bpf: Fixed tailcall issues

Leon Hwang

2025/04/19

中国·西安

Profile

- Author of [eBPF Talk]
- Contributor of pwru, eCapture, drgn
- Creator of bpfsnoop
- Independent bpf subsystem contributor
- BPF contributions: https://bit.ly/4d9eOWa
- Blog: https://blog.leonhw.com
- GitHub: https://github.com/Asphaltt
- bpfsnoop: https://bpfsnoop.com

Agenda

1. Fixed tailcall infinite loop issue caused by trampoline
2. Fixed tailcall hierarchy issue
3. Fixed crash caused by freplace + prog_array
4. Fixed tailcall infinite loop issue caused by freplace

To explain each issue:

- How did I find it?
- What was it?
- How to fix it?

第三届 eBPF开发者大会

www.ebpftravel.com

Prerequisite:

tailcall in bpf2bpf on x86

中国·西安

0: tailcall in bpf2bpf on x86

- How does tailcall in bpf2bpf work?
  - bpf2bpf means a bpf prog calls another bpf prog directly.
  - The callee bpf prog is named subprog.
  - tailcall in bpf2bpf is the subprog has `bpf_tail_call()`.
- `tail_call_cnt` is used to limit the total times of calling `bpf_tail_call()`.
  - `tail_call_cnt` is propagated by %rax between the caller and the callee.

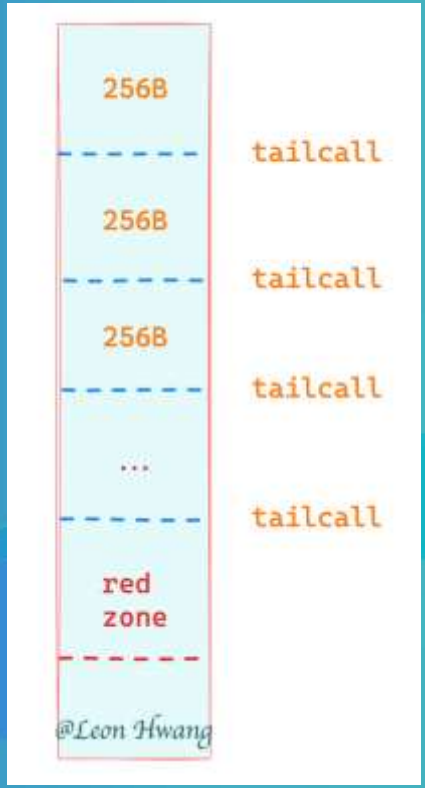
① Fixed tailcall infinite loop issue caused by trampoline

① Fixed tailcall infinite loop issue caused by trampoline

- How did I find it?

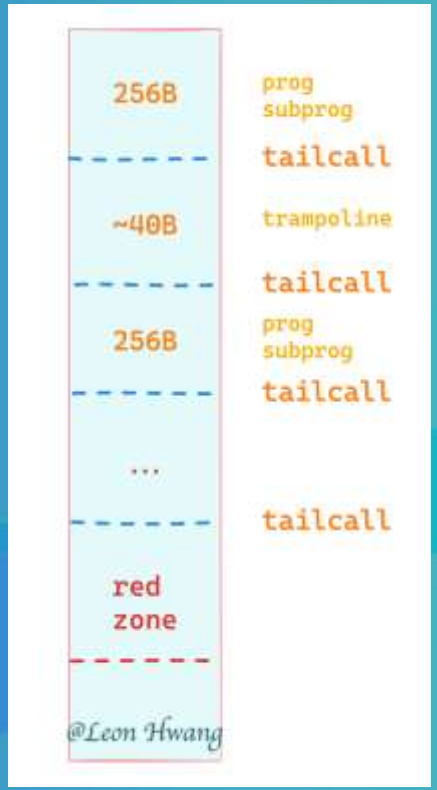
It was found when I studied the 8KiB stack space limit.

- When a prog has tailcall, its stack space limits to 256B.
- When there are 32 tailcalls, the total consumed stack space will be 8Kib + 512B at most.



① Fixed tailcall infinite loop issue caused by trampoline

- How did I find it?
- What if increment the consumed stack space?
- What I tried:
  - Run tailcall in subprog.
  - Tail callee is the entry prog.
  - Trace the subprog with fexit in order to consume more stack with trampoline.

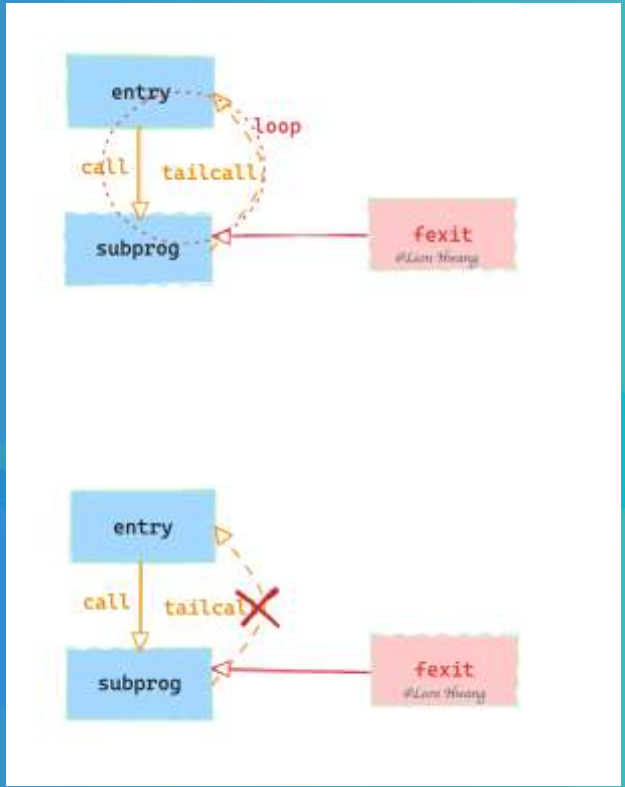


① Fixed tailcall infinite loop issue caused by trampoline

- What was it?

- What did I get?
  - A kernel crash at that time.

- Try again?
  - No tailcall happens => tailcall is skipped.

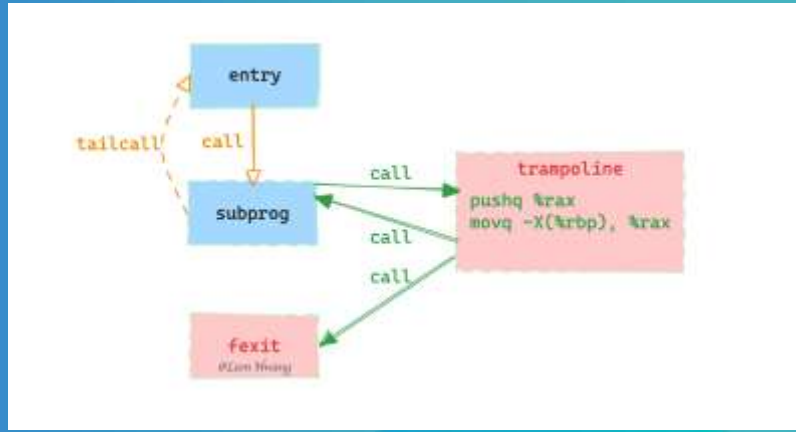


① Fixed tailcall infinite loop issue caused by trampoline

- How to fix it?

Fixed it on x86:

- Propagate `tail_call_cnt` through the trampoline.



On x86: `tail_call_cnt` is propagated from caller to callee by %rax.

Patch: bpf, x64: Fix tailcall infinite loop

② Fixed tailcall hierarchy issue

② Fixed tailcall hierarchy issue

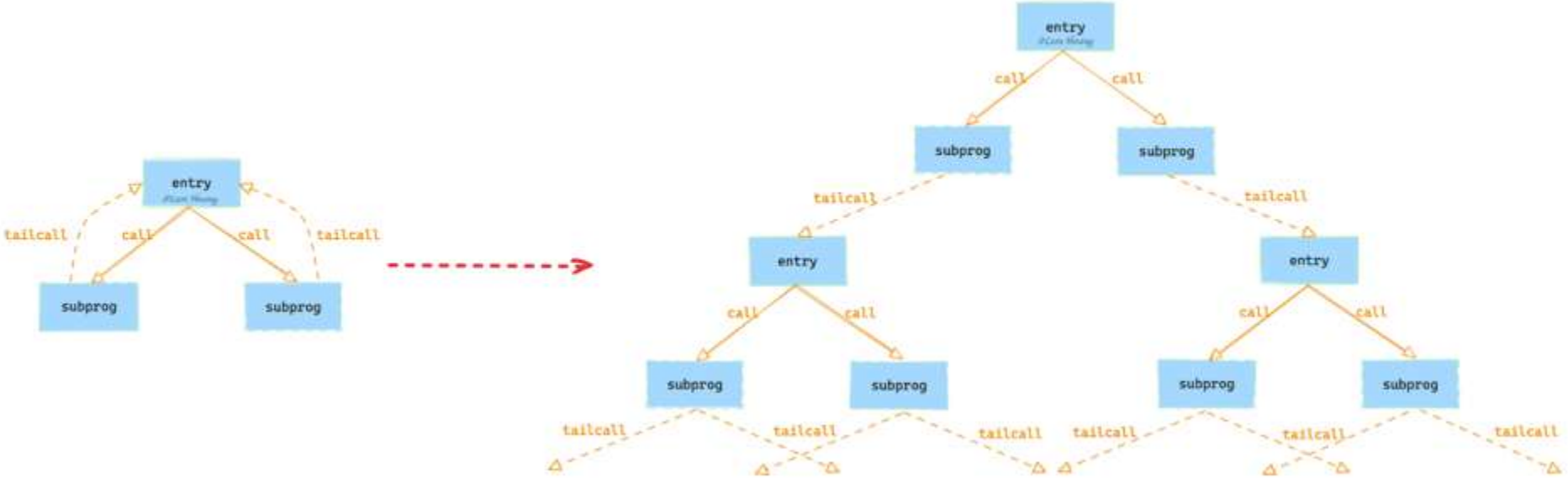
- How did I find it?

It was confirmed by me while discussing the previous “infinite loop” issue with Ilya Leoshkevich, s390x JIT maintainer.

Discussion: should we load it back?

② Fixed tailcall hierarchy issue

- What was it?



② Fixed tailcall hierarchy issue

- How to fix it?

Propagate `tail_call_cnt` by pointer.

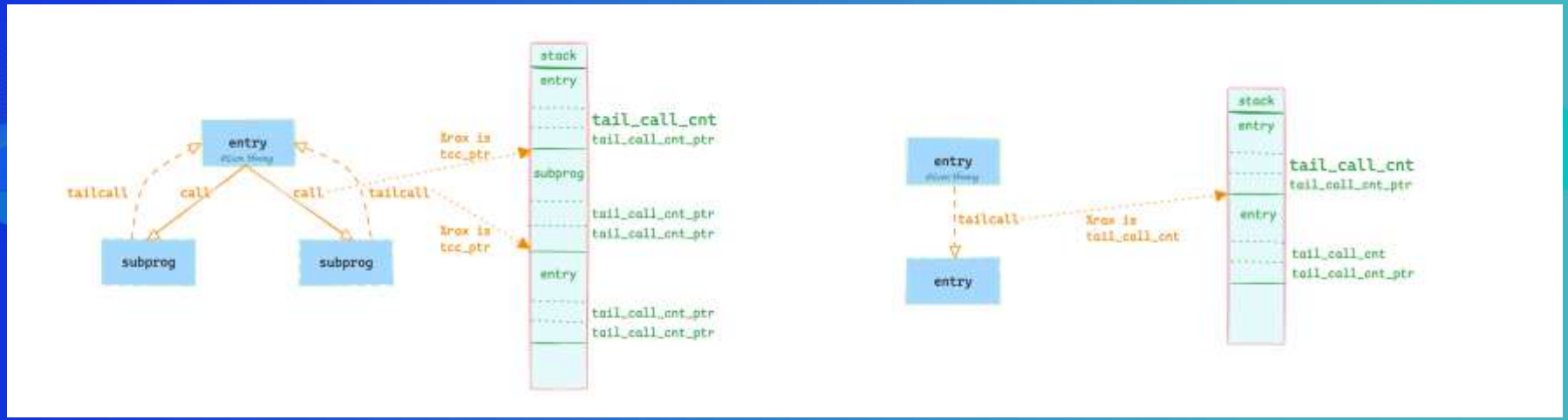
- Save `tail_call_cnt` on stack of entry prog.
- Save `tail_call_cnt_ptr`, too.
- `popq %rax` twice before tailcall's `jmp`.
- Propagate `tail_call_cnt_ptr` to subprogs.
- `pushq %rax` twice in subprogs' prologue.
- In main progs' prologue:
  - If %rax is ptr, `pushq %rax` twice.
  - Else, %rax is `tail_call_cnt`.

② Fixed tailcall hierarchy issue

- How to fix it?

Make `tail_call_cnt` as a runtime global variable.

Patch: bpf: Fix tailcall hierarchy



③ Fixed crash caused by freplace + prog_array

③ Fixed crash caused by freplace + prog_array

- How did I find it?

It was found when I did the POC of the next issue “tailcall infinite loop issue caused by freplace”.

It was easy to trigger the crash:

- Attach the freplace prog to target.
- Update prog_array map with the freplace prog.

```
BUG: kernel NULL pointer dereference, address: 0000000000000004
#PF: supervisor read access in kernel mode
#PF: error_code(0x0000) - not-present page
PGD 0 P4D 0
Oops: 0000 [#1] PREEMPT SMP NOPTI
CPU: 2 PID: 788148 Comm: test_progs Not tainted 6.8.0-31-generic
Hardware name: VMware, Inc. VMware20,1/440BX Desktop Reference Pl
RIP: 0010:bpf_prog_map_compatible+0x2a/0x140
Code: 0f 1f 44 00 00 55 48 89 e5 41 57 41 56 49 89 fe 41 55 41 54
RSP: 001B:ffffb2e0b0fd7ce0 EFLAGS: 00010246
RAX: 0000000000000000 RBX: ffffb2e0807c1000 RCX: 0000000000000000
RDX: 0000000000000000 RSI: ffffb2e0807c1000 RDI: ffff990290259e00
RBP: ffffb2e0b0fd7d08 R08: 0000000000000000 R09: 0000000000000000
R10: 0000000000000000 R11: 0000000000000000 R12: ffff990290259e00
R13: 000000000000001c R14: ffff990290259e00 R15: ffff99028e29c400
FS:  00007b82cbc28140(0000) GS:ffff9903b3f00000(0000) knlGS:00000
CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
CR2: 0000000000000004 CR3: 0000000101286002 CR4: 0000000003706f0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 00000000fffe0ff0 DR7: 0000000000000400
Call Trace:
 <TASK>
 ? show_regs+0x6d/0x80
 ? __die+0x24/0x80
 ? page_fault_oops+0x99/0x1b0
 ? do_user_addr_fault+0x2ee/0x6b0
 ? exc_page_fault+0x83/0x1b0
 ? asm_exc_page_fault+0x27/0x30
 ? bpf_prog_map_compatible+0x2a/0x140
 prog_fd_array_get_ptr+0x2c/0x70
 bpf_fd_array_map_update_elem+0x37/0x130
 bpf_map_update_value+0x1d3/0x260
 map_update_elem+0x1fa/0x360
 __sys_bpf+0x54c/0xa10
 __x64_sys_bpf+0x1a/0x30
 x64_sys_call+0x1936/0x25c0
 do_syscall_64+0x7f/0x180
 ? do_syscall_64+0x8c/0x180
 ? do_syscall_64+0x8c/0x180
 ? irqentry_exit+0x43/0x50
 ? common_interrupt+0x54/0xb0
 entry_SYSCALL_64_after_hwframe+0x73/0x7b
```

③ Fixed crash caused by freplace + prog_array

- What was it?

It was triggered because `prog->aux->dst_prog = NULL`.

However, the crash was not prevented by me, by Tengda Wu instead.
But he did not fix this crash thoroughly.

```
static int bpf_tracing_prog_attach(struct bpf_prog *prog,
                                   int tgt_prog_fd,
                                   u32 btf_id,
                                   u64 bpf_cookie,
                                   struct bpf_verifier_log *log)
{
        prog->aux->dst_prog = NULL;
        prog->aux->dst_trampoline = NULL;
        mutex_unlock(&prog->aux->dst_mutex);

        return bpf_link_settle(&link_primer);
}

static inline enum bpf_prog_type resolve_prog_type(const struct bpf_prog *prog)
{
        return prog->type == BPF_PROG_TYPE_EXT ?
                prog->aux->dst_prog->type : prog->type;
}
```

③ Fixed crash caused by freplace + prog_array

- How to fix it?

Here's Tengda's patch:

Patch: bpf: Fix null-pointer-deref in resolve_prog_type()

Here's my patch:

Patch: bpf: Fix updating attached freplace prog to prog_array map

```
diff --git a/include/linux/bpf_verifier.h b/include/linux/bpf_verifier.h
index e4070fb02b11..ff2a6cdb1fa3 100644
--- a/include/linux/bpf_verifier.h
+++ b/include/linux/bpf_verifier.h
@@ -846,7 +846,7 @@ static inline u32 type_flag(u32 type)
 /* only use after check_attach_btf_id() */
 static inline enum bpf_prog_type resolve_prog_type(const struct bpf_prog *prog)
 {
-       return prog->type == BPF_PROG_TYPE_EXT ?
+       return (prog->type == BPF_PROG_TYPE_EXT && prog->aux->dst_prog) ?
               prog->aux->dst_prog->type : prog->type;
 }
```

```
diff --git a/include/linux/bpf_verifier.h b/include/linux/bpf_verifier.h
index 5cea15c81b8a8..bfd093ac333f2 100644
--- a/include/linux/bpf_verifier.h
+++ b/include/linux/bpf_verifier.h
@@ -874,8 +874,8 @@ static inline u32 type_flag(u32 type)
 /* only use after check_attach_btf_id() */
 static inline enum bpf_prog_type resolve_prog_type(const struct bpf_prog *prog)
 {
-       return (prog->type == BPF_PROG_TYPE_EXT && prog->aux->dst_prog) ?
-              prog->aux->dst_prog->type : prog->type;
+       return (prog->type == BPF_PROG_TYPE_EXT && prog->aux->saved_dst_prog_type) ?
+              prog->aux->saved_dst_prog_type : prog->type;
 }
```

④ Fixed tailcall infinite loop issue caused by freplace

④ Fixed tailcall infinite loop issue caused by freplace

- How did I find it?

It was confirmed by my mind game: can trigger tailcall infinite loop issue by freplace + prog_array?

The answer was YES, after finishing its POC.

```
[   15.310490] BUG: TASK stack guard page was hit at (____ptrval____)
(stack is (____ptrval____)..(____ptrval____))
[   15.310490] Oops: stack guard page: 0000 [#1] PREEMPT SMP NOPTI
[   15.310490] CPU: 1 PID: 89 Comm: test_progs Tainted: G           OE
   6.10.0-rc6-g026dcdae8d3e-dirty #72
[   15.310490] Hardware name: QEMU Ubuntu 24.04 PC (i440FX + PIIX,
1996), BIOS 1.16.3-debian-1.16.3-2 04/01/2014
[   15.310490] RIP: 0010:bpf_prog_3a140cef239a4b4f_subprog_tail+0x14/0x53
[   15.310490] Code: cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
cc cc cc cc cc f3 0f 1e fa 0f 1f 44 00 00 0f 1f 00 55 48 89 e5 f3 0f 1e
fa <50> 58 53 41 55 48 89 fb 49 bd 00 2a 46 82 98 9c ff ff 48 89 df 4c
[   15.310490] RSP: 0018:ffffb500c0aa0000 EFLAGS: 00000202
[   15.310490] RAX: ffffb500c0aa0028 RBX: ffff9c98808b7e00 RCX:
0000000000000cb5
[   15.310490] RDX: 0000000000000000 RSI: ffff9c98822462a80 RDI:
ffff9c98808b7e00
[   15.310490] RBP: ffffb500c0aa0000 R08: 0000000000000000 R09:
0000000000000000
[   15.310490] R10: 0000000000000001 R11: 0000000000000000 R12:
ffffb500c01af000
[   15.310490] R13: ffffb500c01cd000 R14: 0000000000000000 R15:
0000000000000000
[   15.310490] FS:  00007f133b665140(0000) GS:ffff9c98bbd00000(0000)
knlGS:0000000000000000
[   15.310490] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[   15.310490] CR2: ffffb500c0a9fff8 CR3: 0000000102470000 CR4:
00000000000006f0
[   15.310490] Call Trace:
[   15.310490]  <#DF>
[   15.310490]  ? die+0x36/0x90
[   15.310490]  ? handle_stack_overflow+0x4d/0x60
[   15.310490]  ? exc_double_fault+0x117/0x1a0
[   15.310490]  ? asm_exc_double_fault+0x23/0x30
[   15.310490]  ? bpf_prog_3a140cef239a4b4f_subprog_tail+0x14/0x53
[   15.310490]  </#DF>
[   15.310490]  <TASK>
[   15.310490]  bpf_prog_85781a698094722f_entry+0x4c/0x64
[   15.310490]  bpf_prog_1c515f389a9059b4_entry2+0x19/0x1b
[   15.310490]  ...
[   15.310490]  bpf_prog_85781a698094722f_entry+0x4c/0x64
[   15.310490]  bpf_prog_1c515f389a9059b4_entry2+0x19/0x1b
[   15.310490]  bpf_test_run+0x210/0x370
[   15.310490]  ? bpf_test_run+0x128/0x370
[   15.310490]  bpf_prog_test_run_skb+0x388/0x7a0
[   15.310490]  __sys_bpf+0xdbf/0x2c40
[   15.310490]  ? clockevents_program_event+0x52/0xf0
[   15.310490]  ? lock_release+0xbf/0x290
[   15.310490]  __x64_sys_bpf+0x1e/0x30
[   15.310490]  do_syscall_64+0x68/0x140
[   15.310490]  entry_SYSCALL_64_after_hwframe+0x76/0x7e
[   15.310490] RIP: 0033:0x7f133b52725d
```

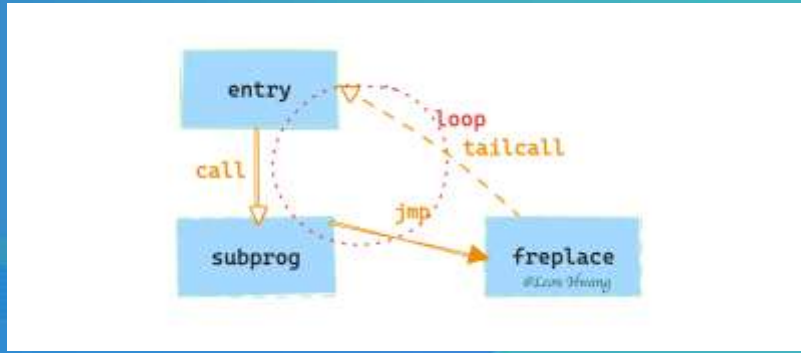
④ Fixed tailcall infinite loop issue caused by freplace

- What was it?

Like the right picture, the bpf progs can run forever until kernel crashes.

Reasons:

1. `entry` has no `tail_call_cnt` at run time.
2. `freplace` prog runs with zeroed `tail_call_cnt` for every time.



④ Fixed tailcall infinite loop issue caused by freplace

- How to fix it?

- Prevent updating prog_array with freplace prog.
- Prevent updating prog_array with prog who has been extended by freplace prog.
- Prevent attaching freplace prog to the prog who has been added into prog_array.

Patch: bpf: Fix tailcall infinite loop caused by freplace

⑤ tailcall issues detection

⑤ tailcall issues detection

- tailcall-issues is the tool to detect tailcall issues.

```
detection results:

issue:  invalid tailcallee
state:  not fixed

issue:  invalid loading offset of tail_call_cnt for bpf2bpf
state:  fixed

issue:  tailcall infinite loop caused by trampoline
state:  not fixed

issue:  tailcall hierarchy
state:  not fixed

issue:  panic caused by updating attached freplace prog to prog array
state:  not exists

issue:  tailcall infinite loop caused by freplace
state:  not exists

root@bpf-dev:~/Projects/leonhwang/tailcall-issues# uname -r
5.15.0-126-generic
```

```
detection results:

issue:  invalid tailcallee
state:  fixed

issue:  invalid loading offset of tail_call_cnt for bpf2bpf
state:  fixed

issue:  tailcall infinite loop caused by trampoline
state:  fixed

issue:  tailcall hierarchy
state:  not fixed

issue:  panic caused by updating attached freplace prog to prog array
state:  cannot detect

issue:  tailcall infinite loop caused by freplace
state:  not fixed

 ~/Projects/leonhwang/tailcall-issues on  issues uname -r
6.8.0-35-generic
```

⑥ tailcall in the future

⑥ tailcall in the future

- Refactor tailcall’s implementation on x86.
- Introduce tailcall tracer.

第三届 eBPF开发者大会

www.ebpftravel.com

Q&A

中国·西安