2021-09-23

CLOUDFLARE

BPF User Experience Rough Edges

Arthur Fabre
Jakub Sitnicki

CLOUDFLARE

Agenda

What can go wrong when:

1. humans write BPF code (Jakub)
2. machines write BPF code (Arthur)

CLOUDFLARE

BPF is great!

Our goals:

- ❏ share know-how
- ❏ trigger discussion
- ❏ show where help is needed

CLOUDFLARE

Here is the story...

Meet our dev **buzz**

```
$ nstat
#kernel
IpInReceives                    238653             0.0
IpForwDatagrams                 54032              0.0
IpInDelivers                    182008             0.0
IpOutRequests                   280068             0.0
IpOutDiscards                   1454               0.0
IpOutNoRoutes                   2489               0.0
IcmpInMsgs                      198                0.0
IcmpInErrors                    5                  0.0
IcmpInDestUnreachs              197                0.0
IcmpInEchos                     1                  0.0
IcmpOutMsgs                     45                 0.0
...
```

... but per cgroup!

CLOUDFLARE

Our benchmark for UX

1. create BPF map
2. load BPF program
3. create BPF link
4. pin BPF link
5. inspect attached program
6. read BPF map from user-space

Sad Panda if we run into problems

CLOUDFLARE

Our test environments

```
vagrant@bullseye:~$ hostnamectl
…
  Operating System: Debian GNU/Linux 11 (bullseye)
            Kernel: Linux 5.10.0-8-amd64
      Architecture: x86-64
```

stable kernel

```
[vagrant@f34 ~]$ hostnamectl
Operating System: Fedora 34 (Cloud Edition)
     CPE OS Name: cpe:/o:fedoraproject:fedora:34
          Kernel: Linux 5.13.13-200.fc34.x86_64
    Architecture: x86-64
```

CLOUDFLARE

Little helper to change UID and raise caps

```
#!/bin/bash
# -*- mode: shell-script -*-
#
# runas: Run command as given user with given capabilities.
#

readonly user="$1"
readonly caps="-all,$2"
shift 2

exec setpriv \
     --reuid=$user --regid=$user --clear-groups \
     --inh-caps=$caps --ambient-caps=$caps --bounding-set=$caps \
     --reset-env -- env PATH=/usr/sbin:/usr/bin "$@"
```

CLOUDFLARE

It is similar to systemd service unit like

```
[Service]
…
User=buzz
Group=buzz
AmbientCapabilities=CAP_BPF CAP_NET_ADMIN CAP_SYS_RESOURCE
CapabilityBoundingSet=CAP_BPF CAP_NET_ADMIN CAP_SYS_RESOURCE
…
```

CLOUDFLARE

(1) Creating a BPF map

CLOUDFLARE

buzz needs a BPF map to count things!

```
int main(void)
{
    int fd = bpf_create_map_name(BPF_MAP_TYPE_ARRAY, "pkt_stats",
                                 /*key_size=*/ 4, /*value_size=*/ 8,
                                 /*max_entries=*/ 1, /*map_flags=*/ 0);
    if (fd < 0)
        error(EXIT_FAILURE, errno, "map create");

    close(fd);
    return EXIT_SUCCESS;
}
```

CLOUDFLARE

Let's create a BPF map!

```
vagrant@bullseye:/lpc-2021$ make
cc -Wall -Wextra -ggdb -o 01/map-create 01/map_create.c -lbpf


vagrant@bullseye:/lpc-2021$ 01/map-create
./map-create: map create: Operation not permitted


vagrant@bullseye:/lpc-2021$ strace -e bpf 01/map-create
bpf(BPF_MAP_CREATE, {map_type=BPF_MAP_TYPE_ARRAY, key_size=4, value_size=8,
 max_entries=1, map_flags=0, inner_map_fd=0, map_name="pkt_stats", ...}, 120)
 = -1 EPERM (Operation not permitted)
01/map-create: map create: Operation not permitted
+++ exited with 1 +++
```

CLOUDFLARE

Nowadays we should expect

```
$ /usr/sbin/sysctl kernel.unprivileged_bpf_disabled
kernel.unprivileged_bpf_disabled = 2

$ grep BPF_UNPRIV_DEFAULT_OFF /boot/config-5.10.0-8-amd64
CONFIG_BPF_UNPRIV_DEFAULT_OFF=y
```

unprivileged BPF disabled by default

need at least CAP_BPF

CLOUDFLARE

Give me superpowers!

```
$ sudo 01/map-create
01/map-create: map create: Operation not permitted
```

CLOUDFLARE

Where are my superpowers?

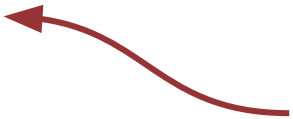
```
$ sudo perf ftrace -G '*sys_bpf' -g '*irq*' 01/map-create
...
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 0)               |  __x64_sys_bpf() {
...
 0)               |    array_map_alloc() {
 0)               |      capable() {
 0)               |        security_capable() {
 0)   0.097 us    |          cap_capable();
 0)   0.102 us    |          apparmor_capable();
 0)   0.519 us    |        }
 0)   0.721 us    |      }
 0)               |      bpf_map_charge_init() {
 0)   0.142 us    |        free_uid();
 0)   0.417 us    |      }
 0)   1.590 us    |    }
 0) + 16.631 us   |  }
```

We only get this far

CLOUDFLARE

In Linux v5.10.46 LTS we have

```
int bpf_map_charge_init(struct bpf_map_memory *mem, u64 size)
{
    …
    user = get_current_user();
    ret = bpf_charge_memlock(user, pages);
    if (ret) {
        free_uid(user);
        return ret;
    }
    …
}
```



We saw this in ftrace output

`bpf_charge_memlock()` used to be `bpf_map_precharge_memlock()` in v4.19.

CLOUDFLARE

Locked memory limit, of course 🤦

Debian Bullseye - 64 MiB

```
vagrant@bullseye:~$ sudo prlimit --memlock
RESOURCE DESCRIPTION                             SOFT     HARD UNITS
MEMLOCK  max locked-in-memory address space 67108864 67108864 bytes
```

Fedora 34 - 64 KiB

```
[vagrant@fedora ~]$ sudo prlimit --memlock
RESOURCE DESCRIPTION                           SOFT  HARD UNITS
MEMLOCK  max locked-in-memory address space 65536 65536 bytes
```

CLOUDFLARE

How to check current memlock usage?

```
$ sudo bpftrace --btf -e '
> BEGIN {
>   printf("%ld bytes", curtask->cred->user->locked_vm.counter << 12);
>   exit();
> }'
Attaching 1 probe...
67158016 bytes
```

PAGE_SHIFT

we're slightly over 64 MiB

CLOUDFLARE

Why are we over the limit?

Usage tracked per **user**, not per **task**.
But checked against a limit set per **task**.

Quota is shared between:

- BPF maps
- AF_XDP UMEM areas
- io_uring objects
- MSG_ZEROCOPY buffers
- perf event buffers

... created under the same **UID** (from task creds).

CLOUDFLARE

Solution? No limit, no problem

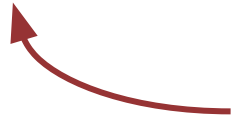
```
$ sudo strace -e prlimit64 \
> bpftool map create /sys/fs/bpf/pkt_stats \
>   type array name pkt_stats \
>   key 4 value 8 entries 1
prlimit64(0, RLIMIT_MEMLOCK, {rlim_cur=RLIM64_INFINITY, rlim_max=RLIM64_INFINITY}, NULL) = 0
+++ exited with 0 +++
```

```
$ sudo strace -e bpf prlimit --memlock=unlimited ./map-create
bpf(BPF_MAP_CREATE, {map_type=BPF_MAP_TYPE_ARRAY, key_size=4, ...}, 120) = 3
+++ exited with 0 +++
```

CLOUDFLARE

Perhaps a better solution? Dedicated UID

```
$ sudo bpftrace --btf -e '
> tracepoint:syscalls:sys_enter_geteuid {
>   printf("%ld bytes", curtask->cred->user->locked_vm.counter << 12);
> }' \
> -c '/usr/bin/setpriv --reuid=buzz id'
Attaching 1 probe...
uid=1001(buzz) gid=0(root) groups=0(root)
0 bytes
```



No allocations yet

CLOUDFLARE

BPF map created 👌

```
$ sudo ./runas buzz +cap_39 strace -e bpf 01/map-create
bpf(BPF_MAP_CREATE, {map_type=BPF_MAP_TYPE_ARRAY, ...}, 120) = 3
+++ exited with 0 +++
```

UID != 0 and CAP_BPF (cap_39)

CLOUDFLARE

Things changed in v5.11

BPF map allocations are now charged against **memory cgroup limit**

```
# mkdir /sys/fs/cgroup/test.slice
# echo $[32*1024*1024] > /sys/fs/cgroup/test.slice/memory.max
# echo $$ > /sys/fs/cgroup/test.slice/cgroup.procs

# bpftool map create /sys/fs/bpf/array_64M type array \
>         key 4 value $[64*1024] entries 1024 name array_64M
Error: map create failed: Cannot allocate memory
#

# echo $[128*1024*1024] > /sys/fs/cgroup/test.slice/memory.max
# bpftool map create /sys/fs/bpf/array_64M type array \
>         key 4 value $[64*1024] entries 1024 name array_64M
#
```

32 MiB limit

64 MiB map

128 MiB limit

CLOUDFLARE

(2) Loading the BPF program

CLOUDFLARE

Our little BPF program

```
SEC("cgroup_skb/ingress")
int count_pkts(struct __sk_buff *skb)
{
    __u32 key = STAT_IN_RECEIVES;
    __u64 *count = bpf_map_lookup_elem(&stats, &key);

    if (count)
        __sync_fetch_and_add(count, 1);

    return 1;
}
```

CLOUDFLARE

Build it, load it

```
$ clang -O2 -g -fno-asynchronous-unwind-tables -Wall \
>       -target bpf -c 02/pkt_counter.c -o 02/pkt_counter.o
```

```
$ sudo ./runas buzz +bpf bpftool prog load 02/pkt_counter.o /sys/fs/bpf/pkt_counter
libbpf: Error in bpf_object__probe_loading():ERROR: strerror_r(524)=22(524). Couldn't
load trivial BPF program. Make sure your kernel supports BPF (CONFIG_BPF_SYSCALL=y)
and/or that RLIMIT_MEMLOCK is set to big enough value.
libbpf: failed to load object '02/pkt_counter.o'
Error: failed to load object file
```

it's not memlock, we're on v5.13

CLOUDFLARE

Trace it, load it

```
$ sudo ./runas buzz +bpf \
> strace -o /tmp/prog_load.strace -e bpf \
> bpftool prog load 02/pkt_counter.o /sys/fs/bpf/pkt_counter
```

```
$ cat /tmp/prog_load.strace
bpf(BPF_PROG_LOAD, {prog_type=BPF_PROG_TYPE_SOCKET_FILTER, ...}, 120)
  = -1 ENOTSUPP (Unknown error 524)
+++ exited with 255 +++
```

CLOUDFLARE

Trace it harder, load it

```
$ sudo perf ftrace -G '*sys_bpf' -g '*irq*' --graph-opts depth=6 \
> ./runas buzz +bpf \
> bpftool prog load 02/pkt_counter.o /sys/fs/bpf/pkt_counter \
> > /tmp/prog_load.ftrace
```

CLOUDFLARE

Trace it harder - ftrace output

```
0)               |  __x64_sys_bpf() {
0)               |    bpf_prog_load() {
                          ...
0)               |      bpf_prog_select_runtime() {
0)   0.137 us    |        bpf_prog_alloc_jited_linfo();
0)               |        bpf_int_jit_compile() {
                            ...
0)               |          bpf_jit_binary_alloc() {
0)   0.350 us    |            bpf_jit_charge_modmem();
0)   0.673 us    |          }
                            ...
0)   0.492 us    |          kfree();
0)   6.658 us    |        }
0)               |        bpf_prog_jit_attempt_done() {
0)   0.333 us    |        }
0)   7.736 us    |      }
                       ...
```

failed?

CLOUDFLARE

`bpf_jit_limit` - limit check for BPF JIT memory

```
int bpf_jit_charge_modmem(u32 pages)
{
    if (atomic_long_add_return(pages, &bpf_jit_current) >
        (bpf_jit_limit >> PAGE_SHIFT)) {
        if (!capable(CAP_SYS_ADMIN)) {
            atomic_long_sub(pages, &bpf_jit_current);
            return -EPERM;
        }
    }

    return 0;
}
```

CLOUDFLARE

Are we over `bpf_jit_limit`?

```
$ sudo sysctl net.core.bpf_jit_limit
net.core.bpf_jit_limit = 1048576
```

1 MiB, set just for test (default 252 MiB)

```
$ sudo bpftrace -e '
> BEGIN {
>   printf("%ld", *kaddr("bpf_jit_current") << 12);
>   exit();
> }'
Attaching 1 probe...
1294336
```

> 1 MiB

Why are we over the limit?

```
$ for ((i = 0; i < 100; i++)); do
>   sudo iptables -A OUTPUT -m bpf --bytecode '4,48 0 0 9,21 0 1 6,6 0 0 1,6 0 0 0' -j ACCEPT
> done
```

```
$ sudo iptables -vnL OUTPUT | head -10
Chain OUTPUT (policy ACCEPT 6 packets, 456 bytes)
 pkts bytes target     prot opt in     out     source               destination
  260 23280 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            match bpf 48 0 0 9,21 0 1 6,6 0 0 1,6 0 0 0
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            match bpf 48 0 0 9,21 0 1 6,6 0 0 1,6 0 0 0
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            match bpf 48 0 0 9,21 0 1 6,6 0 0 1,6 0 0 0
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            match bpf 48 0 0 9,21 0 1 6,6 0 0 1,6 0 0 0
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            match bpf 48 0 0 9,21 0 1 6,6 0 0 1,6 0 0 0
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            match bpf 48 0 0 9,21 0 1 6,6 0 0 1,6 0 0 0
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            match bpf 48 0 0 9,21 0 1 6,6 0 0 1,6 0 0 0
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0            match bpf 48 0 0 9,21 0 1 6,6 0 0 1,6 0 0 0
```

CLOUDFLARE

Why are we hitting the limit?

All BPF programs are charged:

- progs loaded with `bpf(BPF_PROG_LOAD)`
- `seccomp` programs (auto cBPF to BPF translation)
- iptables `xt_bpf` programs (auto cBPF to BPF translation)

Limit is **not** per-netns! **`bpf_jit_limit`** is global.

[bpf] bpf_jit_limit close shave

CLOUDFLARE

(3) Creating a BPF link

CLOUDFLARE

Attach it, but not the “old” way

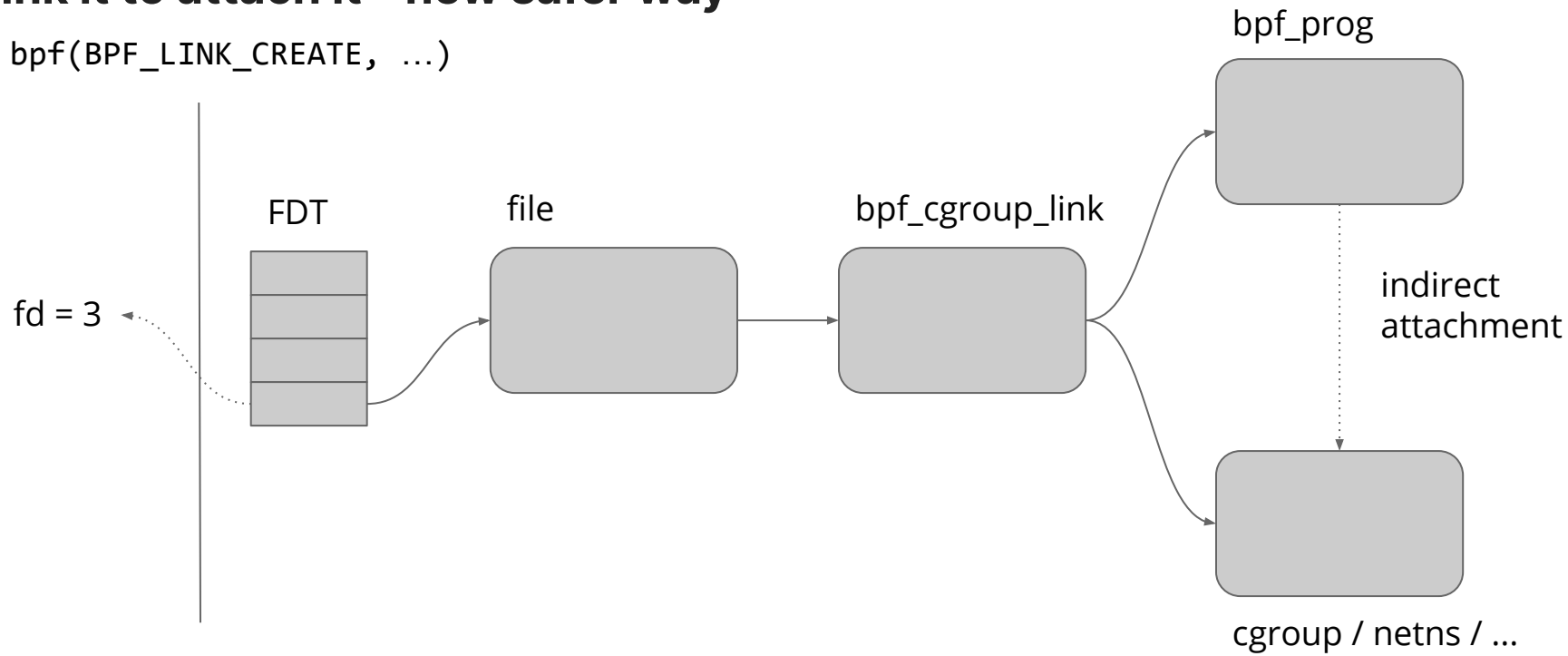
```
$ sudo bpftool prog load ./pkt_counter.o /sys/fs/bpf/pkt_counter
```

```
$ sudo strace -e bpf \
> bpftool cgroup attach /sys/fs/cgroup/test.slice/ ingress pinned /sys/fs/bpf/pkt_counter
bpf(BPF_OBJ_GET, {pathname="/sys/fs/bpf/pkt_counter", bpf_fd=0, file_flags=0}, 120) = 4
bpf(BPF_PROG_ATTACH, {target_fd=3, attach_bpf_fd=4, attach_type=BPF_CGROUP_INET_INGRESS,
attach_flags=0, replace_bpf_fd=0}, 120) = 0
```

discouraged and unsupported by newer prog types

Link it to attach it - new safer way

```
bpf(BPF_LINK_CREATE, ...)
```



link FD closed ⇒ (1) program detached, (2) program destroyed

CLOUDFLARE

buzz needs a tool to create a BPF link

```
/* (1) Open a cgroup FD */
cg_fd = open("/sys/fs/cgroup/…", O_DIRECTORY | O_RDONLY);

/* (2) Create BPF map, load BPF program using BPF skeleton */
obj = pkt_counter__open_and_load();

/* (3) Attach BPF program to cgroup */
link = bpf_program__attach_cgroup(obj->progs.prog, cg_fd);

/* (4) Pin BPF link */
err = bpf_link__pin(link, "/sys/fs/bpf/…");
```

bpftool *can't do this yet*

CLOUDFLARE

(4) Pinning BPF link

CLOUDFLARE

buzz wants to pin a link. But where?

```
buzz@bullseye:~$ ls -ld /sys/fs/bpf
drwx-----T 2 root root 0 Sep 12 10:02 /sys/fs/bpf
```

only root has access

systemd/src/core/mount-setup.c:

```
static const MountPoint mount_table[] = {
    …
    { "bpf", "/sys/fs/bpf", "bpf", "mode=700", MS_NOSUID|MS_NOEXEC|MS_NODEV, …
    …
};
```

[systemd] bpf: mount bpffs by default on boot
[systemd] mount-setup: change bpf mount mode to 0700

CLOUDFLARE

Best practice - dedicated BPF fs instance

```
$ sudo mkdir -p /run/mount/bpf/pkt_counter
$ sudo mount -t bpf -o uid=buzz,gid=buzz,mode=0700 none /run/mount/bpf/pkt_counter
$ ls -ld /run/mount/bpf/pkt_counter
drwx-----T. 2 root root 0 Sep 13 09:12 /run/mount/bpf/pkt_counter
```

bpf file system doesn't understand **uid=** *and* **gid=** *options*

```
$ sudo chown buzz.buzz /run/mount/bpf/pkt_counter
```

..., but systemd **.mount** *units don't support* **User=** *or* **Group=** *options*

[bpf] Can we share /sys/fs/bpf like /tmp?

CLOUDFLARE

(5) Inspecting the attached program

CLOUDFLARE

buzz wants to examine the attached prog

Why would want to do that?

- ❏ find which maps prog is using
- ❏ read prog tag

```
$ sudo ./runas buzz +bpf bpftool link show pinned /run/mount/bpf/pkt_counter/link
23: cgroup  prog 256
        cgroup_id 4966  attach_type ingress
```

```
$ sudo ./runas buzz +bpf bpftool prog show id 256
Error: get by id (256): Operation not permitted
```

CLOUDFLARE

What is stopping us?

```
$ strace -e bpf bpftool prog show id 256
…
bpf(BPF_PROG_GET_FD_BY_ID, {prog_id=256, next_id=0, open_flags=0}, 120) = -1 EPERM
(Operation not permitted)
Error: get by id (256): Operation not permitted
+++ exited with 255 +++
```

```
static int bpf_prog_get_fd_by_id(const union bpf_attr *attr)
{
    …
    if (!capable(CAP_SYS_ADMIN)) // 👈
        return -EPERM;
    …
}
```

CLOUDFLARE

Can’t get from link FD to prog

```
$ sudo bpftool prog show id 256
256: cgroup_skb  name count_pkts  tag 42341c82f6736baa  gpl
        loaded_at 2021-09-09T10:12:59+0000  uid 1001 ⇦ this is us 🐝
        xlated 160B  jited 95B  memlock 4096B  map_ids 103
        btf_id 271
```

kernel knows buzz *created the program!*

CLOUDFLARE

Workaround? Pin links, progs, and maps

```
$ ls -l /run/mount/bpf/pkt_counter
total 0
-rw-------. 1 buzz buzz 0 Sep 22 16:28 link
-r--------. 1 root root 0 Sep 22 10:59 maps.debug
-rw-------. 1 buzz buzz 0 Sep 22 16:28 prog
-r--------. 1 root root 0 Sep 22 10:59 progs.debug
-rw-------. 1 buzz buzz 0 Sep 22 16:28 stats
```

CLOUDFLARE

(6) Reading from a BPF map in user-space

CLOUDFLARE

buzz wants a metrics scraper read BPF map

```
$ chmod o+r /run/mount/bpf/pkt_counter/stats
$ ls -l /run/mount/bpf/pkt_counter/stats
-rw----r--. 1 buzz buzz 0 Sep 15 16:51 /run/mount/bpf/pkt_counter/stats
```

give read access to anyone

```
$ chmod o+x /run/mount/bpf/pkt_counter
$ ls -ld /run/mount/bpf/pkt_counter
drwx-----t. 2 buzz buzz 0 Sep 15 16:51 /run/mount/bpf/pkt_counter
```

make bpf fs instance accessible by anyone

CLOUDFLARE

Can the scraper read it?

```
$ sudo ./runas scraper +bpf bpftool map dump pinned /run/mount/bpf/pkt_counter/stats
Error: bpf obj get (/run/mount/bpf/pkt_counter): Permission denied
```

what is failing?

```
$ sudo ./runas scraper +bpf strace -e bpf \
> bpftool map dump pinned /run/mount/bpf/pkt_counter/stats
bpf(BPF_OBJ_GET, {pathname="/run/mount/bpf/pkt_counter/stats", bpf_fd=0,
file_flags=0}, 120) = -1 EACCES (Permission denied)
Error: bpf obj get (/run/mount/bpf/pkt_counter): Permission denied
```

CLOUDFLARE

Where are we failing?

```
$ sudo perf ftrace -G '*sys_bpf' -g '*irq*' --graph-opts depth=3 \
> ./runas scraper +bpf bpftool map dump pinned …
 0)               |  __x64_sys_bpf() {
…
 0)               |    bpf_obj_get_user() {
 0)   0.505 us    |      bpf_get_file_flag();
 0)               |      user_path_at_empty() {
 0) ! 259.191 us  |      }
 0)   4.750 us    |      inode_permission();
 0)               |      path_put() {
 0) ! 102.123 us  |      }
 0) ! 429.527 us  |    }
 0) ! 493.568 us  |  }
```

suspect?

CLOUDFLARE

Where are we failing?

```
static void *bpf_obj_do_get(const struct filename *pathname,
                            enum bpf_type *type, int flags)
{
    …
    inode = d_backing_inode(path.dentry);
    ret = inode_permission(inode, ACC_MODE(flags));
    if (ret)
        goto out;
    …
}
```

flags come from `attr->file_flags`

passed to `BPF_OBJ_GET`

CLOUDFLARE

What is flags set to?

```
int bpf_get_file_flag(int flags)
{
    if ((flags & BPF_F_RDONLY) && (flags & BPF_F_WRONLY))
        return -EINVAL;
    if (flags & BPF_F_RDONLY)
        return O_RDONLY;
    if (flags & BPF_F_WRONLY)
        return O_WRONLY;
    return O_RDWR;
}
```

CLOUDFLARE

`bpftool map dump` requests RW access

```
$ strace -e bpf bpftool map dump pinned /run/mount/bpf/pkt_counter/stats
bpf(BPF_OBJ_GET, {pathname="/run/mount/bpf/pkt_counter/stats", bpf_fd=0,
file_flags=0}, 120) = -1 EACCES (Permission denied)
Error: bpf obj get (/run/mount/bpf/pkt_counter): Permission denied
+++ exited with 255 +++
```

Need to write a custom dumper.

Easy?

CLOUDFLARE

Dumping a read-only BPF map

libbpf doesn't support setting `file_flags`

must call `bpf()` *syscall directly*

```
union bpf_attr attr = {};

attr.pathname = (uintptr_t)map_path;
attr.file_flags = BPF_F_RDONLY;
map_fd = syscall(SYS_bpf, BPF_OBJ_GET, &attr, sizeof(attr));
if (map_fd < 0)
    error(EXIT_FAILURE, errno, "BPF_OBJ_GET");
```

CLOUDFLARE

After making panda sad 9 times...

```
$ sudo systemd-run --unit pinger --uid=buzz --gid=buzz ping 1.1.1.1
Running as unit: pinger.service


$ sudo ./runas buzz +bpf,+net_admin 05/attach-pkt-counter \
> /sys/fs/cgroup/system.slice/pinger.service /run/mount/bpf/pkt_counter
Pinned map at /run/mount/bpf/pkt_counter/stats
Pinned prog at /run/mount/bpf/pkt_counter/prog
Pinned link at /run/mount/bpf/pkt_counter/link
Attached to cgroup /sys/fs/cgroup/system.slice/pinger.service


$ sudo ./runas scraper +bpf 06/map-dump /run/mount/bpf/pkt_counter/stats
111
$ sudo ./runas scraper +bpf 06/map-dump /run/mount/bpf/pkt_counter/stats
112
```

CLOUDFLARE

Why is panda sad today?

- distros disable unprivileged BPF by default
- per-user locked memory limit can make `BPF_MAP_CREATE` fail (LTS kernel only)
- kernel global JIT memory limit can make `BPF_PROG_LOAD` fail
- `bpftool` can't create BPF links
- systemd limits `/sys/fs/bpf` access to root only
- can't set default owner when mounting `bpf` file system
- can't query programs loaded by the user by prog ID
- `bpftool map dump` requests read+write access by default
- libbpf `BPF_OBJ_GET` wrapper doesn't support `file_flags`

Links

Code:
https://github.com/jsitnicki/lpc-2021-bpf-ux-bench

Email: jakub@cloudflare.com
Twitter: @jkbs0

Image licenses

Image licensed under CC0

Sad Panda image from Hana Emojis Panda Edition Collection licensed under CC BY 3.0

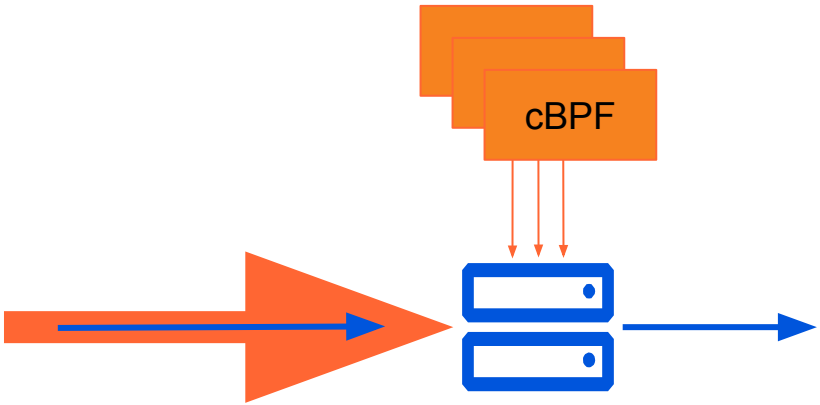
CLOUDFLARE

Autogenerated BPF

CLOUDFLARE

DDOS

- Drop DDOS packets
- Mitigation system generates classic BPF
  - What libpcap / tcpdump uses
  - Matching packets should be dropped



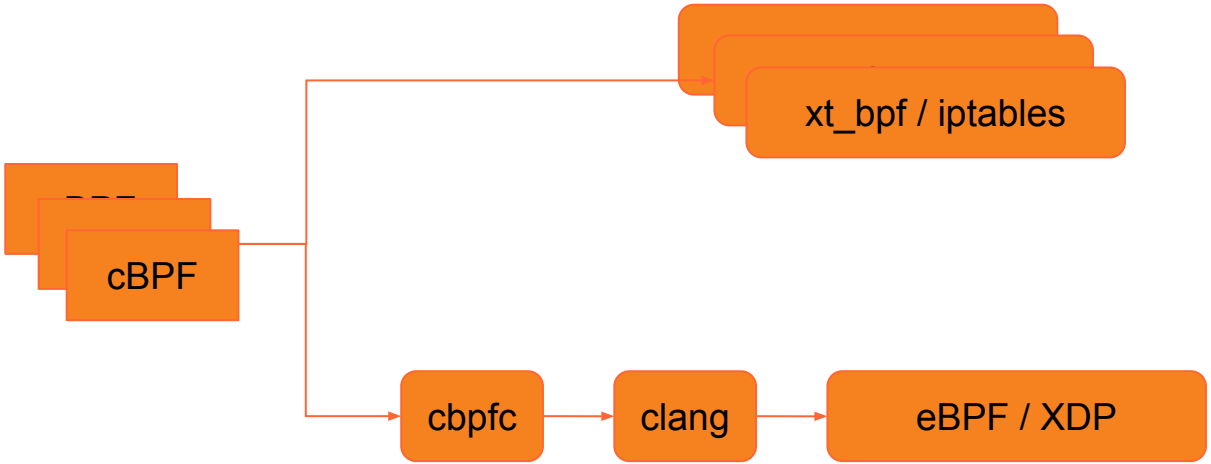
CLOUDFLARE

cBPF

- Very flexible
- Sources:
  - libpcap
  - Custom generators
  - Handwritten
- Run as:
  - iptables with xt_bpf
  - XDP with cbpfc
    - Compile cBPF to C
    - C to eBPF with clang

CLOUDFLARE

cBPF



CLOUDFLARE

Failures

- Rejected by verifier
  - Guard packet accesses:

```
if (data + x > data_end) return 0;
```

  - Check division by 0:

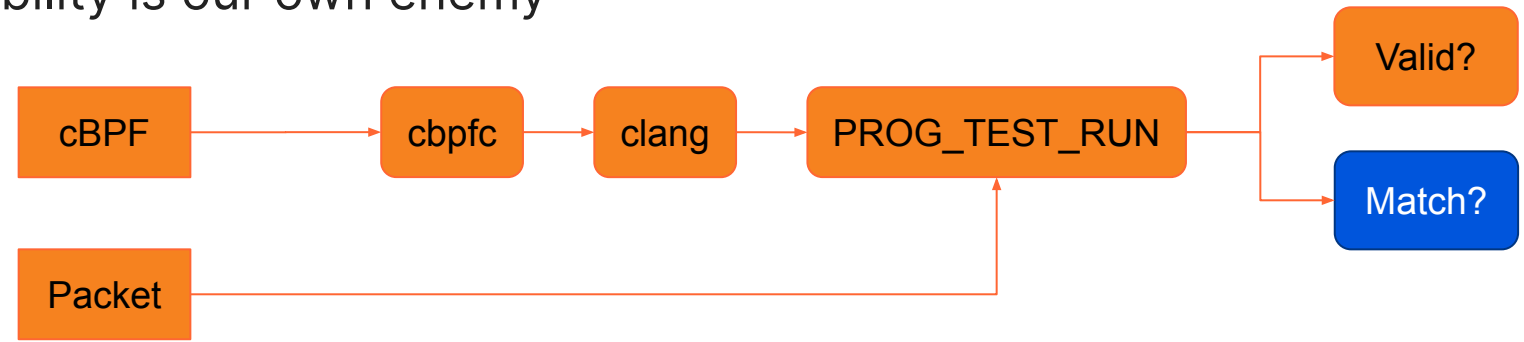
```
if (x == 0) return 0;
```

  - Hardest part
  - Blocks DDOS mitigations
- Incorrect behavior
  - Drop wrong packets
  - Hard to notice

CLOUDFLARE

Testing

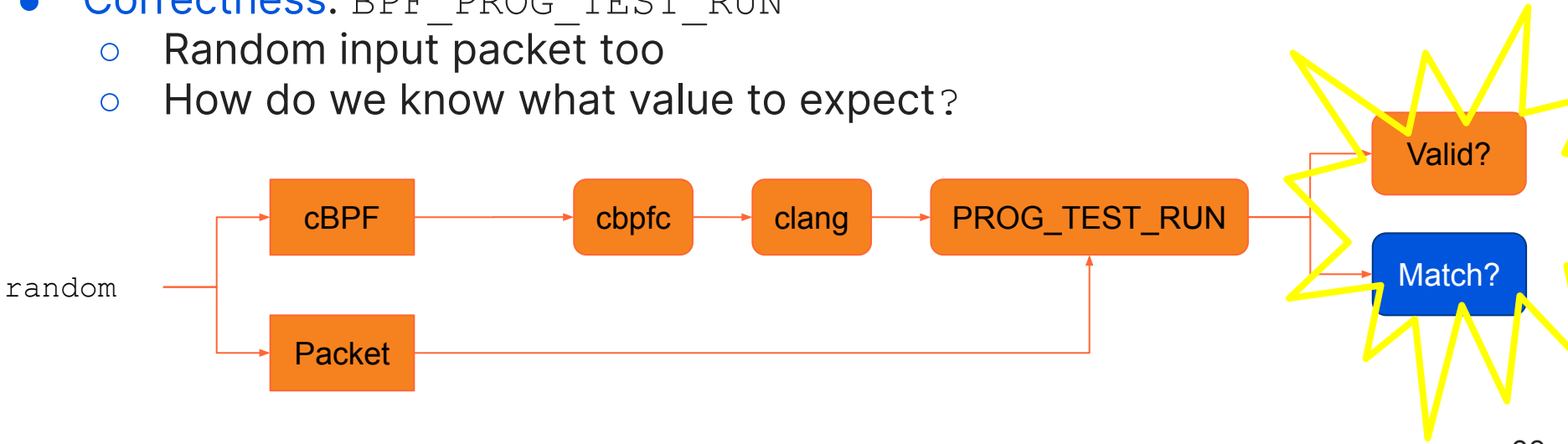
- Unit tests to cover known rules
- Verifier: `BPF_PROG_LOAD`
- Correctness: `BPF_PROG_TEST_RUN`
  - Specially crafted packets as input
- Can only cover so much
  - Flexibility is our own enemy



CLOUDFLARE

Testing Unknowns

- Random cBPF instructions
- Verifier: `BPF_PROG_LOAD`
  - How do we know if cBPF is valid?
- Correctness: `BPF_PROG_TEST_RUN`
  - Random input packet too
  - How do we know what value to expect?



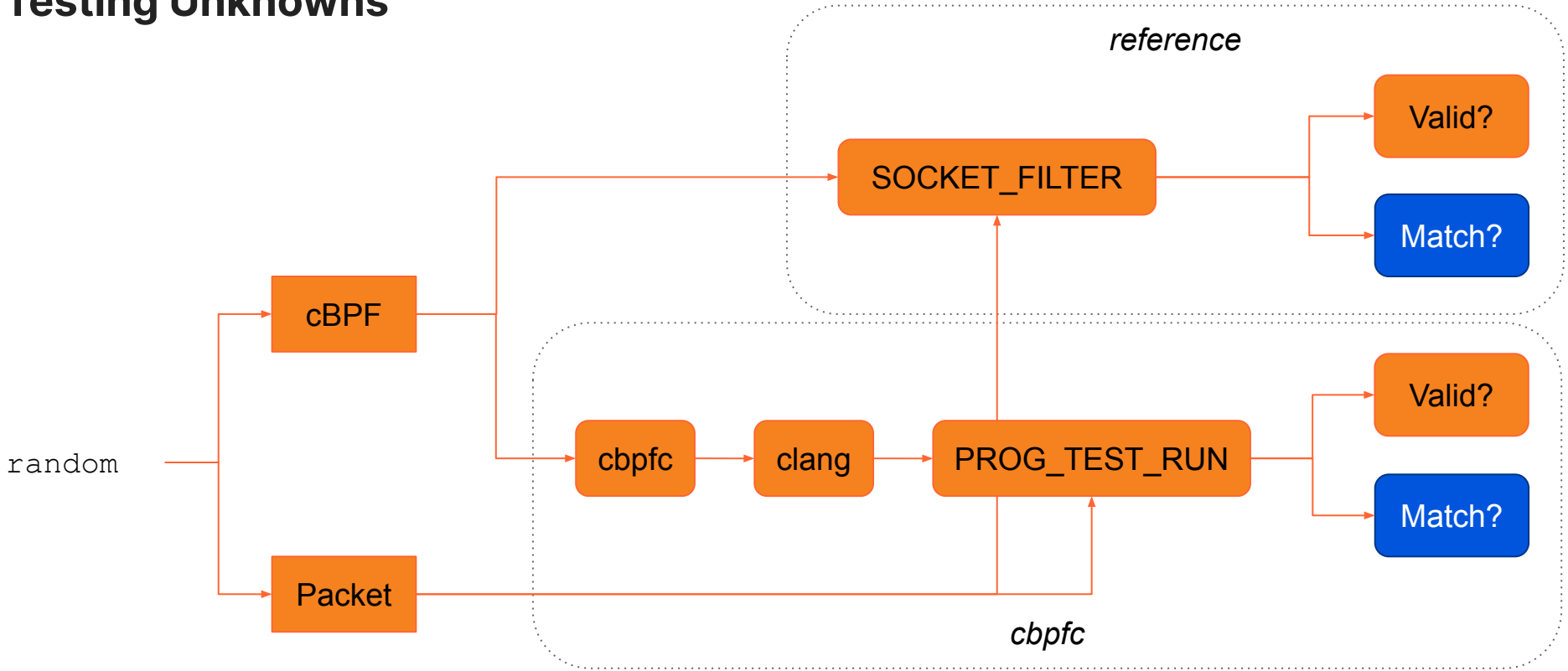
CLOUDFLARE

Testing Unknowns

- Compare to reference implementation!
  - Attach cBPF to unix socket as socket filter
- Verifier: `BPF_PROG_LOAD`
  - Same result as attaching socket filter
- Correctness: `BPF_PROG_TEST_RUN`
  - Run packet through unix socket:
    - Packet dropped: no match
    - Packet passed: match

CLOUDFLARE

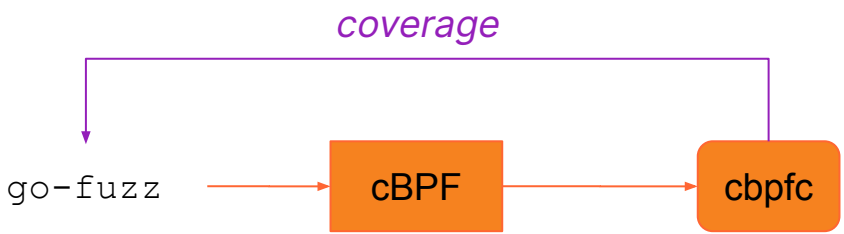
Testing Unknowns



CLOUDFLARE

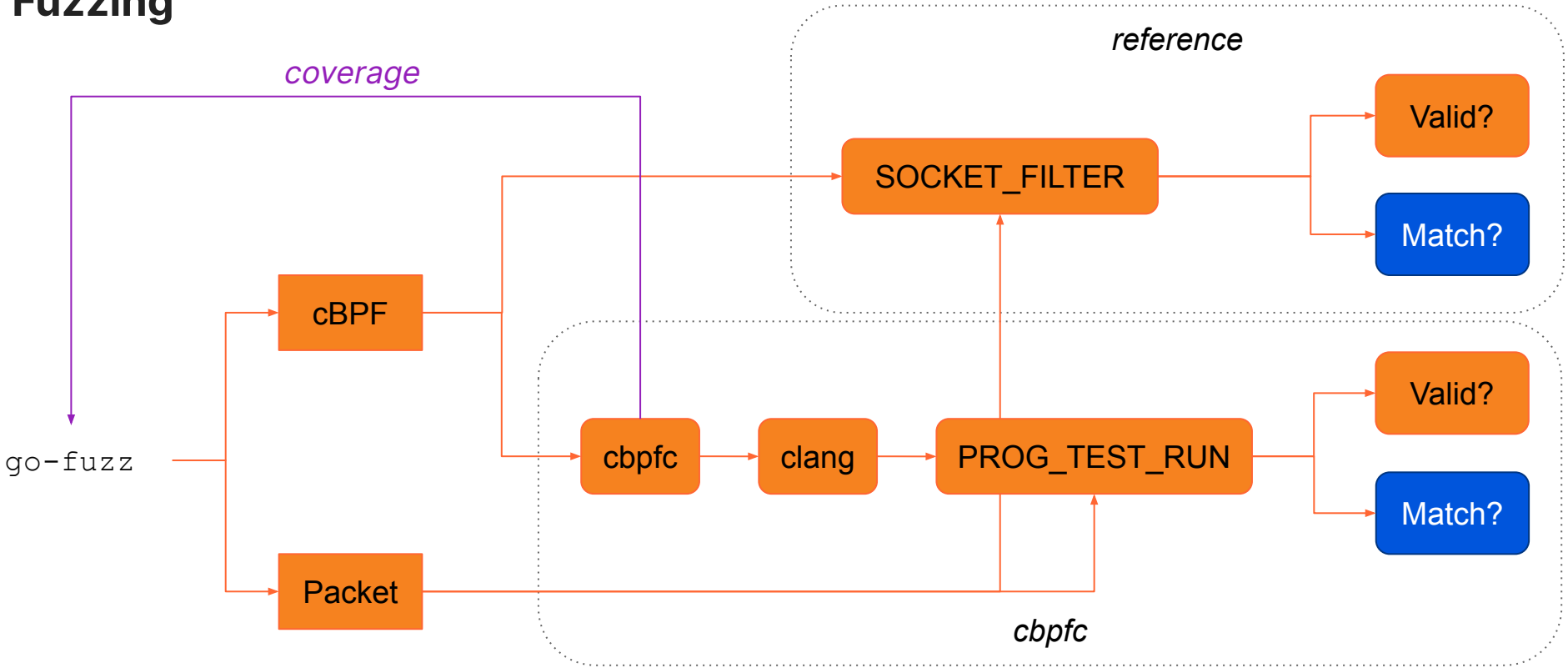
Fuzzing

- One step away from fuzzing!
- go-fuzz



CLOUDFLARE

Fuzzing



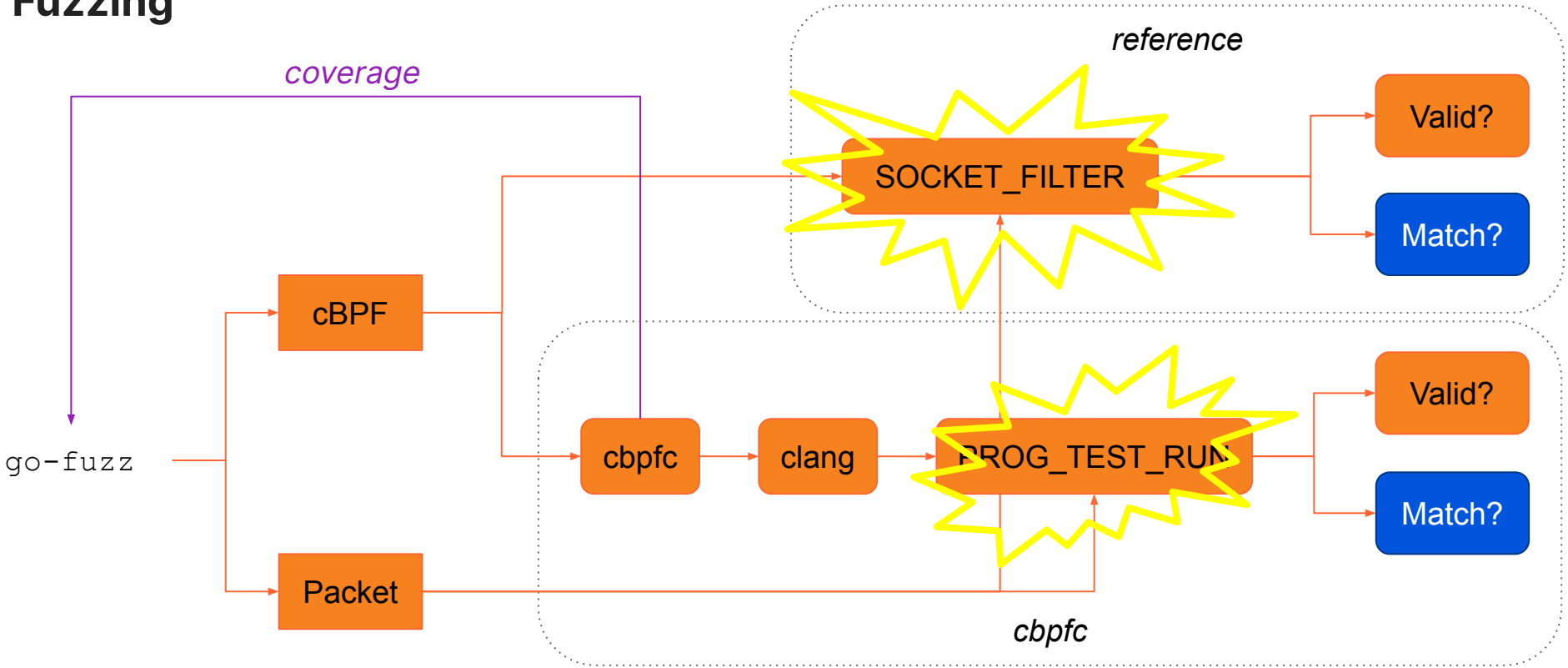
CLOUDFLARE

Fuzzing

- Slow
- Time to first bug: ~ days

CLOUDFLARE

Fuzzing



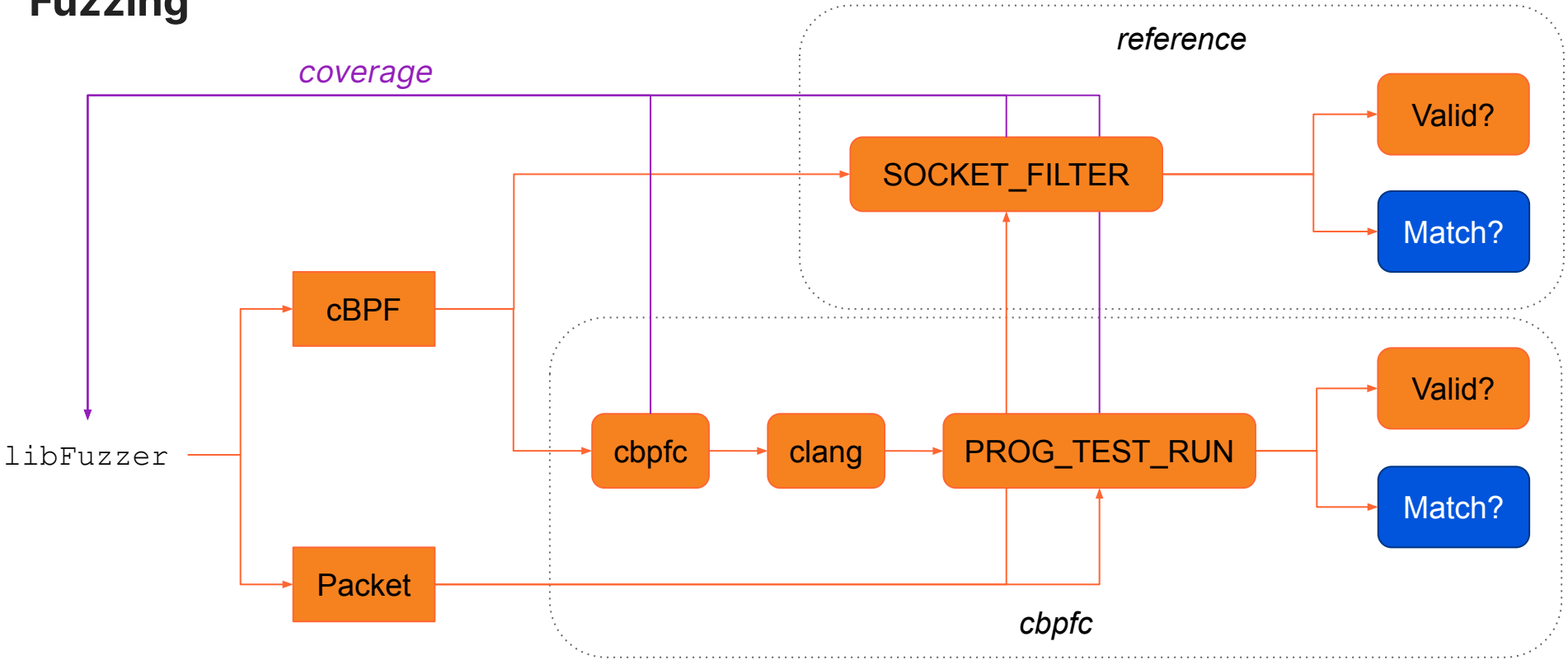
CLOUDFLARE

KCOV

- Collect kernel code coverage
  - https://www.kernel.org/doc/html/v5.0/dev-tools/kcov.html
- Can read covered PCs from userspace
  - `/sys/kernel/debug/kcov`
- Powers syzkaller
- Replace go-fuzz with libFuzzer

CLOUDFLARE

Fuzzing



CLOUDFLARE

Fuzzing

- libFuzzer with userspace and kernel coverage
- Time to first bug: ~ minutes

- Trophies:
  - Missing packet bounds check
  - Missing overflow check
    - Packet offsets need to be `< 2^16`

CLOUDFLARE

Clang Optimizations: Packet Bounds

- Challenge: read last byte of packet!

```
uint8_t *data = (uint8_t *)(long)ctx->data;
uint8_t *data_end = (uint8_t *)(long)ctx->data_end;

size_t size = data_end - data;

if (data + size > data_end) {
    return 0;
}

uint8_t last = *(data + size - 1);
return last;
```

CLOUDFLARE

Clang Optimizations: Packet Bounds

- Challenge: read last byte of packet!

```
0: (b4) w0 = 0
; uint8_t *data_end = (uint8_t *)(long)ctx->data_end;
1: (61) r1 = *(u32 *)(r1 +4)
; if (data + size > data_end) {
2: (2d) if r1 > r1 goto pc+1
R0_w=inv0 R1_w=pkt_end(id=0,off=0,imm=0) R10=fp0
; uint8_t last = *(data + size - 1);
3: (71) r0 = *(u8 *)(r1 -1)
R1 invalid mem access 'pkt_end'
```

CLOUDFLARE

Clang Optimizations: Packet Bounds

- Not clear how to fix this
- Magic annotation so clang doesn’t optimize bounds checks?
  - But optimizations are useful sometimes

CLOUDFLARE

Register Mirroring

```
25: (bf) r4 = r3
R2=pkt(...) R3=inv(id=3) R4_w=inv(id=3)
26: (67) r4 <<= 32
27: (77) r4 >>= 32
R2=pkt(...) R3=inv(id=3) R4_w=inv(id=0,umax_value=4294967295)
; if (x >= 65467 || data + x + 68 > data_end) return 0;
29: (25) if r4 > 0xffba goto pc+15
R2=pkt(...) R3=inv(id=3) R4_w=inv(id=0,umax_value=65466)
; if (x >= 65467 || data + x + 68 > data_end) return 0;
30: (67) r3 <<= 32
31: (77) r3 >>= 32
R2=pkt(...) R3_w=inv(id=0,umax_value=4294967295)
R4_w=inv(id=0,umax_value=65466)
```

CLOUDFLARE

Register Mirroring

- Clang is “correct”
- Should verifier figure out regs are the same?
  - Seems hard
- Explicit casts / using uint64 directly doesn’t help

CLOUDFLARE

Direct eBPF

- Is a world without clang better?
- Generate eBPF directly
  - More predictable
  - Can still link to handwritten C

CLOUDFLARE

Invalid Values

```
R2=pkt(...)
5:   (71) r5 = *(u8 *)(r2 +14)
7:   (b4) r4 = 512
R2=pkt(...) R4=inv512 R5=inv(umax_value=255)
8:   (2d) if r4 > r5 goto pc+8
R2=pkt(...) R4=inv512 R5=inv(umin_value=512,umax_value=255)
11: (0f) r2 += r5
R2=inv(id=0) R5=invP(umin_value=512,umax_value=255)
15: (61) r4 = *(u32 *)(r2 +14)
R7 invalid mem access 'inv'
```

CLOUDFLARE

Invalid Values

- Code is nonsensical
  - Doesn’t happen with clang, only direct to eBPF
- But legal?
  - Other languages would allow this
- Fix verifier to allow this?
- Or deal with it in userspace?

CLOUDFLARE

Fuzzing Challenges

- Hard to suppress errors
  - Verifier output is always different
  - Too many errors to keep fuzzing C
    - Suppress based on PCs?
- How well does KCOV work for JIT vs interpreter?

CLOUDFLARE

Info

- clang-13
- Kernel 5.10.64
- Test cases: https://github.com/arthurfabre/lpc-2021
- https://github.com/cloudflare/cbpfc
  - Fuzzing PR

CLOUDFLARE

Thank you