Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics

Mark Handley, Vern Paxson, and
Christian Kreibich

USENIX Security ‘02

Contributions

- Review of insertion, evasion, DOS attacks against NIDS (Network Intrusion Detection Systems).
- Proposal for “traffic normalization” system that will prevent many of these attacks.
- Discussion of alternative approaches to normalization which might help.
- Evaluation of the tradeoffs of normalization.
- Systematic identification of normalizations.
- Performance information regarding the normalizer.

Evasion Techniques

- Network Intrusion Detection Systems (NIDS) are vulnerable to attacks which exploit their inability to understand how end systems and internal routers handle packets which may be sent into the network.
- Three categories of evasion techniques:
  - Incomplete analysis of protocols.
  - Inability to correctly resolve ambiguities in the same way end systems do.
  - Incomplete knowledge of network topology.

Evasion Techniques: Incomplete NIDS Analysis

- NIDS may incompletely analyze protocol behavior.
  - Fragmented IP packets may slip by NIDS when NIDS do not reassemble fragments.
  - Incomplete if reassembly is implemented in the NIDS, but not for out of order fragments.
- Ptacek & Newsham (1998): Four tested commercial systems failed to correctly reassemble fragments.

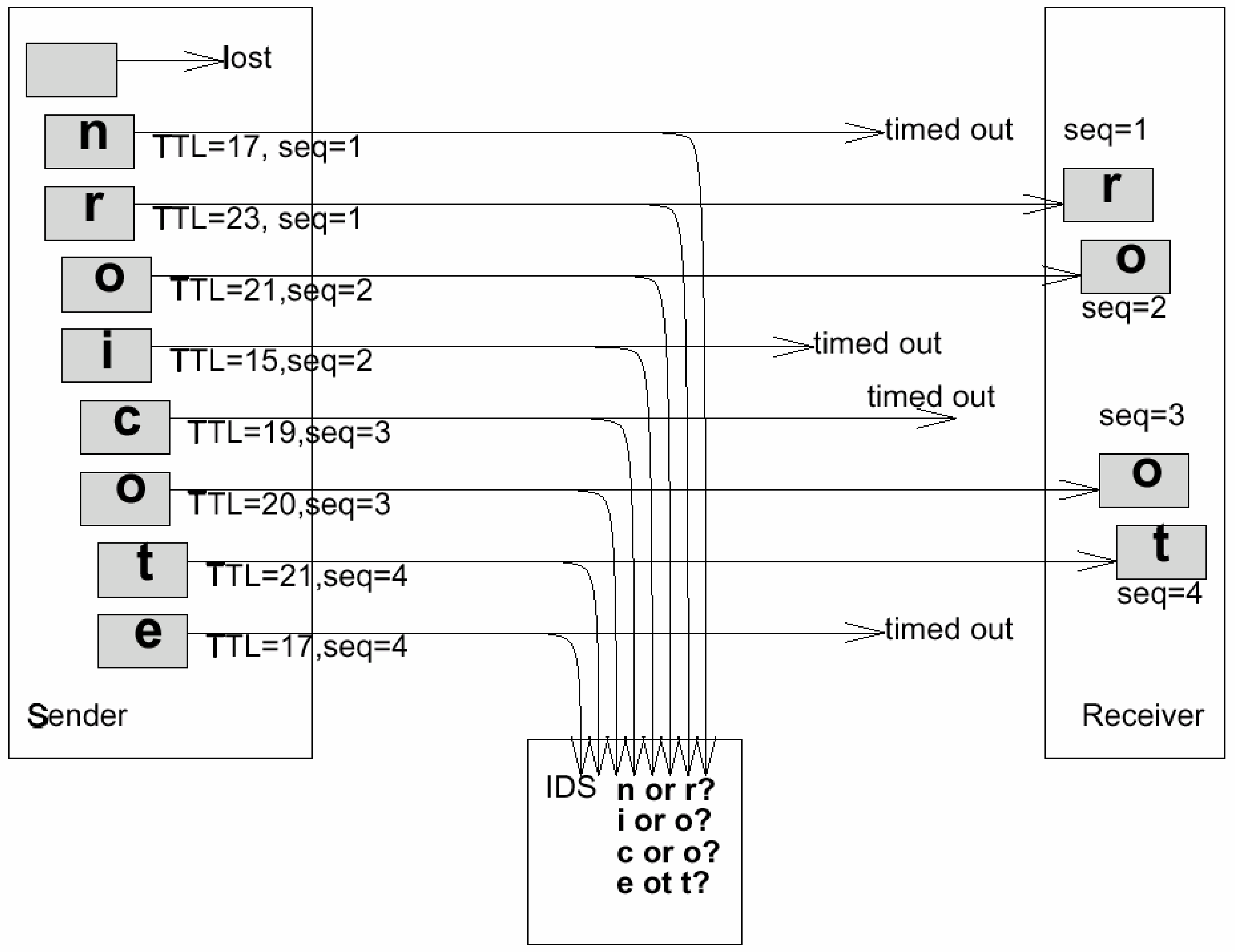
Evasion Techniques: Inability to Predict End System Behavior

- Different end systems handle same input differently. Different operating systems and applications do different things in response to rare events / gray areas in protocols.
  - Overlapping fragments with different data: which fragment is used?
  - Backspace vs. Delete during authentication dialog.

Evasion Techniques: Incomplete Knowledge of Network Topology

- Without explicit knowledge of the internal network topology, NIDS may make assumptions about protocol behavior that are incorrect.
  – TTL setting on incoming packets from attacker causes some packets to be dropped after being inspected by the NIDS.
  – End system only sees packets with longer TTL.
  – NIDS misses attack.

Evasion Example: Prediction and Network Topology



Proposed Solution: Normalizer

- New network forwarding element (“bump in the wire”).
- Before NIDS in network stream.
- Not firewall.
- Similar to “protocol scrubber.”
- Goal is to ensure that the NIDS and the end systems see the same things.
- Alter, insert, drop packets into streams.

Alternative Solutions

- Use host based IDS.
  - Difficult (costly) to deploy and maintain.
- Make intranet information explicit.
  - Difficult to maintain accurately (topology).
  - Too much information to obtain and store (end system protocol handling).
- Bifurcating analysis.
  - NIDS system keeps track of all possibilities in the case of ambiguities: possibly exponential growth in storage and processing requirement.

Normalization Tradeoffs

- Extent: normalization vs. protection.
  - Normalizer can either just normalize, or also do additional protection like firewalling. Paper focuses on just normalization.
- Respecting end-to-end semantics.
  - Simple in the case of protocol violations.
  - Harder when normalization may affect protocol performance: traceroute relies on TTL field .
- Impact on end-to-end performance.
  - Normalization may preserve semantics but affect speed.

Normalization Tradeoffs

- Stateholding.
  - Normalizer must keep some state information to determine how to normalize. This may result in attacks against the normalizer.
- Inbound vs. outbound traffic.
  - Asymetry due to location: before NIDS. Two normalizers?
- Protection vs. offloading traffic.
  - Normalizer may be able to reduce load on the NIDS: i.e. compute checksums and reject.

“Real-world Considerations”

- Cold start:
  - Normalization makes use of state information. But when the normalizer starts up, existing connections are not stored in state tables.
- Attacking the Normalizer:
  - Stateholding attacks: i.e. send many fragments, but never enough to make a complete packet.
  - CPU attacks: reduce speed of normalizer.

Handling stateholding attacks

- Examples:
  - Many overlapping fragments, but never enough to complete a packet.
  - Simple SYN flooding.
  - ACK flooding – normalizer may keep state to handle cold starts.
  - Initial window flooding: after SYN-ACK, attacker floods data before response. Normalizer retains data to prevent inconsistent retransmissions.

Handling stateholding attacks

- Response:
  - If normalizer determines that it is under attack (excessive memory usage), ramp up protection, possibly degrading performance.
  - Instantiate state in response to internal machines' actions rather than just external machines.
  - Focus on preventing DOS against NIDS and normalizer, not internal machines (presumably NIDS is responsible for this).

Normalizing: Examples (IP)

Header length:

- Problem:
  - Packet with invalid header length field may be dropped or accepted by end system; NIDS doesn’t know.
- Solution:
  - Too small or larger than packet: discard packet.
- Effect on semantics:
  - Ill-formed packet already violates protocol semantics.

Normalizing: Examples (IP)

Total packet length:

- Problem:
  - Packet with invalid total length field may be dropped or accepted by end system; NIDS doesn’t know.
- Solution:
  - Discard if field longer than actual. Trim if packet longer than field.
- Effect on semantics:
  - Ill-formed packet already violates protocol semantics.

Normalizing: Examples (IP)

Don’t Fragment (DF) flag:

- Problem:
  - If DF set, and MTU outside bigger than MTU inside, attacker can cause packets to get dropped before reaching target. ICMP indications of drops may not reach normalizer.
- Solution:
  - Clear DF.
- Effect on semantics:
  - Breaks “Path MTU Discovery”.

Normalizing: Examples (IP)

More Fragments (MF) flag, Fragment offset:

- Problem:
  - Overlapping fragments with differing contents. End systems may choose content in unexpected ways.
- Solution:
  - Normalizer performs reassembly.
- Effect on semantics:
  - Allowable for routers to perform reassembly. May result in stateholding attack.

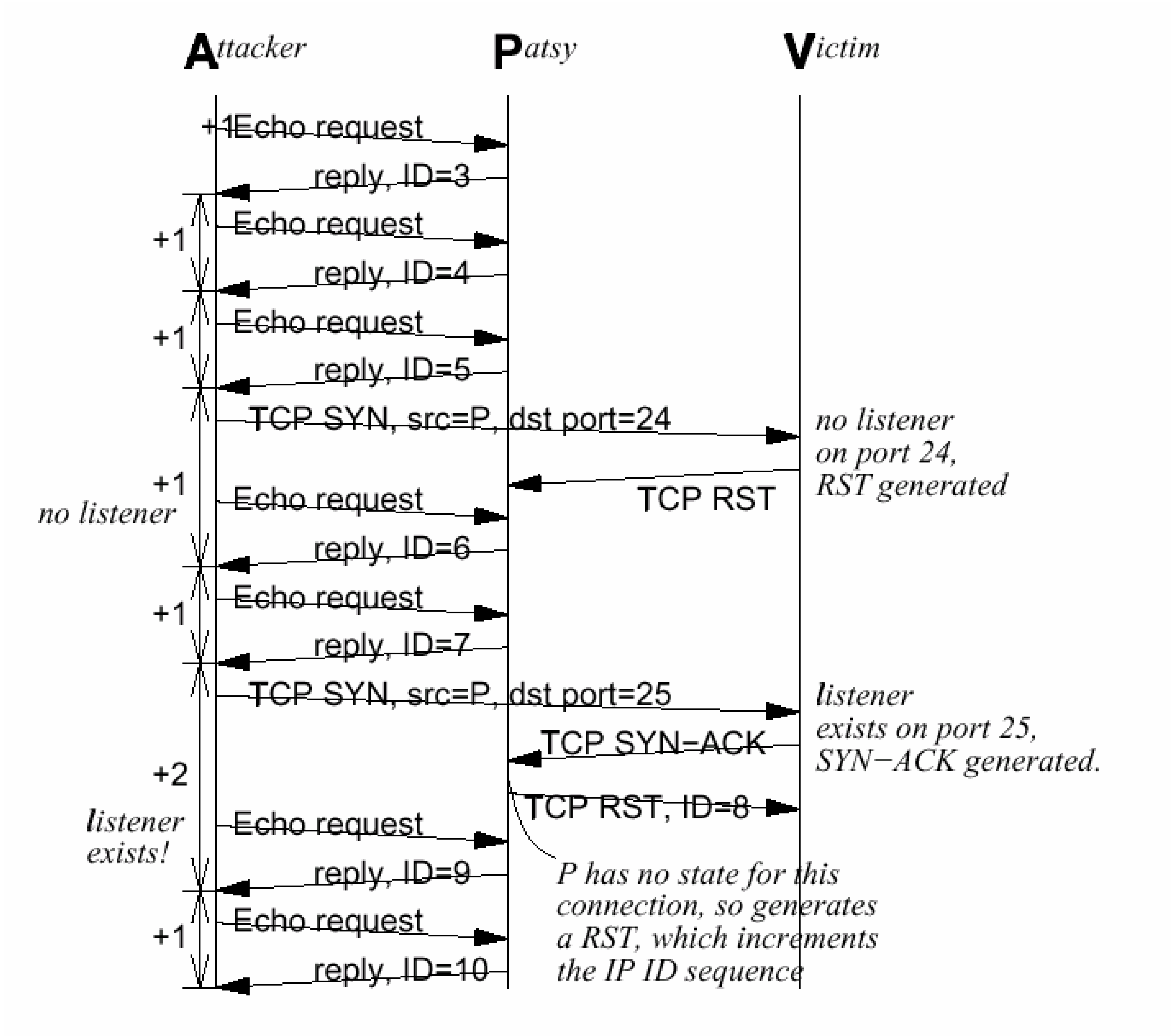
Normalizing: Examples (IP)

TTL (Time-to-live):

- Problem:
  - Discussed above. Attacker may exploit knowledge of network topology to cause some packets to be dropped.
- Solution:
  - Several solutions proposed. Preferred: set all TTL fields to be larger than the longest path inside network.
- Effect on semantics:
  - Routing loops: ouch!
  - Breaks traceroute.
  - Multicast protocols may use expanding ring searches – all internal hosts appear to be at the same depth.

IP Identifier and Stealth Port Scans

- Attacker connects to inside machine (patsy), regularly sends packets and observes the IP identifier (ID), which is incremented regularly on each packet it sends.
- Attacker spoofs packets so that they appear to come from inside machine, sends packets to victim.
- Victim responds to spoofed packets, sends responses to patsy.
- Attacker can determine listening ports on victim through analyzing the timing of missed increments.


Attacker
Patsy
Victim
+1
Echo request
reply, ID=3
Echo request
+1
reply, ID=4
Echo request
+1
reply, ID=5
TCP SYN, src=P, dst port=24
no listener on port 24, RST generated
+1
no listener
TCP RST
Echo request
reply, ID=6
Echo request
+1
reply, ID=7
TCP SYN, src=P, dst port=25
listener exists on port 25, SYN−ACK generated.
TCP SYN−ACK
+2
TCP RST, ID=8
listener exists!
Echo request
reply, ID=9
P has no state for this connection, so generates a RST, which increments the IP ID sequence
Echo request
+1
reply, ID=10

IP Identifier and Stealth Port Scans

- Actually, patsy is also a victim here: appears to be doing port scanning, normally unacceptable.
- Normalizer solution for patsies:
  - Use cryptography to scramble the IP IDs of incoming and outgoing packets. Normalizer must understand how to perform unscrambling in places the ID fields are used, such as ICMP packets.
- Normalizer solution for victims: “Reliable RST.”
  - Normalizer sends keep-alive packet after every RST packet it sends: attacker gets double increment on every probe.

Normalizing: Examples (TCP)

Reliable RST:

- Problem:
  - In TCP, FIN is delivered reliably, but RST is not. If a RST sent by an attacker is dropped due to congestion, etc., but seen by the NIDS then the connection can persist and the attack could evade detection.
- Solution:
  - Reliable RST. After sending RST, send ACK. TCP specifies that response to this ACK will indicate whether the RST was received.
- Preserves protocol semantics.

TCP cold start: avoiding stateholding attacks

- If packet on unknown connection seen after cold start is from trusted network, instantiate state.
- If packet is from untrusted network, transform into ACK packet (remove payload and decrement sequence number) and forward to internal network. Internal network will respond according to protocol, if it thinks the connection exists, resulting in instantiated state.

TCP cold start: window scaling problem

- TCP contains 16 bit window field. Window scaling option specifies left shift be applied to the field, in order to specify larger window sizes.
- Normalizer tracks window size to determine if packets will be accepted at receiver.
- After cold start, state info regarding window scaling is lost; normalizer has no way of knowing current window scale state.
- Normalizer must prevent negotiation of window scale options or have persistent state.

Incompleteness of normalization

- TCP “urgent” pointer sends information to application layer.
- Normalizer must analyze protocol specific information to determine whether or not socket options specified in application will accept urgent pointer packets or not.

Implementation and performance

- Implemented in FreeBSD, Linux. “norm”
- Performance tested with a single trace (100,000 packets) obtained from the LBNL external interface.
- Implemented at user level, rather than in kernel.
- Validation performed by hand; analyzed using protocol analyzer GUI.
- Results after one pass are unchanged with second pass.
- Conclusion: normalizer is capable of handling bi-directional 100Mb/s traffic with commodity PC hardware.

Performance analysis

- Memory to memory copy simulates kernel implementation. Kernel impl. (“click” router) can forward 333,000 packets/s on PC.
- Three traces:
  - T1: Original packet capture.
    - 88% TCP, 10% UDP, 1.5% ICMP.
    - Mean packet size 491 bytes.
  - U1: T1, but TCP headers changed to UDP.
    - Useful for comparing cost of TCP normalizations to IP only.
  - U2: netcat generated trace of 100,000 92 byte UDP packets.
    - Useful for comparing per packet cost vs. per byte cost.

Copy, normalize rates, normalization counts

	Just copy		Normalize and copy	
Trace	pkts/s	bit rate	pkts/s	bit rate
T1	727,270	2856 Mb/s	101,000	397 Mb/s
U1	"	"	378,000	1484 Mb/s
U2	1,015,600	747 Mb/s	626,400	461 Mb/s

Trace	IP	TCP	UDP	ICMP	Total
T1	111,551	757	0	0	112,308

Performance analysis

rnd intv’l	input frags/s	frag’ed bit rate	output pkts/s	output bit rate	pkts in cache
100	299,670	86 Mb/s	9,989	39 Mb/s	70
500	245,640	70 Mb/s	8,188	32 Mb/s	133
1000	202,200	58 Mb/s	6,740	26 Mb/s	211
2000	144,870	41 Mb/s	4,829	19 Mb/s	335

Review of contributions

- Review of insertion, evasion, DOS attacks against NIDS (Network Intrusion Detection Systems).
- Proposal for “traffic normalization” system that will prevent many of these attacks.
- Discussion of alternative approaches to normalization which might help.
- Evaluation of the tradeoffs of normalization.
- Systematic identification of normalizations.
- Performance information regarding the normalizer.