CLOUD NATIVE + OPEN SOURCE
Virtual Summit China 2020

BPF for chaos and tracing in Kubernetes

Wenbo Zhang

About the Presenter

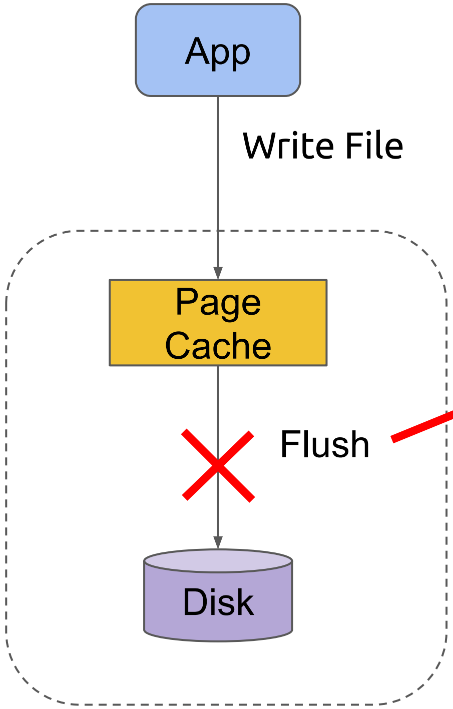
- Wenbo Zhang
  - A PingCAP Development Engineer, focusing on performance analysis and diagnosis of Linux kernel

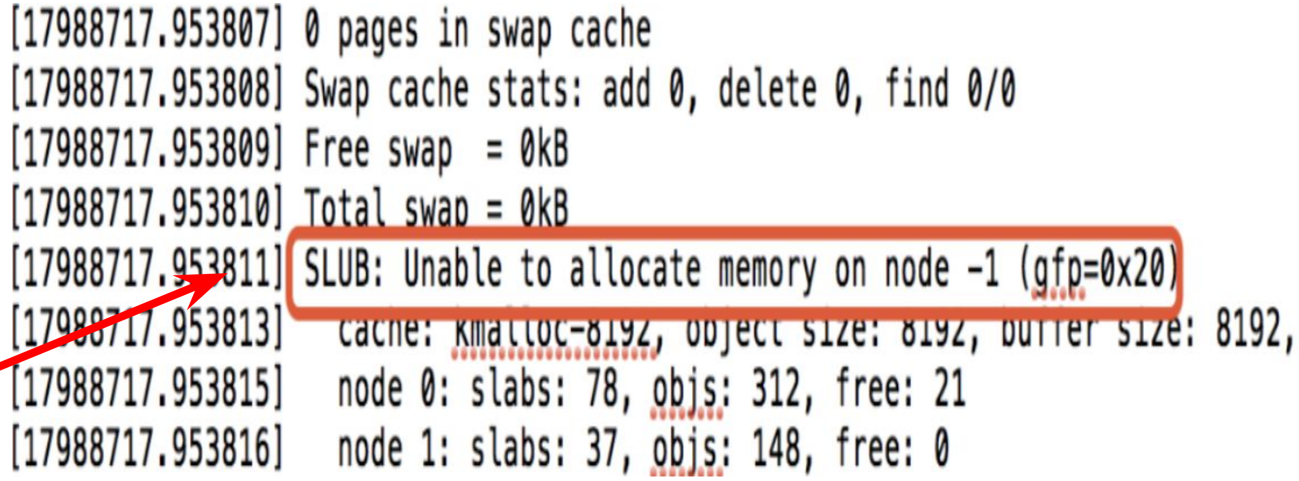
Agenda

- Why we need Chaos
- Kernel Chaos with BPF
- BPF with native support for containers
- Tracing with BPF

Why we need Chaos

- Error happens, any types, anytime, anywhere, any device





```
[17988717.953807] 0 pages in swap cache
[17988717.953808] Swap cache stats: add 0, delete 0, find 0/0
[17988717.953809] Free swap  = 0kB
[17988717.953810] Total swap = 0kB
[17988717.953811] SLUB: Unable to allocate memory on node -1 (gfp=0x20)
[17988717.953813]   cache: kmalloc-8192, object size: 8192, buffer size: 8192,
[17988717.953815]   node 0: slabs: 78, objs: 312, free: 21
[17988717.953816]   node 1: slabs: 37, objs: 148, free: 0
```

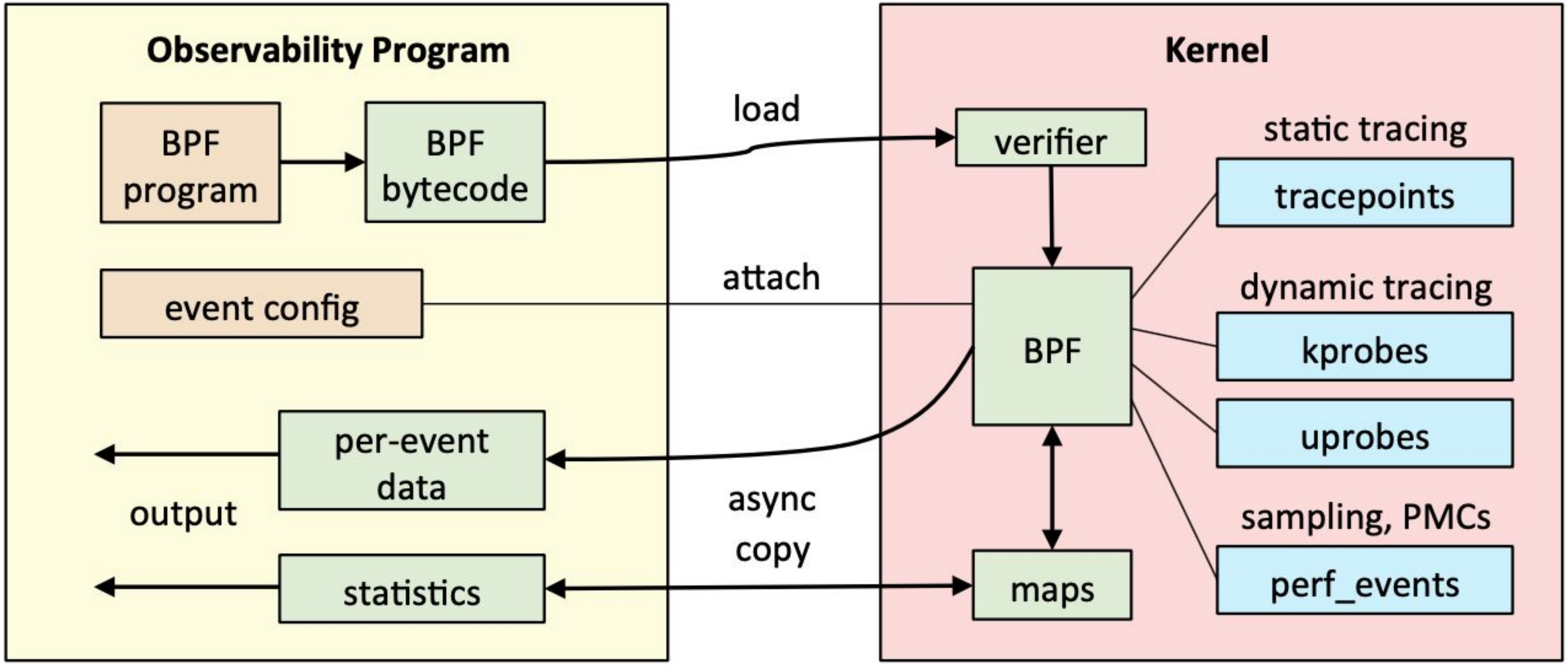
Linux kernel bugs happen in real world!!!

https://pingcap.com/blog/try-to-fix-two-linux-kernel-bugs-while-testing-tidb-operator-in-k8s/

CLOUD NATIVE + OPEN SOURCE
Virtual Summit China 2020

Kernel Chaos With BPF

- Enhanced BPF Tracing Internals

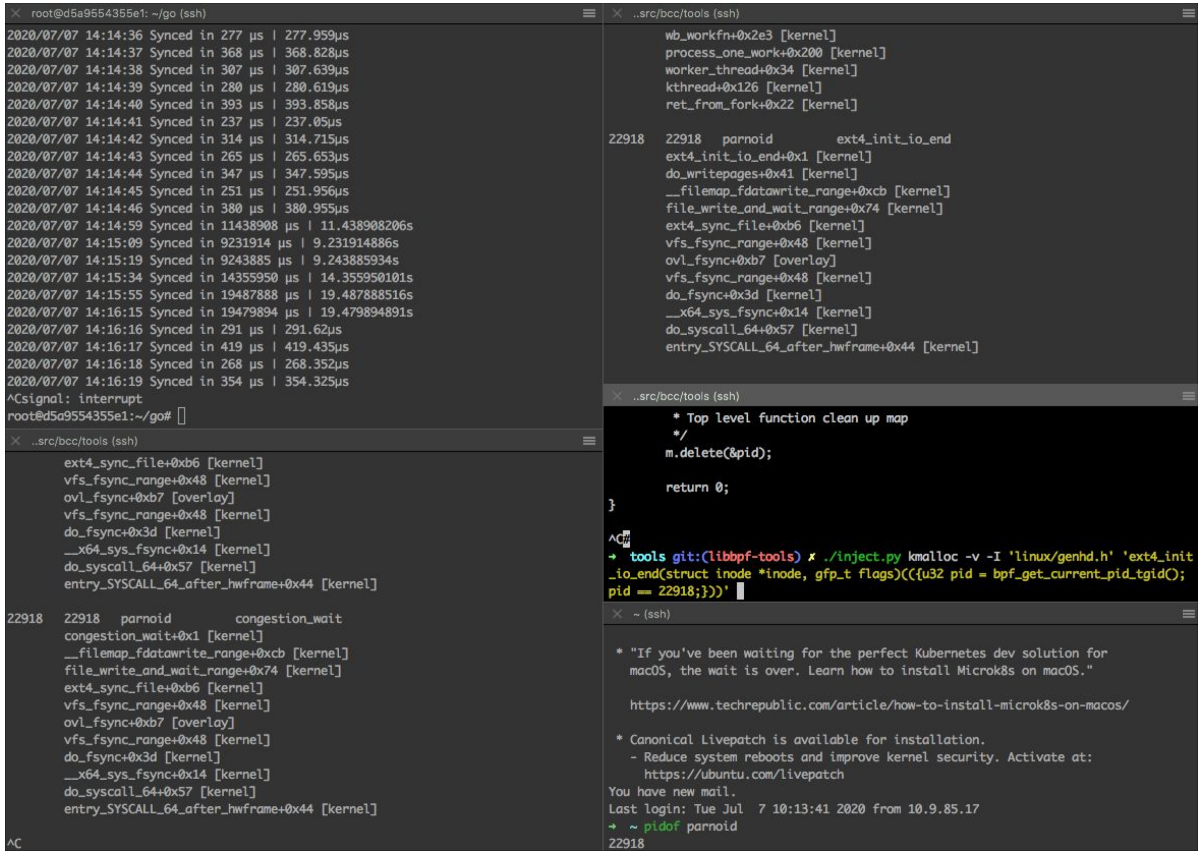


Kernel Chaos With BPF

- Disadvantages of fault injection framework
  - Just making kmalloc() fail universally is unlikely to be helpful
  - The parameters control mechanism is somewhat awkward to use and is not as flexible as one might like
- BPF override return
  - Fault injection for specific paths
  - Support precise filtering
  - Never crash the kernel
    - ALLOW_ERROR_INJECTION
    - override function should only change integer error values

Examples

- Inject congestion wait on a special task’s fsync path



```
2020/07/07 14:14:36 Synced in 277 µs | 277.959µs
2020/07/07 14:14:37 Synced in 368 µs | 368.828µs
2020/07/07 14:14:38 Synced in 307 µs | 307.639µs
2020/07/07 14:14:39 Synced in 280 µs | 280.619µs
2020/07/07 14:14:40 Synced in 393 µs | 393.858µs
2020/07/07 14:14:41 Synced in 237 µs | 237.05µs
2020/07/07 14:14:42 Synced in 314 µs | 314.715µs
2020/07/07 14:14:43 Synced in 265 µs | 265.653µs
2020/07/07 14:14:44 Synced in 347 µs | 347.595µs
2020/07/07 14:14:45 Synced in 251 µs | 251.956µs
2020/07/07 14:14:46 Synced in 380 µs | 380.955µs
2020/07/07 14:14:59 Synced in 11438908 µs | 11.438908206s
2020/07/07 14:15:09 Synced in 9231914 µs | 9.231914886s
2020/07/07 14:15:19 Synced in 9243885 µs | 9.243885934s
2020/07/07 14:15:34 Synced in 14355950 µs | 14.355950101s
2020/07/07 14:15:55 Synced in 19487888 µs | 19.487888516s
2020/07/07 14:16:15 Synced in 19479894 µs | 19.479894891s
2020/07/07 14:16:16 Synced in 291 µs | 291.62µs
2020/07/07 14:16:17 Synced in 419 µs | 419.435µs
2020/07/07 14:16:18 Synced in 268 µs | 268.352µs
2020/07/07 14:16:19 Synced in 354 µs | 354.325µs
^Csignal: interrupt
root@d5a9554355e1:~/go# 
```

```
        ext4_sync_file+0xb6 [kernel]
        vfs_fsync_range+0x48 [kernel]
        ovl_fsync+0xb7 [overlay]
        vfs_fsync_range+0x48 [kernel]
        do_fsync+0x3d [kernel]
        __x64_sys_fsync+0x14 [kernel]
        do_syscall_64+0x57 [kernel]
        entry_SYSCALL_64_after_hwframe+0x44 [kernel]

22918   22918   parnoid         congestion_wait
        congestion_wait+0x1 [kernel]
        __filemap_fdatawrite_range+0xcb [kernel]
        file_write_and_wait_range+0x74 [kernel]
        ext4_sync_file+0xb6 [kernel]
        vfs_fsync_range+0x48 [kernel]
        ovl_fsync+0xb7 [overlay]
        vfs_fsync_range+0x48 [kernel]
        do_fsync+0x3d [kernel]
        __x64_sys_fsync+0x14 [kernel]
        do_syscall_64+0x57 [kernel]
        entry_SYSCALL_64_after_hwframe+0x44 [kernel]

^C
```

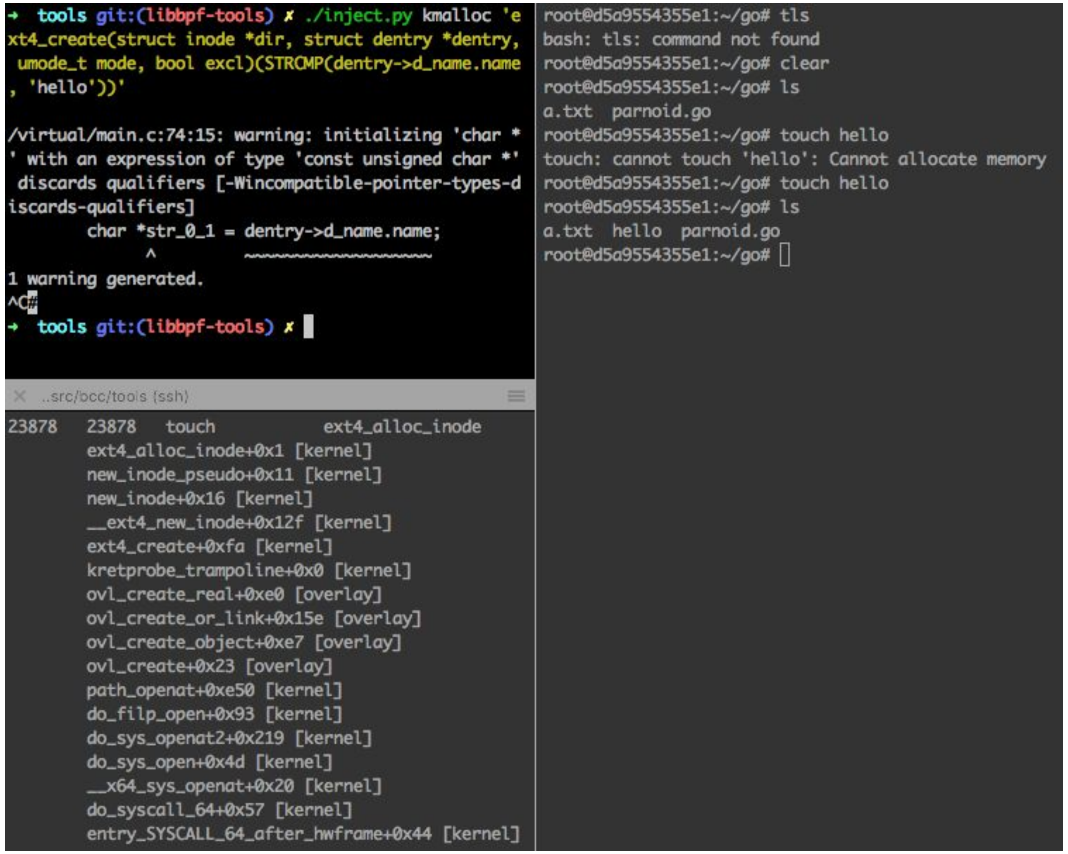
```
        wb_workfn+0x2e3 [kernel]
        process_one_work+0x200 [kernel]
        worker_thread+0x34 [kernel]
        kthread+0x126 [kernel]
        ret_from_fork+0x22 [kernel]

22918   22918   parnoid         ext4_init_io_end
        ext4_init_io_end+0x1 [kernel]
        do_writepages+0x41 [kernel]
        __filemap_fdatawrite_range+0xcb [kernel]
        file_write_and_wait_range+0x74 [kernel]
        ext4_sync_file+0xb6 [kernel]
        vfs_fsync_range+0x48 [kernel]
        ovl_fsync+0xb7 [overlay]
        vfs_fsync_range+0x48 [kernel]
        do_fsync+0x3d [kernel]
        __x64_sys_fsync+0x14 [kernel]
        do_syscall_64+0x57 [kernel]
        entry_SYSCALL_64_after_hwframe+0x44 [kernel]
```

```
         * Top level function clean up map
         */
        m.delete(&pid);

        return 0;
}

^C#
➜  tools git:(libbpf-tools) ✗ ./inject.py kmalloc -v -I 'linux/genhd.h' 'ext4_init
_io_end(struct inode *inode, gfp_t flags)(({u32 pid = bpf_get_current_pid_tgid();
pid == 22918;}))' 
```

```

 * "If you've been waiting for the perfect Kubernetes dev solution for
   macOS, the wait is over. Learn how to install Microk8s on macOS."

   https://www.techrepublic.com/article/how-to-install-microk8s-on-macos/

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch
You have new mail.
Last login: Tue Jul  7 10:13:41 2020 from 10.9.85.17
➜  ~ pidof parnoid
22918
```

Examples

- Inject alloc inode failure on a special open file path



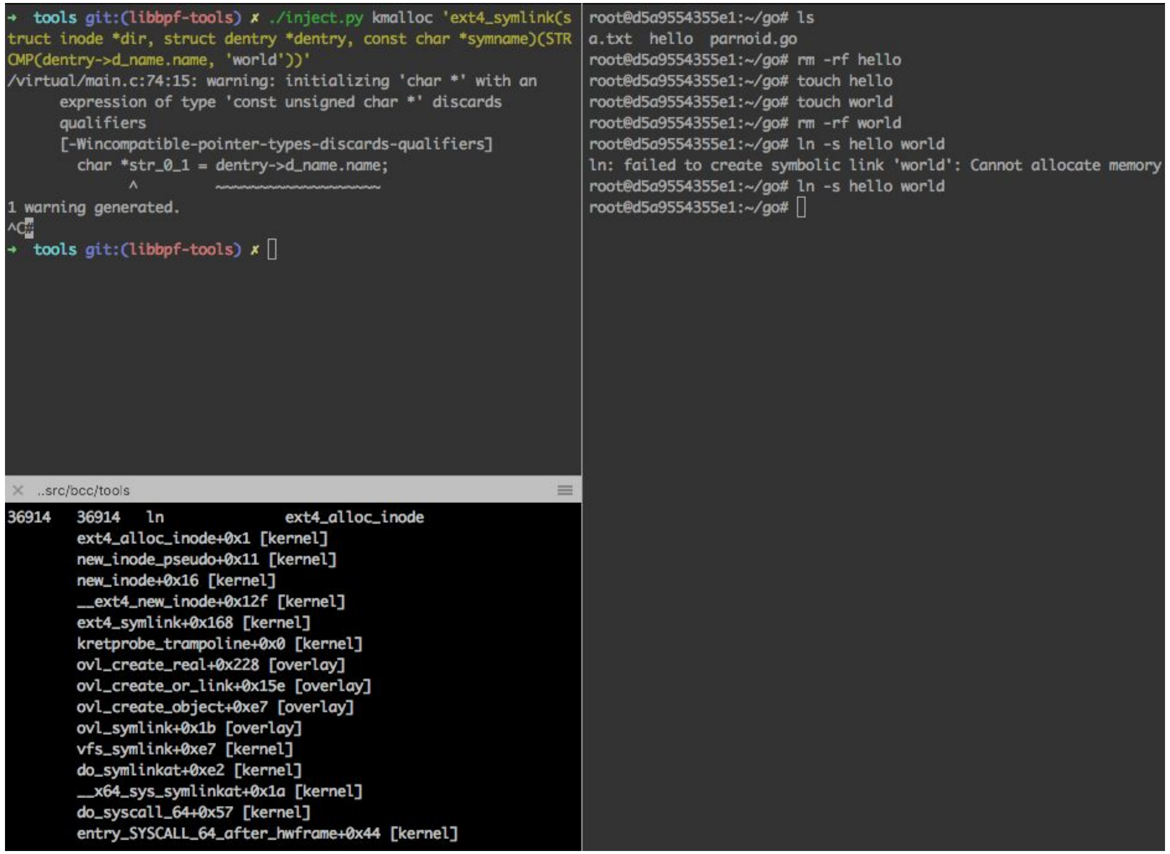
```
➜  tools git:(libbpf-tools) ✗ ./inject.py kmalloc 'e
xt4_create(struct inode *dir, struct dentry *dentry,
 umode_t mode, bool excl)(STRCMP(dentry->d_name.name
, 'hello'))'

/virtual/main.c:74:15: warning: initializing 'char *
' with an expression of type 'const unsigned char *'
 discards qualifiers [-Wincompatible-pointer-types-d
iscards-qualifiers]
        char *str_0_1 = dentry->d_name.name;
              ^         ~~~~~~~~~~~~~~~~~~~
1 warning generated.
^C#
➜  tools git:(libbpf-tools) ✗
```

```
23878   23878   touch            ext4_alloc_inode
        ext4_alloc_inode+0x1 [kernel]
        new_inode_pseudo+0x11 [kernel]
        new_inode+0x16 [kernel]
        __ext4_new_inode+0x12f [kernel]
        ext4_create+0xfa [kernel]
        kretprobe_trampoline+0x0 [kernel]
        ovl_create_real+0xe0 [overlay]
        ovl_create_or_link+0x15e [overlay]
        ovl_create_object+0xe7 [overlay]
        ovl_create+0x23 [overlay]
        path_openat+0xe50 [kernel]
        do_filp_open+0x93 [kernel]
        do_sys_openat2+0x219 [kernel]
        do_sys_open+0x4d [kernel]
        __x64_sys_openat+0x20 [kernel]
        do_syscall_64+0x57 [kernel]
        entry_SYSCALL_64_after_hwframe+0x44 [kernel]
```

```
root@d5a9554355e1:~/go# tls
bash: tls: command not found
root@d5a9554355e1:~/go# clear
root@d5a9554355e1:~/go# ls
a.txt  parnoid.go
root@d5a9554355e1:~/go# touch hello
touch: cannot touch 'hello': Cannot allocate memory
root@d5a9554355e1:~/go# touch hello
root@d5a9554355e1:~/go# ls
a.txt  hello  parnoid.go
root@d5a9554355e1:~/go#
```

Examples

- Inject alloc inode failure on a special symlink file path



```
→  tools git:(libbpf-tools) ✗ ./inject.py kmalloc 'ext4_symlink(s
truct inode *dir, struct dentry *dentry, const char *symname)(STR
CMP(dentry->d_name.name, 'world'))'
/virtual/main.c:74:15: warning: initializing 'char *' with an
      expression of type 'const unsigned char *' discards
      qualifiers
      [-Wincompatible-pointer-types-discards-qualifiers]
        char *str_0_1 = dentry->d_name.name;
              ^         ~~~~~~~~~~~~~~~~~~~
1 warning generated.
^C
→  tools git:(libbpf-tools) ✗
```

```
36914   36914   ln              ext4_alloc_inode
        ext4_alloc_inode+0x1 [kernel]
        new_inode_pseudo+0x11 [kernel]
        new_inode+0x16 [kernel]
        __ext4_new_inode+0x12f [kernel]
        ext4_symlink+0x168 [kernel]
        kretprobe_trampoline+0x0 [kernel]
        ovl_create_real+0x228 [overlay]
        ovl_create_or_link+0x15e [overlay]
        ovl_create_object+0xe7 [overlay]
        ovl_symlink+0x1b [overlay]
        vfs_symlink+0xe7 [kernel]
        do_symlinkat+0xe2 [kernel]
        __x64_sys_symlinkat+0x1a [kernel]
        do_syscall_64+0x57 [kernel]
        entry_SYSCALL_64_after_hwframe+0x44 [kernel]
```

```
root@d5a9554355e1:~/go# ls
a.txt  hello  parnoid.go
root@d5a9554355e1:~/go# rm -rf hello
root@d5a9554355e1:~/go# touch hello
root@d5a9554355e1:~/go# touch world
root@d5a9554355e1:~/go# rm -rf world
root@d5a9554355e1:~/go# ln -s hello world
ln: failed to create symbolic link 'world': Cannot allocate memory
root@d5a9554355e1:~/go# ln -s hello world
root@d5a9554355e1:~/go#
```

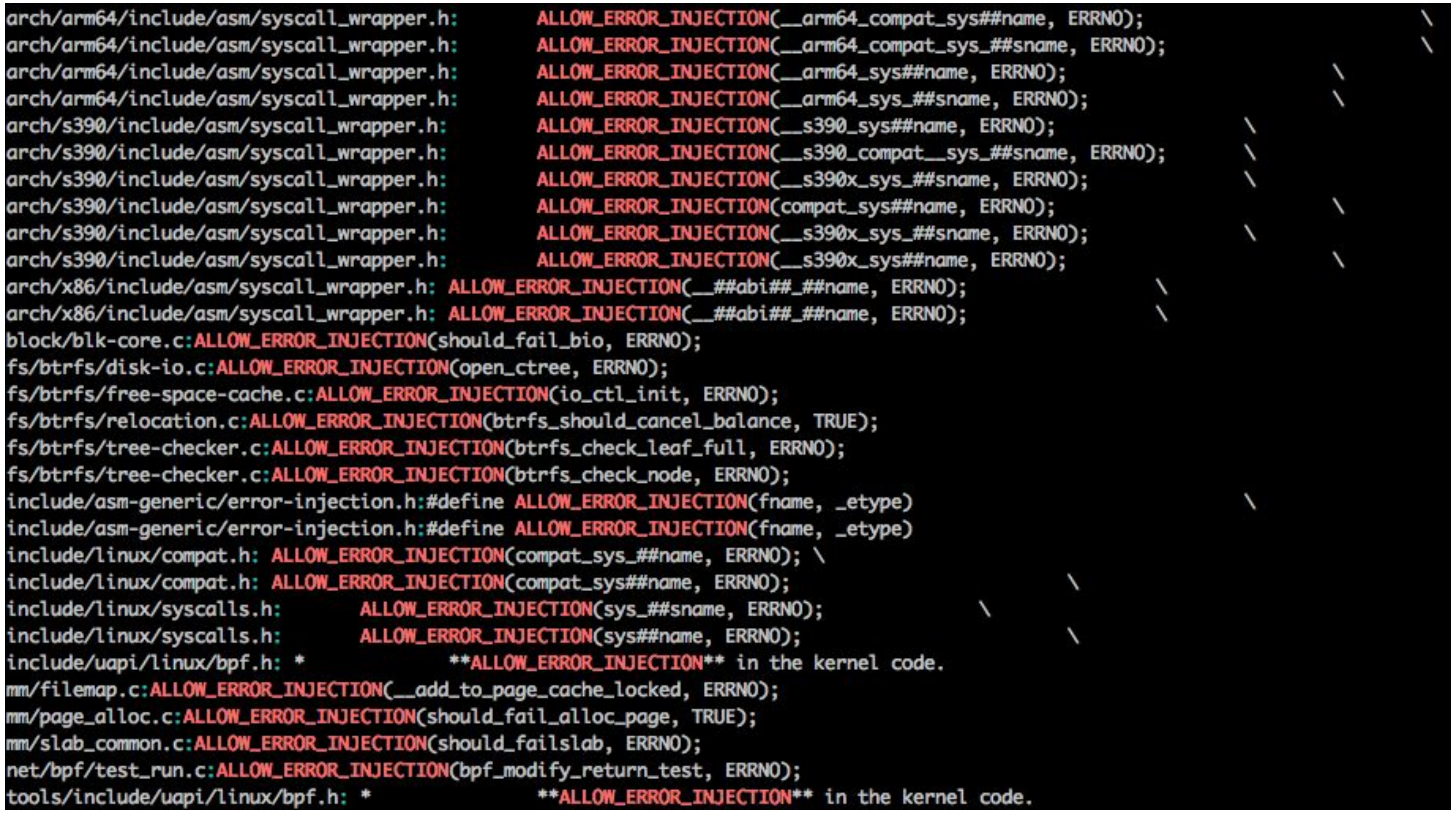
Examples

- Accuracy depends on your familiarity with kernel code
  - eg (from bcc):

```
struct disk_part_tbl *tbl = d->part_tbl;
struct hd_struct **parts = (void *)tbl + sizeof(struct disk_part_tbl);
struct hd_struct **partp = parts + bio->bi_partno;
struct hd_struct *p = *partp;
dev_t disk = p->__dev.devt;
disk == MKDEV(254,16) && bio->bi_iter.bi_sector == 128
```

Where to start

- Injection of syscalls is a good entry point



```
arch/arm64/include/asm/syscall_wrapper.h:       ALLOW_ERROR_INJECTION(__arm64_compat_sys##name, ERRNO);                         \
arch/arm64/include/asm/syscall_wrapper.h:       ALLOW_ERROR_INJECTION(__arm64_compat_sys_##sname, ERRNO);                       \
arch/arm64/include/asm/syscall_wrapper.h:       ALLOW_ERROR_INJECTION(__arm64_sys##name, ERRNO);                        \
arch/arm64/include/asm/syscall_wrapper.h:       ALLOW_ERROR_INJECTION(__arm64_sys_##sname, ERRNO);                      \
arch/s390/include/asm/syscall_wrapper.h:        ALLOW_ERROR_INJECTION(__s390_sys##name, ERRNO);                 \
arch/s390/include/asm/syscall_wrapper.h:        ALLOW_ERROR_INJECTION(__s390_compat__sys_##sname, ERRNO);       \
arch/s390/include/asm/syscall_wrapper.h:        ALLOW_ERROR_INJECTION(__s390x_sys_##sname, ERRNO);              \
arch/s390/include/asm/syscall_wrapper.h:        ALLOW_ERROR_INJECTION(compat_sys##name, ERRNO);                         \
arch/s390/include/asm/syscall_wrapper.h:        ALLOW_ERROR_INJECTION(__s390x_sys_##sname, ERRNO);              \
arch/s390/include/asm/syscall_wrapper.h:        ALLOW_ERROR_INJECTION(__s390x_sys##name, ERRNO);                        \
arch/x86/include/asm/syscall_wrapper.h: ALLOW_ERROR_INJECTION(__##abi##_##name, ERRNO);                 \
arch/x86/include/asm/syscall_wrapper.h: ALLOW_ERROR_INJECTION(__##abi##_##name, ERRNO);                 \
block/blk-core.c:ALLOW_ERROR_INJECTION(should_fail_bio, ERRNO);
fs/btrfs/disk-io.c:ALLOW_ERROR_INJECTION(open_ctree, ERRNO);
fs/btrfs/free-space-cache.c:ALLOW_ERROR_INJECTION(io_ctl_init, ERRNO);
fs/btrfs/relocation.c:ALLOW_ERROR_INJECTION(btrfs_should_cancel_balance, TRUE);
fs/btrfs/tree-checker.c:ALLOW_ERROR_INJECTION(btrfs_check_leaf_full, ERRNO);
fs/btrfs/tree-checker.c:ALLOW_ERROR_INJECTION(btrfs_check_node, ERRNO);
include/asm-generic/error-injection.h:#define ALLOW_ERROR_INJECTION(fname, _etype)                             \
include/asm-generic/error-injection.h:#define ALLOW_ERROR_INJECTION(fname, _etype)
include/linux/compat.h: ALLOW_ERROR_INJECTION(compat_sys_##name, ERRNO); \
include/linux/compat.h: ALLOW_ERROR_INJECTION(compat_sys##name, ERRNO);                         \
include/linux/syscalls.h:       ALLOW_ERROR_INJECTION(sys_##sname, ERRNO);              \
include/linux/syscalls.h:       ALLOW_ERROR_INJECTION(sys##name, ERRNO);                        \
include/uapi/linux/bpf.h: *             **ALLOW_ERROR_INJECTION** in the kernel code.
mm/filemap.c:ALLOW_ERROR_INJECTION(__add_to_page_cache_locked, ERRNO);
mm/page_alloc.c:ALLOW_ERROR_INJECTION(should_fail_alloc_page, TRUE);
mm/slab_common.c:ALLOW_ERROR_INJECTION(should_failslab, ERRNO);
net/bpf/test_run.c:ALLOW_ERROR_INJECTION(bpf_modify_return_test, ERRNO);
tools/include/uapi/linux/bpf.h: *               **ALLOW_ERROR_INJECTION** in the kernel code.
```

Where to start

- You can try kernel chaos on

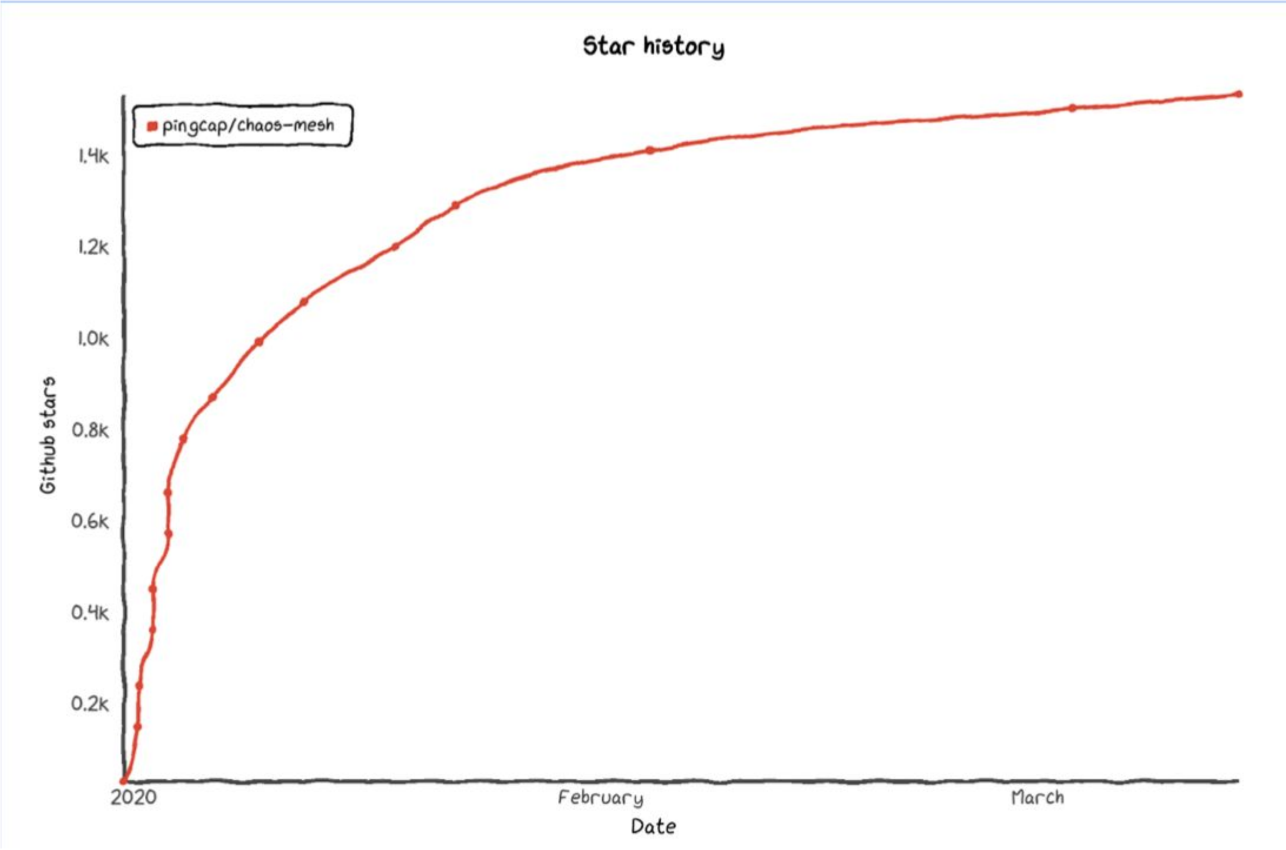
Chaos Mesh®

CLOUD NATIVE + OPEN SOURCE
Virtual Summit China 2020

Chaos Mesh®

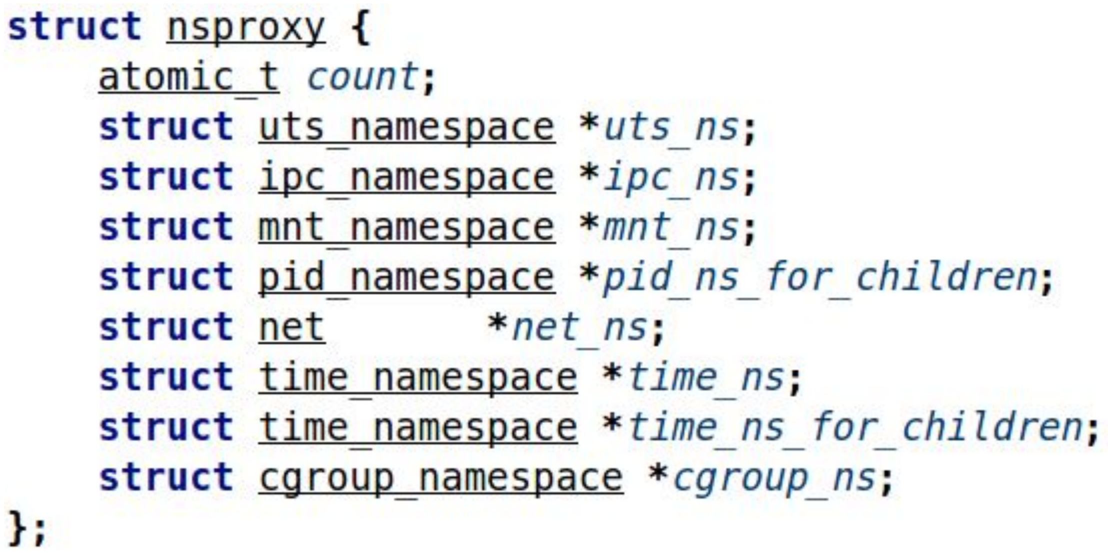
- Community and Ecosystem

  - 1.9k stars on Github
  - 41 contributors
    - PingCAP
    - Red Hat
    - DailyMotion
    - Nvidia
    - Xpeng Motors
    - China Merchants Bank
    - Meituan Dianping
  - 400+ commits
  - Monthly meeting

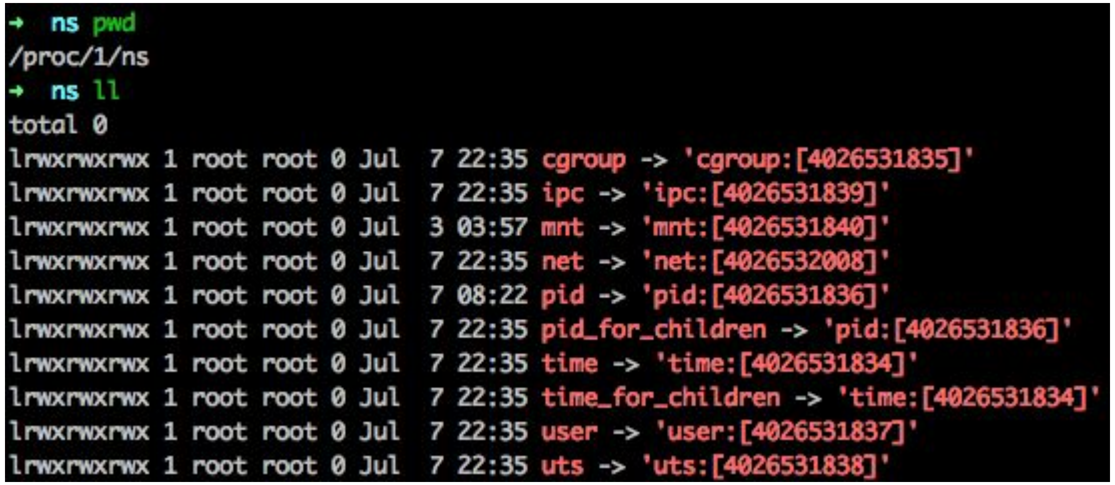


BPF With Native Support For Containers

- Challenges
  - PIDs in host don’t match those seen in containers
  - Kernel currently doesn’t have a container ID
- Solutions
  - If in process context, we can read nsproxy struct in the kernel
    - *(struct task_struct*)task->nsproxy->pid_ns_for_children->ns.inum*
    - *(struct task_struct*)task->nsproxy->uts_ns->name.nodename*



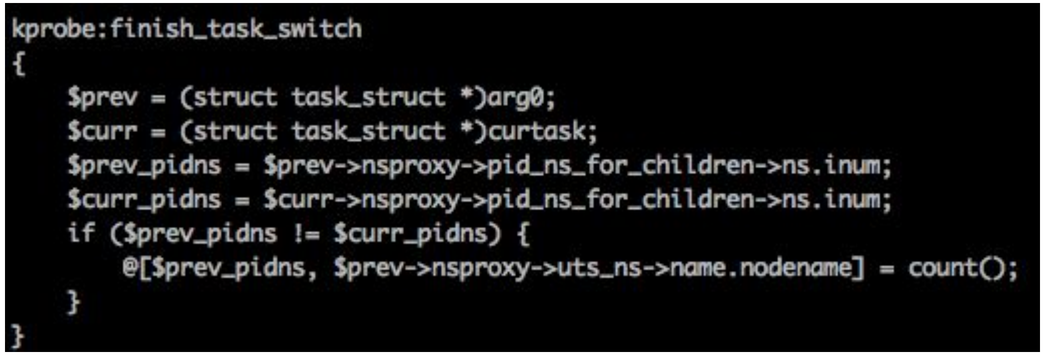
```
struct nsproxy {
    atomic_t count;
    struct uts_namespace *uts_ns;
    struct ipc_namespace *ipc_ns;
    struct mnt_namespace *mnt_ns;
    struct pid_namespace *pid_ns_for_children;
    struct net         *net_ns;
    struct time_namespace *time_ns;
    struct time_namespace *time_ns_for_children;
    struct cgroup_namespace *cgroup_ns;
};
```



```
➜  ns pwd
/proc/1/ns
➜  ns ll
total 0
lrwxrwxrwx 1 root root 0 Jul  7 22:35 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Jul  7 22:35 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Jul  3 03:57 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 root root 0 Jul  7 22:35 net -> 'net:[4026532008]'
lrwxrwxrwx 1 root root 0 Jul  7 08:22 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Jul  7 22:35 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Jul  7 22:35 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Jul  7 22:35 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Jul  7 22:35 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Jul  7 22:35 uts -> 'uts:[4026531838]'
```

BPF With Native Support For Containers

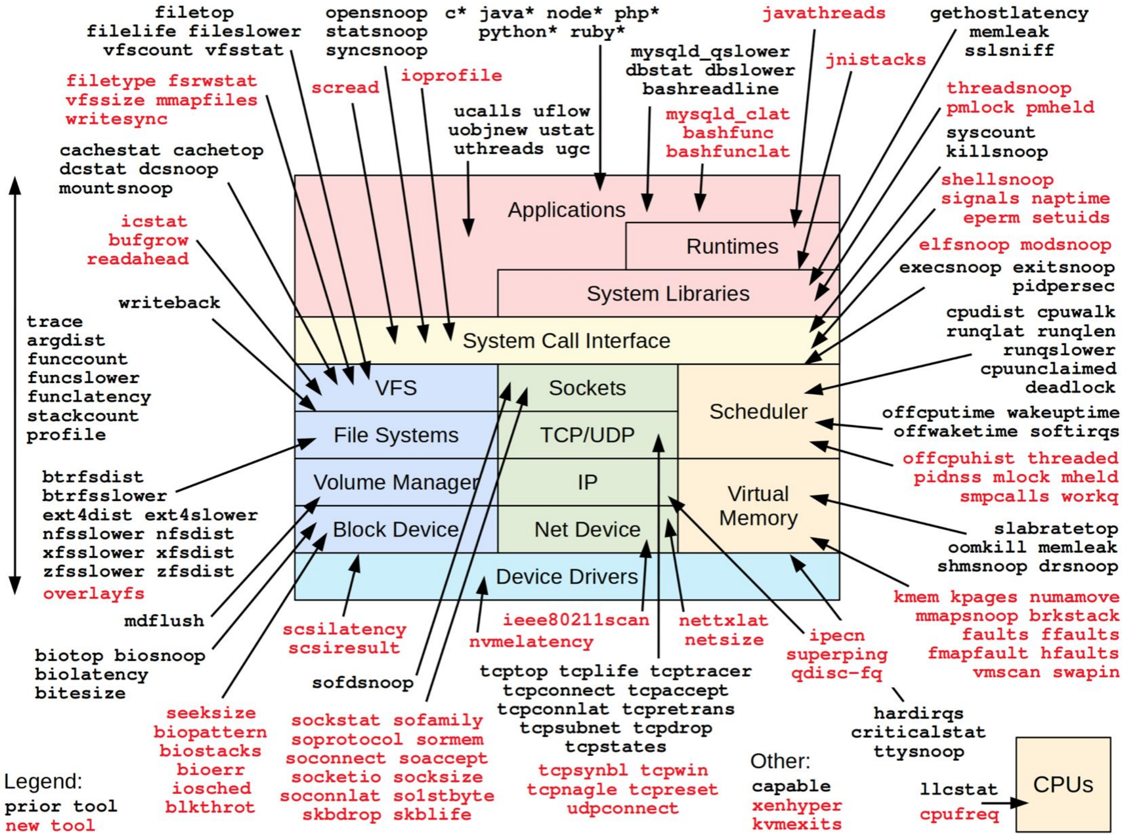
- Example



```
kprobe:finish_task_switch
{
    $prev = (struct task_struct *)arg0;
    $curr = (struct task_struct *)curtask;
    $prev_pidns = $prev->nsproxy->pid_ns_for_children->ns.inum;
    $curr_pidns = $curr->nsproxy->pid_ns_for_children->ns.inum;
    if ($prev_pidns != $curr_pidns) {
        @[$prev_pidns, $prev->nsproxy->uts_ns->name.nodename] = count();
    }
}
```

K8s tracing with BPF

- BPF Tools
  - Over 150+ that you can run to find performance wins and troubleshoot software
  - With kubectl-trace's help bpftrace progs (already a pr to support bcc progs) can be scheduled in Kubernetes cluster

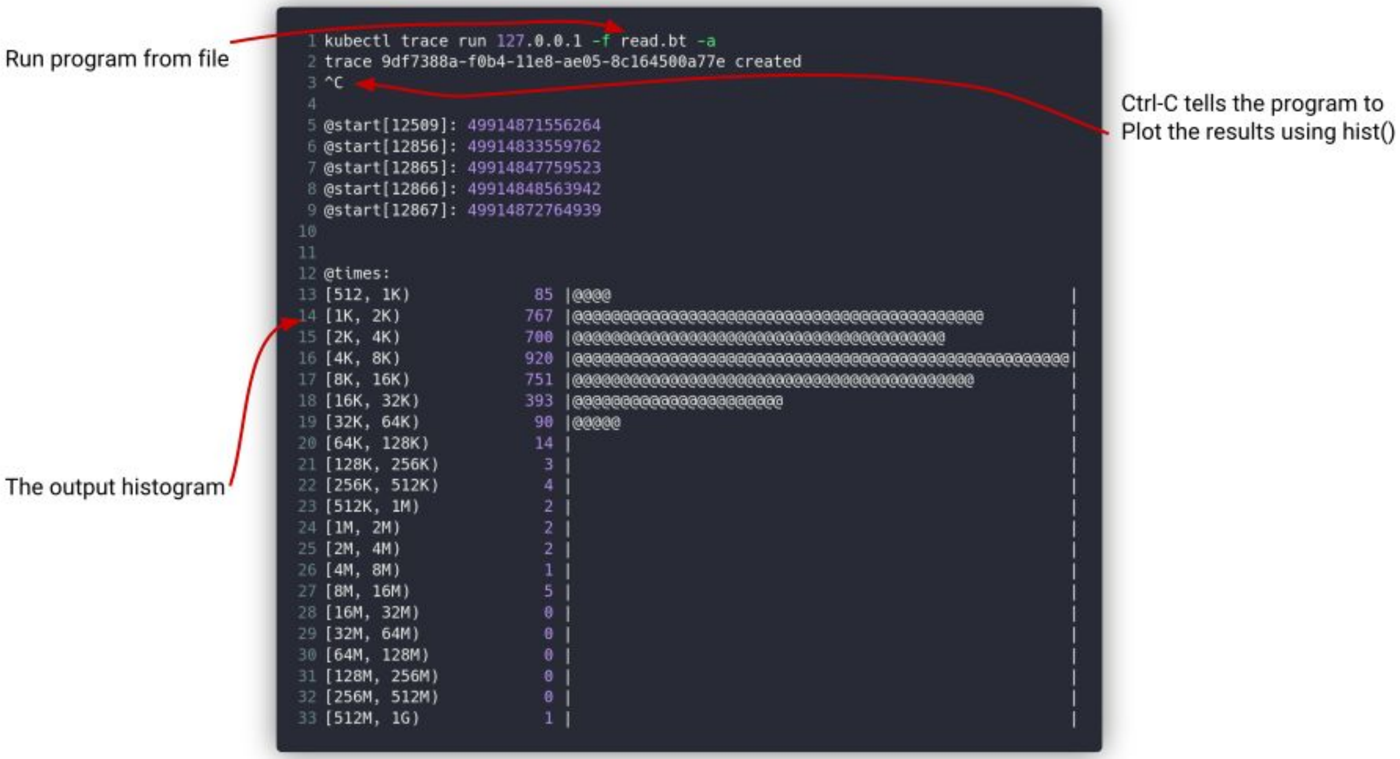


K8s tracing with BPF

- BCC drawbacks
  - Clang/LLVM combo is a big library, resulting in big fat binaries that need to be distributed with your application
  - Clang/LLVM combo is resource-heavy, so when you are compiling BPF code at start up, you'll use a significant amount of resources, potentially tipping over a carefully balanced production workfload. And vice versa, on a busy host, compiling a small BPF program might take minutes in some cases.
  - You are making a big bet that the target system will have kernel headers present, which most of the time is not a problem, but sometimes can cause a lot of headaches. This is also an especially annoying requirement for kernel developers, because they often have to build and deploy custom one-off kernels as part of their development process. And without a custom-built kernel header package, no BCC-based application will work on such kernels, stripping developers of a useful set of tools for debugging and monitoring.
  - BPF program testing and development iteration is quite painful as well, as you are going to get even most trivial compilation errors only in runtime, once you recompile and restart your user-space control application. This certainly increases friction and is not helping to iterate fast.

Tracing with BPF

- Trace node with kubectl-trace



Run program from file

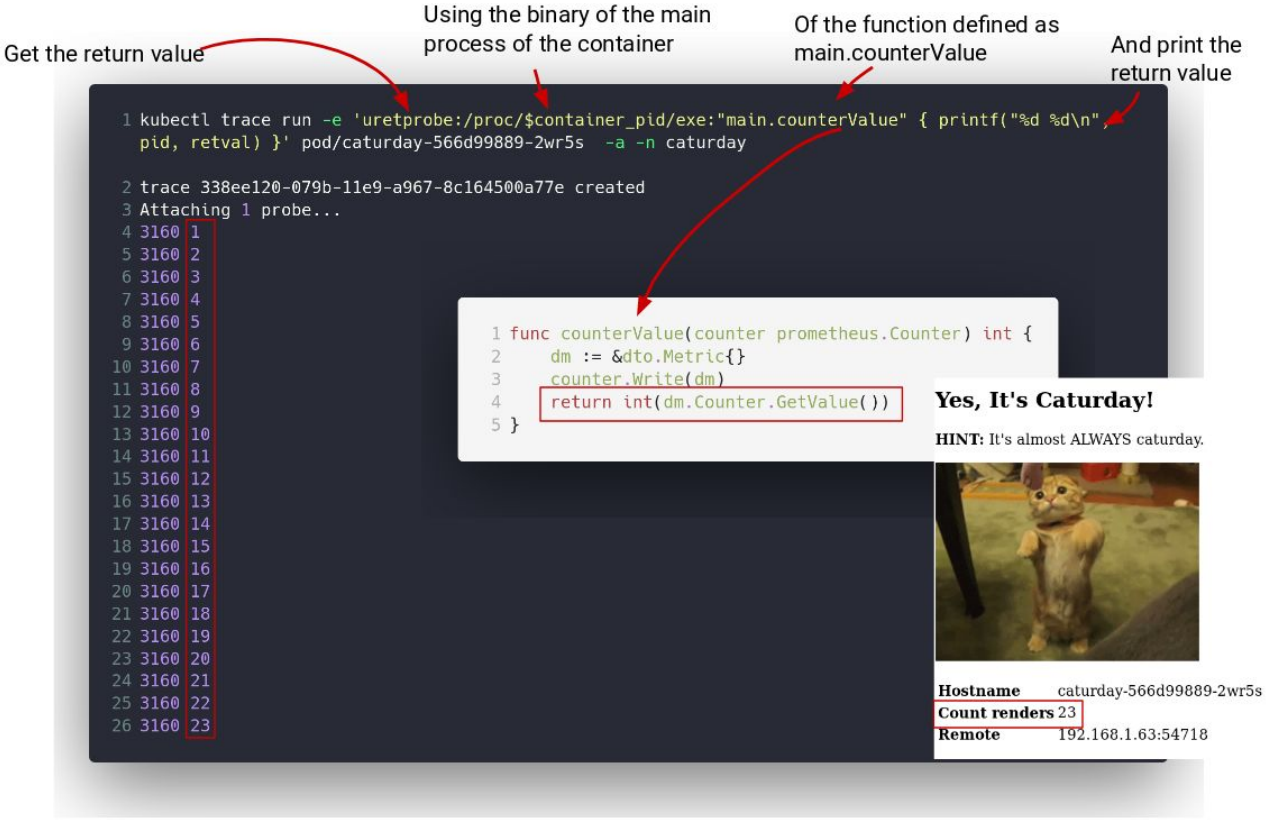
Ctrl-C tells the program to
Plot the results using hist()

The output histogram

```
kubectl trace run 127.0.0.1 -f read.bt -a
trace 9df7388a-f0b4-11e8-ae05-8c164500a77e created
^C

@start[12509]: 49914871556264
@start[12856]: 49914833559762
@start[12865]: 49914847759523
@start[12866]: 49914848563942
@start[12867]: 49914872764939


@times:
[512, 1K)             85 |@@@@                                                |
[1K, 2K)             767 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@         |
[2K, 4K)             700 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@             |
[4K, 8K)             920 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[8K, 16K)            751 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@          |
[16K, 32K)           393 |@@@@@@@@@@@@@@@@@@@@@@                              |
[32K, 64K)            90 |@@@@@                                               |
[64K, 128K)           14 |                                                    |
[128K, 256K)           3 |                                                    |
[256K, 512K)           4 |                                                    |
[512K, 1M)             2 |                                                    |
[1M, 2M)               2 |                                                    |
[2M, 4M)               2 |                                                    |
[4M, 8M)               1 |                                                    |
[8M, 16M)              5 |                                                    |
[16M, 32M)             0 |                                                    |
[32M, 64M)             0 |                                                    |
[64M, 128M)            0 |                                                    |
[128M, 256M)           0 |                                                    |
[256M, 512M)           0 |                                                    |
[512M, 1G)             1 |                                                    |
```

Tracing with BPF

- Trace pod with kubectl-trace



Get the return value

Using the binary of the main process of the container

Of the function defined as main.counterValue

And print the return value

```
kubectl trace run -e 'uretprobe:/proc/$container_pid/exe:"main.counterValue" { printf("%d %d\n",
pid, retval) }' pod/caturday-566d99889-2wr5s  -a -n caturday

trace 338ee120-079b-11e9-a967-8c164500a77e created
Attaching 1 probe...
3160 1
3160 2
3160 3
3160 4
3160 5
3160 6
3160 7
3160 8
3160 9
3160 10
3160 11
3160 12
3160 13
3160 14
3160 15
3160 16
3160 17
3160 18
3160 19
3160 20
3160 21
3160 22
3160 23
```

```
func counterValue(counter prometheus.Counter) int {
    dm := &dto.Metric{}
    counter.Write(dm)
    return int(dm.Counter.GetValue())
}
```

Yes, It's Caturday!

HINT: It's almost ALWAYS caturday.

Hostname	caturday-566d99889-2wr5s
Count renders	23
Remote	192.168.1.63:54718

Tracing with BPF

- Container-Specific Tools
  - runqlat --pidnss -m
  - pidnss
  - blkthrot
  - overlayfs

Thank you!

CLOUD NATIVE + OPEN SOURCE

Virtual Summit China 2020