ISOVALENT
(now part of Cisco)

Update on Static Keys

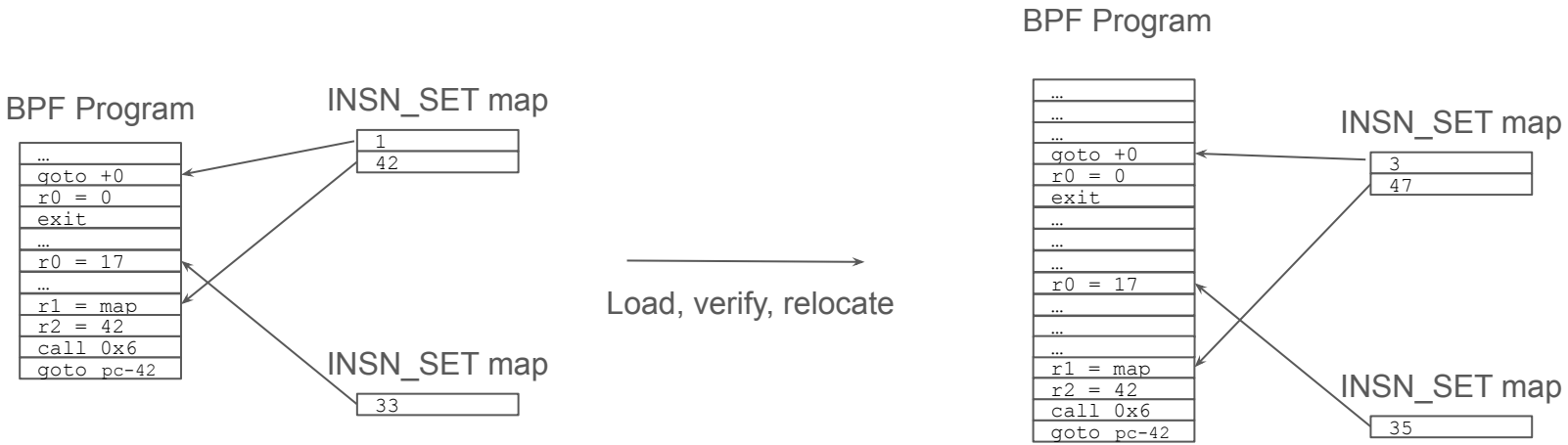
LSF/MM/BPF 2025

Anton Protopopov

BPF Static Keys: RFC

- I’ve recently posted an RFC which implements instruction sets and static keys
- Let’s take a look at API to be on the same page
- Then there are a few open question
- See [1], [2], [3] for more details on the [evolution of] design

A new BPF map: Instruction Set



Instruction Sets: API

```
struct {
        __uint(type, BPF_MAP_TYPE_INSN_SET);
        __type(key, __u32);
        __type(value, __u32);
        __uint(max_entries, N);
} insn_set SEC(".maps")
```

Instruction Sets: API, continued

```
bpf_map_create(BPF_MAP_TYPE_INSN_SET, "name", 4, 4, N, NULL);

for (i = 0; i < N; i++)
        bpf_map_update_elem(map_fd, &i, &val[i], 0);

bpf_map_freeze(map_fd);
```

Instruction Sets: API, continued

```
LIBBPF_OPTS(bpf_prog_load_opts, opts);

opts.fd_array = &map_fd;
opts.fd_array_cnt = 1;

return bpf_prog_load(type, NULL, "GPL", insns, insn_cnt, &opts);
```

(Now the prog is loaded, and the map can be dumped by userspace. See the `prog_tests/bpf_insn_set.c` selftest for examples.)

Instruction Sets: API

- In its simplest form INSN_SET can only be used for debugging. (If this turns out to be useful, this is possible to add xlated -> jitted info as well.)
- For practical use, there will be added more flavours:
  - `BPF_F_STATIC_KEY:` static keys
  - `BPF_F_CALL_TABLE:` indirect calls
  - `BPF_F_JUMP_TABLE:` indirect jumps

Static Keys: Kernel API

```
struct {
        __uint(type, BPF_MAP_TYPE_INSN_SET);
        __type(key, __u32);
        __type(value, __u32);
        __uint(max_entries, N);
        __uint(map_extra, BPF_F_STATIC_KEY);
} key SEC(".maps")
```

Static Keys: Kernel API

```
struct {
        __uint(type, BPF_MAP_TYPE_INSN_SET);
        __type(key, __u32);
        __type(value, __u32);
        __uint(max_entries, N);
        __uint(map_extra, BPF_F_STATIC_KEY);
} key SEC(".maps")
```

RFC defines this helper (but also see next slides)

```
DEFINE_STATIC_KEY(key);
```

Static Keys: Kernel API

```
SEC("smth")
int check_one_key_likely(void *ctx)
{
        if (bpf_static_branch_unlikely(&key))
                do_something_unlikely;
        else
                do_something_by_default;

        return 0;
}
```

Static Keys: libbpf

- Libbpf finds all the required static keys info in `.static_keys` and `.rel.static_keys`, creates and freezes maps, populates the `fd_array/fd_array_cnt`
- Now the `bpf(STATIC_KEY_UPDATE, on/off)` syscall can be used to toggle the branches on/off
- uAPI problem: the key is only defined for one program (see the next slides)

BPF Static Keys: problems with API

```
DEFINE_STATIC_KEY(key);

[static] void foo(void)
{
        if (bpf_static_branch_unlikely(&key))
                ...;
}

SEC("smth") int prog1(void *ctx)
{
        if (bpf_static_branch_likely(&key))
                ...;
        ...
}

SEC("smth") int prog2(void *ctx)
{
        if (bpf_static_branch_likely(&key))
                ...;
        ...
}
```

BPF Static Keys: problems with API

```
DEFINE_STATIC_KEY(key);

[static] void foo(void)
{
        if (bpf_static_branch_unlikely(&key))
                ...;
}

SEC("smth") int prog1(void *ctx)
{
        if (bpf_static_branch_likely(&key))
                ...;
        ...
}

SEC("smth") int prog2(void *ctx)
{
        if (bpf_static_branch_likely(&key))
                ...;
        ...
}
```

From “normal” BPF perspective, this code is correct. However, one particular static key only makes sense in the context of one program. Even if `prog1`/`prog2` do not use the key directly, for the sub-program `foo()` the offsets will be different on each load.

Static Keys: libbpf

- So, on object load libbpf should actually create multiple instances of key: prog1.key, prog2.key
- Should work fine, but now users have to keep track of the keys, e.g., when adding/removing progs to/from an object
- Better to provide a wrapper: `bpf_object__static_key_update()`
- **Should this be done on object-level? Generated for a skeleton? Alternatives?**

BPF Static Keys: which instruction to use?

- Namely, may_goto vs. “special” BPF_JA
- (99% answer is may_goto; RFC still uses BPF_JA and thus should be changed)

Thanks!