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

1 ICMP Echo Req
Src: Dos Target
Dest: brdct addr

3 ICMP Echo Reply
Dest: Dos Target

- 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: \( \times 50 \) amplification

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

Figure 13
Source: Arbor Networks, Inc.
Review: IP Header format

- Connectionless
  - Unreliable
  - Best effort

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4</td>
</tr>
<tr>
<td>Header Length</td>
<td>4</td>
</tr>
<tr>
<td>Type of Service</td>
<td>8</td>
</tr>
<tr>
<td>Total Length</td>
<td>16</td>
</tr>
<tr>
<td>Identification</td>
<td>16</td>
</tr>
<tr>
<td>Flags</td>
<td>3</td>
</tr>
<tr>
<td>Fragment Offset</td>
<td>13</td>
</tr>
<tr>
<td>Time to Live</td>
<td>8</td>
</tr>
<tr>
<td>Protocol</td>
<td>8</td>
</tr>
<tr>
<td>Header Checksum</td>
<td>16</td>
</tr>
<tr>
<td>Source Address</td>
<td>32</td>
</tr>
<tr>
<td>Destination Address</td>
<td>32</td>
</tr>
<tr>
<td>Options</td>
<td>0</td>
</tr>
<tr>
<td>Padding</td>
<td>0</td>
</tr>
<tr>
<td>IP Data</td>
<td>0</td>
</tr>
</tbody>
</table>
Review: TCP Header format

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

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Dest port</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ Number</td>
<td>ACK Number</td>
</tr>
</tbody>
</table>

Fields:
- URG
- ACK
- SYN
- FIN
- Other stuff
Review: TCP Handshake

**SYN:**
- $S_{NC} \leftarrow \text{rand}_{C}$
- $A_{NC} \leftarrow 0$

**SYN/ACK:**
- $S_{NS} \leftarrow \text{rand}_{S}$
- $A_{NS} \leftarrow S_{NC}$

**ACK:**
- $S_{N} \leftarrow S_{NC}$
- $A_{N} \leftarrow S_{NS}$

**States:**
- Listening
- Wait
- Established
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 need only send 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 16th:
  - SYN flood on port 80 to `windowsupdate.com`
  - 50 SYN packets every second.
    - Each packet is 40 bytes.

**MS solution:**
- New name: `windowsupdate.microsoft.com`
- Win update file delivered by Akamai
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 ($AN=\text{SN}_S$, $SN=\text{SN}_C+1$)
  - 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 [MVS’ 01]

- Listen to unused IP addresses space (darknet)

\[
\begin{array}{c}
0 \quad /8 \text{ network} \\
\text{monitor} \\
2^{32}
\end{array}
\]

- 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 (ATLAS ‘07)

- 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 ‘03)

- 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

![Diagram showing the flow of network traffic through a Prolexic Proxy]

- Lots-of-SYNs
- Lots-of-SYN/ACKs
- Few ACKs
- Forward to 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.
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]

![Diagram of SSL/TLS handshake]

- RSA-encrypt speed \( \approx 10 \times \) RSA-decrypt speed

\[ \Rightarrow \text{Single machine can bring down ten web servers} \]

- Similar problem with application DoS:
  - Send HTTP request for some large PDF file

\[ \Rightarrow \text{Easy work for client, hard work for server.} \]
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 .3 sec 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 ⇒ 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

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: \textit{start} and \textit{end} IP addresses
  - Distance: number of hops since edge stored

- Marking procedure for router R

\[
\begin{align*}
\text{if coin turns up heads (with probability } p \text{) then} \\
& \quad \text{write } R \text{ into start address} \\
& \quad \text{write 0 into distance field}
\end{align*}
\]

\[
\begin{align*}
\text{else} \\
& \quad \text{if distance } == \ 0 \text{ write } R \text{ into end field} \\
& \quad \text{increment distance field}
\end{align*}
\]
Edge Sampling: picture

- 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
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

Diagram:
- $R_1$ to $R_2$ to $R_3$ with a packet containing $R_1$, $R_2$, and 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 $\oplus$ 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>
<tr>
<td>7</td>
<td>8</td>
<td>15</td>
</tr>
</tbody>
</table>

- Store in 16 bits?
  - Break into chunks
  - Store start $\oplus$ end
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: Reflector 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 $\rightarrow$ $R_1$ $\rightarrow$ $R_2$ $\rightarrow$ $R_3$ $\rightarrow$ $R_4$ $\rightarrow$ dest

Attack packets dropped
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


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