Resilient Networking

Module 6: DOS

Disclaimer: this course has been created with very valuable input from Günter Schäfer, Mathias Fischer, and the members of the Chair.

Dresden, SS 18
Lecture Outline

DoS background
Classification of DoS attacks
DoS examples
Countermeasures
Identifying the attacker
The Threat...

Honey! I think our network is having another Smurf attack!

(source: Julie Sigwart - "Geeks")
Introduction

What is Denial of Service?
— Denial of Service (DoS) attacks aim at **denying** or **degrading** legitimate users’ access to a service or network resource, or at bringing down the servers offering such services.

Motivations for launching DoS attacks:
— Hacking (just for fun, by “script kiddies”, ...)
— Gaining information leap (→ 1997 attack on bureau of labor statistics server; was possibly launched as unemployment information has implications to the stock market)
— Discrediting an organization operating a system (i.e. web server)
— Revenge (personal, against a company, ...)
— Political reasons (“information warfare”)
Denial of Service Attack Classes

Resource **destruction**:  
— Hacking into systems  
— Making use of implementation weaknesses like buffer overflows  
— Deviation from proper protocol execution  
— Your common TU Dresden Excavator

Resource **depletion** by causing:  
— Storage of (useless) state information  
— High traffic load (requires high overall bandwidth from attacker)  
— Expensive computations ("expensive cryptography"!)  
— Resource reservations that are never used (e.g. bandwidth)
Denial of Service Attacking Techniques

Target resources at the service, or on path for service provision

- Network connectivity (uplink, transit link)
- Computation
- Memory

Classes of DoS: Origin of malicious traffic

- Single source with single / multiple (forged) source addresses
- Multiple sources (Distributed DoS)

Amplification attacks: Leverage asymmetry in protocols

- Send lightweight requests (low cost) that generate heavyweight responses
- Increases damage

Reflector attacks: Generate traffic indirection

- Request service in the name of the victim (e.g. spoofed IP – which protocols?)
- Hides attack source, allows for external amplification
The attacker classifies the compromised systems in:
- Master systems
- Slave systems

Master systems:
- Receive command data from attacker
- Control the slaves

Slave systems:
- Launch the proper attack against the victim
- During the attack there is no traffic from the attacker

--- Control Traffic  Attack Traffic
Botnet Strategies: Partitioning

Each master system only knows some slave systems

Therefore, the network can handle partial failure, caused by detection of some slaves or masters
DDoS – Direct / Reflector

Network Topologies/Attack Types

a.) Master-Slave-Victim
b.) Master-Slave-Reflector-Victim

Side Note: Reflector ≠ Amplification!
Examples: Resource Destruction (1)

Resource Destruction:
Physically/Logically destroy a resource that is vital for targeted service

Hacking:
— Exploiting weaknesses that are caused by careless operation of a system
— Examples: default accounts and passwords not disabled, badly chosen passwords, social engineering (incl. email worms), etc.

Making use of implementation weaknesses
— Buffer Overflows, Format-String-Attacks, ...

Deviation from proper protocol execution:
— Example: exploit IP's fragmentation & reassembly
Examples: Resource Destruction (2)

Example (Teardrop attack): exploit IP’s fragmentation & reassembly
—Send IP fragments to broadcast address

—Operating systems with origins in BSD often respond to this address as a broadcast address

—in order to respond, the packets have to be reassembled first

—if an attacker sends a lot of fragments without ever sending a first / last fragment, the buffer of the reassembling system gets overloaded

Note: Routers use BSD-based TCP/IP stacks -> attack on network infrastructure

More recently: 😂;-)
Defending Against Resource Destruction DoS

Defenses against disabling services:

Hacking:
- Good system administration
- Firewalls, logging & intrusion detection systems

Implementation weakness:
- Code reviews, stress testing, etc.

Protocol deviation:
- Fault tolerant protocol design
- Error logging & intrusion detection systems
- “DoS-aware protocol design”:
  - Be aware of possible DoS attacks when reassembling packets
  - Do not perform expensive operations, reserve memory, etc., before authentication
Resource Depletion - Examples
Background: Internet Control Message Protocol

ICMP: Signaling of error conditions on the Internet
PDUs are transported as IP packets
Payload of IP, identified by value “1” in the protocol field of the IP header

ICMP message:

```
<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unused</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internet Header + 64 bits of Original Data Datagram</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
+--------------------------------------------------------------+
|              Type               |    Code            |    Checksum            |
+--------------------------------------------------------------+
0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
+--------------------------------------------------------------+
|                             unused                             |
+--------------------------------------------------------------+
| Internet Header + 64 bits of Original Data Datagram           |
+--------------------------------------------------------------+
Smurf – ICMP Bandwidth Depletion

Two reasons make ICMP particular interesting for attackers:
— It may be addressed to broadcast addresses
— Routers respond to it

The **Smurf attack** - ICMP echo request to broadcast:
— Routers (sometimes) allow ICMP echo requests to broadcast addresses...
— An attacker sends an ICMP echo request to a *broadcast address* with the *source address* forged to refer to the victim
— All devices in the addressed network respond to the packet
— The victim is flooded with replies to the echo request
— With this technique, the network being abused as an (unaware) attack amplifier is also called a *reflector network*:
The *Transmission Control Protocol (TCP)*:
— provides a connection-oriented, reliable transport service
— uses IP for transport of its PDUs

TCP connection establishment is realized with handshake:

- After handshake, data can be exchanged in both directions
- Both peers may initiate termination of the connection (two-way-handshake)
TCP Connection Management: State Diagram

Stimulus / Reaction (e.g. Receive / Send)

CLOSED

LISTEN

SYN_RCVD

SYN_SENT

ESTABLISHED

FIN_WAIT_1

FIN_WAIT_2

CLOSING

TIME_WAIT

CLOSE_WAIT

LAST_ACK

CLOSED

Passive open

Close

Active open / SYN

SYN/SYN + ACK

SYN + ACK/ACK

Send/SYN

SYN/SYN + ACK

ACK

Close/FIN

FIN/ACK

Timeout after two segment lifetimes

Note: some states are “superstates”, actually containing their own state machine
### Background: Reaction According to Protocol

Reply packets according to protocol specification if state not available

<table>
<thead>
<tr>
<th>Packet Sent</th>
<th>Reaction of Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP SYN (to open port)</td>
<td>TCP SYN ACK</td>
</tr>
<tr>
<td>TCP SYN (to closed port)</td>
<td>TCP RST (ACK)</td>
</tr>
<tr>
<td>TCP ACK</td>
<td>TCP RST (ACK)</td>
</tr>
<tr>
<td>TCP DATA</td>
<td>TCP RST (ACK)</td>
</tr>
<tr>
<td>TCP RST</td>
<td>no response</td>
</tr>
<tr>
<td>TCP NULL</td>
<td>TCP RST (ACK)</td>
</tr>
<tr>
<td>ICMP Echo Request</td>
<td>ICMP Echo Reply</td>
</tr>
<tr>
<td>ICMP TS Request</td>
<td>ICMP TS Reply</td>
</tr>
<tr>
<td>UDP Packet (to open port)</td>
<td>protocol dependent</td>
</tr>
<tr>
<td>UDP Packet (to closed port)</td>
<td>ICMP Port Unreachable</td>
</tr>
<tr>
<td>TCP SYN ACK (to closed port)</td>
<td>----</td>
</tr>
</tbody>
</table>
TCP SYN Flood: Memory Depletion

Category *Storage of useless state information:*

- Here: TCP-SYN flood attack

TCP SYN packets with forged source addresses ("SYN Flood")

TCP SYN ACK packet to assumed initiator ("Backscatter")
DDoS: CPU Exhaustion

Category *CPU exhaustion by expensive computations:*  
— Here: attacking with bogus authentication attempts

---

The attacker usually either needs to receive or guess some values of the second message, that have to be included in the third message for the attack to be successful

— Also, the attacker, must trick the victim repeatedly to perform the expensive computation in order to cause significant damage
Background on Authentication (1)

Definition:
A cryptographic protocol is defined as a **series of steps** and **message exchanges** between **multiple entities** in order to achieve a specific **security objective**

Properties of a protocol (in general):

- **Everyone** involved in the protocol must **know** the protocol and **all** of the **steps** to follow in advance
- **Everyone** involved in the protocol **must agree to follow it**
- The protocol must be **unambiguous**, that is every step is well defined and there is no chance of misunderstanding
- The protocol must be **complete**, i.e. there is a specified action for every possible situation

Additional property of a cryptographic protocol:
- It should not be possible to do or learn more than what is specified in the protocol
Basic variants of authentication:

*Data origin authentication* is the security service that enables entities to verify that a *message* has been *originated* by a *particular entity* and that it has *not* been *altered* afterwards (synonym for this service: *data integrity*)

*Entity authentication* is the security service, that enables communication partners to *verify* the *identity* of their *peer entities*
Entity authentication is more than an exchange of (data-origin-) authentic messages, because of **timeliness**:— Even if Bob receives authentic messages from Alice during a communication, he can not be sure, if:

- Alice is actually participating in the communication **in this specific moment**, or if
- Eve is **replaying** old messages from Alice

— Significant, when authentication is only performed at connection-setup:

- Example: transmission of a (possibly encrypted) PIN when logging in

— Two principle means to ensure timeliness in cryptographic protocols:

- **Timestamps** (require more or less synchronized clocks)
- **Random numbers / Nonces** (challenge-response exchanges)

Most authentication protocols establish a **secret session key** for **securing the session** following the authentication exchange...
Background: Secure Socket Layer (SSL)

SSL was designed in the early 1990’s to primarily protect HTTP sessions and it provides the following security services:

**Peer entity authentication:**
- *Prior to any communications* between a client and a server, an *authentication protocol* is run to authenticate the peer entities
- Upon successful completion of the authentication dialogue an *SSL session* is established between the peer entities

**User data confidentiality:**
- If negotiated upon session establishment, user *data is encrypted*
- Different encryption algorithms can be negotiated: RC4, 3DES, AES, ...

**User data integrity:**
- *HMAC* based on a cryptographic hash function is appended to user data
- The MAC is computed with a negotiated secret in prefix-suffix mode
- Either MD5 or SHA can be negotiated for MAC computation
SSL Authentication: Full Handshake

Client

ClientHello


Server

ServerHello [ServerCertificate] [CertificateRequest] [ServerKeyExchange] ServerHelloDone

[ChangeCipherSpec Finished]

[...] denotes optional messages
SSL CPU-Depletion

Resilient Networking: Denial of Service
Privacy and Security – Thorsten Strufe
Defending Against Resource Depletion DoS

Defenses against resource depletion:

Generally:
— **Rate Control** (ensures availability of other functions on same system)
— Authentication & Accounting

Expensive computations: careful protocol design, verifying the initiator’s “willingness” to spend resources himself (e.g. “client puzzles”)

Memory exhaustion: stateless protocol operation
Attack Sources and Spoofed Addresses

Concerning origin of malicious traffic:

Defenses against single source attacks:
— Disabling of address ranges (helps if addresses are valid)

Defenses against forged source addresses:
— Ingress Filtering at ISPs (if the world was an ideal one...)
— “Verify” source of traffic (e.g. with exchange of “cookies”)
— Tracing back the true source of packets with spoofed addresses

Widely distributed DoS:
— Offloading to Site Delivery Services/CDN
Memory Exhaustion: Stateless Protocols

Basic idea:
- Avoid storing information at server, before DoS attack can be ruled out
- So, as long as no assurance regarding the client has been reached all state is “stored” in the network (transferred back and forth)

<table>
<thead>
<tr>
<th>Stateful Operation</th>
<th>Stateless Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. C → S: Msg₁</td>
<td>1. C → S: Msg₁</td>
</tr>
<tr>
<td>2. S → C: Msg₂</td>
<td>2. S → C: Msg₂, State&lt;sub&gt;S₁&lt;/sub&gt;</td>
</tr>
<tr>
<td>3. C → S: Msg₃</td>
<td>3. C → S: Msg₃, State&lt;sub&gt;S₁&lt;/sub&gt;</td>
</tr>
<tr>
<td>4. S → C: Msg₄</td>
<td>4. S → C: Msg₄, State&lt;sub&gt;S₂&lt;/sub&gt;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Drawback: requires higher bandwidth and more message processing
**CPU Exhaustion: Client Puzzles/Proof of Work**

*Observations and assumptions:*
DoS (also: spam) works because there’s no postage paid (cost) when message is sent
Amplification attacks require few resources at client and cause large load at victim

*Proof of Work:* level the playing fields by making the clients prove that they invested resources
One-way functions are cheap to evaluate, but “impossible” to invert

Good (as any) approach to inversion is guessing, partial guessing may be possible:
— Chances to guess $x$ such that
\[ P[H(x) = yyyyyy0] = 0.5 \]

what about $P[H(x) = yyyy000]$? ;-)  

*Simple Client Puzzles:*
Let server draw a pre-image at random
Provide client with image and request it to provide the pre-image
Countering CPU Exhaustion with Client Puzzles (2)

Basic properties of a client puzzle as required by Aura et al.:
the creation and verification of a puzzle is **inexpensive to a server**, the server can **adjust** the **cost** of solving a puzzle (from zero to “impossible”), the **puzzle can be solved** on most types of client hardware, the **pre-computation** of solutions is **impossible**, the server does **not need to store** any client-specific data while the client solves the puzzle, the **same puzzle** may be given to several clients, while ensuring that knowing the solution of one or more clients does not help a new client in solving the puzzle, a client can **reuse** a **puzzle** by **creating several instances** of it, however, the **solution** to a puzzle **should not be reusable**
Countering CPU Exhaustion with Client Puzzles (3)

Reusable client puzzles according to Aura et al:

1. Server periodically broadcasts random number $N_s$ and difficulty level $k$
2. Every client $C$ can then create a solution to a new instance of this puzzle by:
   • Generating a fresh random number $N_C$
   • Determining with brute force search (= trying all possible values) an $X$ such that:

\[
H(C, N_s, N_C, X) = 00000Y_k
\]

Summary:

• Client puzzles provide an effective means to slow down potential DoS attackers significantly
• At the same time, the length of messages is only increased minimally (about one byte for parameter $k$ and up to eight bytes for the solution $X$)
• This may protect servers at the early stage of a normal authentication where the computations are the most CPU intensive

Aura, Tuomas, Pekka Nikander, Jussipekka Leiwo, "DOS-resistant authentication with client puzzles." Workshop on security protocols. 2000
Verifying the Source of a Request

Problem: Spoofed addresses allow adversaries to hide

Basic solution:
— Before working on a new request, verify if the “initiator” can receive messages, sent to the claimed source of the request
— Only a legitimate client or an attacker which can receive the “cookie”, can send the cookie back to the server
— Of course, an attacker must not be able to guess the content of a cookie

Discussion:
— Advantage: allows to counter simple spoofing attacks
— Drawback: requires one additional message roundtrip
But...

Verifying the source of a request with a cookie exchange can **avoid spending significant computation** or **memory** resources on a bogus request.

What if the attacker is only interested in **exhausting** the access or packet processing **bandwidth** of a victim?

— Obviously, sending cookies to all incoming packets even aggravates the situation!
— Such an attack situation, however, is quite easy to detect: there are simply too many packets coming in.

Problems in such a case:

— Which packets come from **genuine sources** and which are **bogus ones**?
— Even worse: source addresses given in the packets may be spoofed
— Where do the spoofed packets come from?
IP-Address Spoofing

Reprise: DoS-/ DDoS-Attacks
— Direct Attacks (Master – network of slaves)
  • Problem of spoofed source addresses of attack packets sent by the slaves
— Reflector Attacks (Master – (slaves –) reflecting nodes)
  • Problem of address-spoofing: set victims' IP-address as source

Main problem is the possibility to lie about the source address...
Possible Solutions to DDoS-Attacks (1)

Solutions to **Reflector Attacks**: secure available services
— Prevent amplification: Balance effort of request and reply
  e.g.: Prohibit ICMP-Echo-Request to broadcast addresses

• => Reflectors don’t amplify attack magnitude

  (however: does this work with all protocols? DNS?)

— Access-controlled services: provide service to authorized parties only
  e.g.: Prohibit recursive DNS queries for external users
Possible Solutions to DDoS-Attacks (2)

Possible Solutions to *Direct Attacks*:
— Avoid IP-Address spoofing
— Live with spoofed addresses and restrain effect of attacks

• Locate source of attack-packets
• Filter traffic from attacking nodes
• Inform admin/root of attacking networks/node

But: IP is connectionless! Necessary to find means to trace back the traffic to the original source / attacking node!

Identify: zombie, spoofed address, ingress router, routers on path...
Inhibiting Spoofed Addresses: Ingress Filtering (RFC 2267)

Routers block arriving packets with illegitimate source addresses.

IETF BCP 38 (May 2000)
Ingress Filtering (2)

Difficult in the backbone *(how to check if route is valid?)*

Easily possible at access links → ISPs

Problems occur:
— Issues with Mobile-IP (theoretic) and load testing (local)
— Large management overhead at router-level
— Processing overhead at access routers
• (e.g., big ISP running a large AS with numerous IP-Ranges and DHCP)
— Universal deployment needed (cf. the situation today...)

*ISPs don’t really have an incentive in blocking any traffic*
Identify Malicious Nodes: DDoS Attack-Tree

Rooted Tree with
— Victim (V) (root of the tree)
— Routers (R)
— Attackers (A)

Questions with forged IP addresses:
- Where are malicious nodes?
- Which router (ISP) is on attack path?
Identifying Malicious Nodes: Assumptions

Packets are subject to **reordering and loss**

*Resources* at routers are **limited**

*Routers* are usually **not compromised**

Attackers may **generate any packet**

Attackers are **aware of tracing**

**Multitude of attacking packets** (usually many)

*Routes* between A and V are **stable** (in the order of seconds)

Multiple attackers can act in **collusion**
Identify Malicious Nodes: Proposed Solutions

Simple classification of solutions:

Network Logging
  • Log information on processed packets and path

Attack Path Traceback
  • Trace attack path through network

Other / Related
  • Attack Mitigation/Avoidance
Requirements / Evaluation Metrics

1. Involvement of ISP (required or not)
2. Amount of necessary packets to trace attack
3. Effect of partial deployment
4. Resource overhead
   - Processing overhead at routers
   - Memory requirements
   - Bandwidth overhead
5. Ease of Evasion
6. Protection
7. Scalability
8. Performance towards Distributed DoS
9. Performance towards packet transformations
Involvement of ISP

ISPs don’t really have an incentive in preventing „attack-traffic“:
• Paid by number of transmitted bytes
• Receive complaints about service failures (churn!)
• Which traffic is „malicious“ and which is not?
• „Malicious“ for whom?

Incentives of ISPs:
• Infrastructure is expensive
• Management-/ down times are expensive
• Administrators are expensive
Amount of Packets Needed to Track Source

Different types of attacks:

Bandwidth resource exhaustion
  — Continuous stream of packets for the time span of the attack
  — Packet flood to bring link / host down
  — One attacker / multiple attackers (multiple attack paths)

Well targeted packets (resource destruction, e.g. Teardrop attack)

Which attacker can be traced?
Effect of Partial Deployment

What if only a few ISPs deploy the mechanism (at first)?

Still *some* benefit?
— Attackers in the deploying ISPs traceable?
— Ingress of attack packets traceable?
— Cooperation of “islands“ possible – gain in knowledge if two ISPs deploy mechanism which are connected through a third transit domain?
Resource Overhead

Resources in the network are scarce (memory, processing)!

How much processing overhead is implied for the routers
— Additional packet analysis
— Additional functions

How much information has to be stored at routers / in the network
— Log of all processed packets?

If mechanism needs communication:
— In band / out of band?
— How much extra bandwidth is needed to distribute information?
Ease of Evasion, Protection & Scalability

Ease of Evasion:
— How easy is it for an attacker to evade the mechanism?
— Can the attacker send special packets that mislead the mechanism?
• To stay transparent
• To mislead an investigator
• Attack the mechanism itself

Protection:
— What if an attacker subverts one or many network elements on the path: Can the mechanism still produce meaningful results?

Scalability:
— Does the mechanism scale with growing network sizes?
— How much extra configuration is needed (only at new, or at all devices?)
— How much do the elements depend on each other?
Performance: DDoS and Packet Transformation

Ability to handle DDoS:
— Can the mechanism produce meaningful results, if a victim is attacked on different paths?

Ability to handle packet transformation:
— Does the mechanism produce meaningful results (results at all) if the packets are transformed due to:
  • Network Address Translation (NAT)
  • Packet fragmentation
  • Packet duplication
  • Tunneling
Identifying Malicious Nodes: Proposed Solutions

Network Logging
- Local network logging
- Aggregated network logging
- Source Path Identification ("Hash-based IP-Traceback")

Attack Path Traceback
- Input Debugging
- Controlled Flooding
- ICMP Traceback
- Probabilistic Packet Marking ("IP-Traceback")

Other / Related
- Hop-Count Filtering
- Aggregate Based Congestion Control (ACC)
- Secure Overlay Services
Logging Approaches

Log information on processed packets and path

Network logging
— Local network logging:
• All routers log all traffic
• Too much overhead!
• **Does not scale**
  — Aggregated network logging
  — Source Path Identification („Hash-based IP-Traceback“)
Centralized approach:
— Introduction of „Tracking Router“ (TR)
— Forwarding all traffic through TR (via GRE)
— TR logs all traversing traffic
— Creates one single point of failure! Does not scale! *(Although: SDN...)*

[Stone: „Centertrack: An IP Overlay Network for Tracking DoS Floods“]
Source Path Identification

Source Path Identification Engine (*SPIE, aka Hash-based IP Traceback*)

Storage of compressed data in specialized devices
— DGA generate digests of data (*Data Generation Agent*)
— SCAR for storage and retrieval (*SPIE Collection & Reduction Agents*)
— STM for central management (*SPIE Traceback Manager*)

[Snoeren et al.: „Single-Packet IP-Traceback“]
Source Path Identification (2)

„Store all information on traversed packets?“

No! What do we need to store?

Store digests of:

- Constant fields in IP Header (16 bytes)
- First 8 bytes of payload

Still a lot, compress:

Hashed in

*Bloom Filters*

<table>
<thead>
<tr>
<th>Version</th>
<th>IHL</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identification</td>
<td>Flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>Time to Live</td>
<td>Protocol</td>
<td>Header Checksum</td>
<td></td>
</tr>
<tr>
<td>Source Address</td>
<td>Destination Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (if any)</td>
<td>Payload</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Source Path Identification: **Bloom Filters (1)**

24 bytes of each packet hashed with \( k \) hash functions \( h_i \)

Hash values stored in filter:
— To store \( h_i(P) \), write a 1 into position \( 2^{h_i(P)} \) in bloom filter

\[
\begin{align*}
BF(P_0) &= 2^{h_1(P_0)} \text{ or } 2^{h_2(P_0)} \text{ or } \ldots \text{ or } 2^{h_k(P_0)} \\
BF(P_n) &= BF(P_{n-1}) \text{ or } 2^{h_1(P_n)} \text{ or } 2^{h_2(P_n)} \text{ or } \ldots \text{ or } 2^{h_k(P_n)}
\end{align*}
\]
During normal operation DGAs maintain bloom filters, if bloom filter more than 70% “full” (70% of the bits are set to “1”), sent to SCAR

**Detection** if a specific packet was processed:
— Hash packet with k hash functions $h_i$
— If any of the corresponding bits in all stored bloom filters is 0: Packet has not been processed
— All bits of a bloom filter are 1: Packet most probably traversed the DGA

**Path retrieval:**
— Victim contacts STM with pattern “P” of attack packet
— STM distributes pattern “P” to SCARs
— SCARs perform k hashes $h_1(P) .. h_k(P)$ to test which DGA forwarded matching packet
Traceback Approaches

Trace attack path backwards through network

Attack Path Traceback
  — Input Debugging
  — Controlled Flooding
  — ICMP Traceback
  — Probabilistic Packet Marking („IP-Traceback“)
Input Debugging

During attack:
— Trace attack-path „by hand“
— Contact administrator / ISP
— Admin matches ingress port for a given packet pattern of egress port
— Repeat until source is found...

Disadvantages:
— Cumbersome (what if admin X is not available?)
— Slow
— Expensive (manual intervention)
— Not scalable

...Yet the most applied method until today...
Controlled Flooding

During Single Source DoS-Attacks, traversed backbone links on the attack path are (heavily) loaded

Traceback attack path by testing links:
— Measure incoming attack traffic
— From victim to approximate source:
  • Create load on suspect links in the backbone
  • Measure difference in incoming attack traffic: if less attack packets arrive, the link is on the attack path...

Need possibility to create load on links to test with access on end-hosts around the backbone (chargen-service on multiple foreign end-hosts)

-disabled

DoS of the backbone in itself

Testing high speed backbone links using end-hosts difficult (how many dsl-links do you need to saturate one CISCO-12000-Link (10Gbps)?

[Burch & Cheswick: „Tracing Anonymous Packets to Their Approximate Source“]
ICMP Traceback

Assumption:
— DoS attacks are composed of packet floods
— Traceback on probabilistic sample of traffic possible

Approach:
— Routers give destination information about path of packets
— For 1 in 20k IP packets routers send additional ICMP iTrace to destination

Information in the iTrace-Packet:
— TTL → 255 (number of hops between router and destination)
— Timestamp
— Address of router
— Ingress (previous hop) and Egress ports (next hop on path)
— Copy of payload of traced packet (for identification)

[Bellovin: „ICMP Traceback Messages“]
ICMP Traceback: Open Issues

Signaling out of band → additional traffic (even at low rate)

Large amount of packets needed to reconstruct the full attack path (Tradeoff: Amount of ICMP packets vs. speed of path detection)

Victim needs to analyze large amount of iTrace messages

Firewalls (often) drop ICMP messages

Evasion: Possibility to create fake iTrace messages (easy to evade) (Potential solution: set up a PKI and let each router sign iTrace messages...)
Probabilistic Packet Marking (aka „IP Traceback“, PPM)

Approach similar to ICMP Traceback:

- Mark forwarded packets with a very low probability
- In-band signaling to avoid additional bandwidth needs (mark packets directly)

Different marking methods possible
Different signaling (encoding) methods possible

[Savage et al.: „Network Support for IP Traceback“]
PPM Marking: Node Append

Similar to IP Record Route: append each node’s address to IP packet

→ Complete attack path in every received packet

**Marking Procedure at router R:**
For each packet w, append R to w

**Path Reconstruction Procedure at victim v:**
for any packet w from attacker
extract path (R1,...,Rj) from the suffix of w

Pros and Cons:
— Converges quickly, easy to implement
— High bandwidth overhead (especially for small packets)
— Possible additional fragmentation of IP packets
PPM Marking: Node Sampling (1)

Similar to ICMP Traceback, but use **additional IP header field**

Marking Procedure at router R:  
For each packet w, with probability p write R into w.node

Path Reconstruction Procedure at victim v with additional node table NodeTbl (node, count):  
For each packet w from attacker, z \( \leftarrow \) w.node  
if z in NodeTbl  
increment z.count  
else  
insert (z,1) in NodeTbl  
sort NodeTbl by count  
extract path (R1,..,Rj) from ordered fields in NodeTbl

Routers close to victim have higher probability of marking: the higher the count in NodeTbl the closer the router
PPM Marking: Node Sampling (2)

Issues of node sampling:
Additional IP header field needed

Routers far away from victim contribute only few samples (marks are overwritten) and large number of packets needed to recover complete path

\[(p=0.51, d=15: > 42k \text{ packets needed to completely reconstruct attack path})\]

In DDoS with multiple attackers different paths can not easily be distinguished
PPM Marking: Edge Sampling, Marking

Mark packets with:
Backbone edge $e(u, w)$ (start router $u$, end router $w$) and distance $d(u, v)$
Victim $v$ can deduce graph of edges $e$ and reconstruct attack tree

*Marking Procedure at router $R$:*  
For each packet $w$, with probability $p$
  write $R$ into $w$.start and 0 into $w$.distance
else // probability $1-p$
  if $w$.distance = 0 then
    write $R$ into $w$.end
  increment $w$.distance
In order to reconstruct the attack tree

**Path Reconstruction Procedure at victim v with additional attack tree t:**

for each packet w from attacker

if w.distance = 0 then
    insert edge (w.start, v, 0) into t
else
    insert edge (w.start, w.end, w.distance) into t

remove all edges (x,y,d) with d ≠ d(x,v) in t

extract path (R_1,..,R_j) enumerating acyclic paths in t
PPM Encoding

With IP routers using IP addresses, marking of w.start, w.end, w.distance needs 32 + 32 + x bits.

Solution: coding edge as IP(w.start) XOR IP(w.end)

(last hop known (w.distance = 0), others determined through XOR at victim)

→ 32 bit ("edge-id") + x bits (distance)

Transmit only fragment of edge-ids with every packet and mark with higher probability (together with hashed values of the router's edge IP address to distinguish edges → 64 bit per edge)

— Edge-ID fragment 8 bits, offset 3 bits, distance 5 bits → 16 bits

\[ d(a,v) = 2 \]
PPM Encoding: Encapsulation in IP header

Using the „Identification“ field for in-band signaling (16 bit)

<table>
<thead>
<tr>
<th>Version</th>
<th>IHL</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>Flags</td>
<td>Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>Time to Live</td>
<td>Protocol</td>
<td>Header Checksum</td>
<td></td>
</tr>
<tr>
<td>Source Address</td>
<td>Destination Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (if any)</td>
<td>Payload</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

But the ID-Field is needed!? In case of fragmentation:
— Downstream marking: send ICMP Echo Reply („packet lost“)
— Upstream marking: set „don’t fragment“ flag
PPM Advantages and Disadvantages

😊 Stable
😊 Meaningful results under partial deployment
😊 No bandwidth overhead
😊 Low processing overhead

😊 Works mainly for bandwidth exhaustion attacks
 — Many packets needed for reconstructing attack path
 — Fragmented packets can not be traced (e.g. Teardrop attack, however, Teardrop is not bandwidth exhaustion anyway)

😊 Victim under attack needs rather high amount of memory (many packets!) and processing time
😊 In order to avoid spoofing, authentication needed (PKI, signatures)
Related Techniques for Mitigation / Avoidance

Hop-Count Filtering
Aggregate Based Congestion Control (ACC)
Secure Overlay Services
Aggregate Based Congestion Control

Is it possible, to restrain attack traffic in the backbone?
— Traffic is very diverse in the backbone, in general
— However, attack traffic forms an aggregate of similar traffic
  (Identified by analyzing the dropped traffic:
  select the destination addresses with more than twice the mean number of drops and
  cluster these destination addresses to 24bit prefixes)

ACC/pushback is a reactive approach:
— If router/link is congested, can an aggregate be identified?
— If there is an aggregate, limit the rate of aggregate traffic
— If the aggregate persists, perform „pushback“: inform upstream routers to limit rate of the aggregate

[Mahajan, Bellovin & Floyd: „Controlling High Bandwidth Aggregates in the Network“]
Recapitulation: Source Identification of IP Traffic

Problem: nodes may lie about their IP address
Spoofing enables attackers to perform DoS/DDoS attacks
If the source of an attack can be identified, attack traffic can be restrained

Different approaches to identify attacker / routers / ISP on attack path:
— Logging in the network
  • „Aggregated network logging“
  • Source Path Isolation („Hash-based IP Traceback“)
    — Traceback of packet flow
  • Controlled Flooding
  • ICMP Traceback
  • Probabilistic Packet Marking („IP Traceback“)
    — Other Means (Mitigation/Avoidance of attacks)
Conclusion

Increasing dependence of modern information society on availability of communication services

While some DoS attacking techniques can be encountered with “standard” methods, some can not:
— Hacking, exploiting implementation weaknesses, etc. may be encountered with firewalls, testing, monitoring etc.
— Malicious protocol deviation & resource depletion is harder to defend against

Designing DoS-resistant protocols emerges as a crucial task for network engineering:
— Network protocol functions and architecture will have to be (re-)designed with the general risk of DoS in mind
— Base techniques: stateless protocol design, cryptographic measures like authentication, cookies, client puzzles, etc.