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
- DoS mitigations

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 (Feb ’18)

memcached amplification attack: ($ \times 51K$ amplification)

2018: 87,000 exposed memcached servers

⇒ Feb. 2018: 1.35 Tbps attack on GitHub

Simple solution: disable Memcached over UDP (no attack over TCP)
Modern day example

Same attack using other protocols: DNS, NTP, ... (over UDP)

• **DNS amplification:** short DNS query, large response
  - 2006: 0.58M open resolvers on Internet (Kaminsky-Shiffman)
  - 2017: 15M open resolvers (openresolverproject.org)

⇒ 3/2013: DDoS attack generating **309 Gbps** for 28 mins.
  31,000 open DNS resolvers, each outputting 10Mbps.
  Source: 3 networks that allowed source IP spoofing.

• **NTP amplification:**
  - 2014: **400 Gbps** (4500 NTP servers)
Scale, Targeting and Frequency of Attacks

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>Bit Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0-3</td>
</tr>
<tr>
<td>Header Length</td>
<td>4</td>
</tr>
<tr>
<td>Type of Service</td>
<td>5-7</td>
</tr>
<tr>
<td>Total Length</td>
<td>8-15</td>
</tr>
<tr>
<td>Identification</td>
<td>16-19</td>
</tr>
<tr>
<td>Flags</td>
<td>20-21</td>
</tr>
<tr>
<td>Fragment Offset</td>
<td>22-29</td>
</tr>
<tr>
<td>Time to Live</td>
<td>30-31</td>
</tr>
<tr>
<td>Protocol</td>
<td>32-36</td>
</tr>
<tr>
<td>Header Checksum</td>
<td>37-39</td>
</tr>
<tr>
<td>Source Address of Originating Host</td>
<td>40-47</td>
</tr>
<tr>
<td>Destination Address of Target Host</td>
<td>48-55</td>
</tr>
<tr>
<td>Options</td>
<td>56-63</td>
</tr>
<tr>
<td>Padding</td>
<td>64-67</td>
</tr>
<tr>
<td>IP Data</td>
<td>68-69</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>
<tr>
<td>U R G A C K</td>
<td>P S H P S Y N F I N</td>
</tr>
</tbody>
</table>

Other stuff
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**
- **Waiting**
- **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 needs only 128 SYN packets every 3 minutes
- Low rate SYN flood
Low rate SYN flood defenses

The problem:
- server commits resources (memory) before client responds

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

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

- 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.
  - $L = \text{MAC}_{\text{key}} (S\text{Addr}, S\text{Port}, D\text{Addr}, D\text{Port}, S\text{N}_C, T)$
    - key: picked at random during boot
  - $S\text{N}_S = (T \cdot \text{mss} \cdot L)$ \hspace{1cm} (|$L| = 24$ bits)
  - Server does not save state (other TCP options are lost)

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

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

- Listen to unused IP addresses space (darknet)

- 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
Massive floods (e.g. Mirai 9/2016 on Krebs)

Command bot army to flood specific target: (DDoS)
- Flood with SYN, ACK, UDP, and GRE packets
- 623 Gbps (peak) from ≈100K compromised IoT devices
- At web site:
  - Saturates network uplink or network router
  - Random source IP ⇒ attack SYNks look the same as real SYNks
- What to do ???

20
<table>
<thead>
<tr>
<th>Country</th>
<th>% of Mirai botnet IPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vietnam</td>
<td>12.8%</td>
</tr>
<tr>
<td>Brazil</td>
<td>11.8%</td>
</tr>
<tr>
<td>United States</td>
<td>10.9%</td>
</tr>
<tr>
<td>China</td>
<td>8.8%</td>
</tr>
<tr>
<td>Mexico</td>
<td>8.4%</td>
</tr>
<tr>
<td>South Korea</td>
<td>6.2%</td>
</tr>
<tr>
<td>Taiwan</td>
<td>4.9%</td>
</tr>
<tr>
<td>Russia</td>
<td>4.0%</td>
</tr>
<tr>
<td>Romania</td>
<td>2.3%</td>
</tr>
<tr>
<td>Colombia</td>
<td>1.5%</td>
</tr>
</tbody>
</table>

Figure 3: Top countries of origin of Mirai DDoS attacks
Google project shield

Protecting news organizations.

(Commercial service: Akamai, Cloudflare, ...)

Idea: only forward established TCP connections to site

Lots-of-SYNs

Lots-of-SYN/ACKs

Few ACKs

Forward to site

Web site
Stronger attacks: GET flood

- Command bot army to:
  - Complete TCP connection to web site
  - Send short HTTP GET 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. Dyn attack 10/2016)

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

DDoS attack:
- flood DNSProvider.com with DNS queries
- **Random source IP address** in UDP packets
- Takes out entire DNS server (collateral damage)

Dyn attack: used some Mirai-based bots
- At least 100,000 malicious end points
  - Dyn cannot answer many legit DNS queries
  - Disrupted service at Netflix, Github, Twitter, ...
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 Mitigation
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

2017:
- 33% of Auto. Systems are fully spoofable (spoofer.caida.org)
- 23% of announced IP address space is spoofable

Recall: 309 Gbps attack used only 3 networks (3/2013)
2. Client puzzles

- **Idea:** slow down attacker

- **Moderately hard problem:**
  - Given challenge \( C \) find \( X \) such that
  \[
  \text{LSB}_n\left(\text{SHA-1}(C \ || \ X)\right) = 0^n
  \]
  - 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

- **GET 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.
Benefits and limitations

Hardness of challenge:  
- 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 IoT device = 8000
  \[ \Rightarrow \] Impossible to scale to hard puzzles

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

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

- Idea: verify that connection is from a human

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

Goal: identify packet source

Ultimate goal: block attack at the source
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: \textit{start} and \textit{end} IP addresses
  - Distance: number of hops since edge stored

- Marking procedure for router R
  
  \[
  \text{if coin turns up heads (with probability p) then} \]
  
  write R into start address
  write 0 into distance field
  
  \[
  \text{else} \]
  
  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
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
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
  - NTP servers
DoS Attack

- Single Master
- Many bots to generate flood
- Zillions of reflectors to hide bots
  - Kills traceback and pushback methods
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
  - Many commercial solutions: CloudFlare, Akamai, ...

- Many proposals for core redesign
THE END