# Suricata and XDP, Performance with a S like Security

É. Leblond

OISF

Nov. 29, 2018


OISF

OISF

# Who am I?

## Éric Leblond

- Stamus Networks co-founder
  - Editor of a threat hunting solution
  - including Suricata based appliances
- Netfilter core team member
  - Really low personal activity nowadays

## eleblond@oisf.net

- Long time member of OISF
- Suricata developer
  - In charge of packet acquisition
  - 1118 commits since 2010 (I like small patches)

OISF

# About the journey

## Adding bypass feature to Suricata

- 2 years of development to see less and get more done
- Using kick ass technologies before their documentation has been written.

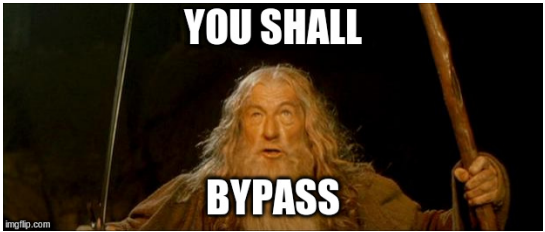
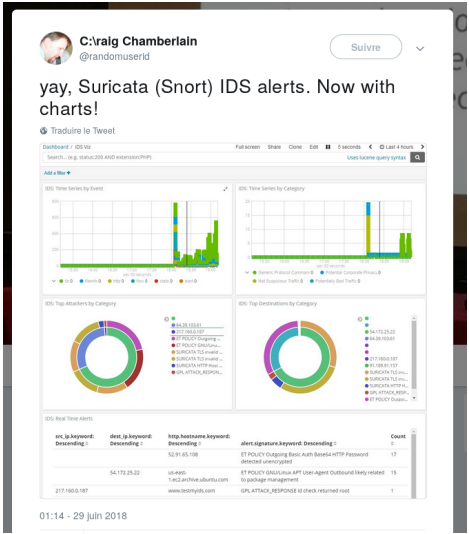



Figure: Summary of talk objectives

OISF

OISF

# What it is not ?



# A signature based IDS

## From individual datagram to detection

- Get packet per packet
- Reconstruct to application layer
- Run detection engine

## Key points

- GPLv2
- Owned by OISF foundation
- 10 years old
- Scalability via multithreading
- Written in C and Rust

## Example signature

```
alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"ET CURRENT_EVENTS [PTsecurity] Grandsoft EK Payload"; flow:established,to_client; content:"200"; http_stat_code; content:"|96 08 FA EC DE C0 22 84 66 58 4A BC 2E|"; http_server_body; fast_pattern; metadata: former_category CURRENT_EVENTS; reference:url,www.malware-traffic-analysis.net/2018/03/15/index3.html; classtype:trojan-activity; sid:2025437; rev:2; metadata:affected_product Windows_XP_Vista_7_8_10_Server_32_64_Bit, attack_target Client_Endpoint, deployment Perimeter, signature_severity Major, created_at 2018_03_21, malware_family GrandSoft_EK, updated_at 2018_03_21;)
```

# Suricata NSM features

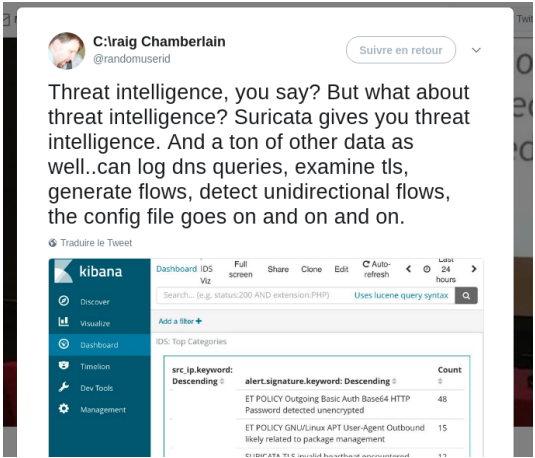
## Supported protocols

- Protocol analysis: http, ftp, smtp, tls, ssh smb, dcerpc, dns, nfs, ntp, ftp-data, tftp, ikev2, krb5, dhcp
- Protocol recognition: imap, msn

## Log example


```
  "timestamp": "2018-06-30T10:07:40.738055+0200",
  "flow_id": 210480145384532,
  "in_iface": "wlp3s0",
  "event_type": "tls",
  "src_ip": "                                        ",
  "src_port": 57784,
  "dest_ip": "2607:5300:0060:5958:0000:0000:0000:0000",
  "dest_port": 443,
  "proto": "TCP",
  "tls": {
    "subject": "CN=www.stamus-networks.com",
    "issuerdn": "C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3",
    "serial": "03:84:4B:EA:4A:17:3D:45:30:74:5B:8C:DD:5A:4B:CC:0C:0C",
    "fingerprint": "4c:84:61:7c:2b:74:2a:c6:5e:47:af:57:02:d4:9e:25:3a:67:ce:b8",
    "sni": "www.stamus-networks.com",
    "version": "TLS 1.2",
    "notbefore": "2018-05-16T09:43:01",
    "notafter": "2018-08-14T09:43:01",
    "ja3": {
      "hash": "a2d9e37641f5ba558913675a08401356",
      "string": "771,49196-49287-52393-49325-49162-49188-49267-49195-49286-49324-49161-49187-49266-49160-49200-49291-52392-49172-49192-492
71-49199-49290-49171-49191-49270-49170-157-49275-49309-53-61-132-192-156-49274-49308-47-60-65-186-10-159-49277-52394-49311-57-107-136-196-
158-49276-49310-51-103-69-190-22,5-0-65281-35-10-11-13-21,23-24-25-21-19,0"
```

# What it is ? or how to please developers




https://twitter.com/randomuserid/status/1012705279098490880

# File related features

## File analysis

- Magic computation and in file data match
- Checksum computation and file extraction to disk
- Supported protocols: http, smtp, smb, ftp, nfs

## Fileinfo example


```
"proto": "TCP",
"http": {
  "hostname": "vcrvcr.3322.org",
  "url": "/ww/aa24.exe",
  "http_user_agent": "MyIE/1.0",
  "http_content_type": "application/octet-stream",
  "http_method": "GET",
  "protocol": "HTTP/1.1",
  "status": 200,
  "length": 24592
},
"app_proto": "http",
"fileinfo": {
  "filename": "/ww/aa24.exe",
  "magic": "PE32 executable (GUI) Intel 80386, for MS Windows, UPX compressed",
  "gaps": false,
  "state": "CLOSED",
  "sha1": "d7c8ff3971d256bede2a3ab97d72bcf7072f6fb6",
  "stored": false,
  "size": 24592
```

OISF

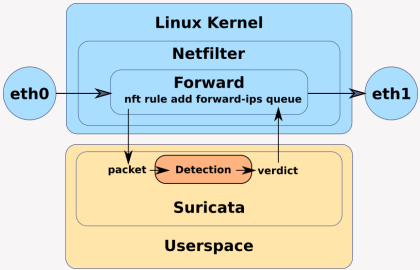
# Suricata live modes

## Intrusion Detection System

- AF_PACKET capture method under Linux
- Get raw packet from card
- Do complete analysis

## Intrusion Prevention System

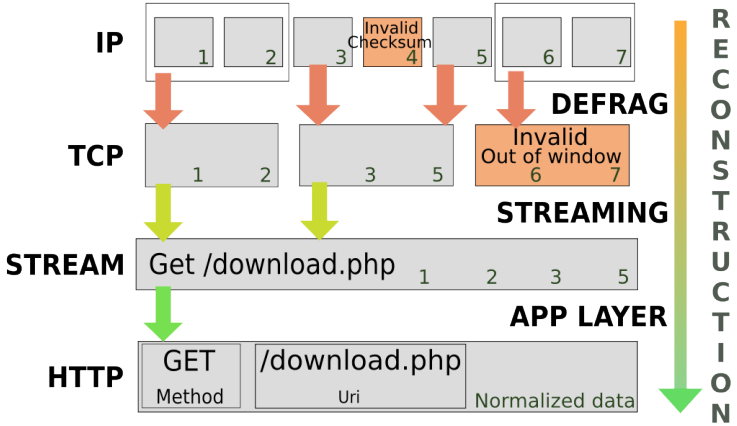
- Netfilter with NFQUEUE on Linux
- Kernel ask userspace for decision on packets



OISF

OISF

# Suricata reconstruction and normalization



OISF

# Impact of loosing packets

## Methodology

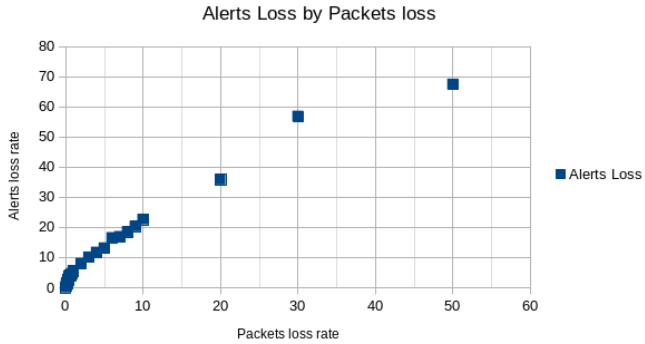
- Use a sample traffic
- Modify the pcap file to have specified random packet loss
- Do it 3 times par packet loss
- Get graph out of that

## Test data

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

OISF

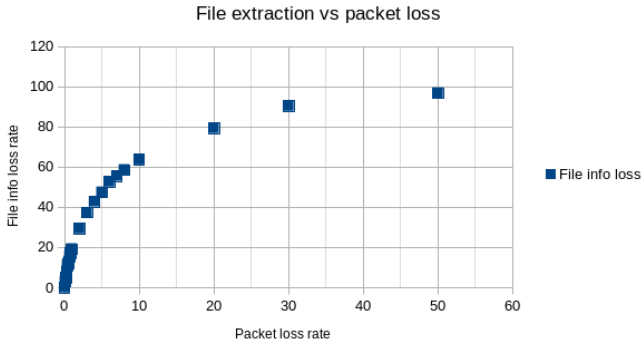
# Alert loss by packet loss




## Some numbers

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

# The case of file extraction

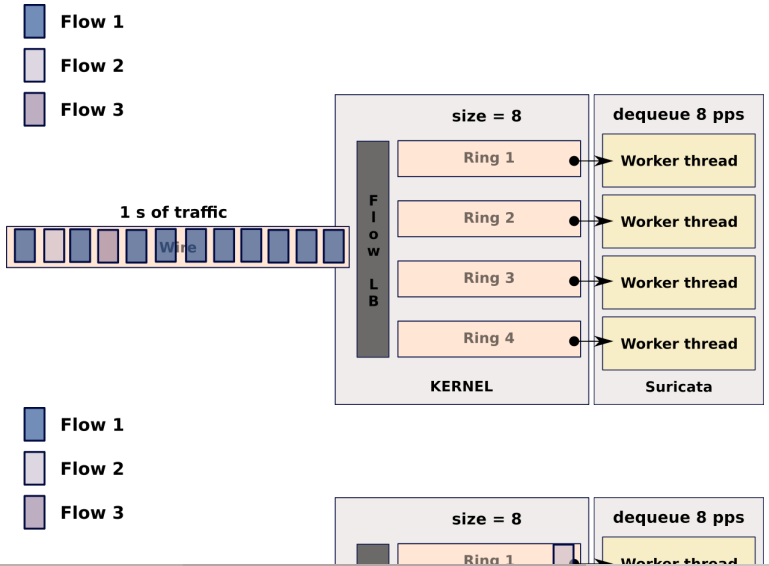



## Some numbers

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

OISF

# 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

OISF

OISF

# 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

OISF

OISF

OISF

# Introducing bypass

## Stop packet handling as soon as possible

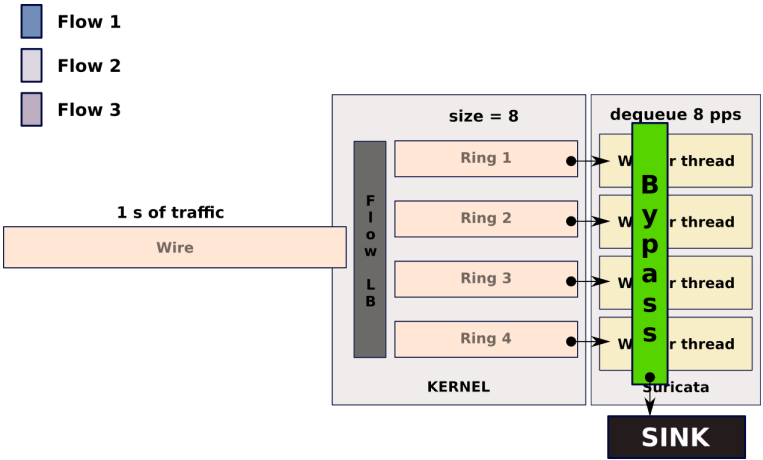
- Tag flow as bypassed
- Maintain table of bypassed flows
- Discard packet if part of a bypassed flow

## Bypass method

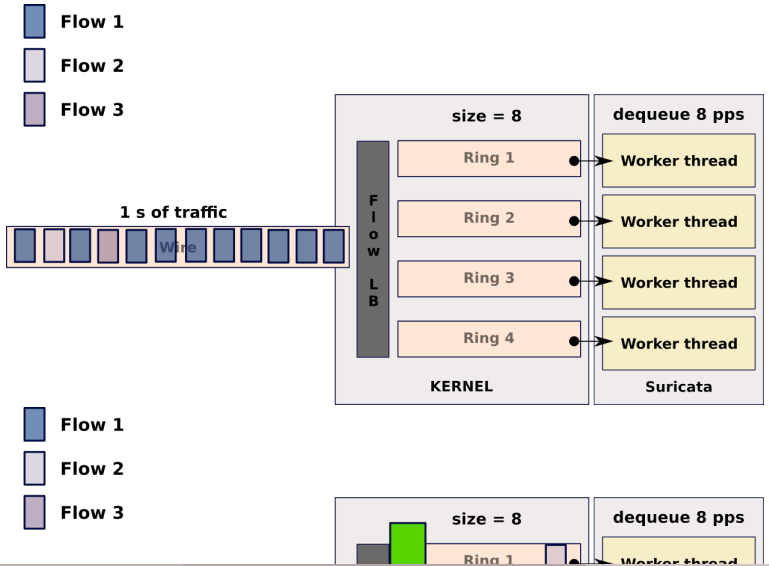
- Local bypass: Suricata discard packet after decoding
- Capture bypass: capture method maintain flow table and discard packets of bypassed flows

OISF

# Bypassing big flow: local bypass



# Bypassing big flow: capture bypass



# Implementation

## Suricata update

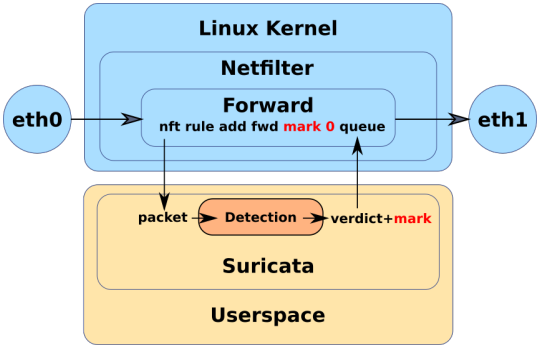
- Add callback function
- Capture method register itself and provide a callback
- Suricata calls callback when it wants to offload

## 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

OISF

# Suricata NFQ and bypass



OISF

# 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`

OISF

# 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;)
```

# And now AF_PACKET bypass

## 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!

OISF

# 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

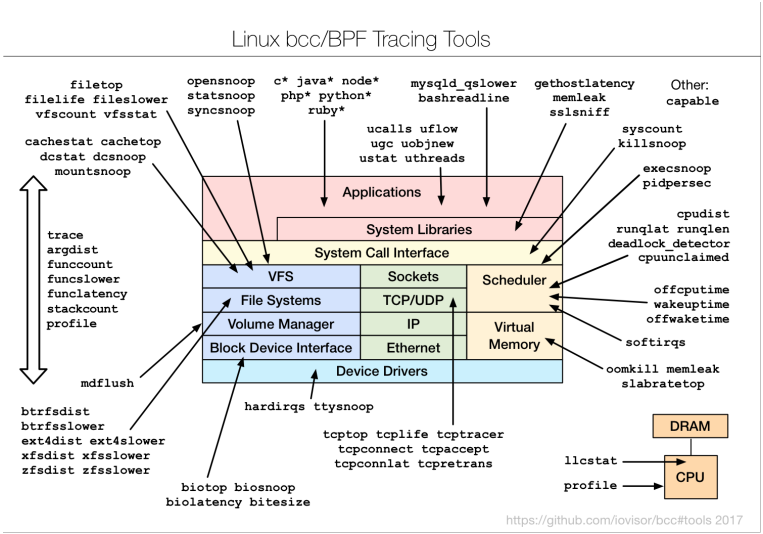
- Write C code
- Use eBPF LLVM backend (since LLVM 3.7)
- Use libbpf
  - Get ELF file
  - Extract and load section in kernel

## BCC: BPF Compiler collection

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

OISF

# BCC tracing tools



# eBPF applied to security

## Advantages

- Really extensible
- Kernel version independant when not intercepting functions
- Extract info all system stacks

## Host security monitoring at Netflix

Linux Monitoring at Scale with eBPF (Brendan Gregg & Alex Maestretti)
`https://youtu.be/44nV6Mj11uw`

OISF

# Example: BCC socket bind 1/2

## Detect network servers

- Get all bind call to detect services
- Output result to console

## A BCC script

- Python code
- eBPF code as C in a string

OISF

# Example: BCC socket bind 2/2

## Demo

- Start sobind
- Start a nc command to listen to port 2233

## Output

```
sudo python ./sobind
PID    COMM         PROTO  PORT  ADDR
9565   nc           TCPv4  2233  0.0.0.0
9572   nc           TCPv4  2233  127.0.0.2
```

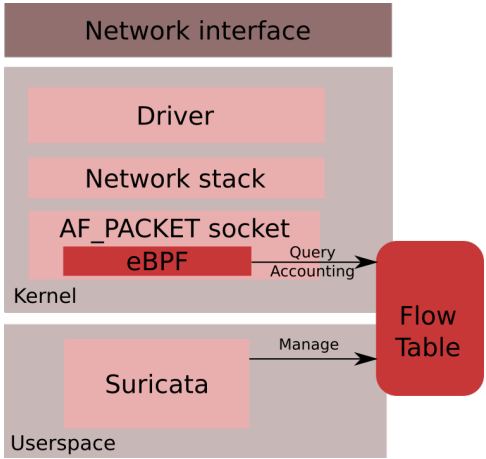
## Key features

- Direct hook in the system call
- No /proc scanning but polling of results
- Get it there:
  `https://gist.github.com/regit/1e591311fa3ba5cd0b8d73940348599a`

OISF

OISF

# Suricata eBPF bypass architecture



# Test methodology

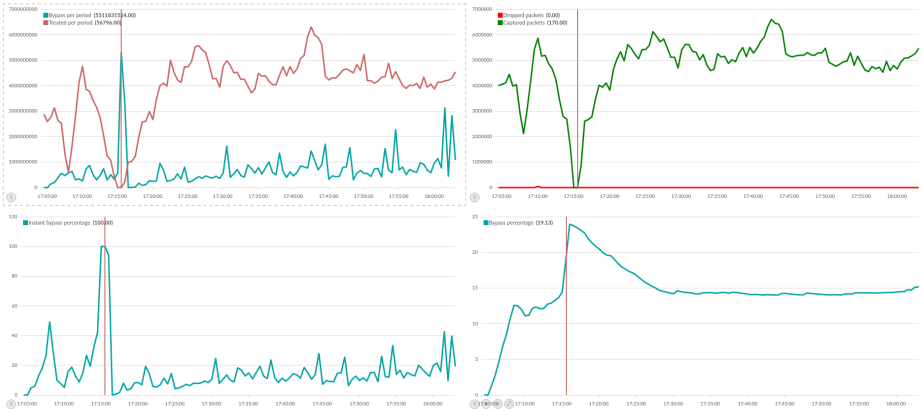
## Test setup

- Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz
- Intel Corporation 82599ES 10-Gigabit SFI/SFP+
- Live traffic:
  - Around 1Gbps to 2Gbps
  - Real users so not reproducible

## Tests

- One hour long run
- Different stream depth values
- Collected Suricata statistics counters (JSON export)
- Graphs done via Timelion (`https://www.elastic.co/blog/timelion-timeline`)

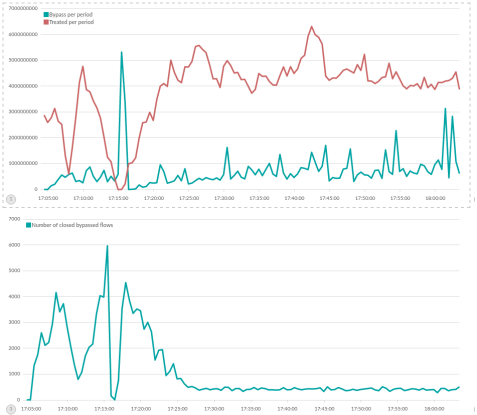
# Results: stream bypass at 512kb



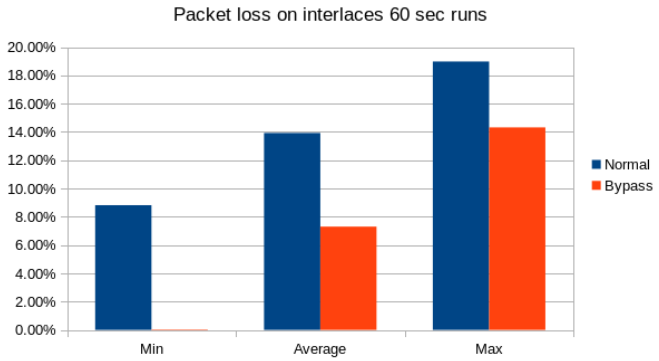
# A few words on graphics

## Tests at 512kb

- We have on big flow that kill the bandwidth
- Capture get almost null
- Even number of closed bypassed flows is low



# Results


Packet loss on interlaces 60 sec runs
20.00%
18.00%
16.00%
14.00%
12.00%
10.00%
8.00%
6.00%
4.00%
2.00%
0.00%
Min
Average
Max
Normal
Bypass

OISF

# A 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

OISF

# Implementation

## Similar to eBPF filter

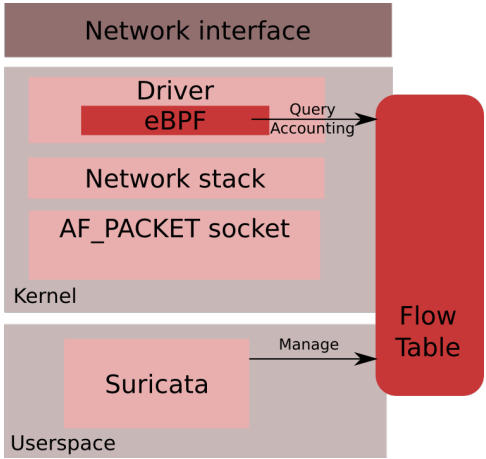
- Same logic for bypass
- Only verdict logic is different

## But annoying difference

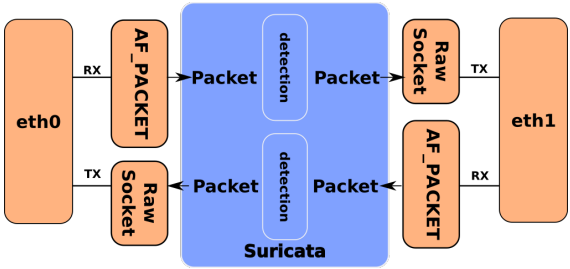
- eBPF code does the parsing
- Need to bind to an interface

OISF

# Suricata XDP architecture



# AF_PACKET IPS mode




## IPS and bypass

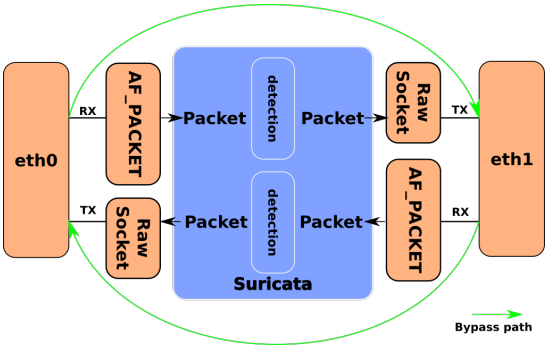
- Can't drop packet at bypass stage
- Need to forward from one iface to another

OISF

# XDP and IPS mode: bypass

## Use TX_REDIRECT

- Direct copy from interface to interface



# Results

## Direct NIC to NIC transfer

- Skip all kernel task
- Wire speed copy
- If eBPF code is fast enough

## Obtained performance

TODO: Ask OISF marketing for some fake numbers to show

OISF

OISF

# Conclusion

## Suricata, eBPF and XDP

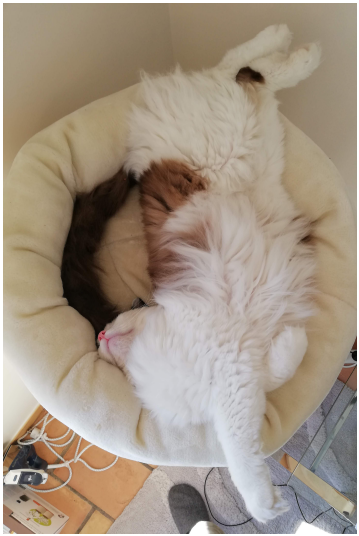
- Available in Suricata 4.1, need Linux 4.16
- Network card bypass for Netronome coming
- AF_XDP capture is now in Linux vanilla

## More information

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

OISF

# Questions ?



## Thanks to

- Jesper Dangaard Brouer
- Alexei Starovoitov
- Daniel Borkmann

## Contact me

- eleblond@oisf.net
- Twitter: @regiteric

## Want more fun ?

- Come to Suricata trainings: `https://suricata-ids.org/training/`
- Suricon: `https://suricon.net/`