The adventures of a Suricate in eBPF land

É. Leblond

Stamus Networks

Oct. 6, 2016

STAMVS NETWORKS

1 Introduction to Suricata
- What's this ?
- A few words on performance

2 Suricata meets eBPF
- AF_PACKET
- Interest of bypass

3 eBPF technology

4 eBPF cluster or the start of the travel

5 eBPF bypass or lost in translation

6 Some results

7 Conclusion

STAMVS NETWORKS

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/

STAMVS NETWORKS

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

STAMVS NETWORKS

A typical signature example

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*

STAMVS NETWORKS

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

STAMVS NETWORKS

CPU intensive

```
  1  [|||||||||||||100.0%]    5  [||||||        26.5%]     9  [|||||||||||||88.7%]    13 [|||||||      35.3%]
  2  [|||||||||||||100.0%]    6  [||||          18.8%]     10 [||||||||||||  62.9%]   14 [|||||        22.4%]
  3  [||||          19.7%]    7  [||||||        27.2%]     11 [||||||||||||  58.2%]   15 [||||         14.9%]
  4  [||||||||      35.6%]    8  [||||||||      33.8%]     12 [|||||||       30.6%]   16 [|||||        23.8%]
  Mem[||||||||||||||||||||||||||||||||||25891/64403MB]     Tasks: 43, 31 thr; 11 running
  Swp[|                                    50/33377MB]     Load average: 7.40 7.24 7.32
                                                           Uptime: 82 days, 23:13:26

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
13679 root       20   0 34.6G 22.4G 5211M S 710. 35.7 69h08:07 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13741 root       18  -2 34.6G 22.4G 5211M R 102. 35.7  6h32:22 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13749 root       18  -2 34.6G 22.4G 5211M R 90.9 35.7  4h02:18 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13758 root       18  -2 34.6G 22.4G 5211M R 70.7 35.7 10h00:47 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13750 root       18  -2 34.6G 22.4G 5211M R 63.3 35.7  3h40:08 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13751 root       18  -2 34.6G 22.4G 5211M S 57.9 35.7  3h20:58 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13744 root       18  -2 34.6G 22.4G 5211M R 35.7 35.7  3h22:16 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13753 root       18  -2 34.6G 22.4G 5211M S 33.7 35.7  3h14:37 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13748 root       18  -2 34.6G 22.4G 5211M S 33.0 35.7  3h11:43 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13742 root       18  -2 34.6G 22.4G 5211M R 31.7 35.7  3h37:38 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13752 root       18  -2 34.6G 22.4G 5211M S 29.6 35.7  3h44:17 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13756 root       18  -2 34.6G 22.4G 5211M R 25.6 35.7  3h33:02 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13747 root       18  -2 34.6G 22.4G 5211M S 25.6 35.7  3h10:13 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13745 root       18  -2 34.6G 22.4G 5211M S 24.9 35.7  3h18:05 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13754 root       18  -2 34.6G 22.4G 5211M R 22.2 35.7  3h15:22 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13746 root       18  -2 34.6G 22.4G 5211M R 20.9 35.7  3h19:41 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13755 root       18  -2 34.6G 22.4G 5211M S 18.9 35.7  3h21:57 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13743 root       18  -2 34.6G 22.4G 5211M R 18.9 35.7  3h12:16 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13760 root       22   2 34.6G 22.4G 5211M S  2.7 35.7 31:04.81 /usr/local/bin/suricata -c /etc/suricata/regit-ya
13761 root       22   2 34.6G 22.4G 5211M S  2.7 35.7 27:38.31 /usr/local/bin/suricata -c /etc/suricata/regit-ya
```

STAMVS NETWORKS

Perf top

```
Samples: 691K of event 'cycles', Event count (approx.): 256764876818
Overhead  Shared Object              Symbol
  64.14%  suricata                   [.] SCACSearch
   3.20%  suricata                   [.] BoyerMoore
   1.16%  suricata                   [.] SigMatchSignatures
   0.90%  libc-2.19.so               [.] memset
   0.87%  [kernel]                   [k] ixgbe_clean_rx_irq
   0.75%  suricata                   [.] IPOnlyMatchPacket
   0.68%  libpthread-2.19.so         [.] pthread_mutex_unlock
   0.64%  [kernel]                   [k] __netif_receive_skb_core
   0.62%  libpthread-2.19.so         [.] pthread_mutex_lock
   0.62%  suricata                   [.] AFPReadFromRing
   0.61%  [kernel]                   [k] irq_entries_start
   0.58%  [kernel]                   [k] tpacket_rcv
   0.55%  libc-2.19.so               [.] __memcmp_sse4_1
   0.52%  [kernel]                   [k] memcpy
   0.42%  [kernel]                   [k] ixgbe_poll
   0.42%  [kernel]                   [k] menu_select
   0.40%  suricata                   [.] StreamTcpPacket
   0.36%  [kernel]                   [k] native_write_msr_safe
   0.35%  [kernel]                   [k] packet_lookup_frame.isra.56
```

STAMVS NETWORKS

Scalability

- Bandwith per core is limited
  - From 150Mb/s
  - To 500Mb/s
- Scaling
  - Using RSS
  - Splitting load on workers

STAMVS NETWORKS

1. Introduction to Suricata
   - What's this ?
   - A few words on performance
2. Suricata meets eBPF
   - AF_PACKET
   - Interest of bypass
3. eBPF technology
4. eBPF cluster or the start of the travel
5. eBPF bypass or lost in translation
6. Some results
7. Conclusion

STAMVS NETWORKS

AF_PACKET

Linux raw socket

- Raw packet capture method
- Socket based or mmap based

STAMVS NETWORKS

AF_PACKET

Linux raw socket

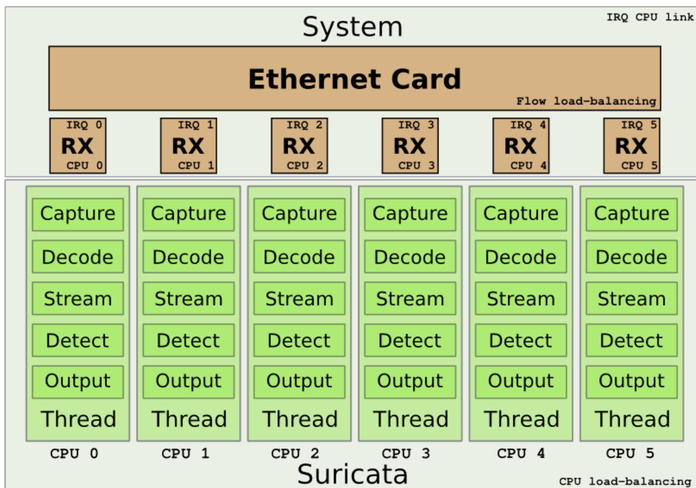
- Raw packet capture method
- Socket based or mmap based

Fanout mode

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

STAMVS NETWORKS

Suricata workers mode



STAMVS NETWORKS

Load balancing and hash symmetry

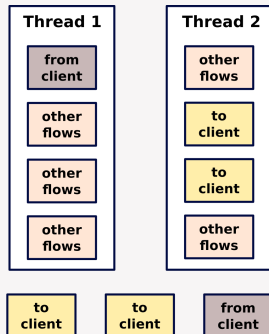
Stream reconstruction

- 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



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

STAMVS NETWORKS

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

NETWORKS

eBPF cluster

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

STAMVS NETWORKS

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

STAMVS NETWORKS

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

STAMVS NETWORKS

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

STAMVS NETWORKS

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

STAMVS NETWORKS

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

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

STAMVS NETWORKS

Implementation

Suricata update

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

STAMVS NETWORKS

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

STAMUS NETWORKS

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

STAMVS NETWORKS

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!

1. Introduction to Suricata
   - What's this ?
   - A few words on performance
2. Suricata meets eBPF
   - AF_PACKET
   - Interest of bypass
3. **eBPF technology**
4. eBPF cluster or the start of the travel
5. eBPF bypass or lost in translation
6. Some results
7. Conclusion

STAMVS NETWORKS

eBPF usage

Handling code

- Need to generate code
- Load code
- Address code from Suricata

Interact with code

- Add elements in hash table
- Query elements
- Delete elements

STAMVS NETWORKS

LLVM backend

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

STAMVS NETWORKS

BCC: BPF Compiler Collection

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

1. Introduction to Suricata
   - What's this ?
   - A few words on performance
2. Suricata meets eBPF
   - AF_PACKET
   - Interest of bypass
3. eBPF technology
4. eBPF cluster or the start of the travel
5. eBPF bypass or lost in translation
6. Some results
7. Conclusion

STAMVS NETWORKS

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

STAMVS NETWORKS

Suricata eBPF cluster

Initial version

- LLVM backend
- Using libelf to load object

STAMVS NETWORKS

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/trace`

STAMUS NETWORKS

1. Introduction to Suricata
   - What's this ?
   - A few words on performance
2. Suricata meets eBPF
   - AF_PACKET
   - Interest of bypass
3. eBPF technology
4. eBPF cluster or the start of the travel
5. **eBPF bypass or lost in translation**
6. Some results
7. Conclusion

STAMVS NETWORKS

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

STAMVS NETWORKS

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

STAMVS NETWORKS

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

```
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

STAMVS NETWORKS

Kernel code and exchange structure

```
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

STAMVS NETWORKS

Userspace code

```
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;
}
```

NETWORKS

Japan and IPv6

Got to be ready

- This is KAME land: `http://www.kame.net/`

STAMUS NETWORKS

IPv6 bypass

IPv6 is the same as IPv4

- Same algorithm
- Second hash table using IPv6 tuple

STAMVS NETWORKS

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

STAMVS NETWORKS

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

STAMVS NETWORKS

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

STAMVS NETWORKS

1. Introduction to Suricata
   - What's this ?
   - A few words on performance
2. Suricata meets eBPF
   - AF_PACKET
   - Interest of bypass
3. eBPF technology
4. eBPF cluster or the start of the travel
5. eBPF bypass or lost in translation
6. Some results
7. Conclusion

STAMVS NETWORKS

Test methodology

Test setup

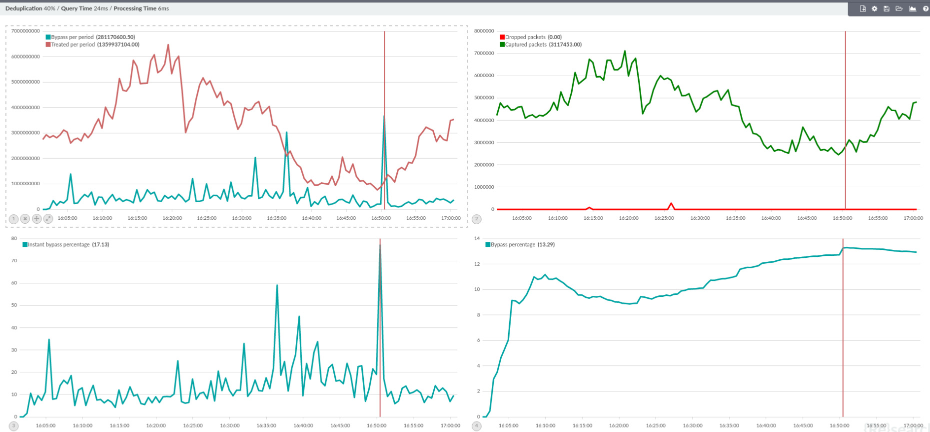
- Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz
- Intel Corporation 82599ES 10-Gigabit SFI/SFP+
- Live traffic:
  - Around 1Gbps to 2Gbps
  - Real users so not reproducible

Tests

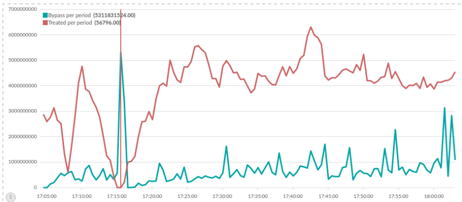
- One hour long run
- Different stream depth values
- Collected Suricata statistics counters (JSON export)
- Graphs done via Timelion (`https://www.elastic.co/blog/timelion-timeline`)

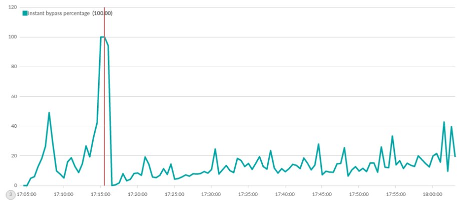
NETWORKS

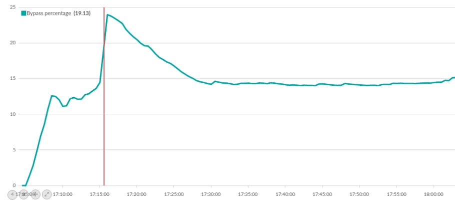
Results: bypass at 1mb



Results: bypass at 512kb







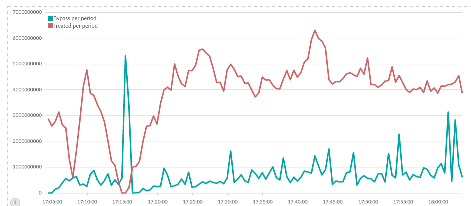
A few words on graphics

Tests at 1mb

- Mark show some really high rate bypass
- Potentialy 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



STAMUS NETWORKS

AF_PACKET bypass and your CPU is peaceful

STAMVS NETWORKS

1. Introduction to Suricata
   - What's this ?
   - A few words on performance
2. Suricata meets eBPF
   - AF_PACKET
   - Interest of bypass
3. eBPF technology
4. eBPF cluster or the start of the travel
5. eBPF bypass or lost in translation
6. Some results
7. Conclusion

STAMUS NETWORKS

Conclusion

Suricata and eBPF

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

More information

- Suricata: `http://www.suricata-ids.org/`
- Suricon, Nov. 16, Washington DC: `http://www.suricon.net/`
- Stamus Networks: `https://www.stamus-networks.com/`
- Suricata eBPF code: `https://github.com/regit/suricata/tree/ebpf-3.8`
- Libbpf update: `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`