

ISOVALENT

BPF Static Keys



Linux Plumbers 2023

Anton Protopopov



Static Keys in the Linux Kernel

- The Linux Kernel [Static Keys API](#) was added in 2009
- It is based on the [`asm goto`](#) feature provided by a compiler (which lets to jump to labels defined in C from inline assembly)
- + An ability to live-patch Linux Kernel code
- “Static keys allows the inclusion of seldom used features in performance-sensitive fast-path kernel code”
- This talk shows how to add this functionality to BPF

Static Keys in the Linux Kernel: example

```
DEFINE_STATIC_KEY_FALSE(key);

void sk_example(volatile u64 *x)
{
    if (static_branch_unlikely(&key))
        *x = 1;
}
```

Static Keys in the Linux Kernel

This key is off by default. The “static” part comes from the fact that we can’t create new keys dynamically—only to compile them

```
DEFINE_STATIC_KEY_FALSE(key);
```

```
void sk_example(volatile u64 *x)
{
    if (static_branch_unlikely(&key))
        *x = 1;
}
```

Static Keys in the Linux Kernel

```
DEFINE_STATIC_KEY_FALSE(key);
```

This key is off by default. The “static” part comes from the fact that we can’t create new keys dynamically—only to compile them

```
void sk_example(volatile u64 *x)
{
    if (static_branch_unlikely(&key))
        *x = 1;
}
```

This is unlikely that it will be turned on. When disabled the check costs nothing

Static Keys in the Linux Kernel

0000000000000070 <sk_example>:

70: e8 00 00 00 00

75: 55

76: 48 89 e5

79: 66 90

7b: 5d

7c: e9 00 00 00 00

81: 48 c7 07 01 00 00 00

88: 5d

89: e9 00 00 00 00

call 75 <sk_example+0x5>

push %rbp

mov %rsp,%rbp

xchg %ax,%ax

pop %rbp

jmp 81 <sk_example+0x11>

movq \$0x1, (%rdi)

pop %rbp

jmp 8e <sk_example+0x1e>

Static Keys in the Linux Kernel

```
00000000000000070 <sk_
 70:  e8 00 00 00 0  <sk_example+0x5>
 75:  55
 76:  48 89 e5      mov    %rsp,%rbp
 79:  66 90      xchg   %ax,%ax
 7b:  5d
 7c:  e9 00 00 00 00  jmp    81 <sk_example+0x11>
 81:  48 c7 07 01 00 00 00  movq   $0x1,(%rdi)
 88:  5d      pop    %rbp
 89:  e9 00 00 00 00  jmp    8e <sk_example+0x1e>
```

The static key is off => the jump is replaced by a NOP.

Static Keys in the Linux Kernel

000000000000000070	<sk_	
70: e8 00 00 00 0		<sk_example+0x5>
75: 55		bp
76: 48 89 e5	mov	%rsp,%rbp
79: eb 06	jmp	81 <sk_example+0x11>
7b: 5d	pop	%rbp
7c: e9 00 00 00 00	jmp	81 <sk_example+0x11>
81: 48 c7 07 01 00 00 00	movq	\$0x1, (%rdi)
88: 5d	pop	%rbp
89: e9 00 00 00 00	jmp	8e <sk_example+0x1e>

If we turn it on, then the NOP is replaced by a jump

Static Keys in the Linux Kernel

000000000000000070 <sk_

70: e8 00 00 00 0

75: 55

76: 48 89 e5

79: eb 06

7b: 5d

7c: e9 00 00 00 00

81: 48 c7 07 01 00 00 00

88: 5d

89: e9 00 00 00 00

If we turn it on, then the NOP
is replaced by a jump

<sk_example+0x5>

bp

mov %rsp,%rbp

jmp 81 <sk_example+0x11>

pop %rbp

jmp 81 <sk_example+0x11>

movq \$0x1, (%rdi)

pop %rbp

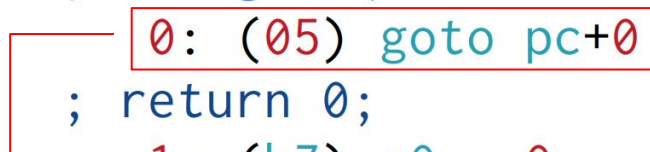
jmp 8e <sk_example+0x1e>

Goal: we want to do the same in BPF

```
__section("kprobe/__x64_sys_getpgid")
int worker(void *ctx)
{
    if (bpf_static_branch_unlikely(&debug_key))
        bpf_printk("__x64_sys_getpgid\n");
    return 0;
}
```

Static Keys in BPF: branch is unlikely, key is off

```
int worker(void * ctx):  
; asm goto("1:"  
0: (05) goto pc+0  
; return 0;  
1: (b7) r0 = 0  
2: (95) exit  
; bpf_printk("__x64_sys_getpgid");  
3: (18) r1 = map[id:31][0]+0  
5: (b7) r2 = 18  
6: (85) call bpf_trace_printk#-79456  
7: (05) goto pc-7
```

A red line originates from the semicolon of the `; return 0;` statement and points to the `1:` label, indicating a branch. The `0: (05) goto pc+0` instruction is enclosed in a red rectangular box.

Static Keys in BPF: branch is unlikely, key is on

```
int worker(void * ctx):
```

```
; asm goto("1:"
```

```
0: (05) goto pc+2
```

```
; return 0;
```

```
1: (b7) r0 = 0
```

```
2: (95) exit
```

```
; bpf_printk("__x64_sys_getpgid");
```

```
3: (18) r1 = map[id:41][0]+0
```

```
5: (b7) r2 = 18
```

```
6: (85) call bpf_trace_printk#-79456
```

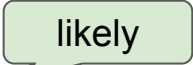
```
7: (05) goto pc-7
```

Second option: we want to prioritize the branch

```
__section("kprobe/__x64_sys_getpgid")
int worker(void *ctx)
{
    if (bpf_static_branch_likely(&debug_key))
        bpf_printk("__x64_sys_getpgid\n");
    return 0;
}
```


Second option: we want to prioritize the branch

```
__section("kprobe/__x64_sys_getpgid")
int worker(void *ctx)
{
    if (bpf_static_branch_likely(&debug_key))
        bpf_printk("__x64_sys_getpgid\n");
    return 0;
}
```



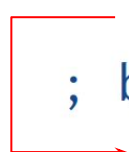
Static Keys in BPF: branch is likely, key is off

```
int worker(void * ctx):  
; asm goto("1:"  
0: (05) goto pc+4  
; bpf_printk("__x64_sys_getpgid\n");  
1: (18) r1 = map[id:39][0]+0  
3: (b7) r2 = 19  
4: (85) call bpf_trace_printk#-81312  
; return 0;  
5: (b7) r0 = 0  
6: (95) exit
```

A red line originates from the right side of the instruction '0: (05) goto pc+4', extends horizontally to the left, then turns vertically downwards, and finally turns horizontally to the right, ending with an arrow pointing to the instruction '5: (b7) r0 = 0'. This indicates a jump from instruction 0 to instruction 5.

Static Keys in BPF: branch is likely, key is on

```
int worker(void * ctx):  
; asm goto("1:"  
0: (05) goto pc+0  
; bpf_printk("__x64_sys_getpgid\n");  
1: (18) r1 = map[id:39][0]+0  
3: (b7) r2 = 19  
4: (85) call bpf_trace_printk#-81312  
; return 0;  
5: (b7) r0 = 0  
6: (95) exit
```

A red line with an arrowhead at the end points from the right side of the instruction '0: (05) goto pc+0' to the left side of the instruction '1: (18) r1 = map[id:39][0]+0', indicating a branch.

Static Keys in BPF: building blocks

In order to have BPF Static Keys we need two items:

- We want to compile `bpf_static_branch_{likely/unlikely}` into code blocks shown above
- We want to be able to toggle branches in a live BPF program:
 - *Normal* branches: `jmp/nop` when key is `on/off`
 - *Inverse* branches: `nop/jmp` when key is `on/off`

Static Keys in BPF: building blocks

In order to have BPF Static Keys we need two items:

- We want to compile `bpf_static_branch_{likely/unlikely}` into code blocks shown above
- We want to be able to toggle branches in a live BPF program:
 - *Normal* branches: `jmp/nop` when key is `on/off`
 - *Inverse* branches: `nop/jmp` when key is `on/off`
- Solution: use ``asm goto`` + extend BPF API

ASM goto: branch is unlikely (x86_64)

```
static __always_inline bool __bpf_static_branch_nop(void *static_key)
```

```
{
```

```
    asm goto("1:\n\t"  
             "goto +0\n\t"  
             ".pushsection .jump_table, \"aw\" \n\t"  
             ".balign 8\n\t"  
             ".long 1b - . \n\t"  
             ".long %l[l_yes] - . \n\t"  
             ".quad %c0 - .\n\t"  
             ".popsection \n\t"  
             ":: \"i\" (static_key)  
             :: l_yes);
```

```
    return false;
```

```
l_yes:
```

```
    return true;
```

```
}
```

```
#define bpf_static_branch_unlikely(static_key) \  
    unlikely(__bpf_static_branch_nop(static_key))
```

```
int worker(void * ctx):
```

```
; asm goto("1:"
```

```
    0: (05) goto pc+0
```

```
; return 0;
```

```
    1: (b7) r0 = 0
```

```
    2: (95) exit
```

```
; bpf_printk("__x64_sys_getpgid");
```

```
    3: (18) r1 = map[id:31][0]+0
```

```
    5: (b7) r2 = 18
```

```
    6: (85) call bpf_trace_printk#-79456
```

```
    7: (05) goto pc-7
```

ASM goto: branch is unlikely (x86_64)

```
static __always_inline bool __bpf_static_branch_nop(void *static_key)
```

```
{
```

```
    asm goto("1:\n\t"  
            "goto +0\n\t"  
            ".pushsection .jump_table, \"aw\" \n\t"  
            ".balign 8\n\t"  
            ".long 1b - . \n\t"  
            ".long %l[l_yes] - . \n\t"  
            ".quad %c0 - . \n\t"  
            ".popsection \n\t"  
            :: "i" (static_key)  
            :: l_yes);
```

```
    return false;
```

```
l_yes:
```

```
    return true;
```

```
}
```

```
#define bpf_static_branch_unlikely(static_key) \  
    unlikely(__bpf_static_branch_nop(static_key))
```

```
int worker(void * ctx):
```

```
; asm goto("1:"
```

```
0: (05) goto pc+0
```

```
; return 0;
```

```
1: (b7) r0 = 0
```

```
2: (95) exit
```

```
; bpf_printk("__x64_sys_getpgid");
```

```
3: (18) r1 = map[id:31][0]+0
```

```
5: (b7) r2 = 18
```

```
6: (85) call bpf_trace_printk#-79456
```

```
7: (05) goto pc-7
```

ASM goto: branch is unlikely (x86_64)

```
static __always_inline bool __bpf_static_branch_nop(void *static_key)
```

```
{
```

```
    asm goto("1:\n\t"  
            "goto +0\n\t"  
            ".pushsection .jump_table, \"aw\" \n\t"  
            ".balign 8\n\t"  
            ".long 1b - . \n\t"  
            ".long %l[l_yes] - . \n\t"  
            ".quad %c0 - . \n\t"  
            ".popsection \n\t"  
            ":: \"i\" (static_key)  
            :: l_yes);
```

```
    return false;
```

```
l_yes:
```

```
    return true;
```

```
}
```

```
#define bpf_static_branch_unlikely(static_key) \  
    unlikely(__bpf_static_branch_nop(static_key))
```

```
int worker(void * ctx):
```

```
; asm goto("1:"
```

```
0: (05) goto pc+0
```

```
; return 0;
```

```
1: (b7) r0 = 0
```

```
2: (95) exit
```

```
; bpf_printk("__x64_sys_getpgid");
```

```
3: (18) r1 = map[id:31][0]+0
```

```
5: (b7) r2 = 18
```

```
6: (85) call bpf_trace_printk#-79456
```

```
7: (05) goto pc-7
```

ASM goto: branch is unlikely (x86_64)

```
static __always_inline bool __bpf_static_branch_nop(void *static_key)
```

```
{
```

```
    asm goto("1:\n\t"  
            "goto +0\n\t"  
            ".pushsection .jump_table, \"aw\" \n\t"  
            ".balign 8\n\t"
```

```
            ".long 1b - . \n\t"
```

```
            ".long %l[l_yes] - . \n\t"
```

```
            ".quad %c0 - . \n\t"
```

```
            ".popsection \n\t"
```

```
            :: "i" (static_key)
```

```
            :: l_yes);
```

```
    return false;
```

```
l_yes:
```

```
    return true;
```

```
}
```

```
#define bpf_static_branch_unlikely(static_key) \  
    unlikely(__bpf_static_branch_nop(static_key))
```

```
int worker(void * ctx):
```

```
; asm goto("1:"
```

```
0: (05) goto pc+0
```

```
; return 0;
```

```
1: (b7) r0 = 0
```

```
2: (95) exit
```

```
; bpf_printk("__x64_sys_getpgid");
```

```
3: (18) r1 = map[id:31][0]+0
```

```
5: (b7) r2 = 18
```

```
6: (85) call bpf_trace_printk#-79456
```

```
7: (05) goto pc-7
```

.rel.jump_table

BPF Static Key: just a map

```
struct {  
    __uint(type, BPF_MAP_TYPE_ARRAY);  
    __type(key, __u32);  
    __type(value, __u32);  
    __uint(map_flags, BPF_F_STATIC_KEY);  
    __uint(max_entries, 1);  
} debug_key __section(".maps");
```

Static Keys in BPF: API

- In order to use static keys a program should be loaded with an array of “static branches”, where each static branch is of the following form

```
struct bpf_static_branch_info {  
    __u32 map_fd;  
    __u32 insn_offset;  
    __u32 jump_target;  
    __u32 flags;  
};
```


Static Keys in BPF: API

- On BPF_PROG_LOAD we pass an array of bpf_static_branch_info structs via attrs:

```
union bpf_attr {  
    ...  
    struct { /* BPF_PROG_LOAD */  
        ...  
        __aligned_u64    static_branches_info;  
        __u32            static_branches_info_size;  
    };  
    ...  
};
```

Static Keys in BPF: API

- On BPF_PROG_LOAD we pass an array of bpf_static_branch_info structs via attrs:

```
union bpf_attr {  
    ...  
    struct { /* BPF_PROG_LOAD */  
        ...  
        __aligned_u64    static_branches_info;  
        __u32            static_branches_info_size;  
    };  
    ...  
};
```

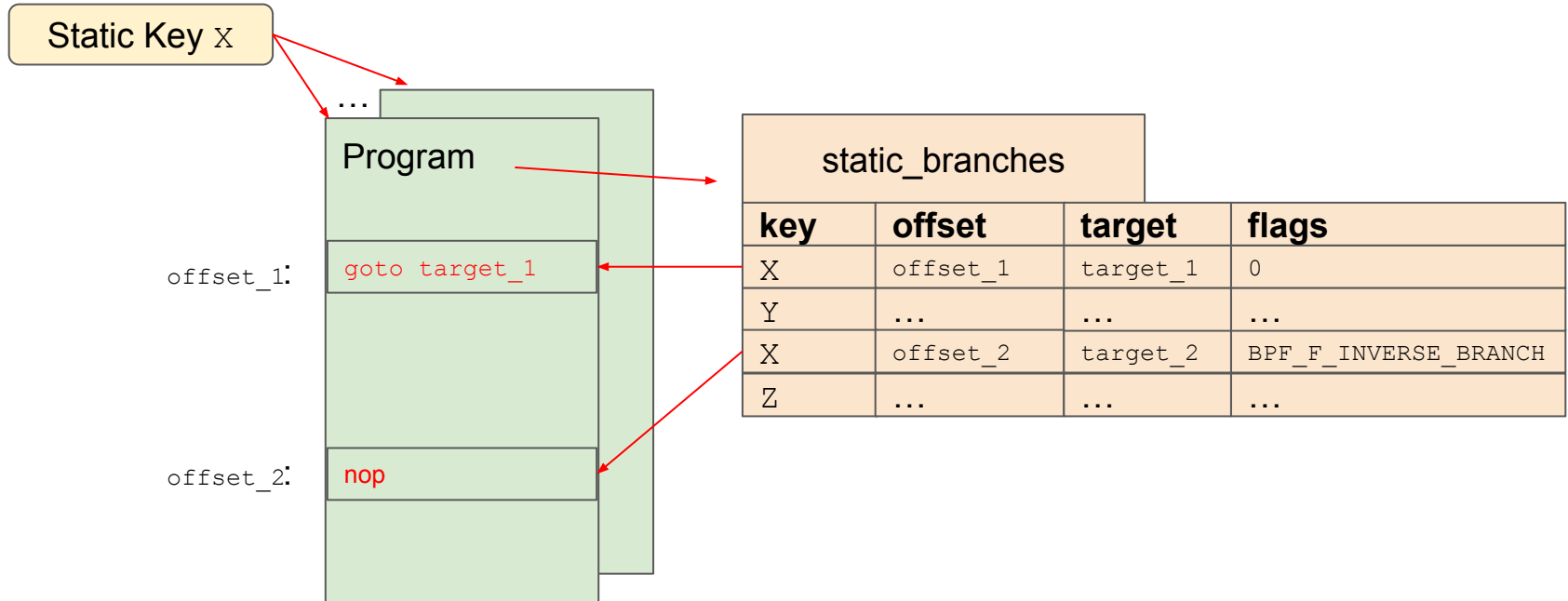
Libbpf will do all the work when proper ".jump_table" and ".rel.jump_table" tables are present

Static Keys in BPF: API

- To toggle branches on/off we just update the map value via the `bpf(BPF_MAP_UPDATE_ELEM)` syscall

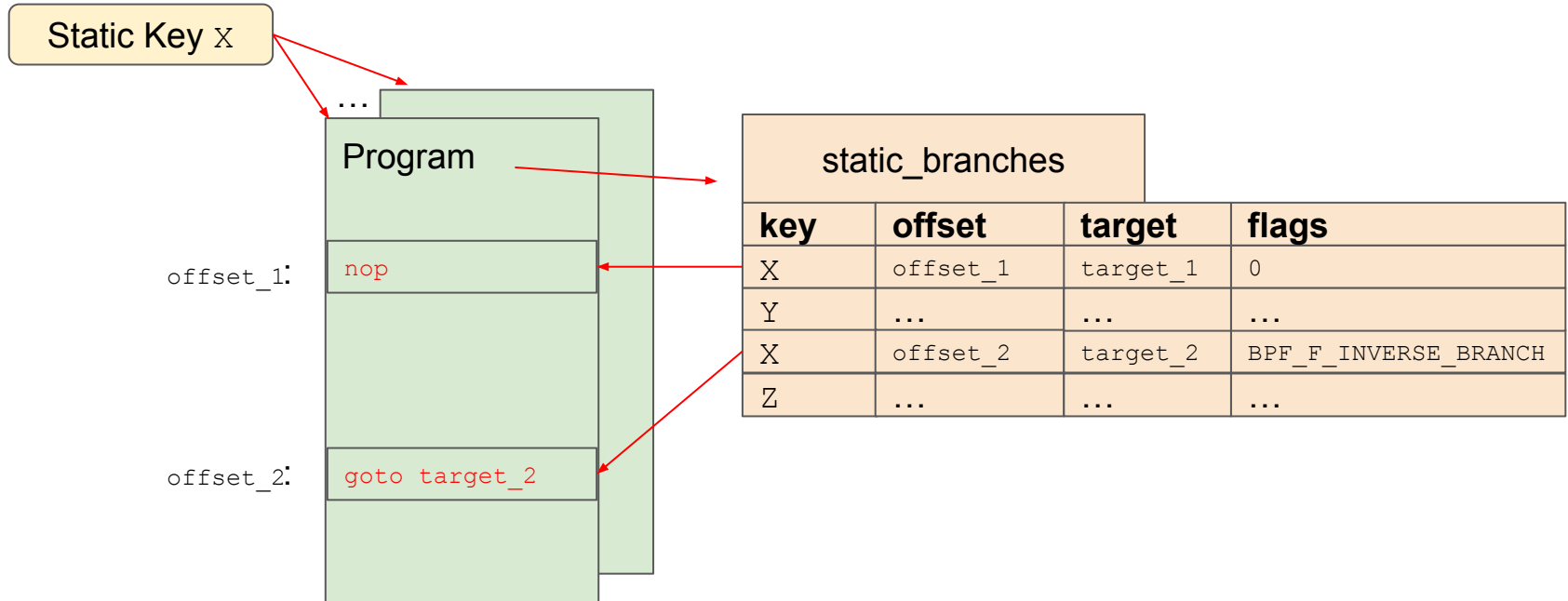
Static Keys in BPF: API

`syscall(BPF_MAP_UPDATE, X, 1)`



Static Keys in BPF: API

`syscall(BPF_MAP_UPDATE, X, 0)`



Static Keys in BPF: API

- We also need to prevent BPF programs from accessing the static keys directly. The solution I chose was to reject programs trying to use static keys as normal maps
- There is `BPF_F_READONLY_PROG`, but it provides different semantics: `bpf_map_is_rdonly = READONLY && frozen`. Map values treated like constants in verifier. This is not what I needed

Static Keys in BPF: life of static branch

BPF_PROG_LOAD

bpf_check()

attrs->bpf_static_branch_info[]

map_fd	flags	offsets
...
map_fd	flags	offsets

(1)

prog->aux->static_branches[]

map_ptr*	flags	insns
...
map_ptr*	flags	insns

bpf_patch_insn_data()

...	18000000beef1011
call 0x76	00000000ffffffff
goto +0	r0 = *(u64 *)r0
r0 = 0	
exit	
...	

(4) x N

Update offsets, if needed

env->used_maps[]

map_ptr*
...
map_ptr*

(3)

(5)

prog->aux->used_maps[]

map_ptr*
...
map_ptr*

(2)

Store used maps in ->used_maps, init verifier env

Static Keys in BPF: Verification

- It turned out that verification is straightforward: just follow two edges of a branch, like in a conditional jump
- (Another option was to follow all static branches referencing the same static key as having same state. However, this is actually not guaranteed, as poking code is a per-instruction operation, so two branches referencing same static key may actually have different on/off states)

ISOVALENT

Questions?

