LINUX PLUMBERS CONFERENCE September 20-24, 2021

Merging the return caller infrastructures

Steven Rostedt

VMware Inc.

The return callers

- kretprobes
- function graph tracing
- BPF direct callers

kretprobe

```
# echo 'r:trylock _raw_spin_trylock ret=$retval' > /sys/kernel/tracing/kprobe_events
# trace-cmd start -e trylock
# trace-cmd show
# tracer: function
# tracer: nop
#
# entries-in-buffer/entries-written: 88/88   #P:8
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID     CPU#  ||||   TIMESTAMP  FUNCTION
#              | |         |   ||||      |         |
              bash-12314   [004] ...3 204275.659738: trylock: (dput+0x1ac/0x3a0 <- _raw_spin_trylock) ret=0x1
              bash-12314   [004] ...4 204275.659738: trylock: (dput+0x1c9/0x3a0 <- _raw_spin_trylock) ret=0x1
            <idle>-0       [002] d.s3 204275.659844: trylock: (note_gp_changes+0x54/0xa0 <- _raw_spin_trylock) ret=0x1
              bash-12314   [004] d.s2 204275.659861: trylock: (note_gp_changes+0x54/0xa0 <- _raw_spin_trylock) ret=0x1
            <idle>-0       [000] d.s2 204275.659861: trylock: (note_gp_changes+0x54/0xa0 <- _raw_spin_trylock) ret=0x0
            <idle>-0       [000] d.s3 204275.660845: trylock: (note_gp_changes+0x54/0xa0 <- _raw_spin_trylock) ret=0x1
            <idle>-0       [001] d.s2 204275.660847: trylock: (note_gp_changes+0x54/0xa0 <- _raw_spin_trylock) ret=0x0
            <idle>-0       [001] d.s3 204275.661842: trylock: (note_gp_changes+0x54/0xa0 <- _raw_spin_trylock) ret=0x1
            <idle>-0       [002] d.s3 204275.663871: trylock: (note_gp_changes+0x54/0xa0 <- _raw_spin_trylock) ret=0x1
```

function graph tracer

```
# trace-cmd start -p function_graph
# trace-cmd show
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 7)   1.002 us    |  rcu_idle_exit();
 7)   0.166 us    |  sched_idle_set_state();
 7)               |  cpuidle_reflect() {
 7)               |    menu_reflect() {
 7)   0.166 us    |      tick_nohz_idle_got_tick();
 7)   0.546 us    |    }
 7)   1.000 us    |  }
 7)   0.185 us    |  arch_cpu_idle_exit();
 7)               |  tick_nohz_idle_exit() {
 7)   0.450 us    |    ktime_get();
 7)   0.197 us    |    nr_iowait_cpu();
 7)               |    tick_nohz_restart_sched_tick() {
 7)   0.190 us    |      timer_clear_idle();
 7)   0.200 us    |      calc_load_nohz_stop();
 7)               |      hrtimer_cancel() {
 7)               |        hrtimer_try_to_cancel() {
 7)   0.255 us    |          hrtimer_active();
 7)               |          _raw_spin_lock_irqsave() {
```

BPF direct trampoline

- I don’t have an example
- BPF does it differently that kretprobe and function graph tracing.

BPF direct trampoline

- I don’t have an example
- BPF does it differently that kretprobe and function graph tracing.
- May not be able to consolidate with BPF trampolines

BPF direct trampoline

- I don’t have an example
- BPF does it differently that kretprobe and function graph tracing.
- May not be able to consolidate with BPF trampolines
- At least we might be able to merge kretprobe and function graph tracing

How it works

- kretprobes and function graph tracing

How it works

- kretprobes and function graph tracing
  - Hijack the return pointer

How it works

- kretprobes and function graph tracing
  - Hijack the return pointer
  - Saves it in a shadow stack

How it works

- kretprobes and function graph tracing
  - Hijack the return pointer
  - Saves it in a shadow stack
  - Replaces return with trampoline

How it works

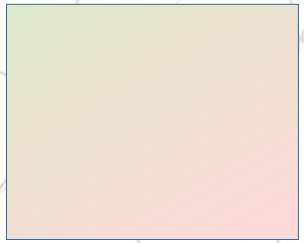
- kretprobes and function graph tracing
  - Hijack the return pointer
  - Saves it in a shadow stack
  - Replaces return with trampoline
  - trampoline returns back to original caller

How it works

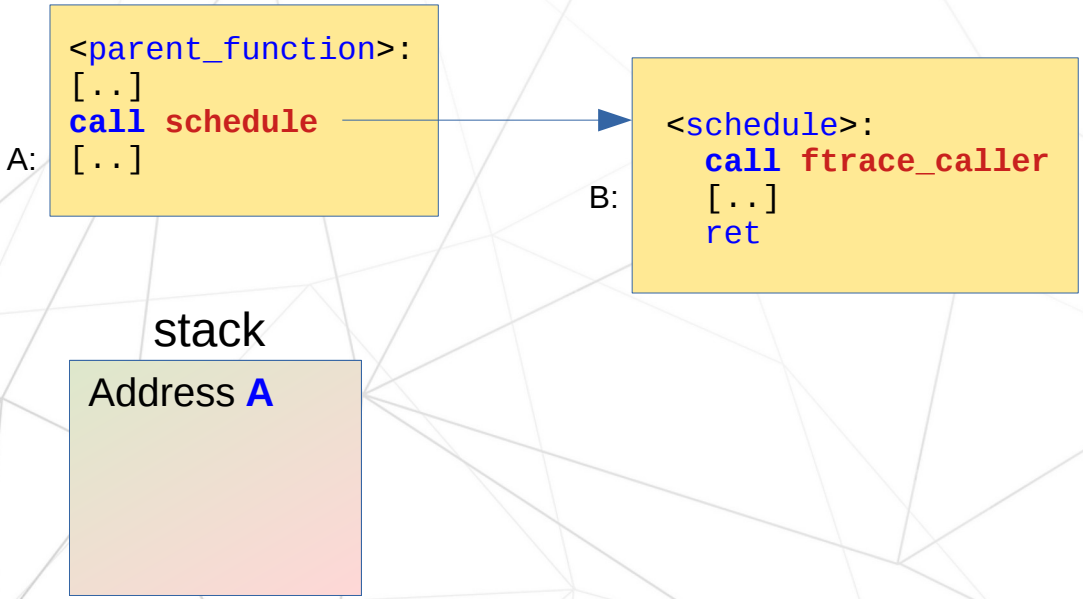
A:

```
<parent_function>:
[..]
call schedule
[..]
```

stack



How it works


<parent_function>:
[..]
call schedule
A:
[..]
<schedule>:
call ftrace_caller
B:
[..]
ret
stack
Address A

How it works

A:
```
<parent_function>:
[..]
call schedule
[..]
```

B:
```
<schedule>:
  call ftrace_caller
  [..]
  ret
```

```
<ftrace_caller>:
  save_regs
  call callback
  restore_regs
   (with new IP)
  ret
```

stack

Address **A**
Address **B**

How it works

A:
```
<parent_function>:
[..]
call schedule
[..]
```

B:
```
<schedule>:
  call ftrace_caller
  [..]
  ret
```

```
<ftrace_caller>:
  save_regs
  call callback
  restore_regs
   (with new IP)
  ret
```

stack

Address **C**
Address **B**

shadow stack

Address **A**

How it works

A:
```
<parent_function>:
[..]
call schedule
[..]
```

B:
```
<schedule>:
  call ftrace_caller
  [..]
  ret
```

```
<ftrace_caller>:
  save_regs
  call callback
  restore_regs
   (with new IP)
  ret
```

stack

Address **C**

shadow stack

Address **A**

How it works

A:
```
<parent_function>:
[..]
call schedule
[..]
```

B:
```
<schedule>:
  call ftrace_caller
  [..]
  ret
```

```
<ftrace_caller>:
  save_regs
  call callback
  restore_regs
   (with new IP)
  ret
```

stack

shadow stack

Address **A**

C:
```
<return_tramp>:
  save_regs
  call ret_callback
  restore_regs
   (with old IP)
  ret
```

How it works

A:
```
<parent_function>:
[..]
call schedule
[..]
```

B:
```
<schedule>:
  call ftrace_caller
  [..]
  ret
```

```
<ftrace_caller>:
  save_regs
  call callback
  restore_regs
   (with new IP)
  ret
```

stack

Address **A**

shadow stack

C:
```
<return_tramp>:
  save_regs
  call ret_callback
  restore_regs
   (with old IP)
  ret
```

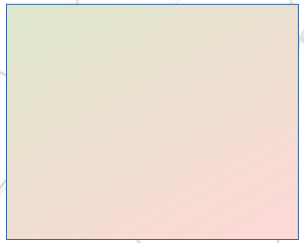
How it works

A:
```
<parent_function>:
[..]
call schedule
[..]
```

B:
```
<schedule>:
  call ftrace_caller
  [..]
  ret
```

```
<ftrace_caller>:
  save_regs
  call callback
  restore_regs
   (with new IP)
  ret
```

stack



shadow stack

C:
```
<return_tramp>:
  save_regs
  call ret_callback
  restore_regs
   (with old IP)
  ret
```

How it works

- BPF direct trampolines

How it works

- BPF direct trampolines
  - Call the traced function from the trampoline

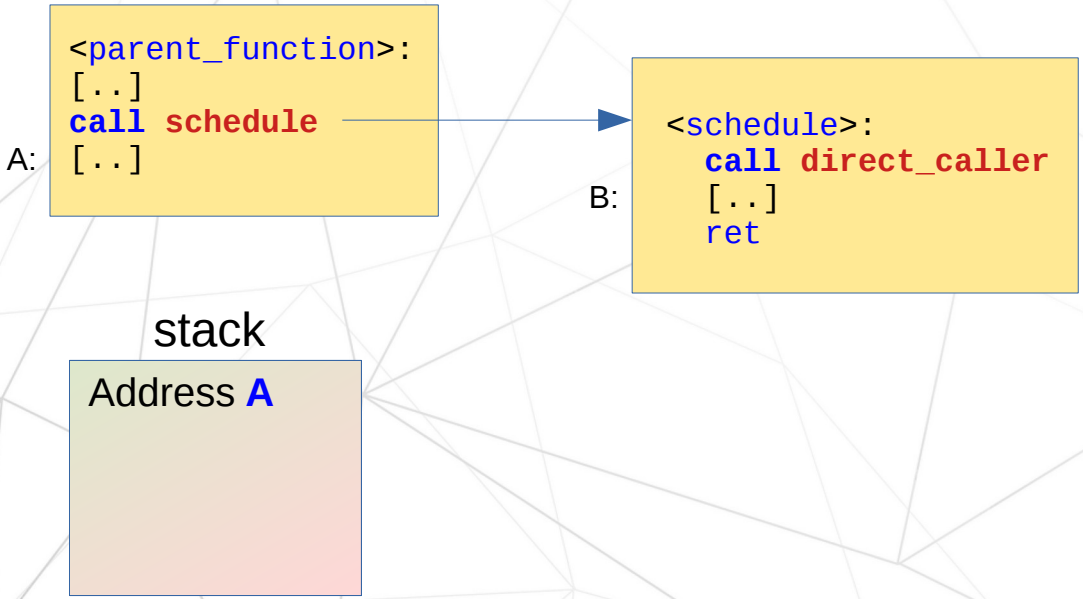
How it works

- BPF direct trampolines
  - Call the traced function from the trampoline
  - Calls return trace function

How it works

- BPF direct trampolines
  - Call the traced function from the trampoline
  - Calls return trace function
  - returns to parent function

How it works


<parent_function>:
[..]
call schedule
A:
[..]
<schedule>:
call direct_caller
B:
[..]
ret
stack
Address A

How it works

```
<parent_function>:
[..]
call schedule
A: [..]
```

```
<schedule>:
  call direct_caller
B:  [..]
  ret
```

```
<direct_caller>:
  save_args
  call callback
C:  restore_args
  call schedule+5
D:  call ret_callback
E:  pop
  ret
```

stack

Address **A**
Address **B**

How it works

```
<parent_function>:
[..]
call schedule
[..]
```
A:

```
<schedule>:
  call direct_caller
  [..]
  ret
```
B:

```
<direct_caller>:
  save_args
  call callback
  restore_args
  call schedule+5
  call ret_callback
  pop
  ret
```
C:
D:
E:

stack

Address **A**
Address **B**
Address **C**

How it works

```
<parent_function>:
[..]
call schedule
A: [..]
```

```
<schedule>:
  call direct_caller
B:  [..]
  ret
```

```
<direct_caller>:
  save_args
  call callback
C:  restore_args
  call schedule+5
D:  call ret_callback
E:  pop
  ret
```

stack

Address **A**
Address **B**
Address **D**

How it works

```
<parent_function>:
[..]
call schedule
[..]
```
A:

```
<schedule>:
  call direct_caller
  [..]
  ret
```
B:

```
<direct_caller>:
  save_args
  call callback
  restore_args
  call schedule+5
  call ret_callback
  pop
  ret
```
C:
D:
E:

stack

Address **A**
Address **B**

How it works

```
A:  <parent_function>:
    [..]
    call schedule
    [..]
```

```
B:  <schedule>:
      call direct_caller
      [..]
      ret
```

```
C:  <direct_caller>:
      save_args
      call callback
      restore_args
      call schedule+5
D:    call ret_callback
E:    pop
      ret
```

stack

Address **A**
Address **B**
Address **E**

How it works

```
A:  <parent_function>:
    [..]
    call schedule
    [..]
```

```
B:  <schedule>:
      call direct_caller
      [..]
      ret
```

```
C:  <direct_caller>:
      save_args
      call callback
      restore_args
      call schedule+5
D:    call ret_callback
E:    pop
      ret
```

stack

Address **A**

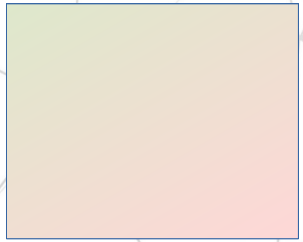
How it works

```
<parent_function>:
[..]
call schedule
A: [..]
```

```
<schedule>:
  call direct_caller
B:  [..]
  ret
```

```
<direct_caller>:
  save_args
  call callback
C:  restore_args
  call schedule+5
D:  call ret_callback
E:  pop
  ret
```

stack



What are the issues?

- Each has different requirements

What are the issues?

- Each has different requirements
  - function_graph only traces the end

What are the issues?

- Each has different requirements
  - function_graph only traces the end
  - kretprobes act like a breakpoint (full set of regs)

What are the issues?

- Each has different requirements
  - function_graph only traces the end
  - kretprobes act like a breakpoint (full set of regs)
  - BPF has function arguments at return

Proposal for kretprobe and function graph tracer

A:
```
<parent_function>:
[..]
call schedule
[..]
```

B:
```
<schedule>:
  call ftrace_caller
  [..]
  ret
```

```
<ftrace_caller>:
  save_regs
  call callback
  restore_regs
   (with new IP)
  ret
```

stack

Address **C**
Address **B**

shadow stack

Address **A**

Proposal for kretprobe and function graph tracer

A:
```
<parent_function>:
[..]
call schedule
[..]
```

B:
```
<schedule>:
  call ftrace_caller
  [..]
  ret
```

```
<ftrace_caller>:
  save_regs
  call callback
  restore_regs
   (with new IP)
  ret
```

stack

Address **C**
Address **B**

shadow stack

Address **A**
[extra data]

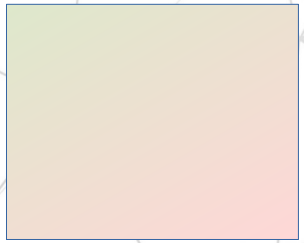
Proposal for kretprobe and function graph tracer

A:
```
<parent_function>:
[..]
call schedule
[..]
```

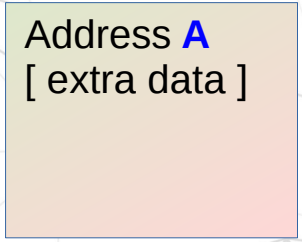
B:
```
<schedule>:
  call ftrace_caller
  [..]
  ret
```

```
<ftrace_caller>:
  save_regs
  call callback
  restore_regs
   (with new IP)
  ret
```

stack



shadow stack



C:
```
<return_tramp>:
  save_regs
  call ret_callback
  restore_regs
   (with old IP)
  ret
```

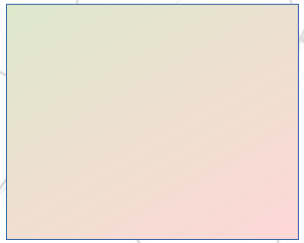
Proposal for kretprobe and function graph tracer

A:
```
<parent_function>:
[..]
call schedule
[..]
```

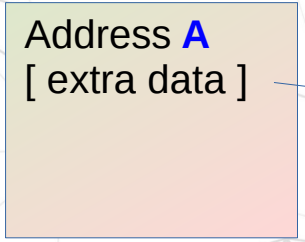
B:
```
<schedule>:
  call ftrace_caller
  [..]
  ret
```

```
<ftrace_caller>:
  save_regs
  call callback
  restore_regs
   (with new IP)
  ret
```

stack



shadow stack



C:
```
<return_tramp>:
  save_regs
  call ret_callback
  restore_regs
   (with old IP)
  ret
```

BPF direct trampolines

- Optimized to be very efficient
  - Customizes the arguments to save

BPF direct trampolines

- Optimized to be very efficient
  - Customizes the arguments to save
  - Requires a unique trampoline per function

BPF direct trampolines

- Optimized to be very efficient
  - Customizes the arguments to save
  - Requires a unique trampoline per function
- Stack args must be copied

BPF direct trampolines

- Optimized to be very efficient
  - Customizes the arguments to save
  - Requires a unique trampoline per function
- Stack args must be copied
- Can not have a generic trampoline for all functions

BPF direct of function with more than 6 args

A:
```
<parent_function>:
[..]
push args
call __skb_flow_dissect
[..]
```

B:
```
<_skb_flow_dissect>:
  call direct_caller
  [..]
  ret
```

stack

ARG 9
ARG 8
ARG 7
Address **A**

BPF direct of function with more than 6 args

```
A:  <parent_function>:
    [..]
    push args
    call __skb_flow_dissect
    [..]
```

stack

ARG 9
ARG 8
ARG 7
Address **A**
Address **B**

```
B:  <_skb_flow_dissect>:
      call direct_caller
      [..]
      ret
```

```
    <direct_caller>:
      save_args
      call callback
C:    restore_args
      call schedule+5
D:    call ret_callback
E:    pop_args
      pop
      ret
```

BPF direct of function with more than 6 args

```
<parent_function>:
[..]
push args
A: call __skb_flow_dissect
[..]
```

```
<_skb_flow_dissect>:
  call direct_caller
B:  [..]
  ret
```

```
<direct_caller>:
  save_args
  call callback
C:  restore_args
  call schedule+5
D:  call ret_callback
E:  pop_args
  pop
  ret
```

stack

ARG 9
ARG 8
ARG 7
Address **A**
Address **B**
ARG 9
ARG 8
ARG 7
Address **C**

BPF direct of function with more than 6 args

```
<parent_function>:
[..]
push args
A: call __skb_flow_dissect
[..]
```

```
<_skb_flow_dissect>:
  call direct_caller
B:  [..]
  ret
```

```
<direct_caller>:
  save_args
  call callback
C:  restore_args
  call schedule+5
D:  call ret_callback
E:  pop_args
  pop
  ret
```

stack

ARG 9
ARG 8
ARG 7
Address **A**
Address **B**
ARG 9
ARG 8
ARG 7
Address **D**

BPF direct of function with more than 6 args

```
A:  <parent_function>:
    [..]
    push args
    call __skb_flow_dissect
    [..]
```

```
B:  <_skb_flow_dissect>:
      call direct_caller
      [..]
      ret
```

```
C:  <direct_caller>:
      save_args
      call callback
      restore_args
      call schedule+5
D:    call ret_callback
E:    pop_args
      pop
      ret
```

stack

ARG 9
ARG 8
ARG 7
Address **A**
Address **B**