Announcements

- No question session this week
Stretch break
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-pong 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
Premise: Many DoS attacks produce backscatter
- random IP source address gets reply
Measuring DoS activity

• Measure backscatter to quantify DoS attacks
  - If you $m$ backscatter packets while monitoring $n$ addresses, then $\sim nm/2^{32}$ attack packets were sent.

• Researchers got lightly-loaded class-A network
  - Represents 1/256 of all 32-bit IP addresses
  - Single workstation observed all traffic to class-A net

• How worrisome are results?

• What are sources of error in experiments?
Limitations of Technique

- Factors that will cause underestimation
  - Ingress filtering by ISPs
  - Packet loss
  - Reflector attacks
  - Attack packets that don’t cause reply (TCP RST bomb)

- Non-attack packets could cause overestimation

- Non-random source IP addresses could affect results either way
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 patterns 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
MULTOPS: Detecting DOS attacks

- **Observation:** Many protocols bidirectional
  - TCP: 0.5–1 ACKs for every data packet
  - DNS, ping: Reply for every request
  - Streaming media (not so easy, but can have heuristics)

- **Substantial imbalance means something is wrong**
  - Many SYNs not getting SYN ACKs (SYN bomb)
  - Many ICMP echo requests not getting replies (ping flood)

- **Attempt to detect problem and filter bad sources**
  - If attackers being egress filtered, will work
  - Can also be run in reverse to detect outgoing attacks
    (E.g., detect if NYU’s network is being used for DoS attack)
Multops tree structure

- Keep aggregate statistics for address prefixes
  - Subdivide ranges in which an attack is detected
  - Keeps detailed statistics for attackers with limited space
  - Defend against attempts to exhaust memory
Tracing forged packets

- **MULTOPS not useful against forged packets**
  - Don’t know which packets to filter 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 chooses 1 in 20,000 packets to trace
  - 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

- Attacker can flood with forged traceback packets
  - People will filter traceback packets to survive
  - How to tell real packets from forged ones?

- 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 away 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 closes 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.