eBPF LEARNING PROCESS

1. Read many articles and documentation about BPF (links in documentation)

Learn about

* History (BCC, libbpf, …)
* BPF CO-RE
* Maps
* Probes and hooks and tracepoints

1. Implement examples using solo.io/bumblebee (links in examples/bumblebee\_examples)

It is a repository that helps to create some simple BPF program building the kernel code and doing by itself the user part.

* See the functioning of a BPF program based on its hook in the kernel
* See a first implementation of a BPF program

1. Implement some examples from scratch (links in documentation/2 eBPF incrontro 3.docx)

These are the examples in the folder examples/first\_examples\_and\_maps.

Start from a simple Hello\_world program to the use of maps

Learn about

* In SEC(“…”) -> hook/hook\_name
* In SEC(“hook/hook\_name\_in\_kernel”) func\_name (params) {…}

-> params is the name of the function in the kernel attached to the hook or some data structures inside the kernel.

From this point, the function and its parameters or the data structure can be accessed by the BPF function.

Each SEC gives the permission to use some helpers: depends on the attach point in the kernel.

You can access to the parameters of the function in the kernel to see its behaviour from the kernel point of view

->(function\_name, param\_func1, param\_func2,…)

* Difference in writing code inside or outside the SEC function (for example writing a function and call it back inside the SEC function)
  + Inside: gives no problem if written correctly
  + Outside: many things to be careful of
    - Return type: sometimes the kernel wants the function to return something, maybe to understand that the function ends.

Example:

void array\_map\_use(…) {…} gives error

int array\_map\_use(…) {… return 0;} does not return an error

both functions are called inside the SEC function without assigning its return value to a variable.

* + - Everything written outside the SEC function is seen as global.

if the function outside the SEC are not declared STATIC, BPF puts them in a single section of the program

->this might exceed the maximum dimension of a BPF program

SOLUTION: declare all the function outside the SEC as “static”.

OTHER SOLUTION: write everything in the SEC function (if possible)

* Map methods

Tried to use update, lookup and delete to play with some type of maps

Learn about:

* + Not all methods are valid for all type of maps

Examples (check the examples/first\_examples\_with\_maps folder):

Methods for ring buffer are valid only for BPF\_MAP\_TYPE\_RINGBUF

Delete is not valid for BPF\_MAP\_TYPE\_ARRAY -> return -EINVAL

BPF\_MAP\_TYPE\_HASH is working with all methods

* + Lookup -> return void \*

It is a generic pointer and it points to the value in the BPF map associated with the reference of the key given to the method.

If used as so and if printed it is just a pointer to a memory cell.

Based on the type of the value in the BPF map, you can see the value associated to the key by doing an explicit cast and dereferencing the pointer.

Returns -> value of element associated to the key in the map

Cast + dereference -> access the value in the map

* + Update and delete -> return long (0 if success, number < 0 if error)