

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

- Paper by:
 - Mark Handley and Vern Paxson (ACIRI)
 - Christian Kreibich (TUM)
- Published in:
 - USENIX 2001
- Review by:
 - Todd Sproull

Introduction

- Problem
 - Fundamental problem with network intrusion detection systems (NIDSs)
 - Skilled hacker has the ability to evade detection of NID
 - Exploits due to ambiguities in the traffic stream
- Solution
 - Develop a traffic normalizer
 - Sits in front of traffic stream to fix potential ambiguities before traffic is passed to a NIDS

Problems in NIDS

- NIDS may not support full range of behavior allowed by a given protocol
 - Attacker can evade NIDS that fails to reassemble IP fragments
 - Attacker sends only IP fragments
 - Since end-systems perform reassembly, attacker has same effect as sending an attack without fragmentation

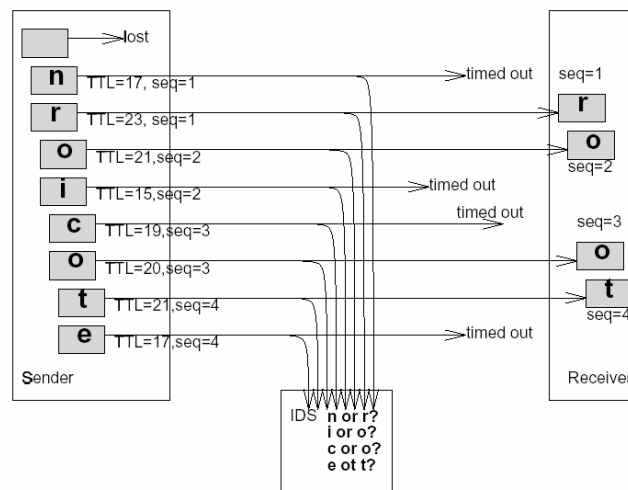
Problems in NIDS

- NIDS may be unable to determine how a end-system will treat a given sequence of packets
 - Different implementations interpret it in different ways
 - End-systems may receive overlapping IP fragments
 - Some systems accept data first received, others take data received last

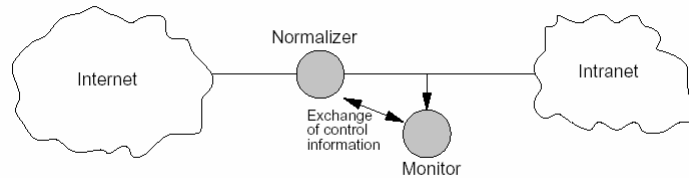
Problems in NIDS

- NIDS may be unable to determine if a packet will ever be seen by an end-system
 - Hackers manipulate the TTL field
 - NIDS assumes all traffic will make it to end-system
 - By varying the TTL field, able to exploit systems with NIDS not detecting the attacking

Example Exploit



Normalizer



- Sits in front of the NIDS
- Acts as a “bump in the wire”
- Removes ambiguities from NIDS

Other approaches

- Use of a host-based IDS
 - Major deployment issues
 - IDS not NIDS
 - Not a network solution, not cheap
- Understand details of intranet
 - Ambiguity can be eliminated if end-systems are known
 - Difficult to construct database of all end-systems

Normalization Tradeoffs

- Design of a traffic normalizer has several tradeoffs
 - Extent of normalization vs. protection
 - Impact on end-to-end semantics
 - Impact on end-to-end performance
 - Amount of state held
 - Work offloaded from the NIDS

Normalization Tradeoffs

- Extent of normalization vs. protection
 - Normalizer generally means “bump on a wire”
 - However, can perform some firewall functionality
 - Prevent known attacks
 - Shut down access to internal machines from external attack

Normalization Tradeoffs

- End-to-end Semantics
 - Packets that are incorrect according to protocol specifications should be fixed (dropped)
 - Packets that are perfectly legal may still be ambiguous to the NIDS
 - Packet with a low TTL may confuse the NIDS
 - Possible solution – Increase TTL (be careful of looping packet!)
- Stateholding
 - NIDS system needs to hold state to understand context of information
 - State is also used to correct ambiguity in data flows
 - NIDS is susceptible to a stateholding attack

Normalization Tradeoffs

- Inbound vs. outbound traffic
 - Normalizer typically processes packets from the Internet to an intranet
 - In the case of processing traffic in both directions, 2 normalizers could be used
- Protection vs. Offloading work
 - Normalizer can also serve as a machine to offload work from the NIDS
 - Normalizer can discard packets with bad checksums

Real world considerations

- Cold start
 - When normalizer starts it finds all-ready established traffic
 - Refrain from establishing state for already active connections
 - A flood of bogus traffic could generate a large amount of state
- Attacking the Normalizer
 - Stateholding attacks
 - CPU overload attacks

Attacks on the normalizer – stateholding

- Some normalizations require state
 - Sending IP fragments requires state within the normalizer
 - Attacker can use up the memory of normalizer by sending a large number of IP fragments without sending a complete packet
 - Attack is defended by restricting the amount of memory allocated to fragments

Attacks on the normalizer CPU overload

- Attacker may overload CPU with large amounts of normalization
 - CPU attacks usually slow the rate of packets being forwarded but do not necessarily cause packets to be dropped
 - Attacker tries to use computational complexity to their advantage
 - Example: An expensive search operation
 - Choice of search algorithm is important

Packets normalized

- NIDS cannot know all possible applications processed by an end-system
- Normalizer focuses on major network protocols
 - IP, ICMP, TCP, UDP
- Paper outlines IP packet header normalizations
- Appendix describes the rest

Examples of normalization

Type Of Service/Diffserv/ECN. These bits have recently been reassigned to differentiated services [11] and explicit congestion notification [15].

Issue: The Diffserv bits might potentially be used to deterministically drop a subset of packets at an internal Diffserv-enabled router, for example by sending bursts of packets that violate the conditioning required by their Diffserv class.

Solution: If the site does not actually use Diffserv mechanisms for incoming traffic, clear the bits.

Effect on semantics: If Diffserv is not being used internally, the bits should be zero anyway, so zeroing them is safe. Otherwise, clearing them breaks use of Diffserv.

Total length. If the total length field does not match the actual total length of the packet as indicated by the link layer, then some end-systems may treat the packet as being one length, some may treat it as being the other, and some may discard the packet.

Solution: Discard packets whose length field exceeds their link-layer length. Trim packets with longer link-layer lengths down to just those bytes indicated by the length field.

Effect on semantics: None, only ill-formed packets are dropped.

Version. A normalizer should only pass packets with IP version fields which the NIDS understands.

Header length. It may be possible to send a packet with an incorrect header length field that arrives at an end-system and is accepted. However, other operating systems or internal routers may discard the packet. Thus the NIDS does not know if the packet will be processed or not.

Solution: If the header length field is less than 20 bytes, the header is incomplete, and the packet should be discarded. If the header length field exceeds the packet length, the packet should be discarded. (See *Total length* below for a discussion of exactly what constitutes the packet length.)

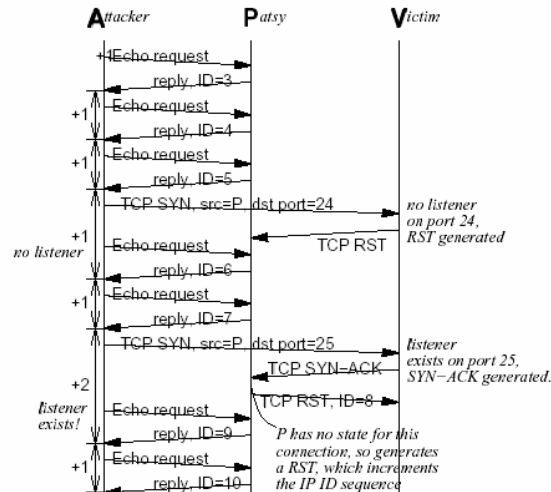
Effect on semantics: Packet is ill-formed—no adverse effect.

Note: If the header length is greater than 20 bytes, this indicates options are present. See IP option processing below.

IP Identifier Attack

- End-systems give away services running on their machines with a stealth port scan
 - Involves 3 parties: Attacker, Victim, Patsy
 - Attacker continuously pings Patsy
 - Attacker then issues a SYN to Victim with the source IP of the Patsy
 - Victim issues TCP RST to Patsy, if no service is running on that port, or a TCP SYN-ACK if service is listening for connections
 - Attacker can look at the ID field in IP to determine if SYN-ACK was issued or not

IP Identifier Attack - Example



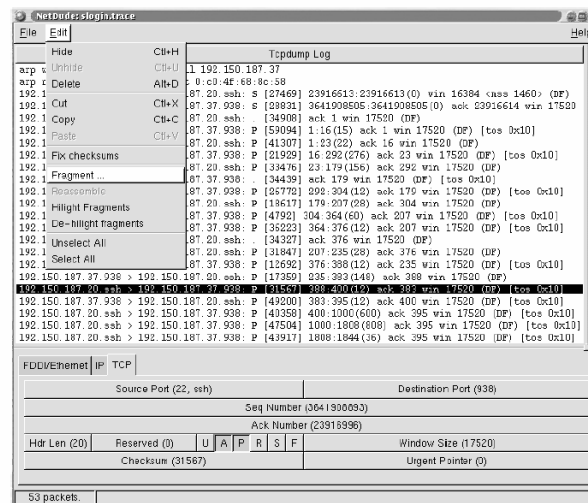
Solution to problem

- **Patsy**
 - Have the normalizer scramble the IP IDs of incoming and outgoing packets
 - Drawback: Diagnostic protocols reporting IP IDs may not be functional
- **Victim**
 - Use reliable Reset (RST)
 - Normalizer transmits a ACK for every RST it sends out
 - Causes the IP ID to increase regardless of service running on particular port or not

Implementation

- Developed “norm”
- User level application
 - Handles IP, TCP, UDP, and ICMP
 - 4,800 lines of C code
 - Uses libpcap to capture packets
 - Raw socket used to forward packets
 - Source code available on sourceforge.net

Test Utility - NetDude



Tests

- **Baseline test consisted of 3 trace files**
 - Trace 1: 100000 packet trace collected from Lawrence Berkley National Laboratory
 - Contained mix of TCP (88%), UDP (10%), ICMP (1.5%), and misc (IGMP, ESP, tunneled IP) (0.2%)
 - Trace 2: Trace derived from Trace 1
 - TCP headers replaced with UDP headers
 - Trace 3: 100000 packet trace
 - Contained 92 byte UDP packets generated with netcat

Results

- Test 1

| <i>Memory-to-memory copy only</i> | | |
|-----------------------------------|-----------|-----------|
| Trace | pkts/sec | bit rate |
| T1,U1 | 727,270 | 2856 Mb/s |
| U2 | 1,015,600 | 747 Mb/s |

- No normalization

- Test 2

| <i>All checks enabled</i> | | |
|---------------------------|----------|-----------|
| Trace | pkts/sec | bit rate |
| T1 | 101,000 | 397 Mb/s |
| U1 | 378,000 | 1484 Mb/s |
| U2 | 626,400 | 461 Mb/s |

| <i>Number of Normalizations</i> | | | | | |
|---------------------------------|---------|-----|-----|------|---------|
| Trace | IP | TCP | UDP | ICMP | Total |
| T1 | 111,551 | 757 | 0 | 0 | 112,308 |

More Results

- Duplicating every TCP packet in Trace 1

| <i>All checks enabled</i> | | |
|---------------------------|----------|----------|
| Trace | pkts/sec | bit rate |
| T1 | 101,000 | 397 Mb/s |
| T1-dup | 60,220 | 236 Mb/s |

More Results

- More stressful test of flooding the normalizer with packets
- Test Setup
 - Use Trace 1, fragment every packet with IP payload > 16 bytes
 - Randomize the order of the fragment stream from 100 packets to 2000 packets

| rnd intv'l | input frags/s | frag'ed bit rate | output pkts/sec | output bit rate | pkts in cache |
|------------|---------------|------------------|-----------------|-----------------|---------------|
| 100 | 299,670 | 86Mb/s | 9,989 | 39Mb/s | 70 |
| 500 | 245,640 | 70Mb/s | 8,188 | 32Mb/s | 133 |
| 1,000 | 202,200 | 58Mb/s | 6,740 | 26Mb/s | 211 |
| 2,000 | 144,870 | 41Mb/s | 4,829 | 19Mb/s | 335 |

Conclusions

- Normalization will become more important as companies place security concern in the hands of NIDS
- Normalizer box seems like an inexpensive solution, yet becomes another box to place in front of your internet connection
- Kernel implementation would have been nice, to see how fast it really runs
- Seems like another good FPX App!