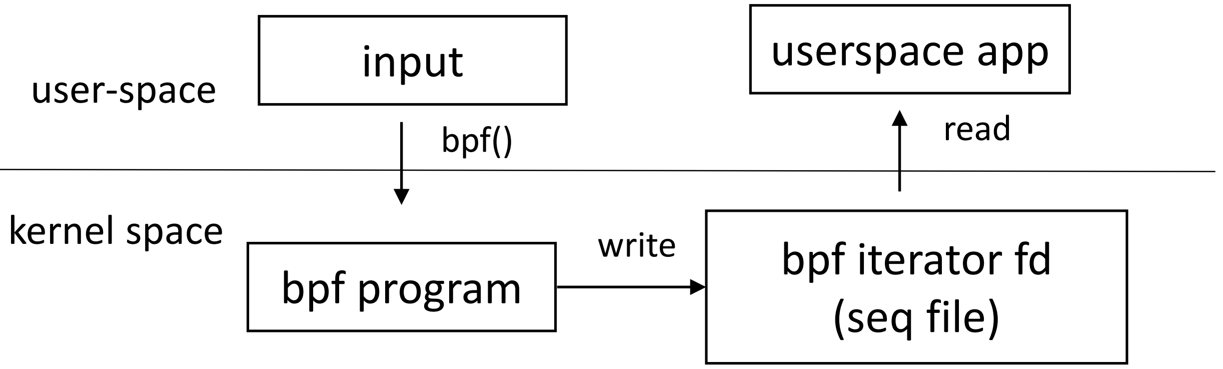
BPF file-system iterator

Agenda

- Introduction
- Use case & Demo
- Problems

Introduction (1)

A BPF iterator is a type of BPF program that allows users to iterate over specific types of kernel objects ...... allow users to define callbacks that should be executed for every entry in a variety of kernel data structures [0]



Introduction (2)

- BPF task-file iterator example

```
SEC("iter/task_file")
int dump_task_file(struct bpf_iter__task_file *ctx)
{
struct seq_file *seq = ctx->meta->seq;
struct task_struct *task = ctx->task;
struct file *file = ctx->file;
__u32 fd = ctx->fd;
/* ...... */
if (ctx->meta->seq_num == 0)
   BPF_SEQ_PRINTF(seq, "    tgid      gid       fd      file\n");
BPF_SEQ_PRINTF(seq, "%8d %8d %8d %ps\n",  task->tgid, task->pid,
                              fd, file->f_op);
}
```

```
skel = bpf_iter_task_file__open_and_load();
linfo.task.tid = getpid();
opts.link_info = &linfo;
opts.link_info_len = sizeof(linfo);
link = bpf_program__attach_iter(prog, &opts);

iter_fd = bpf_iter_create(bpf_link__fd(link));
read(iter_fd, buf, sizeof(buf));
puts(buf);
```

```
tgid      gid       fd      file
  12256    12256       0           tty_fops
  12256    12256       1           tty_fops
  12256    12256       2           tty_fops
  12256    12256       3   ns_file_operations
  12256    12256       4           btf_fops
  12256    12256       5       bpf_map_fops
```

BPF FS iterator (1)

- Iterate over file-system objects
  - super-block/inode/dentry/mount/address_space ?

- Use case
  - folio order in page cache
  - cachestat(): cached/dirty pages of an inode
  - mountinfo: for a specific mount
  - unlinked but pinned inode (du vs df)
  - ?

BPF FS iterator (2)

- fs_mnt

```
SEC("?iter/fs_mnt")
int dump_mnt(struct bpf_iter__fs_mnt *ctx)
{
        struct seq_file *seq = ctx->meta->seq;
        struct vfsmount *mnt = ctx->mnt;
        struct path *root = ctx->root;
        struct mount *r;

        bpf_seq_mountinfo(seq, mnt, root);
        r = bpf_rdonly_cast(container_of(mnt, struct mount, mnt),
                        bpf_core_type_id_kernel(struct mount));
        BPF_SEQ_PRINTF(seq, "id %d parent_id %d mnt_flags 0x%x\n",
                r->mnt_id, r->mnt_parent->mnt_id, r->mnt.mnt_flags);
```

```
linfo.fs.type = BPF_FS_ITER_MNT;
linfo.fs.fd = open(fpath, O_RDONLY);
opts.link_info = &linfo;
opts.link_info_len = sizeof(linfo);
link = bpf_program__attach_iter(skel->progs.dump_mnt,
&opts);
```

```
38 24 0:30 / /tmp rw,nosuid,nodev shared:15 - tmpfs tmpfs rw
id 38 parent_id 24 mnt_flags 0x1003
```

BPF FS iterator (3)

- fs_inode

```
SEC("?iter/fs_inode")
int dump_raw_inode(struct bpf_iter__fs_inode *ctx)
{
        struct seq_file *seq = ctx->meta->seq;
        struct inode *inode = ctx->inode;
        /* ...... */
        bpf_filemap_cachestat(inode, 0, ~0UL, &cs);
        BPF_SEQ_PRINTF(seq, "cache: cached %llu dirty %llu wb %llu
evicted %llu\n", cs.nr_cache, cs.nr_dirty, cs.nr_writeback,
cs.nr_evicted);
        /* TODO: handle BPF_MAX_LOOPS */
        dump.max = ((unsigned long)inode->i_size + 4095) / 4096;
        BPF_SEQ_PRINTF(seq, "orders:\n");
        bpf_loop(dump.max, dump_page_order, &dump, 0);
}
```

```
linfo.fs.type = BPF_FS_ITER_INODE;
linfo.fs.fd = open(fpath, O_WRONLY);
opts.link_info = &linfo;
opts.link_info_len = sizeof(linfo);
link = bpf_program__attach_iter(skel->progs.dump_mnt,
&opts);
```

```
sb: bsize 4096 s_op xfs_super_operations s_type xfs_fs_type name
xfs
ino: inode nlink 1 inum 131 size 10485760, name inode.test
cache: cached 2560 dirty 0 wb 0 evicted 0
orders:
        page offset 0 order 2
        page offset 4 order 2
        page offset 8 order 2
        page offset 12 order 2
        page offset 16 order 4
```

Problems (1)

- need unpriviledged bpf iterator ?
  - could open a file **!=** could retrieve implementation details of a file ?
  - create bpf iterator needs priviledge (CAP_BPF)
  - make it usable for normal user by pinned it in bpf fs
    - one step further: update the input to bpf iterator dynamically (different fd ?) ?

Problems (2)

- Don't pin the file-system ?
  - might work when iterating all inodes in a file-system
  - fs-pin: may not work for fs_mnt
    - fs-pin is only used by acct()
    - iterator: increase the refcount of the original mount to dump its internals
    - kill_pin: is called when the last refcount of mnt is freed

Questions ?