Kernel packet capture technologies

Éric Leblond

Stamus Networks

October 1, 2015
Introduction

Why capture

Libcap and raw socket

AF_PACKET

PF_RING

AF_PACKET goes multi*

Netmap

Latest AF_PACKET evolution

++zero copy

Conclusion
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
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
Googled “How will the world end”.

Was not disappointed.

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

---

Steve Gibson in 2001
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

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
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 Directory 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 reconstructed stream
  - Protocol field analysis
  - Pattern recognition on unzipped content (http_server_body)
- Got around 15000 rules in standard ruleset
- Need to inspect 10Gbps of traffic or more

Éric Leblond (Stamus Networks)
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 originally 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

```c
#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
"640 K ought to be enough for anybody."

Memory contraint design
- No preallocation
- On demand only
IDS design

Monoprocess

No Performance for you, go home now.

Marty Roesch about multithread and network data processing, 2010

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

Table extracted from luca.ntop.org/Ring.pdf
Problem of the socket mode

<table>
<thead>
<tr>
<th>Internal path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data in card buffer</td>
</tr>
<tr>
<td>Data copied to skb</td>
</tr>
<tr>
<td>Data copied to socket</td>
</tr>
<tr>
<td>Data read and copied by userspace</td>
</tr>
</tbody>
</table>
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

```
+---------+---------+ +---------+---------+
| frame 1 | frame 2 | | frame 3 | frame 4 |
+---------+---------+ +---------+---------+
+---------+---------+ +---------+---------+
| frame 5 | frame 6 | | frame 7 | frame 8 |
+---------+---------+ +---------+---------+
```

Components

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

<table>
<thead>
<tr>
<th>Packet Size (bytes)</th>
<th>Linux 2.6.1 with NAPI and standard libpcap</th>
<th>Linux 2.6.1 with NAPI and libpcap-mmap⁶</th>
<th>FreeBSD 4.8 with Polling</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>2.5 %</td>
<td>14.9 %</td>
<td>97.3 %</td>
</tr>
<tr>
<td>512</td>
<td>1.1 %</td>
<td>11.7 %</td>
<td>47.3 %</td>
</tr>
<tr>
<td>1500</td>
<td>34.3 %</td>
<td>93.5 %</td>
<td>56.1 %</td>
</tr>
</tbody>
</table>

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

Graph extracted from [luca.ntop.org/Ring.pdf](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

<table>
<thead>
<tr>
<th>Packet Size (bytes)</th>
<th>Linux 2.6.1 with NAPI and libpcap standard</th>
<th>Linux 2.6.1 with NAPI and libpcap-mmap(^7)</th>
<th>FreeBSD 4.8 with Polling</th>
<th>Linux 2.6.1 with NAPI+PF_RING and extended libpcap</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>2.5 %</td>
<td>14.9 %</td>
<td>97.3 %</td>
<td>75.7 %</td>
</tr>
<tr>
<td>512</td>
<td>1.1 %</td>
<td>11.7 %</td>
<td>47.3 %</td>
<td>47.0 %</td>
</tr>
<tr>
<td>1500</td>
<td>34.3 %</td>
<td>93.5 %</td>
<td>56.1 %</td>
<td>92.9 %</td>
</tr>
</tbody>
</table>

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

Capture → Packet Pool → Detect → Packet Pool → Output

Capture → Packet Pool → Detect → Packet Pool → Output
PF_RING code

Build system and sources

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

SVN stats

```
git log --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
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
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

<table>
<thead>
<tr>
<th>Packet forwarding</th>
<th>Mpps</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeBSD bridging</td>
<td>0.690</td>
</tr>
<tr>
<td>netmap + libpcap emulation</td>
<td>7.500</td>
</tr>
<tr>
<td>netmap, native</td>
<td>10.660</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Open vSwitch</th>
<th>Mpps</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimized, FreeBSD</td>
<td>0.790</td>
</tr>
<tr>
<td>optimized, FreeBSD + netmap</td>
<td>3.050</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Click</th>
<th>Mpps</th>
</tr>
</thead>
<tbody>
<tr>
<td>user space + libpcap</td>
<td>0.400</td>
</tr>
<tr>
<td>linux kernel</td>
<td>2.100</td>
</tr>
<tr>
<td>user space + netmap</td>
<td>3.950</td>
</tr>
</tbody>
</table>

Table by Luigi Rizzo
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
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
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

<table>
<thead>
<tr>
<th>Switch to asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Release ring buffer elements as fast as possible</td>
</tr>
<tr>
<td>- Buffer in userspace</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>An enhanced autofp approach?</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Fast decode</td>
</tr>
<tr>
<td>- Copy data to packet pool of detect thread</td>
</tr>
<tr>
<td>- With a fast decision</td>
</tr>
<tr>
<td>- Release data</td>
</tr>
</tbody>
</table>
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
- netmap: http://info.iet.unipi.it/~luigi/netmap/
- dpdk: http://dpdk.org/