BPF_PROG_TYPE_FLOW_DISSECTOR

Overview

Flow dissector is a routine that parses metadata out of the packets. It's used in the various places in the networking subsystem (RFS, flow hash, etc).

BPF flow dissector is an attempt to reimplement C-based flow dissector logic in BPF to gain all the benefits of BPF verifier (namely, limits on the number of instructions and tail calls).

API

BPF flow dissector programs operate on an `__sk_buff`. However, only the limited set of fields is allowed: `data`, `data_end` and `flow_keys`. `flow_keys` is `struct bpf_flow_keys` and contains flow dissector input and output arguments.

The inputs are:

- `nhoff` - initial offset of the networking header
- `thoff` - initial offset of the transport header, initialized to nhoff
- `n_proto` - L3 protocol type, parsed out of L2 header
- `flags` - optional flags

Flow dissector BPF program should fill out the rest of the `struct bpf_flow_keys` fields. Input arguments `nhoff/thoff/n_proto` should be also adjusted accordingly.

The return code of the BPF program is either BPF_OK to indicate successful dissection, or BPF_DROP to indicate parsing error.

__sk_buff->data

In the VLAN-less case, this is what the initial state of the BPF flow dissector looks like:

```
+------+------+------------+-----------+
| DMAC | SMAC | ETHER_TYPE | L3_HEADER |
+------+------+------------+-----------+
                           ^
                           |
                           +-- flow dissector starts here
```

```
skb->data + flow_keys->nhoff point to the first byte of L3_HEADER
flow_keys->thoff = nhoff
flow_keys->n_proto = ETHER_TYPE
```

In case of VLAN, flow dissector can be called with the two different states.

Pre-VLAN parsing:

```
+------+------+------+-----+-----------+-----------+
| DMAC | SMAC | TPID | TCI |ETHER_TYPE | L3_HEADER |
+------+------+------+-----+-----------+-----------+
                     ^
                     |
                     +-- flow dissector starts here
```

```
skb->data + flow_keys->nhoff point the to first byte of TCI
flow_keys->thoff = nhoff
flow_keys->n_proto = TPID
```

Please note that TPID can be 802.1AD and, hence, BPF program would have to parse VLAN information twice for double tagged packets.

Post-VLAN parsing:

```
+------+------+------+-----+-----------+-----------+
| DMAC | SMAC | TPID | TCI |ETHER_TYPE | L3_HEADER |
+------+------+------+-----+-----------+-----------+
                                       ^
                                       |
                                       +-- flow dissector starts here
```

```
skb->data + flow_keys->nhoff point the to first byte of L3_HEADER
flow_keys->thoff = nhoff
flow_keys->n_proto = ETHER_TYPE
```

In this case VLAN information has been processed before the flow dissector and BPF flow dissector is not required to handle it.

The takeaway here is as follows: BPF flow dissector program can be called with the optional VLAN header and should gracefully

handle both cases: when single or double VLAN is present and when it is not present. The same program can be called for both cases and would have to be written carefully to handle both cases.

Flags

`flow_keys->flags` might contain optional input flags that work as follows:

- `BPF_FLOW_DISSECTOR_F_PARSE_1ST_FRAG` - tells BPF flow dissector to continue parsing first fragment; the default expected behavior is that flow dissector returns as soon as it finds out that the packet is fragmented; used by `eth_get_headlen` to estimate length of all headers for GRO.
- `BPF_FLOW_DISSECTOR_F_STOP_AT_FLOW_LABEL` - tells BPF flow dissector to stop parsing as soon as it reaches IPv6 flow label; used by `___skb_get_hash` and `__skb_get_hash_symmetric` to get flow hash.
- `BPF_FLOW_DISSECTOR_F_STOP_AT_ENCAP` - tells BPF flow dissector to stop parsing as soon as it reaches encapsulated headers; used by routing infrastructure.

Reference Implementation

See `tools/testing/selftests/bpf/progs/bpf_flow.c` for the reference implementation and `tools/testing/selftests/bpf/flow_dissector_load.[hc]` for the loader. bpftool can be used to load BPF flow dissector program as well.

The reference implementation is organized as follows:

- `jmp_table` map that contains sub-programs for each supported L3 protocol
- `_dissect` routine - entry point; it does input `n_proto` parsing and does `bpf_tail_call` to the appropriate L3 handler

Since BPF at this point doesn't support looping (or any jumping back), jmp_table is used instead to handle multiple levels of encapsulation (and IPv6 options).

Current Limitations

BPF flow dissector doesn't support exporting all the metadata that in-kernel C-based implementation can export. Notable example is single VLAN (802.1Q) and double VLAN (802.1AD) tags. Please refer to the `struct bpf_flow_keys` for a set of information that's currently can be exported from the BPF context.

When BPF flow dissector is attached to the root network namespace (machine-wide policy), users can't override it in their child network namespaces.