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
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](image1.png)

OpenWhisk (IBM Cloud functions)

![OpenShift](image2.png)

OpenShift

![Google GKE](image3.png)

Google GKE

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

- 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.
  
  \[ \text{memcg1.usage} = \text{MemUsage}_{p1} \]
  
  \[ \text{memcg3.usage} = \text{MemUsage}_{p2} + \text{MemUsage}_{p3} \]
  
  \[ \text{memcg2.usage} = \text{memcg3.usage} + \text{memcg4.usage} \]

- Processes in the memcg receive OOM errors if the usage exceeds the limitation.
  
  \[ \text{usage(memcg)} < \text{max(memcg)} \]
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.

```c
struct mem_cgroup {
    ... 
    struct page_counter memory; /* Both v1 & v2 */
    union {
        struct page_counter swap;  /* v2 only */
        struct page_counter memsw; /* v1 only */
    };
    /* Legacy consumer-oriented counters */
    struct page_counter kmem;   /* v1 only */
    struct page_counter tcpmem; /* v1 only */
    ..., 
}

struct page_counter {
    atomic_long_t usage;
    ..., 
    unsigned long max;
    ... 
}

bool page_counter_try_charge(struct page_counter *counter, unsigned long nr_pages, ...)
{
    struct page_counter *c;
    ... 
    new = atomic_long_add_return(nr_pages, &c->usage);
    if (new > c->max) {
        ... 
        goto failed;
    }
}
```
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.

```c
static struct sem_array *sem_alloc(size_t nsems)
{
    struct sem_array *sma;
    if (nsems > (INT_MAX - sizeof(*sma)) / sizeof(sma->sems[0]))
        return NULL;
    sma = kzalloc(struct_size(sma, sems, nsems), GFP_KERNEL);
    if (unlikely(!sma))
        return NULL;
    return sma;
}
```

- Impact: normal user in container can exhaust all 16 GB host memory, leading to system-wide OOM errors.
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.
We evaluate CVE-2021-3759 on OpenShift CaaS platform and OpenWhisk FaaS platform.
Attacking CaaS&FaaS Platforms

- 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.

```c
bool page_counter_try_charge(struct page_counter *counter,
   unsigned long nr_pages, ...) {
    struct page_counter *c;
    for (c = counter; c; c = c->parent) {
        long new;
        new = atomic_long_add_return(nr_pages, &c->usage);
    }
    .......
}
```

```c
int __memcg_kmem_charge_page(struct page *page, gfp_t gfp,
   int order) {
    struct obj_cgroup *objcg;
    int ret = 0;
    objcg = get_obj_cgroup_from_current();
    if (objcg) {
        ret = obj_cgroup_charge_pages(objcg, gfp, 1 <<
        -> order);
    }
    .......
}
```
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.

```c
static int bpf_prog_load(...) {
    ....
    /* plain bpf_prog allocation */
    prog = bpf_prog_alloc(bpf_prog_size(attr->insn_cnt),
                          GFP_USER);
    ....
                     missing-account!
}

struct bpf_prog *bpf_prog_alloc(unsigned int size, gfp_t gfp_extra_flags) {
    gfp_t gfp_flags = GFP_KERNEL | __GFP_ZERO | gfp_extra_flags;
    struct bpf_prog *prog;
    int cpu;
    prog = bpf_prog_alloc_no_stats(size, gfp_extra_flags);
    if (!prog)
        return NULL;
    prog->stats = alloc_percpu_gfp(struct bpf_prog_stats,
                                   gfp_flags);
    ....
}
```
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

### 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 IDs and Source Locations

<table>
<thead>
<tr>
<th>Bug ID</th>
<th>Source Location</th>
<th>Function Name</th>
<th>Allocation Interface</th>
<th>Triggered by</th>
<th>Confirm Status</th>
<th>Patch Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch/kid/kernel/657.c:157</td>
<td>alloc_free_strct</td>
<td>kmalloc</td>
<td>clone</td>
<td>Confirmed</td>
<td>Moved</td>
<td></td>
</tr>
<tr>
<td>fs/event/evf.c:417</td>
<td>io_event_init</td>
<td>kmalloc</td>
<td>evfinit</td>
<td>Pending</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>fs/event/evf/1014</td>
<td>epoll_create</td>
<td>kmalloc</td>
<td>epoll_create</td>
<td>Pending</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>fs/1cinit.c:499</td>
<td>iostat</td>
<td>kmalloc</td>
<td>iostat</td>
<td>Confirmed</td>
<td>Moved</td>
<td></td>
</tr>
<tr>
<td>fs/cdev.c:214</td>
<td>iostat</td>
<td>kmalloc</td>
<td>iostat</td>
<td>Confirmed</td>
<td>Moved</td>
<td></td>
</tr>
<tr>
<td>fs/cdev_init.c:64</td>
<td>iostat</td>
<td>kmalloc</td>
<td>iostat</td>
<td>Confirmed</td>
<td>Moved</td>
<td></td>
</tr>
<tr>
<td>fs/cdev_topic.c:100</td>
<td>iostat</td>
<td>kmalloc</td>
<td>iostat</td>
<td>Confirmed</td>
<td>Moved</td>
<td></td>
</tr>
<tr>
<td>fs/exec.c:909</td>
<td>iostat</td>
<td>kmalloc</td>
<td>iostat</td>
<td>Confirmed</td>
<td>Moved</td>
<td></td>
</tr>
<tr>
<td>fs/exec/1114</td>
<td>iostat</td>
<td>kmalloc</td>
<td>iostat</td>
<td>Confirmed</td>
<td>Moved</td>
<td></td>
</tr>
<tr>
<td>fs/exec/topic.c:188</td>
<td>iostat</td>
<td>kmalloc</td>
<td>iostat</td>
<td>Confirmed</td>
<td>Moved</td>
<td></td>
</tr>
<tr>
<td>fs/exec/topic.c:197</td>
<td>iostat</td>
<td>kmalloc</td>
<td>iostat</td>
<td>Confirmed</td>
<td>Moved</td>
<td></td>
</tr>
<tr>
<td>fs/exec/topic.c:293</td>
<td>iostat</td>
<td>kmalloc</td>
<td>iostat</td>
<td>Confirmed</td>
<td>Moved</td>
<td></td>
</tr>
<tr>
<td>fs/exec/topic.c:385</td>
<td>iostat</td>
<td>kmalloc</td>
<td>iostat</td>
<td>Confirmed</td>
<td>Moved</td>
<td></td>
</tr>
<tr>
<td>fs/exec/topic.c:494</td>
<td>iostat</td>
<td>kmalloc</td>
<td>iostat</td>
<td>Confirmed</td>
<td>Moved</td>
<td></td>
</tr>
</tbody>
</table>

### Analysis Precision
- Analysis precision: 66.9% (162/242)

### Impact analysis
- 17 PoCs
- 5 can exhaust more than 16GB host memory.
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](https://github.com/ZJU-SEC/MANTA)