Lessons for today

- **You can trust hostnames IP addresses**
  - Can be forged
- **You can trust hosts on your network**
  - Probably insecure, can be compromised
- **Attackers can take you down, cut you from net**
- **But the good news:**
  - Don't need to trust hostname/IP addr (crypto, last week)
  - Ways of dealing with vulnerable hosts on your net
  - Ways of finding attacker who has taken you down

DNS attacks

- **Can spoof hostname by returning bad PTR record**
  - E.g., I own IP address 1.2.3.4, create record:
    `4.3.2.1.in-addr.arpa PTR www.nyu.edu`
  - You think I'm NYU's web server
- **You could look up** `www.nyu.edu` **to check**
- **But can still be thwarted, using glue records:**

```
4.3.2.1.in-addr.arpa  NS www.nyu.edu
www.nyu.edu           A  1.2.3.4
```

- DNS resolver adds bad `www.nyu.edu` address to cache

Forging source of TCP connection [Morris]

- **Suppose you can forge packets but not eavesdrop**
- **Goal: Forge TCP connection from some IP address**
  - E.g., simulate: `rsh victim 'echo + + >> ~/.rhosts'`
- **An approach: Forge SYN and ACK+data packets**
  - You just won't get SYN+ACK
- **Problem: What initial seq no. must you ACK?**
  - Solution: In some OSes, can predict given previous TCP con
- **Real host might get SYN+ACK, send RST**
  - Use source port on which real server is listening
  - Flood real server with SYNs, so it drops SYN+ACK

Joncheray TCP attack

- **Suppose you can eavesdrop on TCP traffic**
  - But can't cause packets to be dropped
- **Want to hijack existing TCP connections**
  - E.g., take over s/key-authenticated login session
  - Problem: Legitimate packets might interfere w. yours
- **Solution: Put TCP in *desynchronized state***
  - No data in transit, but $\mathrm{Seq}_S \neq \mathrm{Ack}_C$ and $\mathrm{Seq}_C \neq \mathrm{Ack}_S$
  - Actually want: $\mathrm{Seq}_C < \mathrm{Ack}_S$ or $\mathrm{Seq}_C > \mathrm{Ack}_S + \mathrm{Window}_S$
    Means server won't process client packets—out of window!
  - But hosts will repeat last ACK $\rightarrow$ ACK storms
- **How to desynchronize TCP?**

Desynchronizing a TCP connection

- **Early desynchronization**
  - Client connects to server
  - Attacker sends RST to server
  - Attacker sends SYN to server forged to be from client
  - Now server has connection with same ports, different Ack_S
- **Null data desynchronization**
  - Attacker generates a lot of data that will be ignored by the application E.g., NOP operation in telnet does nothing
  - Sends this NULL both to client and to server
  - Drives up Ack_C and Ack_S so they are no longer in range

The problem: securing a network

- **Let's say you run a large network like NYU's**
  - Large, diverse collections of machines
  - Run by many different people (students, departments, etc.)
  - Can't control what software people run
- **Vulnerabilities in software spring up regularly**
  - You name it, someone is probably running in on your net
  - Always a few people who don't patch/upgrade software
- **Means attackers can control machines on your net!**
  - Use your net as vantage point to attack others
  - Send spam, flood packets, snoop net for passwords, etc.
  - Break into other machines on your net

Dealing vulnerable systems

- **Learn who on your network is vulnerable**
  - Monitor + actively scan your net to see what people run
  - Use tricks to determine what OS/version machines have: E.g., order of options, how TCP responds to weird packets
  - When security advisories come out, see who has patched
  - Try to get in touch with users and convince them to patch
- **Discover intruders as they attack**
  - Monitor network for evidence of known attack types
  - If actual attack detected, can cut host from network (whereas can't yank hosts just for being vulnerable)

Detecting network intruders

- **Compile list of known network vulnerabilities**
  - Buffer overruns in servers
  - Servers with bad implementations
    (“login -froot”, telnet w. LD_LIBRARY_PATH)
- **Detect people exploiting such bugs**
- **Detect activities performed by people who’ve penetrated server**
  - Setting up IRC bot
  - Running particular commands, etc.
- **Example IDS system: Bro [Paxson]**

Bro model

- **Attach machine running Bro to "DMZ"**
  - Demilitarized zone – area betw. firewall & outside world
- **Sniff all packets in and out of the network**
- **Process packets to identify possible intruders**
  - Secret, per-network rules identify possible attacks
  - Is it a good idea to keep rules secret?
- **React to any threats**
  - Alert administrators of problems in real time
  - Switch on logging to enable later analysis of potential attack
  - Take action against attackers – E.g., filter all packets from host that seems to be attacking, or to vulnerable machine

Goals of system

- **Keep up with high-speed network**
  - No packet drops
- **Real-time notification**
- **Extensibility**
  - Bro scripting language specifies network events
- **Separate mechanism from policy**
  - Avoid easy mistakes in policy specification
  - So different sites can specify “secret” policies easily
- **Resilience to attack**

Bro architecture

- **Layered architecture:**
  - bpf/libpcap, Event Engine, Policy Script Interpreter
- **Lowest level bpf filter in kernel**
  - Match interesting ports or SYN/FIN/RST packets
  - Match IP fragments
  - Other packets do not get forwarded to higher levels
- **Event engine, written in C++**
  - Knows how to parse particular network protocols
  - Has per-protocol notion of events
- **Policy Script Interpreter**
  - Bro language designed to avoid easy errors

Overload and Crash attacks on Bro

- **Overload goal: prevent monitor from keeping up w. data stream**
  - Leave exact thresholds secret
  - Shed load (e.g., HTTP packets)
- **Crash goal: put monitor out of commission**
  - E.g., run it out of space (too much state)
  - Watchdog timer kills & restarts stuck monitor
  - Also starts tcpdump log, so same crash attack, if repeated, can be analyzed

Subterfuge attacks

- **IP fragments too small to see TCP header**
- **Retransmitted IP fragments w. different data**
- **Retransmitted TCP packets w. different data**
- **TTL/MTU monkeying can hide packets from destination**
  - Compare TCP packet to retransmitted copy
  - Assume one of two endpoints is honest (exploit ACKs)
  - Bifurcating analysis

State and checkpointing

- **Need to keep a lot of session state**
  - Open TCP connections, UDP request-response, IP fragments
  - No timers to garbage collect state (implementation limitation)
- **Checkpointing the system**
  - Want to save state for off-line analysis
  - Want to reclaim memory for old useless state
  - Done by checkpointing and restarting monitoring process
  - Start new copy of monitoring process
  - Kill old copy when new copy has come up to speed

Firewalls

- **Goal: prevent attacks before they happen**
  - Don't just want to detect—want to block
  - Block many packets that might be attacks
  - May stop some legitimate uses of net, too
- **Control traffic between your net and outside world**
  - Prevent outsiders from attacking local machines, *even if those machines are vulnerable*
  - Load a series of rules into your router, to determine which packets to pass, which to block

Common firewalling of IP packets

- **Block potentially dangerous packets**
  - Packets with IP source routing options
  - "Directed broadcast" packets
- **Block *some* incoming ICMP packets**
  - ICMP redirect – definitely (would change routing tables)
  - ICMP echo – If don't want outsiders probing your net
- **Block forged packets at firewall router**
  - Block packets with local addresses coming from outside
  - Block outgoing packets with non-local addresses
    (in case a local machine compromised—more later)
- **Block or reassemble small fragments**
  - So that firewall can examine full UDP/TCP headers

Firewalling TCP

- **Usually want to allow outgoing connections**
  - Some exceptions (E.g., port 25 mail, if worried about your users generating spam)
- **But restrict incoming connections**
  - Only allow connections to well-known servers/services not running vulnerable software
  - Otherwise, can drop the packet (slow for services like ident)
  - Or firewall machine can forge TCP RST packet
- **How to implement**
  - Block all incoming SYN packets w/o ACKs (stateless, good)
  - Or keep state for outgoing connections (also common), Prevents mapping of network with bogus TCP packets

Firewalling UDP & ICMP

- **No stateless tricks like TCP**
- **Must keep state for all packets**
  - E.g., After DNS request, allow response
  - After ICMP echo request, allow reply to come back in
  - Allow ICMP dest unreachable if waiting for UDP reply
- **Firewall reboots will wipe state—disruptive**
- **Some firewalls have protocols for state synchronization**
  - Allows fail-over when primary firewall crashes

DoS attacks

- **In Feb. 2000, Yahoo's router kept crashing**
  - Engineers had problems with it before, but this was worse
  - Turned out they were being flooded with ICMP echo replies
  - Many DDoS attacks followed against high-profile sites
- **Basic Denial of Service attack**
  - Overload a server or network with too many packets
  - Mamize cost of each packet to server in CPU and memory
- **Distributed DoS (DDos) particularly effective:**
  - Penetrate many machines in semi-automatic fashion
  - Make hosts into "zombies" that will attack on command
  - Later start simultaneous widespread attacks on a victim

Smurf attack

- **Yahoo attack was smurf attack**
  - Penetrated hosts on well-connected networks
  - Flooded LAN with broadcast pings “from” yahoo
  - Every host on LAN then replied to Yahoo
  - Attack was *amplified* through uncompromised hosts
- **Can tolerate above by filtering packets**
  - Packets all ICMP echo replies from particular addresses
  - Attack still had to be traced to stop waste
  - But attack packets could be distinguished from most legitimate traffic

The SYN-bomb attack

- **Recall the TCP handshake:**
  - $C \rightarrow S$: SYN, $S \rightarrow C$: SYN-ACK, $C \rightarrow S$: ACK
- **How to implement:**
  - Server inserts connection state in a table
  - Waits for 3rd packet (times out after a minute)
  - Compares each new ack packet to existing connections
- **OS can't handle arbitrary # partial connections**
- **Attack: Send SYN packets from bogus addresses**
  - SYN-ACKs will go off into the void
  - Server's tables fill up, stops accepting connections
  - A few hundred pkts/sec completely disables most servers

Other attacks

- **IP Fragment flooding**
  - Kernel must keep IP fragments around for partial packets
  - Flood it with bogus fragments, as with TCP SYN bomb
- **UDP echo port 7 replies to all packets**
  - Forge packet from port 7, two hosts echo each other
  - Has been fixed in most implementations
- **Standard flooding attacks**
  - Just flood-ping any site
  - Or bombard DNS server with requests

Making attacks hard to stop

- **Make DoS traffic indistinguishable from legit**
  - SYN-bomb ideal, DNS or any UDP service good
  - Flood-ping at least can be filtered anywhere upstream
- **Make source of attack hard to trace**
  - Victims need to trace attack and pull the plug
  - Can forge source IP address so packet origin not obvious
  - Most DoS tools use a random address for each packet
  - Can also use reflectors–bounce attack through 3rd parties

Coping with denial of service

- **Engineering OSes to tolerate attacks**
  - Reduce state required for embryonic TCP connections
  - Increase size of hash table for protocol control blocks
- **Network monitoring box (schuba et al.)**
  - Passively monitors network (like Bro)
  - Uses heuristics to detect SYN bomb attacks
    (e.g., traffic paterns w. invalid source addresses)
  - Monitor engineered to keep little state
  - Send out forged RST packets to free resources on victim

Egress filtering

- **Forged addresses complicate shutting off DoS**
  - Where is flood of packets coming from?
- **Filter forged outgoing packets**
  - Sites should block outgoing packets not from their network
  - ISPs should block packets not from customer's network
- **But still need to detect and shut down attacks**
- **And most attackers can find non-filtered networks anyway**

Tracing forged packets

- **Problem: how to deal with forged packets**
  - Don't know which packets to fliter upstream
  - Can't find attacking machine to pull the plug
- **Need to trace attacks back link-by-link**
  - Goal: List of routers, where prefix is path to attacker
- **Many techniques for tracing, with trade-offs:**
  - Management, Bandwidth, Router CPU, Distributed attacks, Post-mortem capability, Preventative vs. Reactive capability

Input debugging

- **Some routers can trace output to input**
  - Develop attack signature to classify bad packets
  - Router tells you which input port they are from
- **Of course, only router administrator can do this**
- **Must continue on to upstream routers, in other realms**
- **Not all routers have this capability**

CenterTrack

- **Problem: ISPs want to trace attacks themselves**
  - Don't want to involve other administrators for each trace
- **CenterTrack: Employ overlay network**
  - Reroute all of victims traffic through an overlay network
  - Can do this by advertising different route with BGP
  - Send all traffic through central tracking router
  - Run input debugging on tracking router

Controlled flooding

- **Problem: Suppose you want to track attack w/o support from network operators**
- **Solution: *controlled flooding* [Burch&Cheswick]**
  - Exploit attacks that cause other hosts to flood you
  - Use knowledge of network to flood along various links
  - Infer source of real attack from interference with your attack
- **Ingenious, but somewhat evil (exploiting hosts)**

ICMP traceback

- **Goal: Let people trace attacks less destructively**
- **Have routers send tracing traffic**
  - Each router randomly traces 1 in 20,000 packets
  - Sends special ICMP traceback packet including packet and link that it came from
  - Victim can trace attack back from these packets
- **Unfortunatly, hard to implement**
  - Not all routers know input link when processing packet
- **Other weaknesses?**

ICMP traceback disadvantages

- **ICMP traffic sometimes differentiated from TCP**
  - More willing to drop them when under attack
- **Requires key distribution infrastructure**
  - Otherwise, attacker can forge ICMP traceback packets
- **Attacker can flood with forged traceback packets**
  - People will filter traceback packets to survive
  - How to tell real packets from forged ones on fast path?
- **Incremental deployment makes tracing hard**
  - Can’t line up input link to previous node if previous node isn’t generating tracebacks

Packet marking

- **Put tracing information in packets themselves**
- **Node append: The simplest solution**
  - Each router appends its address to every packet
  - Can get attack path from any packet
- **Problem: No room in packets**
  - E.g., with MTU discovery, TCP sends maximum sized segments
  - Would need to fragment, terrible overhead

Node sampling

- **Reserve a single fixed-size node field in header**
  - Just enough to hold IP address of one router—32 bits
- **Routers stamp their addr. in field w. probability p**
- **Eventually victim will get stamps from whole path**
  - Get stamp from d hops upstream with prob. $p(1-p)^{d-1}$
  - Can infer number of hops d from # of pkts. w. stamps
  - If $p > 0.5$, attacker cannot fake closer routers
- **Limitations**
  - Need many packets to trace distant nodes. With $p = 0.5$, $\sim 300,000$ pkts. needed for 95% confidence in router order
  - Even 32 bits hard to find in all packets
  - Hard to separate paths from multiple attackers

Edge sampling

- **Add three fields to each packet: start, end, distance**
- **Router at address A marks packets as follows:**
  - With prob. p: start $\leftarrow A$, distance = 0
  - Else: distance++. If distance was 0, end $\leftarrow A$.
- **To reconstruct path, victim makes graph**
  - Starts with own address, inserts edge for each packet
  - Eliminate edges (start,end,d) if d not distance in graph
- **Works well with multiple attackers**
- **Incremental deployment works well, too**
  - An edge is closest two routers implementing system
- **Still requires non-existent space in IP headers**

Compressed edge sampling [Savage et al.]

- **Save a factor of two by XORing start & end**
  - Packet with $d = 0$ contains address of first router
  - XOR that with address in pkt with $d = 1$ to get next hop, etc.
- **Put only the fragment of an address in each packet**
- **Use checksum to add redundancy**
  - 32-bit checksum of IP address interleaved with address bits
  - Try all possible fragment reconstructions
  - Discarded ones in which checksum does not work out

Implementation

- **Use unused fragment ID in non-fragments**
- **You get 16 bits. Allocate as follows:**
  - 3 bit offset (which 1/8 of address is this)
  - 5 bit distance (32 hops is generally enough for internet)
  - 8 bit edge fragment
- **Distance aligned with TTL for checksum**
  - Makes implementation efficient—no change in IP checksum
- **Issue of fragmentation (though $< 0.25\%$ of traffic)**
  - Upstream fragments: If marked, frag IDs may then differ. So trash pkt & use full edge marking w. low probability
  - Downstream: Can get ugly if IDs reused. Could use DF bit.