Denial-of-Service

David Andersen
CMU CS 15-744

Today's Lecture

- What is Denial of Service?
- Attacks and Defenses
  - Packet-flooding attacks
    - **Attack:** SYN Floods
    - **Defenses:** Ingress Filtering, SYN Cookies, Client puzzles
  - Low-rate attacks
    - **Detection:** Single-packet IP Traceback
- **Network-level defenses:** sinkholes and blackholes

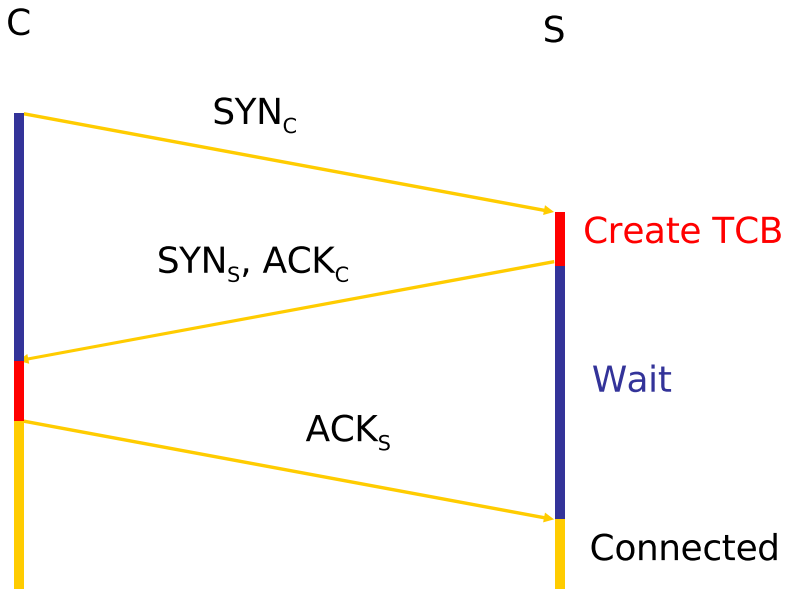
- Inferring Denial of Service Activity
- Distributed Denial of Service
- Worms
- Other resource exhaustion: spam

Denial of Service: What is it?

- *Crash victim* (exploit software flaws)
- Attempt to *exhaust victim's resources*
  - **Network:** Bandwidth
  - **Host**
    - **Kernel:** TCP connection state tables, etc.
    - **Application:** CPU, memory, etc.
  - Often high-rate attacks, but not always

graphics credit: Feamster

TCP: 3-Way Handshake



slide credit: Feamster

Example: TCP SYN Floods

- Each arriving SYN stores state at the server
  - TCP Control Block (TCB)
  - ~ 280 bytes
    - FlowID, timer info, Sequence number, flow control status, out-of-band data, MSS, other options
- Attack:
  - Send TCP SYN packets with bogus src addr
  - Half-open TCB entries exist until timeout
  - Kernel limits on # of TCBs
- Resources exhausted ⇒ requests rejected

Preventing SYN floods

- Principle 1: Minimize state before “auth”
  - (3 way handshake == auth)
  - Compressed TCP state: Very tiny state representation for half-open conns
    - Don't create the full TCB
  - A few bytes per connection == can store 100,000s of half-open connections
-

SYN Cookies (generalizes!)

- Idea: Keep *no* state until auth.
  - In response to SYN, send back self-validating token to source that source must attach to ACK
  - SYN -> SYN/ACK+token -> ACK+token
    - Validates that the receiver's IP is valid
  - How to do in SYN? sequence #s!
    - top 5 bits: time counter
    - next 3: Encode the MSS
    - bottom 24: F(client IP, port, server IP, port, t)
  - Downside to this encoding: Loses options.

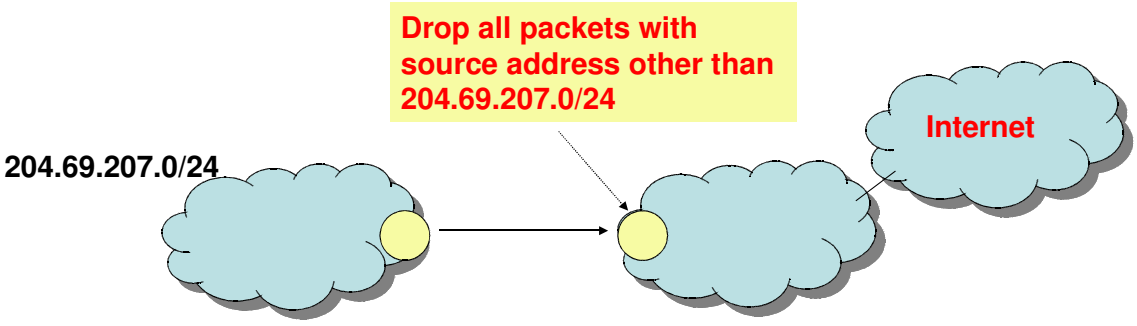
Bandwidth Floods

- 1990s: Brute force from a few machines
  - Pretty easy to stop: Filter the sources
  - Until they spoof their src addr!
- Late 90s, early 00s: Traffic Amplifiers
  - Spoofed source addrs (next)
- Modern era: Botnets
  - Use a worm to compromise 1000s+ of machines
  - Often don't need to bother with spoofing

Reflector Attacks

- Spoof source address
- Send query to service
- Response goes to victim
  - If response >> query, “amplifies” attack
  - Hides real attack source from victim
- Amplifiers:
  - DNS responses (50 byte query -> 400 byte resp)
  - ICMP to broadcast addr (1 pkt -> 50 pkts) (“smurf”)

!Spoofing 1: Ingress/Egress Filtering

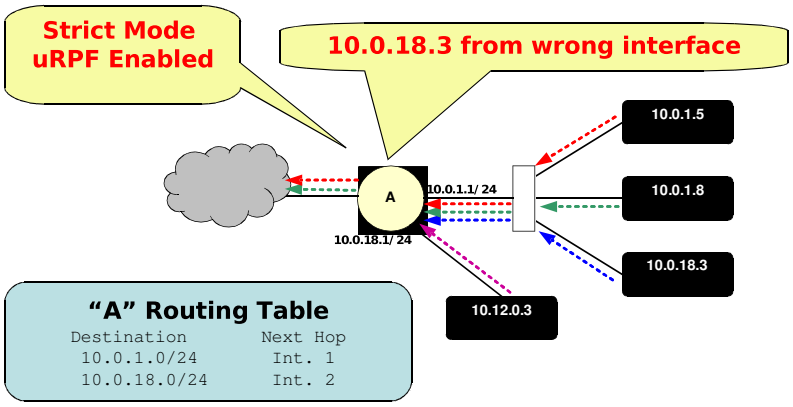


- **RFC 2827:** Routers install filters to drop packets from networks that are not downstream
- Feasible at edges; harder at “core”
- Deployment incentive mismatch

Slide derived from Feamster

!Spoofing 2: uRPF Checks

Accept packet from interface only if forwarding table entry for source IP address matches ingress interface



- Unicast Reverse Path Forwarding
  - Cisco: "**ip verify unicast reverse-path**"
- Requires symmetric routing

Slide Credit: Feamster

Filters & Pushback

- Assumption: Can identify anomalous traffic (Coming in 2nd half of lecture)
- Add "filters" that drop this traffic
  - Access control lists in routers
  - e.g. deny ip from dave.cmu.edu to victim.com tcp port 80
- Pushback: Push filters further into network towards the source
  - Need to know where to push the filters (traceback)
  - Need authentication of filters...
  - Tough problems. Filters usually deployed near victim.

Capabilities

- Filters: *prevent the bad stuff*
- Capabilities: *must have permission to talk*
- Sender must first ask dst for permission
  - If OK, dst gives *capabilitiy* to src
  - capability proves to routers that traffic is OK
- Good feature: stateless at routers

TVA

- Routers put *pre-capability* in src->dst request
  - Timestamp | Hash(src, dst, time, router secret)
  - secret changes slowly
  - dst sees these pre-capabilities and can echo them back to src if it wants to.
  - Routers can verify pre-capability w/out state
- Limited time & b/w:
  - Timestamp | H(pre-caps, N bytes, Time T)
  - dst gives src more N,T as appropriate

More TVA

- Cute trick in paper for monitoring “heavy” flows
- Similar trick for caching capabilities (space overhead in pkts vs. state in routers)

Discussion

- “Denial-Of-Capability” - first pkt must get through
  - But once one pkt makes it, flow is good.
  - Vast improvement over today...
- b/w exhaustion: cooperating attackers give each other many tokens
  - Need fair queueing or similar
- Route changes? Need new capability
- How does dst know if sender is evil or not?

Detecting Attacks

- Intrusion Detection Systems at host or net
  - Assumes attack traffic is identifiable
  - Hard with botnets that request legit web pages
    - Captchas? Computational puzzles?
  - Reasonable with many attacks

Traceback with Packet Markings

- With probability *p*, each router marks packet hdr:
  - Router IP
  - Distance *d* from the source (increments @ each rtr)
- Re-assemble the list of routers w/enough pkts
  - Have to resist spoofing attacks at src
    - But can trace back to where attacker controls
- Improvement: edge sampling (src/dst IP of link)
- Needs new IP header, *or* cute encoding tricks
  - Increases # of packets that must be heard...
  - Everyone abuses these fields. :)

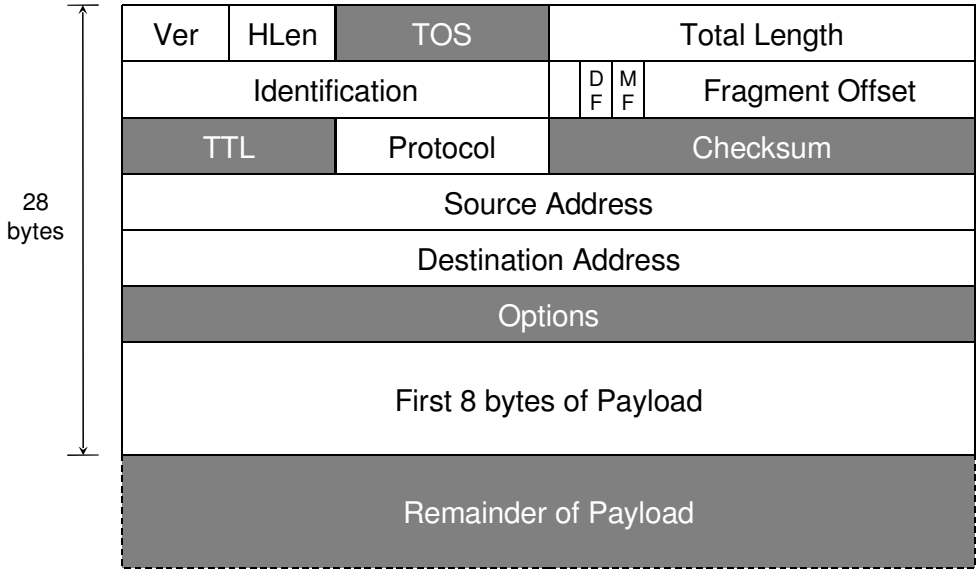
Single packet traceback

- Marking can ID bandwidth floods
- But what about single-pkt DDoS or exploits?
- Strawman 1: Log all packets!
  - 500Gbit/sec through a big core router
  - 4 TB for one minute of logging (!)
  - Just headers? Maybe 100GB/minute. (!)
    - Okay start, but ...
    - A fast hard drive can write 4.2GB/minute...

Log Packet Digests

- Goal: Receiver has copy of the packet
- Ask router: “Have you seen this packet?”
  - Mask out changing parts of pkt (TTL, TOS, checksum, opts)
  - Include first 8 bytes of payload (TCP/UDP headers)
- Hash these bytes -> 32 bits
- Is this enough?
  - BFRouter: 1B packets/sec = 4GBytes/sec
  - That's a lot of pretty fast memory, but *nearly* there

Hash input: Invariant Content



Bloom Filters

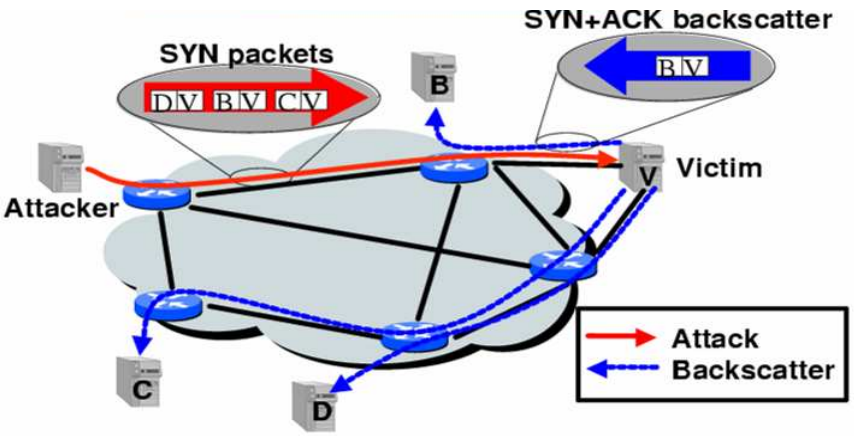
- Useful algorithmic trick: More space efficiency at cost of some false positives
- Compute *k* independent hashes of the packet
  - h1, h2, ..., hk each *n* bits long
- Create an array of size 2^n bits
- Set bits corresponding to h1, h2, ..., hk
  - Can control FP rate by choosing n, k for the expected # elements in array
  - e.g., 8 bits/entry: FP rate 2%

Traceback w/Bloom Filters

- Down to 8 bits/packet: 1GB/sec (reasonable)
- What about false positives?
  - May get false neighbors saying “I saw it!”
  - If false neighbors' neighbors don't, query dies
  - So FP rate must be low enough to prevent query explosion
- To query: Recursively query neighbors of router that says it saw packet

Inferring DoS Activity: Backscatter

IP address spoofing creates random *backscatter.*



pretty diagram courtesy of Feamster

Backscatter Analysis

- Use a big block of addresses (*N* of them)
  - People often use a /16 or /8
- Observe *x* backscatter packets/sec
- How big is actual attack?
  - x * (2^32 / N)
  - Assuming uniform distribution
- Sometimes called “network telescope”
  - 2001: 12,805 attacks in 3 weeks (not all types!)
  - Some > 600,000 packets per second.
  - Today's botnets probably MUCH larger (next time. :)