

Name

bpftool-prog – tool for inspection and simple manipulation of eBPF progs

SYNOPSIS

bpftool [*OPTIONS*] **prog** *COMMAND*

OPTIONS := { { **-j** | **--json** } [{ **-p** | **--pretty** }] | { **-d** | **--debug** } | { **-f** | **--bpffs** } | { **-m** | **--map-compat** } | { **-n** | **--nomount** } | { **-L** | **--use-loader** } | [{ **-S** | **--sign** } **-k** <private_key.pem> **-i** <certificate.x509>] }

COMMANDS := { **show** | **list** | **dump xlated** | **dump jited** | **pin** | **load** | **loadall** | **help** }

PROG COMMANDS

bpftool prog { **show** | **list** } [*PROG*]

bpftool prog dump xlated *PROG* [{ **file** *FILE* | [**opcodes**] [**linum**] [**visual**] }]

bpftool prog dump jited *PROG* [{ **file** *FILE* | [**opcodes**] [**linum**] }]

bpftool prog pin *PROG* *FILE*

bpftool prog { **load** | **loadall** } *OBJ* *PATH* [**type** *TYPE*] [**map** { **idx** *IDX* | **name** *NAME* } *MAP*] [{ **offload_dev** | **xdpmeta_dev** }]

bpftool prog attach *PROG* *ATTACH_TYPE* [*MAP*]

bpftool prog detach *PROG* *ATTACH_TYPE* [*MAP*]

bpftool prog tracelog

bpftool prog tracelog [{ **stdout** | **stderr** } *PROG*]

bpftool prog run *PROG* **data_in** *FILE* [**data_out** *FILE* [**data_size_out** *L*]] [**ctx_in** *FILE* [**ctx_out** *FILE* [**ctx_size_out** *M*]]]

bpftool prog profile *PROG* [**duration** *DURATION*] *METRICS*

bpftool prog help

MAP := { **id** *MAP_ID* | **pinned** *FILE* | **name** *MAP_NAME* }

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

TYPE := {

socket | **kprobe** | **kretprobe** | **classifier** | **action** |
tracepoint | **raw_tracepoint** | **xdp** | **perf_event** | **cgroup/skb** |
cgroup/sock | **cgroup/dev** | **lwt_in** | **lwt_out** | **lwt_xmit** |
lwt_seg6local | **sockops** | **sk_skb** | **sk_msg** | **lirc_mode2** |
cgroup/bind4 | **cgroup/bind6** | **cgroup/post_bind4** | **cgroup/post_bind6** |
cgroup/connect4 | **cgroup/connect6** | **cgroup/connect_unix** |
cgroup/getpeername4 | **cgroup/getpeername6** | **cgroup/getpeername_unix** |
cgroup/getsockname4 | **cgroup/getsockname6** | **cgroup/getsockname_unix** |
cgroup/sendmsg4 | **cgroup/sendmsg6** | **cgroup/sendmsg_unix** |
cgroup/recvmmsg4 | **cgroup/recvmmsg6** | **cgroup/recvmmsg_unix** | **cgroup/sysctl** |
cgroup/getsockopt | **cgroup/setsockopt** | **cgroup/sock_release** |
struct_ops | **fentry** | **fexit** | **freplace** | **sk_lookup**

}

ATTACH_TYPE := {

sk_msg_verdict | **sk_skb_verdict** | **sk_skb_stream_verdict** |
sk_skb_stream_parser | **flow_dissector**

}

METRICS := {

cycles | **instructions** | **l1d_loads** | **llc_misses** |
itlb_misses | **dtlb_misses**

}

DESCRIPTION

bpftool prog { **show** | **list** } [*PROG*]

Show information about loaded programs. If *PROG* is specified show information only about given programs, otherwise list all programs currently loaded on the system. In case of **tag** or

name, *PROG* may match several programs which will all be shown.

Output will start with program ID followed by program type and zero or more named attributes (depending on kernel version).

Since Linux 5.1 the kernel can collect statistics on BPF programs (such as the total time spent running the program, and the number of times it was run). If available, bpftool shows such statistics. However, the kernel does not collect them by defaults, as it slightly impacts performance on each program run. Activation or deactivation of the feature is performed via the **kernel.bpf_stats_enabled** sysctl knob.

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

bpftool prog dump xlated *PROG* [{ **file** *FILE* | [**opcodes**] [**linum**] [**visual**] }

Dump eBPF instructions of the programs from the kernel. By default, eBPF will be disassembled and printed to standard output in human-readable format. In this case, **opcodes** controls if raw opcodes should be printed as well.

In case of **tag** or **name**, *PROG* may match several programs which will all be dumped. However, if **file** or **visual** is specified, *PROG* must match a single program.

If **file** is specified, the binary image will instead be written to *FILE*.

If **visual** is specified, control flow graph (CFG) will be built instead, and eBPF instructions will be presented with CFG in DOT format, on standard output.

If the programs have *line_info* available, the source line will be displayed. If **linum** is specified, the filename, line number and line column will also be displayed.

bpftool prog dump jited *PROG* [{ **file** *FILE* | [**opcodes**] [**linum**] }

Dump jited image (host machine code) of the program.

If *FILE* is specified image will be written to a file, otherwise it will be disassembled and printed to stdout. *PROG* must match a single program when **file** is specified.

opcodes controls if raw opcodes will be printed.

If the prog has *line_info* available, the source line will be displayed. If **linum** is specified, the filename, line number and line column will also be displayed.

bpftool prog pin *PROG* *FILE*

Pin program *PROG* as *FILE*.

Note: *FILE* must be located in *bpffs* mount. It must not contain a dot character ('.'), which is reserved for future extensions of *bpffs*.

bpftool prog { load | loadall } OBJ *PATH* [**type** *TYPE*] [**map { idx** *IDX* | **name** *NAME* } *MAP*] [{ **ofload_dev** | **xdpmeta_dev** } *NAME*] [**pinmaps** *MAP_DIR*] [**autoattach**] [**kernel_btf** *BTF_FILE*]

Load bpf program(s) from binary *OBJ* and pin as *PATH*. **bpftool prog load** pins only the first program from the *OBJ* as *PATH*. **bpftool prog loadall** pins all programs from the *OBJ* under *PATH* directory. **type** is optional, if not specified program type will be inferred from section names. By default bpftool will create new maps as declared in the ELF object being loaded. **map** parameter allows for the reuse of existing maps. It can be specified multiple times, each time for a different map. *IDX* refers to index of the map to be replaced in the ELF file counting from 0, while *NAME* allows to replace a map by name. *MAP* specifies the map to use, referring to it by **id** or through a

pinned file. If **offload_dev** *NAME* is specified program will be loaded onto given networking device (offload). If **xdpmeta_dev** *NAME* is specified program will become device-bound without offloading, this facilitates access to XDP metadata. Optional **pinmaps** argument can be provided to pin all maps under *MAP_DIR* directory.

If **autoattach** is specified program will be attached before pin. In that case, only the link (representing the program attached to its hook) is pinned, not the program as such, so the path won't show in **bpftool prog show -f**, only show in **bpftool link show -f**. Also, this only works when bpftool (libbpf) is able to infer all necessary information from the object file, in particular, it's not supported for all program types. If a program does not support autoattach, bpftool falls back to regular pinning for that program instead.

The **kernel_btf** option allows specifying an external BTF file to replace the system's own vmlinux BTF file for CO-RE relocations. Note that any other feature relying on BTF (such as fentry/fexit programs, struct_ops) requires the BTF file for the actual kernel running on the host, often exposed at */sys/kernel/btf/vmlinux*.

Note: *PATH* must be located in *bpffs* mount. It must not contain a dot character ('.'), which is reserved for future extensions of *bpffs*.

bpftool prog attach *PROG ATTACH_TYPE [MAP]*

Attach bpf program *PROG* (with type specified by *ATTACH_TYPE*). Most *ATTACH_TYPE*s require a *MAP* parameter, with the exception of *flow_dissector* which is attached to current networking name space.

bpftool prog detach *PROG ATTACH_TYPE [MAP]*

Detach bpf program *PROG* (with type specified by *ATTACH_TYPE*). Most *ATTACH_TYPE*s require a *MAP* parameter, with the exception of *flow_dissector* which is detached from the current networking name space.

bpftool prog tracelog

Dump the trace pipe of the system to the console (stdout). Hit <Ctrl+C> to stop printing. BPF programs can write to this trace pipe at runtime with the **bpf_trace_printk()** helper. This should be used only for debugging purposes. For streaming data from BPF programs to user space, one can use perf events (see also **bpftool-map(8)**).

bpftool prog tracelog { stdout | stderr } *PROG*

Dump the BPF stream of the program. BPF programs can write to these streams at runtime with the **bpf_stream_vprintk_impl()** kfunc. The kernel may write error messages to the standard error stream. This facility should be used only for debugging purposes.

bpftool prog run *PROG data_in FILE [data_out FILE [data_size_out L]] [ctx_in FILE [ctx_out FILE [ctx_size_out M]]] [repeat N]*

Run BPF program *PROG* in the kernel testing infrastructure for BPF, meaning that the program works on the data and context provided by the user, and not on actual packets or monitored functions etc. Return value and duration for the test run are printed out to the console.

Input data is read from the *FILE* passed with **data_in**. If this *FILE* is "-", input data is read from standard input. Input context, if any, is read from *FILE* passed with **ctx_in**. Again, "-" can be used to read from standard input, but only if standard input is not already in use for input data. If a *FILE* is passed with **data_out**, output data is written to that file. Similarly, output context is written to the *FILE* passed with **ctx_out**. For both output flows, "-" can be used to print to the standard output (as plain text, or JSON if relevant option was passed). If output keywords are omitted, output data and context are discarded. Keywords **data_size_out** and **ctx_size_out** are used to pass the size (in bytes) for the output buffers to the kernel, although the default of 32 kB should be more than enough for most cases.

Keyword **repeat** is used to indicate the number of consecutive runs to perform. Note that output data and context printed to files correspond to the last of those runs. The duration printed out at the end of the runs is an average over all runs performed by the command.

Not all program types support test run. Among those which do, not all of them can take the **ctx_in/ctx_out** arguments. bpftool does not perform checks on program types.

bpftool prog profile PROG [duration DURATION] METRICS

Profile *METRICS* for bpf program *PROG* for *DURATION* seconds or until user hits <Ctrl+C>. *DURATION* is optional. If *DURATION* is not specified, the profiling will run up to **UINT_MAX** seconds.

bpftool prog 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 features that were included when bpftool was compiled. Optional features include linking against LLVM or libbfd to provide the disassembler for JIT-ted programs (**bpftool prog dump jited**) and usage of BPF skeletons (some features like **bpftool prog profile** 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.

-f, --bpffs

When showing BPF programs, show file names of pinned programs.

-m, --mapcompat

Allow loading maps with unknown map definitions.

-n, --nomount

Do not automatically attempt to mount any virtual file system (such as tracefs or BPF virtual file system) when necessary.

-L, --use-loader

Load program as a "loader" program. This is useful to debug the generation of such programs. When this option is in use, bpftool attempts to load the programs from the object file into the kernel, but does not pin them (therefore, the *PATH* must not be provided).

When combined with the **-d|--debug** option, additional debug messages are generated, and the execution of the loader program will use the **bpf_trace_printk()** helper to log each step of loading BTF, creating the maps, and loading the programs (see **bpftool prog tracelog** as a way to dump those messages).

-S, --sign

Enable signing of the BPF program before loading. This option must be used with **-k** and **-i**. Using this flag implicitly enables **--use-loader**.

-k <private_key.pem>

Path to the private key file in PEM format, required when signing.

-i <certificate.x509>

Path to the X.509 certificate file in PEM or DER format, required when signing.

EXAMPLES

bpftool prog show

```
10: xdp name some_prog tag 005a3d2123620c8b gpl run_time_ns 81632 run_cnt 10
    loaded_at 2017-09-29T20:11:00+0000 uid 0
    xlated 528B jited 370B memlock 4096B map_ids 10
    pids systemd(1)
```

bpftool --json --pretty prog show

```
[{
  "id": 10,
  "type": "xdp",
  "tag": "005a3d2123620c8b",
  "gpl_compatible": true,
  "run_time_ns": 81632,
  "run_cnt": 10,
  "loaded_at": 1506715860,
  "uid": 0,
  "bytes_xlated": 528,
  "jited": true,
  "bytes_jited": 370,
  "bytes_memlock": 4096,
  "map_ids": [10
],
  "pids": [{
    "pid": 1,
    "comm": "systemd"
  }
]
}
```

bpftool prog dump xlated id 10 file /tmp/t

\$ ls -l /tmp/t

```
-rw----- 1 root root 560 Jul 22 01:42 /tmp/t
```

bpftool prog dump jited tag 005a3d2123620c8b

```
0: push %rbp
1: mov %rsp,%rbp
2: sub $0x228,%rsp
3: sub $0x28,%rbp
4: mov %rbx,0x0(%rbp)
```

mount -t bpf none /sys/fs/bpf/

bpftool prog pin id 10 /sys/fs/bpf/prog

bpftool prog load ./my_prog.o /sys/fs/bpf/prog2

```
# ls -l /sys/fs/bpf/
```

```
-rw----- 1 root root 0 Jul 22 01:43 prog
-rw----- 1 root root 0 Jul 22 01:44 prog2
```

```
# bpftool prog dump jited pinned /sys/fs/bpf/prog opcodes
```

```
0:  push    %rbp
    55
1:  mov     %rsp,%rbp
    48 89 e5
4:  sub     $0x228,%rsp
    48 81 ec 28 02 00 00
b:  sub     $0x28,%rbp
    48 83 ed 28
f:  mov     %rbx,0x0(%rbp)
    48 89 5d 00
```

```
# bpftool prog load xdp1_kern.o /sys/fs/bpf/xdp1 type xdp map name rxcnt id 7
```

```
# bpftool prog show pinned /sys/fs/bpf/xdp1
```

```
9: xdp  name xdp_prog1  tag 539ec6ce11b52f98  gpl
    loaded_at 2018-06-25T16:17:31-0700  uid 0
    xlated 488B  jited 336B  memlock 4096B  map_ids 7
```

```
# rm /sys/fs/bpf/xdp1
```

```
# bpftool prog profile id 337 duration 10 cycles instructions llc_misses
```

```
51397 run_cnt
40176203 cycles (83.05%)
42518139 instructions # 1.06 insns per cycle (83.39%)
123 llc_misses # 2.89 LLC misses per million insns (83.15%)
```

Output below is for the trace logs.

Run in separate terminals:

```
# bpftool prog tracelog
```

```
# bpftool prog load -L -d file.o
```

```
bpftool-620059 [004] d... 2634685.517903: bpf_trace_printk: btf_load size 665 r=
bpftool-620059 [004] d... 2634685.517912: bpf_trace_printk: map_create sample_ma
bpftool-620059 [004] d... 2634685.517997: bpf_trace_printk: prog_load sample ins
bpftool-620059 [004] d... 2634685.517999: bpf_trace_printk: close(5) = 0
```

SEE ALSO

bpf(2), **bpf-helpers(7)**, **bpftool(8)**, **bpftool-btf(8)**, **bpftool-cgroup(8)**, **bpftool-feature(8)**, **bpftool-gen(8)**, **bpftool-iter(8)**, **bpftool-link(8)**, **bpftool-map(8)**, **bpftool-net(8)**, **bpftool-perf(8)**, **bpftool-struct_ops(8)**, **bpftool-token(8)**