Suricata Performance with a S like Security

É. Leblond

Stamus Networks

July. 03, 2018
Introduction
- Features
- Reconstruction work

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

Bypass
- Introducing bypass
- Bypass strategy

Hipster technologies to the rescue
- eBPF
- AF_PACKET bypass via eBPF
- XDP support

Conclusion
Introduction
- Features
  - Reconstruction work

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

Bypass
- Introducing bypass
- Bypass strategy

Hipster technologies to the rescue
- eBPF
- AF_PACKET bypass via eBPF
- XDP support

Conclusion
What it is not?
A signature based IDS

From individual traffic to detection
- Get packet per packet
- Reconstruct to application layer
- Run detection engine

Identity
- GPLv2
- owned by OISF (non for profit foundation)
- Scalability via multithreading
- Written in C and Rust

Example signature

```bash
```
Suricata (Bro) 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": "192.168.1.100",
"src_port": 57784,
"dest_ip": "206.250.240.239:50000:0000:0000:0000:0000:0000:0000",
"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",
"sni": "www.stamus-networks.com",
"version": "TLS 1.2",
"notbefore": "2018-05-16T09:43:01",
"notafter": "2018-08-14T09:43:01",
"ja3": { "hash": "a2d9e37641f5ba558913675a08481356",
} 
```
What is it? or how to please developers

Threat intelligence, you say? But what about threat intelligence? Suricata gives you threat intelligence. And a ton of other data as well... can log DNS queries, examine TLS, generate flows, detect unidirectional flows, the config file goes on and on and on.

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

```json
"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,
    "tx_id": 23
}
```
1 Introduction
   - Features
   - Reconstruction work

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

3 Bypass
   - Introducing bypass
   - Bypass strategy

4 Hipster technologies to the rescue
   - eBPF
   - AF_PACKET bypass via eBPF
   - XDP support

5 Conclusion
Suricata reconstruction and normalization

- **IP**: 1 2 3 4 5 6 7
- **TCP**: 1 2 3 4 5
- **STREAM**: Get /download.php
- **HTTP**: GET Method /download.php Uri
- **STREAMING**: DEFrag
- **APP LAYER**: Normalized data
1. Introduction
   - Features
   - Reconstruction work

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

3. Bypass
   - Introducing bypass
   - Bypass strategy

4. Hipster technologies to the rescue
   - eBPF
   - AF_PACKET bypass via eBPF
   - XDP support

5. Conclusion
Introduction
- Features
- Reconstruction work

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

Bypass
- Introducing bypass
- Bypass strategy

Hipster technologies to the rescue
- eBPF
- AF_PACKET bypass via eBPF
- XDP support

Conclusion
Impact of losing packets

Methodology
- Use a sample traffic
- Modify the pcap file to have specified random packet loss
- Do it 3 times per 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. Introduction
   - Features
   - Reconstruction work

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

3. Bypass
   - Introducing bypass
   - Bypass strategy

4. Hipster technologies to the rescue
   - eBPF
   - AF_PACKET bypass via eBPF
   - XDP support

5. Conclusion
The elephant flow problem (1/2)
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
Introduction
- Features
- Reconstruction work

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

Bypass
- Introducing bypass
- Bypass strategy

Hipster technologies to the rescue
- eBPF
- AF_PACKET bypass via eBPF
- XDP support

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
Introduction
- Features
- Reconstruction work

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

Bypass
- Introducing bypass
- Bypass strategy

Hipster technologies to the rescue
- eBPF
- AF_PACKET bypass via eBPF
- XDP support

Conclusion
Introduction
- Features
- Reconstruction work

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

Bypass
- Introducing bypass
- Bypass strategy

Hipster technologies to the rescue
- eBPF
- AF_PACKET bypass via eBPF
- XDP support

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
Bypassing big flow: local bypass

Flow 1
Flow 2
Flow 3

1 s of traffic
Wire

size = 8
Ring 1
Ring 2
Ring 3
Ring 4

KERNEL

dequeue 8 pps
Worker thread
Worker thread
Worker thread
Worker thread

Bypass

Suricata

SINK
Bypassing big flow: capture bypass

Flow 1
Flow 2
Flow 3

1 s of traffic

BYPASS

size = 8
Ring 1
Ring 2
Ring 3
Ring 4

dequeue 8 pps
Worker thread
Worker thread
Worker thread
Worker thread

Suricata
Suricata update

- Add callback function
- Capture method register itself and provide a callback
- Suricata calls callback when it wants to offload
**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
Introduction
- Features
- Reconstruction work

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

Bypass
- Introducing bypass
- Bypass strategy

Hipster technologies to the rescue
- eBPF
- AF_PACKET bypass via eBPF
- XDP support

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
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

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

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

Bypass
- Introducing bypass
- Bypass strategy

Hipster technologies to the rescue
- eBPF
- AF_PACKET bypass via eBPF
- XDP support

Conclusion
Introduction
- Features
- Reconstruction work

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

Bypass
- Introducing bypass
- Bypass strategy

Hipster technologies to the rescue
- eBPF
- AF_PACKET bypass via eBPF
- XDP support

Conclusion
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
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
Introduction

- Features
- Reconstruction work

Problem

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

Bypass

- Introducing bypass
- Bypass strategy

Hipster technologies to the rescue

- eBPF
- AF_PACKET bypass via eBPF
- XDP support

Conclusion
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
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!
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 interfaces 60 sec runs

- Normal
- Bypass

Min: 8.00%
Average: 14.00%
Max: 20.00%
1 Introduction
   • Features
   • Reconstruction work

2 Problem
   • Packet loss impact
   • Elephant flow
   • Work less to get more

3 Bypass
   • Introducing bypass
   • Bypass strategy

4 Hipster technologies to the rescue
   • eBPF
   • AF_PACKET bypass via eBPF
   • XDP support

5 Conclusion
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
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
TODO: Ask OISF marketing for some fake numbers to show
Introduction
- Features
- Reconstruction work

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

Bypass
- Introducing bypass
- Bypass strategy

Hipster technologies to the rescue
- eBPF
- AF_PACKET bypass via eBPF
- XDP support

Conclusion
Conclusion

Suricata, eBPF and XDP

- A fresh but interesting method
- 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/
Questions?

Thanks to
- Jesper Dangaard Brouer
- Alexei Storovoitov
- Daniel Borkmann

Contact me
- el@stamus-networks.com
- Twitter: @regiteric

Want more fun?
- Come to Suricata and SELKS workshop!
- Suricon:
  https://suricon.net/