Quick Introduction to eBPF: Info needed during hands-on session

Preparing Experiment Environment

Install a Virtual Machine

If you want to follow the option to use a virtual machine with pre-installed dependencies read here. The virtual machines are prepared by Sebastiano Miano for the Network Computing coures (2024/2025 DEIB Politecnico di Milano).

You can download the VirtualBox nc-labs-x86.ova image at the following URL: https://networkcomputingpolimi.page.link/labs Before deploying the VM, you must install VirtualBox, compatible with Windows, Linux, and macOS. Download the VirtualBox binaries at the following URL: https://www.virtualbox.org/wiki/Downloads Once installed, open the nc-labs-x86.ova file to boot the VM and access the lab environment.

Required Packages

If you have a Linux system and want to run test locally install the following packages:

- clang (v15)
- libelf
- libbpf (v1.0+)
- bpftool

Install Dependencies On Ubuntu 22.04:

> Also look at `./scripts/install_req.sh`

```
# General stuff
sudo apt update
sudo apt install -y gcc-multilib build-essential libelf-dev linux-tools-`uname -r`
# LLVM/Clang v15
wget https://apt.llvm.org/llvm.sh
sudo bash ./llvm.sh 15
# Libbpf
# install a prebuilt version (v1.0 or more) or follow the instructions at
# https://github.com/libbpf/libbpf
```

Remote Virtual Machine

There a few virtual machine that you can use for the purpose of following the class. You can connect to them with `ssh`.

On the server you can either use vim/nano/... to write the codes, or check the *VS Code* remote development guide if you want to use it as an editor https://code.visualstudio.com/docs/remote/ssh

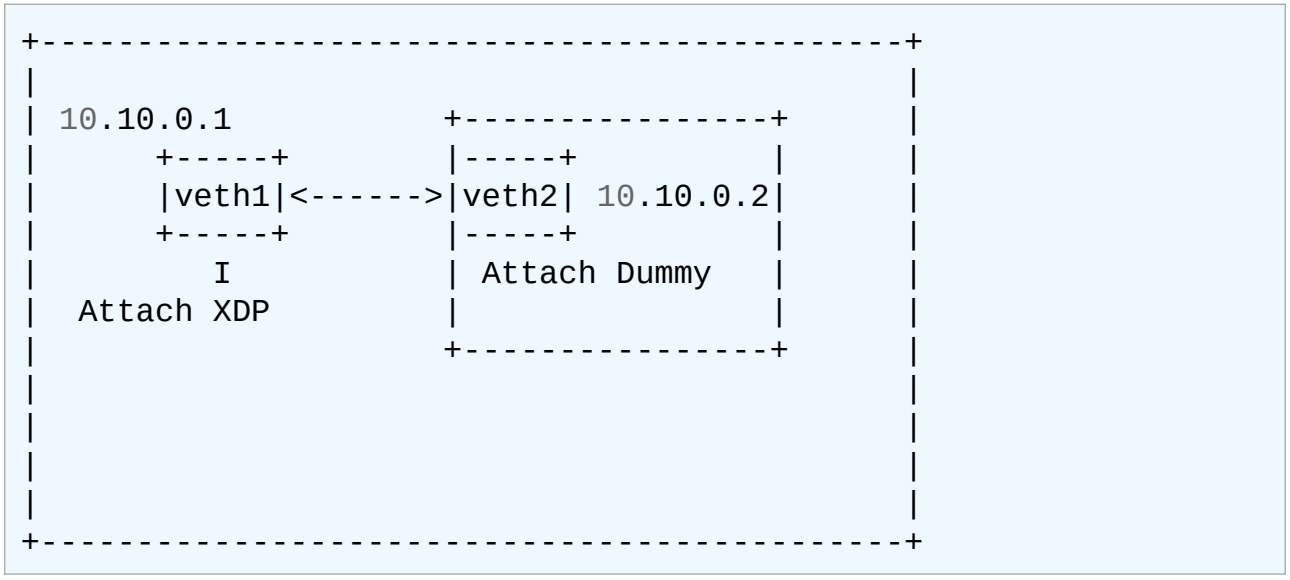
First eBPF Program

```c
#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>

SEC("xdp")
int xdp_main(struct xdp_md *ctx)
{
    bpf_printk("here");
    return XDP_PASS;
}

char LICENSE[] SEC("license") = "GPL";
```

First Time Running eBPF Program

Setup Environment:

```
sudo ip netns add n2
sudo ip link add veth1 type veth peer name veth2 netns n2
sudo ip link set veth1 up
sudo ip addr add 10.10.0.1/24 dev veth1

sudo ip netns exec n2 ip link set veth2 up
sudo ip netns exec n2 ip addr add 10.10.0.2/24 dev veth2
```



Compile eBPF Program:

```
clang -S \
  -target bpf \
  -g -O2 -emit-llvm \
  -o NAME.bpf.ll NAME.bpf.c

llc -mcpu=probe -march=bpf -filetype=obj -o NAME.bpf.o NAME.bpf.ll

bpftool gen skeleton NAME.bpf.o name SKEL_NAME > NAME.skel.h
```

Compile Loader Program:

```
clang -g -O2 -o ./loader ./loader.c -lbpf -lelf
```

Reading BPF Trace Logs:

```
sudo cat /sys/kernel/tracing/trace_pipe
```

Generating Packets

Running NetCat Server (listen for packets):

```
nc -l -u 10.10.0.1 8080
```

Running NetCat Sending Packets:

```
printf "hello world\n" | nc -W 1 -N -u 10.10.0.1 8080
```

Using IPROUTE2 To Load XDP Programs

```
#! /bin/bash
ip link set dev veth2 xdp off
sudo ip link set dev veth2 xdp obj first.bpf.o sec xdp

on_signal() {
        ip link set dev veth2 xdp off
        exit 0
}

trap "on_signal" SIGINT SIGHUP
echo Hit Ctrl-C
while [[ true ]]; do
        sleep 5
done
```

Defining an eBPF MAP

```
struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __type(key, struct key);
    __type(value, struct value);
    __uint(max_entries, 16);
} table SEC(".maps");
```

MAP Types:

- BPF_MAP_TYPE_ARRAY
- BPF_MAP_TYPE_HASH
- BPF_MAP_TYPE_LRU_HASH

BPFTOOL

Listing attached eBPF Networking Programs:

```
sudo bpftool net
```

Listing Loaded eBPF Programs:

```
sudo bpftool prog
```