Application-layer traffic processing with eBPF: a simple HTTP Filter

Simple HTTP filter

eBPF application that parses HTTP packets and extracts (and prints on screen) the URL contained in the GET/POST request.

https://github.com/netgroup-polito/ebpf-test

Usage Example (launch program then use your browser):

```
$ sudo python http-parse-complete.py
GET /pipermail/iovisor-dev/ HTTP/1.1
HTTP/1.1 200 OK
GET /favicon.ico HTTP/1.1
HTTP/1.1 404 Not Found
GET /pipermail/iovisor-dev/2016-January/thread.html HTTP/1.1
HTTP/1.1 200 OK
GET /pipermail/iovisor-dev/2016-January/000046.html HTTP/1.1
HTTP/1.1 200 OK
```

Implementation using BCC

- eBPF socket filter.
- Filters IP and TCP packets, containing "HTTP", "GET", "POST" in payload and all subsequent packets belonging to the same session, having the same (ip_src,ip_dst,port_src,port_dst).
- Program is loaded as PROG_TYPE_SOCKET_FILTER and attached to a socket, bind to eth0.
- Matching packets are forwarded to user space, others dropped by the filter.

- Python script reads filtered raw packets from the socket, if necessary reassembles packets belonging to the same session, and prints on stdout the first line of the HTTP GET/POST request.

Filter HTTP traffic

eBPF

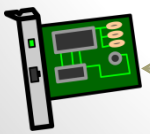
BPF program of type SOCKET_FILTER

BPF session Map

http-parse-complete.c

```
/*eBPF program.
Filter IP and TCP packets, having payload not empty
and containing "HTTP", "GET", "POST"  as first bytes
of payload AND ALL the other packets having same
(src_ip,dst_ip,src_port,dst_port) this means
belonging to the same "session".
this additional check avoids url truncation, if url
is too long  userspace script, if necessary,
reassembles urls splitted in more packets.
*/
1.int http_filter(struct __sk_buff *skb) {
2.    //code ...
3.    //send packet to userspace
4.    KEEP:
5.    return -1;
6.    //drop the packet
7.    DROP:
8.    return 0;
9.}
```

Raw Socket Bind to eth0



kernel

Filtered packets on socket

userspace

http-parse-complete.py

- Load eBPF program.
- Attach it to raw socket bind to eth0.
- Initialize sessions Map.
  - Key:(ip_src,ip_dst,port_src,port_dst)
  - Value: timestamp
- Read filtered packet (all packets of HTTP session)
- Perform some check to eventually reassemble splitted packets.
- print the url contained in HTTP GET/POST request.

GET /path/to/file/index.html HTTP/1.0
POST /path/to/file/index.html HTTP/1.0

- *This complete version solve the problem of long urls splitted in more packets*

Final Remarks

- We used eBPF for a part of the processing.
- In this case (Application-layer traffic) some eBPF constaints forced us to split this type of analysis in part in eBPF and in part in userspace.
- This hybrid approach is the only way because we cannot perform complex HTTP payload analysis inside ebpf program, mainly because of limitations on string operation.