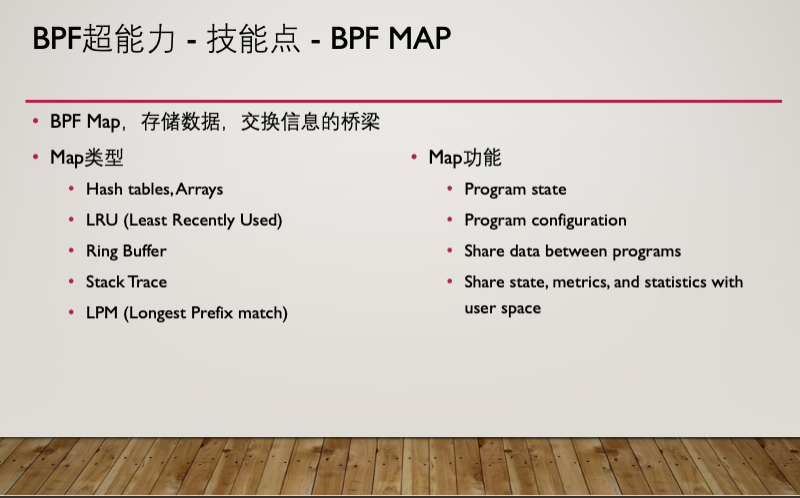
第一个核心技能点是**「BPF Hooks」**，即BPF钩子，也就是在内核中，哪些地方可以加载BPF程序，在目前的Linux内核中已经有了近10种的钩子，如下所示：

* kernel functions (kprobes)
* userspace functions (uprobes)
* system calls
* fentry/fexit
* Tracepoints
* network devices (tc/xdp)
* network routes
* TCP congestion algorithms
* sockets (data level)



// SPDX-License-Identifier: GPL-2.0 OR BSD-3-Clause

/\* Copyright (c) 2020 Facebook \*/

#include "vmlinux.h"

#include <bpf/bpf\_helpers.h>

#include <bpf/bpf\_tracing.h>

#include <bpf/bpf\_core\_read.h>

#include "bootstrap.h"

char LICENSE[] SEC("license") = "Dual BSD/GPL";

struct {

\_\_uint(type, BPF\_MAP\_TYPE\_HASH);

\_\_uint(max\_entries, 8192);

\_\_type(key, pid\_t);

\_\_type(value, u64);

} exec\_start SEC(".maps");

struct {

\_\_uint(type, BPF\_MAP\_TYPE\_RINGBUF);

\_\_uint(max\_entries, 256 \* 1024);

} rb SEC(".maps");

const volatile unsigned long long min\_duration\_ns = 0;

SEC("tp/sched/sched\_process\_exec")

int handle\_exec(struct trace\_event\_raw\_sched\_process\_exec \*ctx)

{

struct task\_struct \*task;

unsigned fname\_off;

struct event \*e;

pid\_t pid;

u64 ts;

/\* remember time exec() was executed for this PID \*/

pid = bpf\_get\_current\_pid\_tgid() >> 32;

ts = bpf\_ktime\_get\_ns();

bpf\_map\_update\_elem(&exec\_start, &pid, &ts, BPF\_ANY);

/\* don't emit exec events when minimum duration is specified \*/

if (min\_duration\_ns)

return 0;

/\* reserve sample from BPF ringbuf \*/

e = bpf\_ringbuf\_reserve(&rb, sizeof(\*e), 0);

if (!e)

return 0;

/\* fill out the sample with data \*/

task = (struct task\_struct \*)bpf\_get\_current\_task();

e->exit\_event = false;

e->pid = pid;

e->ppid = BPF\_CORE\_READ(task, real\_parent, tgid);

bpf\_get\_current\_comm(&e->comm, sizeof(e->comm));

fname\_off = ctx->\_\_data\_loc\_filename & 0xFFFF;

bpf\_probe\_read\_str(&e->filename, sizeof(e->filename), (void \*)ctx + fname\_off);

/\* successfully submit it to user-space for post-processing \*/

bpf\_ringbuf\_submit(e, 0);

return 0;

}

SEC("tp/sched/sched\_process\_exit")

int handle\_exit(struct trace\_event\_raw\_sched\_process\_template\* ctx)

{

struct task\_struct \*task;

struct event \*e;

pid\_t pid, tid;

u64 id, ts, \*start\_ts, duration\_ns = 0;

/\* get PID and TID of exiting thread/process \*/

id = bpf\_get\_current\_pid\_tgid();

pid = id >> 32;

tid = (u32)id;

/\* ignore thread exits \*/

if (pid != tid)

return 0;

/\* if we recorded start of the process, calculate lifetime duration \*/

start\_ts = bpf\_map\_lookup\_elem(&exec\_start, &pid);

if (start\_ts)

duration\_ns = bpf\_ktime\_get\_ns() - \*start\_ts;

else if (min\_duration\_ns)

return 0;

bpf\_map\_delete\_elem(&exec\_start, &pid);

/\* if process didn't live long enough, return early \*/

if (min\_duration\_ns && duration\_ns < min\_duration\_ns)

return 0;

/\* reserve sample from BPF ringbuf \*/

e = bpf\_ringbuf\_reserve(&rb, sizeof(\*e), 0);

if (!e)

return 0;

/\* fill out the sample with data \*/

task = (struct task\_struct \*)bpf\_get\_current\_task();

e->exit\_event = true;

e->duration\_ns = duration\_ns;

e->pid = pid;

e->ppid = BPF\_CORE\_READ(task, real\_parent, tgid);

e->exit\_code = (BPF\_CORE\_READ(task, exit\_code) >> 8) & 0xff;

bpf\_get\_current\_comm(&e->comm, sizeof(e->comm));

/\* send data to user-space for post-processing \*/

bpf\_ringbuf\_submit(e, 0);

return 0;

}