BPF drgn tools

drgn scripts is a convenient and easy to use mechanism to retrieve arbitrary kernel data structures. drgn is not relying on kernel UAPI to read the data. Instead it's reading directly from `/proc/kcore` or vmcore and pretty prints the data based on DWARF debug information from vmlinux.

This document describes BPF related drgn tools.

See drgn/tools for all tools available at the moment and drgn/doc for more details on drgn itself.

bpf_inspect.py

Description

bpf_inspect.py is a tool intended to inspect BPF programs and maps. It can iterate over all programs and maps in the system and print basic information about these objects, including id, type and name.

The main use-case bpf_inspect.py covers is to show BPF programs of types `BPF_PROG_TYPE_EXT` and `BPF_PROG_TYPE_TRACING` attached to other BPF programs via `freplace`/`fentry`/`fexit` mechanisms, since there is no user-space API to get this information.

Getting started

List BPF programs (full names are obtained from BTF):

```
% sudo bpf_inspect.py prog
    27: BPF_PROG_TYPE_TRACEPOINT         tracepoint__tcp__tcp_send_reset
  4632: BPF_PROG_TYPE_CGROUP_SOCK_ADDR   tw_ipt_bind
 49464: BPF_PROG_TYPE_RAW_TRACEPOINT     raw_tracepoint__sched_process_exit
```

List BPF maps:

```
% sudo bpf_inspect.py map
  2577: BPF_MAP_TYPE_HASH                tw_ipt_vips
  4050: BPF_MAP_TYPE_STACK_TRACE         stack_traces
  4069: BPF_MAP_TYPE_PERCPU_ARRAY        ned_dctcp_cntr
```

Find BPF programs attached to BPF program `test_pkt_access`:

```
% sudo bpf_inspect.py p | grep test_pkt_access
   650: BPF_PROG_TYPE_SCHED_CLS          test_pkt_access
   654: BPF_PROG_TYPE_TRACING            test_main                        linked:[650->25: BPF_TRAMP_FEXIT test_pkt_access->tes
   655: BPF_PROG_TYPE_TRACING            test_subprog1                    linked:[650->29: BPF_TRAMP_FEXIT test_pkt_access->tes
   656: BPF_PROG_TYPE_TRACING            test_subprog2                    linked:[650->31: BPF_TRAMP_FEXIT test_pkt_access->tes
   657: BPF_PROG_TYPE_TRACING            test_subprog3                    linked:[650->21: BPF_TRAMP_FEXIT test_pkt_access->tes
   658: BPF_PROG_TYPE_EXT                new_get_skb_len                  linked:[650->16: BPF_TRAMP_REPLACE test_pkt_access->g
   659: BPF_PROG_TYPE_EXT                new_get_skb_ifindex              linked:[650->23: BPF_TRAMP_REPLACE test_pkt_access->g
   660: BPF_PROG_TYPE_EXT                new_get_constant                 linked:[650->19: BPF_TRAMP_REPLACE test_pkt_access->g
```

It can be seen that there is a program `test_pkt_access`, id 650 and there are multiple other tracing and ext programs attached to functions in `test_pkt_access`.

For example the line:

```
658: BPF_PROG_TYPE_EXT                new_get_skb_len                  linked:[650->16: BPF_TRAMP_REPLACE test_pkt_access->get_
```

, means that BPF program id 658, type `BPF_PROG_TYPE_EXT`, name `new_get_skb_len` replaces (`BPF_TRAMP_REPLACE`) function `get_skb_len()` that has BTF id 16 in BPF program id 650, name `test_pkt_access`.

Getting help:

System Message: WARNING/2 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\bpf\[linux-master][Documentation][bpf]drgn.rst`, line 75)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

    % sudo bpf_inspect.py
    usage: bpf_inspect.py [-h] {prog,p,map,m} ...

    drgn script to list BPF programs or maps and their properties
    unavailable via kernel API.

    See https://github.com/osandov/drgn/ for more details on drgn.

    optional arguments:
      -h, --help      show this help message and exit

    subcommands:
      {prog,p,map,m}
        prog (p)      list BPF programs
        map (m)       list BPF maps
```

Customization

The script is intended to be customized by developers to print relevant information about BPF programs, maps and other objects.

For example, to print `struct bpf_prog_aux` for BPF program id 53077:

System Message: WARNING/2 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\bpf\[linux-master][Documentation][bpf]drgn.rst`, line 101)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

    % git diff
    diff --git a/tools/bpf_inspect.py b/tools/bpf_inspect.py
    index 650e228..aea2357 100755
    --- a/tools/bpf_inspect.py
    +++ b/tools/bpf_inspect.py
    @@ -112,7 +112,9 @@ def list_bpf_progs(args):
             if linked:
                 linked = f" linked:[{linked}]"

    -        print(f"{id_:>6}: {type_:32} {name:32} {linked}")
    +        if id_ == 53077:
    +            print(f"{id_:>6}: {type_:32} {name:32}")
    +            print(f"{bpf_prog.aux}")
```

```
        def list_bpf_maps(args):
```

It produces the output:

```
% sudo bpf_inspect.py p
 53077: BPF_PROG_TYPE_XDP                tw_xdp_policer
*(struct bpf_prog_aux *)0xffff8893fad4b400 = {
        .refcnt = (atomic64_t){
                .counter = (long)58,
        },
        .used_map_cnt = (u32)1,
        .max_ctx_offset = (u32)8,
        .max_pkt_offset = (u32)15,
        .max_tp_access = (u32)0,
        .stack_depth = (u32)8,
        .id = (u32)53077,
        .func_cnt = (u32)0,
        .func_idx = (u32)0,
        .attach_btf_id = (u32)0,
        .linked_prog = (struct bpf_prog *)0x0,
        .verifier_zext = (bool)0,
        .offload_requested = (bool)0,
        .attach_btf_trace = (bool)0,
        .func_proto_unreliable = (bool)0,
        .trampoline_prog_type = (enum bpf_tramp_prog_type)BPF_TRAMP_FENTRY,
        .trampoline = (struct bpf_trampoline *)0x0,
        .tramp_hlist = (struct hlist_node){
                .next = (struct hlist_node *)0x0,
                .pprev = (struct hlist_node **)0x0,
        },
        .attach_func_proto = (const struct btf_type *)0x0,
        .attach_func_name = (const char *)0x0,
        .func = (struct bpf_prog **)0x0,
        .jit_data = (void *)0x0,
        .poke_tab = (struct bpf_jit_poke_descriptor *)0x0,
        .size_poke_tab = (u32)0,
        .ksym_tnode = (struct latch_tree_node){
                .node = (struct rb_node [2]){
                        {
                                .__rb_parent_color = (unsigned long)18446612956263126665,
                                .rb_right = (struct rb_node *)0x0,
                                .rb_left = (struct rb_node *)0xffff88a0be3d0088,
                        },
                        {
                                .__rb_parent_color = (unsigned long)18446612956263126689,
                                .rb_right = (struct rb_node *)0x0,
                                .rb_left = (struct rb_node *)0xffff88a0be3d00a0,
                        },
                },
        },
        .ksym_lnode = (struct list_head){
                .next = (struct list_head *)0xffff88bf481830b8,
                .prev = (struct list_head *)0xffff888309f536b8,
        },
        .ops = (const struct bpf_prog_ops *)xdp_prog_ops+0x0 = 0xffffffff820fa350,
        .used_maps = (struct bpf_map **)0xffff889ff795de98,
        .prog = (struct bpf_prog *)0xffffc9000cf2d000,
        .user = (struct user_struct *)root_user+0x0 = 0xffffffff82444820,
        .load_time = (u64)2408348759285319,
        .cgroup_storage = (struct bpf_map *[2]){},
        .name = (char [16])"tw_xdp_policer",
        .security = (void *)0xffff889ff795d548,
        .offload = (struct bpf_prog_offload *)0x0,
        .btf = (struct btf *)0xffff8890ce6d0580,
        .func_info = (struct bpf_func_info *)0xffff889ff795d240,
        .func_info_aux = (struct bpf_func_info_aux *)0xffff889ff795de20,
        .linfo = (struct bpf_line_info *)0xffff888a707afc00,
        .jited_linfo = (void **)0xffff8893fad48600,
        .func_info_cnt = (u32)1,
        .nr_linfo = (u32)37,
        .linfo_idx = (u32)0,
        .num_exentries = (u32)0,
        .extable = (struct exception_table_entry *)0xffffffffa032d950,
        .stats = (struct bpf_prog_stats *)0x603fe3a1f6d0,
        .work = (struct work_struct){
                .data = (atomic_long_t){
                        .counter = (long)0,
                },
                .entry = (struct list_head){
                        .next = (struct list_head *)0x0,
                        .prev = (struct list_head *)0x0,
                },
                .func = (work_func_t)0x0,
        },
        .rcu = (struct callback_head){
                .next = (struct callback_head *)0x0,
                .func = (void (*)(struct callback_head *))0x0,
        },
}
```