Making Memory Account Accountable: Analyzing and Detecting Memory Missing-account bugs for Container Platforms

Yutian Yang
Zhejiang University
ytyang@zju.edu.cn

Wenbo Shen
Zhejiang University

Xun Xie
Zhejiang University

Kangjie Lu
University of Minnesota, Twin Cities

Mingsen Wang
Zhejiang University

Tianyu Zhou
Zhejiang University

Chenggang Qin
Ant Group

Wang Yu
Ant Group

Kui Ren
Zhejiang University

UNIVERSITY OF MINNESOTA ANT GROUP ZHEJIANG UNIVERSITY 浙江大學

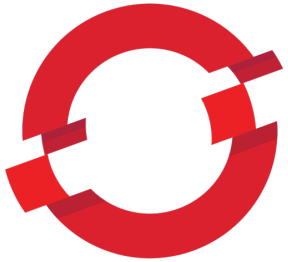
Outline

- Background and the problem
- Our contributions
- Exploitability and impact of missing-account bugs
  - Native runtimes & secure runtimes
  - CaaS platforms & FaaS platforms
- Automated bug detection
- Takeaways

Background

- OS-level virtualization (containers) is widely adopted in cloud platforms to virtualize and share hardware resources among different users.

OpenWhisk
(IBM Cloud functions)

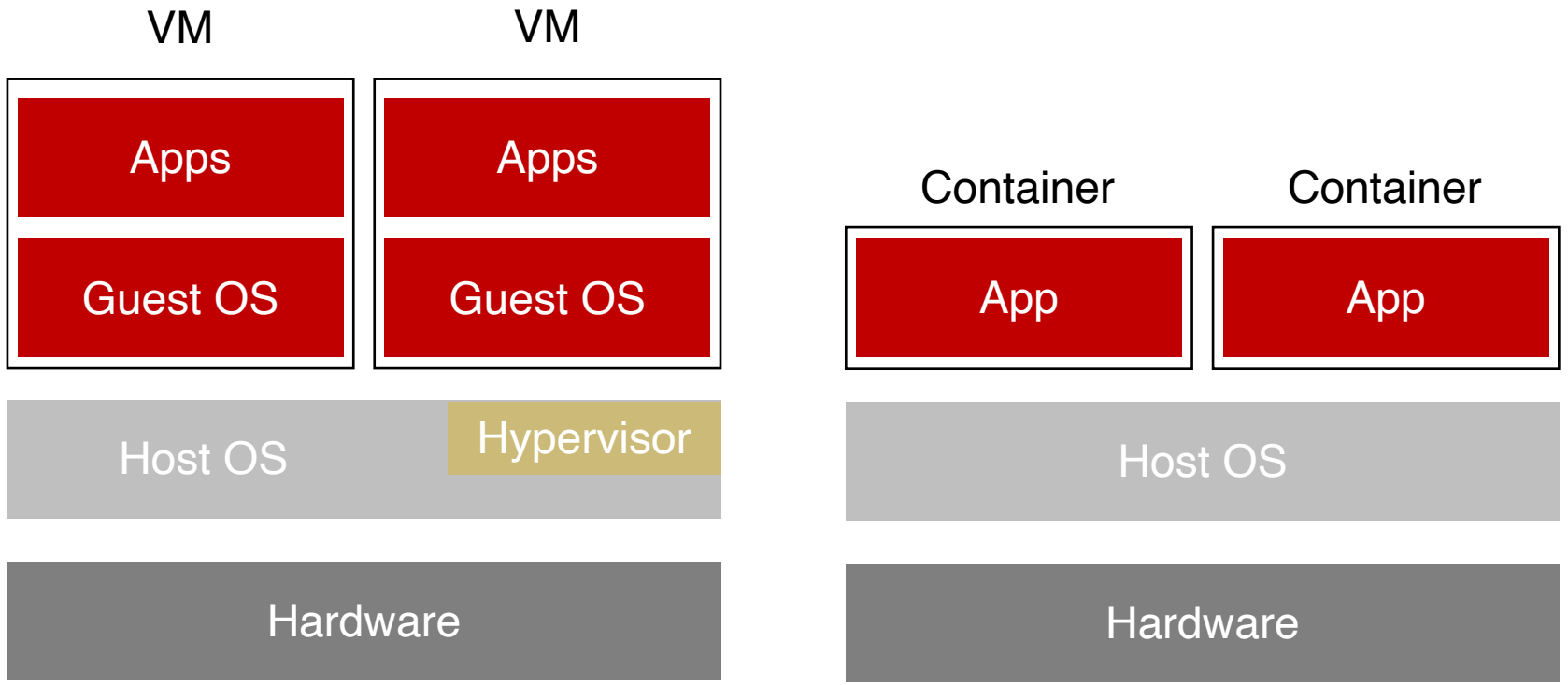


Google GKE

.....

Background

- Compared with VM, OS-level virtualization (containers) is more lightweight and efficient. Different containers directly interact with the same host OS kernel.



Background (cont.)

- Memory usage accounting and limitation serves as the cornerstone for container techniques.

- Linux kernel proposes memory control groups (memcg) to account for and limit memory usage at per-process granularity.

- Our preliminary study shows that memcg is error-prone.
  - Numerous memory allocation sites in Linux kernel, which can be missed by memcg.
  - Accounting flags can be missing.
  - Existing bugs.

Problem

- Does memcg precisely account for memory usage? Does it miss any memory usage?
  - Memcg is not precise and it has missing-account bugs.
- What are the security impacts of memory missing-account bugs?
- How to efficiently discover these bugs in complex OS kernels?

Our Contribution

ZHEJIANG UNIVERSITY

- In-depth analysis of exploitability and impacts.
  - Memory missing-account bugs is exploitable on different container runtimes and container platforms.
  - The attack leads to DoS on the host nodes or even the whole cluster.
- Automated detection with new techniques.
  - Counter-based interface identification.
  - Alloc-charging mapping analysis.
  - Accounting flag analysis.
  - Dynamic validation.
- Community impact.
  - 53 exploitable missing-account bugs.
  - 37 confirmed, 18 patches merged.
  - 2 new CVEs.

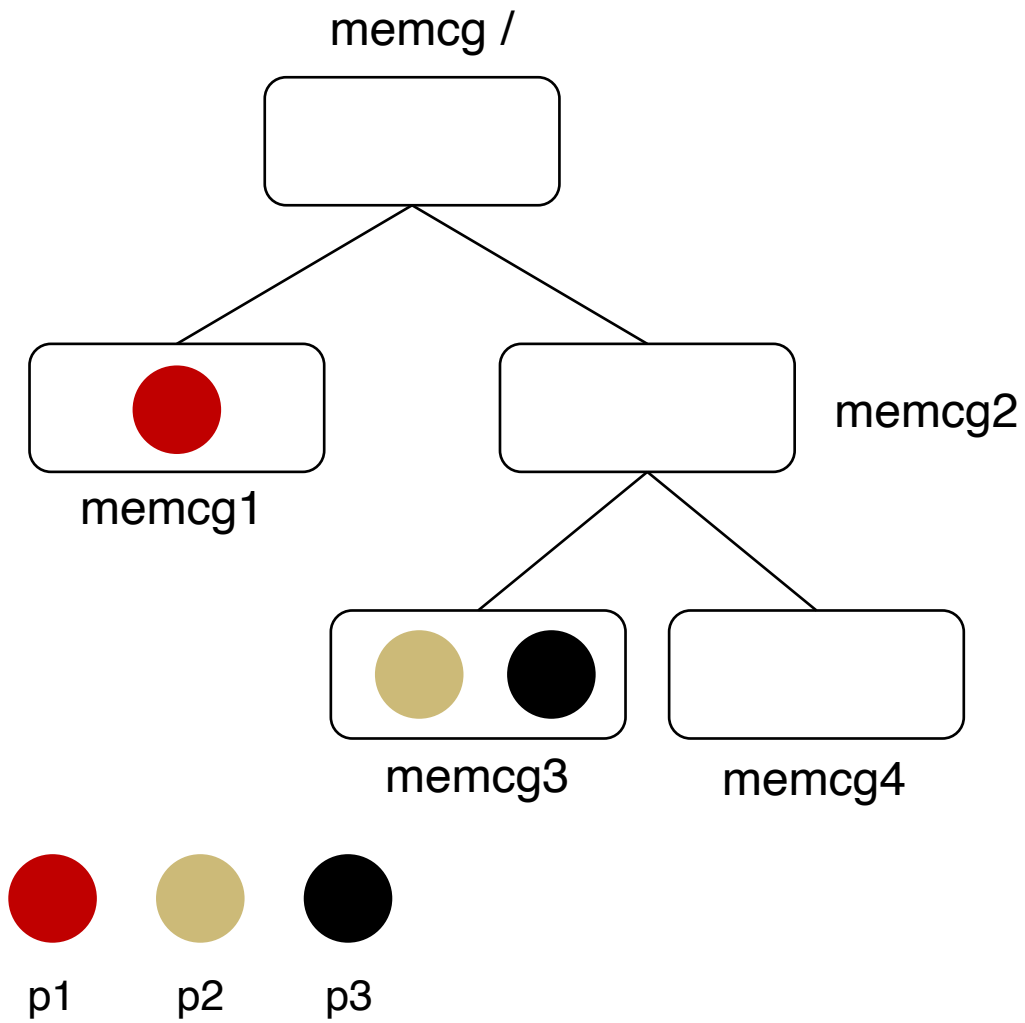
Memcg 101

- Memcg instances are organized as a tree.

  memcg1.usage = $MemUsage_{p1}$
  memcg3.usage = $MemUsage_{p2}$ + $MemUsage_{p3}$
  memcg2.usage = memcg3.usage + memcg4.usage

- Processes in the memcg receive OOM errors if the usage exceeds the limitation.

  usage(memcg) < max(memcg)



Memcg 101

- A memcg instance maintains page counters to record memory usage.

- After memory (de)allocation happens, the kernel invokes accounting interface to increase/decrease the page counters.

```
1  struct mem_cgroup {
2      ...
3      struct page_counter memory;/* Both v1 & v2 */
4      union {
5          struct page_counter swap;     /* v2 only */
6          struct page_counter memsw;     /* v1 only */
7      };
8      /* Legacy consumer-oriented counters */
9      struct page_counter kmem;     /* v1 only */
10     struct page_counter tcpmem;     /* v1 only */
11     ...
12 }
13 struct page_counter {
14     atomic_long_t usage;
15     ...
16     unsigned long max;
17     ...
18 }
19
20 bool page_counter_try_charge(struct page_counter *counter, unsigned
   ↪ long nr_pages, ...)
21 {
22     struct page_counter *c;
23     ...
24     new = atomic_long_add_return(nr_pages, &c->usage);
25     if (new > c->max) {
26         ...
27         goto failed;
28     }
29 }
```

Outline

- Background and the problem
- Our contributions
- Exploitability and impact of missing-account bugs
  - Native runtimes & secure runtimes
  - CaaS platforms & FaaS platforms
- Automated bug detection
- Takeaway

Attacking Native Runtimes

- We use Docker as the native runtime.
- The missing-account bug, CVE-2021-3759.

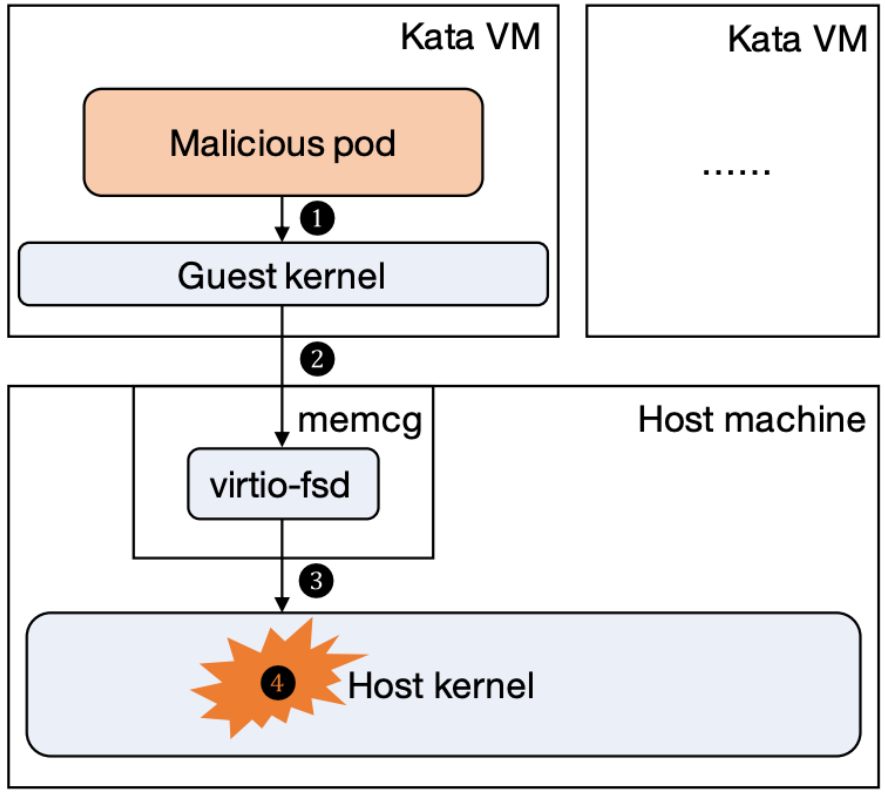
```
1   static struct sem_array *sem_alloc(size_t nsems)
2   {
3       struct sem_array *sma;
4
5       if (nsems > (INT_MAX - sizeof(*sma)) / sizeof(sma->sems[0]))
6           return NULL;
7
8       sma = kvzalloc(struct_size(sma, sems, nsems), GFP_KERNEL);
9       if (unlikely(!sma))
10          return NULL;
11
12      return sma;
13  }
```

GFP_KERNEL_ACCOUNT

- Impact: normal user in container can exhaust all 16 GB host memory, leading to system-wide OOM errors.

Attacking Secure Runtimes

- We use Kata Containers as the secure runtime.
- We exploit the POSIX lock missing-account bug, CVE-2022-0480.



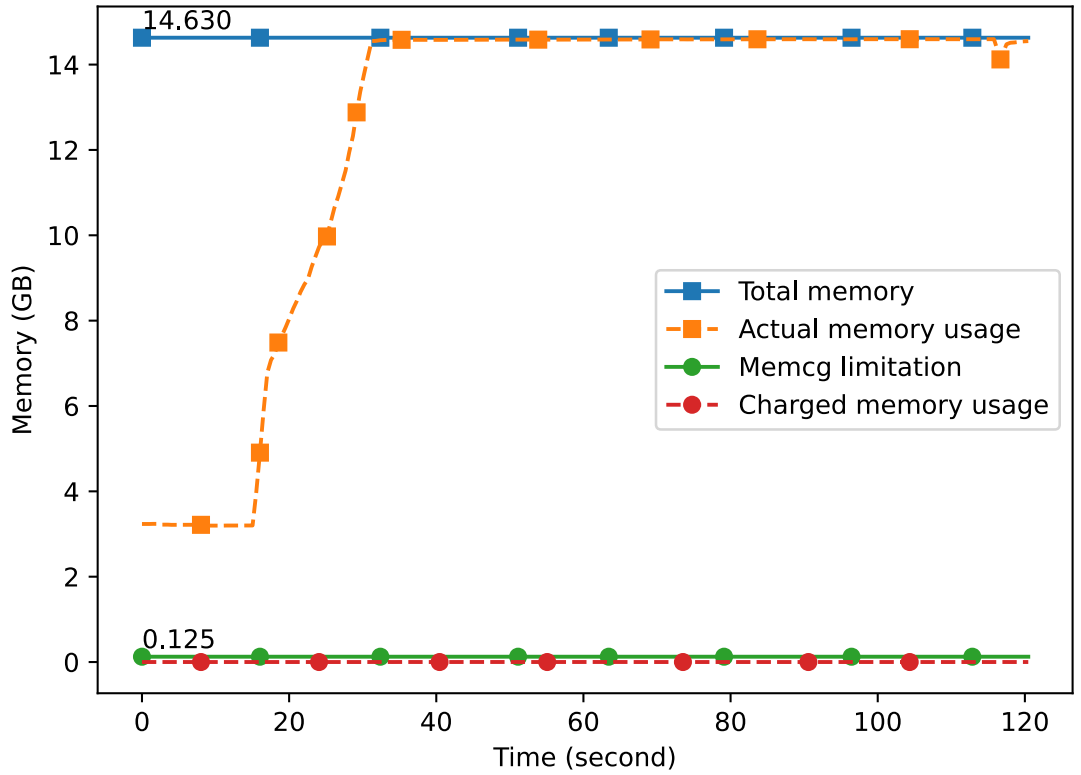
1. Attacker's pod allocates Posix locks by fcntl() syscall.
2. The guest kernel forwards the request to virtio-fsd daemon.
3. virtio-fsd allocates Posix locks on the host machine.
4. Missing-charged lock objects lead to memory exhaustion.

Attacking Secure Runtimes

- We use Kata Containers as the secure runtime.
- We exploit the POSIX lock missing-account bug, CVE-2022-0480.
- Impact: normal user in a Kata container can exhaust all 16 GB host memory, leading to system-wide OOM errors.

Attacking CaaS&FaaS Platforms

- We evaluate CVE-2021-3759 on OpenShift CaaS platform and OpenWhisk FaaS platform.



Attacking CaaS&FaaS Platforms

ZHEJIANG UNIVERSITY

- We evaluate CVE-2021-3759 on OpenShift CaaS platform and OpenWhisk FaaS platform.

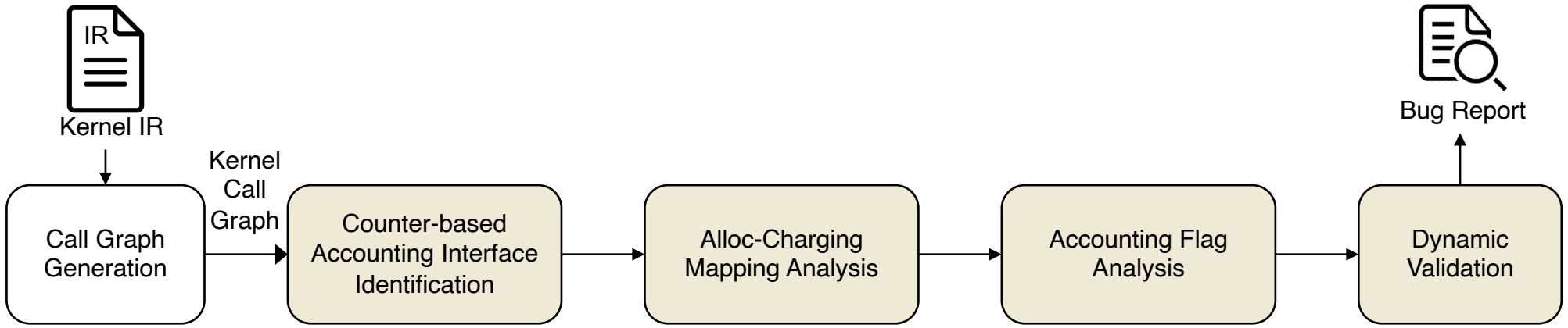
- Impact.
  - For CaaS, the attacked node is crashed and all containers on it are down.
  - For FaaS, all worker nodes are crashed and all function run fails.

Outline

- Background and the problem
- Our contributions
- Exploitability and impact of missing-account bugs
  - Native runtimes & secure runtimes
  - CaaS platforms & FaaS platforms
- Automated bug detection
- Takeaway

MANTA Analyzer

- Challenges:
  - Undocumented accounting interfaces.
  - Mapping memory allocation to accounting interface call among the complex execution paths.



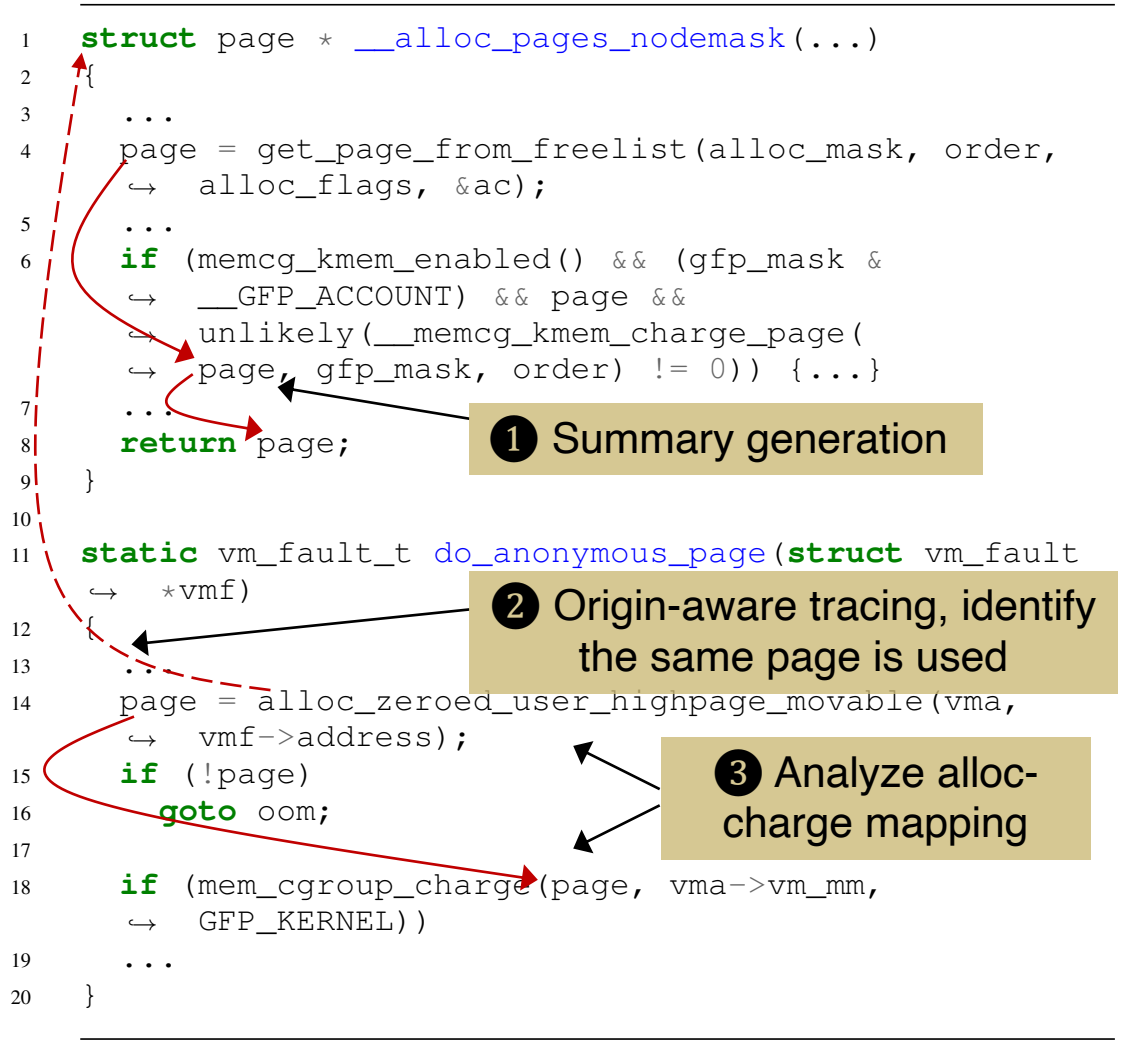
Accounting Interface Identification

- Our observation: all accounting interfaces finally increase/decrease the page counter.
- The traversal ends when it reaches out of memcg module.

```
1  bool page_counter_try_charge(struct page_counter *counter,
2                               unsigned long nr_pages, ...) {
3      struct page_counter *c;
4
5      for (c = counter; c; c = c->parent) {
6          long new;
7          new = atomic_long_add_return(nr_pages, &c->usage);
8          ......                                  page counter add
9      }
10     ......
11 }
12
13 int __memcg_kmem_charge_page(struct page *page, gfp_t gfp,
14                               int order) {
15     struct obj_cgroup *objcg;
16     int ret = 0;
17
18     objcg = get_obj_cgroup_from_current();
19     if (objcg) {
20         ret = obj_cgroup_charge_pages(objcg, gfp, 1 <<
           ↪  order);                                          Invoke
21         ......
22     }
23     ......
24 }
```

Alloc-Charging Mapping Analysis

- Our observation: pre-compute the summary of each function instead of recursively following the complex call chain.
- The summary generation is in a bottom-up order on call chains.
- MANTA raises a missing-account alarm for page that is not accounted nor escaped from current function.



Accounting Flag Analysis

- Bit-wise inter-procedural value tracing.

```
1  static int bpf_prog_load(...) {
2      ......
3      /* plain bpf_prog allocation */
4      prog = bpf_prog_alloc(bpf_prog_size(attr->insn_cnt),
       ↪  GFP_USER);
5      ......   missing-account!
6  }
7
8  struct bpf_prog *bpf_prog_alloc(unsigned int size, gfp_t
   ↪  gfp_extra_flags) {
9      gfp_t gfp_flags = GFP_KERNEL | __GFP_ZERO |
       ↪  gfp_extra_flags;
10     struct bpf_prog *prog;
11     int cpu;
12                          allocation site
13     prog = bpf_prog_alloc_no_stats(size, gfp_extra_flags);
14     if (!prog)
15         return NULL;
16                          allocation site
17     prog->stats = alloc_percpu_gfp(struct bpf_prog_stats,
       ↪  gfp_flags);
18     ......
19 }
```

Dynamic Validation

- For the 242 reported bugs, we further evaluate the triggerability from user space applications.

- Method: in-kernel instrumentation + user-space test cases.

  LTP test cases

  Common user-space tools (SeLinux, auditd, etc.)

  Manually developed test cases

Analysis Results

ZHEJIANG UNIVERSITY

- Analysis Results
  - Alerts: 242
  - Triggerable: 162
  - Exploitable & Reported: 53
  - Confirmed: 37
  - Patched: 18
  - Analysis precision: 66.9% (162/242)
- Impact analysis
  - 17 PoCs
  - 5 can exhaust more than 16GB host memory.

Bug ID	Source location	Function name	Allocation interface	Triggered by	Confirm Status	Patch Status
1	arch/x86/kernel/ldt.c:157	alloc_ldt_struct	kmalloc	clone	Confirmed	Merged
2	fs/eventfd.c:417	do_eventfd	kmalloc	eventfd	Pending	-
3	fs/eventpoll.c:1014	ep_alloc	kzalloc	epoll_create	Pending	-
4	fs/fcntl.c:899	fasync_alloc	kmem_cache_alloc	fcntl	Confirmed	Merged
5	fs/fs_context.c:234	alloc_fs_context	kzalloc	fsopen	Confirmed	Merged
6	fs/fs_context.c:634	legacy_init_fs_context	kzalloc	fsopen	Confirmed	Merged
7	fs/fsopen.c:100	fscontext_alloc_log	kzalloc	fsopen	Pending	-
8	fs/io-wq.c:1089	io_wq_create	kzalloc	io_uring_setup	Confirmed	Pending
9	fs/io-wq.c:1093	io_wq_create	kzalloc_node	io_uring_setup	Confirmed	Pending
10	fs/io-wq.c:1114	io_wq_create	kzalloc_node	io_uring_setup	Confirmed	Pending
11	fs/io_uring.c:1180	io_ring_ctx_alloc	kzalloc	io_uring_setup	Confirmed	Pending
12	fs/io_uring.c:1184	io_ring_ctx_alloc	kmem_cache_alloc	io_uring_setup	Confirmed	Pending
13	fs/io_uring.c:1197	io_ring_ctx_alloc	kmalloc	io_uring_setup	Confirmed	Pending
14	fs/io_uring.c:7254	__io_sqe_files_scm	kzalloc	io_uring_register	Pending	-
15	fs/io_uring.c:7507	alloc_fixed_file_ref_node	kzalloc	io_uring_register	Pending	-
16	fs/io_uring.c:7546	io_sqe_files_register	kzalloc	io_uring_register	Pending	-
17	fs/io_uring.c:7853	io_uring_alloc_task_context	kmalloc	io_uring_setup	Confirmed	Pending
18	fs/io_uring.c:8044	io_mem_alloc	__get_free_pages	io_uring_setup	Confirmed	Pending
19	fs/io_uring.c:9541	io_register_personality	kmalloc	io_uring_register	Pending	-
20	fs/locks.c:346	locks_alloc_lock	kmem_cache_zalloc	flock, fcntl	Confirmed	Merged
21	fs/locks.c:258	locks_get_lock_context	kmem_cache_alloc	flock	Confirmed	Merged
22	fs/namespace.c:177	alloc_vfsmnt	kmem_cache_zalloc	mount	Confirmed	Merged
23	fs/namespace.c:3267	alloc_mnt_ns	kzalloc	clone	Confirmed	Pending
24	fs/notify/group.c:121	fsnotify_alloc_group	kzalloc	inotify_init1	Confirmed	Merged
25	fs/notify/inotify/inotify_user.c:630	inotify_new_group	kmalloc	inotify_init1	Confirmed	Merged
26	fs/select.c:658	core_sys_select	kvmalloc	select	Confirmed	Merged
27	fs/signalfd.c:278	do_signalfd4	kmalloc	signalfd4	Pending	-
28	fs/timerfd.c:412	timerfd_create	kzalloc	timerfd_create	Pending	-
29	ipc/namespace.c:45	create_ipc_ns	kzalloc	clone	Confirmed	Pending
30	ipc/sem.c:514	sem_alloc	kvzalloc	semget	Confirmed	Merged
31	ipc/sem.c:1853	get_undo_list	kzalloc	semop	Confirmed	Merged
32	ipc/sem.c:1938	find_alloc_undo	kzalloc	semop	Confirmed	Merged
33	kernel/bpf/core.c:117	bpf_prog_alloc	alloc_percpu_gfp	bpf, prctl	Confirmed	Merged
34	kernel/bpf/core.c:85	bpf_prog_alloc_no_stats	__vmalloc	bpf, prctl	Confirmed	Merged
35	kernel/bpf/core.c:89	bpf_prog_alloc_no_stats	kzalloc	bpf, prctl	Confirmed	Merged
36	kernel/events/core.c:11142	perf_event_alloc	kzalloc	perf_event_open	Pending	-
37	kernel/events/core.c:4443	alloc_perf_context	kzalloc	perf	Pending	-
38	kernel/nsproxy.c:56	create_nsproxy	kmem_cache_alloc	setns	Confirmed	Pending
39	kernel/pid_namespace.c:90	create_pid_namespace	kmem_cache_zalloc	clone	Confirmed	Pending
40	kernel/seccomp.c:1481	init_listener	kzalloc	prctl	Pending	-
41	kernel/seccomp.c:565	seccomp_prepare_filter	kzalloc	prctl	Pending	-
42	kernel/signal.c:435	__sigqueue_alloc	kmem_cache_alloc	rt_sigqueueinfo	Confirmed	Merged
43	kernel/time/namespace.c:91	clone_time_ns	kmalloc	clone	Confirmed	Pending
44	kernel/time/namespace.c:97	clone_time_ns	alloc_page	clone	Confirmed	Pending
45	kernel/time/posix-timers.c:458	alloc_posix_timer	kmem_cache_zalloc	timer_create	Confirmed	Merged
46	kernel/user_namespace.c:105	create_user_ns	kmem_cache_zalloc	clone	Confirmed	Pending
47	mm/hugetlb.c:868	resv_map_alloc	kmalloc	memfd_create	Pending	-
48	mm/hugetlb.c:869	resv_map_alloc	kmalloc	memfd_create	Pending	-
49	mm/slab_common.c:245	create_cache	kmem_cache_zalloc	clone	Confirmed	Pending
50	net/core/net_namespace.c:423	net_alloc	kzalloc	clone	Pending	-
51	security/keys/key.c:277	key_alloc	kmem_cache_alloc	add_key	Confirmed	Pending
52	security/keys/key.c:282	key_alloc	kmemdup	add_key	Confirmed	Pending
53	security/keys/key.c:81	key_user_lookup	kzalloc	add_key	Confirmed	Pending

Takeaways

- It is hard to implement memory accounting correctly. Incorrect memory account can raise severe security threats.
  - DoS on native&secure runtimes.
  - DoS on CaaS&FaaS platforms.

- Static analysis is an effective way to find out and mitigate memory accounting bugs.
  - 53 reported bugs, 37 confirmed and 18 patches merged.

- We open source our tools and call upon more efforts on improving memcg.
  - https://github.com/ZJU-SEC/MANTA

?