BPF_PROG_TYPE_CGROUP_SOCKOPT

`BPF_PROG_TYPE_CGROUP_SOCKOPT` program type can be attached to two cgroup hooks:

- `BPF_CGROUP_GETSOCKOPT` - called every time process executes `getsockopt` system call.
- `BPF_CGROUP_SETSOCKOPT` - called every time process executes `setsockopt` system call.

The context (`struct bpf_sockopt`) has associated socket (`sk`) and all input arguments: `level`, `optname`, `optval` and `optlen`.

BPF_CGROUP_SETSOCKOPT

`BPF_CGROUP_SETSOCKOPT` is triggered *before* the kernel handling of sockopt and it has writable context: it can modify the supplied arguments before passing them down to the kernel. This hook has access to the cgroup and socket local storage.

If BPF program sets `optlen` to -1, the control will be returned back to the userspace after all other BPF programs in the cgroup chain finish (i.e. kernel `setsockopt` handling will *not* be executed).

Note, that `optlen` can not be increased beyond the user-supplied value. It can only be decreased or set to -1. Any other value will trigger `EFAULT`.

Return Type

- `0` - reject the syscall, `EPERM` will be returned to the userspace.
- `1` - success, continue with next BPF program in the cgroup chain.

BPF_CGROUP_GETSOCKOPT

`BPF_CGROUP_GETSOCKOPT` is triggered *after* the kernel handing of sockopt. The BPF hook can observe `optval`, `optlen` and `retval` if it's interested in whatever kernel has returned. BPF hook can override the values above, adjust `optlen` and reset `retval` to 0. If `optlen` has been increased above initial `getsockopt` value (i.e. userspace buffer is too small), `EFAULT` is returned.

This hook has access to the cgroup and socket local storage.

Note, that the only acceptable value to set to `retval` is 0 and the original value that the kernel returned. Any other value will trigger `EFAULT`.

Return Type

- `0` - reject the syscall, `EPERM` will be returned to the userspace.
- `1` - success: copy `optval` and `optlen` to userspace, return `retval` from the syscall (note that this can be overwritten by the BPF program from the parent cgroup).

Cgroup Inheritance

Suppose, there is the following cgroup hierarchy where each cgroup has `BPF_CGROUP_GETSOCKOPT` attached at each level with `BPF_F_ALLOW_MULTI` flag:

```
A (root, parent)
 \
  B (child)
```

When the application calls `getsockopt` syscall from the cgroup B, the programs are executed from the bottom up: B, A. First program (B) sees the result of kernel's `getsockopt`. It can optionally adjust `optval`, `optlen` and reset `retval` to 0. After that control will be passed to the second (A) program which will see the same context as B including any potential modifications.

Same for `BPF_CGROUP_SETSOCKOPT`: if the program is attached to A and B, the trigger order is B, then A. If B does any changes to the input arguments (`level`, `optname`, `optval`, `optlen`), then the next program in the chain (A) will see those changes, *not* the original input `setsockopt` arguments. The potentially modified values will be then passed down to the kernel.

Large optval

When the `optval` is greater than the `PAGE_SIZE`, the BPF program can access only the first `PAGE_SIZE` of that data. So it has to options:

- Set `optlen` to zero, which indicates that the kernel should use the original buffer from the userspace. Any modifications done by the BPF program to the `optval` are ignored.
- Set `optlen` to the value less than `PAGE_SIZE`, which indicates that the kernel should use BPF's trimmed `optval`.

When the BPF program returns with the `optlen` greater than `PAGE_SIZE`, the userspace will receive `EFAULT` errno.

Example

Example

See `tools/testing/selftests/bpf/progs/sockopt_sk.c` for an example of BPF program that handles socket options.