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

Mark Handley, Vern Paxson, Christian Kreibich

Presented By: Chris Neasbitt

Outline

- Introduction
- Normalization
- Tradeoffs
- Attacks & Defenses
- Normalization Approach
- Normalization Examples
- Evaluation
- Results

Introduction

- Passive NIDS face a fundamental problem
  - Attackers can utilize the ambiguities of network protocols to evade detection
- Exploitable ambiguities
  - NIDS may lack complete analysis of the full range of behavior for a particular protocol
  - NIDS may lack information of the victim's end-system
  - NIDS may lack a detailed understanding of the network topology

Normalization

- Attempts to solve the evasion by ambiguity by normalizing all network traffic
- Normalizer
  - is placed in the direct path of all network traffic
  - normalizes the packet stream to the NIDS to remove ambiguities
- Other Solutions
  - Host based IDS
  - Intranet Details Database
  - Bifurcating analysis

Tradeoffs

- Normalization vs. Protection
- End-to-End Semantics
- End-to-End Performance
- Amount of State Held
- Inbound vs. Outbound Traffic

Tradeoffs

- Normalization vs. Protection
  - the normalizer's position in the network makes it ideal to prevent some known attacks
  - ideal place for an Intranet firewall
- End-to-End Semantics
  - preserve the semantic meaning of the traffic in the face of normalization
- End-to-End Performance
  - traffic normalization could have an adverse affect on performance of network applications

Tradeoffs

- Amount of State Held
  - reconstruction of a flow for analysis requires some amount of state to be held
  - State Explosion
  - Triage
- Inbound vs. Outbound Traffic
  - Main task is to protect the NIDS from ambiguities
  - While located in front of the NIDS to normalize incoming traffic, two can be used to normalize both directions

Attacks and Defenses

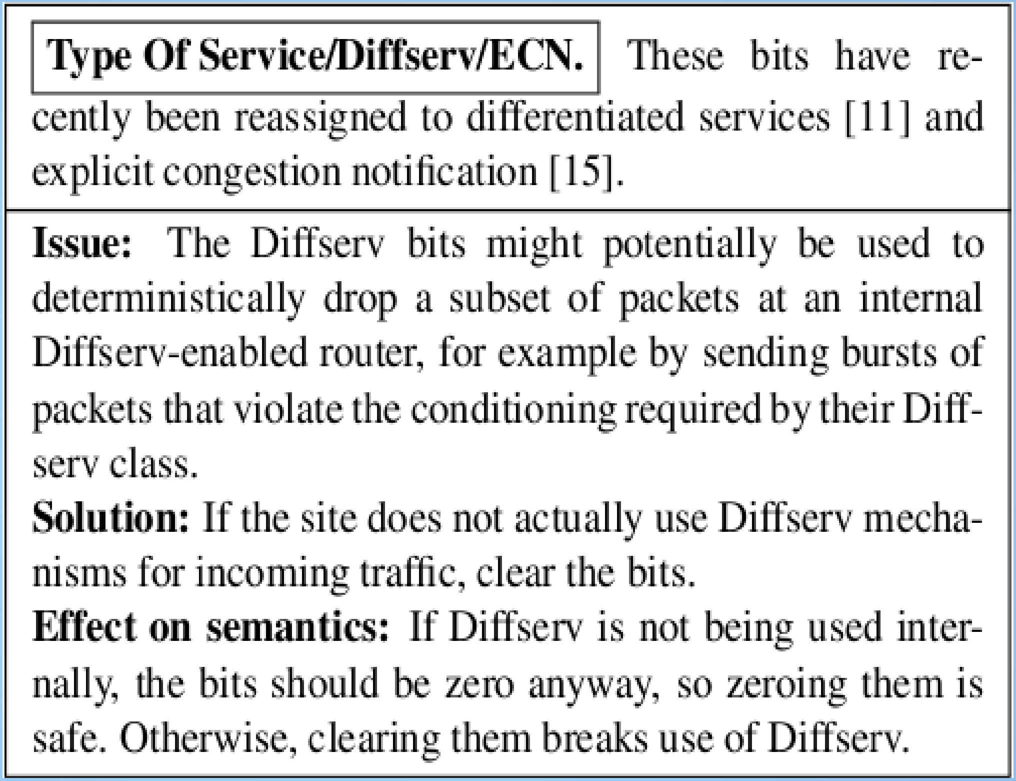
- An attacker may attempt to subvert the normalizer
  - Stateholding Attacks
    - SYN Flood
    - Fragmentation
    - Can monitor memory usage and scale back stateholding on some flows
  - CPU Overload Attacks
    - Combined with stateholding attacks
    - Decrease system performance by analyzing a flood of difficult to normalize flows

Attacks and Defenses

- Cold Start Attack
  - a certain amount of state is required to normalize some flows
  - this state can not acquired when the normalizer first starts
    - flows already in progress
  - an attacker can attempt to evade the normalizer by keeping a long term connection open
    - when the normalizer is restarted, initiate the malicious flow

Normalization Approach

- Systematic analysis of the possible ambiguities caused by each header element
    - range of values
    - semantics
    - take the approach of normalizing everything



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.

Normalization Examples

- IP identifier and Stealth Port Scans
- Reliable RST
- Cold Start of TCP
- Incompleteness

Normalization Examples

- IP identifier and Stealth Port Scans
  - Use the incrementing IP identifier of a patsy machine to detect services running on a victims machine
  - IP id scrambling and Reliable RST
- Reliable RST
  - sends a keep-alive ACK to the receiver of a RST
  - receiver should resend a response back
    - RST if the connection closed, or ACK if the connection is not
    - Each of the alternatives leaves the normalizer in an unambiguous state

Normalization Examples

- Cold Start of TCP
  - if the traffic is outbound then initialize state
  - if inbound, incoming packet transformed into a keep-alive and then send to its destination
    - if a connection exists, the receiver should send back a response ACK
    - if not, will respond with a RST or not at all
  - window scaling is still an issue
    - need window scaling factor

Normalization Examples

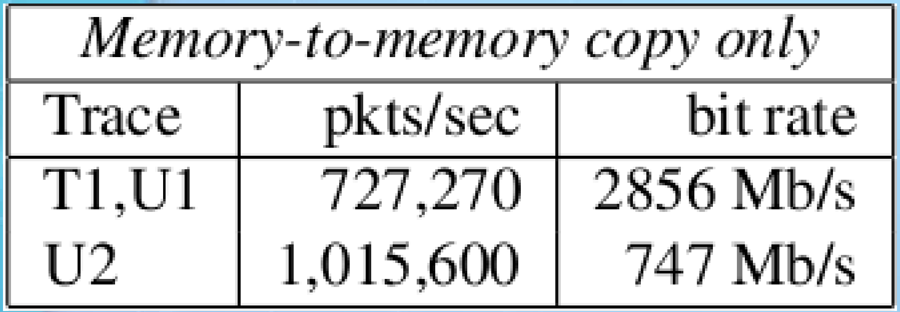
- Incompleteness
  - cannot remove all ambiguities
  - sometimes the application semantics are necessary to removing ambiguities
    - its unrealistic for a normalizer to know all of the application semantics for the applications running in the intranet
    - TCP urgent pointer

Evaluation

- Implemented a fairly complete normalizer prototype called norm
  - IP, TCP, UDP, and ICMP
  - 4,800 lines of C code
  - utilizes libpcap
  - runs as a user mode application
- Utilized 3 trace files in testing
  - T1: 100K trace from LBNL containing mostly TCP
  - U1: derived from T1 replacing every TCP header with a UDP header
  - U2: 100K trace of entirely 92-byte UDP packets

Results

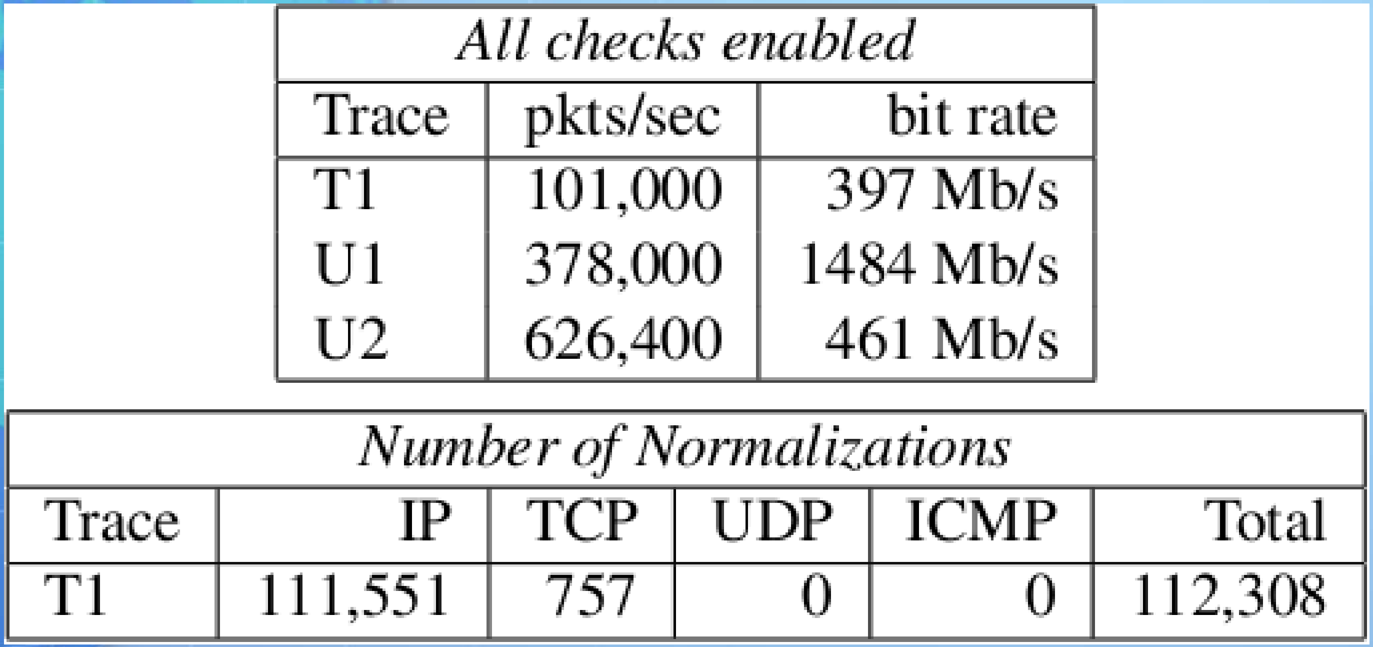
- FreeBSD 4.2, 1.1GHz AMD Thunderbird machine
- Memory to Memory Copy



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

Results

- All checks enabled normalization of both inbound and outbound traffic

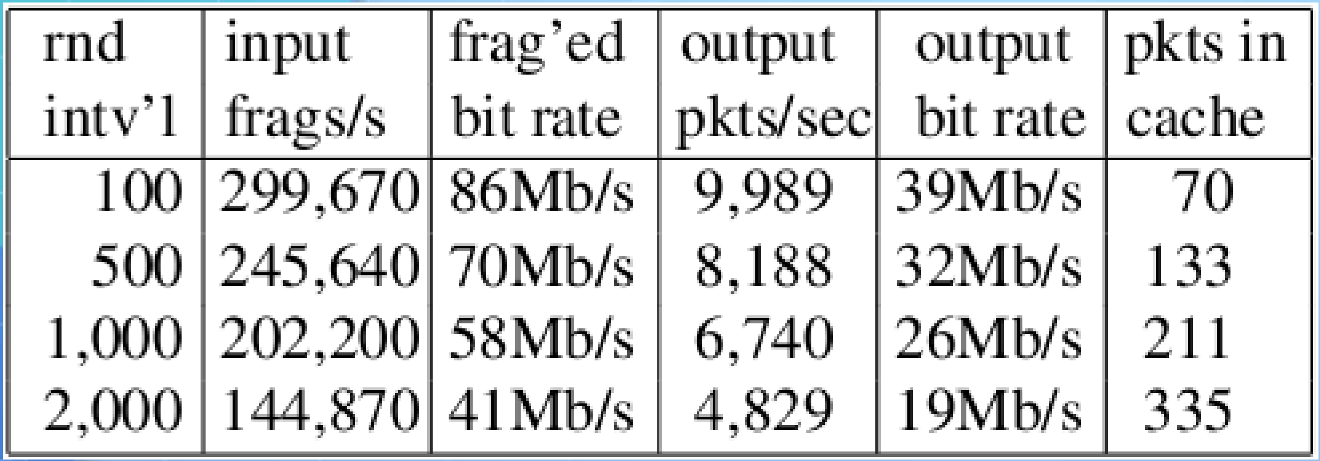


All checks enabled		
Trace	pkts/sec	bit rate
T1	101,000	397 Mb/s
U1	378,000	1484 Mb/s
U2	626,400	461 Mb/s

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

Results

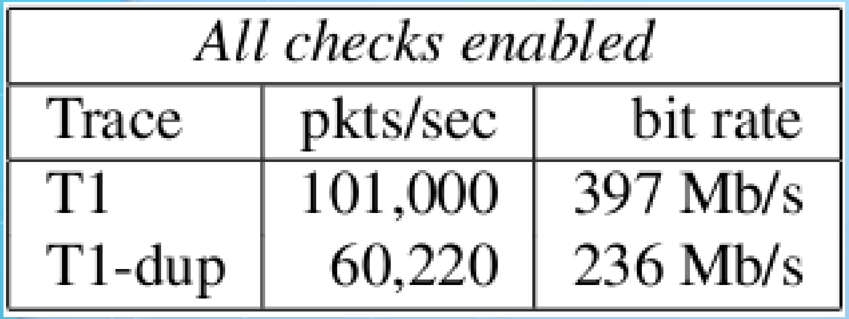
- Packet fragmentation test
  - took every packet in the T1 trace and fragmented it if the payload was over 16 bytes.
  - increasingly randomized the order of the packets



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

Results

- Inconsistent TCP retransmissions
  - duplicated every packet in T1



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

Questions?