Why eBPF and XDP in Suricata matters

É. Leblond

OISF

Nov. 15, 2018

1 Problem
- Packet loss impact
- Elephant flow
- Work less to get more

2 Bypass
- Introducing bypass
- Bypass strategy

3 Hipster technologies to the rescue
- eBPF
- AF_PACKET bypass via eBPF
- eBPF support
- XDP support
- It’s just the beginning

4 Conclusion

1. Problem
    - Packet loss impact
    - Elephant flow
    - Work less to get more
2. Bypass
    - Introducing bypass
    - Bypass strategy
3. Hipster technologies to the rescue
    - eBPF
    - AF_PACKET bypass via eBPF
    - eBPF support
    - XDP support
    - It’s just the beginning
4. Conclusion

Impact of loosing packets

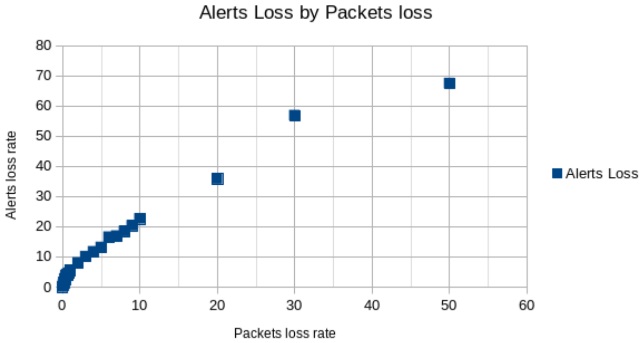
Methodology

- Use a sample traffic
- Modify the pcap file to have specified random packet loss
- Do it 3 times par packet loss
- Get graph out of that

Test data

- Using a test pcap of 445Mo.
- Real traffic but lot of malicious behaviors
- Traffic is a bit old

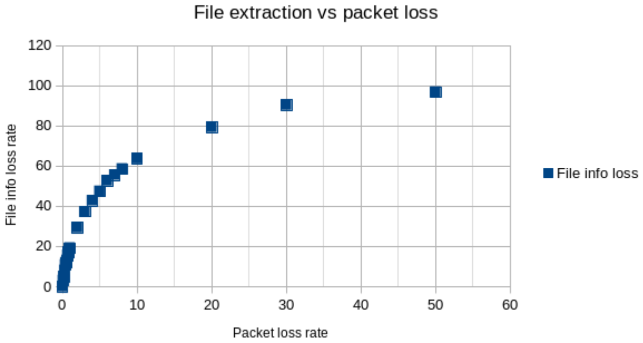
Alert loss by packet loss



Some numbers

- 10% missed alerts with 3% packets loss
- 50% missed alerts with 25% packets loss

The case of file extraction



Some numbers

- 10% failed file extraction with 0.4% packets loss
- 50% failed file extraction with 5.5% packets loss

1 Problem

- Packet loss impact
- Elephant flow
- Work less to get more

2 Bypass

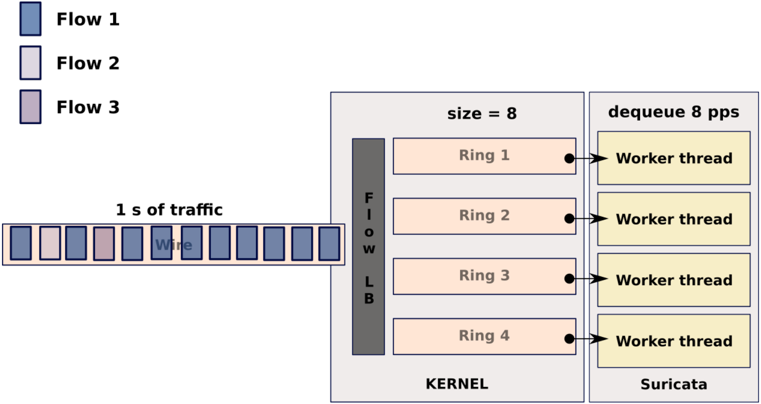
- Introducing bypass
- Bypass strategy

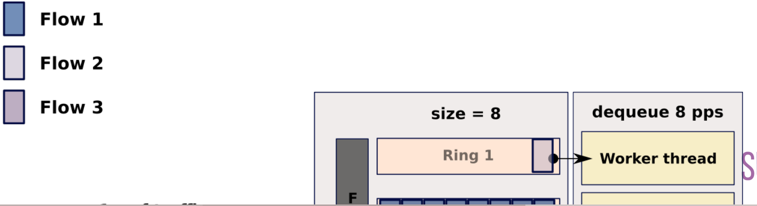
3 Hipster technologies to the rescue

- eBPF
- AF_PACKET bypass via eBPF
- eBPF support
- XDP support
- It's just the beginning

4 Conclusion

The elephant flow problem (1/2)





The elephant flow problem (2/2)

Ring buffer overrun

- Limited sized ring buffer
- Overrun cause packets loss
- that cause streaming malfunction

Ring size increase

- Work around
- Use memory
- Fail for non burst
    - Dequeue at N
    - Queue at speed N+M

1. Problem
    - Packet loss impact
    - Elephant flow
    - Work less to get more
2. Bypass
    - Introducing bypass
    - Bypass strategy
3. Hipster technologies to the rescue
    - eBPF
    - AF_PACKET bypass via eBPF
    - eBPF support
    - XDP support
    - It’s just the beginning
4. Conclusion

Stream depth method

Attacks characteristic

- In most cases attack is done at start of TCP session
- Generation of requests prior to attack is not common
- Multiple requests are often not even possible on same TCP session

Stream reassembly depth

- Reassembly is done till `stream.reassembly.depth` bytes.
- Stream is not analyzed once limit is reached
- Individual packet continue to be inspected

1. Problem
    - Packet loss impact
    - Elephant flow
    - Work less to get more
2. Bypass
    - Introducing bypass
    - Bypass strategy
3. Hipster technologies to the rescue
    - eBPF
    - AF_PACKET bypass via eBPF
    - eBPF support
    - XDP support
    - It’s just the beginning
4. Conclusion

1. Problem
   - Packet loss impact
   - Elephant flow
   - Work less to get more

2. Bypass
   - Introducing bypass
   - Bypass strategy

3. Hipster technologies to the rescue
   - eBPF
   - AF_PACKET bypass via eBPF
   - eBPF support
   - XDP support
   - It's just the beginning

4. Conclusion

Introducing bypass

Stop packet handling as soon as possible

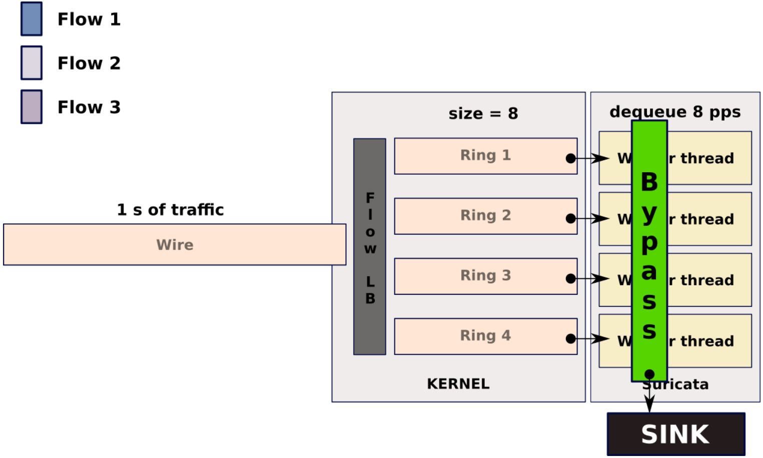
- Tag flow as bypassed
- Maintain table of bypassed flows
- Discard packet if part of a bypassed flow

Bypass method

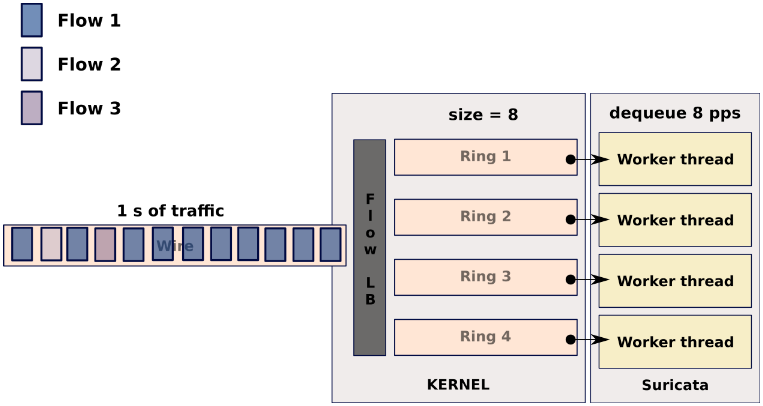
- Local bypass: Suricata discard packet after decoding
- Capture bypass: capture method maintain flow table and discard packets of bypassed flows

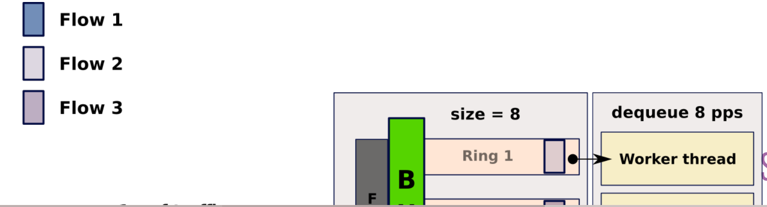
SURICON

Bypassing big flow: local bypass



Bypassing big flow: capture bypass





Implementation

Suricata update

- Add callback function
- Capture method register itself and provide a callback
- Suricata calls callback when it wants to offload

NFQ bypass in Suricata 3.2

- Update capture register function
- Written callback function
  - Set a mark with respect to a mask on packet
  - Mark is set on packet when issuing the verdict

1. Problem
   - Packet loss impact
   - Elephant flow
   - Work less to get more
2. Bypass
   - Introducing bypass
   - Bypass strategy
3. Hipster technologies to the rescue
   - eBPF
   - AF_PACKET bypass via eBPF
   - eBPF support
   - XDP support
   - It's just the beginning
4. Conclusion

Stream depth bypass

Stop all treatment after bypass

- Go beyond what is currently done
- Disable individual packet treatment once stream depth is reached

Activating stream depth bypass

- Set `stream.bypass` to yes in YAML

TLS bypass

- `encrypt-handling: bypass`

Selective bypass

Ignore some traffic

- Ignore intensive traffic like Netflix
- Can be done independently of stream depth
- Can be done using generic or custom signatures

The bypass keyword

- A new `bypass` signature keyword
- Trigger bypass when signature match
- Example of signature

```
pass http any any -> any any (content:"suricata.io"; \\
              http_host; bypass; sid:6666; rev:1;)
```

1. Problem
   - Packet loss impact
   - Elephant flow
   - Work less to get more
2. Bypass
   - Introducing bypass
   - Bypass strategy
3. Hipster technologies to the rescue
   - eBPF
   - AF_PACKET bypass via eBPF
   - eBPF support
   - XDP support
   - It's just the beginning
4. Conclusion

1 Problem
- Packet loss impact
- Elephant flow
- Work less to get more

2 Bypass
- Introducing bypass
- Bypass strategy

3 Hipster technologies to the rescue
- eBPF
- AF_PACKET bypass via eBPF
- eBPF support
- XDP support
- It's just the beginning

4 Conclusion

SURICON

Extended Berkeley Packet Filter

Berkeley Packet Filter

- Virtual machine inside kernel
- Arithmetic operations and tests on the packet data
- Filters are injected by userspace in kernel via syscall

Extended BPF

- Extended virtual machine: more operators, data and function access
- Various attachment points
  - Socket
  - Syscall
  - Traffic control
- Kernel and userspace shared structures
  - Hash tables
  - Arrays

LLVM backend

From C file to eBPF code

- Write C code
- Use eBPF LLVM backend (since LLVM 3.7)
- Use libbpf
  - Get ELF file
  - Extract and load section in kernel

BCC: BPF Compiler collection

- Inject eBPF into kernel from high level scripting language
- Trace syscalls and kernel functions
- `https://github.com/iovisor/bcc`

1. Problem
   - Packet loss impact
   - Elephant flow
   - Work less to get more
2. Bypass
   - Introducing bypass
   - Bypass strategy
3. Hipster technologies to the rescue
   - eBPF
   - AF_PACKET bypass via eBPF
   - eBPF support
   - XDP support
   - It’s just the beginning
4. Conclusion

SURICON

And now AF_PACKET

What's needed

- Suricata to tell kernel to ignore flows
- Kernel system able to
  - Maintain a list of flow entries
  - Discard packets belonging to flows in the list
  - Update from userspace

eBPF filter using maps

- eBPF introduce maps
- Different data structures
  - Hash, array, . . .
  - Update and fetch from userspace
- Looks good!

1. Problem
   - Packet loss impact
   - Elephant flow
   - Work less to get more
2. Bypass
   - Introducing bypass
   - Bypass strategy
3. Hipster technologies to the rescue
   - eBPF
   - AF_PACKET bypass via eBPF
   - eBPF support
   - XDP support
   - It’s just the beginning
4. Conclusion

Filtering

from BPF to eBPF

- Forget about `or` joined list: `not (1.2.3.4 or 2.3.4.5 or 12.3.34.4 or ...)`
- Maintain list in maps
- Search in list in constant time

More on maps

- Pinning
- Access from external tool

Available example filters

- filter.c: drop IPv6
- vlan_filter.c: accept packet for a set of VLANs

Pinned maps

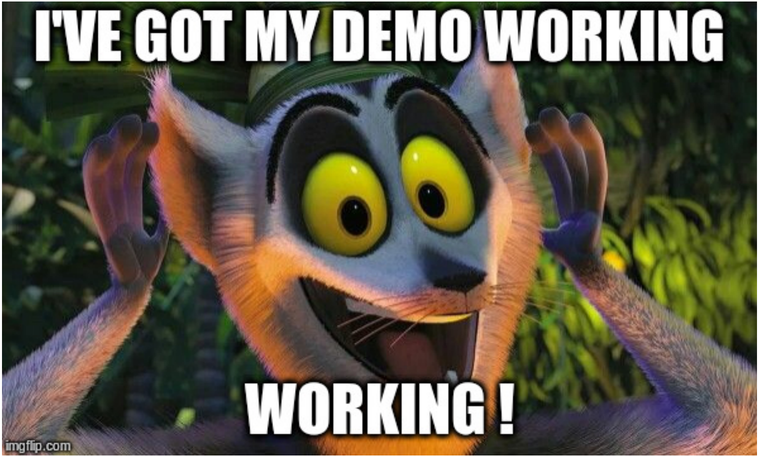
Expose maps to system

- Read and update map from external tools
- Update BPF filter dynamically

Demo

- On the wings of Murphy

Murphy will decide if I need to pass this slide fast



SURICON

Load balancing

Custom load balancer

- Return integer
- Readig socket determined by taking modulo

Available example filter

- lb.c: IP pair load balancing

Bypass

eBPF bypass

- Suricata specialized filter
- Flow tables for IPv4 and IPv6
- Bypass function add entry to flow table

Flow handling

- Dedicated thread in Suricata
- Dump table and handle cleaning

1. Problem
   - Packet loss impact
   - Elephant flow
   - Work less to get more
2. Bypass
   - Introducing bypass
   - Bypass strategy
3. Hipster technologies to the rescue
   - eBPF
   - AF_PACKET bypass via eBPF
   - eBPF support
   - XDP support
   - It's just the beginning
4. Conclusion

eXtreme Data Path

Reaching bare metal performance

- Answer to high performance need
  - DDoS fight
  - Custom protocol implementation
- Run userspace code
- When Linux network stack do too much

Motivation

- Avoid cost of skb creation
- "Kill" DPDK
  - Universal solution and APIs
  - Avoid non Linux application on Linux

SURICON

A recent Linux kernel feature

Run a eBPF code the earliest possible

- in the driver
- in the card
- before the regular kernel path

Act on data

- Drop packet (eXtreme Drop Performance)
- Transmit to kernel
- Rewrite and transmit packet to kernel
- Redirect to another interface
- CPU load balance

Implementation in Suricata

Similar to eBPF filter

- Same logic for bypass
- Only verdict logic is different

But annoying difference

- eBPF code does the parsing
- Need to bind to an interface

IPS and bypass

What about IPS bypass ?

- XDP_DROP is dropping
- Bypassing imply dropping

To light speed and beyond

- XDP_REDIRECT to send packet to TX queue of other NIC
- Direct transmit from hardware to hardware

CPU redirect

Non symetric RSS

- Non symetric hash function
- Low entropy key not always supported
- RSS=1 and burn one CPU

CPU Redirect to the rescue

- Load balance in XDP eBPF code
- skb creation is done in all CPUs

Stripping tunnels

Big Tunnel

- Can be an elephant flow
- Tunnelized flows can be non elephant
- Treating ad load balancing on internal flows can save the day

Strip tunnel header

- Decode tunnel header
- Find offset
- Move pointer to new start

1. Problem
   - Packet loss impact
   - Elephant flow
   - Work less to get more
2. Bypass
   - Introducing bypass
   - Bypass strategy
3. Hipster technologies to the rescue
   - eBPF
   - AF_PACKET bypass via eBPF
   - eBPF support
   - XDP support
   - It’s just the beginning
4. Conclusion

Complete hardware offload

- Join work with Netronome team
- Almost there
- Test to start soon

AF_XDP

New capture method

- Get packet at XDP stage
- Fully skip the Linux network stack

New architecture

- Shared memory
- User and Kernel lists

SURICON

1. Problem
   - Packet loss impact
   - Elephant flow
   - Work less to get more
2. Bypass
   - Introducing bypass
   - Bypass strategy
3. Hipster technologies to the rescue
   - eBPF
   - AF_PACKET bypass via eBPF
   - eBPF support
   - XDP support
   - It’s just the beginning
4. Conclusion

Conclusion

Suricata, eBPF and XDP

- Available in Suricata 4.1, need Linux 4.16
- Network card bypass for Netronome coming
- AF_XDP capture is now in Linux vanilla

More information

- Septun II: `https://github.com/pevma/SEPTun-Mark-II/`
- Suricata doc: `http://suricata.readthedocs.io/en/latest/capture-hardware/ebpf-xdp.html`

Questions ?

Thanks to

- Jesper Dangaard Brouer
- Alexei Starovoitov
- Daniel Borkmann

Contact me

- eleblond@oisf.net
- Twitter: @regiteric