BPF sk_lookup program

BPF sk_lookup program type (`BPF_PROG_TYPE_SK_LOOKUP`) introduces programmability into the socket lookup performed by the transport layer when a packet is to be delivered locally.

When invoked BPF sk_lookup program can select a socket that will receive the incoming packet by calling the `bpf_sk_assign()` BPF helper function.

Hooks for a common attach point (`BPF_SK_LOOKUP`) exist for both TCP and UDP.

Motivation

BPF sk_lookup program type was introduced to address setup scenarios where binding sockets to an address with `bind()` socket call is impractical, such as:

1. receiving connections on a range of IP addresses, e.g. 192.0.2.0/24, when binding to a wildcard address `INADRR_ANY` is not possible due to a port conflict,
2. receiving connections on all or a wide range of ports, i.e. an L7 proxy use case.

Such setups would require creating and `bind()`'ing one socket to each of the IP address/port in the range, leading to resource consumption and potential latency spikes during socket lookup.

Attachment

BPF sk_lookup program can be attached to a network namespace with `bpf(BPF_LINK_CREATE, ...)` syscall using the `BPF_SK_LOOKUP` attach type and a netns FD as attachment `target_fd`.

Multiple programs can be attached to one network namespace. Programs will be invoked in the same order as they were attached.

Hooks

The attached BPF sk_lookup programs run whenever the transport layer needs to find a listening (TCP) or an unconnected (UDP) socket for an incoming packet.

Incoming traffic to established (TCP) and connected (UDP) sockets is delivered as usual without triggering the BPF sk_lookup hook.

The attached BPF programs must return with either `SK_PASS` or `SK_DROP` verdict code. As for other BPF program types that are network filters, `SK_PASS` signifies that the socket lookup should continue on to regular hashtable-based lookup, while `SK_DROP` causes the transport layer to drop the packet.

A BPF sk_lookup program can also select a socket to receive the packet by calling `bpf_sk_assign()` BPF helper. Typically, the program looks up a socket in a map holding sockets, such as `SOCKMAP` or `SOCKHASH`, and passes a `struct bpf_sock *` to `bpf_sk_assign()` helper to record the selection. Selecting a socket only takes effect if the program has terminated with `SK_PASS` code.

When multiple programs are attached, the end result is determined from return codes of all the programs according to the following rules:

1. If any program returned `SK_PASS` and selected a valid socket, the socket is used as the result of the socket lookup.
2. If more than one program returned `SK_PASS` and selected a socket, the last selection takes effect.
3. If any program returned `SK_DROP`, and no program returned `SK_PASS` and selected a socket, socket lookup fails.
4. If all programs returned `SK_PASS` and none of them selected a socket, socket lookup continues on.

API

In its context, an instance of `struct bpf_sk_lookup`, BPF sk_lookup program receives information about the packet that triggered the socket lookup. Namely:

- IP version (`AF_INET` or `AF_INET6`),
- L4 protocol identifier (`IPPROTO_TCP` or `IPPROTO_UDP`),
- source and destination IP address,
- source and destination L4 port,
- the socket that has been selected with `bpf_sk_assign()`.

Refer to `struct bpf_sk_lookup` declaration in `linux/bpf.h` user API header, and bpf-helpers(7) man-page section for `bpf_sk_assign()` for details.

Example

See `tools/testing/selftests/bpf/prog_tests/sk_lookup.c` for the reference implementation.