6.858 Lecture 12
TCP/IP security

Threat model for network security:
• Adversary can intercept / modify network traffic.
• Adversary can send packets.
• Adversary has full control of their own machines.
• Adversary can participate in protocols (usually).
  o Often not feasible to keep bad guys out of a large systems.

Eavesdropping on packets.
• Important to keep in mind, but relatively well understood.
• Any data sent over the network can be observed by an adversary.

Sending / spoofing packets.
• IP allows sender to construct an arbitrary packet.
• In particular, sender can fill in any source address.
• Can pretend that a packet is coming from any address.
• What can an adversary do with this?

Easy target: trigger bugs in some implementation.
• Author isn't so interested in this class of problems.
• Instead, want to look at "protocol-level problems".
• What is a protocol-level problem?
  o A problem inherent in the design.
  o A correct implementation will have this problem.
• Why is it so important?
  o Can fix implementation bugs.
  o To fix protocol-level bugs, might need to change protocol!
  o Might be incompatible with existing systems.
  o As we will see, sometimes possible to come up with compatible fixes.

TCP sequence number attack.
Standard handshake (figure on the right side of page 2):
C: SRC=C, DST=S, SYN(SNc)
S: SRC=S, DST=C, SYN(SNs), ACK(SNc)
C: SRC=C, DST=S, ACK(SNs)
C: SRC=C, DST=S, data(SNc), ACK(SNs)

How does the adversary know the data is coming from the client?
• Only the client should have been able to receive the second message.
• Thus, only the client should know SNs.
• Third message is rejected, unless it has the right SNs value.
Suppose adversary A wants to simulate a connection to S from C. (Assume A knows C’s IP address -- usually not a big deal in practice.)
A: SRC=C, DST=S, SYN(SNc)
S: SRC=S, DST=C, SYN(SNs), ACK(SNc)
A: SRC=C, DST=S, ACK(SNs) -- but how to guess SNs?
A: SRC=C, DST=S, data(SNc)

Where does the adversary get SNs?
- TCP specification suggested a specific way to choose them.
- In particular, increment at a ~constant rate: ~250,000 per second.
- Why so specific?
  - Subtle interactions with reused connections (src/dst port numbers).
  - Want to avoid old packets (from past conns) interfering with new conn.
  - [Ref: RFC 1185 appendix]
- If adversary knows a recent sequence number, can guess the next one.
  - Impl would actually bump ISN every second, making it easy to guess.

What happens to the real packet that S sends to C (second pkt)?
- C would assume the packet is from an old conn, send RST in response.
- Even if that RST was sent, adversary could try to race before RST arrives.
- Luckily, there was another curious bug; will get to it later.
But why do sequence number attacks turn into a security problem?

1. Spoof connections to applications that rely on IP addresses.
   - E.g., Berkeley remote access tools: rlogin, rsh, rcp.
   - Allowed login without a password, if connection came from a "trusted" system.
     - Required connection to come from a trusted source port (512-1023).
       - Why this requirement?
         - Trusted rlogin/rsh/rcp program sent the client's username.
         - If username was the same as the account on the server, no password needed.
         - E.g.: "rsh athena.dialup.mit.edu ls".
   - Made a bad assumption about what the TCP layer provided.
     - Assumed TCP conn from an IP address meant it really came from that host.
   - If adversary can guess SNs, then can simulate connection from trusted host.
     - Issue any command using rsh.
     - Could change the user's .rhosts file to allow login from attacker's host.
     - Then connect directly without having to simulate a connection.
   - Host-based authentication seems like a bad plan.
     - Especially relying on "trusted" vs "untrusted" ports on a machine.
     - Still in some use today: e.g., SMTP for outgoing mail.
   - Actually rlogin authentication was even worse: they authenticated by hostname.
     - Where does hostname come from? Reverse DNS lookup.
     - E.g., 18.26.4.9: find the PTR record of 9.4.26.18.in-addr.arpa.
     - Owner of that domain can set PTR record to any hostname!
     - (Can make a slight improvement: check if host resolves to same addr.)
     - Similar problems show up in log files: log resolved (untrusted) hostname.
2. Denial of service attack: connection reset.
   • Once we know SNC, can send a RST packet.
   • Worse yet: server will accept a RST packet for any SNC value within window.
   • With a large window ($\sim 32K = 2^{15}$), only need $2^{32}/2^{15} = 2^{17}$ guesses.

How bad is a connection reset?
   • One target of such attacks were the TCP connections between BGP routers.
   • Causes routers to assume link failure, could affect traffic for minutes.
   • Solutions:
     o TTL hack (255).
     o MD5 header authentication (very specialized for router-to-router links).

3. Hijack existing connections.
   • In similar vein, can also inject data into an existing connection.
   • All adversary needs to know is the current SNC.

How to mitigate this problem?
   • Baseline: don’t rely on IP addresses for authentication.
     o Use encryption / authentication at a higher level.
     o Next lecture: Kerberos.
     o But still, want to fix the situation we’re in, for TCP.
   • ISPs can filter packets sent by their customers.
     o Often done today for small customers, but not consistently.
     o Not straightforward for customers with complex networks, multihoming...

How to patch up TCP?
   • Can’t choose ISN’s in a completely random way, without violating TCP spec.
     o Might break connection (port) reuse guarantees.
   • Random increments?
     o Should preserve increment rate ($\sim 250k$/second).
     o Not a huge amount of randomness (say, low 8 bits per increment).
   • Aside: must be careful about how we generate random numbers!
     o Common PRNG: linear congruential generator: $R_k = A*R_{k-1}+B \mod N$.
     o Not secure: given one pseudo-random value, can guess the next one!
     o Lots of better cryptographically secure PRNGs are available.
       ▪ Ideally, use your kernel’s built-in PRNG (/dev/random
       /dev/urandom)
   • However, SN values for different src/dst pairs never interact!
   • So, can choose the ISN using a random offset for each src/dst pair.
     o Nice trick: $ISN = ISN_{oldstyle} + F(srcip, srcport, dstip, dstport, secret)$
     o $F$ is some pseudo-random function; roughly, think SHA1.
Requires no extra state to keep track of per-connection ISNs.

Are sequence number attacks still relevant?
- Most operating systems implement the per-connection ISN workaround above.
  - Ref: Linux secure_tcp_sequence_number in net/core/secure_seq.c
- But other protocols suffer from almost identical problems -- e.g., DNS.
  - DNS runs over UDP, no seq numbers, just ports, and dst port fixed (53).
  - If adversary knows client is making a query, can fake a response.
    - Just need to guess src port, often predictable.
  - Problem gained popularity in 2008, though well-understood by djb before.
    - Ref: [http://cr.yp.to/djbdns/forgery.html](http://cr.yp.to/djbdns/forgery.html)
    - Ref: [http://unixwiz.net/techtips/iguide-kminsky-dns-vuln.html](http://unixwiz.net/techtips/iguide-kminsky-dns-vuln.html)
- Solution: carefully take advantage of all possible randomness!
  - DNS queries contain 16-bit query ID, and can randomize ~16 bit src port.
- Solution: deploy DNSSEC (signed DNS records, including missing records).
- One problem: key distribution (who is allowed to sign each domain?)
  - Another problem: name enumeration (to sign "no such name" responses).
  - Slow adoption, not much incentive to upgrade, non-trivial costs.
  - Costs include both performance and administrative (key/cert management).

SYN flooding.
- Note that server must store some state when it receives a SYN packet.
  - Called a half-open connection: replied with SYN-ACK, waiting for the ACK.
- What if it receives SYN messages from many sources?
  - Many implementations try to keep state for all half-open connections.
  - But eventually run out of memory, must reject connections!
- Annoying problem: we don’t even know who we’re keeping state for!
  - Adversary could have a single host, and generate SYNs from many src IPs.
- Denial-of-service attack: big asymmetry between client + server resources.
  - Client Spoofs a single packet (less than 1 millisecond).
  - Server wastes memory until connection times out (minutes).

Defense for SYN flooding: SYN cookies.
- Idea: make the server stateless, until it receives that third packet (ACK).
- Why is this tricky?
  - Need to ensure an adversary can’t make up a conn from any src address.
  - Previously, this was done by storing ISNs, and expecting it in the ACK.
- Use a bit of cryptography to achieve similar goal.
- Encode server-side state into sequence number.
  - ISNs = MAC_k(src/dst addr+port, timestamp) || timestamp
- Timestamp is coarse-grained (e.g., minutes).
- Server stores secret key k, not shared with anyone else.
- Detailed ref: [http://cr.yp.to/syncookies.html](http://cr.yp.to/syncookies.html)

- Server computes seq as above when sending SYN-ACK response.
- Server can verify state is intact by verifying hash (MAC) on ACK’s seq.
  - Not quite ideal: need to think about replay attacks within timestamp.
- Another problem: if third packet lost, noone retransmits.
  - Maybe not a big deal in case of a DoS attack.
  - Only a problem for protocols where server speaks first.

Another DoS attack vector: bandwidth amplification.
- Send ICMP echo request (ping) packets to the broadcast address of a network.
  - E.g., 18.26.7.255.
  - Used to be that you’d get an ICMP echo reply from all machines on network.
  - What if you fake a packet from victim’s address? Victim gets all replies.
  - Find a subnet with 100 machines on a fast network: 100x amplification!

- Can we fix this?
  - Routers now block "directed broadcast" (packets sent to broadcast address).

- Modern-day variant: DNS amplification.
  - DNS is also a request-response service.
  - With a small query, server might send back a large response.
  - With DNSSEC, responses contain lots of signatures, so they’re even larger!
  - Since DNS runs over UDP, source address is completely unverified.

- Can we fix the DNS attack?
  - Actually quite hard! Root name servers must answer to queries from anyone.

- What if we had a chance to re-design DNS from scratch?
  - One possible plan: query must be as big as response (require padding).
  - General technique: force client to expend at least as much work.

TCP congestion control.
- Receiver can get the sender to speed up, by ACKing unreceived segments. Or send more ACKs (e.g., send ACK for each byte instead of every packet).

Routing protocols: overly-trusting of participants.
- ARP: within a single Ethernet network.
  - To send IP packet, need the Ethernet MAC address of router / next hop.
  - Address Resolution Protocol (ARP): broadcast a request for target’s MAC.
  - Anyone can listen to broadcast, send a reply; no authentication.
- Adversary can impersonate router, intercept packets, even on switched net.
- Potential solution: make the switch in charge of ARP.
  - Not widely deployed: would require managing MAC/IP addresses carefully.

- DHCP: again, within a single Ethernet network.
  - Client asks for IP address by sending a broadcast request.
  - Server responds, no authentication (some specs exist but not widely used).
    - If you just plugged into a network, might not know what to expect.
  - Lots of fields: IP address, router address, DNS server, DNS domain list, ...
  - Adversary can impersonate DHCP server to new clients on the network.
    - Can choose their DNS servers, DNS domains, router, etc.
  - Also, DoS attack on server: ask for lots of leases, from many MAC addrs.
  - Solution: make the switch in charge of DHCP (forward reqs to real server).
    - Not widely deployed: would require careful switch configuration.
    - Even more complicated on a wireless network.

- BGP: Internet-wide (similar to RIP attacks described in paper).
  - Any BGP participant router can announce route to a prefix.
  - What if adversary has a router? Can announce any prefix or route.
  - Is this problem still relevant?
    - Spammers often exploit this: announce an unused address, and send spam.
    - Gets around IP-level blacklisting of spam senders: choose almost any IP!
  - How to fix?
    - SBGP: cryptographic signing of route announcements.
    - Must know who is allowed to announce every particular IP prefix.
    - Requires someone to distribute keys / certificates for every IP prefix.
    - Bootstrapping problem is tricky; some performance overheads too.
    - Getting some traction but still not widely deployed.

Many other problems too.
- ICMP messages like redirect: no authentication, basically unused now.
- Exposing too much information (netstat, SNMP, finger): mostly fixed.
- identd ("Authentication Service"): bad design, no real authentication.
- Email: real problem but no practical solutions yet.
  - Authentication vs authorization.
  - E.g., PGP would not solve the spam problem.
- Passwords in protocols: supporting ONLY passwords isn't so great.
We’ll talk about alternatives in a few weeks.

- FTP data transfer protocol.
  - Server connects back to client to send a file to the client.
  - Client tells the server what IP address and port number to use.
  - Could be used for port-scanning from server's IP.
  - Could be used to send any traffic (embedded in file) from server's IP.
    - E.g., back to IP authentication problems: rlogin, spam, etc.

How do adversaries know what software / protocol you are running?

- Probing:
  - Check if a system is listening on a well-known port.
  - Protocols / systems often send an initial banner message.
- nmap can guess OS by measuring various impl-specific details.
- Use DNS to look up the hostname for an IP address; may give hints.
- Guessing: assume system is vulnerable, try to exploit bug.

How do adversaries know the IP address of the system to attack?

- traceroute to find routers along the way, for BGP attacks.
- Can also just scan the entire Internet: only $2^{32}$ addresses.
  - 1 Gbps (100 MB/s) network link, 64 byte minimum packets.
  - ~1.5M packets per second.
  - $2^{32}=4B$ packets in ~2500 seconds, or 45 minutes.
  - zmap: implementation of this [ Ref: https://zmap.io/ ]

Why are things so insecure at the TCP/IP level?

- Historically, designers did not worry as much about security.
  - Even Bellovin says: "The Internet in 1989 was a much friendlier place".
  - Original Internet had a small number of relatively trustworthy users.
  - Design requirements changed over time.
- End-to-end argument in action.
  - Must provide security at the application level anyway.
  - Things are "good enough" at the transport level to let application work.
- Some fixes do get added, but only for the worst problems / easier solutions.

How to improve security?

- Protocol-compatible fixes to TCP implementations.
- Firewalls.
  - Partial fix, but widely used.
  - Issue: adversary may be within firewalled network.
  - Issue: hard to determine if packet is "malicious" or not.
  - Issue: even for fields that are present (src/dst), hard to authenticate.
  - TCP/IP's design not a good match for firewall-like filtering techniques.
  - E.g., IP packet fragmentation: TCP ports in one packet, payload in another.
- Implement security on top of TCP/IP: SSL/TLS, Kerberos, SSH, etc.
- Beware: this paper isn't clear on encryption vs. authentication.
  - Will talk about this more in next lecture on Kerberos.
- Use cryptography (encryption, signing, MACs, etc).
  - Quite a hard problem: protocol design, key distribution, trust, etc.
- Some kinds of security hard to provide on top: DoS-resistance, routing.
- Deployment of replacement protocols: SBGP, DNSSEC.