SPANIDS

A Parallel High-bandwidth Network Intrusion Detection Platform

Lambert Schaelicke
&
SPANIDS Group
University of Notre Dame
Outline

- Network Intrusion Detection Overview
- Performance Requirements
- Parallel NIDS: Spanids
- Prototype Development
1. Network Intrusion Detection Overview
Network Intrusion Detection

Data security and integrity has become a major concern.

- prevent: firewalls, encryption, anti-virus tools, patches etc.
  - first line of defense
  - always catching up with attackers

- detect: alert administrator of attack or intrusion
  - allows reaction to unpreventable attacks
  - network & host-based intrusion detection

Network IDS complements other defense mechanisms.
General-purpose NIDS Architecture

- sensor analyzes all network packets
- real-time detection of attacks
- filters out packets of interest
- ‘database’ back-end for long-term storage and alert correlation
NIDS Approaches

Rule-based

- apply set of rules to each packet
- issue alert upon match
- limited to known attacks

Anomaly-based

- establish normal network behavior
- issue alert upon deviation from norm
- can potentially detect previously unknown attacks
Snort NIDS

Open-source Intrusion Detection

- popular NIDS software with large user base
- frequently updated rule set

Rule-based

- inspect packet header and/or payload
- covers ICMP, TCP, UDP, ... and all popular application protocols
- specific attacks and policy violations
- over 2000 default rules
Snort Rule Example

MS SQL Buffer Overflow Attack

    alert udp $EXTERNAL_NET any -> $SQL_SERVERS any
      (msg:"MS-SQL probe response overflow attempt";
       content:"|05|"; depth:1; byte_test:2,>512,1;
       content:"|3b|"; distance:0; isdataat:512,relative;
       content:"!|3b|"; within:512;
       classtype:attempted-user;
       sid:2329; rev:2;)

Bro NIDS

Rule-based, stateful analysis

- keeps track of connections
- applies scripted policies to connections and/or payload
- kernel-level packet filter may reduce analysis load
- open-source, but less well-supported than Snort
Alternative: Anomaly Detection

Establish Normal Behavior

- track high-level traffic statistics
  - packet rates, sizes, ports ...
- in most cases automatically extracted during learning phase

Alert upon Deviation from Norm

- so far successful mostly for DOS attacks
- too many false positives in general scenarios
- difficult to adjust to changing baseline behavior
Attacking & Avoiding the NIDS

Rule-based

- overwhelm NIDS with high-intensity traffic
- deviate from known attack patterns

Anomaly-based

- minimize perturbation of normal traffic
  - e.g. slow ramp-up of DOS attack, slow port scan
- insert attacks during learning period

Noise: Generate Large Number of non-critical Alerts
2. NIDS Performance
NIDS Performance

1. Qualitative = Sophistication
   - detect all true attacks and intrusions
   - minimize false alerts

2. Quantitative = Speed
   - inspect all packets under all network conditions
   - packet loss degrades quality

Both factors are important!
Snort Performance Requirements

1. Header Inspection
   - constant per-packet cost
   - limited by interrupt handling and protocol processing overhead

2. Payload Analysis
   - scales with payload size
   - limited by main memory bandwidth

Most rules combine both types of analysis.
Snort Performance Characteristics

- 1.2 Ghz P-3, saturated 100 Mbit Ethernet, 512-byte frames
- standard Snort distribution includes over 2000 rules
Measurement Approach

Constant network traffic

- `ttcp` session over 100 Mbit Ethernet between Linux hosts
- fixed data and packet rate – nearly 100% saturation
- passive tap forwards traffic to test system

Increase number of Snort rules

- observe packet loss
- record number of rules when exceeding 2.5 % loss
- separate header and payload rules
Experimental Setup

Six Different NIDS Platforms

- x86 Intel & AMD, 180 to 2400 Mhz, single and dual processor
- Linux and FreeBSD
- gauge architectural and OS sensitivity

Traffic Scenarios

- 64 – 512 – 1000 – 1452 byte packets
- cover complete spectrum of packet sizes
Results Normalization

# Header Rules * Packets / Second

- approximately constant cost per packet
- account for varying packet rates in tests

# Payload Rules * Mbytes / Second

- approximately constant per-byte cost
- account for varying effective bandwidth
Header Rule Performance

<table>
<thead>
<tr>
<th>Rules</th>
<th>64</th>
<th>512</th>
<th>1000</th>
<th>1452</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packets/s (million)</td>
<td>0.0</td>
<td>2.5</td>
<td>5.0</td>
<td>7.5</td>
</tr>
</tbody>
</table>

- Opteron 64
- P4/2400-dual
- P3/1200
- P3/800-single
- Opteron-32
- P4/1800
- P3/600
- PPro/180
Payload Rule Performance

Rules * Mbyte/s

Packet Size (bytes)

0 500 1,000 1,500 2,000 2,500

64 512 1000 1452

P4/2400-dual
Opteron-64
P3/1200
P3/800-single
Opteron-32
P4/1800
P3/600
PPro/180
OS Sensitivity

- Linux outperforms BSD for header rules
- small per-packet cost – interrupt handling dominates
Multiprocessor Sensitivity

- some benefit from offloading interrupts
- minor impact on payload analysis
Performance: Conclusions

CPU Speed Not a Good Indicator of NIDS Capabilities

- main memory bandwidth critical
- deep pipelines hurt NIDS performance
- some difference between kernels
- little benefit from multiprocessing
  - but server-class memory subsystems helps
3. Parallel NIDS: Spanids
The Sensor Bottleneck

Single-stream Design Limits Performance

- general-purpose system overwhelmed by interrupts
  - GigE: up to 1.5 million packets per second
  - > 1 µs per interrupt: CPU is saturated
- limited bandwidth
  - 125 Mbyte/s saturates 32-bit PCI bus
  - 3 data transfers: DMA, kernel read, user write

PC/Workstation incapable of receiving Gb/s traffic – unsuitable as high-bandwidth NIDS platform.
Parallel NIDS

Distribute traffic over multiple sensors.

• exploit concurrency in network traffic
• greater aggregate performance
• retains benefit of commodity hardware & software
  - cost-effective & flexible
  - rides desktop performance curve
  - benefits from software improvements
• scalable: just add sensors
Parallel Architecture

- custom load balancer distributes packets across sensor array
- off-the-shelf sensors running standard NIDS software
- aggregate and forward alerts to analysis engine
Parallel NIDS Challenges

Effective Distribution of Work

- (nearly) equal load for all sensors

Scalability

- scale with offered network load
- load balancing must not become bottleneck
- avoid maintaining per-connection or per-packet state
Minimize Vulnerabilities

Throughput & Scalability

- must keep up with peak network load
- adapt to load changes

Limited View of Each Sensor

- potential loss of precision
- avoid distributing connections over multiple sensors
- current NIDS software keeps limited state anyway
Parallel NIDS Approaches

**Flow Balancing: TopLayer**

- round-robin distribution of connections
- based on Cisco router
- stateful: keeps track of active connections

**Parallel Routing**

- round-robin scatter to array of routers
- forward flows to sensor array
Other High-speed NIDS Approaches

Pre-Filtering

- analyze only ‘interesting’ packets
- software (Bro) or hardware filter
- which packets?

High-end System & Optimized Driver

- may support very light-weight NIDS processing
- does not scale to next generation networks
The SPANIDS Approach

- hash connection ID to large number of buckets: fixed-size state
  - src/dest IP, src/dest port
  - avoid breaking up network flow
- assign buckets to sensors
• 60 minute trace of ND Campus feed

• 12 bit XOR hash of source/destination IP address, port numbers
Dynamic Loadbalancing

- reassign hash buckets when sensor load exceeds threshold
  - which bucket?
- in initiated by flow control feedback from sensor
  - based on packet buffer utilization
Hot Buckets

- observe bucket load: packet and/or data rate
- rehash when exceeding threshold
- potentially multiple levels of hashing
Bucket Move vs. Promote

- per-sensor list of N hot buckets
- table of sensor packet rates

- compare hottest buckets with average sensor rate
  - above certain percentage: promote to rehash
Evaluation

Trace-driven Simulation

- event-driven models of load balancer and sensors
- approximate sensor performance & issue flow control messages
- functionally accurate model of load balancer operation

The Trace Challenge

- real-life traces are benign & don’t stress the load balancer
- must design for, and test worst-case
- solution: synthetic traces
• recover from imbalance, move-only
4. Spanids Prototype
Prototype Overview

Low-cost Sensors

- general-purpose rack-mount Opteron servers
- Linux running Snort

Scalable Loadbalancing

- FPGA-based custom hardware combined with commodity switch
- evaluate variety of distribution schemes

Handle Gigabit/s traffic with precision of single-stream design.
Prototype Architecture

- FPGA-based loadbalancer rewrites Ethernet MAC address
- external commodity switch forwards packets
- two GigE ports on FPGA, off-the-shelf platforms available
Loadbalancer Architecture

- two full-duplex GigE interfaces
- PCI target
- hash-based loadbalancer
• input FIFO synchronizes GigE clock domains

• forward packet to router and delay pipeline
Loadbalancer Packet Flow (2)

- hash packet header & determine target sensor
- delay packet to compensate for routing delay
- process flow control information
Loadbalancer Packet Flow (3)

- rewrite destination MAC address
- recalculate CRC and forward packet
- update performance monitors in PCI subsystem
Configuration and Monitoring

Initialization

- hardware requests sign-on via broadcast UDP packet
- sensors reply with MAC address
- load balancer builds initial hash tables

Monitoring

- through host PCI bus
- status & global statistics
- per-sensor statistics with snapshot capability
Evaluation

Trace-based

- synthetic traffic for systematic testing
- replay traces from ND campus and other sources
- aggregate sender/receiver to saturate GigE link

Life Traffic

- access to ND campus internet link
Prototype Hardware

- Virtex-II FPGA board with PCI interface (Dini Group)
- dual GigE daughtercards (Metanetworks)
- PCI riser card to fit in host PCI slot
Control Software

- configures load balancer hardware
- monitors performance
- not involved in actual load balancing
Status

- NIDS performance requirements established
- trace-based simulator completed
- initial version of synthetic trace generator completed
- currently 15% of FPGA resources utilized
  - Gigabit Ethernet & PCI Interface
  - performance monitor
  - packet decoder & hash functions
- flow control module added to Linux packet socket
Acknowledgements

- Co-PI: Curt Freeland
- Graduate students: Thomas Slabach, Kyle Wheeler, Anthony Schorer
- Many undergraduate students
- Funded by NSF ANIR