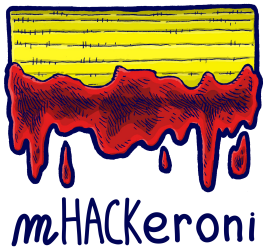
Il Javascript del Kernel Linux?

Un introduzione ad eBPF!

@whoami

consorzio nazionale
interuniversitario
per le telecomunicazioni

- Pasquale Caporaso, phd student, security researcher for CNIT
- Ex-Malware Analyst for Leonardo spa
- Research in cyber security, malware and operating systems
- Addicted to CTFs



- Tor Vergata CTF team? Anyone?

@social

- Linkedin:

  https://www.linkedin.com/in/pasquale-caporaso-4a19b41aa

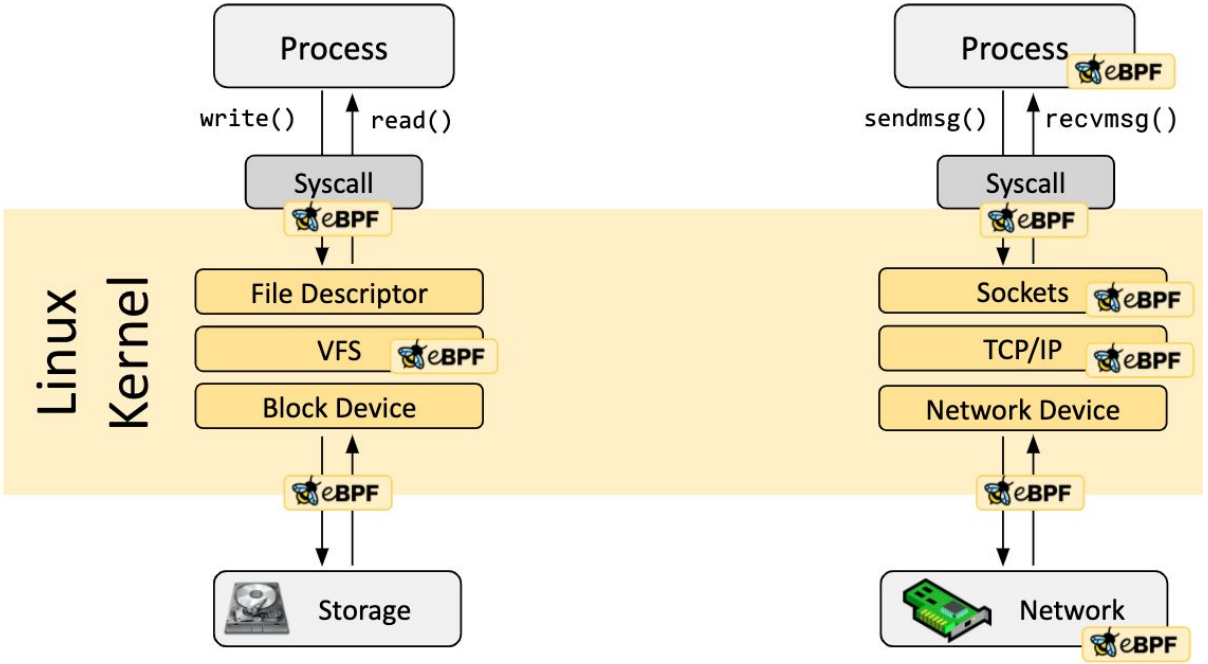
- Mail: pasquale.caporaso@cnit.it
- Telegram: @Capo80

- Github:

  https://github.com/Capo80

What is eBPF?

- run sandboxed programs inside the Linux Kernel (eBPF probes)



https://ebpf.io/what-is-ebpf*

What is eBPF?

- run sandboxed programs inside the Linux Kernel (eBPF probes)

What does javascript have to do with that?*

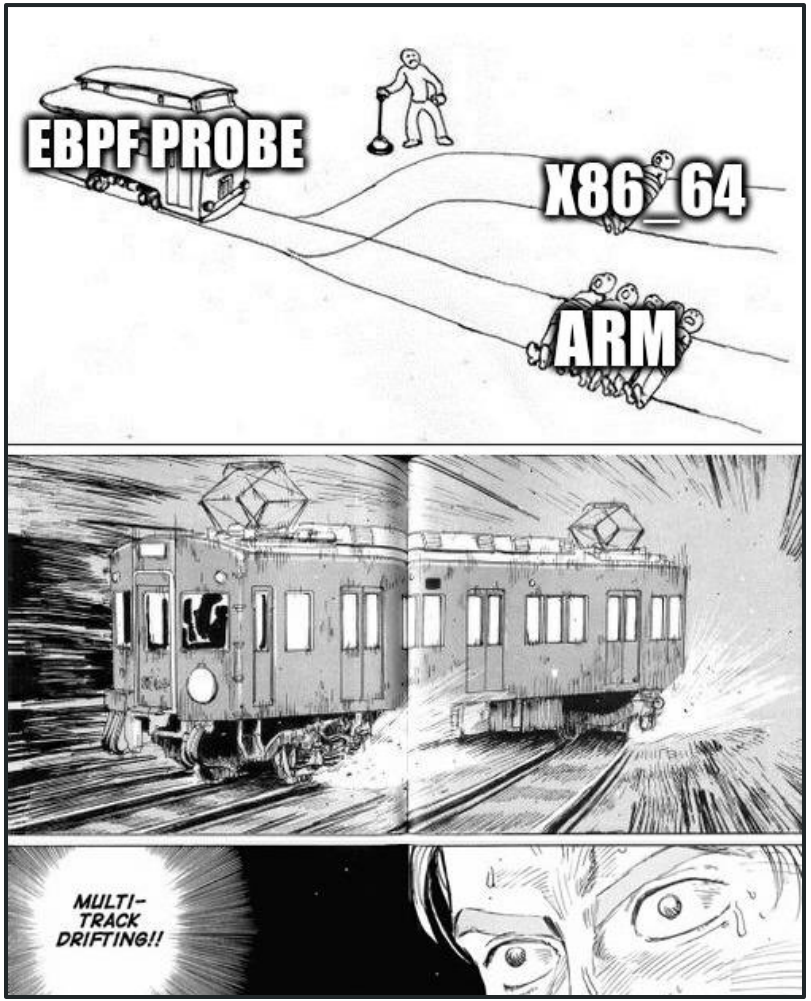
Safety

Portability

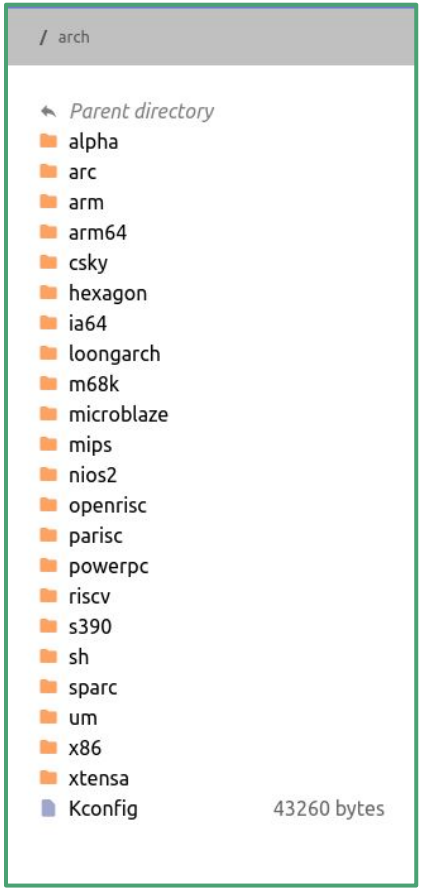
Performance

https://ebpf.io/what-is-ebpf*

Portability



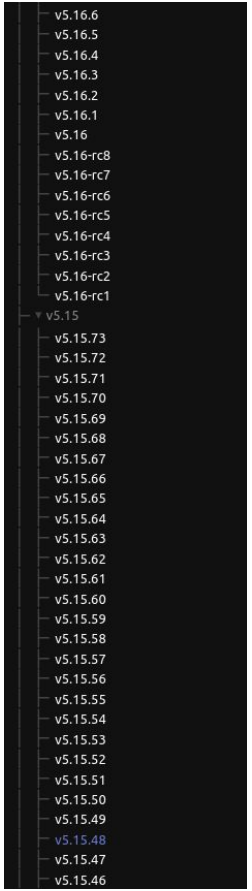
The problem of portability in the Kernel



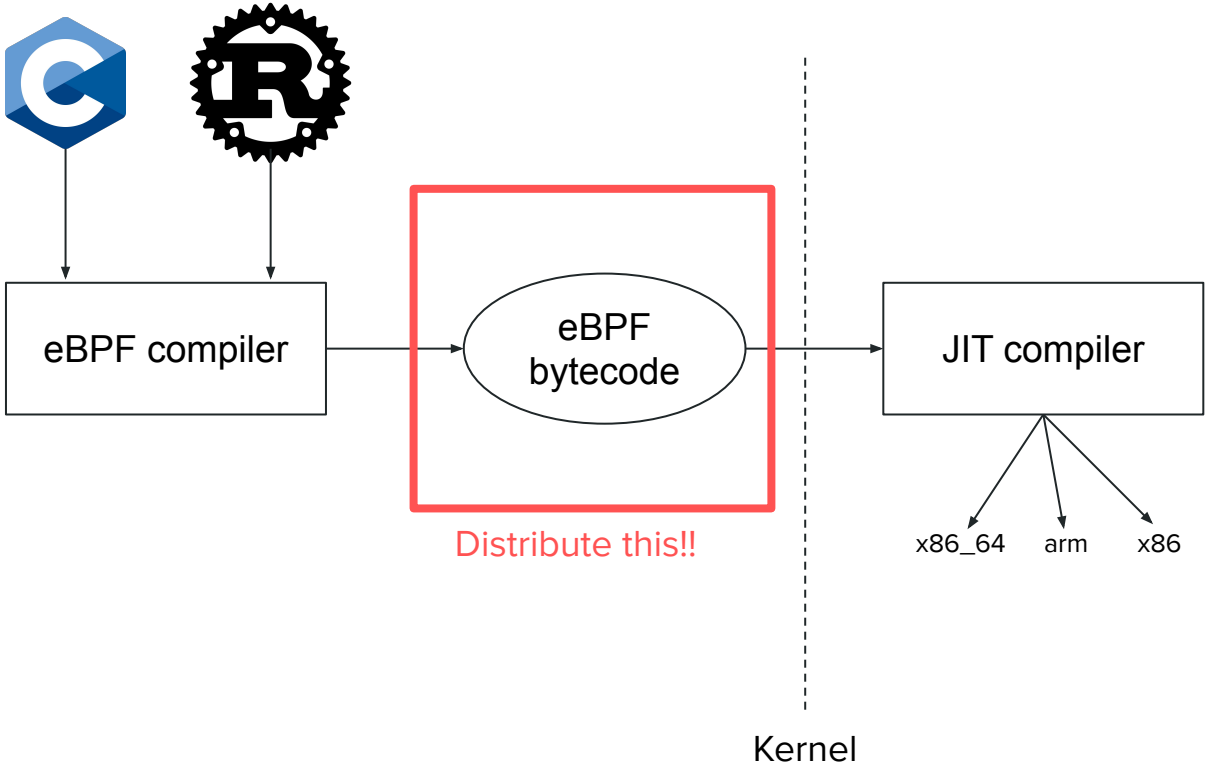
Two issues:

Kernel versions

System architecture



Solving system architecture issues



Solving Kernel versions

Main problem: structures

Example:

```
struct mnt_namespace {
        atomic_t                count;
        struct ns_common        ns;
        struct mount *  root;
        struct list_head        list;
        struct user_namespace   *user_ns;
        struct ucounts          *ucounts;
        u64                     seq;    /* Seque
        wait_queue_head_t poll;
        u64 event;
        unsigned int            mounts; /* # of
        unsigned int            pending_mounts;
} __randomize_layout;
```

Kernel 5.5

```
struct mnt_namespace {
        struct ns_common        ns;
        struct mount *  root;
        /*
         * Traversal and modification of .list is
         * - taking namespace_sem for write, OR
         * - taking namespace_sem for read AND ta
         */
        struct list_head        list;
        spinlock_t              ns_lock;
        struct user_namespace   *user_ns;
        struct ucounts          *ucounts;
        u64                     seq;    /* Sequen
        wait_queue_head_t poll;
        u64 event;
        unsigned int            mounts; /* # of m
        unsigned int            pending_mounts;
} __randomize_layout;
```

Kernel 5.17

The past (kinda): BCC

Include your C code in the program
Compile on the user machine

Problems:

- ugly
- adds a lot of dependencies

```
from bcc import BPF, utils
from optparse import OptionParser

# load BPF program
code="""
#include <uapi/linux/ptrace.h>

struct perf_delta {
    u64 clk_delta;
    u64 inst_delta;
    u64 time_delta;
};

/*
Perf Arrays to read counter values for open
perf events.
*/
BPF_PERF_ARRAY(clk, MAX_CPUS);
```

*https://github.com/iovisor/bcc/blob/master/examples/perf/ipc.py

The future (kinda): CO-RE

Compile Once - Run Everywhere

```
pid_t pid; bpf_probe_read(&pid, sizeof(pid), &task->pid);
```



```
pid_t pid; bpf_core_read(&pid, sizeof(pid), &task->pid);
```

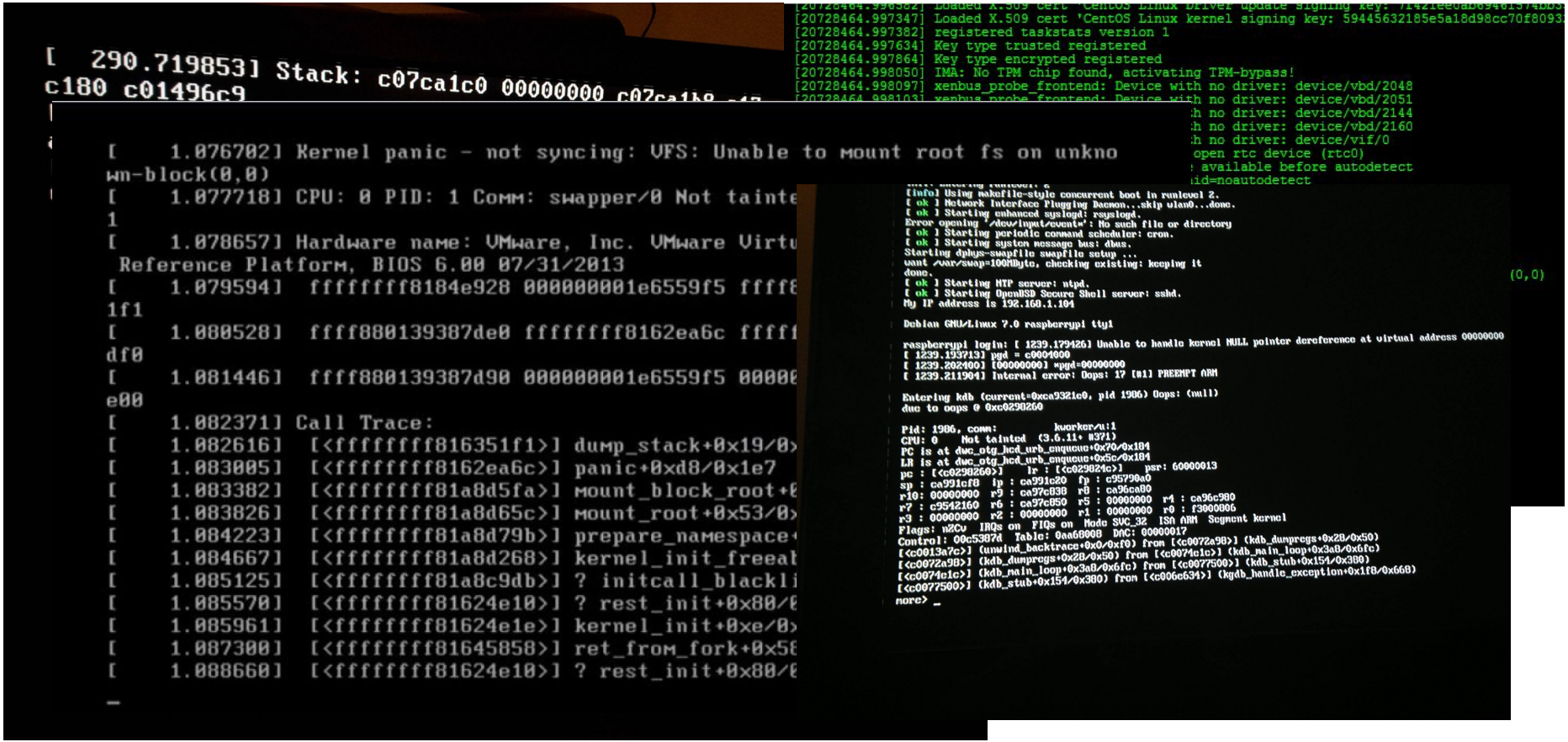
*https://nakryiko.com/posts/bpf-portability-and-co-re/

The future (kinda): CO-RE

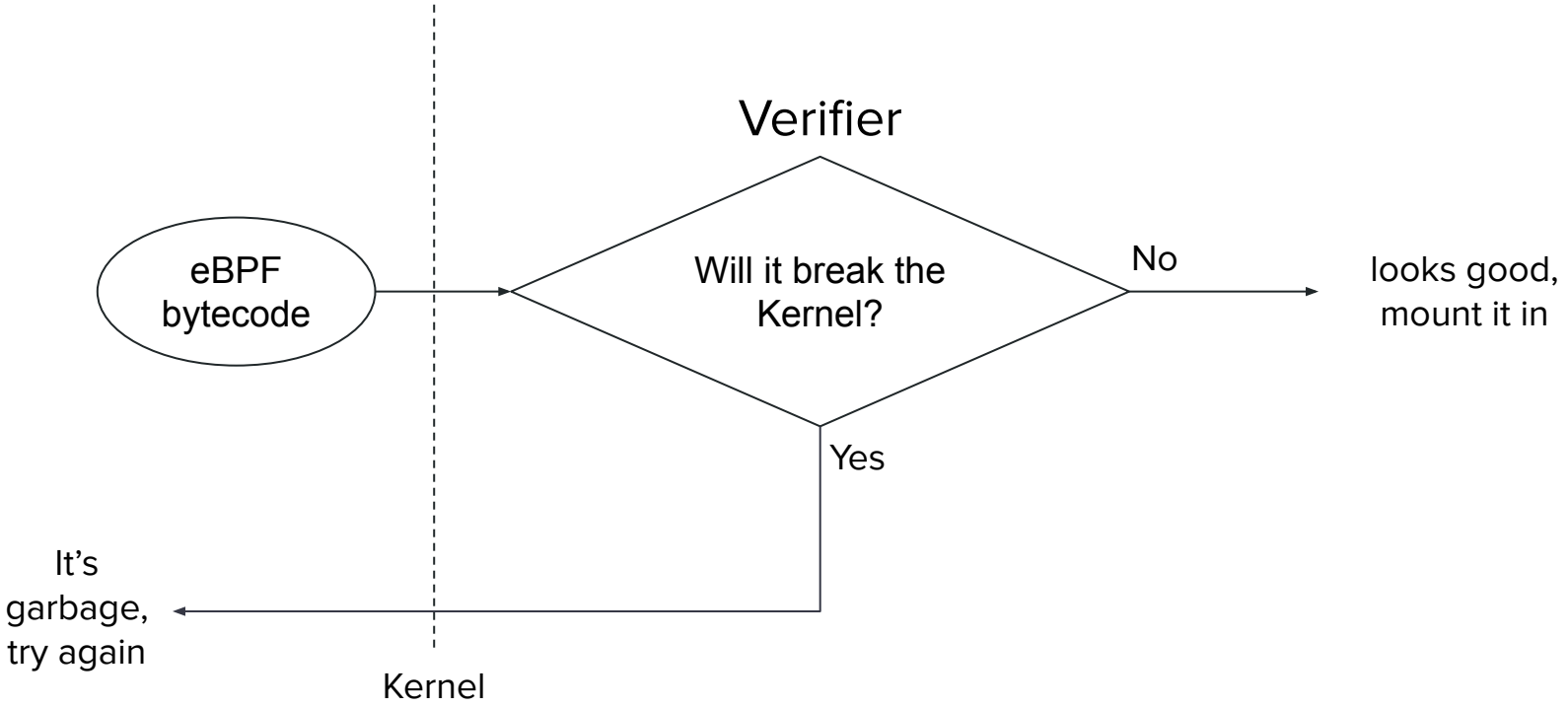
*https://nakryiko.com/posts/bpf-portability-and-co-re/

Safety

The Kernel developer experience



Introducing the eBPF verifier



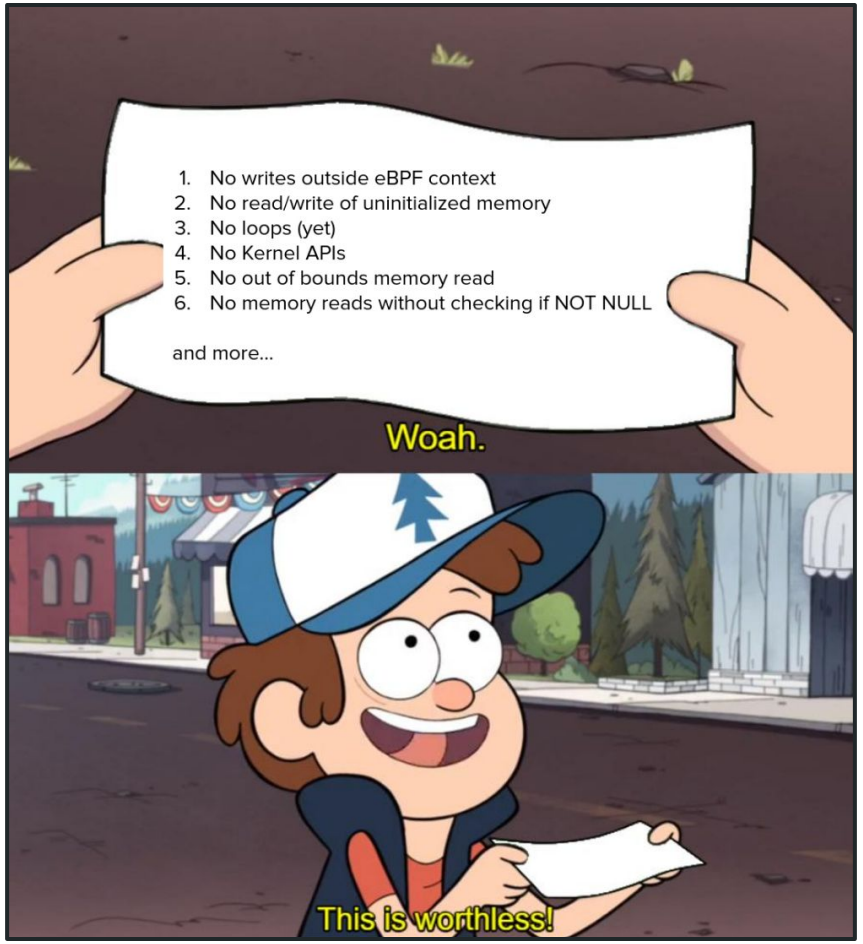
How? eBPF rules

1. No writes outside eBPF context
2. No read/write of uninitialized memory
3. No loops (yet)
4. No Kernel APIs
5. No dynamic memory allocation
6. No out of bounds memory read
7. No memory reads without checking if NOT NULL
8. Max stack size is 512 bytes

and more...

https://docs.kernel.org/bpf/verifier.html

Use cases


1. No writes outside eBPF context
2. No read/write of uninitialized memory
3. No loops (yet)
4. No Kernel APIs
5. No out of bounds memory read
6. No memory reads without checking if NOT NULL
and more...
Woah.
This is worthless!

Network monitoring and traffic manipulation

eCapture 旁观者

Linux kernel monitoring

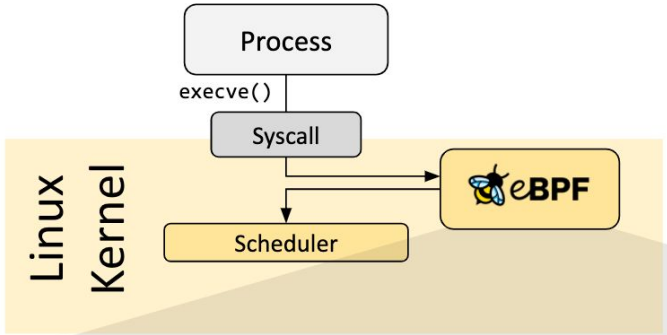
aqua tracee

Falco

parca

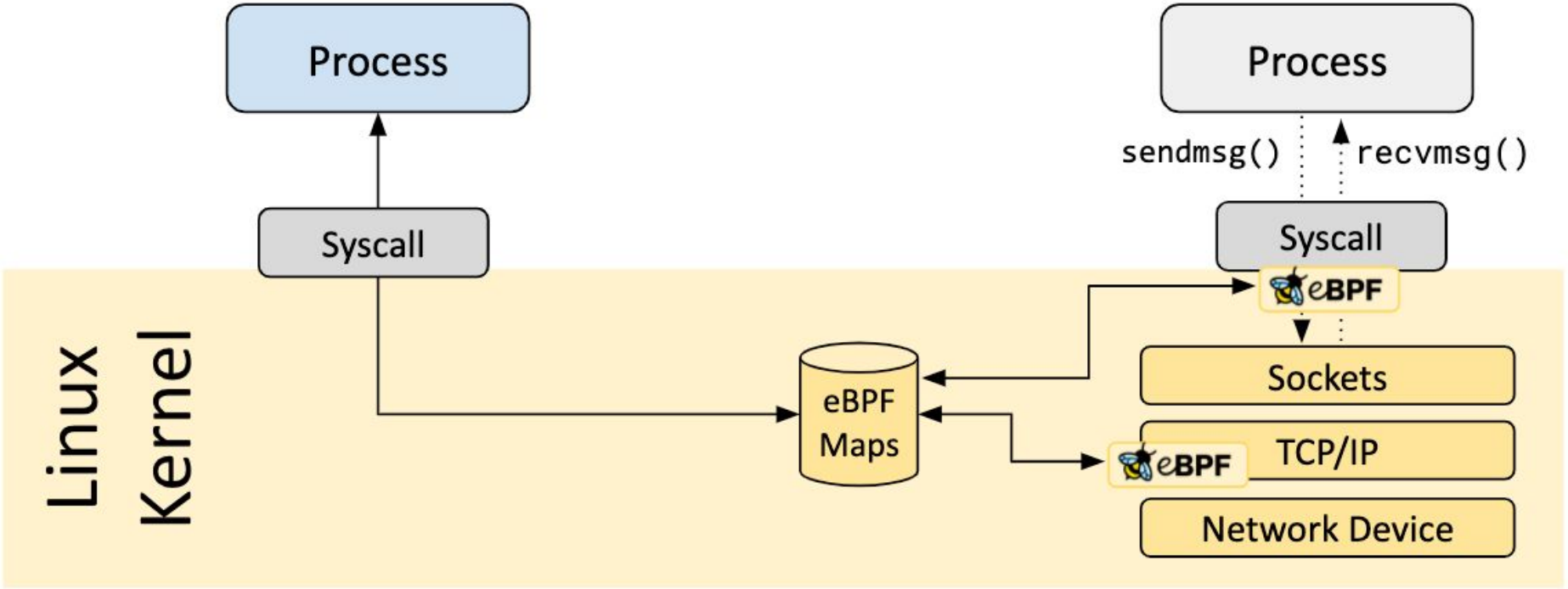
Programming guide

eBPF application architecture

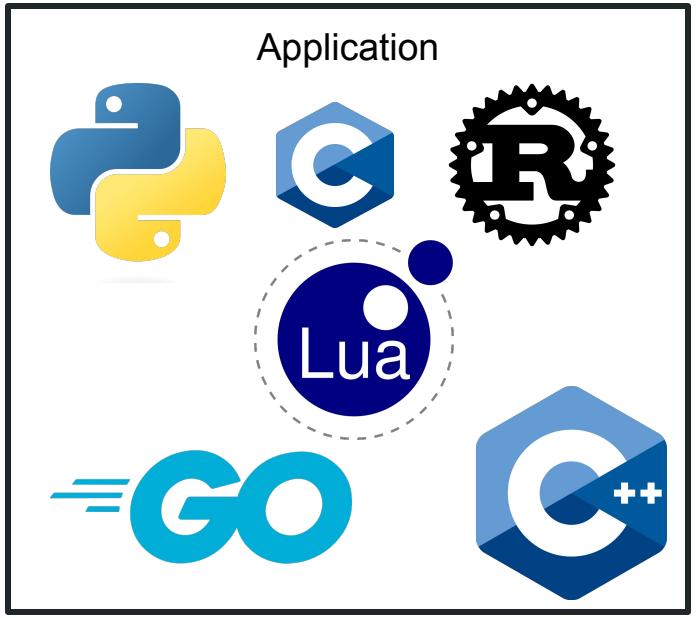


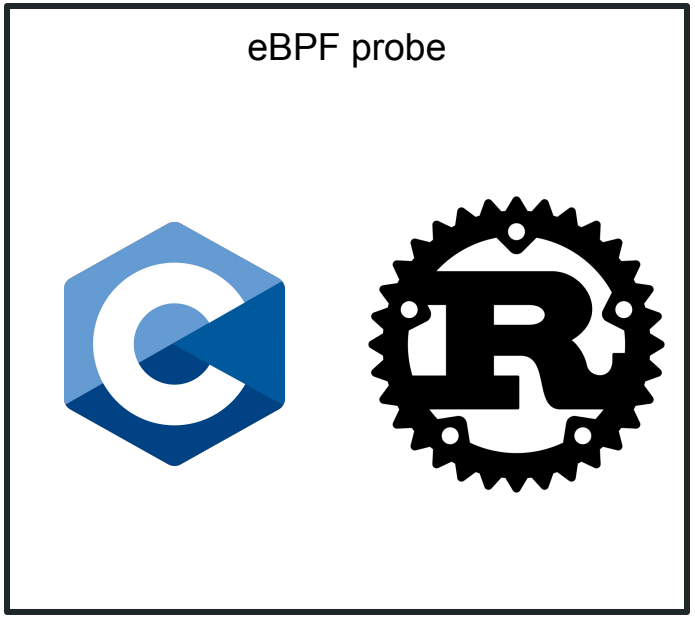

```c
int syscall__ret_execve(struct pt_regs *ctx)
{
        struct comm_event event = {
                .pid = bpf_get_current_pid_tgid() >> 32,
                .type = TYPE_RETURN,
        };

        bpf_get_current_comm(&event.comm, sizeof(event.comm));
        comm_events.perf_submit(ctx, &event, sizeof(event));

        return 0;
}
```

eBPF application architecture



eBPF application architecture





Libraries

BCC (python, lua, c, cpp) no CO-RE:

- https://github.com/iovisor/bcc

libbpf (c, rust) CO-RE:

- https://github.com/libbpf

ebpf (go) CO-RE:

- https://github.com/cilium/ebpf

Programming tips

#1: Learn your errors

- https://docs.kernel.org/bpf/verifier.html

The verifier is very mean, you need to understand it

Programming tips

#2: Loop unrolling is the way

```
#pragma unroll
    for (int i = 0; i < LEN; i++) {
        void *addr = &(data->submit_p->buf[data->buf_off]);
        if (data->buf_off > MAX_PERCPU_BUFSIZE - sizeof(u64))
            // not enough space - return
            goto out;
        if (bpf_probe_read(addr, sizeof(u64), (void *) &ptr[i]) != 0)
            goto out;
        elem_num++;
        data->buf_off += sizeof(u64);
    }
```

1 million instructions means very big loops

Programming tips

#3: Use maps for everything

```
BPF_HASH(bin_args_map, u64, bin_args_t, 256);                   // persist args for send_bin funtion
BPF_HASH(sys_32_to_64_map, u32, u32, 1024);                     // map 32bit to 64bit syscalls
BPF_HASH(params_types_map, u32, u64, 1024);                     // encoded parameters types for event
BPF_HASH(process_tree_map, u32, u32, 10240);                    // filter events by the ancestry of the traced process
BPF_LRU_HASH(task_info_map, u32, task_info_t, 10240);           // holds data for every task
BPF_HASH(network_config, u32, int, 1024);                       // holds the network config for each iface
BPF_HASH(ksymbols_map, ksym_name_t, u64, 1024);                 // holds the addresses of some kernel symbols
BPF_HASH(syscalls_to_check_map, int, u64, 256);                 // syscalls to discover
BPF_LRU_HASH(sock_ctx_map, u64, net_ctx_ext_t, 10240);          // socket address to process context
BPF_LRU_HASH(network_map, net_id_t, net_ctx_t, 10240);          // network identifier to process context
BPF_ARRAY(config_map, config_entry_t, 1);                       // various configurations
BPF_ARRAY(file_filter, path_filter_t, 3);                       // filter vfs_write events
BPF_PERCPU_ARRAY(bufs, buf_t, MAX_BUFFERS);                     // percpu global buffer variables
BPF_PROG_ARRAY(prog_array, MAX_TAIL_CALL);                      // store programs for tail calls
BPF_PROG_ARRAY(prog_array_tp, MAX_TAIL_CALL);                   // store programs for tail calls
BPF_PROG_ARRAY(sys_enter_tails, MAX_EVENT_ID);                  // store syscall specific programs for tail calls from sys_enter
BPF_PROG_ARRAY(sys_exit_tails, MAX_EVENT_ID);                   // store syscall specific programs for tail calls from sys_exit
BPF_PROG_ARRAY(sys_enter_submit_tail, MAX_EVENT_ID);            // store program for submitting syscalls from sys_enter
BPF_PROG_ARRAY(sys_exit_submit_tail, MAX_EVENT_ID);             // store program for submitting syscalls from sys_exit
```

512 bytes is not much for stack size

Programming tips

#4: Limit your indexes

```
buf[buf_off & (MAX_BUFSIZE - 1)]          # if MAX_BUFSIZE multiple of 2
```

```
if (buf_off > MAX_BUFSIZE)
        buf_off = MAX_BUFSIZE
```

You will need to fight your compiler

Programming tips

#5: LSM hooks are the best

```
int security_mmap_file(struct file *file, unsigned long prot,
                       unsigned long flags);
int security_mmap_addr(unsigned long addr);
int security_file_mprotect(struct vm_area_struct *vma, unsigned long reqprot,
                           unsigned long prot);
int security_file_lock(struct file *file, unsigned int cmd);
int security_file_fcntl(struct file *file, unsigned int cmd, unsigned long arg);
void security_file_set_fowner(struct file *file);
int security_file_send_sigiotask(struct task_struct *tsk,
                                 struct fown_struct *fown, int sig);
int security_file_receive(struct file *file);
int security_file_open(struct file *file);
int security_task_alloc(struct task_struct *task, unsigned long clone_flags);
void security_task_free(struct task_struct *task);
int security_cred_alloc_blank(struct cred *cred, gfp_t gfp);
void security_cred_free(struct cred *cred);
int security_prepare_creds(struct cred *new, const struct cred *old, gfp_t gfp);
void security_transfer_creds(struct cred *new, const struct cred *old);
void security_cred_getsecid(const struct cred *c, u32 *secid);
int security_kernel_act_as(struct cred *new, u32 secid);
int security_kernel_create_files_as(struct cred *new, struct inode *inode);
int security_kernel_module_request(char *kmod_name);
int security_kernel_load_data(enum kernel_load_data_id id);
int security_kernel_read_file(struct file *file, enum kernel_read_file_id id);
int security_kernel_post_read_file(struct file *file, char *buf, loff_t size,
                                   enum kernel_read_file_id id);
int security_task_fix_setuid(struct cred *new, const struct cred *old,
                             int flags);
int security_task_fix_setgid(struct cred *new, const struct cred *old,
                             int flags);
int security_task_setpgid(struct task_struct *p, pid_t pgid);
int security_task_getpgid(struct task_struct *p);
int security_task_getsid(struct task_struct *p);
void security_task_getsecid(struct task_struct *p, u32 *secid);
int security_task_setnice(struct task_struct *p, int nice);
int security_task_setioprio(struct task_struct *p, int ioprio);
int security_task_getioprio(struct task_struct *p);
int security_task_prlimit(const struct cred *cred, const struct cred *tcred,
                          unsigned int flags);
```

https://elixir.bootlin.com/linux/v5.9/source/include/linux/security.h

Programming tips

#6: Use bpf_trace_printk to debug

- cat here for the output:
  /sys/kernel/debug/tracing/trace_pipe

```
  irqbalance-814     [005] d...1  6696.135995: bpf_trace_printk: Hello, World!\n
  irqbalance-814     [005] d...1  6696.136010: bpf_trace_printk: Hello, World!\n
  irqbalance-814     [005] d...1  6696.136028: bpf_trace_printk: Hello, World!\n
  irqbalance-814     [005] d...1  6696.136043: bpf_trace_printk: Hello, World!\n
  irqbalance-814     [005] d...1  6696.136060: bpf_trace_printk: Hello, World!\n
systemd-oomd-739     [002] d...1  6696.292693: bpf_trace_printk: Hello, World!\n
systemd-oomd-739     [002] d...1  6696.292762: bpf_trace_printk: Hello, World!\n
systemd-oomd-739     [002] d...1  6696.292816: bpf_trace_printk: Hello, World!\n
systemd-oomd-739     [002] d...1  6696.293014: bpf_trace_printk: Hello, World!\n
systemd-oomd-739     [002] d...1  6696.293610: bpf_trace_printk: Hello, World!\n
systemd-oomd-739     [002] d...1  6696.542488: bpf_trace_printk: Hello, World!\n
```

libbpf also has a wrapper bpf_printk that is better:

- nice blogpost: https://nakryiko.com/posts/bpf-tips-printk/

Programming tips

#7: When in doubt look at tracee

- https://github.com/aquasecurity/tracee/blob/main/pkg/ebpf/c/tracee.bpf.c

Lots of useful macros, helper functions and programming guidelines

Code example

BCC example - no CO-RE

- easier
- CO-RE still not fully supported
- i have more experience on this

How to Install:

- https://github.com/iovisor/bcc/blob/master/INSTALL.md
- Install from source is generally better

Check if it works:

- run this: bcc/examples/tracing/hello_fields.py

BCC example - no CO-RE

```
sudo: upython3: command not found
capo80@deep-purple:~/Desktop/eBPF/bcc/examples/tracing$ sudo python3 hello_fields.py
TIME(s)            COMM             PID    MESSAGE
867.721875000      gsd-media-keys   2657   Hello, World!
867.724049000      gsd-media-keys   8993   Hello, World!
867.851915000      gnome-terminal-  4925   Hello, World!
867.854413000      bash             9002   Hello, World!
867.855335000      lesspipe         9004   Hello, World!
867.856218000      lesspipe         9004   Hello, World!
867.856383000      lesspipe         9006   Hello, World!
867.857492000      bash             9002   Hello, World!
871.164314000      gsd-media-keys   2657   Hello, World!
871.166437000      gsd-media-keys   9012   Hello, World!
871.291167000      gnome-terminal-  4925   Hello, World!
871.294080000      bash             9020   Hello, World!
871.295055000      lesspipe         9021   Hello, World!
871.296106000      lesspipe         9021   Hello, World!
871.296292000      lesspipe         9023   Hello, World!
871.297581000      bash             9020   Hello, World!
872.017271000      firefox          3369   Hello, World!
^Cc
```

@social

- Linkedin:

  https://www.linkedin.com/in/pasquale-caporaso-4a19b41aa

- Mail: pasquale.caporaso@cnit.it
- Telegram: @Capo80

- Github:

  https://github.com/Capo80