CLOUDFLARE

Speedrun through SOCKMAP

Jakub Sitnicki
Systems Engineer
Cloudflare

SREcon23
Dublin, Ireland
October, 2023

$ whoami

Linux (kernel) Team @ Cloudflare

🌽 roll out fresh kernels

🐞 squash bugs

🕵️ troubleshoot stuff

✨ prototype features

DISCLAIMER: This is not my mug

$ whoami

Linux (kernel) Team @ Cloudflare

- 🌽 roll out fresh kernels
- 🐞 squash bugs
- 🕵️ troubleshoot stuff
- ✨ prototype features

SOCKMAP co-maintainer @ Linux upstream

- 🤏 small-time (= feature) maintainer
- 🪲 fix bugs
- 🧐 review patches
- 🤔 answer questions

DISCLAIMER: This is not my mug

About this talk

Will be helpful if you know a bit about:

- ❏ network programming (`socket`, `connect`, `sendmsg`, `recvmsg`)
- ❏ basics of eBPF (what are BPF maps, programs, hooks, `bpftool`)
- ❏ building blocks of containers (cgroups, namespaces)

Goals:

- ❏ know that SOCKMAP exists
- ❏ have idea how / when / what for use it
- ❏ feel ready to dive deeper

Agenda

1 What is SOCKMAP? What can SOCKMAP do?

2 Evolution of SOCKMAP

3 How to set up SOCKMAP?

4 How to get sockets into a SOCKMAP?

5 Supported configurations

6 Real life use cases

Section

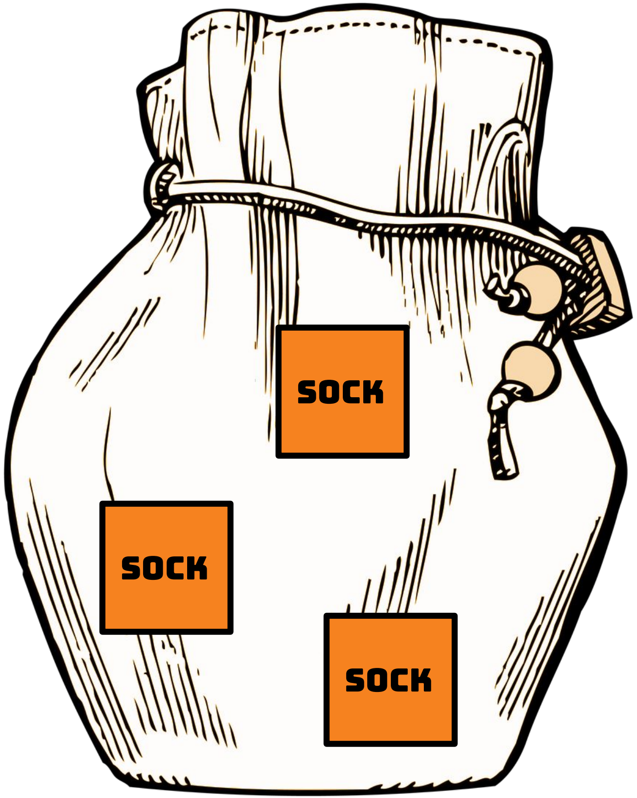
What is SOCKMAP?
&
What can SOCKMAP do?

What is SOCKMAP?

Two things

What is SOCKMAP?

Collection / container
for socket references
in Linux kernel



Rawpixel, CC0

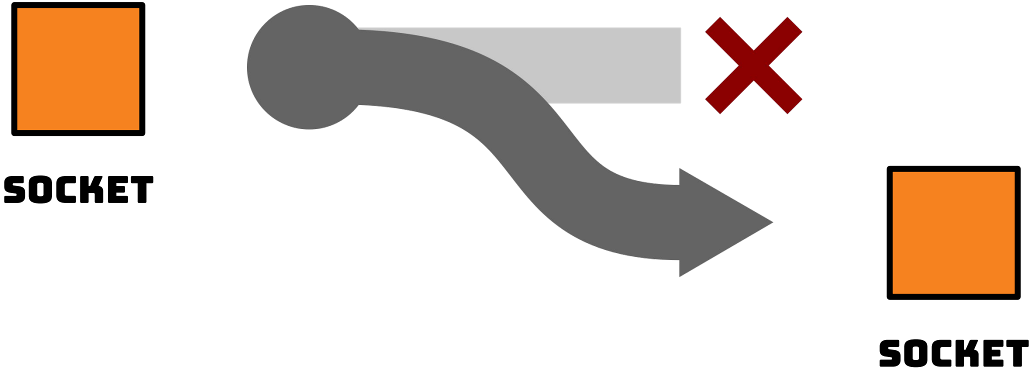
What is SOCKMAP?

SOCKMAP
API

1. **container for sockets**

- BPF map (K/V store)
- holds weak refs to sockets

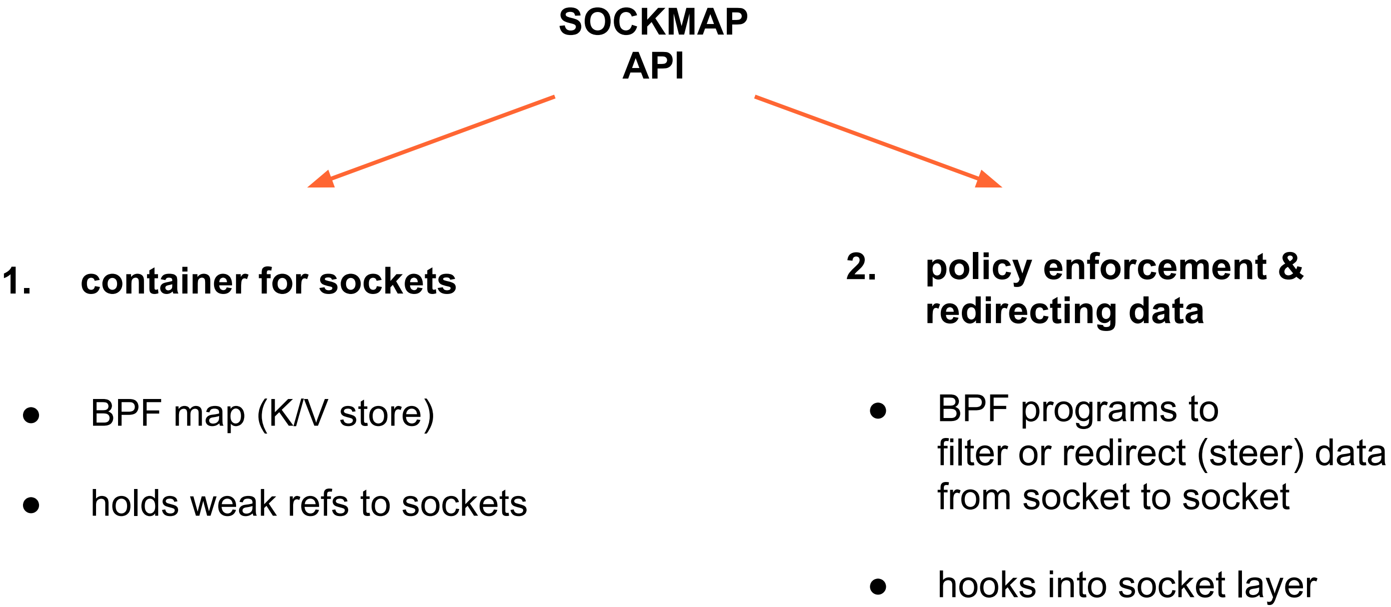
What is SOCKMAP?



Linux API for enforcing policy
and
redirecting data between sockets

Wikimedia Commons, CC BY 3.0 DEED

What is SOCKMAP?



SOCKMAP API

1. **container for sockets**

- BPF map (K/V store)
- holds weak refs to sockets

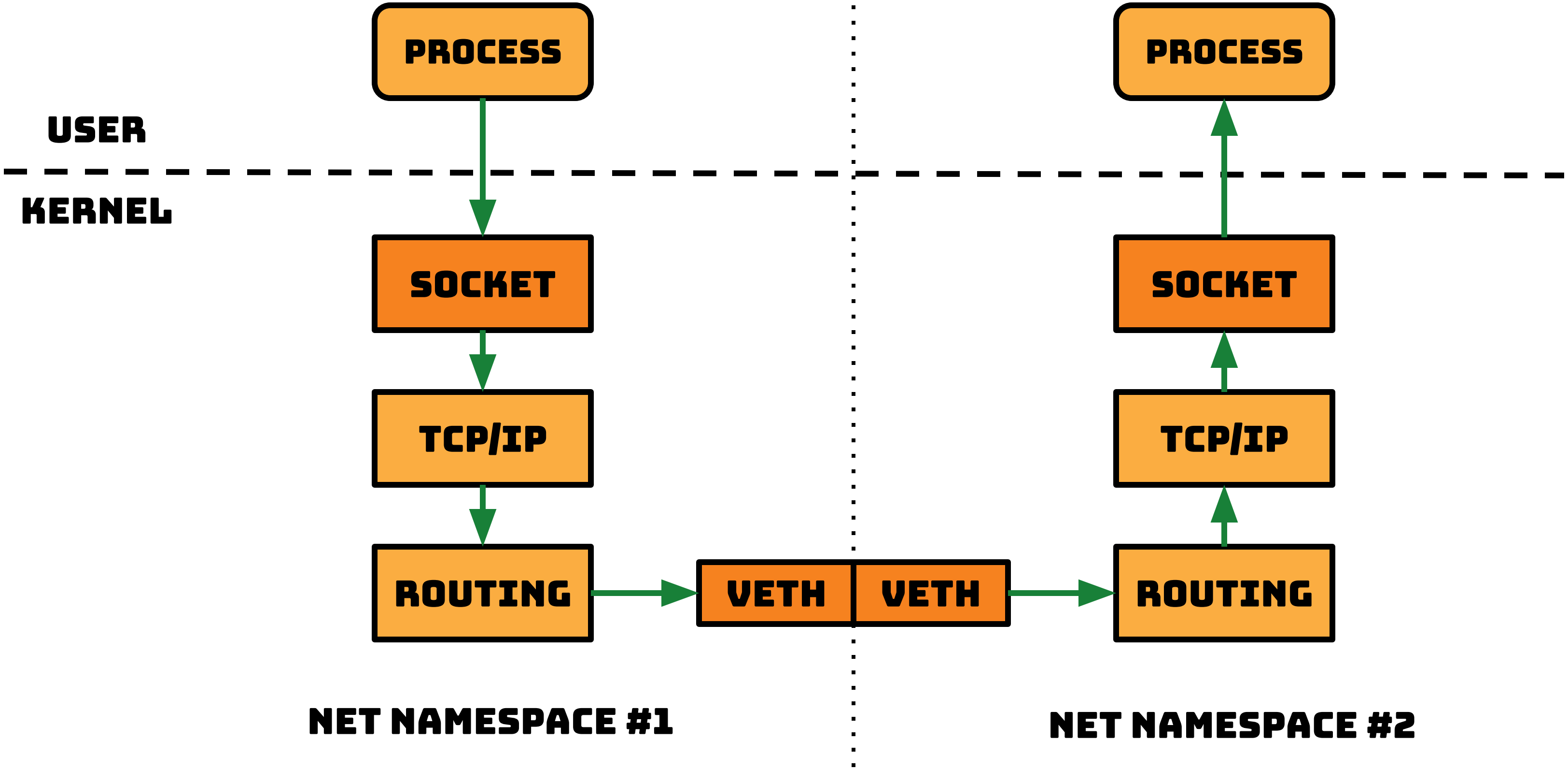
2. **policy enforcement & redirecting data**

- BPF programs to filter or redirect (steer) data from socket to socket
- hooks into socket layer

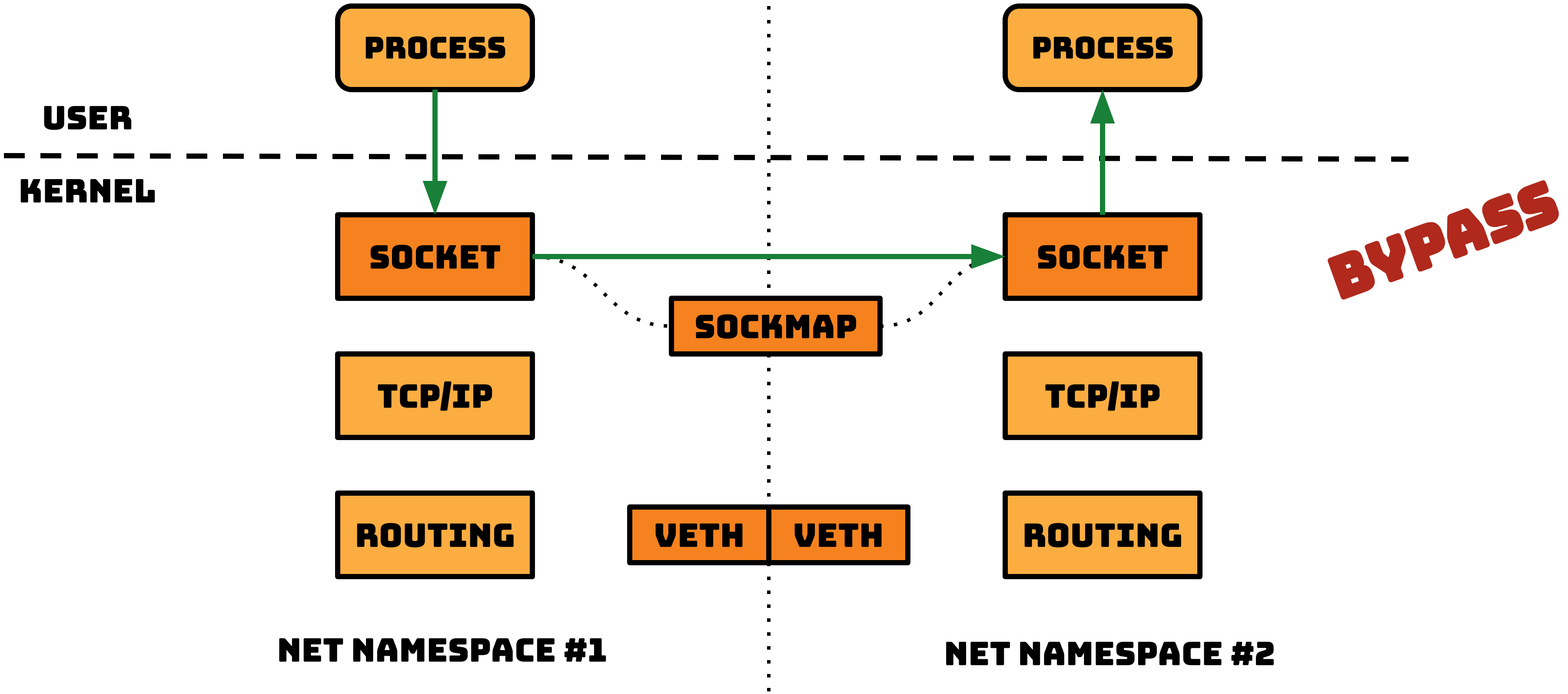
What can SOCKMAP do for you?

publicdomainpictures.net, CC0

What can SOCKMAP do for … container networking



What can SOCKMAP do for … container networking



Test it → Setup a pair of network namespaces

Create network namespaces

```
# ip netns add A
# ip netns add B
```

Link network namespaces with a veth pair

```
# ip -n A link add name veth0 type veth peer name veth0 netns B
```

Bring up the links inside network namespaces

```
# ip -n A link set dev lo up
# ip -n B link set dev lo up
# ip -n A link set dev veth0 up
# ip -n B link set dev veth0 up
```

Assign addresses to links inside network namespaces

```
# ip -n A addr add 10.0.0.1/24 dev veth0
# ip -n B addr add 10.0.0.2/24 dev veth0
```

Test it → Run TCP request-response benchmark

Run TCP server

```
# ip netns exec A \
  sockperf server -i 10.0.0.1 --tcp --daemonize
```

Run TCP client

```
# ip netns exec B \
  sockperf ping-pong -i 10.0.0.1 --tcp --time 30
…
sockperf: [Total Run] RunTime=30.000 sec; Warm up time=400 msec; SentMessages=2599753;
ReceivedMessages=2599752
…
sockperf: ====> avg-latency=5.748 (std-dev=2.010, mean-ad=0.322, median-ad=0.220,
siqr=0.239, cv=0.350, std-error=0.001, 99.0% ci=[5.745, 5.751])
sockperf: # dropped messages = 0; # duplicated messages = 0; # out-of-order messages = 0
sockperf: Summary: Latency is 5.748 usec
```

5.748 ± 2.010 usec

Test it → Configure SOCKMAP bypass

Load BPF programs and create BPF maps

```
# bpftool prog loadall redir_bypass.bpf.o /sys/fs/bpf pinmaps /sys/fs/bpf
```

Attach BPF program to BPF map

```
# bpftool prog attach \
  pinned /sys/fs/bpf/sk_msg_prog sk_msg_verdict \
  pinned /sys/fs/bpf/sock_map
```

Create a test cgroup

```
# mkdir /sys/fs/cgroup/unified/test.slice
```

Attach BPF program to cgroup

```
# bpftool cgroup attach \
  /sys/fs/cgroup/unified/test.slice \
  cgroup_sock_ops pinned /sys/fs/bpf/sockops_prog
```

Test it → Repeat the test with SOCKMAP bypass

```
Spawn client and server inside the test cgroup
# echo $$ > /sys/fs/cgroup/unified/test.slice/cgroup.procs

Run TCP server
# ip netns exec A \
  sockperf server -i 10.0.0.1 --tcp --daemonize

Run TCP client
# ip netns exec B \
   sockperf ping-pong -i 10.0.0.1 --tcp --time 30
sockperf: [Total Run] RunTime=30.000 sec; Warm up time=400 msec; SentMessages=3189584;
ReceivedMessages=3189583
…
sockperf: ====> avg-latency=4.686 (std-dev=2.862, mean-ad=0.250, median-ad=0.216,
siqr=0.173, cv=0.611, std-error=0.002, 99.0% ci=[4.682, 4.690])
sockperf: # dropped messages = 0; # duplicated messages = 0; # out-of-order messages = 0
sockperf: Summary: Latency is 4.686 usec
```

4.686 ± 2.862 usec

Test it → Compare without and with SOCKMAP bypass

before: 5.748 ± 2.010 usec

↓ - 18.5%

after: 4.686 ± 2.862 usec

Run the benchmark yourself:
https://github.com/jsitnicki/srecon-2023-sockmap/blob/main/examples/redir-bypass/test_redir_bypass.sh

Section

Evolution of SOCKMAP

How did we get here?

Kernel Connection Multiplexor

Added infrastructure later reused by SOCKMAP - Psock and stream parser program.

2016

v4.6

```
+------------+   +------------+   +------------+   +------------+
| KCM socket |   | KCM socket |   | KCM socket |   | KCM socket |
+------------+   +------------+   +------------+   +------------+
      |                 |               |                |
      +-----------+     |               |     +----------+
                  |     |               |     |
               +----------------------------------+
               |           Multiplexor            |
               +----------------------------------+
                 |   |           |           |  |
       +---------+   |           |           |  ------------+
       |             |           |           |              |
+----------+  +----------+  +----------+  +----------+ +----------+
|  Psock   |  |  Psock   |  |  Psock   |  |  Psock   | |  Psock   |
+----------+  +----------+  +----------+  +----------+ +----------+
      |              |           |            |             |
+----------+  +----------+  +----------+  +----------+ +----------+
| TCP sock |  | TCP sock |  | TCP sock |  | TCP sock | | TCP sock |
+----------+  +----------+  +----------+  +----------+ +----------+
```

[PATCH v2 net-next 00/13] kcm: Kernel Connection Multiplexor (KCM)

How did we get here?

Kernel Connection Multiplexor

Added infrastructure later reused by SOCKMAP - Psock and stream parser program.

2016

v4.6

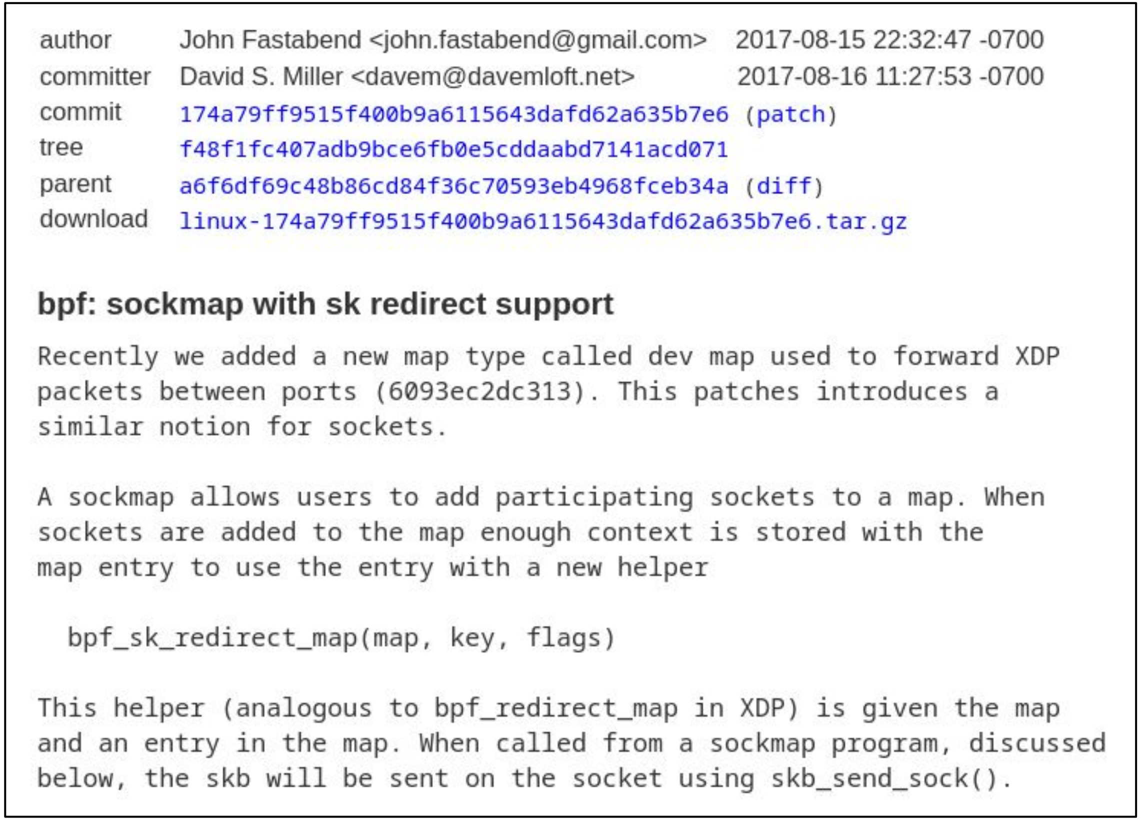
2017

v4.14

SOCKMAP initial version

Filtering and redirect on ingress to socket layer.

Counterpart of XDP DEVMAP.



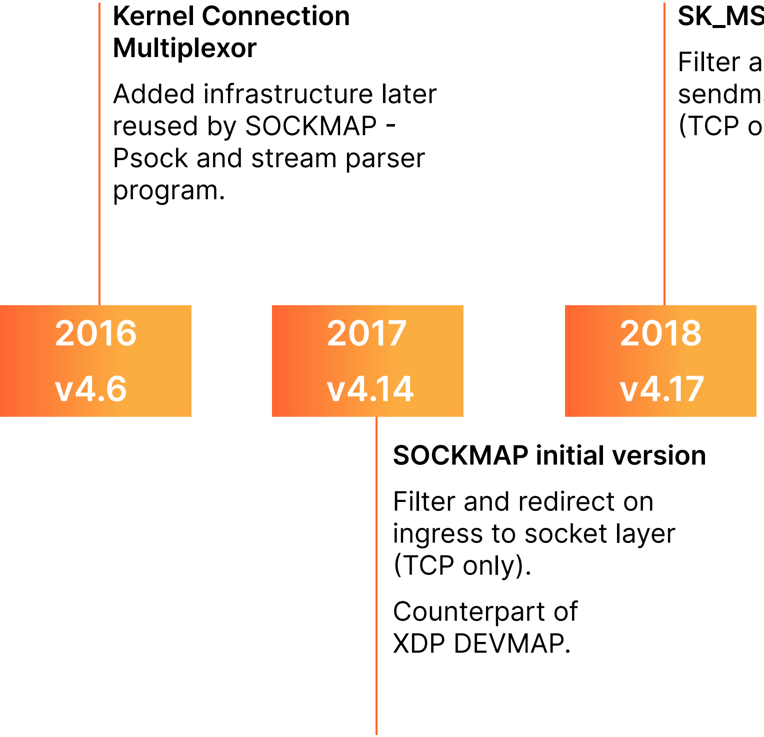
author	John Fastabend <john.fastabend@gmail.com>	2017-08-15 22:32:47 -0700
committer	David S. Miller <davem@davemloft.net>	2017-08-16 11:27:53 -0700
commit	174a79ff9515f400b9a6115643dafd62a635b7e6 (patch)	
tree	f48f1fc407adb9bce6fb0e5cddaabd7141acd071	
parent	a6f6df69c48b86cd84f36c70593eb4968fceb34a (diff)	
download	linux-174a79ff9515f400b9a6115643dafd62a635b7e6.tar.gz	

bpf: sockmap with sk redirect support

```
Recently we added a new map type called dev map used to forward XDP
packets between ports (6093ec2dc313). This patches introduces a
similar notion for sockets.

A sockmap allows users to add participating sockets to a map. When
sockets are added to the map enough context is stored with the
map entry to use the entry with a new helper

  bpf_sk_redirect_map(map, key, flags)

This helper (analogous to bpf_redirect_map in XDP) is given the map
and an entry in the map. When called from a sockmap program, discussed
below, the skb will be sent on the socket using skb_send_sock().
```

commit 174a79ff9515 ("bpf: sockmap with sk redirect support")

How did we get here?



Kernel Connection Multiplexor

Added infrastructure later reused by SOCKMAP - Psock and stream parser program.

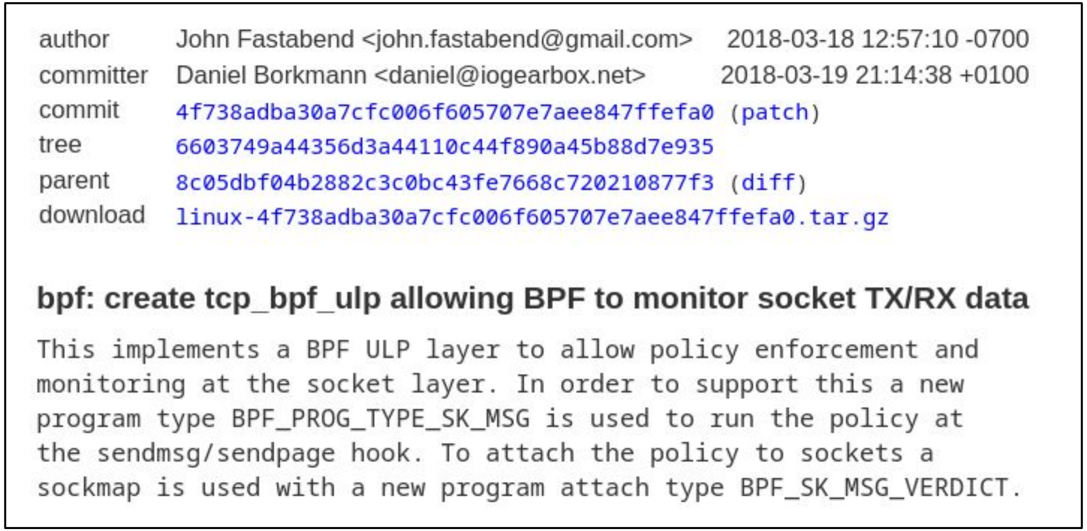
SOCKMAP initial version

Filter and redirect on ingress to socket layer (TCP only).

Counterpart of XDP DEVMAP.

SK_MSG program

Filter and redirect at sendmsg() time (TCP only).



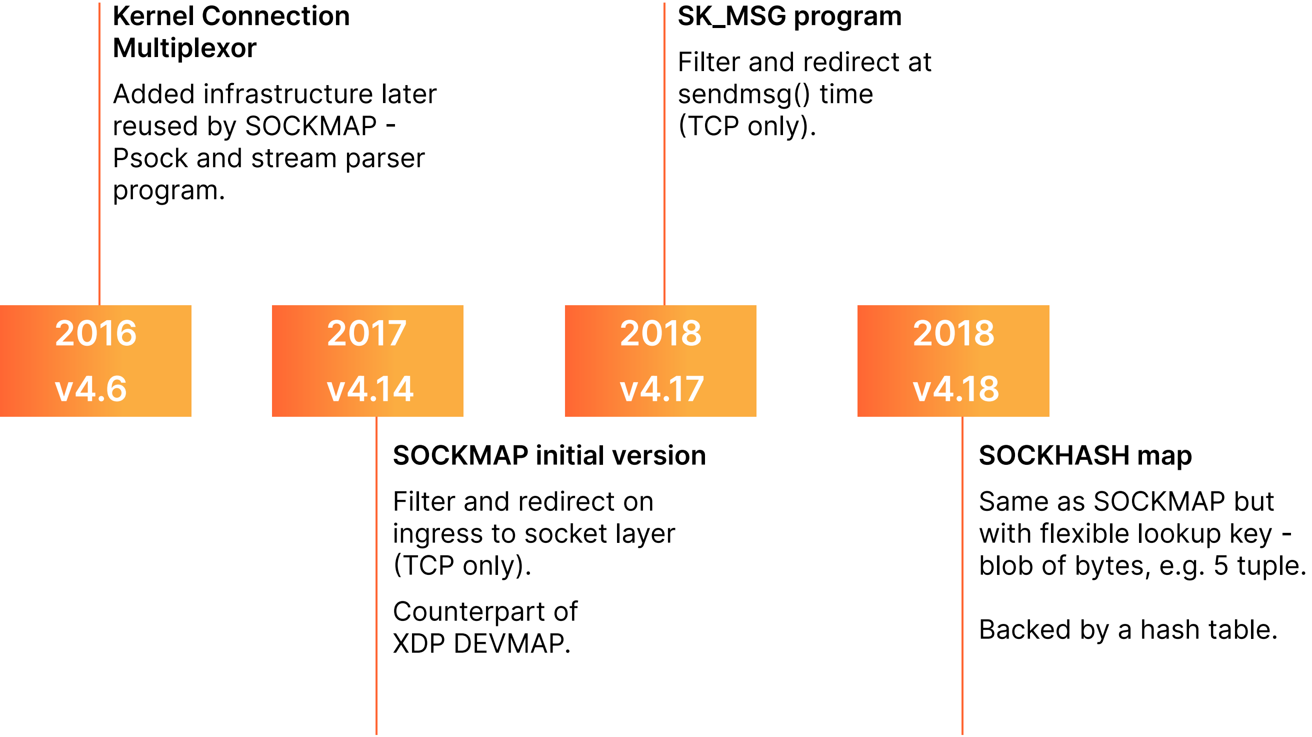
author	John Fastabend <john.fastabend@gmail.com>	2018-03-18 12:57:10 -0700
committer	Daniel Borkmann <daniel@iogearbox.net>	2018-03-19 21:14:38 +0100
commit	4f738adba30a7cfc006f605707e7aee847ffefa0 (patch)	
tree	6603749a44356d3a44110c44f890a45b88d7e935	
parent	8c05dbf04b2882c3c0bc43fe7668c720210877f3 (diff)	
download	linux-4f738adba30a7cfc006f605707e7aee847ffefa0.tar.gz	

bpf: create tcp_bpf_ulp allowing BPF to monitor socket TX/RX data

```
This implements a BPF ULP layer to allow policy enforcement and
monitoring at the socket layer. In order to support this a new
program type BPF_PROG_TYPE_SK_MSG is used to run the policy at
the sendmsg/sendpage hook. To attach the policy to sockets a
sockmap is used with a new program attach type BPF_SK_MSG_VERDICT.
```

commit 4f738adba30a ("bpf: create tcp_bpf_ulp allowing BPF to monitor socket TX/RX data")

How did we get here?



2016
v4.6

Kernel Connection Multiplexor

Added infrastructure later reused by SOCKMAP - Psock and stream parser program.

2017
v4.14

SOCKMAP initial version

Filter and redirect on ingress to socket layer (TCP only).

Counterpart of XDP DEVMAP.

2018
v4.17

SK_MSG program

Filter and redirect at sendmsg() time (TCP only).

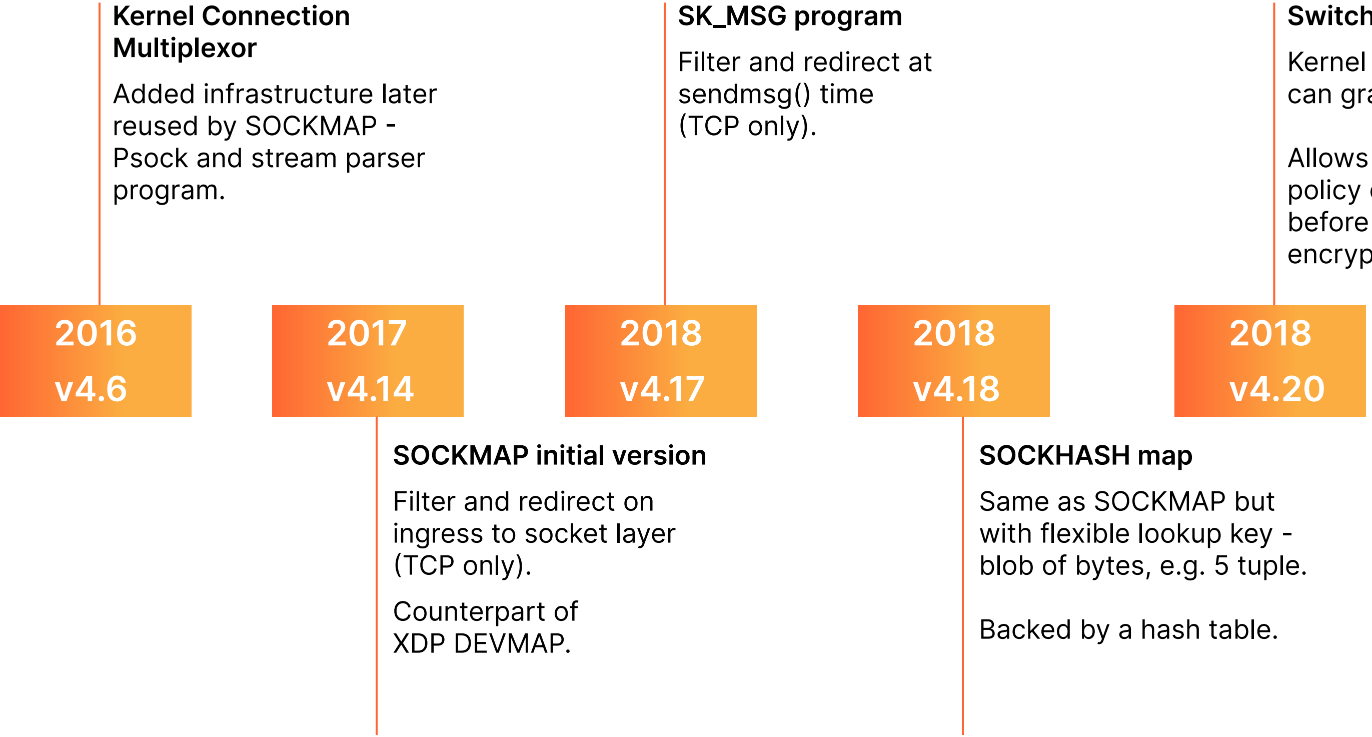
2018
v4.18

SOCKHASH map

Same as SOCKMAP but with flexible lookup key - blob of bytes, e.g. 5 tuple.

Backed by a hash table.

How did we get here?



2016

v4.6

Kernel Connection Multiplexor

Added infrastructure later reused by SOCKMAP - Psock and stream parser program.

2017

v4.14

SOCKMAP initial version

Filter and redirect on ingress to socket layer (TCP only).

Counterpart of XDP DEVMAP.

2018

v4.17

SK_MSG program

Filter and redirect at sendmsg() time (TCP only).

2018

v4.18

SOCKHASH map

Same as SOCKMAP but with flexible lookup key - blob of bytes, e.g. 5 tuple.

Backed by a hash table.

2018

v4.20

Switch to sk_msg API

Kernel TLS and SK_MSG can gracefully coexist.

Allows introspection / policy enforcement before in-kernel encryption.

commit 4f738adba30a ("bpf: create tcp_bpf_ulp allowing BPF to monitor socket TX/RX data")

How did we get here?

Store TCP or UDP sockets

SOCKMAP becomes a generic BPF map for sockets.

It can hold both connected and listening TCP sockets, and any bound UDP socket.

2020

v5.7

How did we get here?

Store TCP or UDP sockets

SOCKMAP becomes a generic BPF map for sockets.

It can hold both connected and listening TCP sockets, and any bound UDP socket.

2020

v5.7

2020

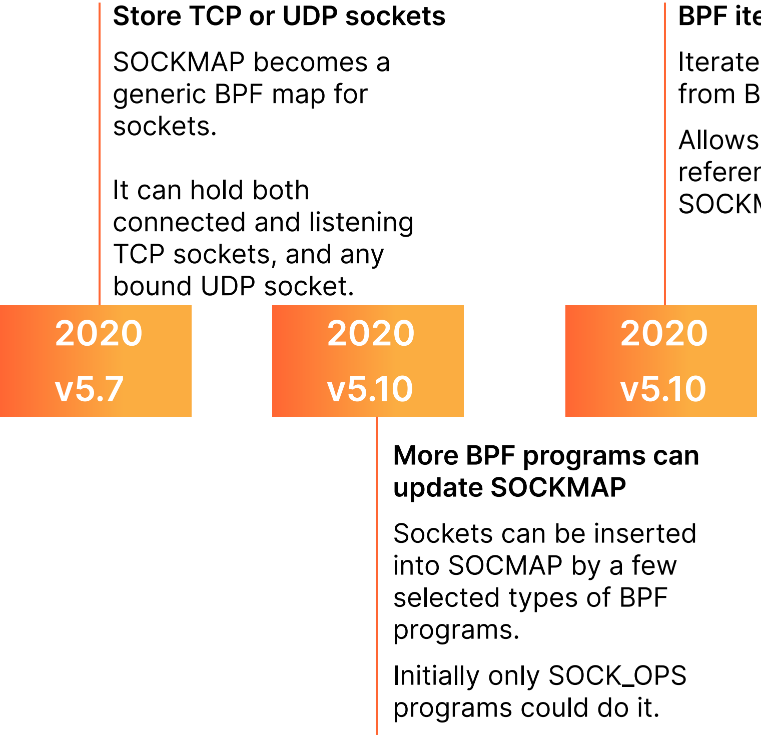
v5.10

More BPF programs can update SOCKMAP

Sockets can be inserted into SOCMAP by a few selected types of BPF programs.

Initially only SOCK_OPS programs could do it.

How did we get here?



2020

v5.7

Store TCP or UDP sockets

SOCKMAP becomes a generic BPF map for sockets.

It can hold both connected and listening TCP sockets, and any bound UDP socket.

2020

v5.10

More BPF programs can update SOCKMAP

Sockets can be inserted into SOCMAP by a few selected types of BPF programs.

Initially only SOCK_OPS programs could do it.

2020

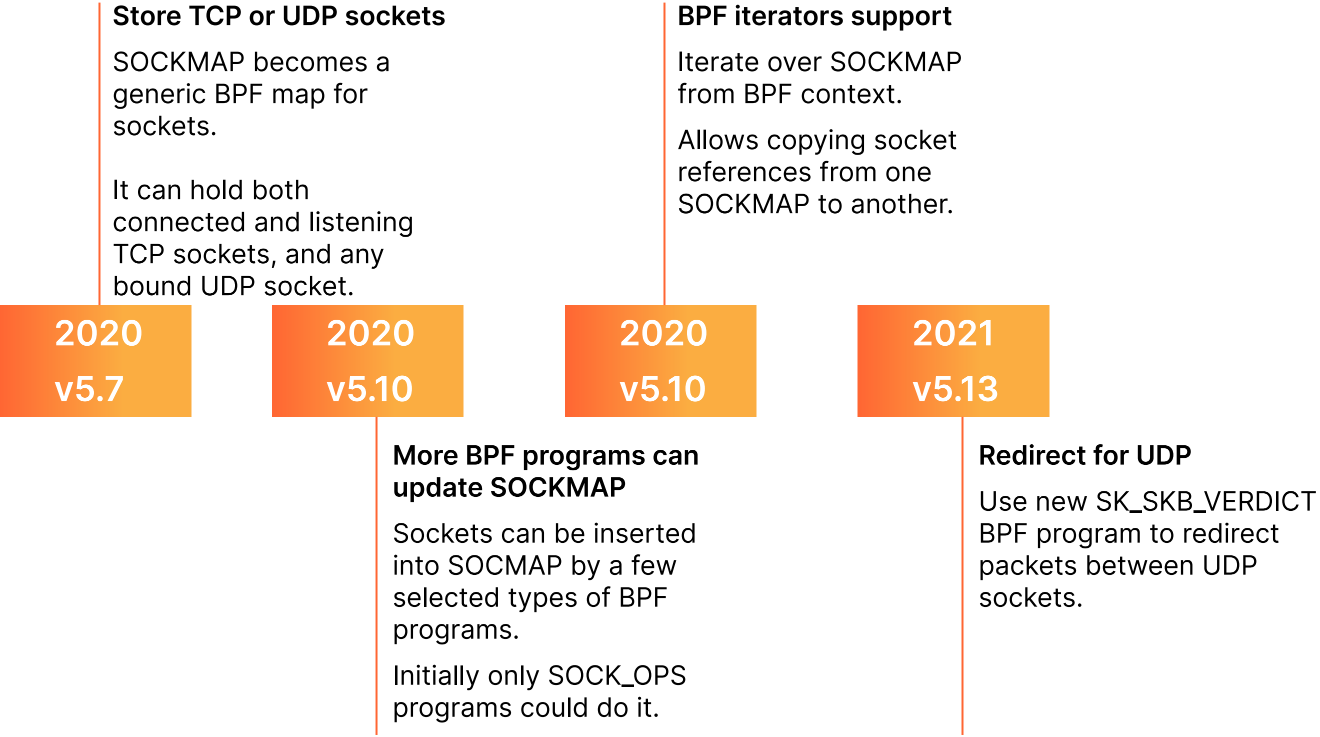
v5.10

BPF iterators support

Iterate over SOCKMAP from BPF context.

Allows copying socket references from one SOCKMAP to another.

How did we get here?



2020

v5.7

Store TCP or UDP sockets

SOCKMAP becomes a generic BPF map for sockets.

It can hold both connected and listening TCP sockets, and any bound UDP socket.

2020

v5.10

More BPF programs can update SOCKMAP

Sockets can be inserted into SOCMAP by a few selected types of BPF programs.

Initially only SOCK_OPS programs could do it.

2020

v5.10

BPF iterators support

Iterate over SOCKMAP from BPF context.

Allows copying socket references from one SOCKMAP to another.

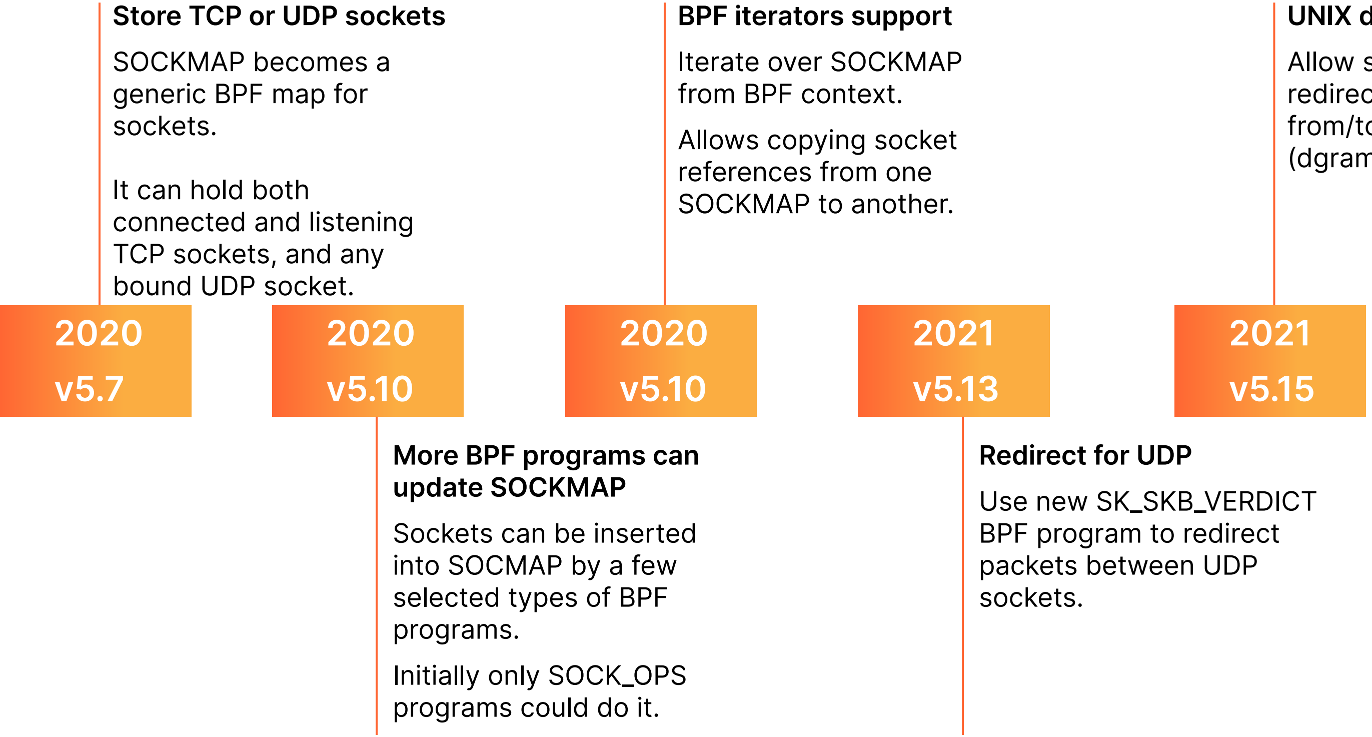
2021

v5.13

Redirect for UDP

Use new SK_SKB_VERDICT BPF program to redirect packets between UDP sockets.

How did we get here?



2020

v5.7

Store TCP or UDP sockets

SOCKMAP becomes a generic BPF map for sockets.

It can hold both connected and listening TCP sockets, and any bound UDP socket.

2020

v5.10

More BPF programs can update SOCKMAP

Sockets can be inserted into SOCMAP by a few selected types of BPF programs.

Initially only SOCK_OPS programs could do it.

2020

v5.10

BPF iterators support

Iterate over SOCKMAP from BPF context.

Allows copying socket references from one SOCKMAP to another.

2021

v5.13

Redirect for UDP

Use new SK_SKB_VERDICT BPF program to redirect packets between UDP sockets.

2021

v5.15

UNIX domain sockets

Allow storing and redirecting packets from/to UNIX sockets (dgram and stream).

How did we get here?

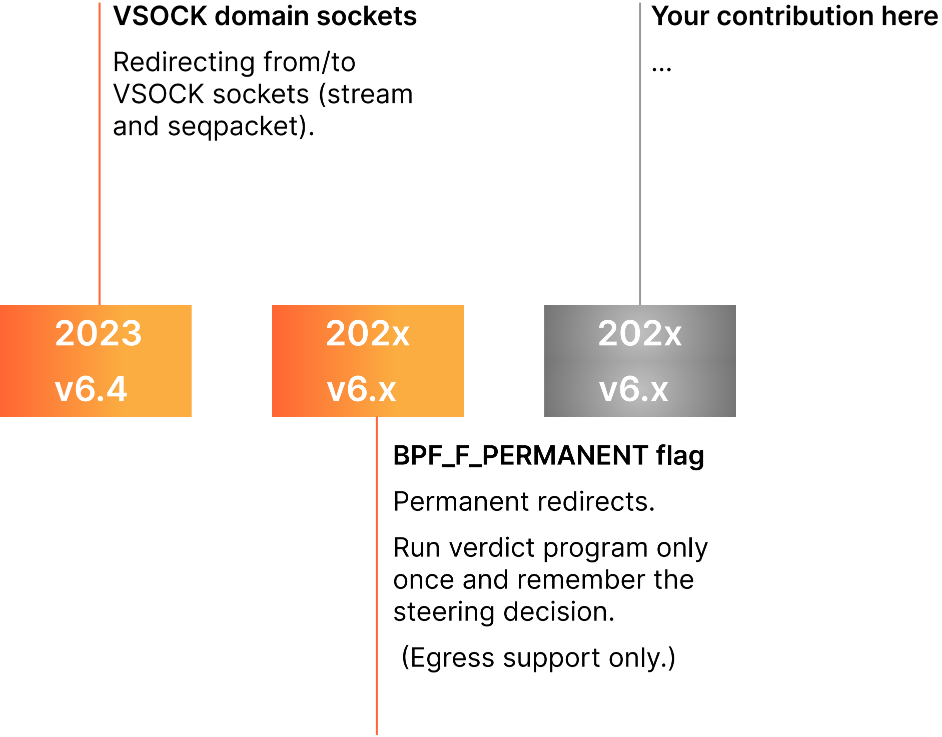
VSOCK domain sockets

Redirecting from/to VSOCK sockets (stream and seqpacket).

2023

v6.4

How did we get here?



VSOCK domain sockets

Redirecting from/to VSOCK sockets (stream and seqpacket).

Your contribution here

…

2023
v6.4

202x
v6.x

202x
v6.x

BPF_F_PERMANENT flag

Permanent redirects.

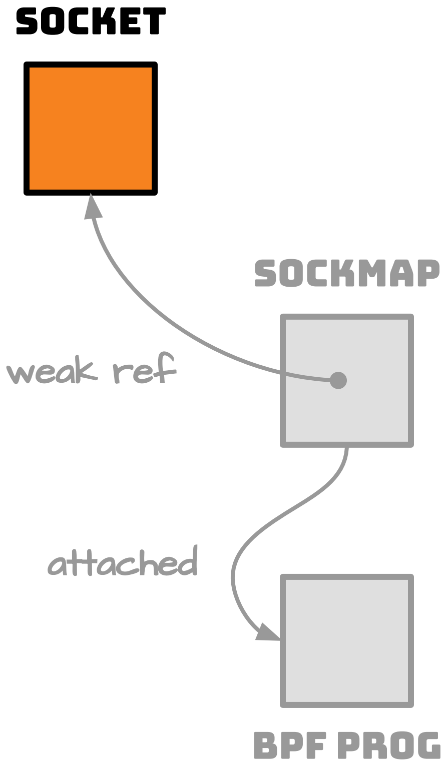
Run verdict program only once and remember the steering decision.

(Egress support only.)

Section

How to set up SOCKMAP?

① Open a connected (established) socket



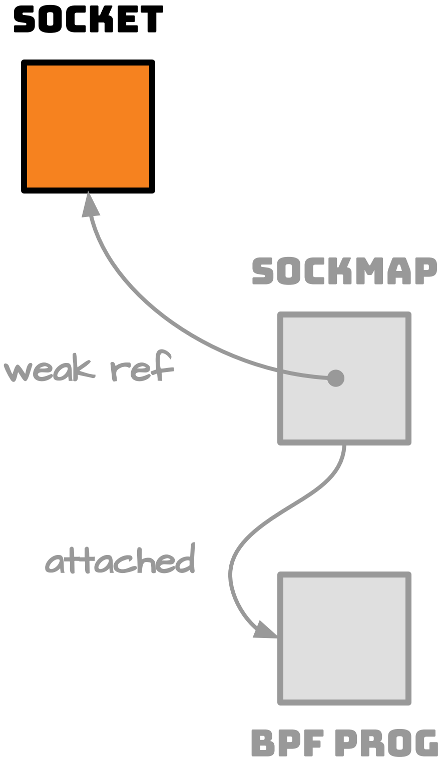
active open

```
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 8
connect(8, {sa_family=AF_INET,
            sin_port=htons(41895),
            sin_addr=inet_addr("127.0.0.1")}, 16) = 0
```

passive open

```
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 7
bind(7, {sa_family=AF_INET,
         sin_port=htons(41895),
         sin_addr=inet_addr("127.0.0.1")}, 16) = 0
listen(7, 4096)                               = 0
accept(7, NULL, NULL)                         = 9
```

What sockets can you use?



connected (established) socket:

❏ TCP

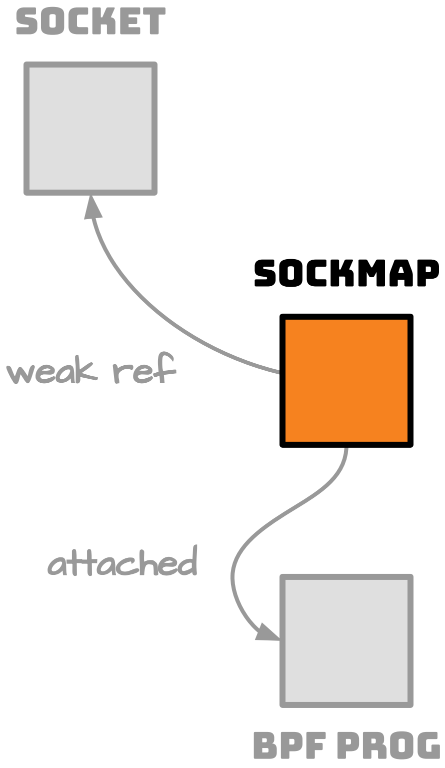
❏ UDP

❏ UNIX (STREAM, DGRAM)

❏ VSOCK (STREAM, SEQPACKET)

`man 7 {tcp,udp,unix,vsock}`

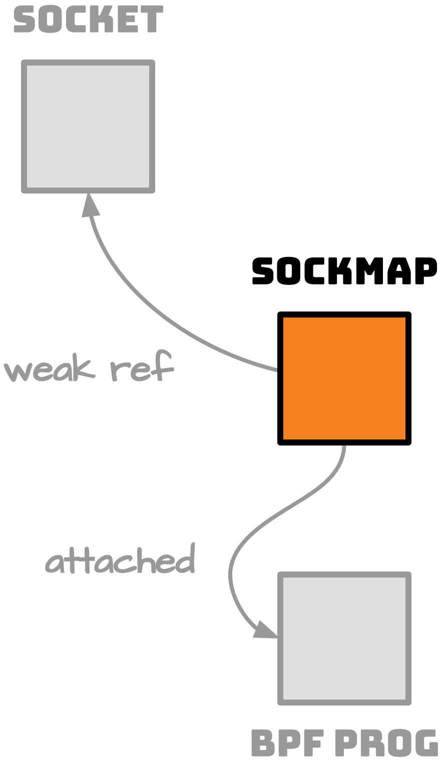
② Create a BPF map - SOCKMAP or SOCKHASH



from a program using the `bpf()` syscall
or a library wrapper (`libbpf`)

```
bpf(BPF_MAP_CREATE, {map_type=BPF_MAP_TYPE_SOCKMAP,
                     key_size=4,
                     value_size=8,
                     max_entries=1,
                     map_flags=0,
                     …}, 72) = 5
```

What BPF maps can you use?



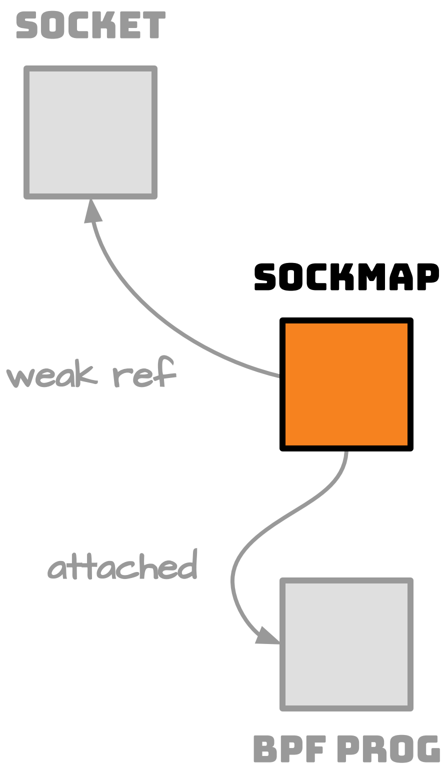
Map types:

- ❏ **BPF_MAP_TYPE_SOCKMAP**
  - ❏ **key size always 4B**
- ❏ **BPF_MAP_TYPE_SOCKHASH**
  - ❏ **arbitrary key size**

Not to be confused with
BPF_MAP_TYPE_REUSEPORT_SOCKARRAY

https://docs.kernel.org/bpf/map_sockmap.html

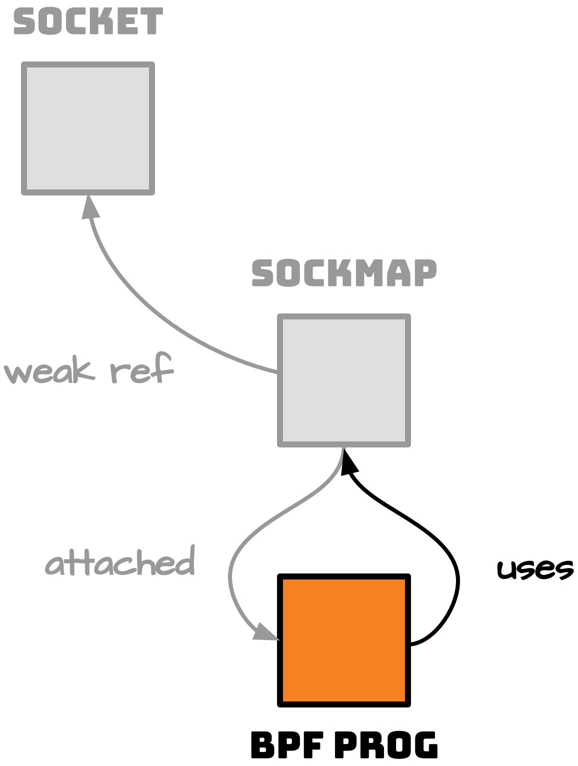
② Create a BPF map - From command line



using `bpftool map create`

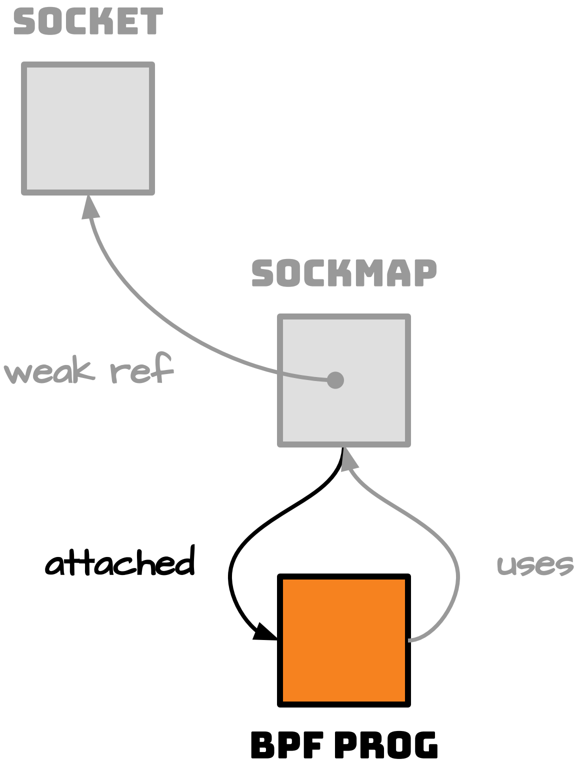
```
# bpftool map create                                          \
    /sys/fs/bpf/sockmap `# path on bpffs`                     \
    type sockmap        `# sockmap or sockhash`               \
    key 4               `# always 4 bytes for sockmap`        \
    value 8             `# use 8 bytes for dump to work`      \
    entries 1                                                 \
    name sockmap
#
# bpftool map show pinned /sys/fs/bpf/sockmap
3: sockmap  name sockmap  flags 0x0
        key 4B  value 8B  max_entries 1  memlock 328B
```

③ Load a BPF program - SK_MSG or SK_SKB type



```
bpf(BPF_PROG_LOAD, {prog_type=BPF_PROG_TYPE_SK_MSG,
                    insn_cnt=6,
                    insns=0xcf2a70,
                    expected_attach_type=BPF_SK_MSG_VERDICT,
                    …}, 128) = 6
```

What BPF programs can you use?

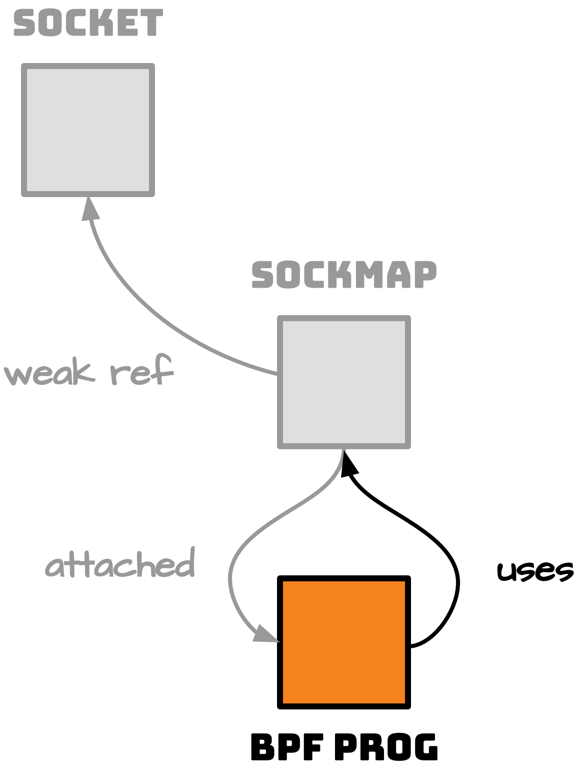


Program types:

- ❏ BPF_PROG_TYPE_SK_MSG
- ❏ BPF_PROG_TYPE_SK_SKB

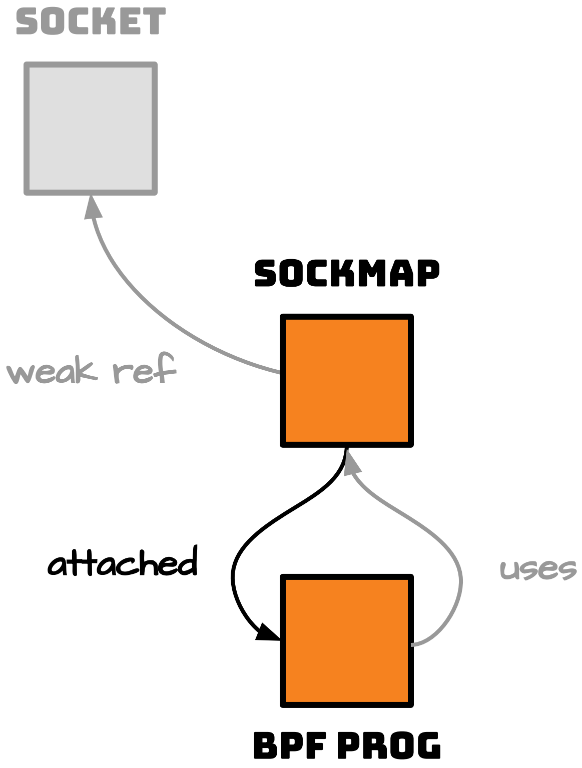
https://docs.kernel.org/bpf/map_sockmap.html

③ Load a BPF program - SK_MSG or SK_SKB type



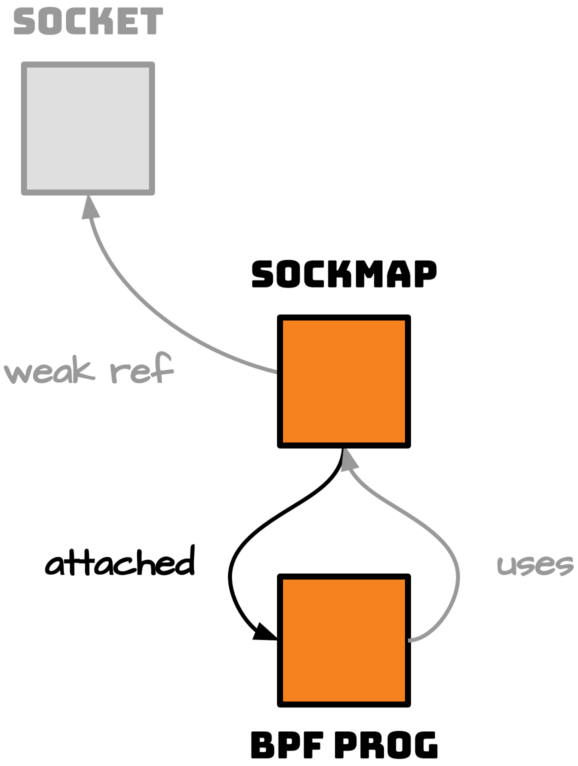
```
# bpftool prog dump xlated id 42
int prog_msg_redir_ingress(struct sk_msg_md * msg):
   0: (18) r2 = map[id:17]
   …
   5: (95) exit
# bpftool map show id 17
17: sockmap  name output  flags 0x0
        key 4B  value 8B  max_entries 1  memlock 328B
        pids sockmap-redir-m(331)
```

④ Attach BPF program to SOCKMAP



```
bpf(BPF_PROG_ATTACH, {target_fd=5,
                      attach_bpf_fd=6,
                      attach_type=BPF_SK_MSG_VERDICT,
                      attach_flags=0,
                      replace_bpf_fd=0}, 20) = 0
```

What BPF programs can you use?



Attach types (hooks):

❏ BPF_SK_MSG_VERDICT

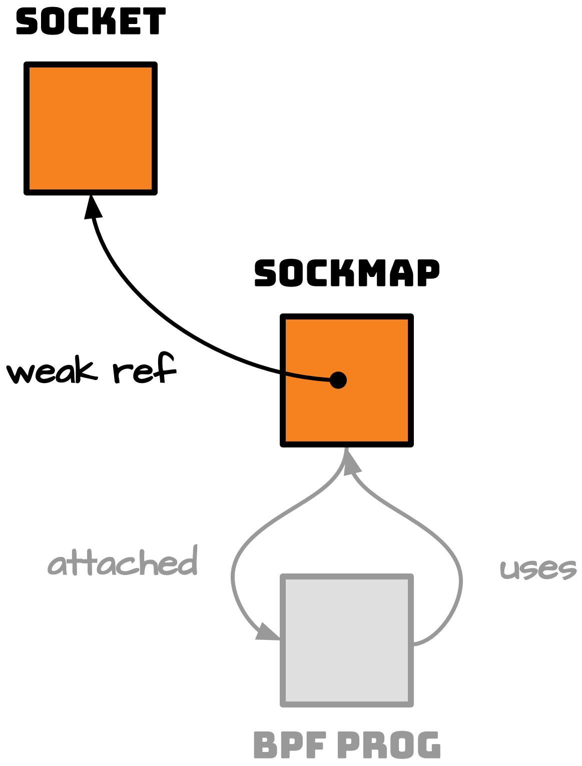
❏ BPF_SK_SKB_VERDICT

❏ BPF_SK_SKB_STREAM_PARSER

❏ BPF_SK_SKB_STREAM_VERDICT

https://docs.kernel.org/bpf/map_sockmap.html

⑤ Insert socket into SOCKMAP



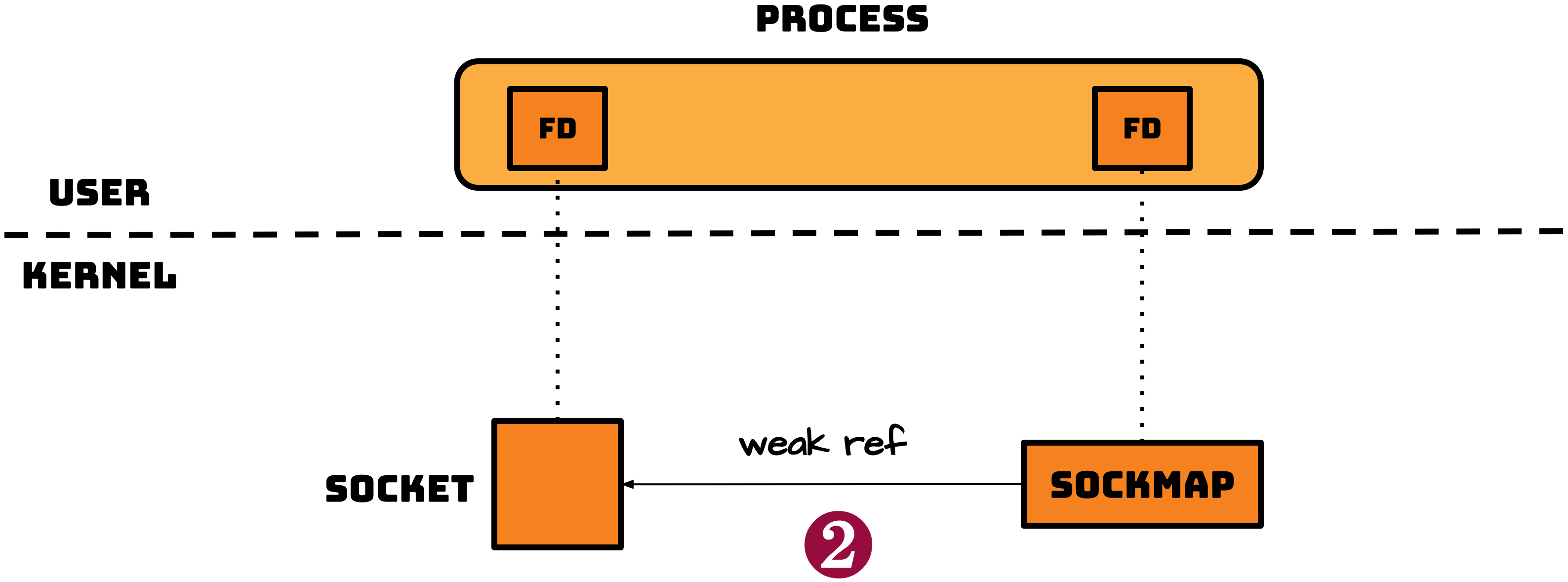
```
bpf(BPF_MAP_UPDATE_ELEM, {map_fd=5,
                          key=0x7ffeb3803870,
                          value=0x7ffeb3803868,
                          flags=BPF_NOEXIST}, 32) = 0
```

⚠ must be done after attaching the program

Section

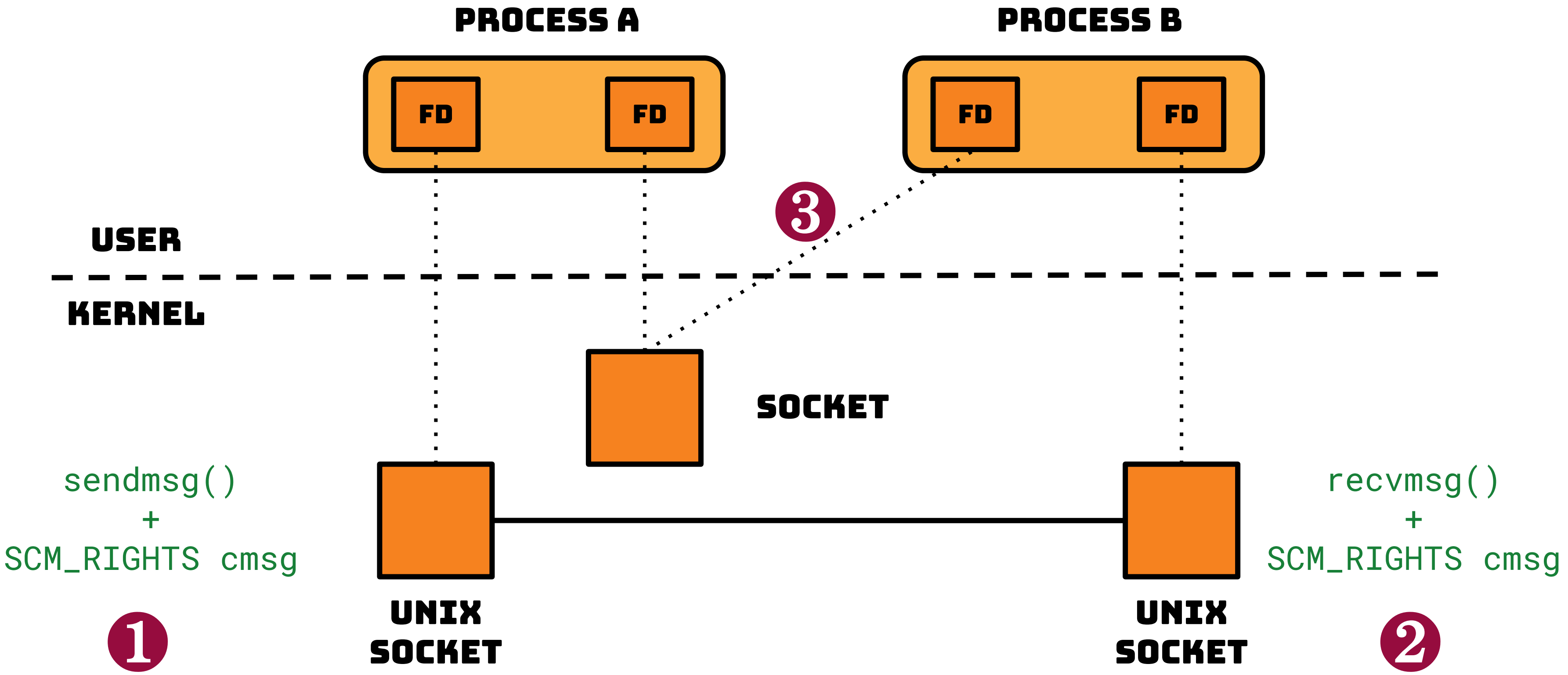
How to get sockets into a SOCKMAP?

Easy case



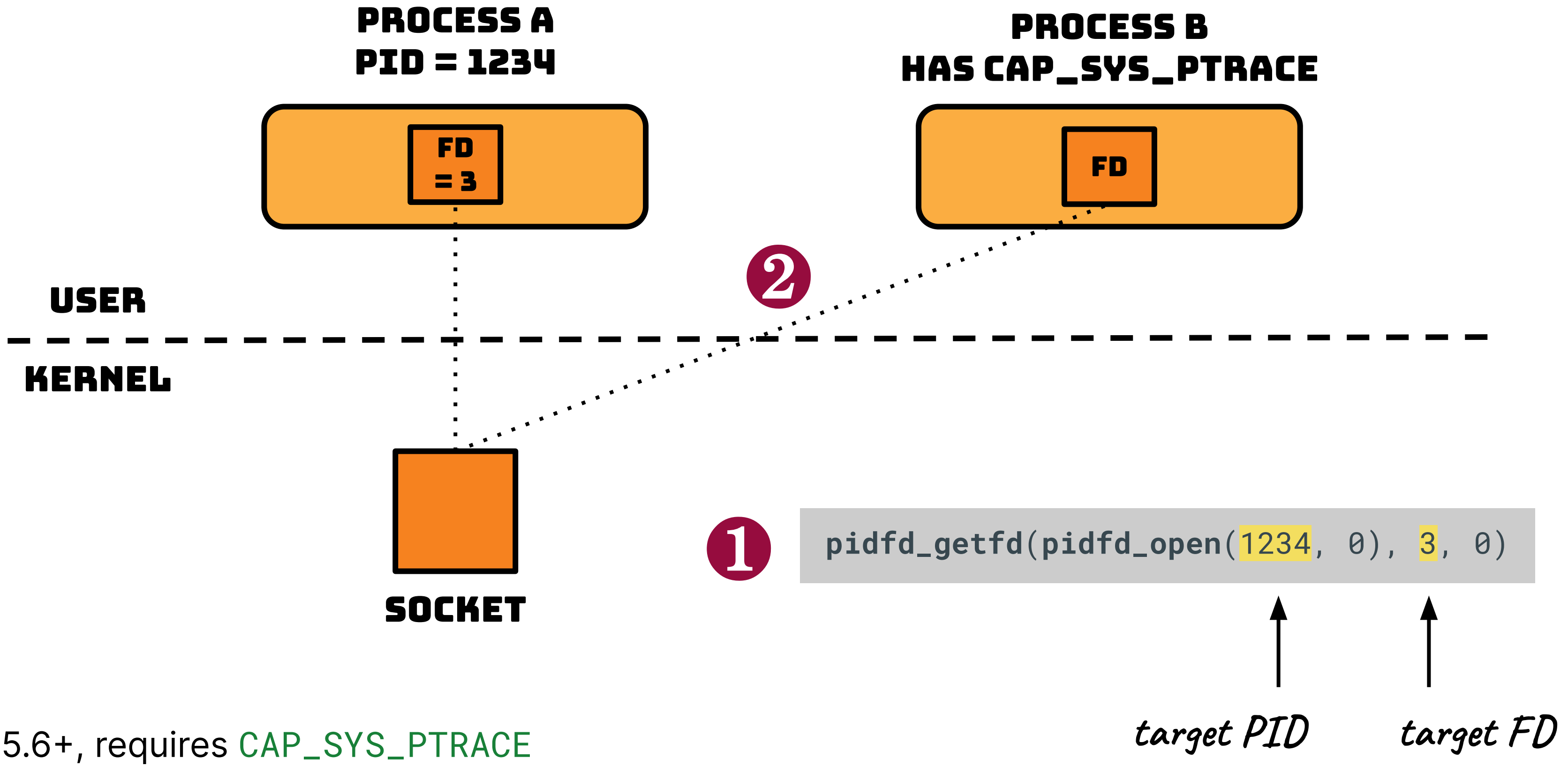
❶ `bpf(BPF_MAP_UPDATE_ELEM, …)`

Socket FD handover with SCM_RIGHTS



https://manpages.debian.org/unstable/manpages/unix.7.en.html#SCM_RIGHTS
https://blog.cloudflare.com/know-your-scm_rights/

“Steal” a socket FD



PROCESS A
PID = 1234

FD
= 3

PROCESS B
HAS CAP_SYS_PTRACE

FD

USER

KERNEL

SOCKET

❷

❶

```
pidfd_getfd(pidfd_open(1234, 0), 3, 0)
```

target PID

target FD

Linux 5.6+, requires `CAP_SYS_PTRACE`

https://manpages.debian.org/unstable/manpages-dev/pidfd_getfd.2.en.html

Use BPF sock_ops program attached to cgroup (TCP only)

```
SEC("sockops")
int sockops_prog(struct bpf_sock_ops *ctx)
{
     if (!ctx->sk)
          return SK_PASS;

     switch (ctx->op) {
     case BPF_SOCK_OPS_ACTIVE_ESTABLISHED_CB:
          bpf_sock_map_update(ctx, &sock_map, &(__u32){ 0 }, BPF_ANY);
          break;
     case BPF_SOCK_OPS_PASSIVE_ESTABLISHED_CB:
          bpf_sock_map_update(ctx, &sock_map, &(__u32){ 1 }, BPF_ANY);
          break;
     }

     return SK_PASS;
}
```

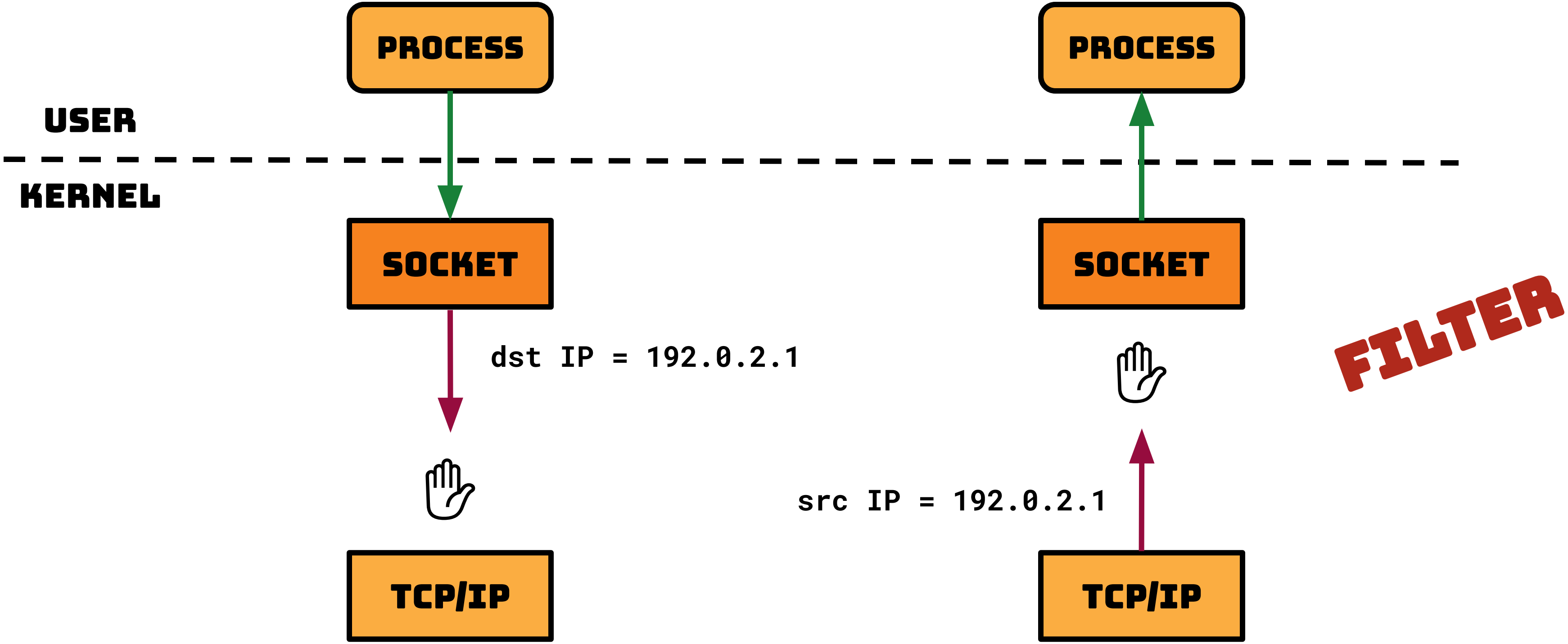
Section

Supported configurations

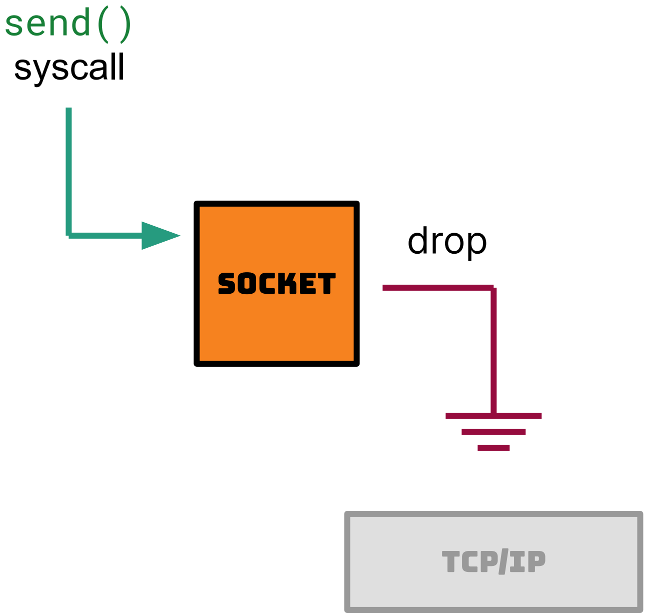
POLICY

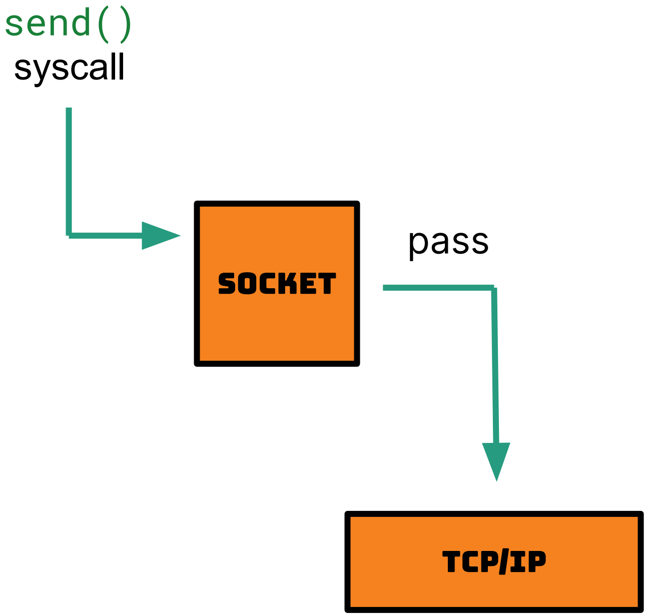
Rawpixel, CC0

Policy use case → API endpoints

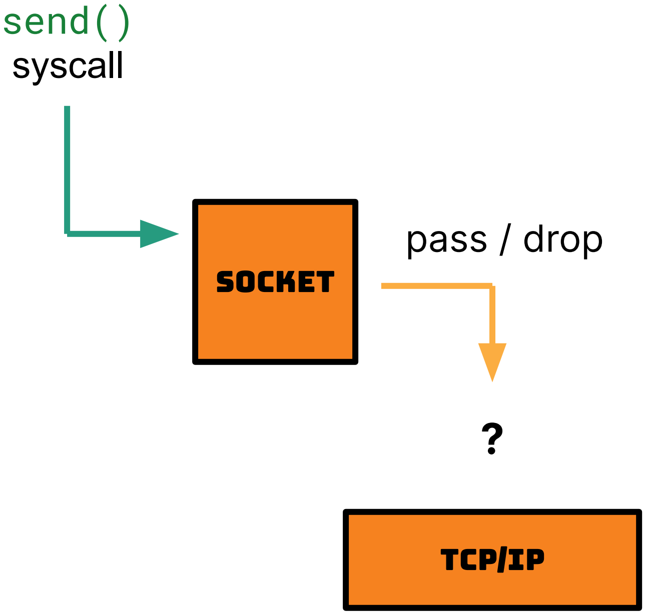


Policy on egress - pass or drop





Policy on egress → How?

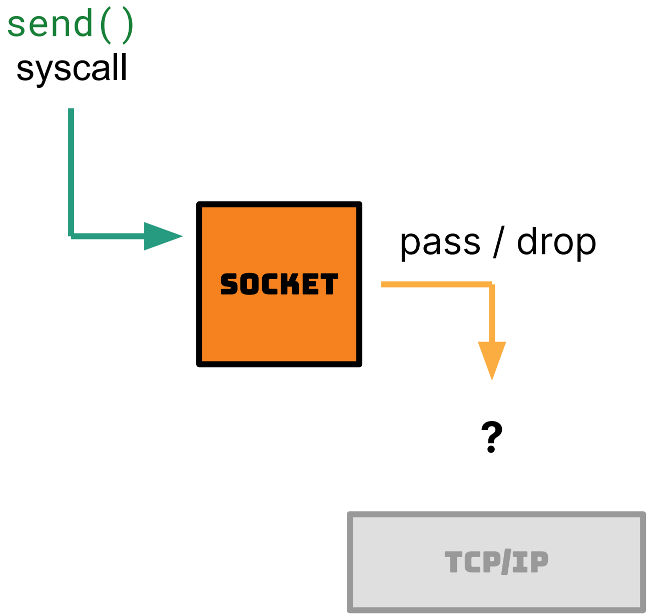


BPF_PROG_TYPE_**SK_MSG** prog

→ attached BPF_**SK_MSG_VERDICT** hook

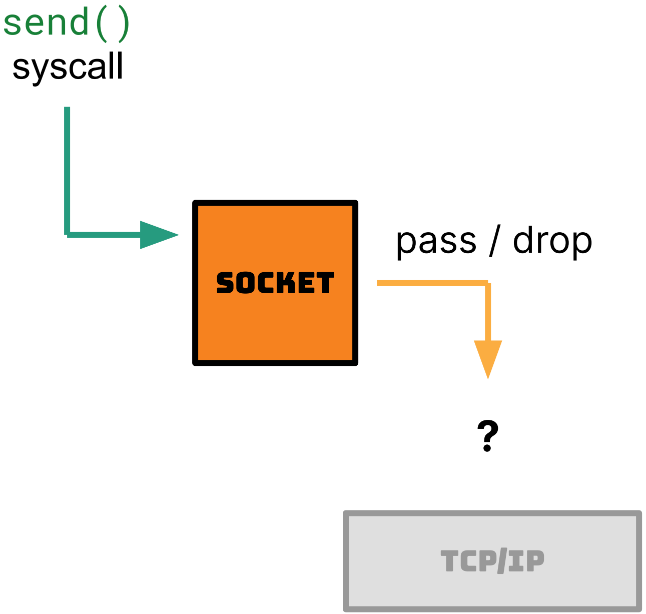
→ returns SK_**PASS** or SK_**DROP** verdict

Policy on egress → Program input



```
struct sk_msg_md {
        __bpf_md_ptr(void *, data);
        __bpf_md_ptr(void *, data_end);

        __u32 family;
        __u32 remote_ip4;
        __u32 local_ip4;
        __u32 remote_ip6[4];
        __u32 local_ip6[4];
        __u32 remote_port;
        __u32 local_port;
        __u32 size;

        __bpf_md_ptr(struct bpf_sock *, sk);
};
```

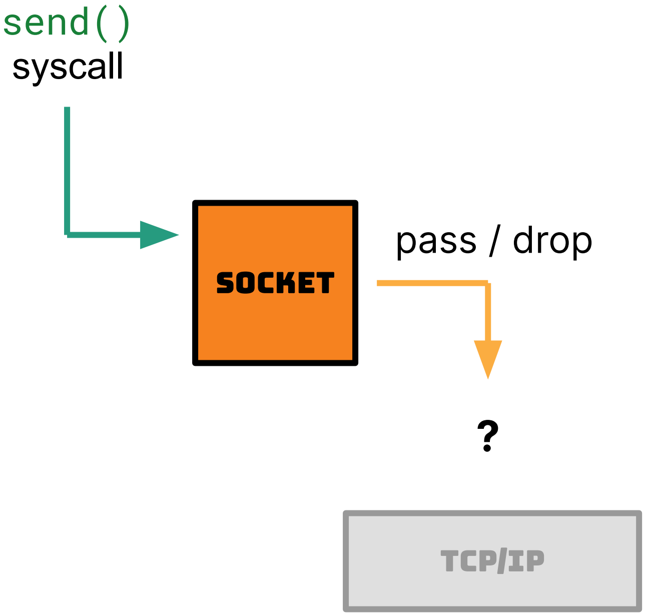
Policy on egress → Program input



```
struct sk_msg_md {
        __bpf_md_ptr(void *, data);
        __bpf_md_ptr(void *, data_end);

        __u32 family;
        __u32 remote_ip4;
        __u32 local_ip4;
        __u32 remote_ip6[4];
        __u32 local_ip6[4];
        __u32 remote_port;
        __u32 local_port;
        __u32 size;

        __bpf_md_ptr(struct bpf_sock *, sk);
};
```

message data

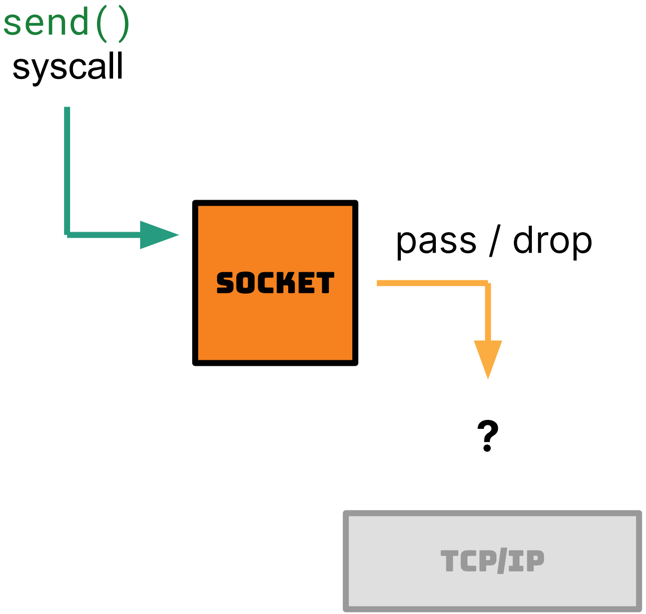
Policy on egress → Program input



```
struct sk_msg_md {
        __bpf_md_ptr(void *, data);
        __bpf_md_ptr(void *, data_end);

        __u32 family;
        __u32 remote_ip4;
        __u32 local_ip4;
        __u32 remote_ip6[4];
        __u32 local_ip6[4];
        __u32 remote_port;
        __u32 local_port;
        __u32 size;

        __bpf_md_ptr(struct bpf_sock *, sk);
};
```

src & dst info

Policy on egress → Program input



send()
syscall

SOCKET

pass / drop

?

TCP/IP

```
struct sk_msg_md {
        __bpf_md_ptr(void *, data);
        __bpf_md_ptr(void *, data_end);

        __u32 family;
        __u32 remote_ip4;
        __u32 local_ip4;
        __u32 remote_ip6[4];
        __u32 local_ip6[4];
        __u32 remote_port;
        __u32 local_port;
        __u32 size;

        __bpf_md_ptr(struct bpf_sock *, sk);
};
```

other socket info

Policy on egress → Example

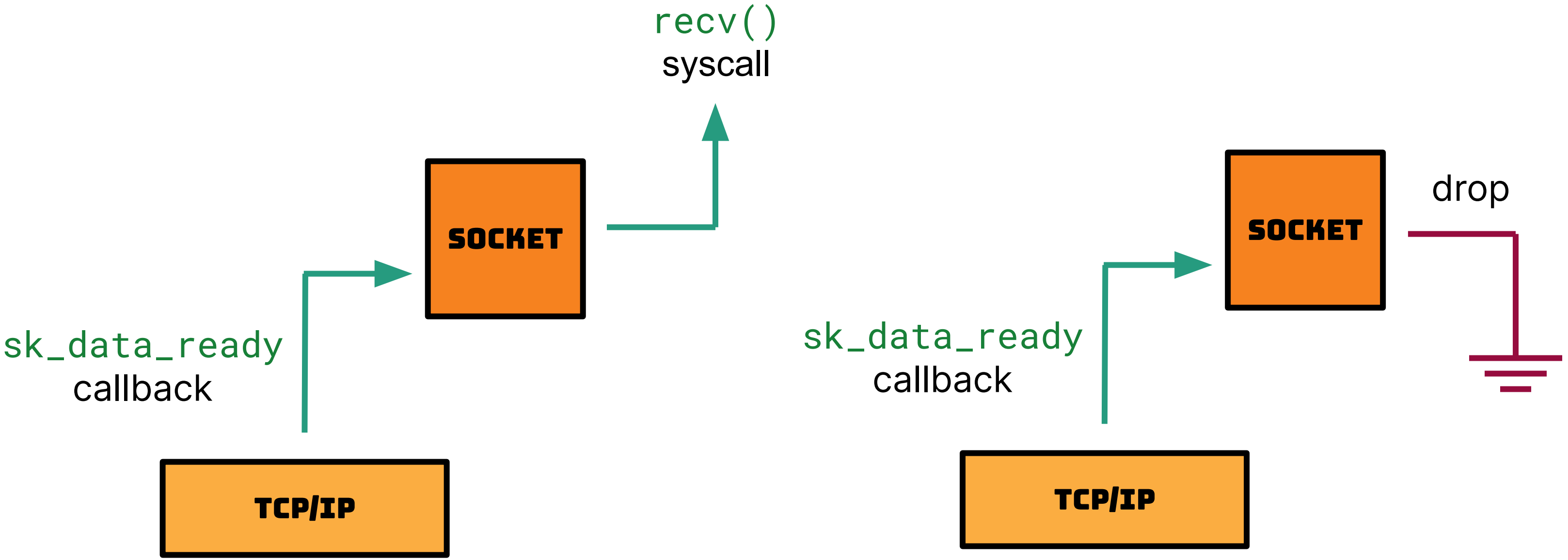
```
#define TEST_NET_1_ADDR IP4(192,   0,   2, 0)
#define TEST_NET_1_MASK IP4(255, 255, 255, 0)

SEC("sk_msg")
int sk_msg_prog(struct sk_msg_md *msg)
{
     if (msg->family != AF_INET)
          return SK_PASS;
     if ((msg->remote_ip4 & TEST_NET_1_MASK) != TEST_NET_1_ADDR)
          return SK_PASS;

     /* Drop anything destined to 192.0.2.0/24 documentation range */
     return SK_DROP;
}
```

Policy on egress → Test run

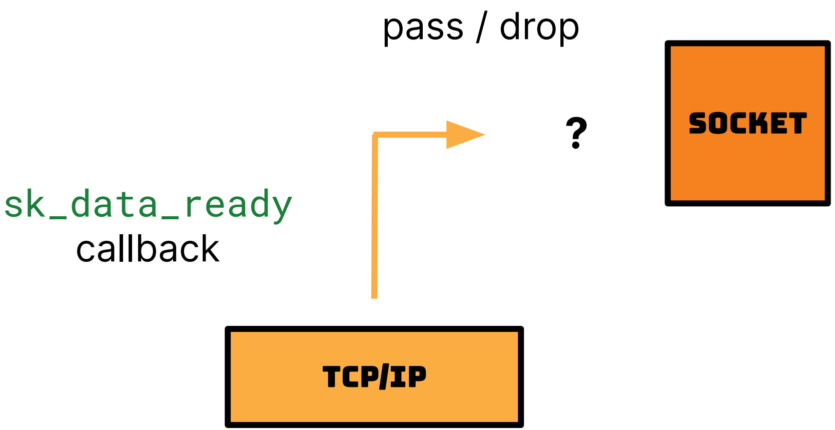
```
# nc -lke /bin/true 1234 &
#
# echo -n a | strace -e sendto nc 127.0.0.1 1234
sendto(3, "a", 1, 0, NULL, 0)           = 1
+++ exited with 0 +++
#
# echo -n b | strace -e sendto nc 192.0.2.1 1234
sendto(3, "b", 1, 0, NULL, 0)           = -1 EACCES (Permission denied)
Ncat: Permission denied.
+++ exited with 1 +++
```

https://github.com/jsitnicki/srecon-2023-sockmap/tree/main/examples/policy-egress

Policy on ingress



Policy on ingress → How?

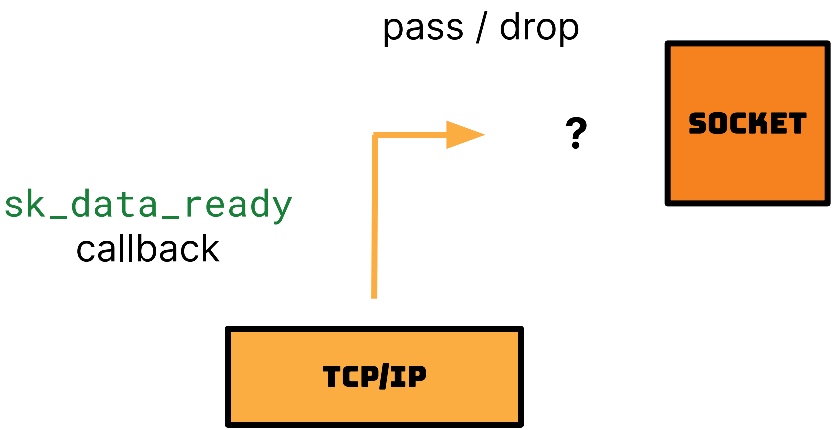


BPF_PROG_TYPE_**SK_SKB** prog

→ attached to BPF_**SK_SKB_VERDICT** hook

→ returns SK_**DROP** or SK_**PASS** verdict

Policy on ingress → Program input

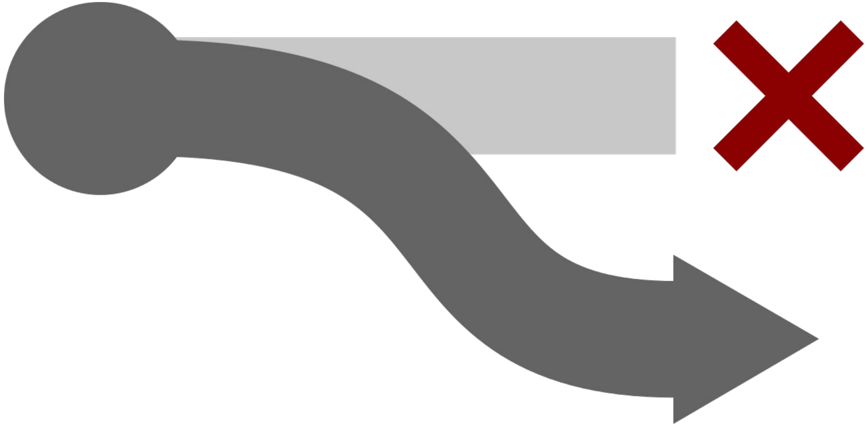


packet payload &
metadata from network stack

```
struct __sk_buff {
        …
        // too many fields to include here
        // see the link at the bottom
        …
};
```

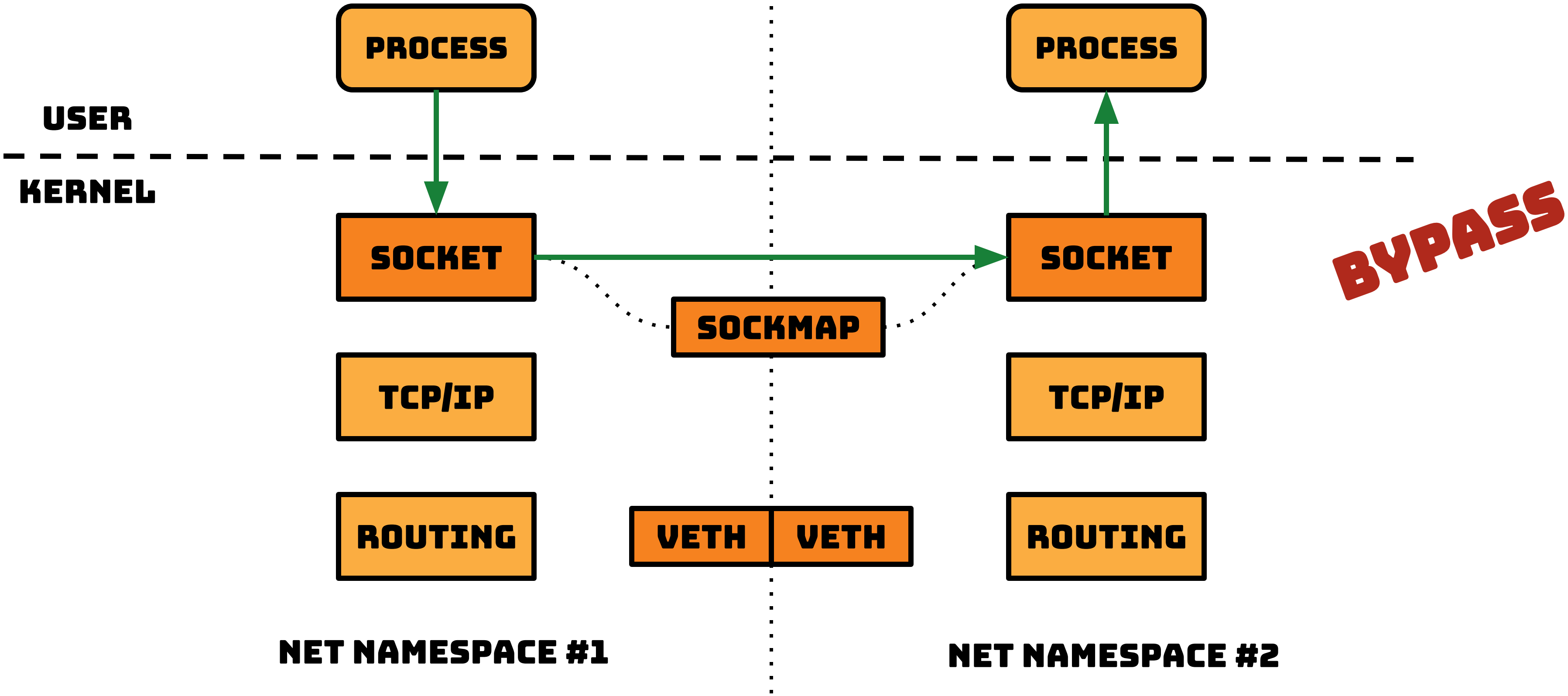
https://elixir.bootlin.com/linux/v6.5/source/include/uapi/linux/bpf.h#L6005

REDIRECT

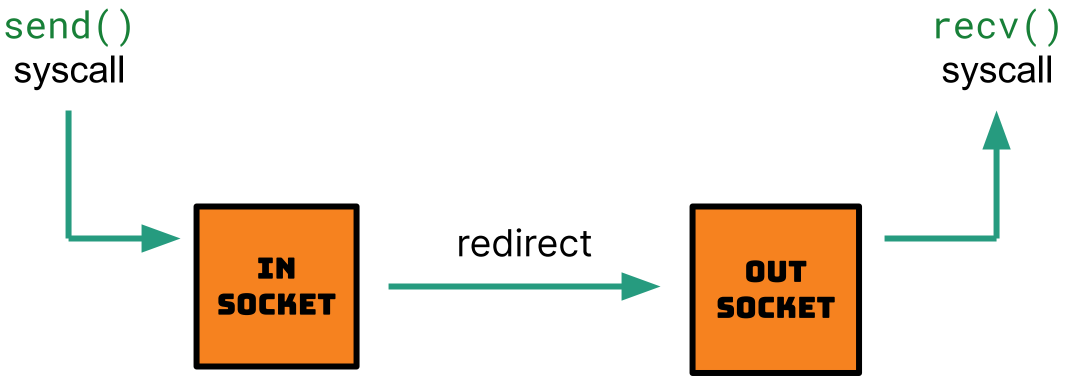


Wikimedia Commons, CC BY 3.0 DEED

Redirect use case → Bypass for containers



Redirect → send to local



send() syscall

IN SOCKET

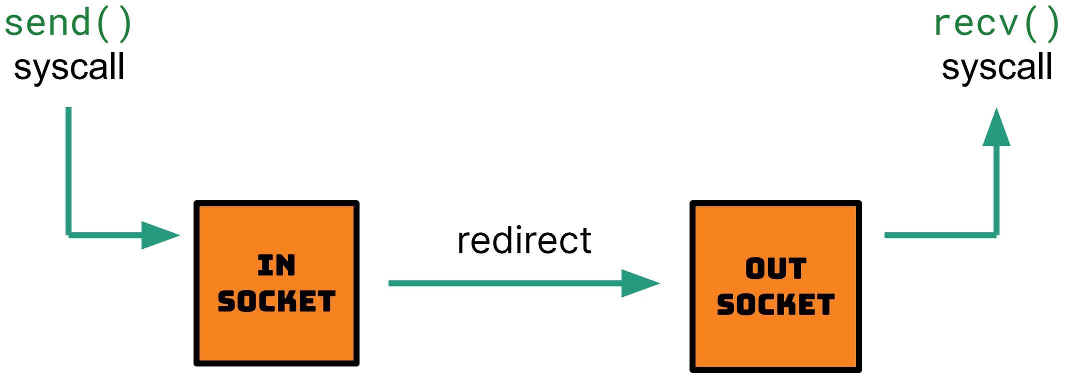
redirect

OUT SOCKET

recv() syscall

Like socketpair() or pipe()

Redirect → send to local → How?



BPF_PROG_TYPE_**SK_MSG** prog

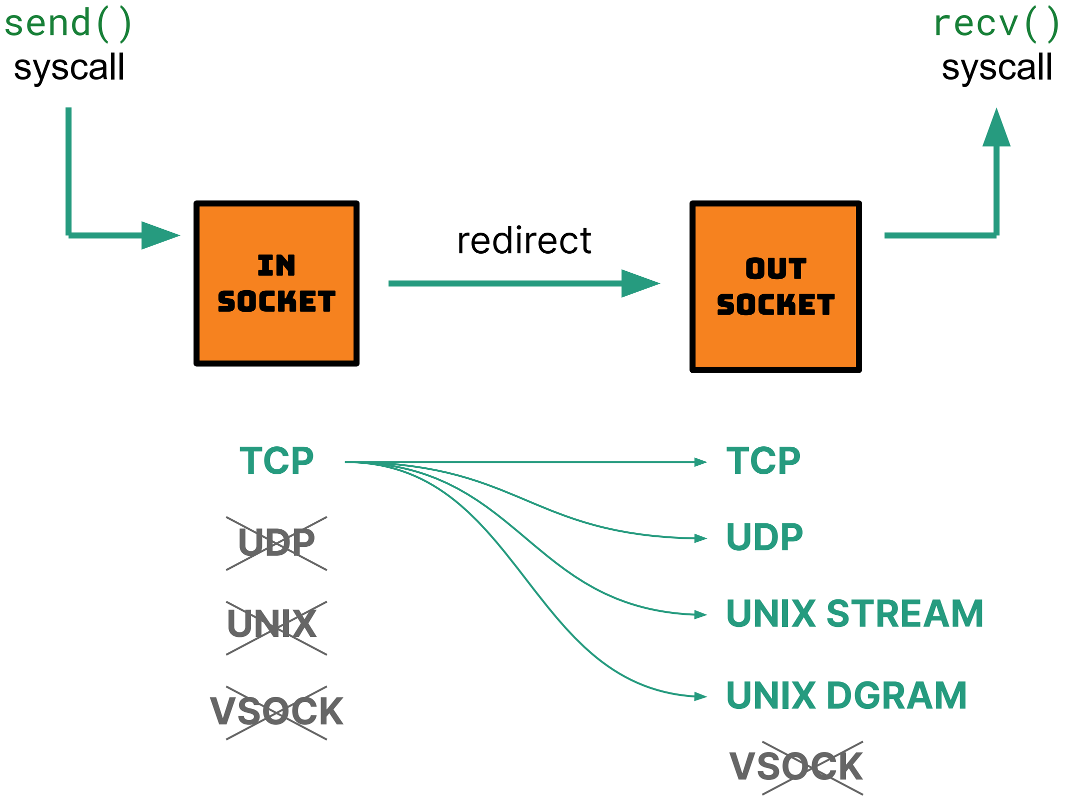
→ attached to BPF_**SK_MSG_VERDICT** hook

→ calls bpf_**msg**_redirect_hash/map() with BPF_F_**INGRESS** flag

→ returns SK_**PASS**

selects target socket

Redirect → send to local → What?



send()
syscall

IN SOCKET

redirect

OUT SOCKET

recv()
syscall

TCP

~~UDP~~

~~UNIX~~

~~VSOCK~~

TCP

UDP

UNIX STREAM

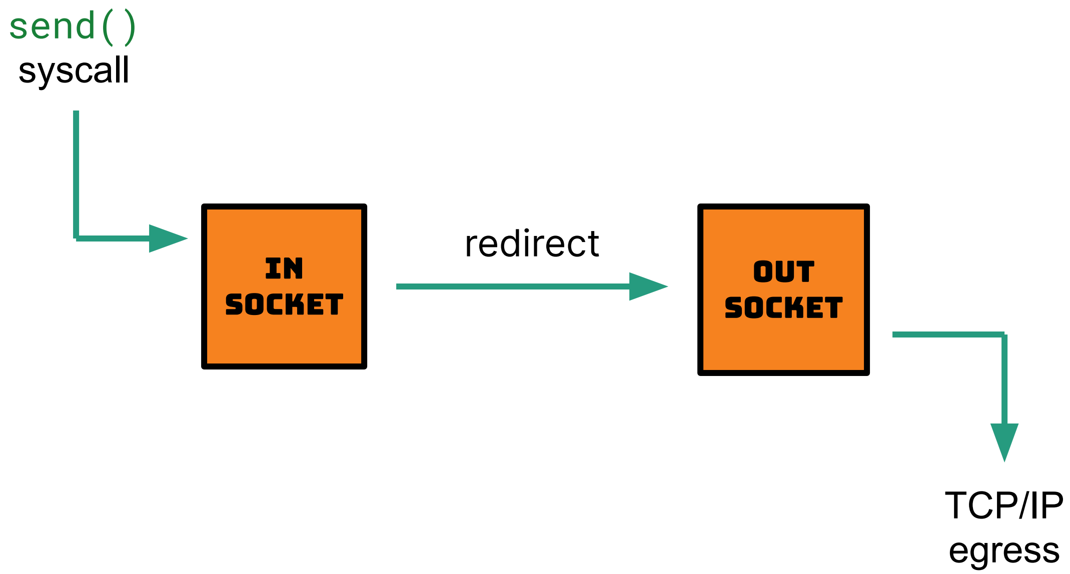
UNIX DGRAM

~~VSOCK~~

Redirect → send to local → Example

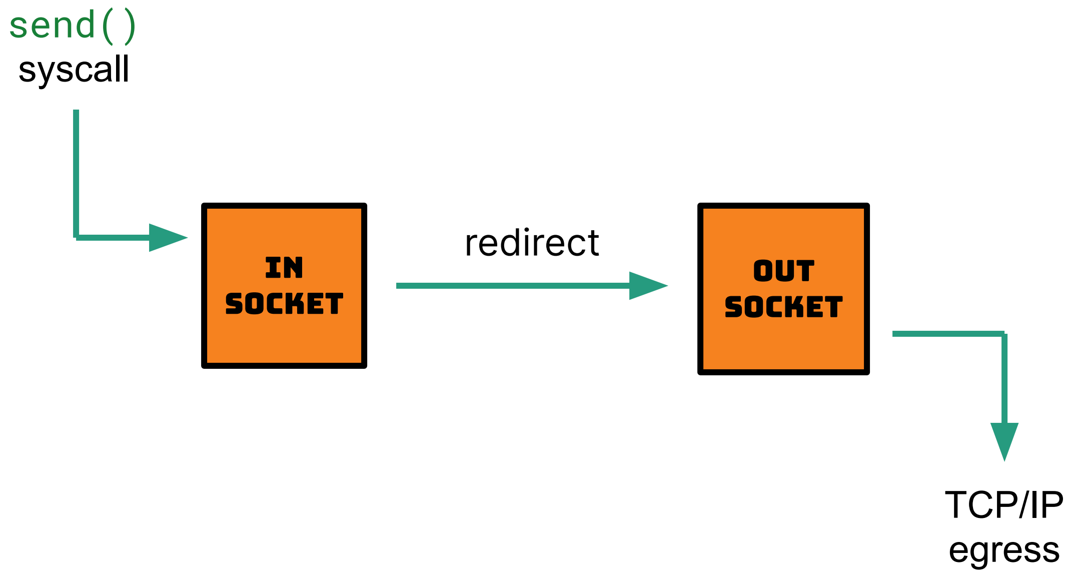
```
SEC("sk_msg")
int sk_msg_redir_ingress(struct sk_msg_md *msg)
{
    __u32 key = 0;

    if (msg->remote_port == bpf_htonl(53))
        key = 1;

     return bpf_msg_redirect_map(msg, &sockmap, key, BPF_F_INGRESS);
}
```

Redirect → send to egress



Sort of like splice()
(pipe → socket)

Redirect → send to egress → How?



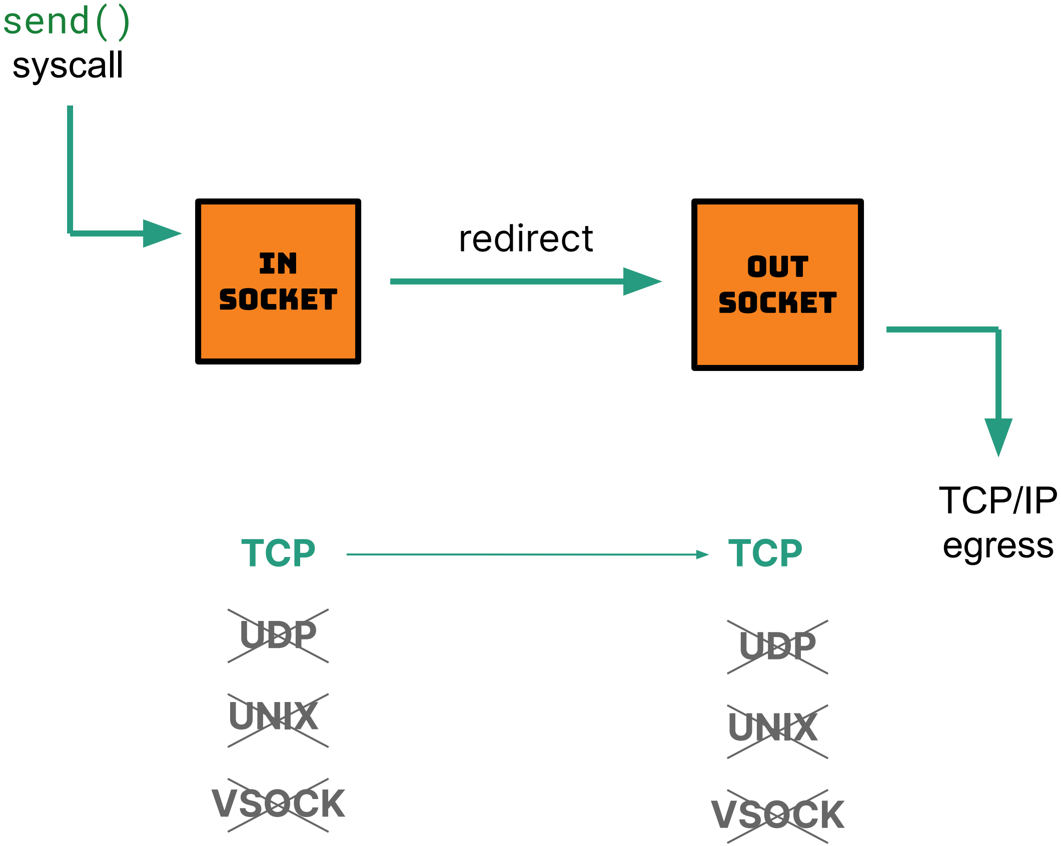
BPF_PROG_TYPE_**SK_MSG** prog

→ attached to BPF_**SK_MSG_VERDICT** hook

→ calls bpf_**msg**_redirect_hash/map() without any flags

→ returns SK_**PASS**

Redirect → send to egress → What?



send()
syscall

IN SOCKET

redirect

OUT SOCKET

TCP/IP
egress

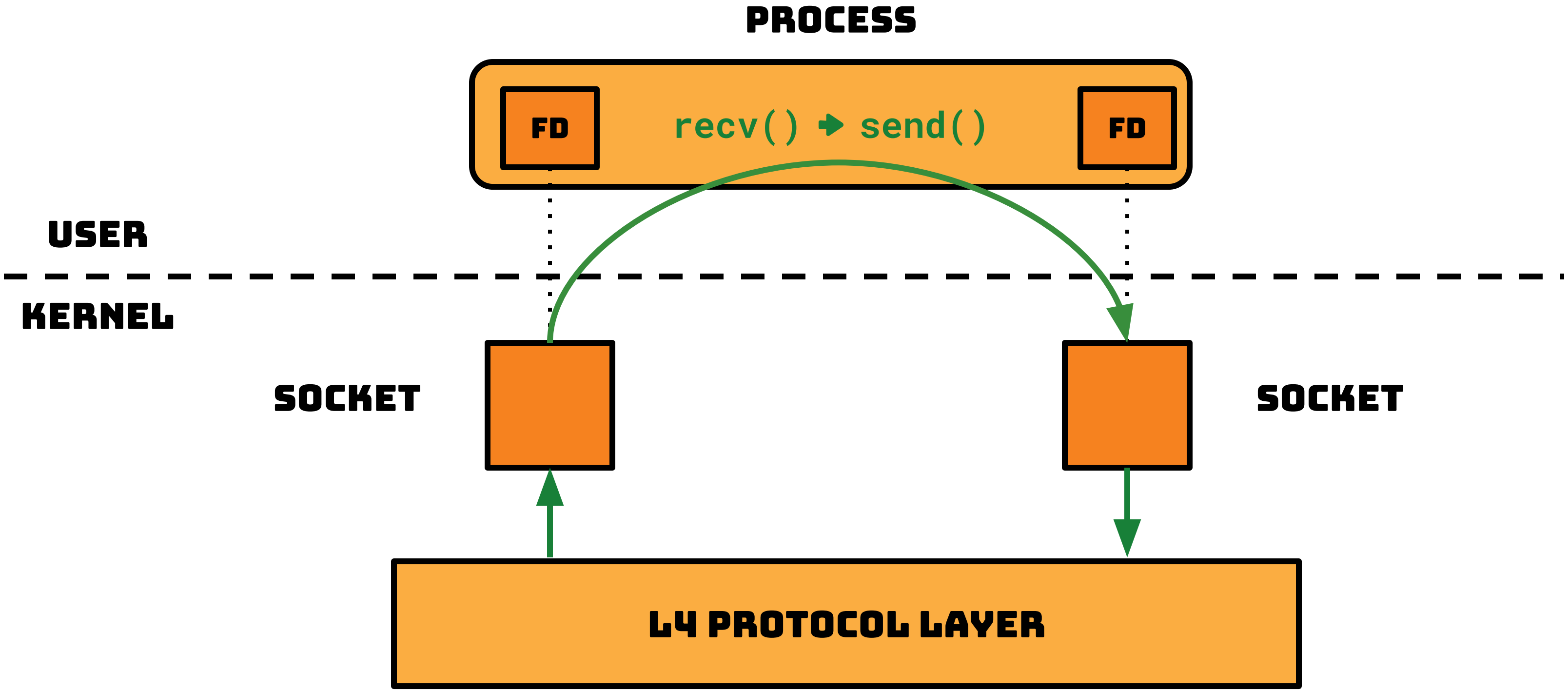
TCP → TCP

~~UDP~~ ~~UDP~~

~~UNIX~~ ~~UNIX~~

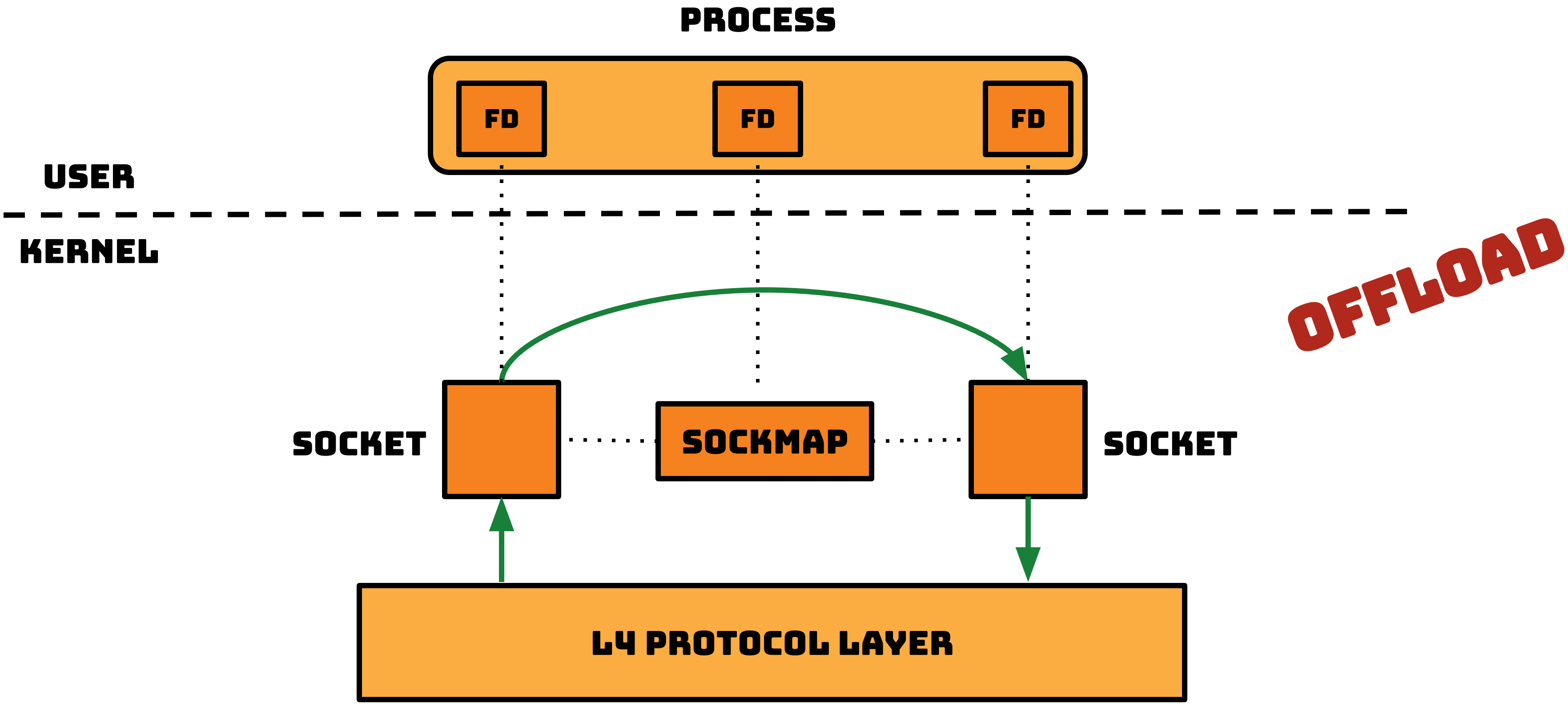
~~VSOCK~~ ~~VSOCK~~

Redirect use case → L7 network proxy

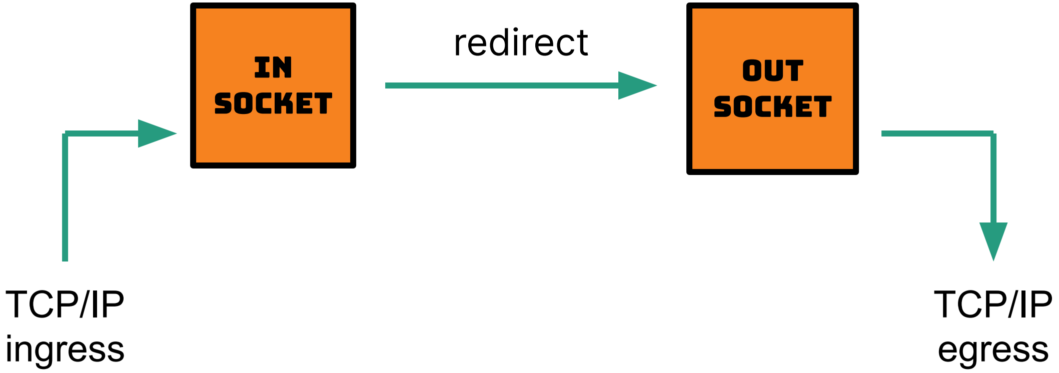


Examples: `socat`, `systemd-socket-proxyd`

Redirect use case → L7 network proxy

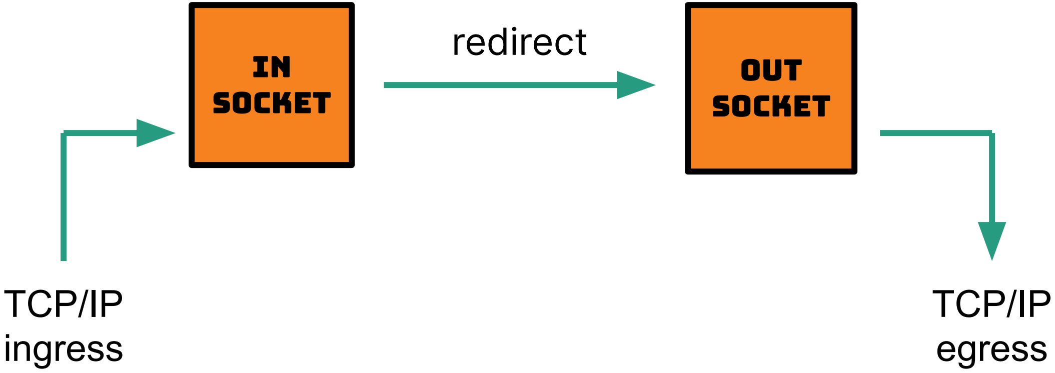


Redirect → ingress to egress



Like double `splice()`
(socket → pipe → socket)

Redirect → ingress to egress → How?



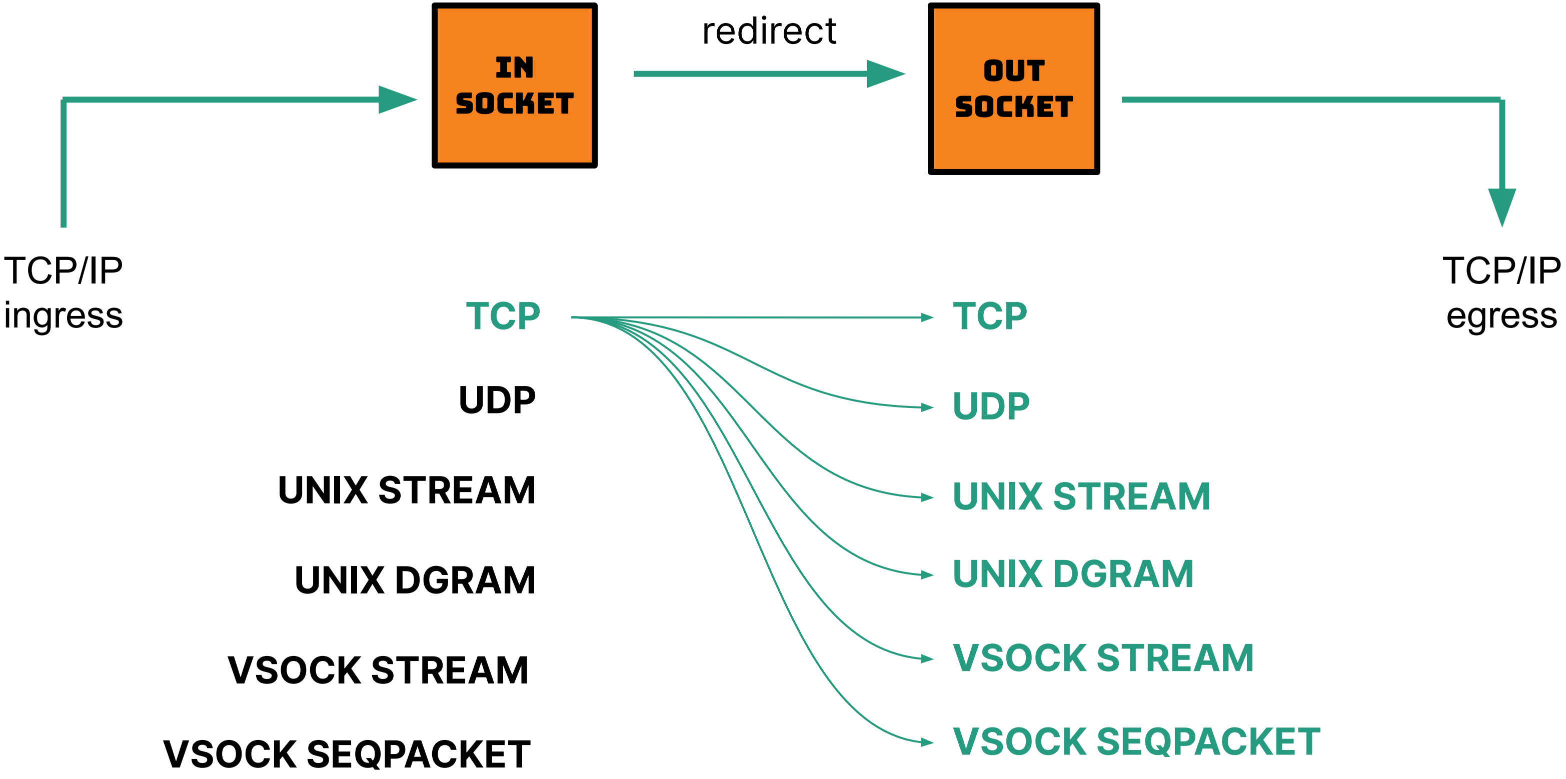
BPF_PROG_TYPE_**SK_SKB** prog

→ attached to BPF_**SK_SKB_VERDICT** hook

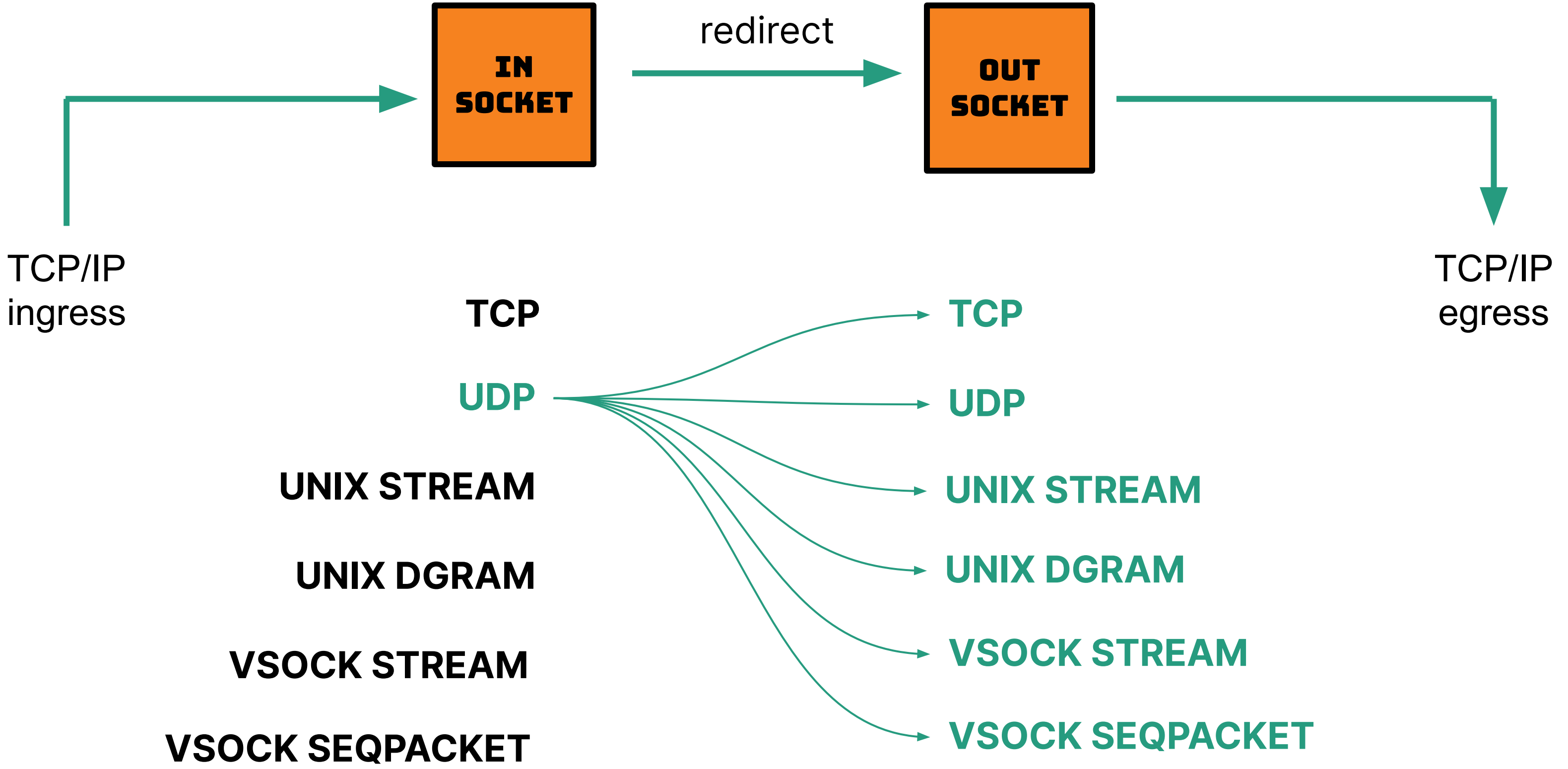
→ calls bpf_sk_redirect_hash/map() without any flags

→ returns SK_**PASS**

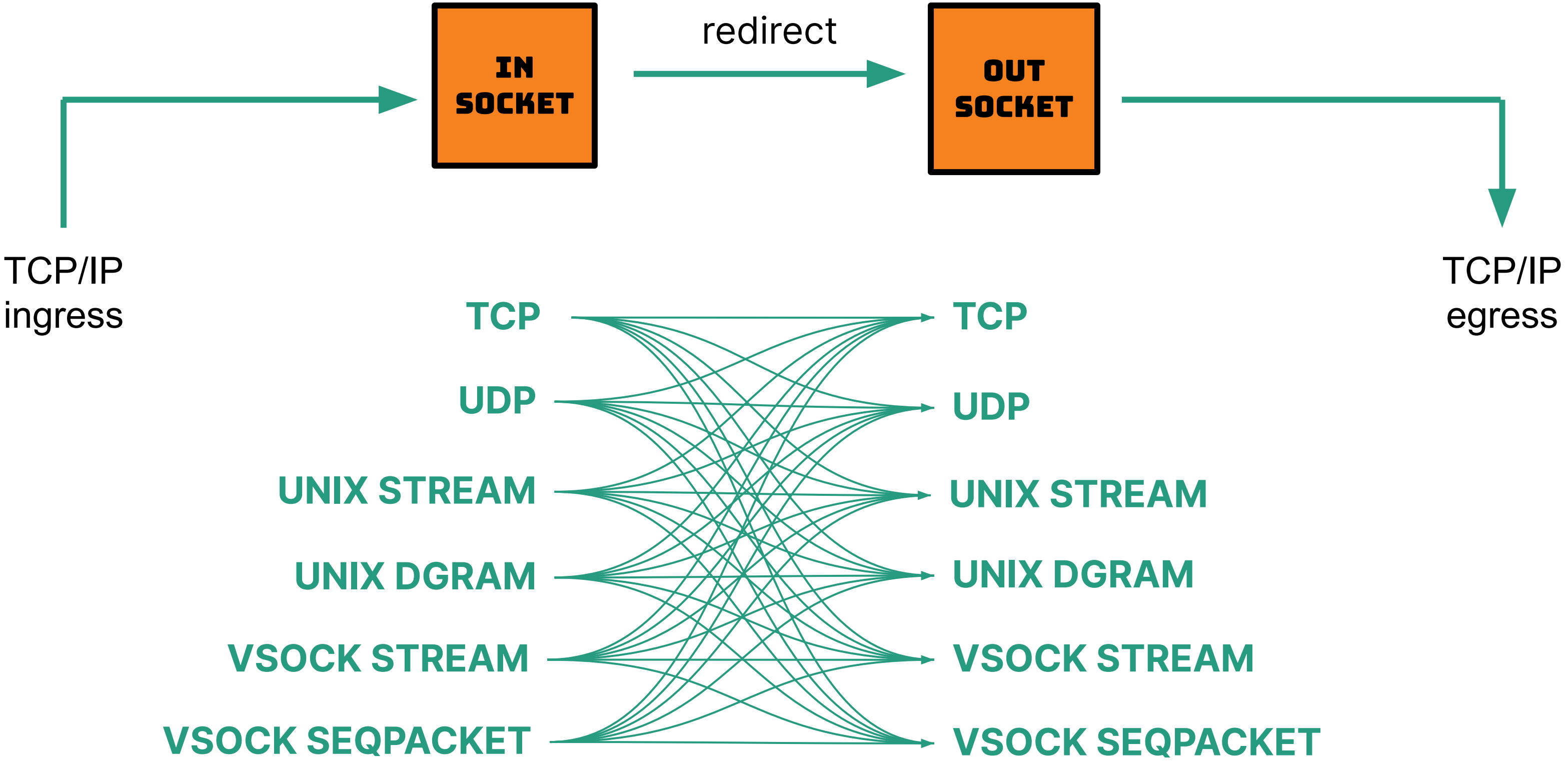
Redirect → ingress to egress → What?



Redirect → ingress to egress → What?

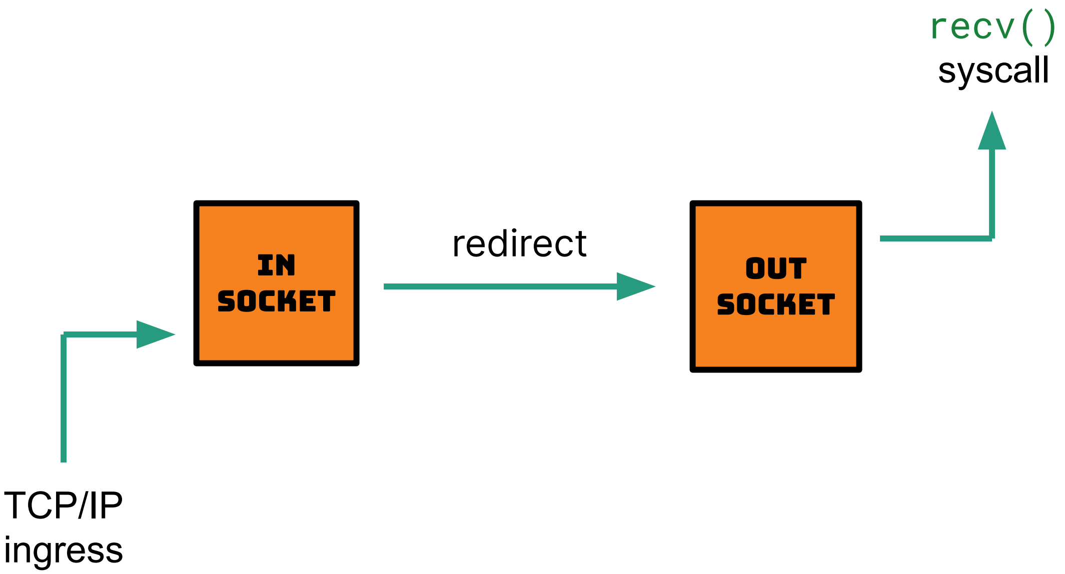


Redirect → ingress to egress → What?



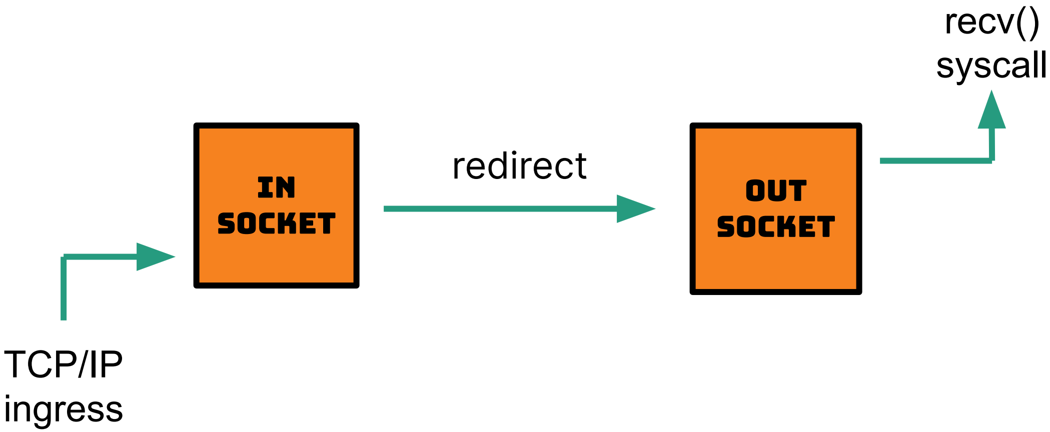
any to any

Redirect → ingress to local



Also sort of like `splice()`
(socket → pipe)

Redirect → ingress to local → How?



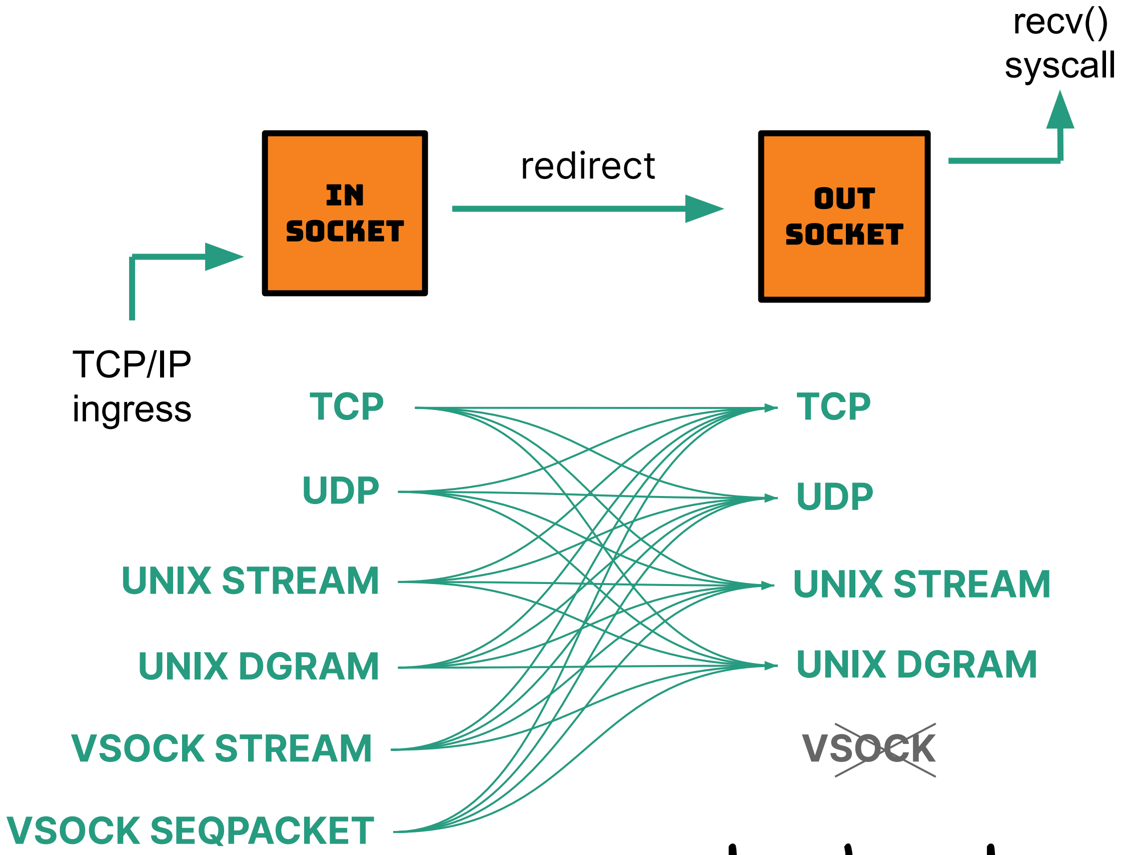
BPF_PROG_TYPE_**SK_SKB** prog

→ attached to BPF_**SK_SKB_VERDICT** hook

→ calls bpf_**sk**_redirect_hash/map() with BPF_F_**INGRESS** flag

→ returns SK_**PASS**

Redirect → ingress to local → What?



recv()
syscall

IN SOCKET

redirect

OUT SOCKET

TCP/IP
ingress

TCP

UDP

UNIX STREAM

UNIX DGRAM

VSOCK STREAM

VSOCK SEQPACKET

TCP

UDP

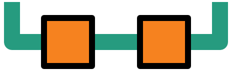
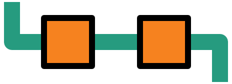
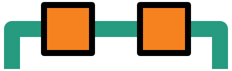
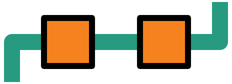
UNIX STREAM

UNIX DGRAM

~~VSOCK~~

any to almost any

Cheatsheet - Redirect with SOCKMAP

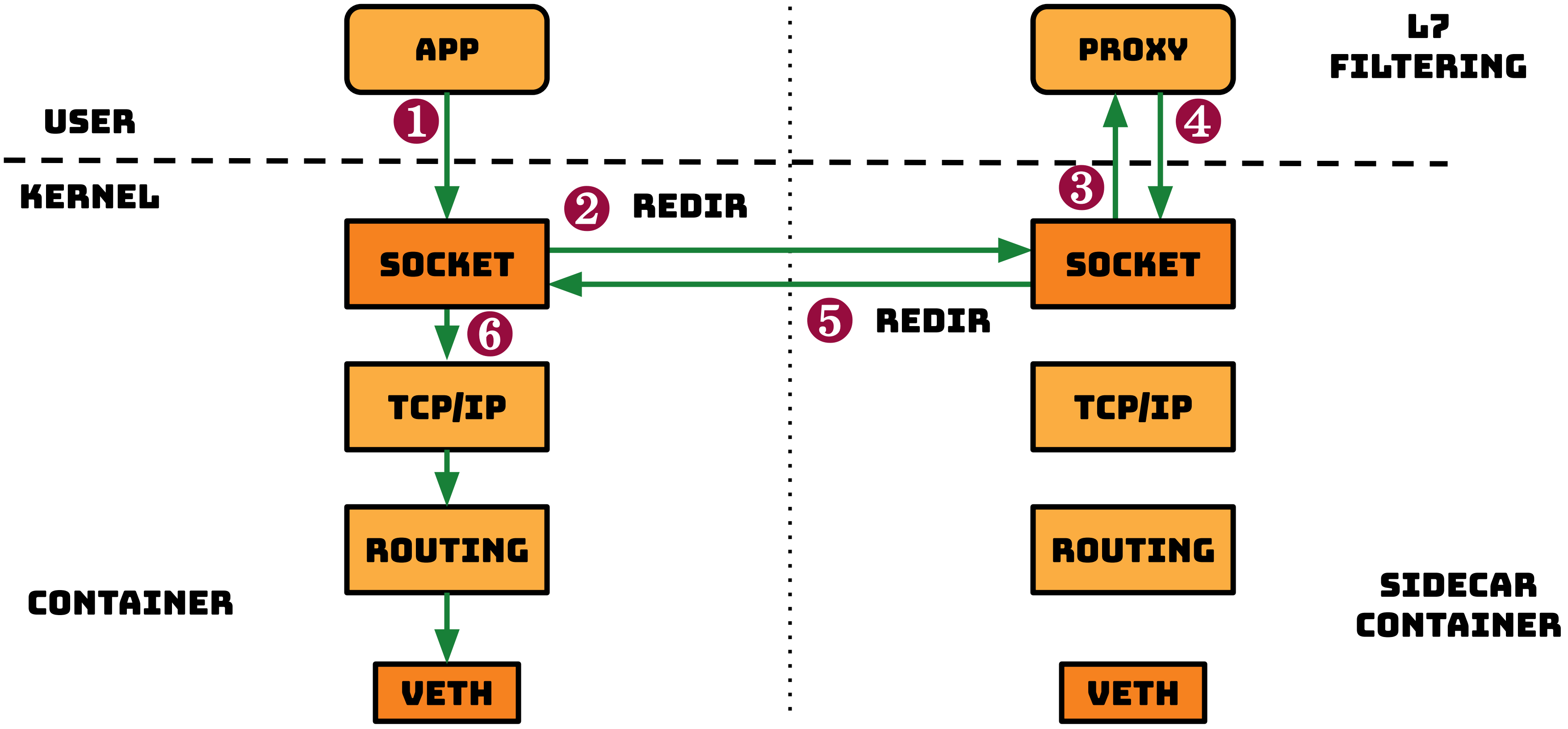
redirect scenario	program type BPF_PROG_TYPE_*	attach type BPF_*	redirect function	redirect flag	in socket type	out socket type
send to local 	SK_MSG	SK_MSG_VERDICT	bpf_msg_redirect_*()	BPF_F_INGRESS	TCP	any but VSOCK
send to egress 	SK_MSG	SK_MSG_VERDICT	bpf_msg_redirect_*()	none	TCP	TCP
ingress to egress 	SK_SKB	SK_SKB_VERDICT	bpf_sk_redirect_*()	none	any	any
ingress to local 	SK_SKB	SK_SKB_VERDICT	bpf_sk_redirect_*()	BPF_F_INGRESS	any	any but VSOCK

Section

CLOUDFLARE

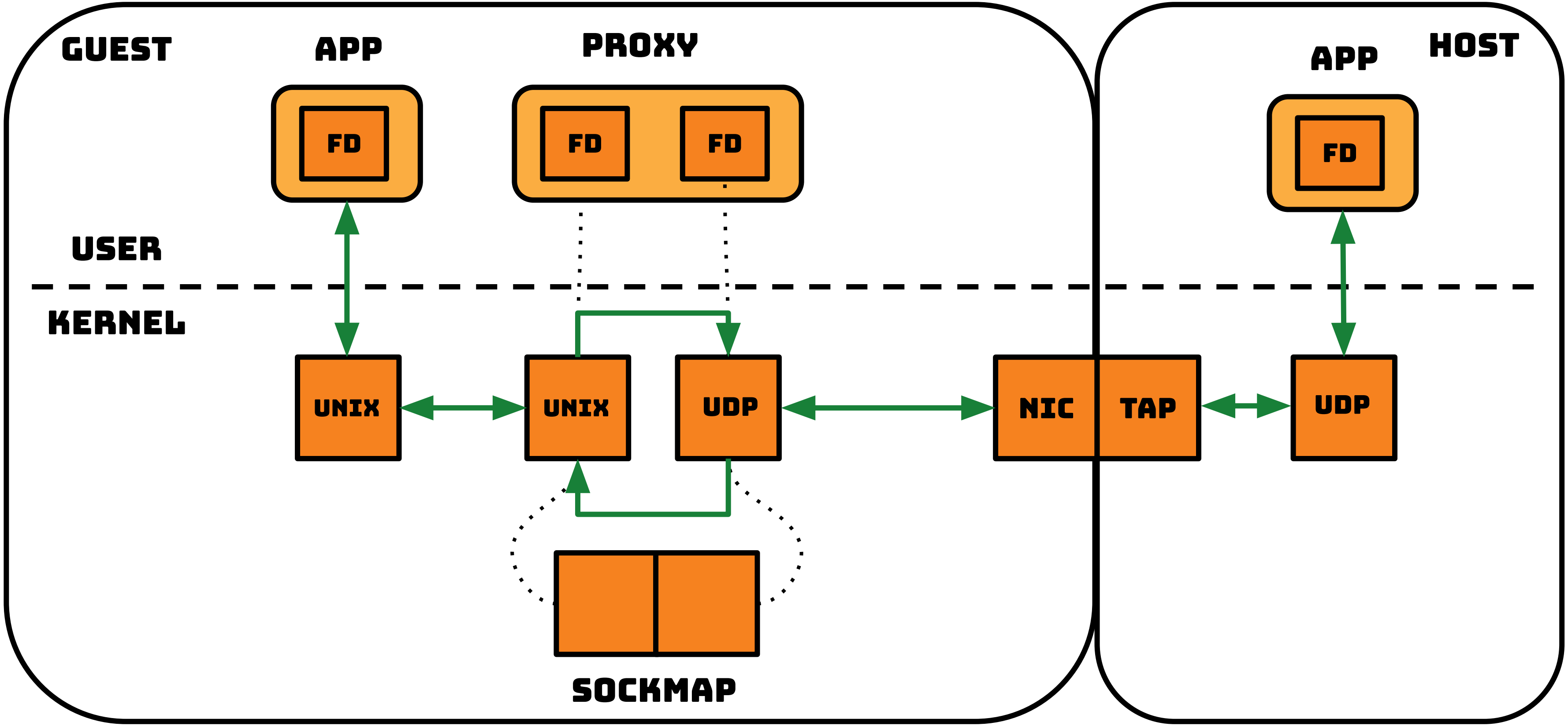
Real life use-cases

Cilium project (CNI for K8S)



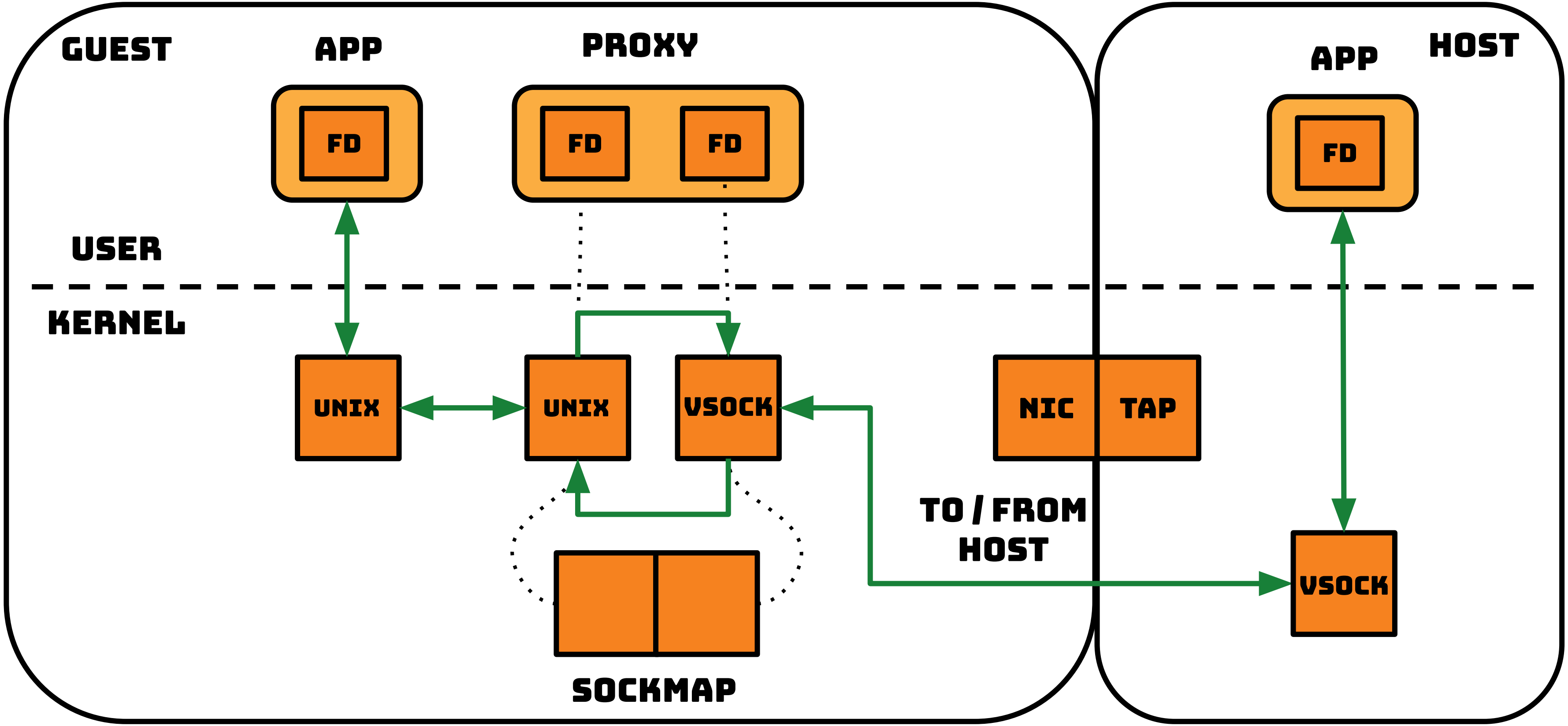
https://cilium.io/blog/2019/02/12/cilium-14/

Bytedance (TikTok)



https://lore.kernel.org/bpf/20210331023237.41094-1-xiyou.wangcong@gmail.com/

Bytedance (TikTok) → Improved



https://lore.kernel.org/all/20230327-vsock-sockmap-v4-0-c62b7cd92a85@bytedance.com/

Section

Where to learn more?

Documentation & Resources

1) Linux Kernel → BPF Documentation → SOCKMAP and SOCKHASH map

   Includes links to unit tests with API usage examples

2) LPC 2018: Combining kTLS and BPF for Introspection and Policy Enforcement

   See Daniel & John talk about Cilium SOCKMAP + kTLS use case (video, slides, paper)

3) Cloudflare Blog: SOCKMAP - TCP splicing of the future

   Read Marek review SOCKMAP from L7 proxy perspective

4) eBPF Summit 2020: Steering connections to sockets with BPF socket lookup hook

   Another use case for SOCKMAP as a container (video, slides, code)

Documentation & Resources

Code + Slides

https://github.com/jsitnicki/srecon-2023-sockmap

```
$ logout
```

Thank you

`jakub@cloudflare.com`

`@jkbs0` @ ~~Twitter~~ X

Mailing lists:

`bpf@vger.kernel.org`

`netdev@vger.kernel.org`

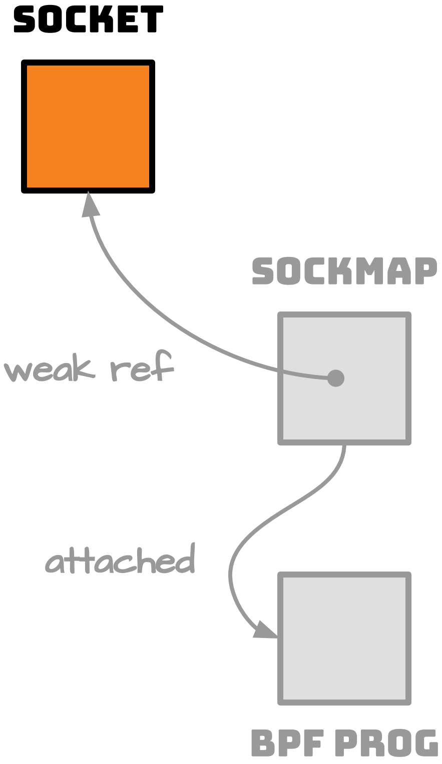
`#ebpf-kernel-dev` @ `cilium.slack.com`

code & slides repo

CLOUDFLARE

Overflow slides

What sockets you can't use



connected (established) socket:

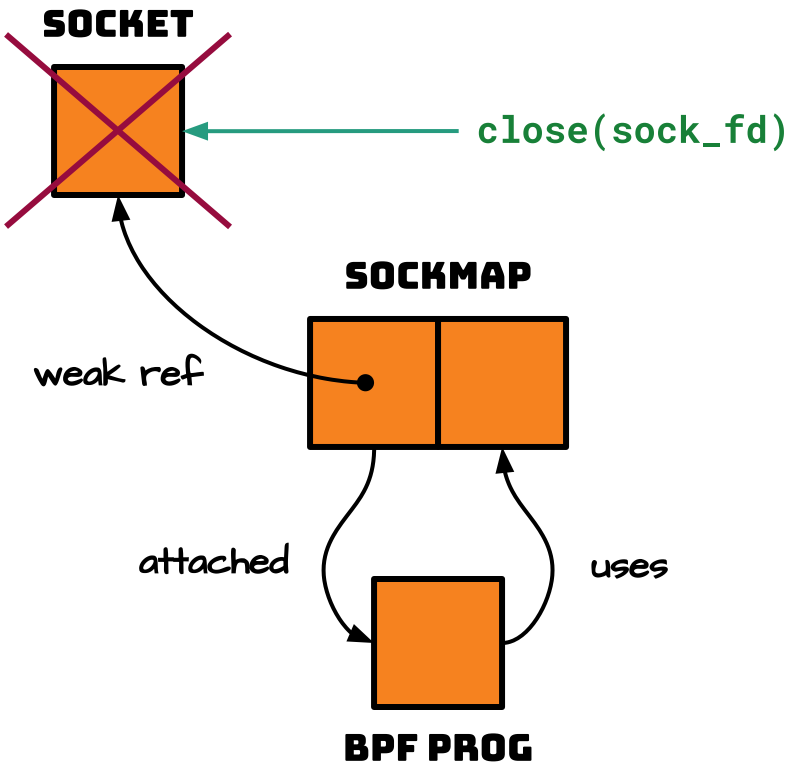
- ❏ UNIX SEQPACKET
- ❏ VSOCK DGRAM

`man 7 {tcp,udp,unix,vsock}`

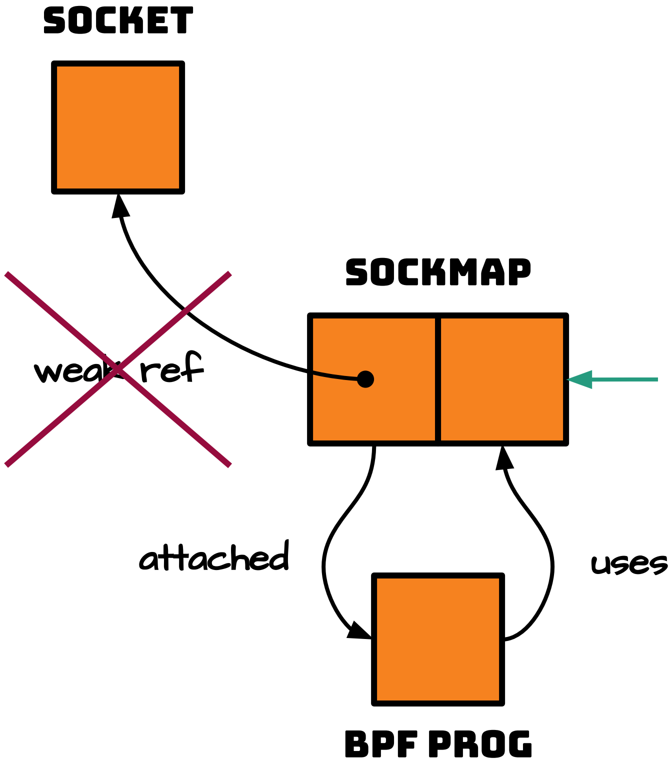
How to tear it down?

[pxhere.com, CC0](https://pxhere.com)

(A) destroy the socket



(B) remove socket from sockmap



SOCKET

SOCKMAP

~~weak ref~~

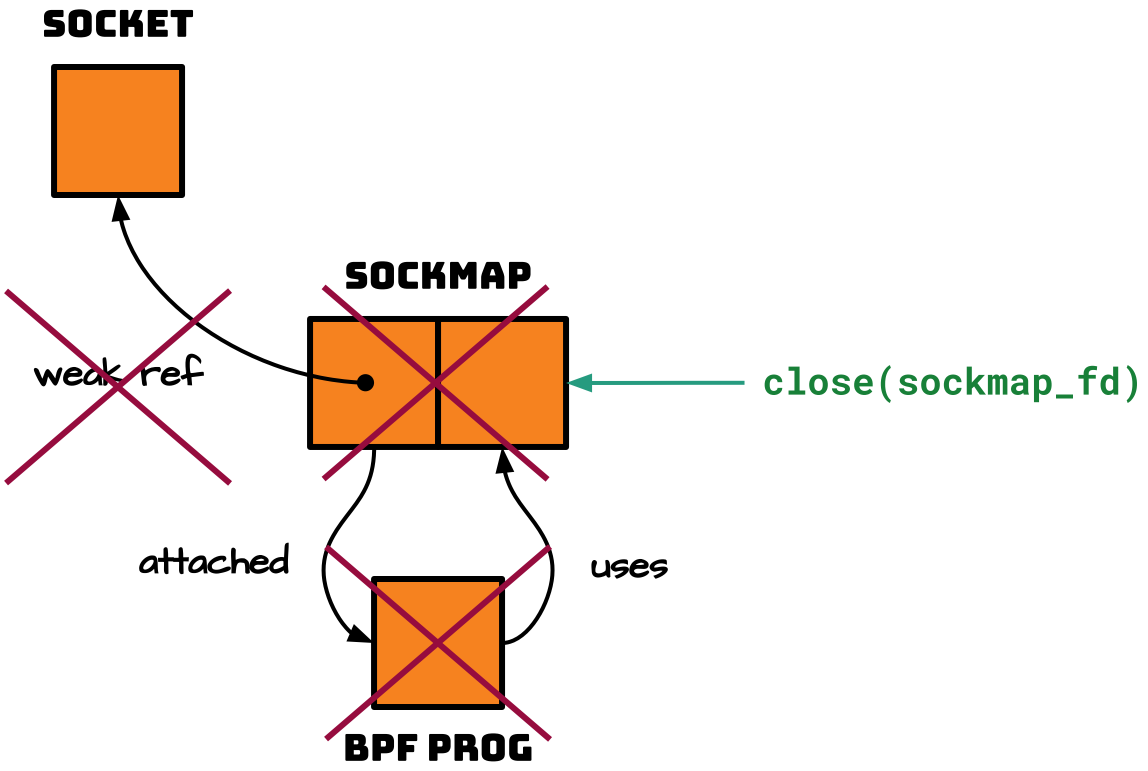
attached

uses

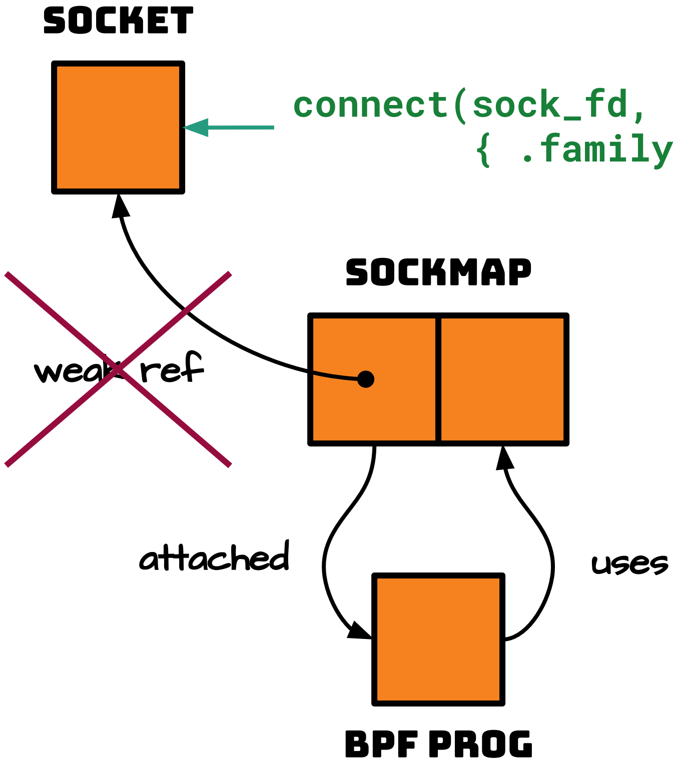
BPF PROG

`bpf(BPF_MAP_DELETE_ELEM, …)`

(C) destroy the sockmap



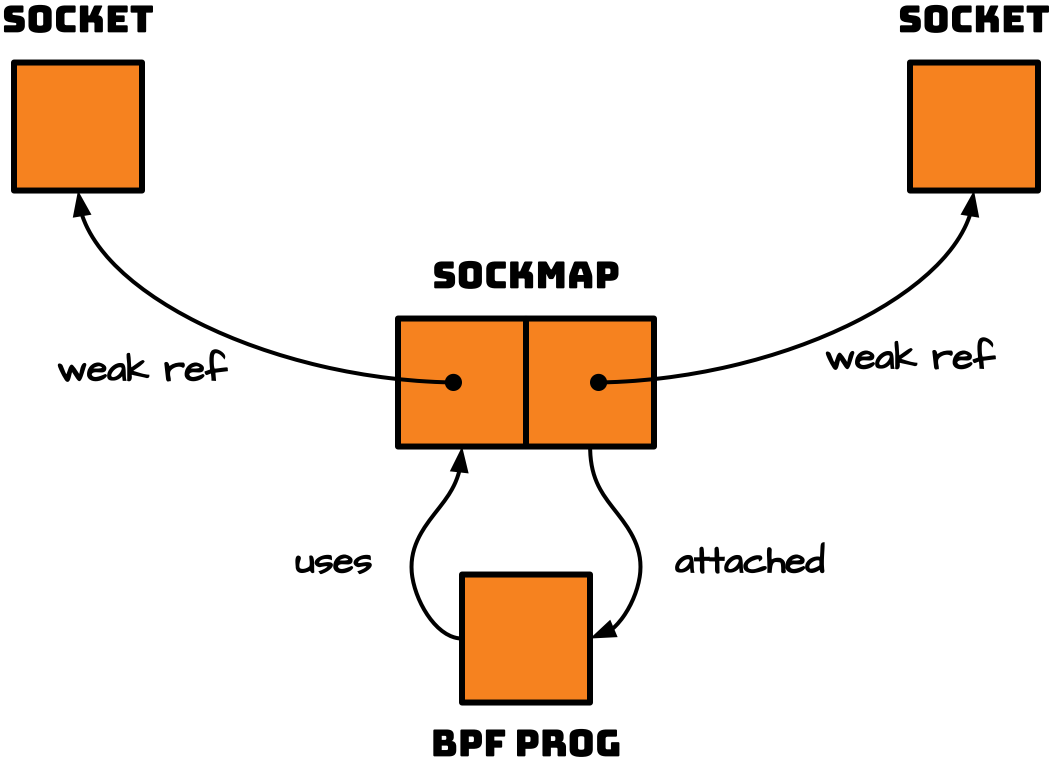
(D) disconnect the socket (rare)



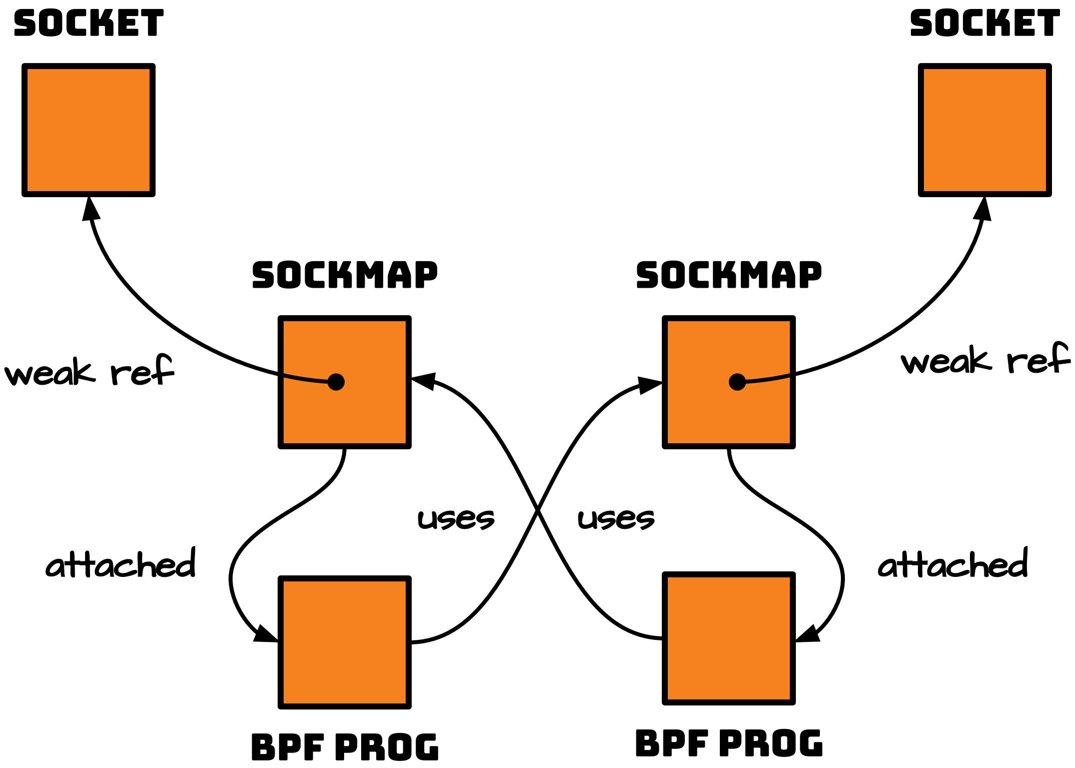
```
connect(sock_fd,
        { .family = AF_UNSPEC }, …)
```

💡 See `connect(2)` man page

Two sockets in one sockmap



Two sockets in two sockmaps


SOCKET
SOCKET
SOCKMAP
SOCKMAP
weak ref
weak ref
uses
uses
attached
attached
BPF PROG
BPF PROG