Unwanted Traffic: Denial of Service Attacks

Dan Boneh
What is network DoS?

Goal: take out a large site with little computing work

How: Amplification

- Small number of packets $\Rightarrow$ big effect

Two types of amplification attacks:

- DoS bug:
  - Design flaw allowing one machine to disrupt a service

- DoS flood:
  - Command bot-net to generate flood of requests
DoS can happen at any layer

This lecture:

- Sample Dos at different layers (by order):
  - Link
  - TCP/UDP
  - Application
- Generic DoS solutions
- Network DoS solutions

Sad truth:

- Current Internet not designed to handle DDoS attacks
Warm up: 802.11b DoS bugs

- Radio jamming attacks: trivial, not our focus.

- Protocol DoS bugs: [Bellardo, Savage, ’03]

  - NAV (Network Allocation Vector):
    - 15-bit field. Max value: 32767
    - Any node can reserve channel for NAV seconds
    - No one else should transmit during NAV period
    - ... but not followed by most 802.11b cards

  - De-authentication bug:
    - Any node can send deauth packet to AP
    - Deauth packet unauthenticated
    - ... attacker can repeatedly deauth anyone
Smurf amplification DoS attack

- Send ping request to broadcast addr (ICMP Echo Req)
- Lots of responses:
  - Every host on target network generates a ping reply (ICMP Echo Reply) to victim

Prevention: reject external packets to broadcast address
Modern day example  (Mar ’13)

DNS Amplification attack:  (×50 amplification)

2006:  0.58M open resolvers on Internet (Kaminsky-Shiffman)
2014:  28M open resolvers (openresolverproject.org)

Influence of Network Traffic

Figure 13
Source: Arbor Networks, Inc.

Feb. 2014: 400 Gbps via NTP amplification (4500 NTP servers)
Review: IP Header format

- Connectionless
  - Unreliable
  - Best effort

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Version of the IP protocol</td>
</tr>
<tr>
<td>Header Length</td>
<td>Length of the IP header in 32-bit words</td>
</tr>
<tr>
<td>Type of Service</td>
<td>Type of service provided by the sender</td>
</tr>
<tr>
<td>Total Length</td>
<td>Total length of packet in bytes</td>
</tr>
<tr>
<td>Identification</td>
<td>Identification number</td>
</tr>
<tr>
<td>Flags</td>
<td>Flags for packet integrity</td>
</tr>
<tr>
<td>Fragment Offset</td>
<td>Offset of fragment in the packet</td>
</tr>
<tr>
<td>Time to Live</td>
<td>Time to live before discard</td>
</tr>
<tr>
<td>Protocol</td>
<td>Protocol number</td>
</tr>
<tr>
<td>Header Checksum</td>
<td>Checksum of the header</td>
</tr>
<tr>
<td>Source Address of Originating Host</td>
<td>Final destination address of originating host</td>
</tr>
<tr>
<td>Destination Address of Target Host</td>
<td>Final destination address of target host</td>
</tr>
<tr>
<td>Options</td>
<td>Options</td>
</tr>
<tr>
<td>Padding</td>
<td>Padding</td>
</tr>
<tr>
<td>IP Data</td>
<td>IP data</td>
</tr>
</tbody>
</table>
Review: TCP Header format

TCP:
- Session based
- Congestion control
- In order delivery
Review: TCP Handshake

**SYN:**
- \( SN_c \leftarrow \text{rand}_c \)
- \( AN_c \leftarrow 0 \)

**SYN/ACK:**
- \( SN_s \leftarrow \text{rand}_s \)
- \( AN_s \leftarrow SN_c \)

**ACK:**
- \( SN \leftarrow SN_c \)
- \( AN \leftarrow SN_s \)

- Listening
- Wait
- Established

Store \( SN_c, SN_s \)
TCP SYN Flood I: low rate (DoS bug)

Single machine:
- SYN Packets with random source IP addresses
- Fills up backlog queue on server
- No further connections possible
SYN Floods

(Phrack 48, no 13, 1996)

<table>
<thead>
<tr>
<th>OS</th>
<th>Backlog queue size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 1.2.x</td>
<td>10</td>
</tr>
<tr>
<td>FreeBSD 2.1.5</td>
<td>128</td>
</tr>
<tr>
<td>WinNT 4.0</td>
<td>6</td>
</tr>
</tbody>
</table>

Backlog timeout: 3 minutes

- Attacker needs only 128 SYN packets every 3 minutes
- Low rate SYN flood
A classic SYN flood example

- **MS Blaster worm** (2003)
  - Infected machines at noon on Aug 16\textsuperscript{th}:
    - SYN flood on port 80 to \texttt{windowsupdate.com}
    - 50 SYN packets every second.
      - each packet is 40 bytes.

- **MS solution:**
  - new name: \texttt{windowsupdate.microsoft.com}
Low rate SYN flood defenses

Non-solution:
- Increase backlog queue size or decrease timeout

Correct solution (when under attack):
- **[Syncookies]**: remove state from server
- Small performance overhead
Syncookies

[Bernstein, Schenk]

- Idea: use secret key and data in packet to gen. server SN

- Server responds to Client with SYN-ACK cookie:
  - $T = 5$-bit counter incremented every 64 secs.
  - $L = \text{MAC}_{\text{key}}(\text{SAddr}, \text{SPort}, \text{DAddr}, \text{DPort}, \text{SN}_C, T)$ [24 bits]
    - key: picked at random during boot
  - $\text{SN}_S = (T \cdot \text{mss} \cdot L)$ ( $|L| = 24$ bits )
  - Server does not save state (other TCP options are lost)

- Honest client responds with ACK ($\text{AN} = \text{SN}_S$, $\text{SN} = \text{SN}_C + 1$)
  - Server allocates space for socket only if valid $\text{SN}_S$
SYN floods: backscatter [MVS’ 01]

- SYN with forged source IP → SYN/ACK to random host
Backscatter measurement

- Listen to unused IP address space (darknet)

0 \[\text{monitor}\] \[2^{32}\]

- Lonely SYN/ACK packet likely to be result of SYN attack

- 2001: 400 SYN attacks/week
- 2013: 773 SYN attacks/24 hours  (arbor networks ATLAS)

- Larger experiments: (monitor many ISP darknets)
  - Arbor networks
Estonia attack

Attack types detected:
- 115 ICMP floods, 4 TCP SYN floods

Bandwidth:
- 12 attacks: 70-95 Mbps for over 10 hours

All attack traffic was coming from outside Estonia

Estonia’s solution:
- Estonian ISPs blocked all foreign traffic until attacks stopped
  ⇒ DoS attack had little impact inside Estonia
SYN Floods II: Massive flood
(e.g BetCris.com)

Command bot army to flood specific target: (DDoS)

- 20,000 bots can generate 2Gb/sec of SYNs (2003)
- At web site:
  - Saturates network uplink or network router
  - Random source IP ⇒
    - attack SYNs look the same as real SYNs

- What to do ???
Prolexic / CloudFlare

Idea: only forward established TCP connections to site

Lots-of-SYNs

Lots-of-SYN/ACKs

Few ACKs

Prolexic Proxy

Forward to site

Web site
Other junk packets

<table>
<thead>
<tr>
<th>Attack Packet</th>
<th>Victim Response</th>
<th>Rate: attk/day [ATLAS 2013]</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP SYN to open port</td>
<td>TCP SYN/ACK</td>
<td>773</td>
</tr>
<tr>
<td>TCP SYN to closed port</td>
<td>TCP RST</td>
<td></td>
</tr>
<tr>
<td>TCP ACK or TCP DATA</td>
<td>TCP RST</td>
<td></td>
</tr>
<tr>
<td>TCP RST</td>
<td>No response</td>
<td></td>
</tr>
<tr>
<td>TCP NULL</td>
<td>TCP RST</td>
<td></td>
</tr>
<tr>
<td>ICMP ECHO Request</td>
<td>ICMP ECHO Response</td>
<td>50</td>
</tr>
<tr>
<td>UDP to closed port</td>
<td>ICMP Port unreachable</td>
<td>387</td>
</tr>
</tbody>
</table>

Proxy must keep floods of these away from web site
Stronger attacks: TCP con flood

- Command bot army to:
  - Complete TCP connection to web site
  - Send short HTTP HEAD request
  - Repeat

- Will bypass SYN flood protection proxy

- ... but:
  - Attacker can no longer use random source IPs.
    - Reveals location of bot zombies
  - Proxy can now block or rate-limit bots.
A real-world example: GitHub

Javascript-based DDoS:

```
function imgflood() {
    var TARGET = 'victim-website.com/index.php?'
    var rand = Math.floor(Math.random() * 1000)
    var pic = new Image()
    pic.src = 'http://'+TARGET+rand+'=val'
}
setInterval(imgflood, 10)
```

Would HTTPS prevent this DDoS?
DNS DoS Attacks (e.g. bluesecurity ’06)

- DNS runs on UDP port 53
  - DNS entry for victim.com hosted at victim_isp.com

- DDoS attack:
  - flood victim_isp.com with requests for victim.com
  - Random source IP address in UDP packets

- Takes out entire DNS server: (collateral damage)
  - bluesecurity DNS hosted at Tucows DNS server
  - DNS DDoS took out Tucows hosting many many sites

- What to do ???
DNS DoS solutions

- **Generic DDoS solutions:**
  - Later on. Require major changes to DNS.

- **DoS resistant DNS design:** (e.g. CloudFlare)
  - **CoDoNS**: [Sirer’ 04]
    - Cooperative Domain Name System
  - P2P design for DNS system:
    - DNS nodes share the load
    - Simple update of DNS entries
    - Backwards compatible with existing DNS
DoS via route hijacking

- YouTube is 208.65.152.0/22 (includes $2^{10}$ IP addr)
  youtube.com is 208.65.153.238, ...

- Feb. 2008:
  - Pakistan telecom advertised a BGP path for
    208.65.153.0/24 (includes $2^8$ IP addr)
  - Routing decisions use most specific prefix
  - The entire Internet now thinks
    208.65.153.238 is in Pakistan

- Outage resolved within two hours
  ... but demonstrates huge DoS vuln. with no solution!
**DoS at higher layers**

- SSL/TLS handshake [SD’03]

- RSA-encrypt speed $\approx 10 \times$ RSA-decrypt speed

$\Rightarrow$ Single machine can bring down ten web servers

- Similar problem with application DoS:
  - Send HTTP request for some large PDF file
  - Easy work for client, hard work for server.
DoS Mitigation
1. Client puzzles

- Idea: slow down attacker

- Moderately hard problem:
  - Given challenge $C$ find $X$ such that
    $$\text{LSB}_n \left( \text{SHA-1} \left( C \ || \ X \right) \right) = 0^n$$
  - Assumption: takes expected $2^n$ time to solve
  - For $n=16$ takes about .3sec on 1Ghz machine
  - Main point: checking puzzle solution is easy.

- During DoS attack:
  - Everyone must submit puzzle solution with requests
  - When no attack: do not require puzzle solution
Examples

- **TCP connection floods** (RSA ‘99)
  - Example challenge: \( C = \text{TCP server-seq-num} \)
  - First data packet must contain puzzle solution
    - Otherwise TCP connection is closed

- **SSL handshake DoS**: (SD’03)
  - Challenge C based on TLS session ID
  - Server: check puzzle solution before RSA decrypt.

- Same for application layer DoS and payment DoS.
Benefits and limitations

- Hardness of challenge: n
  - Decided based on DoS attack volume.

- Limitations:
  - Requires changes to both clients and servers
  - Hurts low power legitimate clients during attack:
    - Clients on cell phones and tablets cannot connect
Memory-bound functions

- CPU power ratio:
  - high end server / low end cell phone = 8000
  ⇒ Impossible to scale to hard puzzles

- Interesting observation:
  - Main memory access time ratio:
    - high end server / low end cell phone = 2

- Better puzzles:
  - Solution requires many main memory accesses
    - Dwork-Goldberg-Naor, Crypto ‘03
    - Abadi-Burrows-Manasse-Wobber, ACM ToIT ‘05
2. CAPTCHAs

- Idea: verify that connection is from a human

- Applies to application layer DDoS [Killbots ’05]
  - During attack: generate CAPTCHAs and process request only if valid solution
  - Present one CAPTCHA per source IP address.
3. Source identification

Goal: identify packet source

Ultimate goal: block attack at the source
1. Ingress filtering  (RFC 2827, 3704)

- Big problem: DDoS with spoofed source IPs

- Ingress filtering policy: ISP only forwards packets with legitimate source IP (see also SAVE protocol)
Implementation problems

ALL ISPs must do this. Requires global trust.
- If 10% of ISPs do not implement \(\Rightarrow\) no defense
- No incentive for deployment

2014:
- 25% of Auto. Systems are fully spoofable (spoofer.cmand.org)
- 13% of announced IP address space is spoofable

Recall: 309 Gbps attack used only 3 networks (3/2013)
2. Traceback  [Savage et al. ’00]

**Goal:**
- Given set of attack packets
- Determine path to source

**How:** change routers to record info in packets

**Assumptions:**
- Most routers remain uncompromised
- Attacker sends many packets
- Route from attacker to victim remains relatively stable
Simple method

- Write path into network packet
  - Each router adds its own IP address to packet
  - Victim reads path from packet

Problem:
- Requires space in packet
  - Path can be long
  - No extra fields in current IP format
    - Changes to packet format too much to expect
Better idea

- DDoS involves many packets on same path
- Store one link in each packet
  - Each router probabilistically stores own address
  - Fixed space regardless of path length
Edge Sampling

- Data fields written to packet:
  - Edge: *start* and *end* IP addresses
  - Distance: number of hops since edge stored

- Marking procedure for router R
  - if coin turns up heads (with probability $p$) then
    - write R into start address
    - write 0 into distance field
  - else
    - if distance == 0 write R into end field
    - increment distance field
Packet received

- $R_1$ receives packet from source or another router
- Packet contains space for start, end, distance
Edge Sampling: picture

- Begin writing edge
  - $R_1$ chooses to write start of edge
  - Sets distance to 0

![Diagram of edge sampling process]

1. $R_1$ writes the start of the edge
2. Sets distance to 0
Edge Sampling

- Finish writing edge
  - $R_2$ chooses not to overwrite edge
  - Distance is 0
    - Write end of edge, increment distance to 1
Edge Sampling

- Increment distance
  - $R_3$ chooses not to overwrite edge
  - Distance $>0$
    - Increment distance to 2
Path reconstruction

- Extract information from attack packets
- Build graph rooted at victim
  - Each (start,end,distance) tuple provides an edge
- # packets needed to reconstruct path

\[
E(X) < \frac{\ln(d)}{p(1-p)^{d-1}}
\]

where \( p \) is marking probability, \( d \) is length of path
Details: where to store edge

- Identification field
  - Used for fragmentation
  - Fragmentation is rare
  - 16 bits

- Store edge in 16 bits?
  - Break into chunks
  - Store start and end

<table>
<thead>
<tr>
<th>offset</th>
<th>distance</th>
<th>edge chunk</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>Header Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Service</td>
<td></td>
</tr>
<tr>
<td>Total Length</td>
<td></td>
</tr>
<tr>
<td>Identification</td>
<td></td>
</tr>
<tr>
<td>Flags</td>
<td>Fragment Offset</td>
</tr>
<tr>
<td>Time to Live</td>
<td></td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
</tr>
<tr>
<td>Header Checksum</td>
<td></td>
</tr>
<tr>
<td>Source Address of Originating Host</td>
<td></td>
</tr>
<tr>
<td>Destination Address of Target Host</td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td></td>
</tr>
<tr>
<td>Padding</td>
<td></td>
</tr>
<tr>
<td>IP Data</td>
<td></td>
</tr>
</tbody>
</table>
More traceback proposals

- Advanced and Authenticated Marking Schemes for IP Traceback
  - Song, Perrig. IEEE Infocomm ’01
  - Reduces noisy data and time to reconstruct paths

- An algebraic approach to IP traceback
  - Stubblefield, Dean, Franklin. NDSS ’02

- Hash-Based IP Traceback
  - Snoeren, Partridge, Sanchez, Jones, Tchakountio, Kent, Strayer. SIGCOMM ‘01
Problem: Reflectors attacks [Paxson ’01]

- **Reflector:**
  - A network component that responds to packets
  - Response sent to victim (spoofed source IP)

- **Examples:**
  - DNS Resolvers: UDP 53 with victim.com source
    - At victim: DNS response
  - Web servers: TCP SYN 80 with victim.com source
    - At victim: TCP SYN ACK packet
  - Gnutella servers
DoS Attack

- Single Master
- Many bots to generate flood
- Zillions of reflectors to hide bots
  - Kills traceback and pushback methods
Capability based defense
Capability based defense

- Anderson, Roscoe, Wetherall.  
  - Preventing internet denial-of-service with capabilities.  SIGCOMM ‘04.

- Yaar, Perrig, and Song.  

- Yang, Wetherall, Anderson.  
  - A DoS-limiting network architecture.  SIGCOMM ’05
Capability based defense

Basic idea:
- Receivers can specify what packets they want

How:
- Sender requests capability in SYN packet
  - Path identifier used to limit # reqs from one source
- Receiver responds with capability
- Sender includes capability in all future packets

Main point: Routers only forward:
  - Request packets, and
  - Packets with valid capability
Capability based defense

- Capabilities can be revoked if source is attacking
  - Blocks attack packets close to source

Source AS → R₁ → R₂ → R₃ → R₄ → dest

- Attack packets dropped

Transit AS

Dest AS
Pushback Traffic Filtering
Pushback filtering


- Ioannidis, Bellovin. Implementing Pushback: Router-Based Defense Against DoS Attacks. *NDSS* ’02

Pushback Traffic Filtering

Assumption: DoS attack from few sources

Iteratively block attacking network segments.
Overlay filtering
Overlay filtering

Keromytis, Misra, Rubenstein. SOS: Secure Overlay Services. SIGCOMM ’02.


Take home message:

- Denial of Service attacks are real. Must be considered at design time.

- Sad truth:
  - Internet is ill-equipped to handle DDoS attacks
  - Commercial solutions: CloudFlare, Prolexic

- Many good proposals for core redesign.
THE END