Fixing a Memory Forensics Blind Spot: Linux Kernel Tracing

Andrew Case / Golden G. Richard III

bpftrace/eBPF Tools

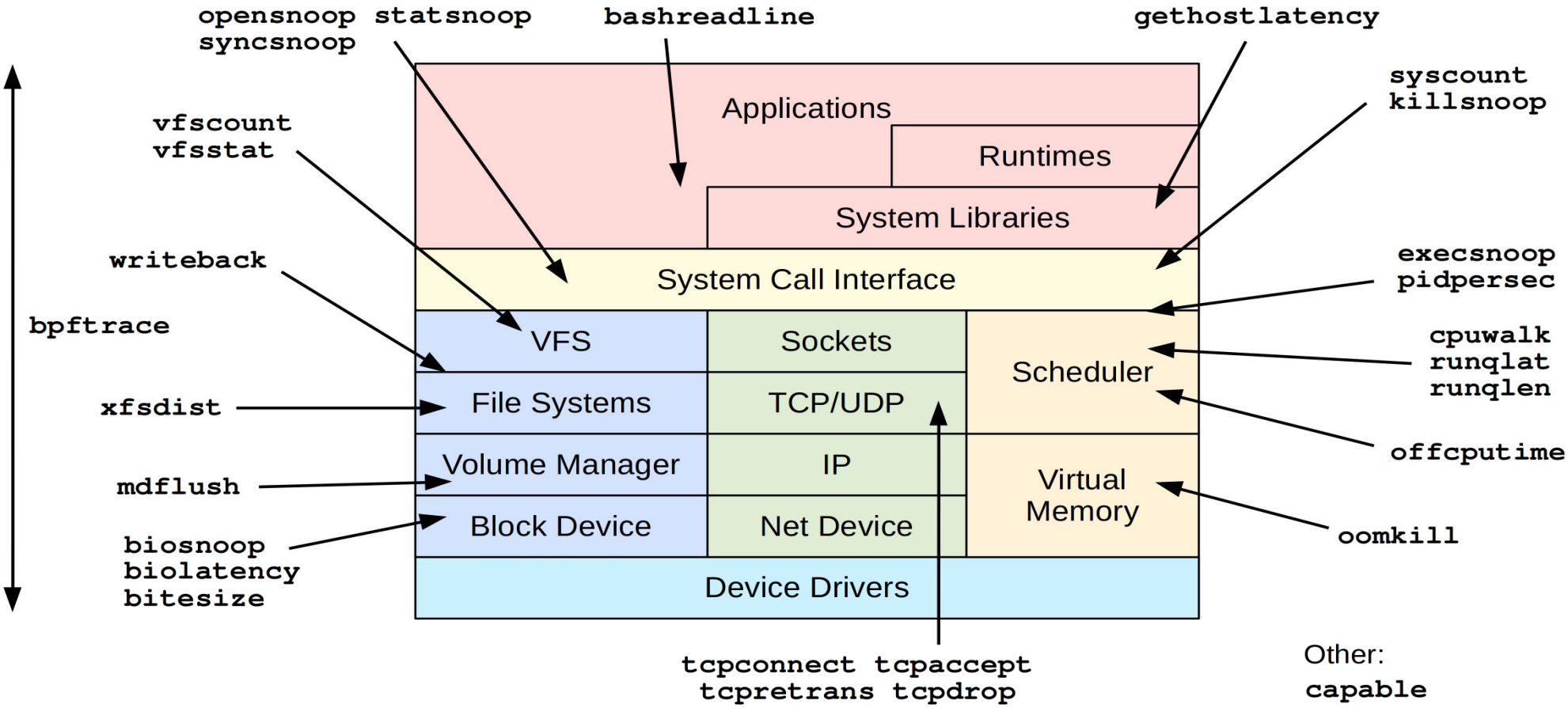


Diagram by Brendan Gregg, early 2019. https://github.com/iovisor/bpftrace

https://www.brendangregg.com/ebpf.html

```
# ./opensnoop.bt
Attaching 3 probes...
Tracing open syscalls... Hit Ctrl-C to end.
PID    COMM               FD ERR PATH
2440   snmp-pass           4   0 /proc/cpuinfo
2440   snmp-pass           4   0 /proc/stat
25706  ls                  3   0 /etc/ld.so.cache
25706  ls                  3   0 /lib/x86_64-linux-gnu/libselinux.so.1
25706  ls                  3   0 /lib/x86_64-linux-gnu/libc.so.6
25706  ls                  3   0 /lib/x86_64-linux-gnu/libpcre.so.3
25706  ls                  3   0 /lib/x86_64-linux-gnu/libdl.so.2
25706  ls                  3   0 /lib/x86_64-linux-gnu/libpthread.so.0
```

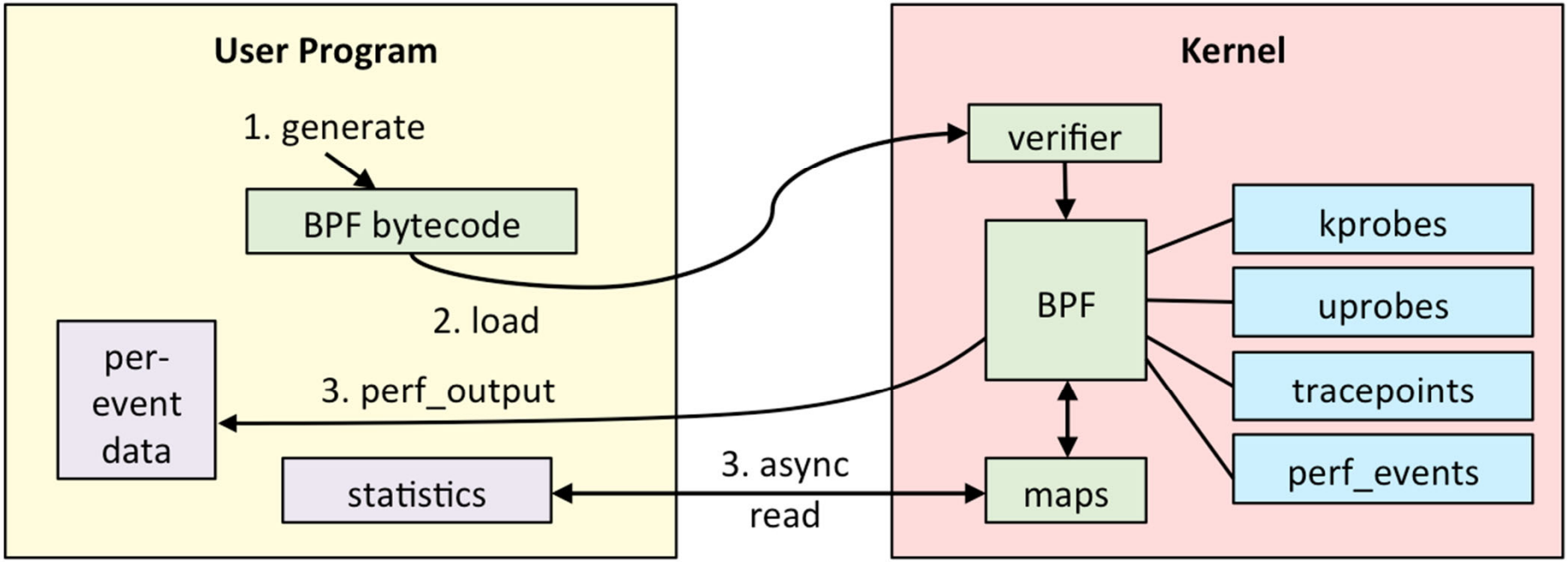
```
# ./bashreadline.bt
Attaching 2 probes...
Tracing bash commands... Hit Ctrl-C to end.
TIME      PID    COMMAND
06:40:06  5526   df -h
06:40:09  5526   ls -l
06:40:18  5526   echo hello bpftrace
```

```
# ./execsnoop.bt
Attaching 3 probes...
TIME(ms)    PID    ARGS
2460        3466   ls --color=auto -lh execsnoop.bt execsnoop.bt.0
3996        3467   man ls
4005        3473   preconv -e UTF-8
4005        3473   preconv -e UTF-8
4005        3473   preconv -e UTF-8
4005        3473   preconv -e UTF-8
4005        3473   preconv -e UTF-8
4005        3474   tbl
4005        3474   tbl
```

```
BEGIN
{
        printf("%-10s %-5s %s\n", "TIME(ms)", "PID", "ARGS");
}

tracepoint:syscalls:sys_enter_exec*
{
        printf("%-10u %-5d ", elapsed / 1e6, pid);
        join(args->argv);
}
```

- Example output and tools: https://github.com/iovisor/bpftrace/tree/master/tools



http://www.brendangregg.com/ebpf.html

eBPF is Widely Used in Production

- Netflix
  - https://netflixtechblog.com/how-netflix-uses-ebpf-flow-logs-at-scale-for-network-insight-e3ea997dca96
- Google
  - https://cloud.google.com/blog/products/containers-kubernetes/bringing-ebpf-and-cilium-to-google-kubernetes-engine
- Facebook
  - https://www.infoq.com/presentations/facebook-google-bpf-linux-kernel/
- Capital One, Adobe, Cloudflare, Digital Ocean
  - https://ebpf.io/summit-2020/

Prevalence in Distributions

- Volexity maintains a database of kernels released by common distributions
  - Debian, Ubuntu, CentOS, Red Hat, SuSe, AWS, …
- There is also an API for querying metadata of these kernels, including configuration options
- At the time of testing, the database held 14,762 kernels

Prevalence in Distributions Results

- The bpftrace documentation lists which kernel options are required for all tracing features and that the version must be >= 4.9
- Of the 14,762 kernels, 5,386 were >= 4.9
- Initially run showed 82.9% of kernels had all features
  - Analysis of missing ones showed new features were added in 4.11
  - We then updated the script to skip these features on 4.9 and 4.10
- Final run showed 5,191 of the 5,368 (96.3%) of the kernels had all features
  - The few still missing are an Ubuntu variant (kvm) that would not be seen in normal production environments

ftrace

- Initially, a function tracing interface for kernel modules
- Now also supports tracing events
- There is also a userland interface

```
static struct ftrace_hook demo_hooks[] = {
        HOOK("sys_clone",  fh_sys_clone,  &real_sys_clone),
        HOOK("sys_execve", fh_sys_execve, &real_sys_execve),
};
```

```
err = ftrace_set_filter_ip(&hook->ops, hook->address, 0, 0);
if (err) {
        pr_debug("ftrace_set_filter_ip() failed: %d\n", err);
        return err;
}

err = register_ftrace_function(&hook->ops);
if (err) {
        pr_debug("register_ftrace_function() failed: %d\n", err);
        ftrace_set_filter_ip(&hook->ops, hook->address, 1, 0);
        return err;
}
```

https://jvns.ca/blog/2017/03/19/getting-started-with-ftrace/

https://github.com/ilammy/ftrace-hook

```
$ python vol.py -f data.lime --profile=LinuxRKDevx64 linux_ftrace
Volatility Foundation Volatility Framework 2.6
Offset             Function           Symbol                          Traces
------------------ ------------------ ------------------------------  ----------------
0xffffffffc05c2160 0xffffffffc05c0000 fh_ftrace_thunk [ftrace_hook]   __x64_sys_execve
0xffffffffc05c20a0 0xffffffffc05c0000 fh_ftrace_thunk [ftrace_hook]   __x64_sys_clone
```

tracepoints

- Allow for hooking functions that define tracepoints
- 1,000+ functions define these on production kernels
- Used POC to hook *mm_probe_page_free* and *mm_probe_page_alloc*

```
struct tracepoints_table interests[] = {
    {.name = "mm_page_free", .fct = probe_mm_page_free},
    {.name = "mm_page_alloc", .fct = probe_mm_page_alloc}, };

static void lookup_tracepoints(struct tracepoint *tp, void *ignore) {
   int i;
   FOR_EACH_INTEREST(i) {
      if (strcmp(interests[i].name, tp->name) == 0) interests[i].value = tp;
   }
}

int __init my_trace_init(void) {
    for_each_kernel_tracepoint(lookup_tracepoints, NULL);

    FOR_EACH_INTEREST(i) {
       ...

       tracepoint_probe_register(interests[i].value, interests[i].fct, NULL);
    }

    ...
```

https://hugoguiroux.blogspot.com/2016/01/hooking-into-kernel-real-time-code.html

```
$ python vol.py -f data.lime --profile=Linuxthisx64 linux_tracepoints
Volatility Foundation Volatility Framework 2.6
Offset             Tracepoint    Hooks
------------------ ------------  --------------------------
0xffffffff9e4fe4c0 mm_page_free  probe_mm_page_free [my_module]
0xffffffff9e4fe440 mm_page_alloc probe_mm_page_alloc [my_module]
```

kprobe – kernel interface

- Allows for hooking kernel functions by name or address
- *pre_handler* runs before first instruction
- *post_handler* runs after first instruction
- *fault_handler* runs if an initial fault

```
static struct kprobe kp = {
    .symbol_name        = "proc_sys_open",

};


static int __init kprobe_init(void)
{
    kp.pre_handler = handler_pre;
    kp.post_handler = handler_post;
     kp.fault_handler = handler_fault;

    ret = register_kprobe(&kp);

    if (ret < 0) {
       printk(KERN_INFO "register_kprobe failed, returned %d\n", ret);
       return ret;
      }

    printk(KERN_INFO "Planted kprobe at %p\n", kp.addr);
```

https://github.com/spotify/linux/blob/master/samples/kprobes/kprobe_example.c

```
# python vol.py -f data.lime --profile=LinuxRKDevx64 linux_kprobes
Volatility Foundation Volatility Framework 2.6
Target Address     Target Symbol Pre Handler        Pre Handler Symbol
------------------ ------------- ------------------ ------------------
0xffffffff9d6ecc50 proc_sys_open 0xffffffffc0785034 handler_pre [kprobe_example]
```

kprobe – userland interface

```
# echo 'p:testopen do_sys_open filename=+0(%si):string' >> /sys/kernel/debug/tracing/kprobe_events
# echo 1 > /sys/kernel/debug/tracing/events/kprobes/testopen/enable
# cat /tmp/this_file_does_not_exist
# grep does_not_exist /sys/kernel/debug/tracing/trace
cat-26136 testopen: (do_sys_open+0x0/0x210) filename="/tmp/this_file_does_not_exist"
```

Trace Event Handlers

```
root@bob:/sys/kernel/debug/tracing/events# find . -name format | wc -l
1529
```

```
root@bob:/sys/kernel/debug/tracing/events/syscalls# cat sys_enter_open/format
name: sys_enter_open
ID: 602
format:
        field:unsigned short common_type;  offset:0;     size:2;       signed:0;
        field:unsigned char common_flags;  offset:2;     size:1;       signed:0;
        field:unsigned char common_preempt_count; offset:3;     size:1;       signed:0;
        field:int common_pid;       offset:4;     size:4;       signed:1;

        field:int __syscall_nr;     offset:8;     size:4;       signed:1;
        field:const char * filename;       offset:16;    size:8;       signed:0;
        field:int flags;     offset:24;    size:8;       signed:0;
        field:umode_t mode;  offset:32;    size:8;       signed:0;

print fmt: "filename: 0x%08lx, flags: 0x%08lx, mode: 0x%08lx", ((unsigned long)(REC->filename)), ((unsigned long)(REC->flags)),
        ((unsigned long)(REC->mode))
```

https://www.kernel.org/doc/html/v4.18/trace/events.html

```
# python vol.py -f data.lime --profile=Linuxthisx64 linux_kprobes
Volatility Foundation Volatility Framework 2.6.1
Target Address     Target Symbol Pre Handler        Pre Handler Symbol
------------------ ------------- ------------------ ------------------
0xffffffff9d663120 do_sys_open   0xffffffff9d57c460 kprobe_dispatcher


# python vol.py -f data.lime --profile=LinuxRKDevx64 linux_trace_events
Volatility Foundation Volatility Framework 2.6.1
Target Symbol Format
------------- ------
testopen      "(%lx) filename=\"%s\"", REC->__probe_ip, __get_str(filename)
```

kretprobe

- Allows hooking a function's entry and exit
- Legit use is tracking life of a function call
- Re-uses kprobe infrastructure for entry hooking
- Can be used to *modify* a function's return value

```
static char func_name[NAME_MAX] = "netlink_sendskb";

static struct kretprobe my_kretprobe = {
        .handler                = ret_handler,
        .entry_handler          = entry_handler,
        .data_size              = sizeof(struct my_data),
        .maxactive              = 1,
};

static int __init kretprobe_init(void)
{
        int ret;

        my_kretprobe.kp.symbol_name = func_name;
        ret = register_kretprobe(&my_kretprobe);

...
```

https://github.com/spotify/linux/blob/master/samples/kprobes/kretprobe_example.c

```
$ python vol.py -f data.lime --profile=Linuxuxnewx64 linux_kprobes
Volatility Foundation Volatility Framework 2.6.1
Target             Kernel Symbol            Target Symbol            Pre Handler        Pre Handler Module Pre Handler Symbol
------------------ ------------------------ ------------------------ ------------------ ------------------ ------------------------
0xffffffffc09a6000 netlink_sendskb          netlink_sendskb          0xffffffffb6770f70 kernel             pre_handler_kretprobe
```

```
$ python vol.py -f data.lime --profile=Linuxuxnewx64 linux_kretprobes
Volatility Foundation Volatility Framework 2.6.1
Target             Pre Handler        Module               Symbol        Post Handler       Module             Symbol
------------------ ------------------ -------------------- ------------- ------------------ ------------------ ------
netlink_sendskb    0xffffffffc09a4000 kretprobe_example    entry_handler 0xffffffffc09a4040 kretprobe_example ret_handler
```

eBPF

- Allows userland to run programs in kernel space
- Programs are verified before being allowed to execute
- eBPF heavily relies on previously described facilities

```
BEGIN
{
        printf("%-10s %-5s %s\n", "TIME(ms)", "PID", "ARGS");
}

tracepoint:syscalls:sys_enter_exec*
{
        printf("%-10u %-5d ", elapsed / 1e6, pid);
        join(args->argv);
}
```

```
# ./execsnoop.bt
Attaching 3 probes...
TIME(ms)   PID   ARGS
1220640127 6238  cat /etc/passwd
```

https://www.infoq.com/articles/gentle-linux-ebpf-introduction/

```
$ python vol.py -f data.lime --profile=LinuxRKDevx64 linux_ebpf
Volatility Foundation Volatility Framework 2.6
Address            Name              Type
------------------ ----------------- ------------------------
0xffffc046403f3000 BEGIN             BPF_PROG_TYPE_KPROBE
0xffffc046403f5000 sys_enter_execv   BPF_PROG_TYPE_TRACEPOINT
0xffffc046403f7000 sys_enter_execv   BPF_PROG_TYPE_TRACEPOINT

$ python vol.py -f data.lime --profile=LinuxRKDevx64 linux_tracepoints
Volatility Foundation Volatility Framework 2.6
Offset             Tracepoint Hooks
------------------ ---------- -----------------
0xffffffff9700e420 sys_enter  perf_syscall_enter

$ python vol.py -f data.lime --profile=LinRKDevx64 linux_perf_events_ebpf
Volatility Foundation Volatility Framework 2.6
PID  Process Name Program Name    Program Address
---- ------------ --------------  ------------------
1109 bpftrace     BEGIN           0xffffc046403f3000
1109 bpftrace     sys_enter_execv 0xffffc046403f5000
1109 bpftrace     sys_enter_execv 0xffffc046403f7000
```

Wrapup

Upcoming talks on Offensive Uses of eBPF

With Friends Like eBPF, Who Needs Enemies? – Black Hat

Warping Reality - creating and countering the next generation of Linux rootkits using eBPF - DefCon

A huge thank you to Gus Moreira for his feedback and plugin development!

Questions/Comments?

- andrew@dfir.org
- @attrc
- https://www.volexity.com

- golden@cct.lsu.edu
- @nolaforensix
- https://www.cct.lsu.edu/~golden/