MyWebUniversity
python
Rocky Linux
Manual Pages
FPDF Library
PDF generator
FPDF
Full credit is given to the above companies including the OS that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'bpftool-btf.8'

$ man bpftool-btf.8

BPFTOOL-BTF(8) BPFTOOL-BTF(8)

NAME

bpftool-btf - tool for inspection of BTF data

SYNOPSIS

bpftool [OPTIONS] btf COMMAND

OPTIONS := { { -j | --json } [{ -p | --pretty }] | { -d | --debug }

| { -l | --legacy } | { -B | --base-btf } }

COMMANDS := { dump | help }

BTF COMMANDS

bpftool btf { show | list } [id BTF_ID]

bpftool btf dump BTF_SRC [format FORMAT]

bpftool btf help

BTF_SRC := { id BTF_ID | prog PROG | map MAP [{key | value | kv | all}] | file FILE }

FORMAT := { raw | c }

MAP := { id MAP_ID | pinned FILE }

PROG := { id PROG_ID | pinned FILE | tag PROG_TAG }

DESCRIPTION

bpftool btf { show | list } [id BTF_ID]

Show information about loaded BTF objects. If a BTF ID is specified, show information only about given BTF object, oth? erwise list all BTF objects currently loaded on the system.

Since Linux 5.8 bpftool is able to discover information about processes that hold open file descriptors (FDs) against BTF objects. On such kernels bpftool will automatically emit this information as well.

bpftool btf dump BTF_SRC

Dump BTF entries from a given BTF_SRC.

When id is specified, BTF object with that ID will be loaded and all its BTF types emitted.

When map is provided, it's expected that map has associated BTF object with BTF types describing key and value. It's pos? sible to select whether to dump only BTF type(s) associated with key (key), value (value), both key and value (kv), or all BTF types present in associated BTF object (all). If not specified, kv is assumed.

When prog is provided, it's expected that program has associ? ated BTF object with BTF types.

When specifying FILE, an ELF file is expected, containing .BTF section with well-defined BTF binary format data, typi? cally produced by clang or pahole.

format option can be used to override default (raw) output format. Raw (raw) or C-syntax (c) output formats are sup? ported.

bpftool btf help

Print short help message.

OPTIONS

-h, --help

Print short help message (similar to bpftool help).

-V, --version

Print bpftool's version number (similar to bpftool version), the number of the libbpf version in use, and optional fea?

tures that were included when bpftool was compiled. Optional
features include linking against libbfd to provide the disas?
sembler for JIT-ted programs (bpftool prog dump jited) and
usage of BPF skeletons (some features like bpftool prog pro?
file or showing pids associated to BPF objects may rely on
it).

-j, --json

Generate JSON output. For commands that cannot produce JSON,
this option has no effect.

-p, --pretty

Generate human-readable JSON output. Implies -j.

-d, --debug

Print all logs available, even debug-level information. This
includes logs from libbpf as well as from the verifier, when
attempting to load programs.

-l, --legacy

Use legacy libbpf mode which has more relaxed BPF program re?
quirements. By default, bpftool has more strict requirements
about section names, changes pinning logic and doesn't sup?
port some of the older non-BTF map declarations.

See
https://github.com/libbpf/libbpf/wiki/Libbpf:-the-road-to-v1.0
for details.

-B, --base-btf FILE

Pass a base BTF object. Base BTF objects are typically used
with BTF objects for kernel modules. To avoid duplicating all
kernel symbols required by modules, BTF objects for modules
are "split", they are built incrementally on top of the ker?
nel (vmlinux) BTF object. So the base BTF reference should
usually point to the kernel BTF.

When the main BTF object to process (for example, the module
BTF to dump) is passed as a FILE, bpftool attempts to autode?
tect the path for the base object, and passing this option is

        optional. When the main BTF object is passed through other
        handles, this option becomes necessary.

EXAMPLES

```
# bpftool btf dump id 1226
  [1] PTR '(anon)' type_id=2
  [2] STRUCT 'dummy_tracepoint_args' size=16 vlen=2
          'pad' type_id=3 bits_offset=0
          'sock' type_id=4 bits_offset=64
  [3] INT 'long long unsigned int' size=8 bits_offset=0 nr_bits=64 encoding=(none)
  [4] PTR '(anon)' type_id=5
  [5] FWD 'sock' fwd_kind=union
```

This gives an example of default output for all supported BTF kinds.

```
$ cat prog.c
  struct fwd_struct;
  enum my_enum {
          VAL1 = 3,
          VAL2 = 7,
  };
  typedef struct my_struct my_struct_t;
  struct my_struct {
          const unsigned int const_int_field;
          int bitfield_field: 4;
          char arr_field[16];
          const struct fwd_struct *restrict fwd_field;
          enum my_enum enum_field;
          volatile my_struct_t *typedef_ptr_field;
  };
  union my_union {
          int a;
          struct my_struct b;
  };
  struct my_struct struct_global_var __attribute__((section("data_sec"))) = {
          .bitfield_field = 3,
        .enum_field = VAL1,
  };
  int global_var __attribute__((section("data_sec"))) = 7;
  __attribute__((noinline))
  int my_func(union my_union *arg1, int arg2)
  {
        static int static_var __attribute__((section("data_sec"))) = 123;
        static_var++;
        return static_var;
  }
$ bpftool btf dump file prog.o
  [1] PTR '(anon)' type_id=2
  [2] UNION 'my_union' size=48 vlen=2
        'a' type_id=3 bits_offset=0
        'b' type_id=4 bits_offset=0
  [3] INT 'int' size=4 bits_offset=0 nr_bits=32 encoding=SIGNED
  [4] STRUCT 'my_struct' size=48 vlen=6
        'const_int_field' type_id=5 bits_offset=0
        'bitfield_field' type_id=3 bits_offset=32 bitfield_size=4
        'arr_field' type_id=8 bits_offset=40
        'fwd_field' type_id=10 bits_offset=192
        'enum_field' type_id=14 bits_offset=256
        'typedef_ptr_field' type_id=15 bits_offset=320
  [5] CONST '(anon)' type_id=6
  [6] INT 'unsigned int' size=4 bits_offset=0 nr_bits=32 encoding=(none)
  [7] INT 'char' size=1 bits_offset=0 nr_bits=8 encoding=SIGNED
  [8] ARRAY '(anon)' type_id=7 index_type_id=9 nr_elems=16
  [9] INT '__ARRAY_SIZE_TYPE__' size=4 bits_offset=0 nr_bits=32 encoding=(none)
  [10] RESTRICT '(anon)' type_id=11
  [11] PTR '(anon)' type_id=12
  [12] CONST '(anon)' type_id=13
  [13] FWD 'fwd_struct' fwd_kind=union
  [14] ENUM 'my_enum' size=4 vlen=2
```

```
          'VAL1' val=3

          'VAL2' val=7

  [15] PTR '(anon)' type_id=16

  [16] VOLATILE '(anon)' type_id=17

  [17] TYPEDEF 'my_struct_t' type_id=4

  [18] FUNC_PROTO '(anon)' ret_type_id=3 vlen=2

          'arg1' type_id=1

          'arg2' type_id=3

  [19] FUNC 'my_func' type_id=18

  [20] VAR 'struct_global_var' type_id=4, linkage=global-alloc

  [21] VAR 'global_var' type_id=3, linkage=global-alloc

  [22] VAR 'my_func.static_var' type_id=3, linkage=static

  [23] DATASEC 'data_sec' size=0 vlen=3

          type_id=20 offset=0 size=48

          type_id=21 offset=0 size=4

          type_id=22 offset=52 size=4
```

The following commands print BTF types associated with specified map's

key, value, both key and value, and all BTF types, respectively. By de?

fault, both key and value types will be printed.

```
# bpftool btf dump map id 123 key

  [39] TYPEDEF 'u32' type_id=37

# bpftool btf dump map id 123 value

  [86] PTR '(anon)' type_id=87

# bpftool btf dump map id 123 kv

  [39] TYPEDEF 'u32' type_id=37

  [86] PTR '(anon)' type_id=87

# bpftool btf dump map id 123 all

  [1] PTR '(anon)' type_id=0

  .

  .

  .

  [2866] ARRAY '(anon)' type_id=52 index_type_id=51 nr_elems=4
```

All the standard ways to specify map or program are supported:

```
# bpftool btf dump map id 123

# bpftool btf dump map pinned /sys/fs/bpf/map_name

# bpftool btf dump prog id 456

# bpftool btf dump prog tag b88e0a09b1d9759d

# bpftool btf dump prog pinned /sys/fs/bpf/prog_name

# bpftool btf dump file /sys/kernel/btf/i2c_smbus

(or)

# I2C_SMBUS_ID=$(bpftool btf show -p | jq '.[] | select(.name=="i2c_smbus").id')

# bpftool btf dump id ${I2C_SMBUS_ID} -B /sys/kernel/btf/vmlinux

  [104848] STRUCT 'i2c_smbus_alert' size=40 vlen=2

          'alert' type_id=393 bits_offset=0

          'ara' type_id=56050 bits_offset=256

  [104849] STRUCT 'alert_data' size=12 vlen=3

          'addr' type_id=16 bits_offset=0

          'type' type_id=56053 bits_offset=32

          'data' type_id=7 bits_offset=64

  [104850] PTR '(anon)' type_id=104848

  [104851] PTR '(anon)' type_id=104849

  [104852] FUNC 'i2c_register_spd' type_id=84745 linkage=static

  [104853] FUNC 'smbalert_driver_init' type_id=1213 linkage=static

  [104854] FUNC_PROTO '(anon)' ret_type_id=18 vlen=1

          'ara' type_id=56050

  [104855] FUNC 'i2c_handle_smbus_alert' type_id=104854 linkage=static

  [104856] FUNC 'smbalert_remove' type_id=104854 linkage=static

  [104857] FUNC_PROTO '(anon)' ret_type_id=18 vlen=2

          'ara' type_id=56050

          'id' type_id=56056

  [104858] FUNC 'smbalert_probe' type_id=104857 linkage=static

  [104859] FUNC 'smbalert_work' type_id=9695 linkage=static

  [104860] FUNC 'smbus_alert' type_id=71367 linkage=static

  [104861] FUNC 'smbus_do_alert' type_id=84827 linkage=static
```

SEE ALSO

bpf(2), bpf-helpers(7), bpftool(8), bpftool-cgroup(8), bpftool-fea?

ture(8), bpftool-gen(8), bpftool-iter(8), bpftool-link(8), bpftool-map(8), bpftool-net(8), bpftool-perf(8), bpftool-prog(8), bpftool-struct_ops(8)