The adventures of a Suricate in eBPF land

É. Leblond
Stamus Networks
Oct. 6, 2016
Introduction to Suricata

- What’s this?
- A few words on performance

Suricata meets eBPF

- AF_PACKET
- Interest of bypass

eBPF technology

eBPF cluster or the start of the travel

eBPF bypass or lost in translation

Some results

Conclusion
What is Suricata

- IDS and IPS engine
- Get it here: http://www.suricata-ids.org
- Open Source (GPLv2)
- Initially publicly funded, now funded by consortium members
- Run by Open Information Security Foundation (OISF)
- More information about OISF at http://www.openinfosecfoundation.org/
Suricata Features

- High performance, scalable through multi threading
- Advanced Protocol handling
  - Protocol recognition
  - Protocol analysis: field extraction, filtering keywords
  - Transaction logging in extensible JSON format
- File identification, extraction, on the fly MD5 calculation
  - HTTP
  - SMTP
- TLS handshake analysis, detect/prevent things like Diginotar
- Lua scripting for detection
- Hardware acceleration support:
  - Endace
  - Napatech,
  - CUDA
  - PF_RING
Signature example: Chat facebook

```
alert http $HOME_NET any -> $EXTERNAL_NET any \
  (msg:"ET CHAT Facebook Chat about netdev"; \
  flow:established,to_server; content:"POST"; http_method; \
  content:"/ajax/chat/send.php"; http_uri; content:"facebook.com"; http_host; \
  content:"netdev"; http_client_body; \
  reference:url,www.emergingthreats.net/cgi-bin/cvsweb.cgi/sigs/POLICY/POLICY_Facebook_Chat; \
  sid:2010784; rev:4; \
  )
```

This signature tests:

- The HTTP method: **POST**
- The page: `/ajax/chat/send.php`
- The domain: `facebook.com`
- The body content: `netdev`
No passthrough

**All signatures are inspected**
- Different from a firewall
- More than 15000 signatures in standard rulesets

**Optimization on detection engine**
- Tree pre filtering approach to limit the set of signatures to test
- Multi pattern matching on some buffers
CPU intensive

Tasks: 43, 31 thr; 11 running
Load average: 7.40 7.24 7.32
Uptime: 82 days, 23:13:26
Perf top

Samples: 691K of event 'cycles', Event count (approx.): 256764876818

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>64.14%</td>
<td>suricata</td>
<td>[.] SCACSearch</td>
</tr>
<tr>
<td>3.20%</td>
<td>suricata</td>
<td>[.] BoyerMoore</td>
</tr>
<tr>
<td>1.16%</td>
<td>suricata</td>
<td>[.] SigMatchSignatures</td>
</tr>
<tr>
<td>0.90%</td>
<td>libc-2.19.so</td>
<td>[.] memset</td>
</tr>
<tr>
<td>0.87%</td>
<td>[kernel]</td>
<td>[k] ixgbe_clean_rx_irq</td>
</tr>
<tr>
<td>0.75%</td>
<td>suricata</td>
<td>[.] IPOnlyMatchPacket</td>
</tr>
<tr>
<td>0.68%</td>
<td>libpthread-2.19.so</td>
<td>[.] pthread_mutex_unlock</td>
</tr>
<tr>
<td>0.64%</td>
<td>[kernel]</td>
<td>[k] __netif_receive_skb_core</td>
</tr>
<tr>
<td>0.62%</td>
<td>libpthread-2.19.so</td>
<td>[.] pthread_mutex_lock</td>
</tr>
<tr>
<td>0.62%</td>
<td>suricata</td>
<td>[.] AFPReadFromRing</td>
</tr>
<tr>
<td>0.61%</td>
<td>[kernel]</td>
<td>[k] irq_entries_start</td>
</tr>
<tr>
<td>0.58%</td>
<td>[kernel]</td>
<td>[k] tpacket_rcv</td>
</tr>
<tr>
<td>0.55%</td>
<td>libc-2.19.so</td>
<td>[.] __memcmp_sse4_1</td>
</tr>
<tr>
<td>0.52%</td>
<td>[kernel]</td>
<td>[k] memcpy</td>
</tr>
<tr>
<td>0.42%</td>
<td>[kernel]</td>
<td>[k] ixgbe_poll</td>
</tr>
<tr>
<td>0.42%</td>
<td>[kernel]</td>
<td>[k] menu_select</td>
</tr>
<tr>
<td>0.40%</td>
<td>suricata</td>
<td>[.] StreamTcpPacket</td>
</tr>
<tr>
<td>0.36%</td>
<td>[kernel]</td>
<td>[k] native_write_msr_safe</td>
</tr>
<tr>
<td>0.35%</td>
<td>[kernel]</td>
<td>[k] packet_lookup_frame.isra.56</td>
</tr>
</tbody>
</table>
Scalability

- Bandwidth per core is limited
  - From 150Mb/s
  - To 500Mb/s

- Scaling
  - Using RSS
  - Splitting load on workers
Introduction to Suricata

- What’s this?
- A few words on performance

Suricata meets eBPF

- AF_PACKET
- Interest of bypass

eBPF technology

eBPF cluster or the start of the travel

eBPF bypass or lost in translation

Some results

Conclusion
Linux raw socket

- Raw packet capture method
- Socket based or mmap based
Linux raw socket

- Raw packet capture method
- Socket based or mmap based

Fanout mode

- Load balancing over multiple sockets
- Multiple load balancing functions
  - Flow based
  - CPU based
  - RSS based
  - eBPF based
Suricata workers mode

The adventures of a Suricate in eBPF land

Oct. 6, 2016
Stream reconstruction
- Using packets sniffed from network
- To reconstruct TCP stream as seen by remote application

Non symmetrical hash break
- Out of order packets

Effect of non symmetrical hash

Order in stream engine
Broken symmetry

**History**

- T. Herbert introduce asymmetrical hash function in flow
  - Kernel 4.2
- Users did start to complain
- And our quest did begin
- Fixed in 4.6 and pushed to stable by David S. Miller
Broken symmetry

History

- T. Herbert introduce asymmetrical hash function in flow
  - Kernel 4.2
- Users did start to complain
- And our quest did begin
- Fixed in 4.6 and pushed to stable by David S. Miller

Intel NIC RSS hash

- XL510 hash is not symmetrical
- XL710 could be symmetrical
  - Hardware is capable
  - Driver does not allow it
  - Patch proposed by Victor Julien
Userspace to the rescue

- Program your own hash function in userspace
- Available since Linux 4.3
- Developed by Willem de Bruijn
- Using eBPF infrastructure by Alexei Storovoitov

eBPF cinematic

- Syscall to load the BPF code in kernel
- Setsockopt to set returned fd as cluster BPF
The big flow problem

Ring buffer overrun

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

Bypassing big flow

- Limiting treatment time at maximum
- Stopping it earlier as possible
  - local bypass: Suricata limit handling
  - capture bypass: interaction with lower layer
Stream depth

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

- Suricata reassemble TCP sessions till `stream.reassembly.depth` bytes.
- Stream is not analyzed once limit is reached
Introducing bypass

Principle

- No need to get packet from kernel after stream depth is reached
- If there is
  - no file store
  - or other operation

Usage

Set `stream.bypass` option to `yes` in Suricata config file to 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

```
cmd.alert http any any -> any any (content: "netdevconf.org"; \\n  http_host; bypass; sid:6666; rev:1;)
```
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

Coded for NFQ
- 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
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
- nftables is too late even in ingress
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
- `nftables` is too late even in ingress

### eBPF filter using maps
- eBPF introduce maps
- Different data structures
  - Hash, array, ...
  - Update and fetch from userspace
- Looks good!
Introduction to Suricata
- What’s this?
- A few words on performance

Suricata meets eBPF
- AF_PACKET
- Interest of bypass

eBPF technology

eBPF cluster or the start of the travel

eBPF bypass or lost in translation

Some results

Conclusion
### Handling code
- Need to generate code
- Load code
- Address code from Suricata

### Interact with code
- Add elements in hash table
- Query elements
- Delete elements
From C file to eBPF code

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

- Instrument eBPF filter
- Multi language
  - Python
  - Lua
  - C++
- Transparent handling of kernel interaction

Cinematic

- eBPF C code is a side file or integrated into code
- C code is dynamically built when script is started
- It is injected to kernel
- Post processing is done
Introduction to Suricata
- What’s this?
- A few words on performance

Suricata meets eBPF
- AF_PACKET
- Interest of bypass

eBPF technology

eBPF cluster or the start of the travel

eBPF bypass or lost in translation

Some results

Conclusion
Importing mechanism

- Syscall to load the object inside kernel
- A file descriptor is returned
- It can be used by setsockopt to define the cluster using provided fd
Initial version

- LLVM backend
- Using libelf to load object
Suricata eBPF cluster

Initial version
- LLVM backend
- Using libelf to load object

Time saver
- Debug message from kernel eBPF code
- bpt_trace_printk() function
- `cat /sys/kernel/tracing/tracing/trace`
Introduction to Suricata
- What’s this?
- A few words on performance

Suricata meets eBPF
- AF_PACKET
- Interest of bypass

eBPF technology

eBPF cluster or the start of the travel

eBPF bypass or lost in translation

Some results

Conclusion
AF_PACKET bypass

Logic is the same
- Using eBPF filter this time
- Syscall to load eBPF
- Linking via setsockopt
- Need to use a eBPF map of type hash

Here comes the map
- Map is used by kernel and userspace
- eBPF file can’t contain absolute reference
- Maps must be created by userspace
- Relocation must be done in ELF file
AF_PACKET bypass

Logic is the same

- Using eBPF filter this time
- Syscall to load eBPF
- Linking via setsockopt
- Need to use a eBPF map of type hash

Here comes the map

- Map is used by kernel and userspace
- eBPF file can’t contain absolute reference
- Maps must be created by userspace
- Relocation must be done in ELF file

Game Over
Switch to libbpf

Library from tools/lib/bpf

- Provide high level function to load eBPF elf file
- Create maps for user
- Do the relocation

Sample usage

```c
struct bpf_object *bpfobj = bpf_object__open(path);
bpf_object__load(bpfobj);
pfd = bpf_program__fd(bpfprog);
/* store the map in our array */
bpf_map__for_each(map, bpfobj) {
    map_array[last].fd = bpf_map__fd(map);
    map_array[last].name = strdup(bpf_map__name(map));
    last++;
}
```
Libbpf implementation

libbpf is work in progress
- Not network ready
- Missing a few filter types
- Missing functions to interact

Patchset in progress
- Cleaning of initially proposed code
- Adding missing features
Kernel code and exchange structure

```c
struct pair {
    uint64_t time;
    uint64_t packets;
    uint64_t bytes;
};

struct bpf_map_def SEC("maps") flow_table_v4 = {
    .type = BPF_MAP_TYPE_HASH,
    .key_size = sizeof(struct flowv4_keys),
    .value_size = sizeof(struct pair),
    .max_entries = 32768,
};

value = bpf_map_lookup_elem(& flow_table_v4, &tuple);
if (value) {
    __sync_fetch_and_add(&value->packets, 1);
    __sync_fetch_and_add(&value->bytes, skb->len);
    value->time = bpf_ktime_get_ns();
    return 0;
}
return -1;
```
Sharing data

- Data is updated with stats
- Getting last flow activity time allow Suricata to handle timeout
struct flowv4_keys {
  __be32 src;
  __be32 dst;
  union {
    __be32 ports;
    __be16 port16[2];
  };
  __u32 ip_proto;
};

while (bpf_map__get_next_key(mapfd, &key, &next_key) == 0) {
  bpf_map__lookup_elem(mapfd, &key, &value);
  clock_gettime(CLOCK_MONOTONIC, &curtime);
  if (curtime.tv_sec * 1000000000 - value.time > BYPASSED_FLOW_TIMEOUT) {
    flowstats->count++;
    flowstats->packets += value.packets;
    flowstats->bytes += value.bytes;
  }
  bpf_map__delete_elem(fd, key);
}

key = next_key;
Got to be ready

This is KAME land: http://www.kame.net/
IPv6 is the same as IPv4

- Same algorithm
- Second hash table using IPv6 tuple
IPv6 bypass

IPv6 is the same as IPv4
- Same algorithm
- Second hash table using IPv6 tuple

Really?
- Parsing is a bit different due to next header
- IPv6 hash table is failing to load in kernel
Let's call a friend

The exercise of adding the egress counterpart and IPv6 support is left to the reader

Daniel Borkmann in tc_bpf.8
IPv6 bypass

Two hash tables
- A bug in libbpf
- Invalid offset computation of map definition
- Fixed by mimic tc_bpf.c code (thanks Daniel Borkmann)

IPv6 parsing
- For now, sending weird packets to userspace
Introduction to Suricata
- What’s this?
- A few words on performance

Suricata meets eBPF
- AF_PACKET
- Interest of bypass

eBPF technology

eBPF cluster or the start of the travel

eBPF bypass or lost in translation

Some results

Conclusion
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](https://www.elastic.co/blog/timelion-timeline))
Results: bypass at 1mb
Results: bypass at 512kb
A few words on graphics

Tests at 1mb
- Mark show some really high rate bypass
- Potentially a big high speed flow

Tests at 512kb
- We have on big flow that kill the bandwidth
- Capture get almost null
- Even number of closed bypassed flows is low
AF_PACKET bypass and your CPU is peaceful
Introduction to Suricata
- What’s this?
- A few words on performance

Suricata meets eBPF
- AF_PACKET
- Interest of bypass

eBPF technology

eBPF cluster or the start of the travel

eBPF bypass or lost in translation

Some results

Conclusion
Conclusion

Suricata and eBPF

- A fresh but interesting method
- Bypass looks promising
- More tests to come

More information

- **Stamus Networks**: [https://www.stamus-networks.com/](https://www.stamus-networks.com/)
- **Suricata eBPF code**: [https://github.com/regit/suricata/tree/ebpf-3.8](https://github.com/regit/suricata/tree/ebpf-3.8)
- **Libbpf update**: [https://github.com/regit/linux/tree/libbpf-network-v5](https://github.com/regit/linux/tree/libbpf-network-v5)
Questions?

Thanks to
- Alexei Storovoitov
- Daniel Borkmann
- David S. Miller

Contact me
- Mail: eleblond@stamus-networks.com
- Twitter: @regiteric

More information
- Suricata eBPF code: https://github.com/regit/suricata/tree/ebpf-3.8