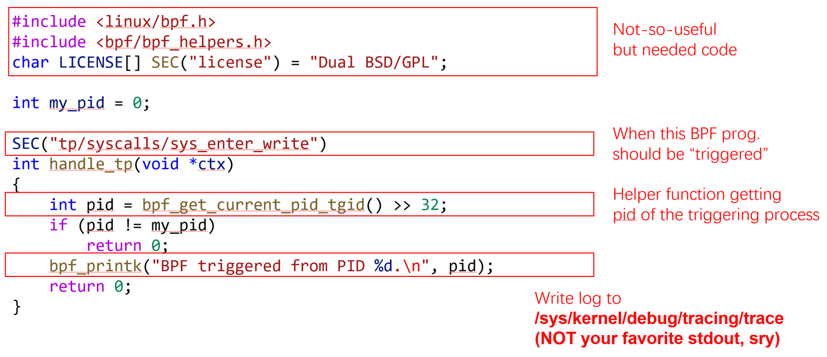
CS 315 BPF for Kernel Security.

1 What is BPF?

BPF was originally introduced to facilitate flexible network packet filtering. Instead of inspecting packets in the user space, users can provide BPF instructions specifying packet filter rules, which are directly executed in the kernel. BPF allows configurable packet filtering without costly context switching and data copying. Modern Linux kernel features extended BPF (eBPF), a Linux subsystem which supports a wide range of use cases, such as kernel profiling, load balancing, and firewalls. Popular applications such as Docker, Katran, and kernel debugging utilities like Kprobes utilize or are built directly on top of BPF.

Basically, BPF means "hooks"; it allows users to execute certain code when a kernel event occur.



```c
#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>
char LICENSE[] SEC("license") = "Dual BSD/GPL";

int my_pid = 0;

SEC("tp/syscalls/sys_enter_write")
int handle_tp(void *ctx)
{
    int pid = bpf_get_current_pid_tgid() >> 32;
    if (pid != my_pid)
        return 0;
    bpf_printk("BPF triggered from PID %d.\n", pid);
    return 0;
}
```

Not-so-useful but needed code
When this BPF prog. should be "triggered"
Helper function getting pid of the triggering process
Write log to /sys/kernel/debug/tracing/trace (NOT your favorite stdout, sry)

You might find the following resources useful.

https://man7.org/linux/man-pages/man7/bpf-helpers.7.html

https://docs.kernel.org/bpf/libbpf/program_types.html

Common Notes:

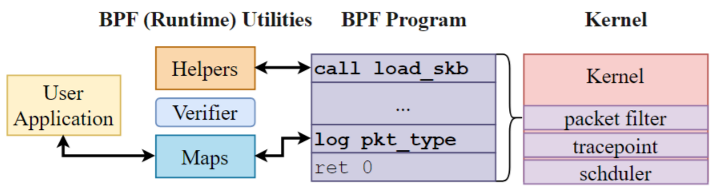
- BPF does not have "loops", you should avoid things like `for(int i = 0; i < val; i++)`.

  You can write "deterministic loops", like `for(int i = 0; i < 10; i++)`; this will simply repeat your code 10 times in the final binary.

- **Be sure to check out the documents.** Though it looks like C programs (it is in some way), but it CANNOT use most things that you are familar with.

2 How to execute a BPF program?

BPF programs consists of two parts: user-space loader and in-kernel BPF program. The first one is a normal C program, in charge of loading the program into the kernel. The second one is the real deal that executes inside the kernel. They communicate via a special data structure named BPF maps.



The following link is a example user-space loader that loads the super simple BPF program.

https://github.com/libbpf/libbpf-bootstrap/blob/master/examples/c/minimal.c

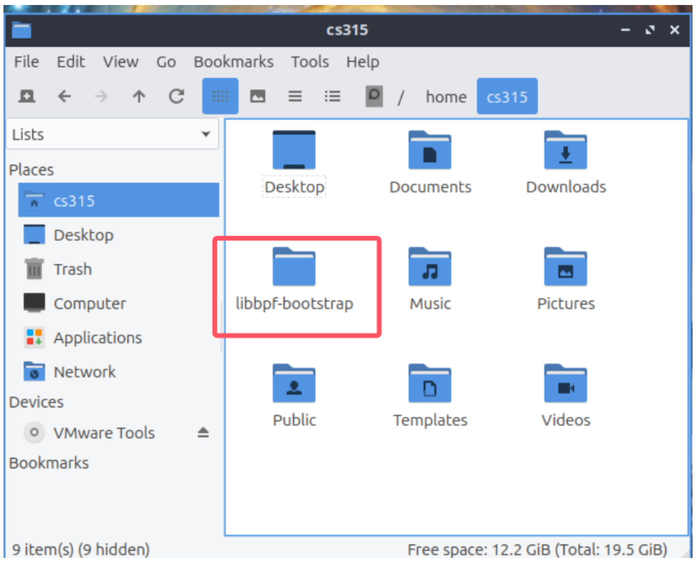
The following link is the super simple BPF program.

https://github.com/libbpf/libbpf-bootstrap/blob/master/examples/c/minimal.bpf.c

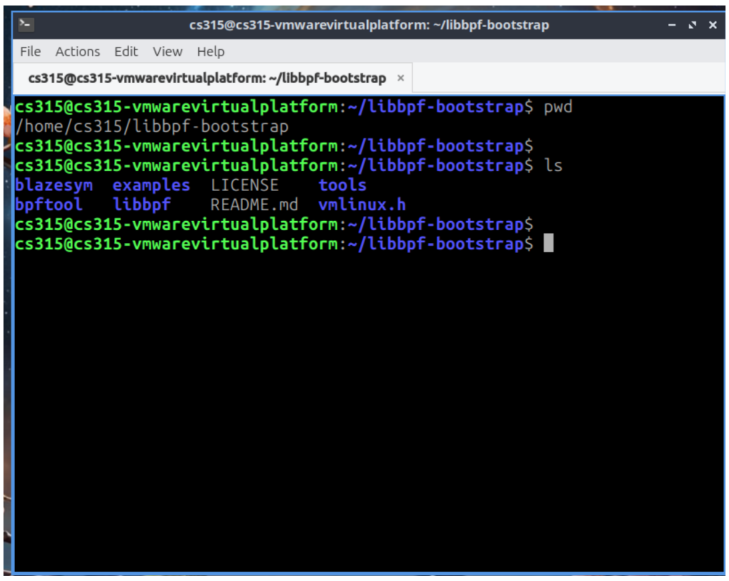
3 Let's try it out.

Compile BPF programs.

There should be folder named `libbpf-bootstrap` in your home folder like this.



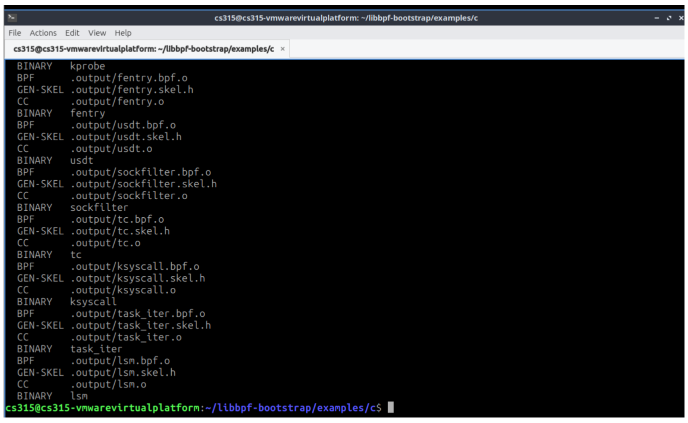
Open a terminal in this folder like this (If you don't know how to do this in Linux, ask TA or consider changing a major).


cs315@cs315-vmwarevirtualplatform: ~/libbpf-bootstrap
File Actions Edit View Help

```
cs315@cs315-vmwarevirtualplatform:~/libbpf-bootstrap$ pwd
/home/cs315/libbpf-bootstrap
cs315@cs315-vmwarevirtualplatform:~/libbpf-bootstrap$
cs315@cs315-vmwarevirtualplatform:~/libbpf-bootstrap$ ls
blazesym  examples  LICENSE    tools
bpftool   libbpf    README.md  vmlinux.h
cs315@cs315-vmwarevirtualplatform:~/libbpf-bootstrap$
cs315@cs315-vmwarevirtualplatform:~/libbpf-bootstrap$ 
```

We see an `example` folder, where you can find some simple BPF programs written in C.

Run `cd example/c` and `make` and you should see a bunch of outputs.



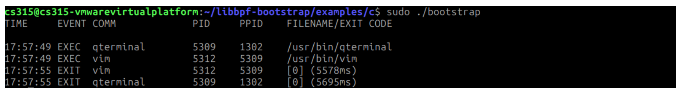
cs315@cs315-vmwarevirtualplatform: ~/libbpf-bootstrap/examples/c
File Actions Edit View Help
cs315@cs315-vmwarevirtualplatform: ~/libbpf-bootstrap/examples/c

```
 BINARY   kprobe
 BPF      .output/fentry.bpf.o
 GEN-SKEL .output/fentry.skel.h
 CC       .output/fentry.o
 BINARY   fentry
 BPF      .output/usdt.bpf.o
 GEN-SKEL .output/usdt.skel.h
 CC       .output/usdt.o
 BINARY   usdt
 BPF      .output/sockfilter.bpf.o
 GEN-SKEL .output/sockfilter.skel.h
 CC       .output/sockfilter.o
 BINARY   sockfilter
 BPF      .output/tc.bpf.o
 GEN-SKEL .output/tc.skel.h
 CC       .output/tc.o
 BINARY   tc
 BPF      .output/ksyscall.bpf.o
 GEN-SKEL .output/ksyscall.skel.h
 CC       .output/ksyscall.o
 BINARY   ksyscall
 BPF      .output/task_iter.bpf.o
 GEN-SKEL .output/task_iter.skel.h
 CC       .output/task_iter.o
 BINARY   task_iter
 BPF      .output/lsm.bpf.o
 GEN-SKEL .output/lsm.skel.h
 CC       .output/lsm.o
 BINARY   lsm
cs315@cs315-vmwarevirtualplatform:~/libbpf-bootstrap/examples/c$
```

Explore & try BPF.

> **NOTE:** Please take some time to read through the source codes of these examples, e.g., `minimal.bpf.c` and `minimal.c`. Try to understand what they do.

You can run `sudo ./bootstrap` and see how BPF programs capture the kernel events.



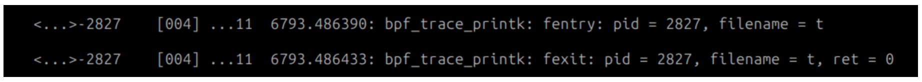
```
cs315@cs315-vmwarevirtualplatform:~/libbpf-bootstrap/examples/c$ sudo ./bootstrap
TIME     EVENT COMM             PID     PPID    FILENAME/EXIT CODE

17:57:49 EXEC  qterminal        5309    1302    /usr/bin/qterminal
17:57:49 EXEC  vim              5312    5309    /usr/bin/vim
17:57:55 EXIT  vim              5312    5309    [0] (5578ms)
17:57:55 EXIT  qterminal        5309    1302    [0] (5695ms)
```

4 Practice!

Listen to syscalls.

`example/c/fentry.bpf.c` is a BPF program that executes upon `do_unlinkat`. It's used to monitor process's file delete request.



```
  <...>-2827    [004] ...11  6793.486390: bpf_trace_printk: fentry: pid = 2827, filename = t

  <...>-2827    [004] ...11  6793.486433: bpf_trace_printk: fexit: pid = 2827, filename = t, ret = 0
```

- Task

  By modify `fentry.bpf.c`, try extending its tracked system calls to `open/read/write/close`.

  Output the log in the format like the following:

  `open: pid = <pid>, filename = <filename>`

  Submit the screenshot of the output of the BPF program you executed, and the program you used to trigger the BPF program.

- Task

  By study `bootstrap.c` and `bootstrap.bpf.c`, try to learn how to use BPF maps to transmit data between BPF program and the user-space loader.

  Use that instead of `bpf_trace_printk` to log informations about system calls.

  Output in the **same** format as previous tasks.

  Submit the screenshot of the output of the BPF program you executed, and the program you used to trigger the BPF program.

LSM BPF program.

`example/c/lsm.bpf.c` gives a very simple example of Linux Security Module (LSM) hooks, which is attached to `lsm/bpf`.

NOTE: This `lsm.bpf.c` is a BPF program that rejects all subsequent `bpf()` syscalls. After running `sudo ./lsm`, you can try to run another BPF programs like `sudo ./minimal` to see the effect of `./lsm`.

There are some other types of BPF LSM programs. Please try to use it to do the following things.

- Task:

  Try modify `lsm.bpf.c` and implement a hook on `lsm/sock_connect` and stop all connection to the IP address `www.baidu.com` and try to `curl baidu.com`.

  Submit the screenshot of `curl baidu.com`

  Submit the screenshot of the BPF log in the following format

  ```
  stopped: pid = <pid>, ipaddr = <..>, protocol = TCP/UDP/..
  ```