

uprobe optimization and override

jiri olsa / isovalent @ cisco

UPROBE OPTIMIZATION

**on top of nop5 and beyond
on x86_64**

UPROBE QUICK COURSE

```
<foo>:
```

```
401120:  push %rbp
```

```
401121:  mov  %rsp,%rbp
```

```
401124:  sub  $0x10,%rsp
```

```
...
```

UPROBE QUICK COURSE

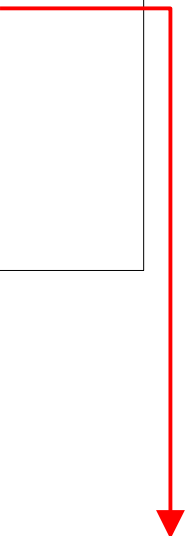
<foo>:

401120: ~~push %rbp~~ **int3**

401121: mov %rsp,%rbp

401124: sub \$0x10,%rsp

...



run bpf program

orig instruction execution

emulation

single step

UPROBE QUICK COURSE

<foo>:

401120: ~~push %rbp~~ **int3**

401121: mov %rsp,%rbp

401124: sub \$0x10,%rsp

...

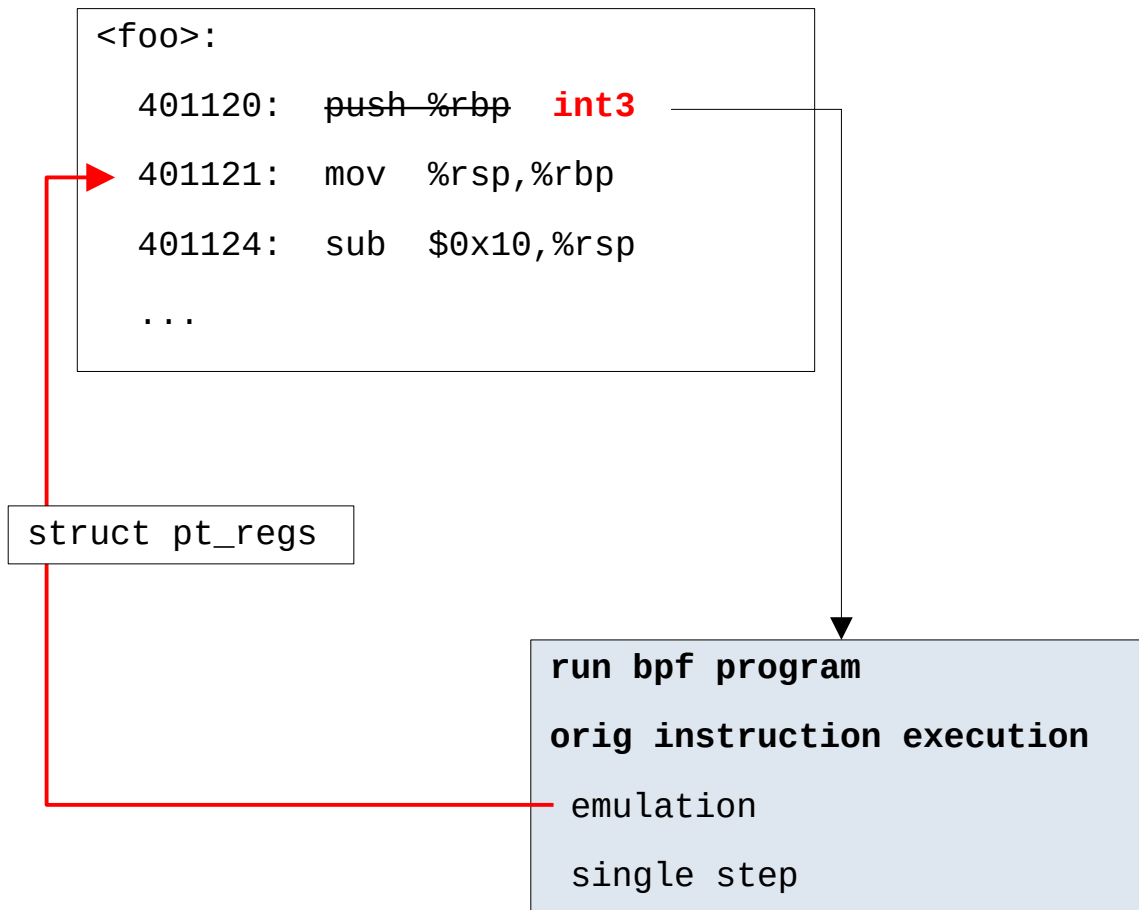
struct pt_regs

run bpf program

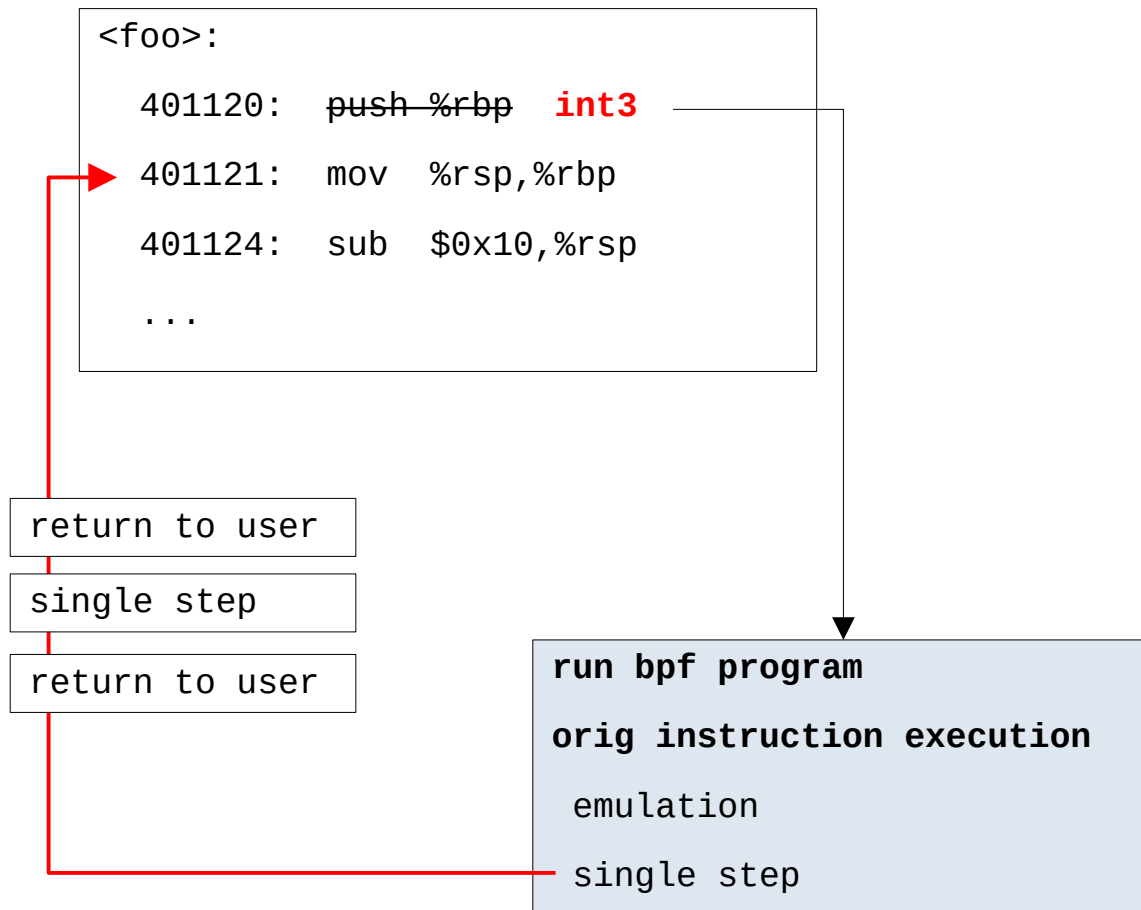
orig instruction execution

emulation

single step



UPROBE QUICK COURSE



UPROBE QUICK COURSE

<foo>:

401120: ~~push %rbp~~ **int3**

401121: mov %rsp,%rbp

401124: sub \$0x10,%rsp

...

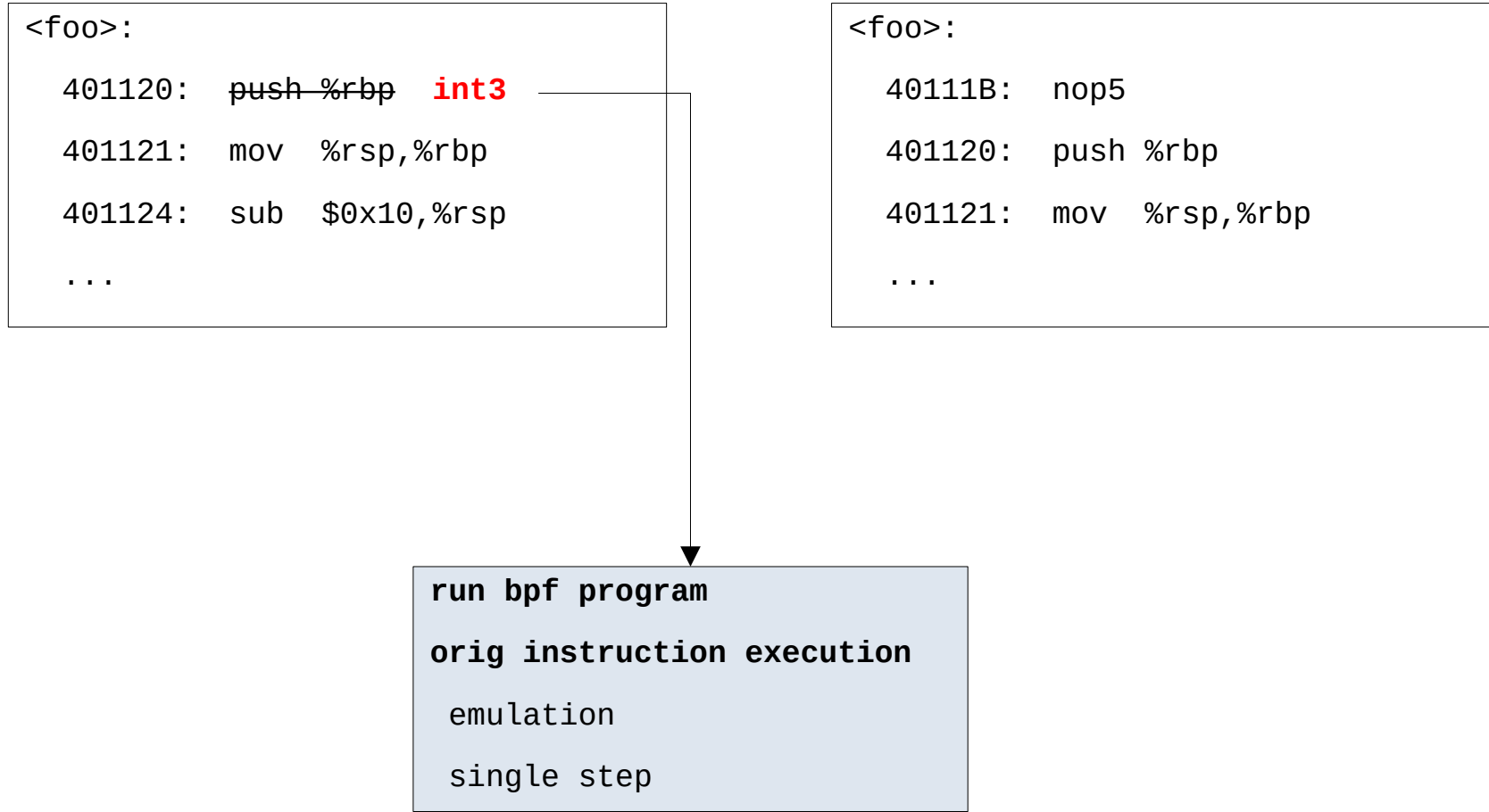
<foo>:

40111B: nop5

401120: push %rbp

401121: mov %rsp,%rbp

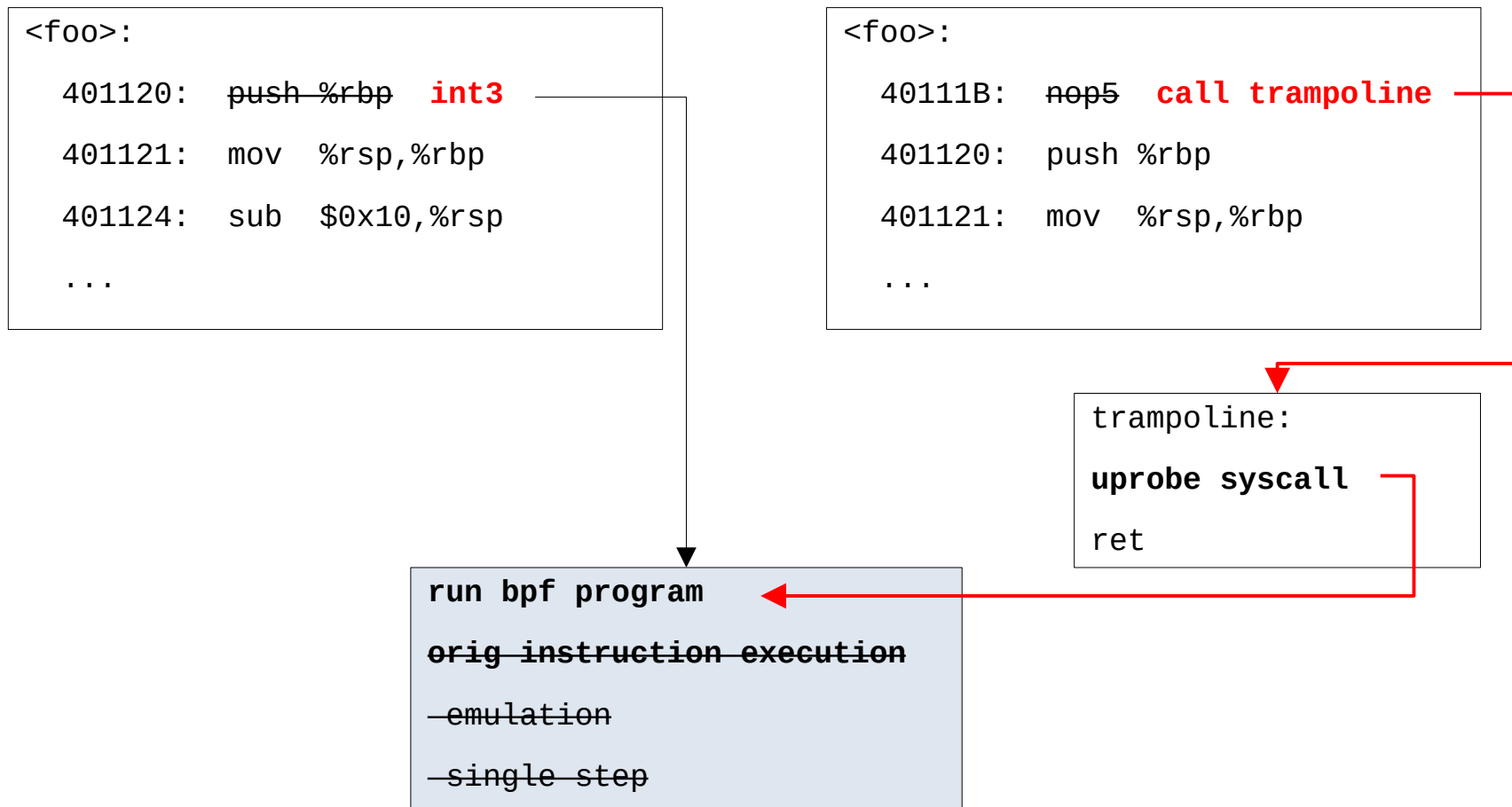
...



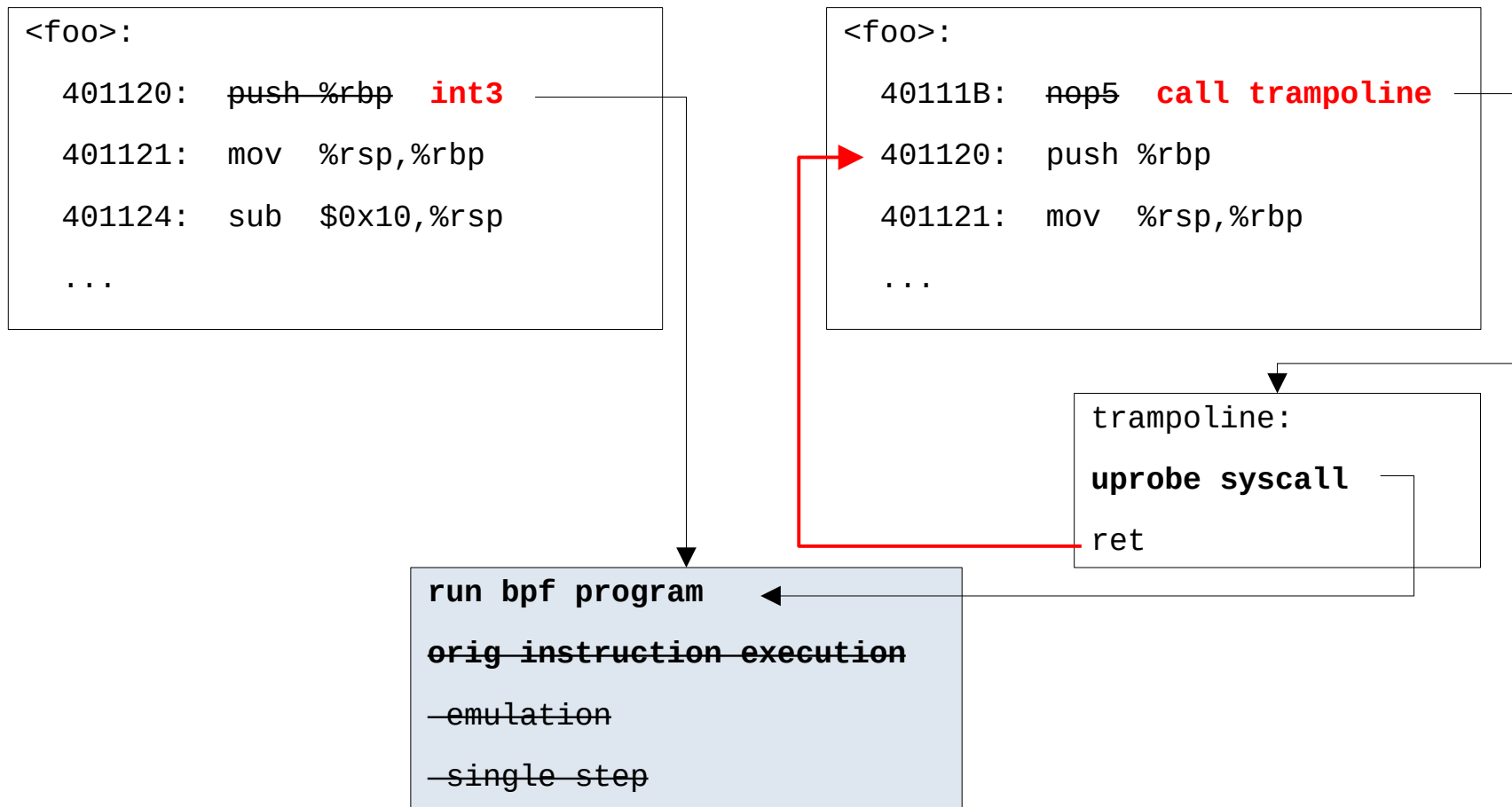
A diagram illustrating the UPROBE quick course workflow. It starts with two boxes of assembly code. The left box shows a function frame for <foo> with instructions at addresses 401120, 401121, and 401124. The instruction at 401120 is `push %rbp` with `int3` in red, and it is crossed out with a line. The right box shows a similar function frame with instructions at 40111B, 401120, and 401121. An arrow points from the crossed-out instruction in the left box to a light blue box at the bottom containing four options: `run bpf program`, `orig instruction execution`, `emulation`, and `single step`.

run bpf program
orig instruction execution
emulation
single step

UPROBE QUICK COURSE



UPROBE QUICK COURSE



USDT OPTIMIZATION

make usdt macros emit **nop5**

backward compatibility with old kernels?

```
...  
STAP_PROBE(test, usdt0)  
USDT(test, usdt0)  
...
```

→ **nop1**

```
...  
movq    $0x15, 0x28(%rsp)  
movb     $0x0, 0x4(%rsp)  
mov      %r12, 0x10(%rsp)  
nop1  
lea      0x7f2f106(%rip), %r15  
lea      0x197ae3d(%rip), %rsi  
lea      0x8(%rsp), %rdx  
...
```

USDT OPTIMIZATION

make usdt macros emit nop5

backward compatibility with old kernels?

```
...  
STAP_PROBE(test, usdt0)  
USDT(test, usdt0)  
...
```

```
...  
movq    $0x15, 0x28(%rsp)  
movb    $0x0, 0x4(%rsp)  
mov     %r12, 0x10(%rsp)  
nop5  
lea     0x7f2f106(%rip), %r15  
lea     0x197ae3d(%rip), %rsi  
lea     0x8(%rsp), %rdx  
...
```

USDT OPTIMIZATION

make usdt macros emit ~~nop5~~ ***nop1,nop5***

backward compatibility with old kernels!

```
...  
STAP_PROBE(test, usdt0)  
USDT(test, usdt0)  
...
```

```
...  
movq    $0x15,0x28(%rsp)  
movb    $0x0,0x4(%rsp)  
mov     %r12,0x10(%rsp)  
→ nop1, nop5  
lea     0x7f2f106(%rip),%r15  
lea     0x197ae3d(%rip),%rsi  
lea     0x8(%rsp),%rdx  
...
```

UPROBE OPTIMIZATION

cover more sites/prologues

on top of emulated instructions

UPROBE OPTIMIZATION

```
<foo>:  
77ec3f: 55                push    %rbp  
77ec40: 48 89 e5          mov     %rsp,%rbp  
77ec43: 48 81 ec 00 03 00 00 sub     $0x300,%rsp  
...
```

UPROBE OPTIMIZATION

```
<foo>:  
77ec3f: 55                push    %rbp  
77ec40: 48 89 e5         mov     %rsp,%rbp  
77ec43: 48 81 ec 00 03 00 00 sub     $0x300,%rsp  
...
```

```
<foo>:  
77ec3f: e8 XX XX XX XX    call trampoline  
77ec44: 81 ec 00 03 00 00  
77ec4a:  
...
```

```
trampoline:  
uprobe syscall  
ret
```

A red arrow originates from the 'call trampoline' instruction in the second code block and points to the 'trampoline:' label in the third code block, indicating the control flow jump.

UPROBE OPTIMIZATION

```
<foo>:  
77ec3f: 55 push %rbp  
77ec40: 48 89 e5 mov %rsp,%rbp  
77ec43: 48 81 ec 00 03 00 00 sub $0x300,%rsp  
...
```

```
<foo>:  
77ec3f: e8 XX XX XX XX call trampoline  
77ec44: 81 ec 00 03 00 00  
77ec4a:  
...
```

trampoline:

```
uprobe syscall  
ret
```

run bpf program

emulate:

- push %rbp
- mov %rsp,%rbp
- sub \$0x300,%rsp

UPROBE OPTIMIZATION

```
<foo>:  
77ec3f: 55          push    %rbp  
77ec40: 48 89 e5     mov     %rsp,%rbp  
77ec43: 48 81 ec 00 03 00 00 sub     $0x300,%rsp  
...
```

```
<foo>:  
77ec3f: e8 XX XX XX XX      call trampoline  
77ec44: 81 ec 00 03 00 00  
77ec4a: ←  
...
```

trampoline:

```
uprobe syscall  
ret
```

run bpf program ←

emulate:

- push %rbp
- mov %rsp,%rbp
- sub \$0x300,%rsp

HUM.. call update

<foo>:

77ec3f: 55 -> cc -> e8 push %rbp

→ 77ec40: XX XX XX mov %rsp,%rbp

→ 77ec43: XX 81 ec 00 03 00 00 sub \$0x300,%rsp

...

HUM.. call update

<foo>:

77ec3f: 55 -> cc -> e8 push %rbp

→ 77ec40: XX XX XX mov %rsp,%rbp

→ 77ec43: XX 81 ec 00 03 00 00 sub \$0x300,%rsp

...

<foo>:

77ec3f: 55 -> cc -> e8 push

77ec40: cc XX XX mov

77ec43: cc 81 ec 00 03 00 00 sub

...

HUM.. call update

figure out early optimization

stop process

HUM.. need sysctl knob? ;-)

```
<foo>:
77ec3f: 55                push    %rbp
77ec40: 48 89 e5          mov     %rsp,%rbp
77ec43: 48 81 ec 00 03 00 00 sub     $0x300,%rsp
...
```

```
<foo>:
77ec3f: e8 XX XX XX XX      call trampoline
77ec44: 81 ec 00 03 00 00
77ec4a:
...
```

```

...
jmp/call 77ec43
...
```

UPROBE OVERRIDE

control user space execution

guard app against CVEs

UPROBE OVERRIDE

uprobe program can write to context (v6.18)

ip change does not execute original instruction

```
SEC("uprobe")
int BPF_UPROBE(test)
{
    ctx->ax = 0xdead;
    ctx->ip = new_ip;
    return 0;
}
```

UPROBE OVERRIDE

function override like in kernel

```
<foo>:
```

```
push    %rbp
```

```
mov     %rsp,%rbp
```

```
sub     $0x300,%rsp
```

```
mov     %edi,%eax
```

```
mov     %al,-0x2f4(%rbp)
```

```
lea     -0x160(%rbp),%rax
```

```
mov     $0x30,%edx
```

```
mov     $0x0,%esi
```

```
mov     %rax,%rdi
```

```
rip=(%rsp)
```

```
rsp=%rsp + 8
```

UPROBE OVERRIDE

deep in the function

```
<foo>:  
push    %rbp  
mov     %rsp,%rbp  
sub     $0x300,%rsp  
mov     %edi,%eax  
mov     %al,-0x2f4(%rbp)  
lea     -0x160(%rbp),%rax  
mov     $0x30,%edx  
mov     $0x0,%esi  
mov     %rax,%rdi
```

rip=?

rsp=?

restore other callee saved registers

thanks, questions?