaries who can tamper with standard DNS:
    - The recursive resolver
    - MITM/in-path attacker between the recursive resolver and the authoritative nameservers
    - MITM/in-path attacker between the client and the recursive resolver
- If you authenticate the IP address, these adversaries could still tamper with your TCP/UDP connection to that IP!
  - If you don't use TLS, DNSSEC can't protect your unencrypted connection to a server
  - If you use TLS, an attacker that tampers with your DNS connection can't masquerade as the domain name anyway…

Security Implication: Authenticating IPs Doesn't Help Much

- More important use: Use DNSSEC to sign TXT (text) records
  - TXT records store values attached to the domain in the DNS itself
    - Unlike IPs, there is no follow-up connection that can be tampered with
  - Example: **DomainKeys Identified Mail** (**DKIM**): Store a public key used to sign emails from a domain in a TXT record
    - `mail._domainkey.cs161.org. TXT "v=DKIM1; h=sha256; k=rsa; p={cryptogoop}"`
    - `mail._domainkey.cs161.org. RRSIG TXT {cryptogoop}`
  - Example: Domain ownership verification: Store a one-time value as a TXT record to prove you have control over the DNS/DNSSEC ZSKs
    - `cs161.org. TXT "google-site-verification={code}"`
    - `cs161.org. RRSIG TXT {cryptogoop}`

DNSSEC: Summary

- 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

Next: Denial of Service

- Denial of service
  - Availability
  - Application-level DoS
    - Algorithmic complexity attacks
  - Network-level DoS
    - Distributed DoS (DDoS)
    - Amplified DoS
  - SYN flooding
    - SYN cookies
  - Defenses

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)
    - Some might sell the service to others (“booters/stress-testers”)

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

Link

DDoS makes a phishing e-mail look real

Munir Kotadia *November 8, 2006*

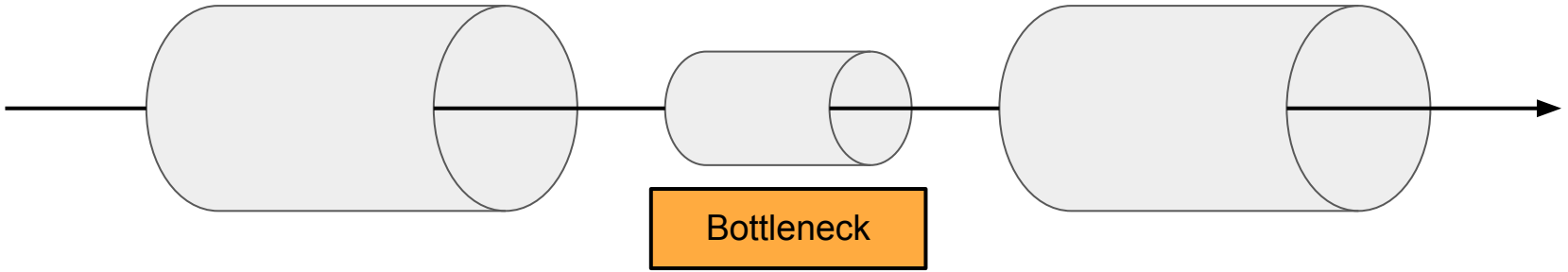
Just as Internet users learn that clicking on a link in an e-mail purporting to come from their bank is a bad idea, phishers seem to be developing a new tactic -- launch a DDoS attack on the Web site of the company whose customers they are targeting and then send e-mails "explaining" the outage and offering 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
- Recall: Memory safety mitigations
  - Examples: Stack canaries, ASLR, pointer authentication
  - Mitigations turn exploits into crashes
  - The attacker cannot execute their own shellcode anymore, but they can still crash the program
  - Exploits that crash the program can still be used for DoS attacks

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: Make sure that the worst-case runtime does not depend on user input

Algorithmic Complexity Attacks: Examples

- Most common target: Hash tables
  - Average lookup time: $O(1)$
  - Worst-case lookup time: $O(n)$, if every entry has a hash collision
  - Attack: Choose inputs that all hash to the same value
  - Defense: Use a good hash (SHA1, GHASH) and add a random salt when you create the hash table
- Quicksort
  - Average runtime: $O(n \log n)$
  - Worst-case runtime: $O(n^2)$ if you always choose a bad pivot
  - Attack: If the algorithm always uses the first element, input a sorted list (next pivot is always the next-lowest element)!
  - Defense: Choose a *random* pivot so the attacker can't choose a malicious input
- **Takeaway**: Make sure that the worst-case running time is not predictably dependant on the input!

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 the front-end (or all) of your service for you with a huge amount of resources
    - Cloudflare will present CAPTCHAs when it sees suspicious requests

Cat Break

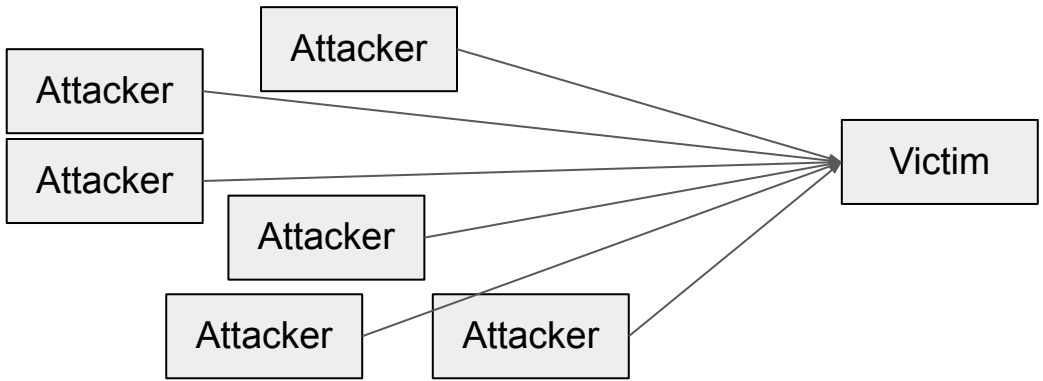
Network-Level DoS

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

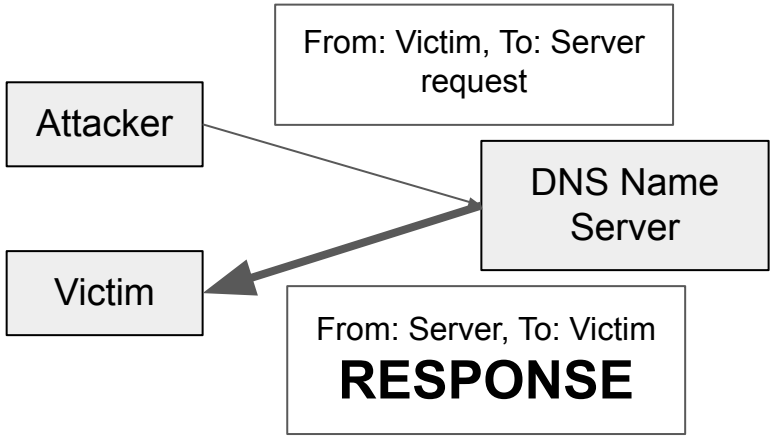
Distributed Denial-of-Service (DDoS)

- **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
    - 3/4s of ISPs limit the ability to spoof packets through **egress filtering**: Stopping outbound packets whose source IP is not from the ISP itself

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
  - Packet filter needs to be *before* the **bottleneck**

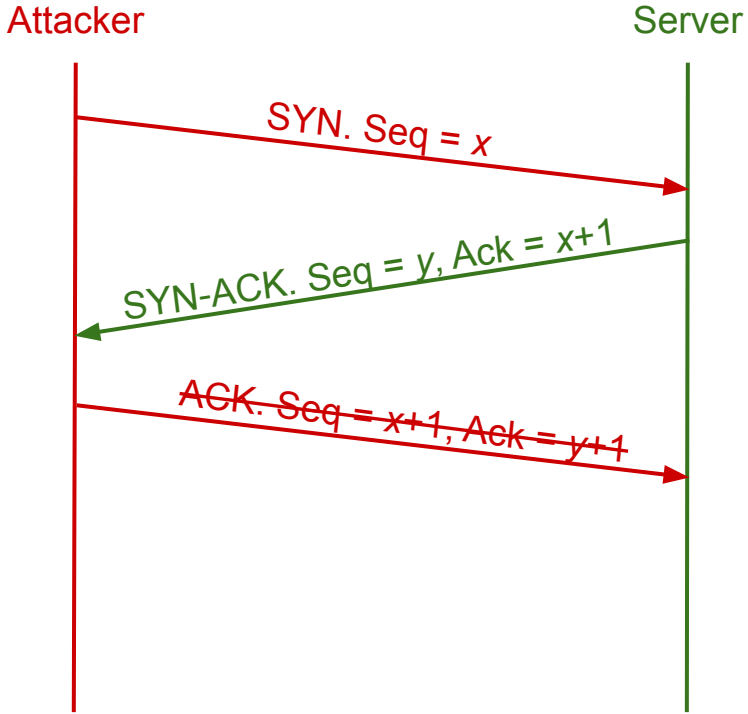
Network-Level DoS: Defenses

- **Overprovisioning**: Purchase enough networking bandwidth and equipment to make it harder for attackers to overwhelm the network
  - Again, depends on your threat model
- Overprovisioning is aided by services which do this for you…
  - 10,000 customers. But only one or two is ever “attacked” at one time
  - Being a piece of popular content (**Flash Crowd**) is often indistinguishable from an attack!
- **Content Delivery Networks** provide this:
  - Use DNS to redirect requests to “closest” CDN server
    - CDN’s authority name-servers decide based on the IP doing the query… Which usually identifies the ISP if not finer geography
  - CDN server **caches** all content possible
    - Only if not in the cache or uncacheable does it forward to the real (hidden) webserver
  - CDN can also implement filtering

SYN Flooding and SYN Cookies

SYN Flooding

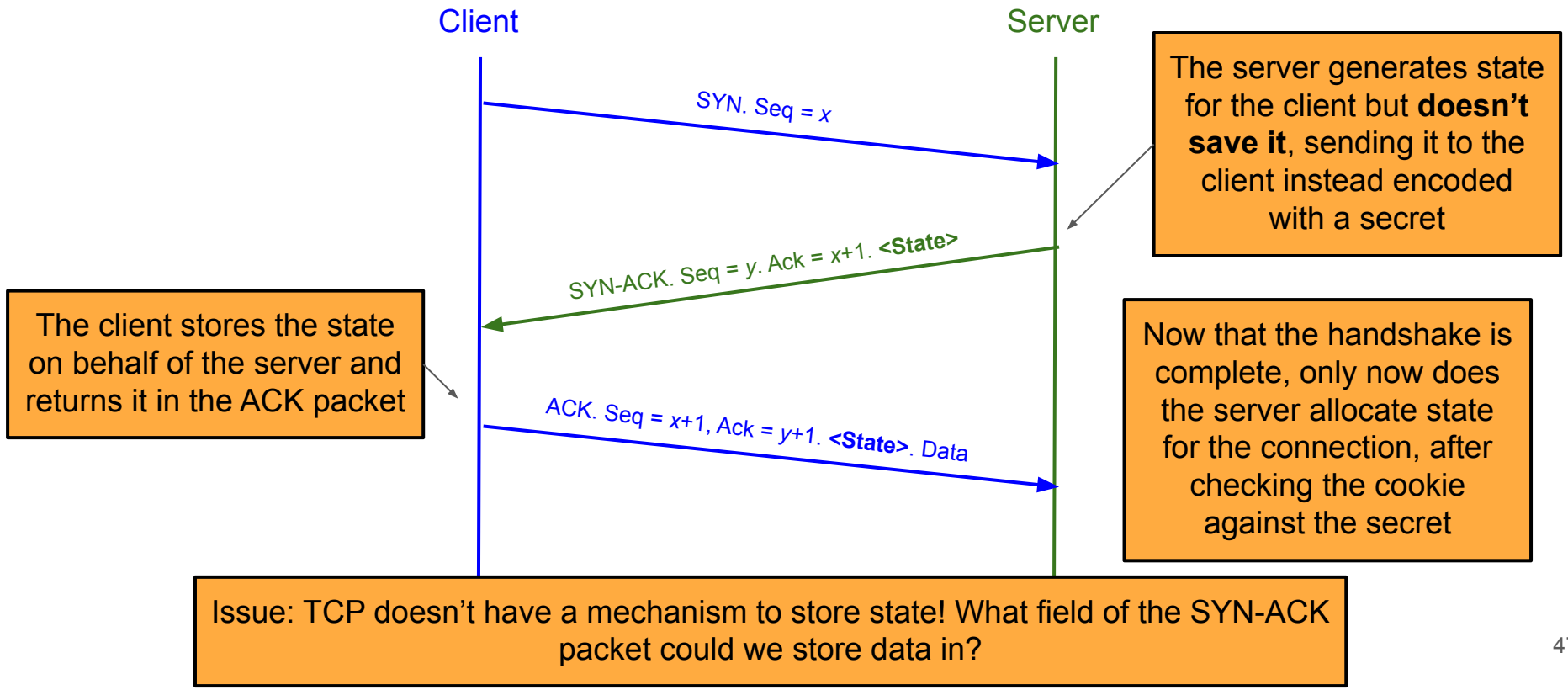
- A type of DoS that exploits many TCP connections
  - Each connection established by the server needs to allocate some memory
    - Used to store sequence numbers, ACK numbers, buffered data, etc.
  - Idea: Establish many connections with the server, causing it to consume a lot of memory
- TCP state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



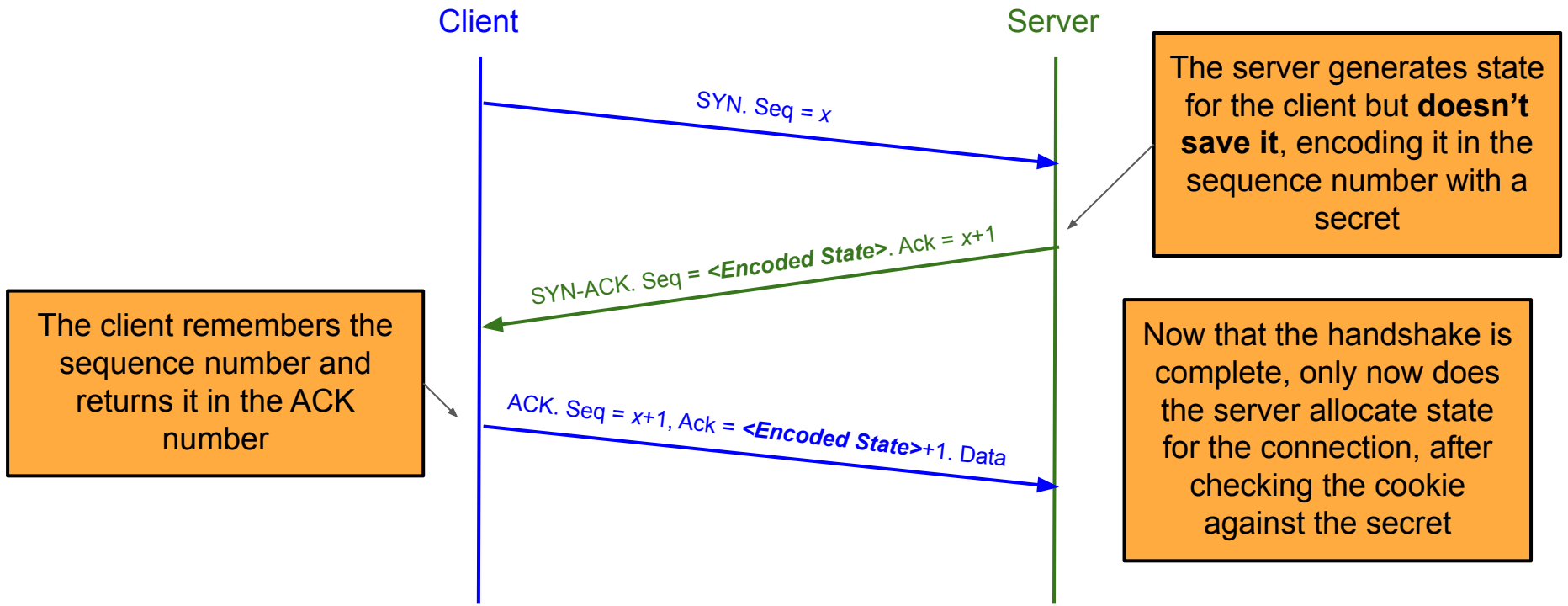
SYN Flooding: Defenses

- **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



Practical SYN Cookies



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!
- Do we have a primitive that makes it easy to do this?
  - *HMAC(k, {IPs, Ports, client ISN})* -> Server's ISN
  - Verify on ACK that sent ISN is a valid HMAC
  - HMAC is a great primitive whenever you want a **cookie** that is stored elsewhere that the server can validate

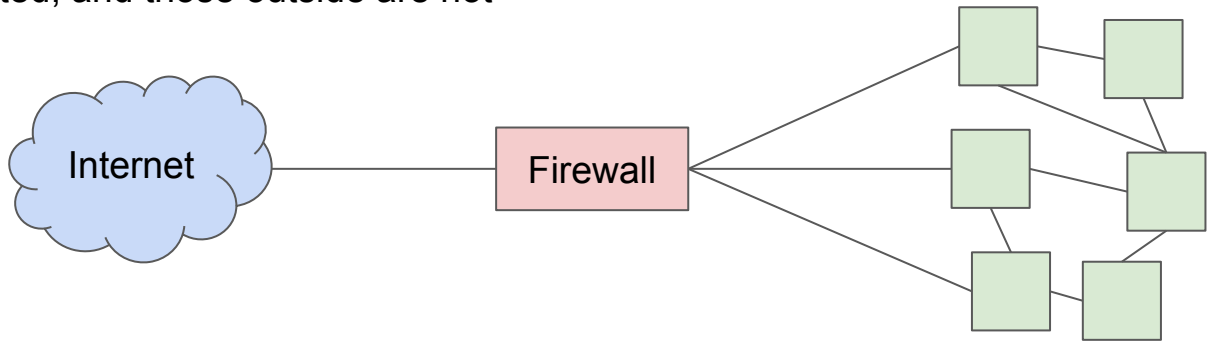
Firewalls

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!

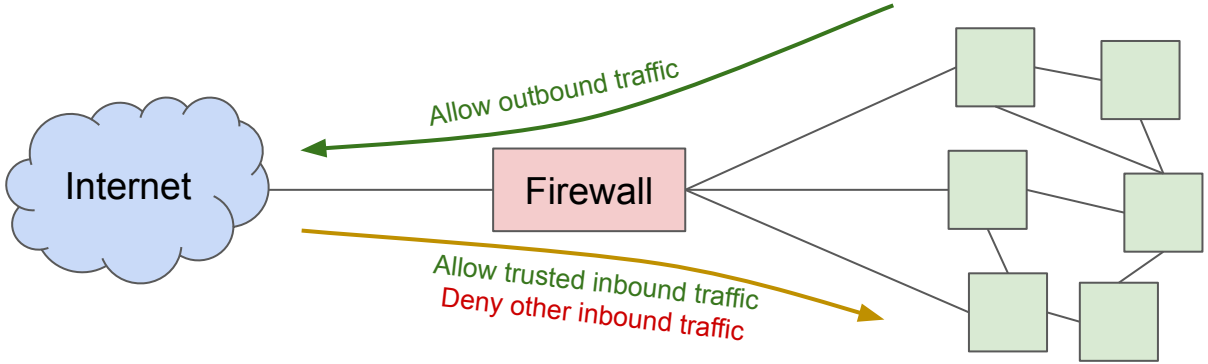
Firewalls and Security Policies

- 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), manually configured
    - 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?
    - Aside: user complaints are useful. Pay attention to them and try to make it so they don't complain, or if you can't fix the complaint, explain ***why***.
- 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

Oh, and FTP is a nightmare...

- FTP, File Transfer Protocol, is a really really ***really*** old network protocol
  - First standardized in 1985!
  - But still used ***enough*** that it often needs supporting
- Most client/server protocols: Client creates a TCP connection to the server, requests data
- FTP
  - Client creates a TCP connection to the server for control traffic
  - But to send an actual file...The client starts a TCP service to ***receive*** a connection from the server!
  - FTP requires that the firewall understand the FTP protocol to work!
- **Takeaway**: Supporting **legacy protocols** is often really really annoying

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

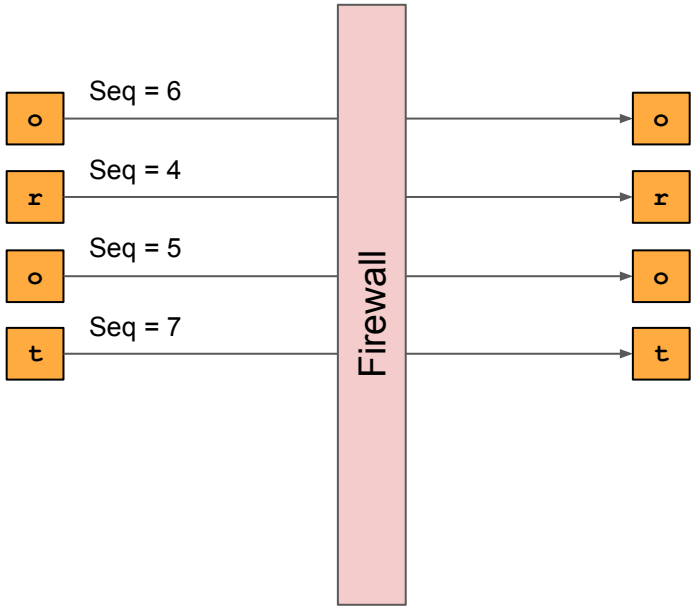
From: A	To: B
Seq = 4	
`Hello world`	

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

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

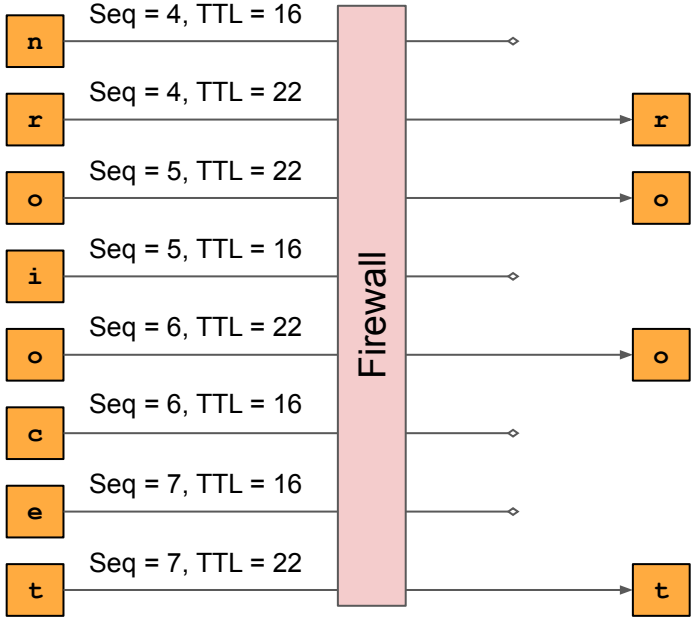
Subverting Packet Filters

- 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
- Example of **Evasion**, constructing network traffic that is parsed by the monitor in one fashion but the end host by another