ISOVALENT
(now part of Cisco)

Instruction Pointers, Static Keys, Jump Tables

LSF/MM/BPF 2024

Anton Protopopov

Recap: BPF Static Keys API

- The previous version of BPF Static Keys wasn’t too successful (not generic enough)
- Let’s take a look on how static keys can be implemented in more generic way, such that the core parts can be reused for other features (jump labels, etc.)

Recap: BPF Static Keys API, Example

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

BPF Static Keys: 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
```

BPF Static Keys: branch is unlikely, key is off

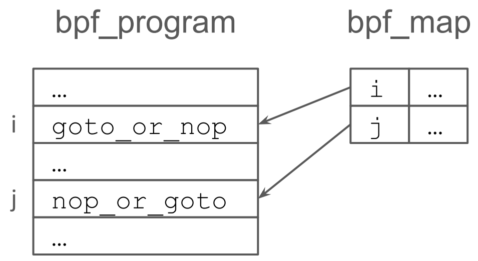
```
int worker(void * ctx):
; asm goto("1:"
   0: (05) goto pc+2
; return 0;
   1: (b7) r0 = 0
   2: (95) exit
; bpf_printk("__x64_sys_getpgid");
   3: (18) r1 = map[id:41][0]+0
   5: (b7) r2 = 18
   6: (85) call bpf_trace_printk#-79456
   7: (05) goto pc-7
```

BPF Static Keys: let’s build API, v2

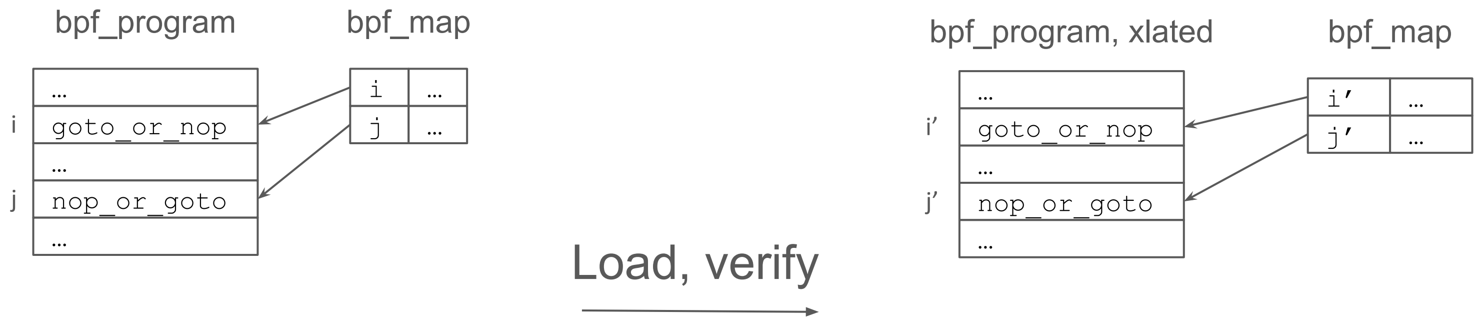
To implement Static Keys the following primitives are needed:

- Two new BPF instructions *goto_or_nop/nop_or_goto* (so that the verifier check such instructions as JA with two branches)
- A mechanism to reference a *set* of such instructions
- A mechanism to patch all instructions referenced by a *set*:
    - `syscall/kfunc(`*`set`*`, on/off)`
- Typical way to represent objects in BPF is a map, so let’s introduce a new map, BPF_MAP_TYPE_INSN_SET

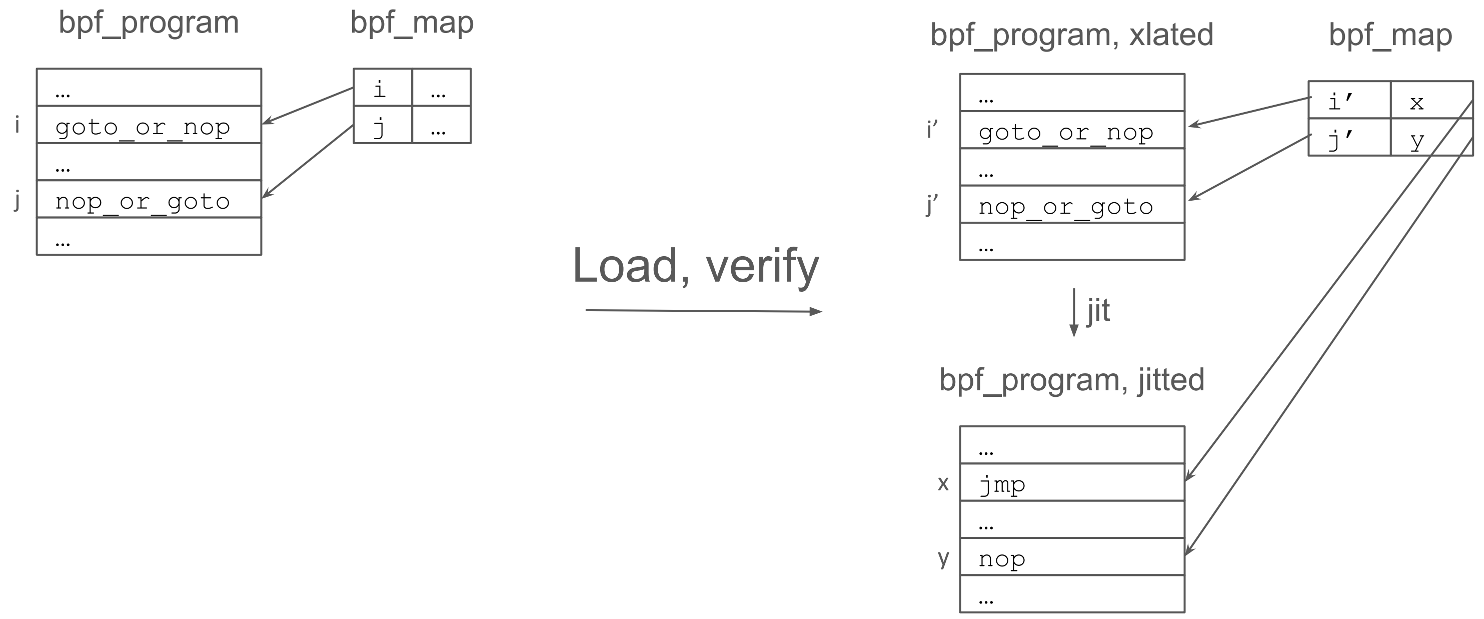
BPF Static Keys: let’s build API, v2, comics



BPF Static Keys: let’s build API, v2, comics



BPF Static Keys: let’s build API, v2, comics



BPF Static Keys: let’s build API, v2, comics

bpf_program, xlated

	…
i	`goto_or_nop`
	…
j	`nop_or_goto`
	…

bpf_map

`i`	`x`
`j`	`y`

bpf_program, jitted
syscall(BPF_STATIC_KEY_UPDATE, map_fd, 1)

	…
x	`jmp`
	…
y	`nop`
	…

bpf_program, jitted
syscall(BPF_STATIC_KEY_UPDATE, map_fd, 0)

	…
x	`nop`
	…
y	`jmp`
	…

BPF_MAP_TYPE_INSN_SET

The new map works as follows:

- Before program load a map is populated with instructions offsets
- On program load map becomes read-only to userspace (and it’s always read-only on the BPF side)
- On program load every instructions in this map is relocated when instructions are added/deleted
- During JIT native instructions/offsets are saved in the map

BPF_MAP_TYPE_INSN_SET, continued

A new API should be added to PROG_LOAD

- INSN_SET map (in the context of static keys) is not referenced by a BPF program, so we need to mechanism to tell the verifier about it
- Andrii proposed to use *attr.fd_array* by adding a new field *attr.fd_array_cnt*, however, it turned out that this array is sparse
- So, I propose to add two new fields to attrs:
    - *attr.bind_fd_array/attr.bind_fd_array_cnt*
- Functionality is similar to *bpf(BPF_PROG_BIND_MAP)*, but atomic

BPF Static Keys API

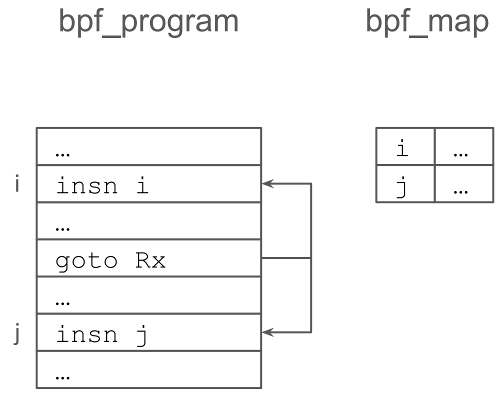
In summary, to support static keys this is required to:

- Add new instructions *goto_or_nop/nop_or_goto*
- Add new map BPF_MAP_TYPE_INSN_SET
- Add new *attr.bind_fd_array/attr.bind_fd_array_cnt* fields
- Add a new syscall (and, maybe, kfunc)

    -
      ```
      bpf(STATIC_KEY_UPDATE,
              attrs={.key = map_fd, .on = <bool>})
      ```

INSN_SET: jump tables

- The original request to generalize static keys implementation was to implement jump tables in BPF
- With INSN_SET we can implement *goto Rx* (or *goto *Rx*)



INSN_SET: jump tables

- Verifier must check that *Rx* was loaded from a map of type INSN_SET, say M
- The *goto rX* instruction must reference the same map M, so that *visit_insn()* can build a proper graph
- Something like
  BPF_JMP | **BPF_X** |BPF_JA, SRC=Rx, DST=0, off=0, imm=fd(M)
- Some care is needed to verify access to the map from BPF programs (readonly access)

Jump tables: use cases and C interface

- Implement long switches vs. a long list of if..else
- Implement interpreters:
  *void *array[] = { &&spam, &&eggs, &&bacon };*
  …
  *goto *array[i];*
- Can this be automatically translated to proper low-level code?
  - Create a map M_array for array
  - *goto *array[i]* translates to
    - *R1 = map_lookup(M_array, i)*
    - *goto *R1   # .imm=fd(M_array)*
- **Q**: how hard is the llvm/gcc part?

Questions?