Kernel packet capture technologies

Éric Leblond

Stamus Networks

October 1, 2015

Éric Leblond

Co-founder of Stamus Networks

- Company providing network probe based on Suricata
- Focusing on bringing you the best of Suricata IDS technology

Open source hacker

- Suricata core developer
- Netfilter core team member

Raw socket: definition

A raw socket is an internet socket that allows direct sending and receiving of Internet Protocol packets without any protocol-specific transport layer formatting.

Wikipedia

"The End of the Internet"

[raw socket ...] spells catastrophe for the integrity of the Internet.

Steve Gibson in 2001

"The End of the Internet"

- Talking about introduction of raw socket in MS Windows
  - Allow users to write any packets
  - Could be used to abuse protocol and [poorly implemented] OS
- More info at `http://www.informit.com/articles/article.aspx?p=27289`

Raw socket: usage

Send and receive

- Send low level message: icmp, igmp
- Implement new protocol in userspace

Sniffing

- Capture traffic
- Promiscuous mode
- Use by network monitoring tools
  - Debugging tools: tcpdump, wireshark
  - Monitoring tools: iptraf, ntop, NSA
  - Intrusion detection systems: snort, bro, suricata

Network Intrusion Detection System: definition

An intrusion detection system (IDS) is a device or software application that monitors network or system activities for malicious activities or policy violations and produces reports to a management station.

Wikipedia

Network Intrusion Detection System: challenge

IDS detection rule

```
alert http $EXTERNAL_NET any -> $HTTP_SERVERS any (msg:"ET WEB_SPECIFIC_APPS Webmin Directo
ry Traversal"; flow:to_server,established; content:"POST"; http_method; content:"/save_en
v.cgi"; http_uri; fast_pattern:only; content:"&user="; http_client_body; content:"|2e 2e 2
f|"; distance:0; http_client_body; reference:url,sites.utexas.edu/iso/2014/09/09/arbitrary-
file-deletion-as-root-in-webmin/; classtype:misc-attack; sid:2019157; rev:3;)
```

Some data

- Complexity of rule
  - Work on recontructed stream
  - Protocol field analysis
  - Pattern recognition on ungzipped content (http_server_body)
- Got around 15000 rules in standard ruleset
- Need to inspect 10Gbps of trafic or more

Suricata: Open source & multi threaded IDS

- IDS and IPS engine
- Get it here: `http://www.suricata-ids.org`
- Project started in 2008
- Open Source (GPLv2)
- Funded by consortium members (and originaly US government)
- Run by Open Information Security Foundation (OISF)
- More information about OISF at `http://www.oisf.net/`

Suricata Features

- High performance, scalable through multi threading
- Protocol identification
- File identification, extraction, on the fly MD5 calculation
- TLS handshake analysis, detect/prevent things like Diginotar
- Hardware acceleration support:
- Useful logging like HTTP request log, TLS certificate log, DNS logging
- Lua scripting for detection

libpcap

- Multi OS abstraction for packet capture
- All *nix, Windows
- Multi layer: Network, USB, ...

Raw socket: the initial implementation

A dedicated socket type

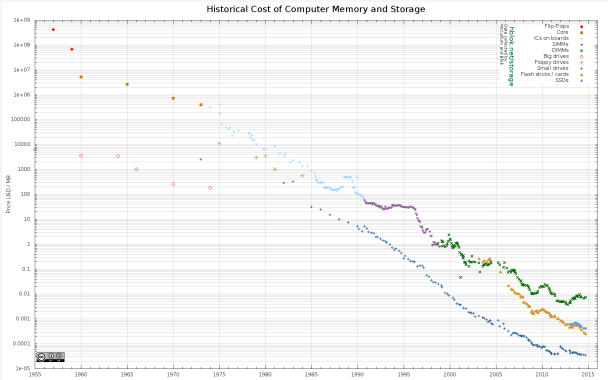
```
#include <sys/socket.h>
#include <netinet/in.h>
raw_socket = socket(AF_INET, SOCK_RAW, int protocol);
```

Straight socket mode

- Get packet per packet via recvmsg
- Optional ioctl
  - Get timestamp

Memories of another time

"640 K ought to be enough for anybody."



Memory contraint design

- No preallocation
- On demand only

Disclaimer

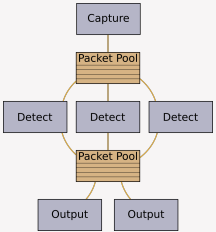
IDS design

Monoprocess

No Performance for you, go home now.

Marty Roesch about multithread and network data processing, 2010

Suricata architecture



NAPI (2001-200?)

Reducing interrupts usage

- Interrupts tempest at high packet rate
- All CPU time is sued to handle the interrupts
- NIC driver needs to be updated

No direct change for packet capture

- Change internal to device driver
- Direct performance impact on packet capture

NAPI performance

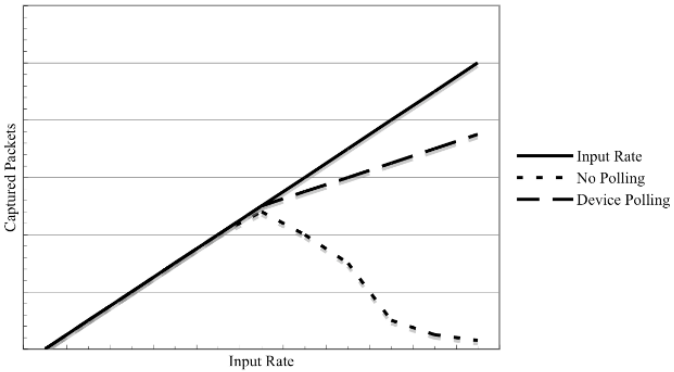


Figure 1. – Packet Capture Performance: Polling vs. non-polling

Table extracted from `luca.ntop.org/Ring.pdf`

Problem of the socket mode

Internal path

- Data in card buffer
- Data copied to skb
- Data copied to socket
- Data read and copied by userspace

Memory map approach

Sharing is the solution

- Kernel expose some memory
- Userspace access memory directly
- Spare a message sending for every packets

mmap internal path

- Data in card buffer
- Data copied to skb
- Data copied to ring buffer
- Userspace access data via pointer in ring buffer

TPACKET_V2

- setup
  - socket(): creation of the capture socket
  - setsockopt(): allocation of the circular buffer (ring) via PACKET_RX_RING option
  - mmap(): mapping of the allocated buffer to the user process
- capture
  - poll(): to wait for incoming packets
- shutdown
  - close(): destruction of the capture socket and deallocation of all associated resources.

Memory organization

Ascii art

```
        block #1                    block #2
+---------+---------+     +---------+---------+
| frame 1 | frame 2 |     | frame 3 | frame 4 |
+---------+---------+     +---------+---------+

        block #3                    block #4
+---------+---------+     +---------+---------+
| frame 5 | frame 6 |     | frame 7 | frame 8 |
+---------+---------+     +---------+---------+
```

Components

- Frame contains a datagram data
- Blocks are physically contiguous region of memory

Performance

Packet Size (bytes)	Linux 2.6.1 with NAPI and standard libpcap	Linux 2.6.1 with NAPI and libpcap-mmap[6]	FreeBSD 4.8 with Polling
64	2.5 %	14.9 %	97.3 %
512	1.1 %	11.7 %	47.3 %
1500	34.3 %	93.5 %	56.1 %

Table 2. – Percentage of captured packets (generated by stream.c) using kernel polling

Graph extracted from `luca.ntop.org/Ring.pdf`

Suricata architecture

MMAP option

- Support of TPACKET_V2
- Zero copy mode

Implied changes

- Access data via pointer to ring buffer cell
- Release data callback

PF_RING original design (2004)

Architecture

- ring design
- mmap
- capture only interface
  - skip kernel path
  - put in ring buffer and discard
- user access the ring buffer

Project

- Project started by Luca Deri
- Available as separate sources

PF_RING performance

Packet Size (bytes)	Linux 2.6.1 with NAPI and libpcap standard	Linux 2.6.1 with NAPI and libpcap-mmap[7]	FreeBSD 4.8 with Polling	Linux 2.6.1 with NAPI+PF_RING and extended libpcap
64	2.5 %	14.9 %	97.3 %	75.7 %
512	1.1 %	11.7 %	47.3 %	47.0 %
1500	34.3 %	93.5 %	56.1 %	92.9 %

Table 3. – Percentage of captured packets (generated by stream.c) using kernel polling

- Show real improvement on small size packets
- Pre optimisation result
- Better result in following version due to a better poll handling

Table extracted from `luca.ntop.org/Ring.pdf`

PF_RING going multicore (around 2008?)

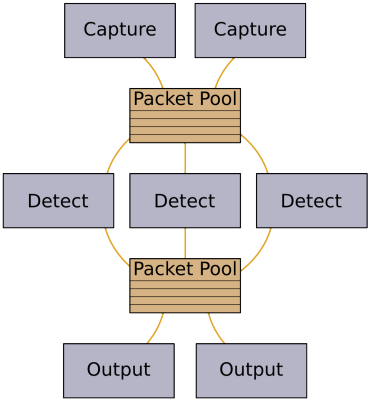
Sharing the load

- Each core has a finite bandwidth capability
  - Multicore CPU were introduced in 2006
  - Sharing load become common
- Previously separate hardware was used to split the network load

Straight forward solution

- Allow multiple sockets to be attached to one interface
- Load balance over the attached sockets

Suricata autofp multi reader



PF_RING code

Build system and sources

- Custom build system
- No autotools or cmake
- Include patched drivers

SVN stats

```
git log --format=format:"%s" | sort | uniq -c | sort -n | tail -n10
 15 Minor change
 20 fix
 20 minor changes
 22 lib refresh
 30 Library refresh
 43 minor change
 67 minor fix
```

David Miller in da place

AF_PACKET load balancing (2011)

Multiple sockets on same interface

- Kernel does load balancing
- Multiple algorithms

LB algorithm

- Round-robin
- Flow: all packets of a given flow are send to the same socket
- CPU: all packets treated in kernel by a CPU are send to the same socket

AF_PACKET CPU Load balancing

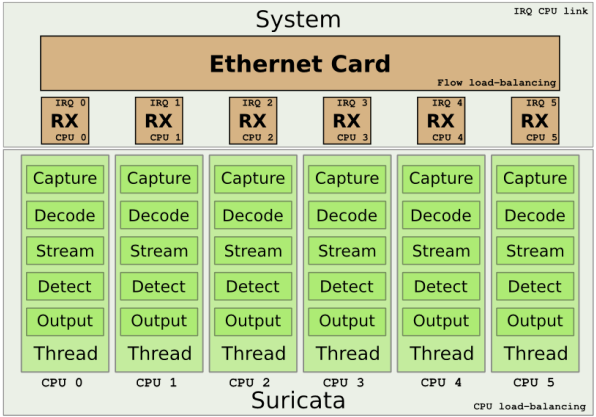
RSS queues

- Multiqueue NIC have multiple TX RX
- Data can be split in multiple queues
  - Programmed by user
  - Flow load balanced

RSS queues load balancing

- NIC does load balancing using hash function
- CPU affinity is set to ensure we keep the cache line

Suricata workers mode



tpacket_v3 (2011)

The problem

- Cell are fixed size
- Size is the one of biggest packet (MTU)
- Small packets use same memory as big one

Variable size cells

- Ring buffer
- Update memory mapping to enable variable sizes
- Use a get pointer to next cell approach

Netmap (2012)

- Similar approach than PF_RING
  - skip kernel path
  - put in ring buffer and discard
- User access the ring buffer
- Paired with network card ring

More info `http://queue.acm.org/detail.cfm?id=2103536`

Performances

Packet forwarding	Mpps
FreeBSD bridging	0.690
netmap + libpcap emulation	7.500
netmap, native	10.660
Open vSwitch	**Mpps**
optimized, FreeBSD	0.790
optimized, FreeBSD + netmap	3.050
Click	**Mpps**
user space + libpcap	0.400
linux kernel	2.100
user space + netmap	3.950

Table by Luigi Rizzo

AF_PACKET rollover option (2013)

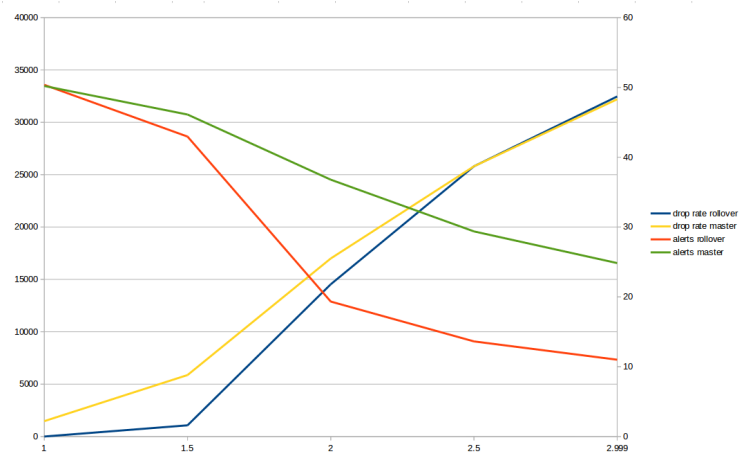
Single intensive flow

- Load balancing is flow based
- One intensive flow saturate core capacity
- Load needs to be shared

Principle

- Move to next ring when ring is full
- As a load balancing mode
- As a fallback method

Rollover and suricata (1/2)



Graph by Victor Julien

Rollover and suricata (2/2)

A TCP streaming issue

- Rollover activation lead to out of order packets
- Fool TCP stream reconstruction by suricata
- Result in invalid streams

Possible solution

- Evolve autofop multicapture
- Decode and dispatch packets

DPDK (2012-)

Data Plane Development Kit

- set of libraries and driver
- design for fast packet processing
- impact on software architecture

Architecture

- multicore framework
- huge page memory
- ring buffers
- poll-mode drivers

Suricata workers mode limit

Packet treatment can be really long

- Involve I/O on disk or network
- Huge computation like regular expression

Ring buffers are limited in size

- A slow packet can block a whole buffer
- Suricata need to dequeue faster

Need to evolve Suricata architecture

Switch to asynchronous

- Release ring buffer elements as fast as possible
- Buffer in userspace

An enhanced autofp approach?

- Fast decode
- Copy data to packet pool of detect thread
- With a fast decision
- Release data

Conclusion (1/2)

A small subject and a huge evolution

- Has follow evolution of hardware architecture
- Always need to deal with more speed
  - 10Gbps is common
  - 100Gbps is in sight

Multiple technologies

- Vanilla kernel propose some solutions
- Patching may be required to do more

Conclusion (2/2)

Do you have questions ?

Contact me

- Mail: `eleblond@stamus-networks.com`
- Twitter: @Regiteric

More information

- Suricata: `http://www.suricata-ids.org`
- PF_RING: `http://www.ntop.org/products/packet-capture/pf_ring/`
- netmap: `http://info.iet.unipi.it/~luigi/netmap/`
- dpdk: `http://dpdk.org/`