Merging packets with System Events using eBPF

Luca Deri <deri@ntop.org>, @lucaderi
Samuele Sabella <sabella@ntop.org>, @sabellasantamamie
About Us

• Luca: lecturer at the University of Pisa, CS Department, founder of the ntop project.
• Samuele: student at Unipi CS Department, junior engineer working at ntop.
• ntop develops open source network traffic monitoring applications. ntop (circa 1998) is the first app we released and it is a web-based network monitoring application.
• Today our products range from traffic monitoring, high-speed packet processing, deep-packet inspection (DPI), IDS/IPS acceleration, and DDoS Mitigation.
• See http://github.com/ntop/
What is Network Traffic Monitoring?

• The key objective behind network traffic monitoring is to ensure *availability and smooth operations on a computer network*. Network monitoring incorporates network sniffing and packet capturing techniques in monitoring a network. Network traffic monitoring generally requires reviewing each incoming and outgoing packet.

https://www.techopedia.com/definition/29977/network-traffic-monitoring
ntop Ecosystem (2009): Packets Everywhere
ntop Ecosystem (2019): Still Packets [1/2]
What’s Wrong with Packets?

• Nothing in general but…
  ◦ It is a paradigm good for monitoring network traffic from outside of systems on a passive way.
  ◦ Encryption is challenging DPI techniques (BTW ntop maintains an open source DPI toolkit called nDPI).
  ◦ Virtualisation techniques reduce visibility when monitoring network traffic as network manager are blind with respect to what happens inside systems.
  ◦ Developers need to handle fragmentation, flow reconstruction, packet loss/retransmissions… metrics that would be already available inside a system.
From Problem Statement to a Solution

• Enhance network visibility with system introspection.
• Handle virtualisation as first citizen and don’t be blind (yes we want to see containers interaction).
• Complete our monitoring journey and…
  ◦ System Events: processes, users, containers.
  ◦ Flows
  ◦ Packets
• …bind system events to network traffic for enabling continuous drill down: system events uncorrelated with network traffic are basically useless.
Early Experiments: Sysdig [1/3]

• ntop has been an early sysdig adopter adding in 2014 sysdig events support in PF_RING, ntopng, nProbe.
Early Experiments: Sysdig [2/3]
Early Experiments: Sysdig [3/3]

• Despite all our efforts, this activity has NOT been a success for many reasons:
  ◦ Too much CPU load (in average +10-20% CPU load) due to the design of sysdig (see later).
  ◦ People do not like to install agents on systems as this might create interferences with other installed apps.
  ◦ Sysdig requires a new kernel module that sometimes is not what sysadmins like as it might invalidate distro support.
  ◦ Containers were not so popular in 2014, and many people did not consider system visibility so important at that time.
How Sysdig Works

• As sysdig focuses on system calls for tracking a TCP connections we need to:
  ◦ Discard all non TCP related events (sockets are used for other activities on Linux such as Unix sockets)
  ◦ Track socket() and remember the socketId to process/thread
  ◦ Track connect() and accept() and remember the TCP peers/ports.
  ◦ Collect packets and bind each of them to a flow (i.e. this is packet capture again, using sysdig instead of libpcap).

• This explains the CPU load, complexity…
Welcome to eBPF

The following set of patches adds BPF support to trace filters.

Trace filters can be written in C and allow safe read-only access to any kernel data structure. Like systemtap but with safety guaranteed by kernel.

The user can do:
```
cat bpf_program > /sys/kernel/debug/tracing/.../filter
if tracing event is either static or dynamic via kprobe_events.
```

eBPF is great news for ntop as
- It gives the ability to avoid sending everything to user-space but perform in kernel computations and send metrics to user-space.
- We can track more than system calls (i.e. be notified when there is a transmission on a TCP connection without analyzing packets).
- It is part of modern Linux systems (i.e. no kernel module needed).
libebpfflow Overview [1/2]

```c
struct netInfo {
    __u16 sport;
    __u16 dport;
    __u8  proto;
    __u32 latency_usec;
};

struct taskInfo {
    __u32 pid; /* Process Id */
    __u32 tid; /* Thread Id */
    __u32 uid; /* User Id */
    __u32 gid; /* Group Id */
    char task[COMMAND_LEN], *full_task_path;
};

// ----- ----- STRUCTS AND CLASSES ----- ----- //
struct ipv4_kernel_data {
    __u64 saddr;
    __u64 daddr;
    struct netInfo net;
};

struct ipv6_kernel_data {
    unsigned __int128 saddr;
    unsigned __int128 daddr;
    struct netInfo net;
};

typedef struct {
    __u64 ktime;
    char ifname[IFNAMSIZ];
    struct timeval event_time;
    __u8  ip_version:4, sent_packet:4;

    union {
        struct ipv4_kernel_data v4;
        struct ipv6_kernel_data v6;
    } event;

    struct taskInfo proc, father;

    char cgroup_id[CGROUP_ID_LEN];
    } eBPFevent;
```
libebpfflow Overview [2/2]

// Attaching probes ------ //
if (userarg_eoutput && userarg_tcp) {
    // IPv4
    AttachWrapper(&ebpf_kernel, "tcp_v4_connect",  "trace_connect_entry",     BPF_PROBE_ENTRY);
    AttachWrapper(&ebpf_kernel, "tcp_v4_connect",  "trace_connect_v4_return", BPF_PROBE_RETURN);
    // IPv6
    AttachWrapper(&ebpf_kernel, "tcp_v6_connect",  "trace_connect_entry",     BPF_PROBE_ENTRY);
    AttachWrapper(&ebpf_kernel, "tcp_v6_connect",  "trace_connect_v6_return", BPF_PROBE_RETURN);
}

if (userarg_einput && userarg_tcp)
    AttachWrapper(&ebpf_kernel, "inet_csk_accept", "trace_accept_return",     BPF_PROBE_RETURN);

if (userarg_retr)
    AttachWrapper(&ebpf_kernel, "tcp_retransmit_skb", "trace_tcp_retransmit_skb", BPF_PROBE_ENTRY);

if (userarg_tcpclose)
    AttachWrapper(&ebpf_kernel, "tcp_set_state", "trace_tcp_close", BPF_PROBE_ENTRY);

if (userarg_einput && userarg_udp)
    AttachWrapper(&ebpf_kernel, "inet_recvmsg",    "trace_inet_recvmsg_entry",  BPF_PROBE_ENTRY);
    AttachWrapper(&ebpf_kernel, "inet_recvmsg",    "trace_inet_recvmsg_return", BPF_PROBE_RETURN);

if (userarg_eoutput && userarg_udp) {
    AttachWrapper(&ebpf_kernel, "udp_sendmsg",     "trace_udp_sendmsg_entry",   BPF_PROBE_ENTRY);
    AttachWrapper(&ebpf_kernel, "udpv6_sendmsg",   "trace_udp_sendmsg_entry",   BPF_PROBE_ENTRY);
}
Gathering Information Through eBPF

• In Linux every task has associated a struct (i.e. `task_struct`) that can be retrieved by invoking the function `bpf_get_current_task` provided by eBPF. By navigating through the kernel structures it can be gathered:
  ◦ `uid`, `gid`, `pid`, `tid`, process name and executable path
  ◦ `cgroups` associated with the task.
  ◦ `connection details`: source and destination ip/port, bytes send and received, protocol used.
Containers Visibility: cgroups and Docker

• For each container Docker creates a cgroup whose name corresponds to the container identifier.

• Therefore by looking at the task cgroup the docker identifier can be retrieved and further information collected.
TCP Under the Hood: accept

A probe has been attached to `inet_csk_accept`
- Used to accept the next outstanding connection.
- Returns the socket that will be used for the communication, NULL if an error occurs.
- Information is collected both from the socket returned and from the `task_struct` associated with the process that triggered the event.

In a similar fashion events concerning retransmissions and socket closure can be monitored.
TCP Under the Hood: connect

An hash table, indexed with thread IDs, has been used:

◦ When connect is invoked the socket is collected from the function arguments and stored together with the kernel time.

◦ When the function terminates the execution, the return value is collected and the thread ID is used to retrieve the socket from the hash table.

◦ The kernel time is used to calculate the connection latency.
Integrating eBPF with ntopng

• We have done an early integration of eBPF with ntopng using the libebpflow library we developed:
  ◦ Incoming TCP/UDP events are mapped to packets monitored by ntopng.
  ◦ We’ve added user/process/flow integration and partially implemented process and user statistics.

• Work in progress
  ◦ Container visibility (including pod), retransmissions… are reported by eBPF but not yet handled inside ntopng.
  ◦ To do things properly we need to implement a system interface in ntopng where to send all eBPF events.
**ntopng with eBPF: Flows**

### Active Flows

<table>
<thead>
<tr>
<th>Application</th>
<th>L4 Proto</th>
<th>Client</th>
<th>Server</th>
<th>Duration</th>
<th>Breakdown</th>
<th>Actual Thpt</th>
<th>Total Bytes</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info</td>
<td>ICMP</td>
<td>217.29.75.4</td>
<td>pc gamer, nic.it</td>
<td>19:04:30</td>
<td>Client</td>
<td>0 bit/s</td>
<td>1.32 MB</td>
<td>Echo Reply</td>
</tr>
<tr>
<td>Info</td>
<td>IMAPS</td>
<td>TCP</td>
<td>93.62.150.157</td>
<td>12:16:18</td>
<td>Client</td>
<td>0 bit/s</td>
<td>370.53 KB</td>
<td></td>
</tr>
<tr>
<td>Info</td>
<td>IMAPS</td>
<td>TCP</td>
<td>146.48.88.155</td>
<td>04:47:03</td>
<td>Client</td>
<td>0 bit/s</td>
<td>407.59 KB</td>
<td></td>
</tr>
<tr>
<td>Info</td>
<td>SSL:Dropbox</td>
<td>TCP</td>
<td>bolt.dropbox.com</td>
<td>01:27:35</td>
<td>Client</td>
<td>0 bit/s</td>
<td>785.7 KB</td>
<td>bolt.dropbox.com</td>
</tr>
<tr>
<td>Info</td>
<td>SSL:Dropbox</td>
<td>TCP</td>
<td>bolt.dropbox.com</td>
<td>01:37</td>
<td>Client</td>
<td>0 bit/s</td>
<td>1.21 KB</td>
<td></td>
</tr>
<tr>
<td>Info</td>
<td>MDNS</td>
<td>UDP</td>
<td>muite, nic.it</td>
<td>06:53</td>
<td>Client</td>
<td>0 bit/s</td>
<td>7.54 KB</td>
<td></td>
</tr>
<tr>
<td>Info</td>
<td>MDNS</td>
<td>UDP</td>
<td>muite, nic.it</td>
<td>01:37</td>
<td>Client</td>
<td>0 bit/s</td>
<td>1.21 KB</td>
<td></td>
</tr>
<tr>
<td>Info</td>
<td>SSL:Telegram</td>
<td>TCP</td>
<td>149.154.167.91</td>
<td>01:42</td>
<td>Client</td>
<td>0 bit/s</td>
<td>3.27 KB</td>
<td></td>
</tr>
<tr>
<td>Info</td>
<td>SSL:ntop</td>
<td>TCP</td>
<td>i7.ntop.org</td>
<td>00:06</td>
<td>Client</td>
<td>0 bit/s</td>
<td>6.53 KB</td>
<td>i7.ntop.org</td>
</tr>
<tr>
<td>Info</td>
<td>SSL:ntop</td>
<td>TCP</td>
<td>i7.ntop.org</td>
<td>00:06</td>
<td>Client</td>
<td>0 bit/s</td>
<td>6.29 KB</td>
<td>i7.ntop.org</td>
</tr>
</tbody>
</table>
ntopng with eBPF: Users + Processes
ntopng with eBPF: Processes + Protocols
Current eBPF Work Items: UDP

- Contrary to TCP, in UDP we need to handle packets. To avoid overloading the system we are using an in-kernel LRU to minimise load: is there a better option available that avoids us playing with packets at all?
- As in UDP each packet can have a different destination, intercepting up in the stack some metadata info are missing (local IP/Ethernet is computed after routing decision).
- Better multicast handling.
BCC/eBPF Pitfalls

• BCC (BPF Compiler Collection) has limitations in terms of:
  ◦ Function complexity/length: memory/stack and loop unroll are limited.
  ◦ Sometimes its behaviour is non deterministic.

• Inability to read message drops number.

• Packet decoding can be a nightmare due to restrictions on function calls

- Frame 1: 217 bytes on wire (1736 bits), 217 bytes captured (1736 bits) on interface 0
- Ethernet II, Src: [MAC address], Dst: [MAC address]
- Internet Protocol Version 4, Src: [IP address], Dst: [IP address]
- Generic Routing Encapsulation (ERSPAN)
- Encapsulated Remote Switch Packet ANalysis Type II
- Ethernet II, Src: [MAC address], Dst: [MAC address]
- 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 21
- Internet Protocol Version 4, Src: [IP address], Dst: [IP address]
- User Datagram Protocol, Src Port: 64556, Dst Port: 3389
- Data (121 bytes)
Conclusions

• With eBPF it is now possible to have full system and network visibility in an integrated fashion.
• Contrary to Sysdig, eBPF load on the system is basically unnoticeable and no kernel module is necessary (i.e. issues of early work are now solved).
• Container/user/process information allows us to enhance network communications with metadata that is great not just for visibility but also for spotting malicious system activities.
• eBPF will be part of ntopng 4.0 due in late spring.