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

Authors: Mark Handley, Vern Paxson, Christian Kreibich

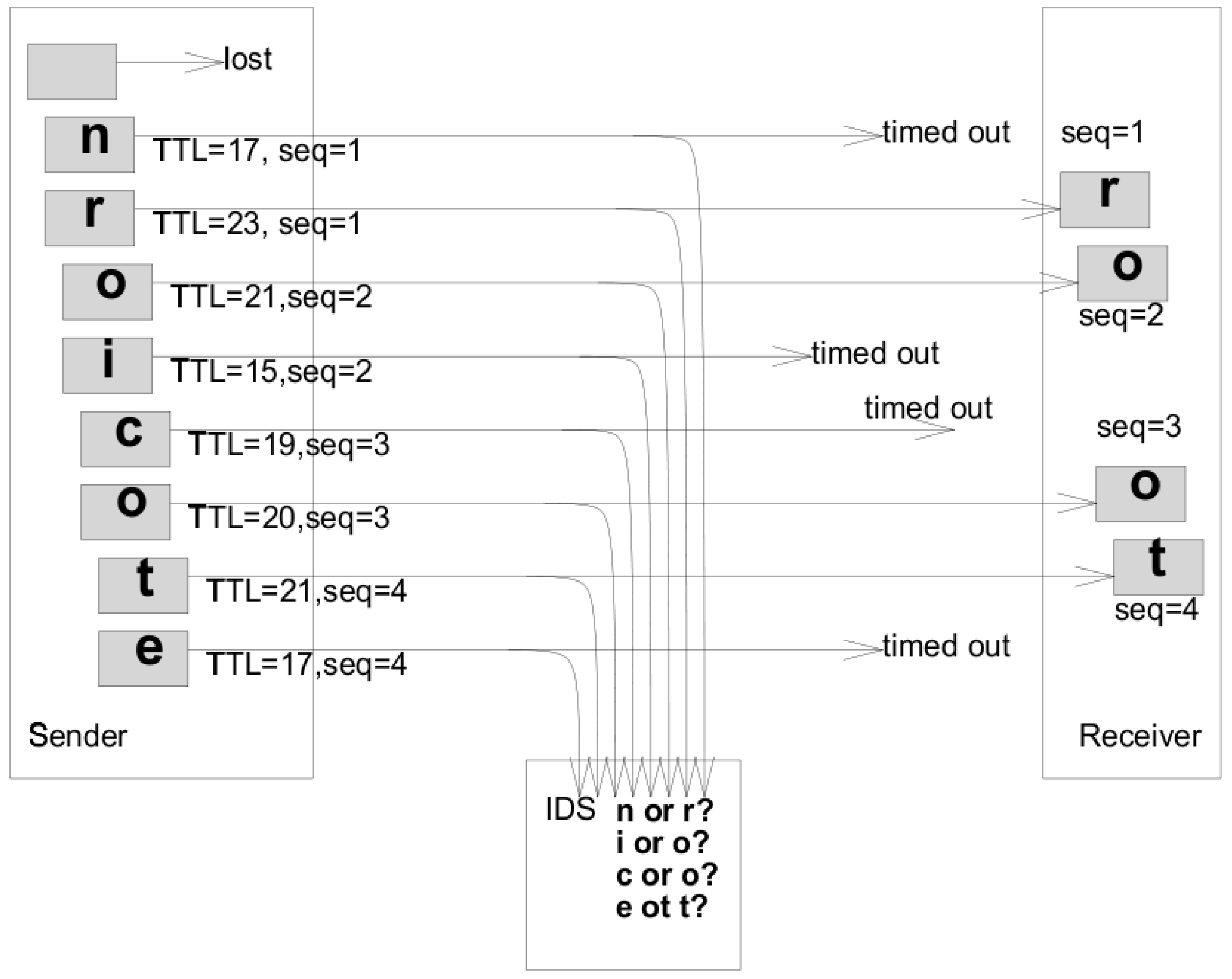
Exploitable Ambiguities

- NIDS does not have full range of behavior allowed by a protocol.
  - Fragmenting IP packets can evade systems if reassembly by monitors is incorrect or incomplete.
- NIDS can't determine unspecified behavior.
  - For IP fragments that differ in the overlapped region, an end-system may either take the overlapped region of the lower or upper fragment.
- NIDS usually doesn't have a detailed knowledge of network topology.
  - A packet with a low TTL field may not reach the end-system before expiring.

Exploitable Ambiguities

- The first problem is not an issue with a good NIDS implementation with complete protocol analysis.
- For the other problems, end-system behavior can't really be determined without external knowledge.
- Attackers can use probing to determine end-system characteristics to construct such attacks.

Exploitable Ambiguities



Exploitable Ambiguities

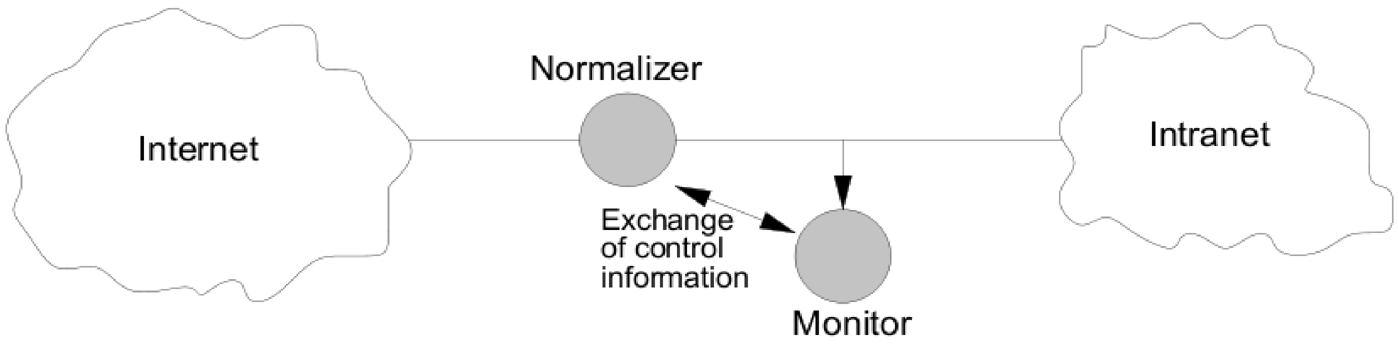
- Inconsistent "retransmissions"
  - Either n or r, o or i, etc.
  - TCP stacks may either accept the first or second packet, so there is no correct rule that can be used
- Manipulated TTL fields
  - Packets with low TTL will expire before reaching the end-system
  - NIDS will need to know the amount of hops between itself and the victim

Exploitable Ambiguities

- Such behavior could be considered anomalous traffic, but detecting that both
  - Degrades NIDS precision from knowing the specifics of an attack to knowing that an attack might be going on
  - Will likely result in false positives due to legitimatally unusual traffic
- All issues need to be addressed or NIDS will eventually become completely circumventable

Traffic Normalizer

- A network forwarding element
- Normalizes packets to remove potential ambiguities, so NIDS won't have to consider those
  - Replace later inconsistent retransmissions with data from original transmission, which results in 'noct'
- It is not a firewall, as its purpose is to make everything unambigious to the NIDS.



Other Approaches

- Host-based IDS: IDS on all end-system hosts, eliminates all ambiguities as it has access to protocol state above IP/TCP
  - This is not practical due to deployment
- Knowing intranet details: with knowledge of end-system implementation and network topology, amibiguities can be resolved
  - Constructing such a database may not be practical
- Bifurication analysis: splits analysis context in case of ambiguity
  - Easy in case of delete vs backspace, only two possible interpretations
  - 'Root' vs 'nice' becomes 32 possibilities, leading to exponential explosions when there are a large amount of packets

Normalization Tradeoffs

- Increasing degree of normalization and protection leads to needing to hold more state, a decrease in performance, and impacts end-to-end semantics
- Extent of normalization vs protection
  - A firewall can also be used on the same box as the normalizer to prevent known attacks
- Impact on end-to-end semantics
  - Dropping packets that can't be correct (e.g. IP fragments with same offset but different data) won't hurt operation of connection because it's already incorrect
  - A low TTL may cause ambiguities, but increasing it may cause adverse effects
  - Erodes protocol semantics, so it should be determined on an on-site basis

Normalization Tradeoffs

- Impact on end-to-end performance
    - Effects protocol performance, but this depends on protocol, network, etc.
- Amount of state held
    - State needs to be held to understand incoming data-- this can be attacked (stateholding attack) by exhausting the NIDS' amount of states and then launching an attack undetected because NIDS "fails open"
    - Normalizer also needs to hold state, but it can "fail closed" or perform "triage"-- drop states that don't appear to be progressing, making it difficult for attackers to fake active connections, but this could be utilized by an attacker to perform a DoS

Normalization Tradeoffs

- Work offloaded from NIDS
  - Moving some work from NIDS to normalizer-- for instance, checking for bad checksums could be done by normalizer so NIDS doesn't have to use cycles verifying checksums

Real-world Considerations

- Cold Start
  - After starting up, the normalizer is confronted with already-established connections that it has no information on
  - Attackers could either force a cold start by crashing the monitor, or keep a connection alive for a long time which increases the probability of spanning a cold start

Real-world Considerations

- Attacking the normalizer
- Memory usage, flooding, etc.
  - Memory could be used up by forcing normalizers to hold many fragmented packets, but this is defeated by bounding the amount of memory used
- Floods are difficult to defend
  - SYN flooding to same or multiple hosts
  - ACK flooding, as a cold-start normalizer may be designed to instantiate state for each unknown connection
  - Initial window flooding, which floods data after SYN/SYN-ACK to force normalizer to store data without ACKs

Real-world Considerations

- In order to defend against being under attack, operate under assumption of either being under attack or not by monitoring memory
  - Normal behavior when not attacked, conservative strategy when attacked
- Conservative strategy: only instantiate state when traffic is seen from inside host when under attack
- CPU overload attacks
  - Forces the normalizer to forward packets at a slower rate, combined with stateholding attacks to force normalizer to perform expensive searches
- Search algorithms need to be implemented carefully to reduce this problem

A Systematic Approach

- Since it's not possible to know application state at end-systems, the normalizer is focused on IP/ICMP and TCP/UDP
- Since there are many potential protocol ambiguities, walking through packet headers to consider all of those
- Many normalizations are extremely unlikely evasion scenarios, but this is good design

Normalizing IP Headers

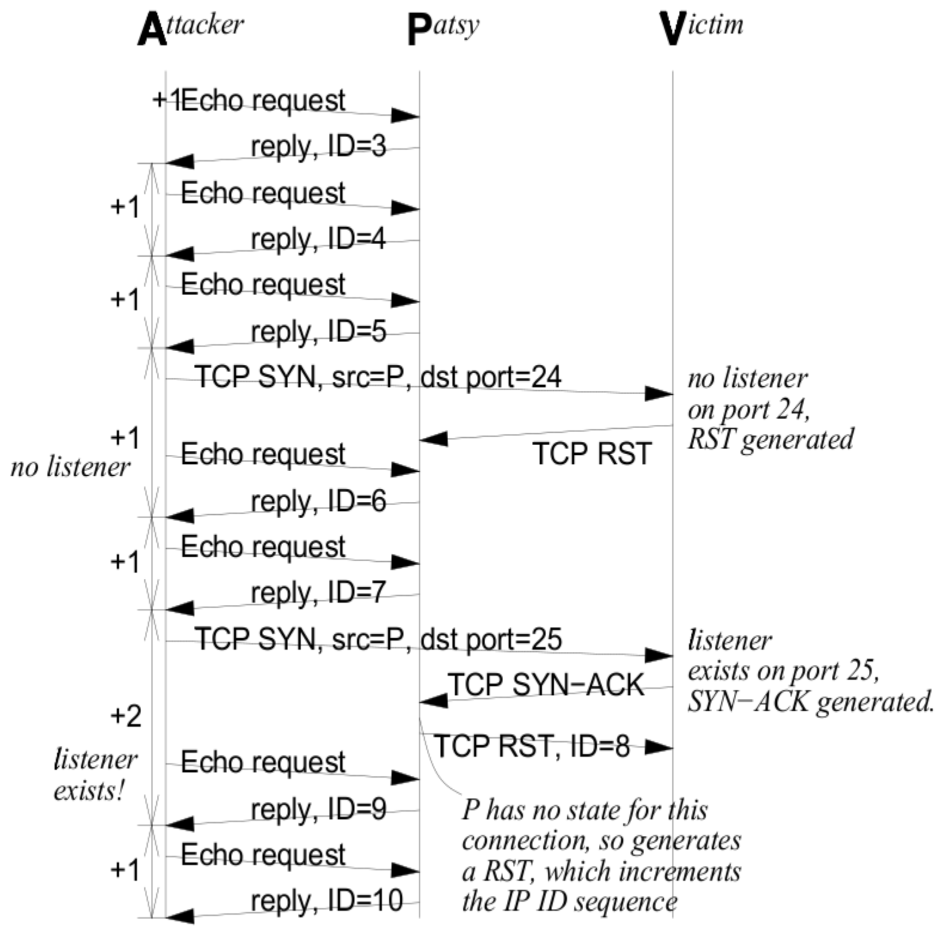
- Header length: accepting an incorrect header length is implementation-specific, so drop packets <20 bytes and header lengths longer than packet length
- TOS/Diffserv/ECN: clear bits if site does not use such mechanisms
- Total length: discard or trim packets
- IP Identifier: Covered later
- Must Be Zero: clear bit to zero (for potential future uses)
- Don't Fragment: clear (or discard for non-zero fragmentation offset)
- More Fragments/Fragment Offset: reassemble incoming fragments, re-fragment them afterwards if necessary

Normalizing IP Headers

- TTL:
  - Measure hops to every end-host, ignore packets with insufficient hops--but internal routing may change
  - Use ICMP time-exceeded-in-transit packets, but may not be able to tell which were discarded
  - Re-set TTL to larger than longest path if necessary, but routing loops may rapdily consume bandwidth
- IP header checksum: discard invalid checksums
- Source address: Discard localhost, 0/0, broadcast, etc.
- Destination address: Drop invalid destination
- IP options: Remove options, but may impact connectivity
- Padding: Zero the padding bytes

IP Identifier and Stealth Port Scans

- Using pasties, attackers can determine which ports are open without revealing themselves
- Solution: scramble (reversible) IP IDs to prevent internal hosts from being pasties, but this might break diagnostics protocols
- "Reliable RST" for victims: Forces IP ID increase, generates extra traffic (covered later)



TCP Normalizations

- “Reliable RST”-- RSTs are not reliable, unlike SYN and FIN
  - Assume RST does terminate the connection: attacker could keep sending traffic and the monitor doesn't have the appropriate state to interpret traffic
  - Assume RST doesn't terminate: may have to hold state information indefinitely
- Sending an keep-alive packet results in
  - RST accepted, so a RST is sent back
  - RST was dropped or ignored, so an ACK is sent back
  - Keep-alive was dropped
  - Response to keep-alive was lost

Reliable RST

- If a RST is received, tear down state (rule 1)
- Otherwise, keep state (rules 2-4), rarely does this happen unnecessarily

TCP Cold Start

- Assuming the normalizer is between a trusted and untrusted network:
  - Initialize state if the packet comes from the trusted network
  - If the packet comes from the untrusted network, transform it into a keep-alive and forward it
- If the connection is legitimate, the state is instantiated after receving an ACK from the trusted network
- If it's not, either an RST or nothing will be sent, so there's no need to keep state
- Window scaling option-- negotiated during connection establishment, so there's no way of telling if the option was used or not-- normalization requires stripping the window-scale option from all SYN packets, which can be costly

Incompleteness of Normalization

- Not everything can be normalized
- Urgent pointer in TCP-- accepting or rejecting them are based on socket options
  - "robot" with urgent pointing at "b" may either be "robot" or "root"
- Bifurication analysis could be used by assuming relevant option stays enabled or disabled for entire connection
- This shows that not all problems can be solved with normalizers

Implementation

- Trace-driven approach: input packet trace, inspect output trace after transformation
- Generated traffic similar to an attacker using nmap/fragrouter, also developed trace editor to modify headers
- Evaluation process: edit existing trace -> feed to norm -> examine trace for correctness -> feed to norm again

Performance

- Emulates kernel performance by reading entire trace file into memory
- Bit rates with all checks enabled were 400+ Mb/s for trace files
- Attacks were tested by fragmenting packets in trace, found that triage needs to be done in the worst case
- Inconsistent TCP retransmissions also slows down normalizer