CS315 Computer Security Term Project

Docker Escape - Final Report

11712121 胡玉斌/11811902 顾同舟/11812102 黄炜杰/11812103 张文灏/11812104 石文轩

🏃 Docker Escape?

Docker escape is to **escape from the preset limitations of containerization**.

Linux provides multitudinous of limitations for containers:

- Permission limitations
  - Isolation among containers
  - Isolation between containers and host
- Resource limitations
  - CPU cores
  - Memory usage
  - Derived process numbers

Why break the permission limitations?

Hackers can take control of the host machine!

e.g. Docker containers farm (Amazon, Ali Cloud, Tecent Cloud...)

~~So it's super hard.~~

Why break the resource limitations?

In a multi-tenant container environment, an adversarial container is able to:

1. Amplify the amount of consumed resources.
2. Significantly slow-down containers on the same host. **(DoS Attack)**
3. Gain extra unfair advantages on the system resources.

Escape from permission limitation

CVE-2019-5736 (RunC Escape)

RunC Escape

Overwriting the host system’s RunC binary within a container.

Starting containers depends on RunC.

Approach

1. Overwritten RunC binary by hack's code with *CVE-2019-5736 RunC vulnerability*.
2. Waitting for a new container starting.
3. 💥

Demo in week 10.

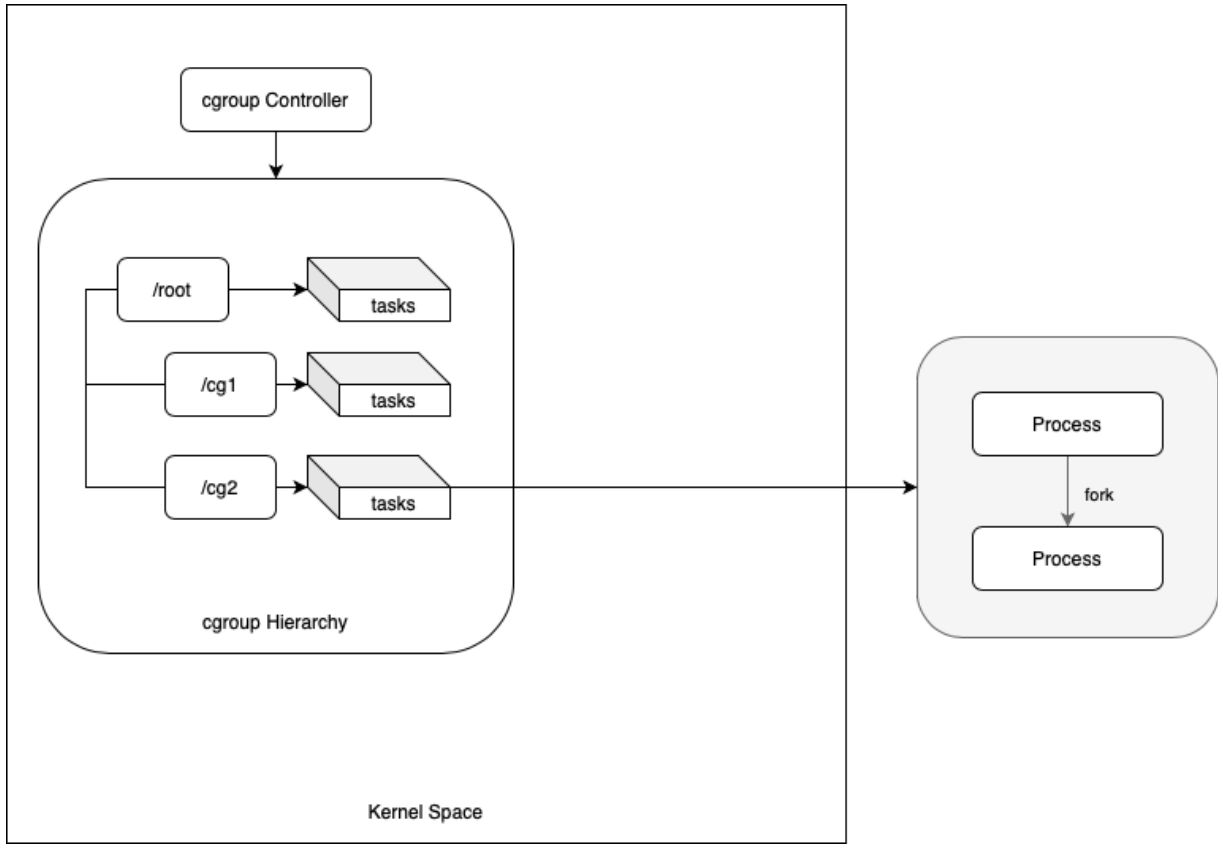
Escape from resource limitation

X. Gao, Z. Gu, Z. Li, H. Jamjoom, and C. Wang, “Houdini’s Escape: Breaking the Resource Rein of Linux Control Groups,” in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London United Kingdom, Nov. 2019, pp. 1073–1086, doi: 10.1145/3319535.3354227.

Background

Docker containers depend on **cgroups** for resource management.

Partitions a group of processes and their children into **hierarchical groups**, applies different controllers to manage and limit various system resources (CPU time, computer memory, block I/O, etc.)



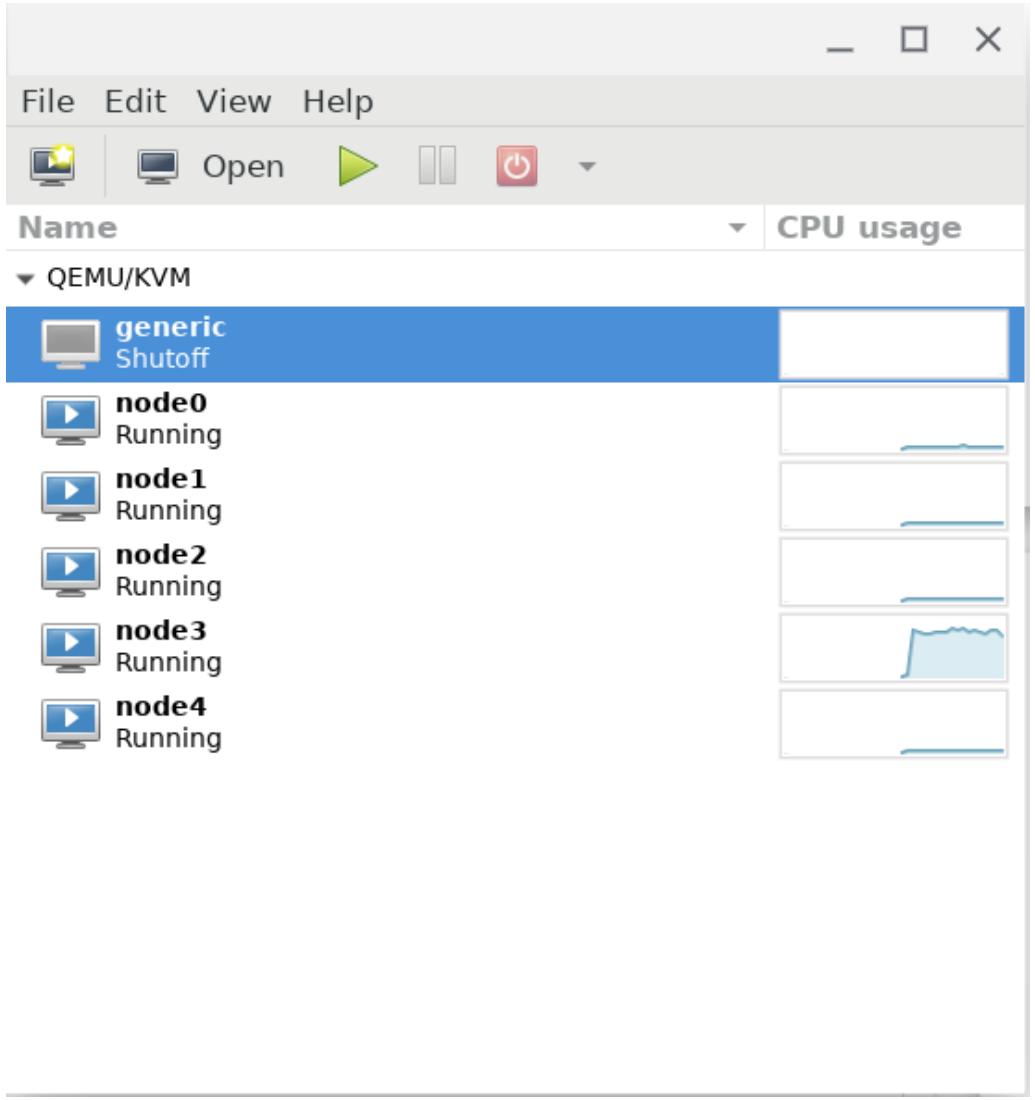
Base Principle: cgroups miscounting

We will give 5 different fancy ways to trigger cgroups miscounting. But first...

Environment Setup

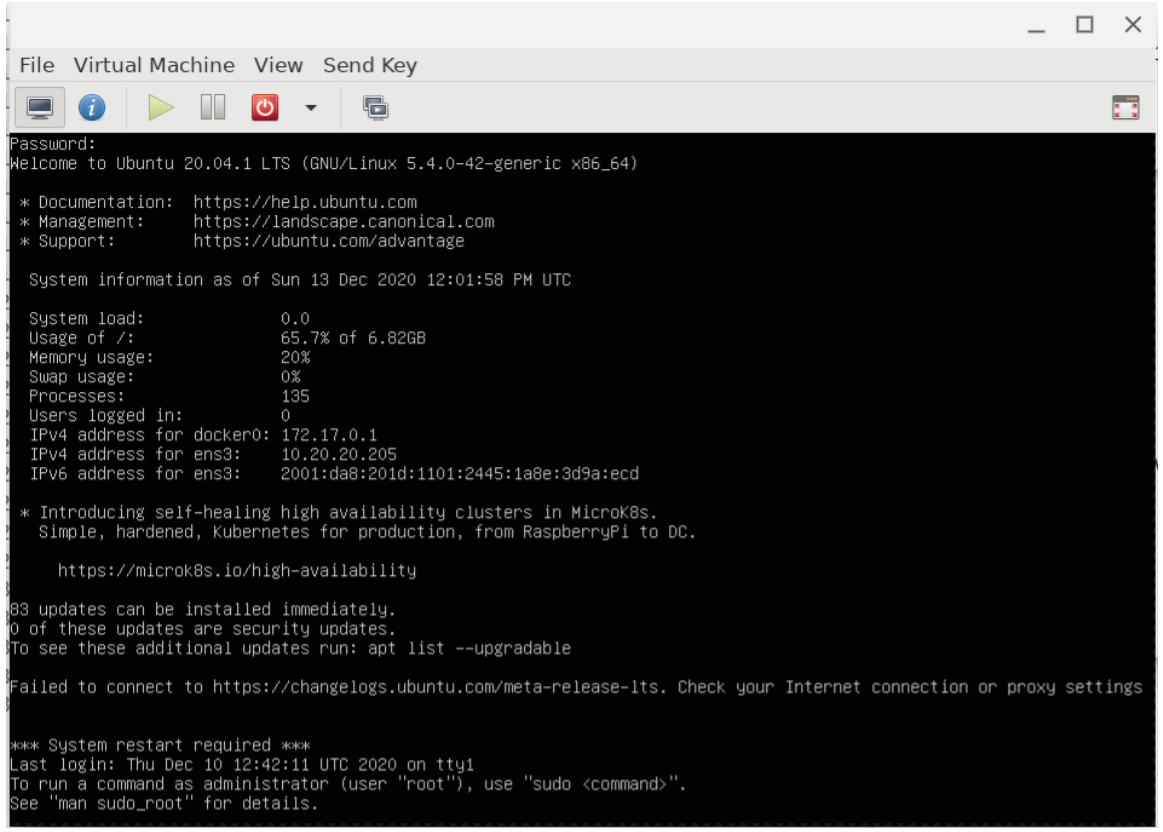
Environment Setup

- Host Configuration
  - CPU: Intel Xeon E5-2697 v4 (36 x 2.3GHz)
  - Mem: 512GB
  - Disk: 1TB SSD
  - OS: Ubuntu 18.04
  - Kernel: Linux 4.15.0



Environment Setup

- Guest Configuration
  - Hypervisor: QEMU + KVM
  - CPU: 4 Cores
  - Mem: 2GB
  - Disk: VirtIO + QCOW2
  - Net: VirtIO + NAT
  - OS: Ubuntu 20.04
  - Docker: 19.03.11

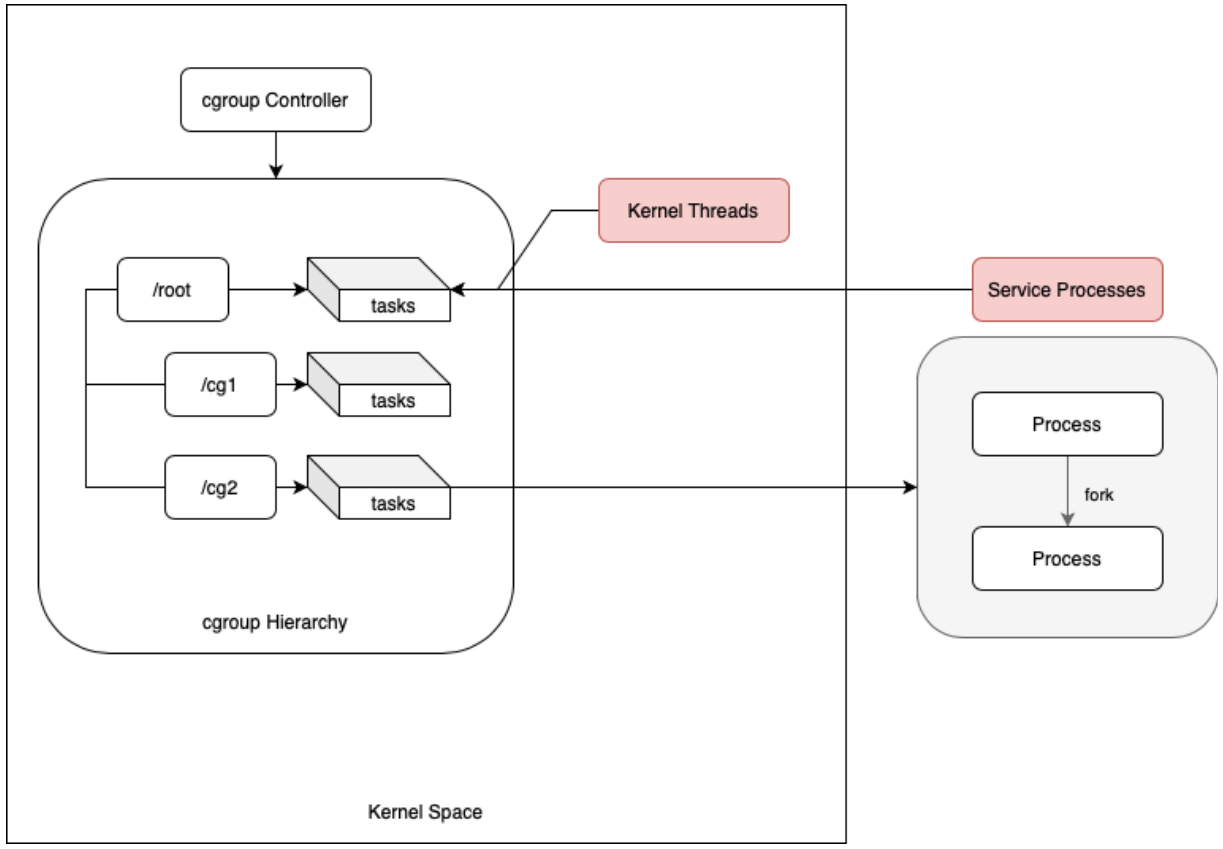


Part 1: Exploiting Service Processes

Conainer Engine

What will happen when sending command to container?

- → CLI Client (i.e. `docker`)
- → Docker REST API
- → Docker Daemon (Service Processes)
  (i.e. `dockerd`)
- → tty Driver (Kernel Threads)
  - → Executing work queue (*kworker* kernel thread)
  - → LDISC



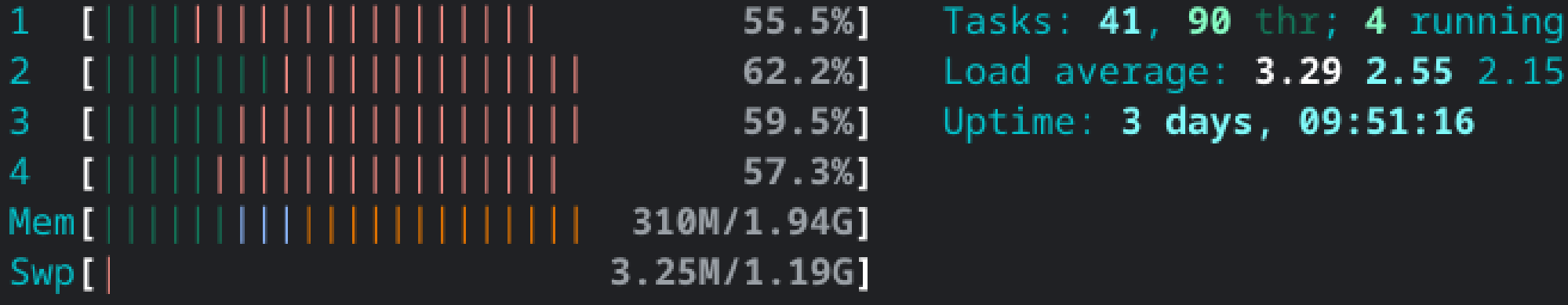
How to exploit it?

Single Line Command `for((;;)); do lsmod; done`

Repeatedly generate tons of meaningless output to stress Docker Daemon and tty Driver

Evaluation

- Create container
  - Command: `docker run -itd --name atk4 --cpus=1 ubuntu:20.04 bash`
  - CPU Limit: 100% Single Core
- Launch attack
  - Command: `docker exec -it atk4 bash`, `for((;;)); do lsmod; done`
  - Comparsion: `for((;;)); do lsmod > /dev/null; done`


1 [|||||||||||||||||| 55.5%]
2 [|||||||||||||||||||| 62.2%]
3 [|||||||||||||||||||| 59.5%]
4 [|||||||||||||||||||| 57.3%]
Mem[|||||||||||||||||| 310M/1.94G]
Swp[| 3.25M/1.19G]
Tasks: 41, 90 thr; 4 running
Load average: 3.29 2.55 2.15
Uptime: 3 days, 09:51:16
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
2302 root 20 0 991M 98176 45660 S 42.0 4.8 3h10:26 dockerd --group docker
4116956 root 20 0 395M 57812 33508 S 32.4 2.8 1:21.90 docker exec -it atk4 ba
4114570 t 20 0 13900 5516 4024 S 32.4 0.3 1:13.10 sshd: t@pts/1
303970 root 20 0 106M 6940 4280 S 44.1 0.3 5h13:02 containerd-shim -namesp
4117042 root 20 0 395M 57812 33508 S 0.0 2.8 0:23.12 docker exec -it atk4 ba
2376 root 20 0 991M 98176 45660 S 9.0 4.8 14:14.18 dockerd --group docker
4116991 root 20 0 395M 57812 33508 S 9.6 2.8 0:25.16 docker exec -it atk4 ba
303978 root 20 0 106M 6940 4280 S 0.0 0.3 46:32.74 containerd-shim -namesp
2371 root 20 0 991M 98176 45660 S 10.3 4.8 52:25.74 dockerd --group docker
2450 root 20 0 991M 98176 45660 S 5.5 4.8 14:32.68 dockerd --group docker
2434 root 20 0 991M 98176 45660 S 0.0 4.8 15:41.41 dockerd --group docker
304024 root 20 0 106M 6940 4280 S 10.3 0.3 39:13.10 containerd-shim -namesp
2372 root 20 0 991M 98176 45660 S 6.2 4.8 13:56.40 dockerd --group docker

ss

```
  1  [|||||||                    19.5%]   Tasks: 38, 80 thr; 1 running
  2  [|||||||                    17.8%]   Load average: 1.06 0.94 0.98
  3  [|||||||||||||||            45.3%]   Uptime: 3 days, 10:37:35
  4  [||||||||                   22.0%]
  Mem[|||||||||||||||||||    291M/1.94G]
  Swp[|                     3.25M/1.19G]
```

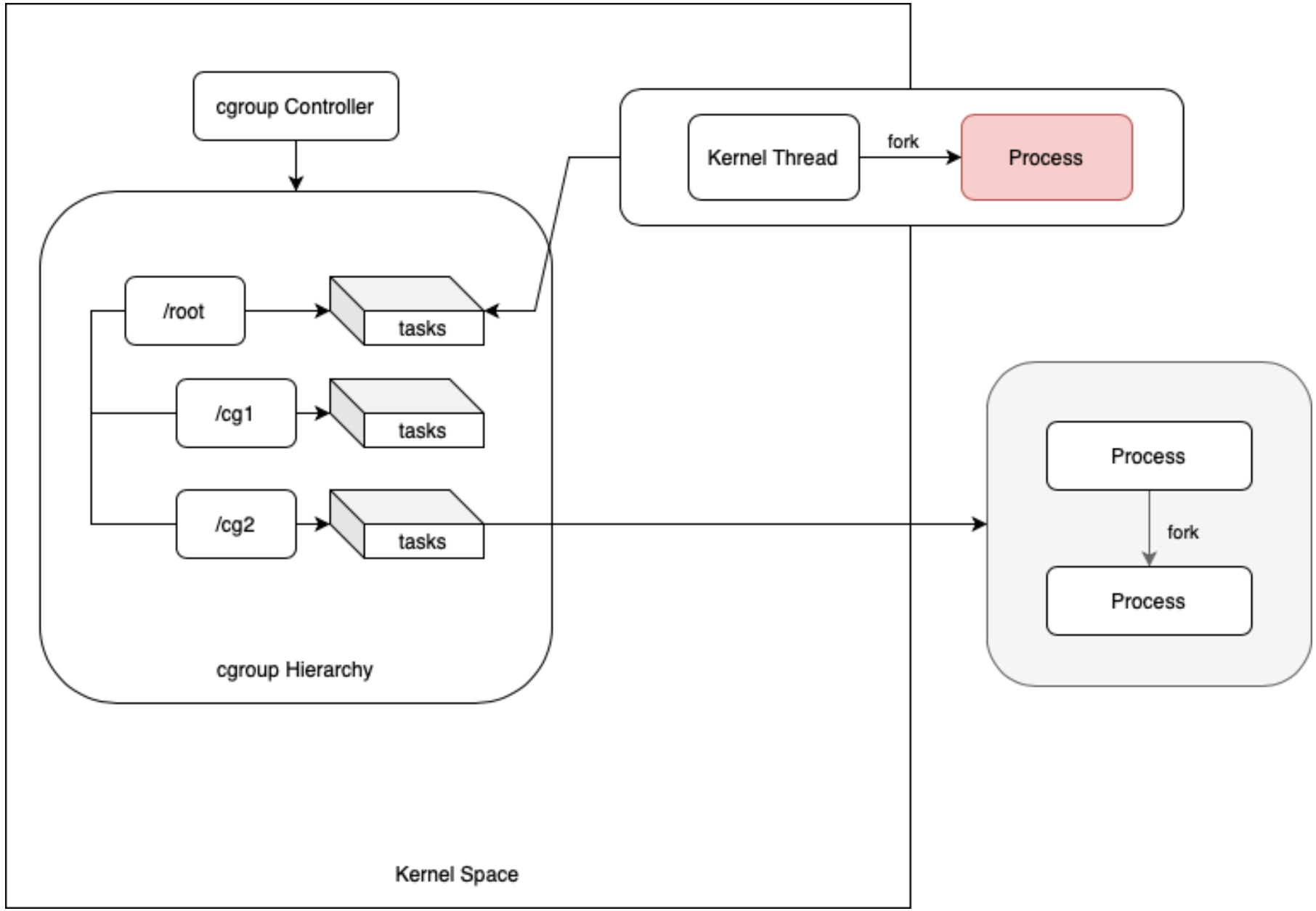
PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
4093965	t	20	0	8276	5156	3460	S	10.0	0.3	4:13.55	-bash
431652	t	20	0	7948	3984	3368	R	0.7	0.2	0:00.13	htop
1716870	t	20	0	8272	4304	3368	S	0.7	0.2	2:36.09	htop
2380	root	20	0	901M	46180	24444	S	0.0	2.3	25:54.64	containerd --config /va
2302	root	20	0	991M	101M	45660	S	0.0	5.1	3h11:52	dockerd --group docker
2371	root	20	0	991M	101M	45660	S	0.0	5.1	52:43.11	dockerd --group docker
122513	root	20	0	901M	46180	24444	S	0.0	2.3	0:09.79	containerd --config /va
819	root	20	0	232M	7820	6796	S	0.0	0.4	0:06.00	/usr/lib/accountsservic
817	root	20	0	232M	7820	6796	S	0.0	0.4	0:06.06	/usr/lib/accountsservic
137953	root	20	0	901M	46180	24444	S	0.0	2.3	0:09.66	containerd --config /va
139171	root	20	0	901M	46180	24444	S	0.0	2.3	0:10.75	containerd --config /va
554	root	RT	0	273M	17952	8184	S	0.0	0.9	0:27.15	/sbin/multipathd -d -s
2415	root	20	0	901M	46180	24444	S	0.0	2.3	4:48.45	containerd --config /va

Result

Occupy about **250%** resource

Part 2: Exploiting Processes Forked by Kernel Thread

Exception Handling: Core Dump


cgroup Controller
Kernel Thread
fork
Process
/root
tasks
/cg1
tasks
/cg2
tasks
cgroup Hierarchy
Process
fork
Process
Kernel Space

Exception and Core Dump

- `Core Dump` is a kernel function.
- When exception happening, the process will halt, and invoke the `Core Dump` function.
- The `Core Dump` kernel function will invoke the core dump application in **usermode**.

How to use it?

Core Dump application is created by kernel thread, rather than container. Thus it won't be controlled by cgroup's resouce limit.

In Ubuntu, the default core dump application is Apport. Apport produces big dump files.

Attack Strategy

1. Enable Core Dump in container.
2. Run `div 0` continuously in malicious container.

Apport will continuously generate core dump files.

Experiment

- One container attached to CPU core 0, limited with CPU usage 5%.
- Measure CPU usage.

15% | 11 GB | 20 kB↓ 5.1 kB↑ | Send Snippet... | Command

```
CONTAINER ID   NAME              CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O   P
IDS
50fdfd7ccf69   docker_escape_1   1.17%     2.746MiB / 31.41GiB   0.01%     8.6kB / 0B    0B / 0B     5
a012b23b20a1   docker_escape     0.00%     1.531MiB / 31.41GiB   0.00%     10.6kB / 0B   0B / 0B     1
```

```
docker run -itd --name=docker_escape --cpuset-cpus="0" --cpu-
shares=100 -v ./:/data/docker_sahre ubuntu:18.04
```

```
  1  [||                  4.5%]   9  [|                  2.0%]   17 [|                  2.6%]   25 [                  0.0%]
  2  [||||||||||       52.6%]   10 [                    0.0%]   18 [|                  2.0%]   26 [||                 2.6%]
  3  [||||||||         46.1%]   11 [|                  2.6%]   19 [|                  2.0%]   27 [|                  0.7%]
  4  [                  0.0%]   12 [|                  2.0%]   20 [||                 2.0%]   28 [|                  2.0%]
  5  [|                 2.0%]   13 [                    0.0%]   21 [|                  2.0%]   29 [|                  0.7%]
  6  [                  0.0%]   14 [                    0.0%]   22 [                    0.0%]   30 [|                  2.0%]
  7  [                  0.0%]   15 [                    0.0%]   23 [                    0.0%]   31 [|                  2.0%]
  8  [                  0.0%]   16 [|                   0.7%]   24 [|                  2.0%]   32 [||                 3.3%]
  Mem[|||||||                  1.41G/31.4G]   Tasks: 126, 379 thr; 18 running
  Swp[                         0K/975M]   Load average: 0.52 0.33 0.33
                                          Uptime: 1 day, 00:36:41
```

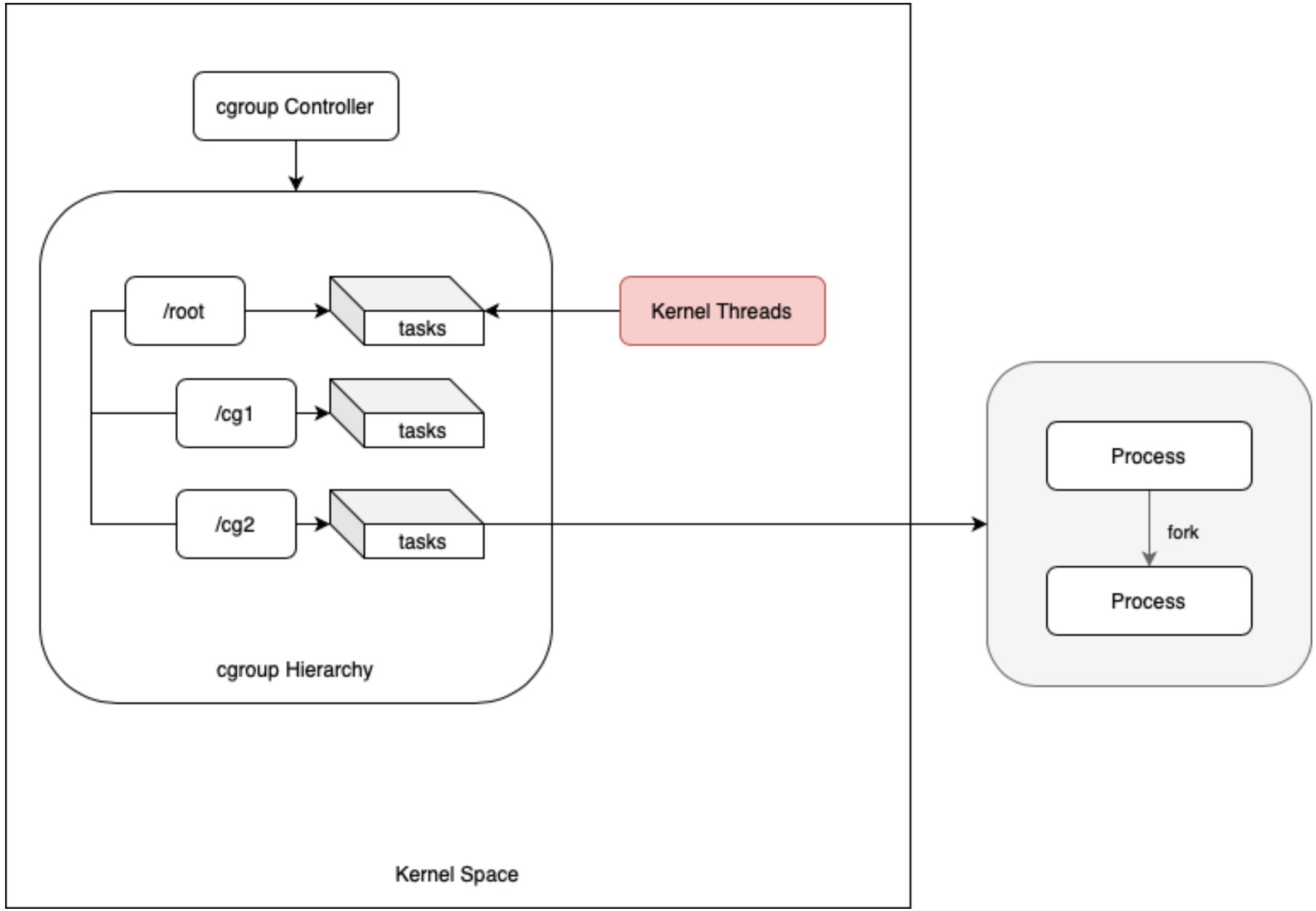
PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
10758	root	20	0	2004M	106M	53256	S	0.0	0.3	0:07.45	/usr/bin/dockerd -H fd:// --containerd
10759	root	20	0	2004M	106M	53256	S	0.0	0.3	0:00.00	/usr/bin/dockerd -H fd:// --containerd
10760	root	20	0	2004M	106M	53256	S	0.0	0.3	0:01.76	/usr/bin/dockerd -H fd:// --containerd
10761	root	20	0	2004M	106M	53256	S	0.0	0.3	0:01.44	/usr/bin/dockerd -H fd:// --containerd
10762	root	20	0	2004M	106M	53256	S	0.0	0.3	0:00.00	/usr/bin/dockerd -H fd:// --containerd
10764	root	20	0	2004M	106M	53256	S	0.0	0.3	0:02.18	/usr/bin/dockerd -H fd:// --containerd
10767	root	20	0	2004M	106M	53256	S	0.0	0.3	0:01.29	/usr/bin/dockerd -H fd:// --containerd
10768	root	20	0	2004M	106M	53256	S	0.0	0.3	0:02.26	/usr/bin/dockerd -H fd:// --containerd
10769	root	20	0	2004M	106M	53256	S	0.0	0.3	0:02.44	/usr/bin/dockerd -H fd:// --containerd
10770	root	20	0	2004M	106M	53256	S	1.3	0.3	0:01.66	/usr/bin/dockerd -H fd:// --containerd
10771	root	20	0	2004M	106M	53256	S	0.0	0.3	0:01.71	/usr/bin/dockerd -H fd:// --containerd
10772	root	20	0	2004M	106M	53256	S	0.0	0.3	0:01.29	/usr/bin/dockerd -H fd:// --containerd
10773	root	20	0	2004M	106M	53256	S	0.0	0.3	0:00.76	/usr/bin/dockerd -H fd:// --containerd
10774	root	20	0	2004M	106M	53256	S	0.0	0.3	0:08.79	/usr/bin/dockerd -H fd:// --containerd
10775	root	20	0	2004M	106M	53256	S	0.7	0.3	0:01.04	/usr/bin/dockerd -H fd:// --containerd
10776	root	20	0	2004M	106M	53256	S	0.0	0.3	0:02.63	/usr/bin/dockerd -H fd:// --containerd
10777	root	20	0	2004M	106M	53256	S	0.0	0.3	0:02.04	/usr/bin/dockerd -H fd:// --containerd
10778	root	20	0	2004M	106M	53256	S	0.0	0.3	0:01.67	/usr/bin/dockerd -H fd:// --containerd
10779	root	20	0	2004M	106M	53256	S	0.0	0.3	0:02.54	/usr/bin/dockerd -H fd:// --containerd
10780	root	20	0	2004M	106M	53256	S	0.0	0.3	0:02.56	/usr/bin/dockerd -H fd:// --containerd
10781	root	20	0	2004M	106M	53256	S	0.0	0.3	0:01.19	/usr/bin/dockerd -H fd:// --containerd
10782	root	20	0	2004M	106M	53256	S	0.0	0.3	0:02.81	/usr/bin/dockerd -H fd:// --containerd
10783	root	20	0	2004M	106M	53256	S	0.0	0.3	0:02.19	/usr/bin/dockerd -H fd:// --containerd
10784	root	20	0	2004M	106M	53256	S	0.7	0.3	0:02.46	/usr/bin/dockerd -H fd:// --containerd
11040	root	20	0	2004M	106M	53256	S	0.0	0.3	0:00.72	/usr/bin/dockerd -H fd:// --containerd
11045	root	20	0	2004M	106M	53256	S	0.0	0.3	0:02.22	/usr/bin/dockerd -H fd:// --containerd
11046	root	20	0	2004M	106M	53256	S	0.0	0.3	0:01.11	/usr/bin/dockerd -H fd:// --containerd
11751	root	20	0	2004M	106M	53256	S	0.0	0.3	0:04.09	/usr/bin/dockerd -H fd:// --containerd
12113	root	20	0	2004M	106M	53256	S	0.0	0.3	0:01.61	/usr/bin/dockerd -H fd:// --containerd
12114	root	20	0	2004M	106M	53256	S	0.0	0.3	0:01.05	/usr/bin/dockerd -H fd:// --containerd
12115	root	20	0	2004M	106M	53256	S	0.0	0.3	0:00.68	/usr/bin/dockerd -H fd:// --containerd
12116	root	20	0	2004M	106M	53256	S	0.0	0.3	0:01.12	/usr/bin/dockerd -H fd:// --containerd
12117	root	20	0	2004M	106M	53256	S	0.0	0.3	0:00.97	/usr/bin/dockerd -H fd:// --containerd
12118	root	20	0	2004M	106M	53256	S	0.0	0.3	0:00.65	/usr/bin/dockerd -H fd:// --containerd
12119	root	20	0	2004M	106M	53256	S	0.0	0.3	0:00.82	/usr/bin/dockerd -H fd:// --containerd

Result

CPU usage increased: 2000%

Part 3: Exploiting Kernel Thread

Data Synchronization


cgroup Controller
/root
tasks
/cg1
tasks
/cg2
tasks
Kernel Threads
cgroup Hierarchy
Process
fork
Process
Kernel Space

Writeback Strategy

- Cached data are written back only when the page is **evicted**
- Separate processes for **synchronization** and **I/O**
  (Could be utilized to escape from `cgroup`)

System Call: `sync`

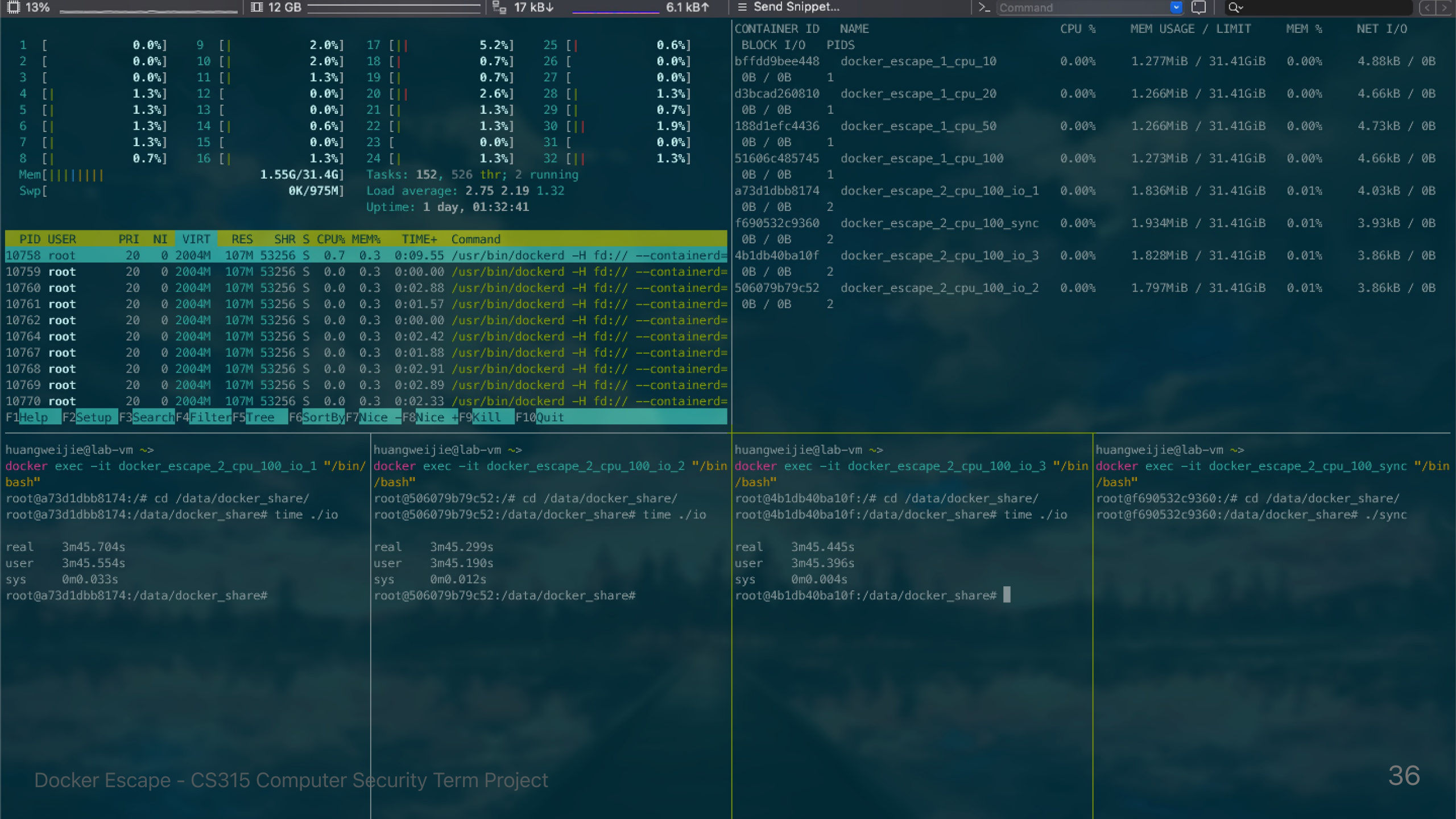
- Create kernel thread
- Scan the entire file system (not only the container, but the **host OS**)
- Force writing all **dirty** pages back into disk

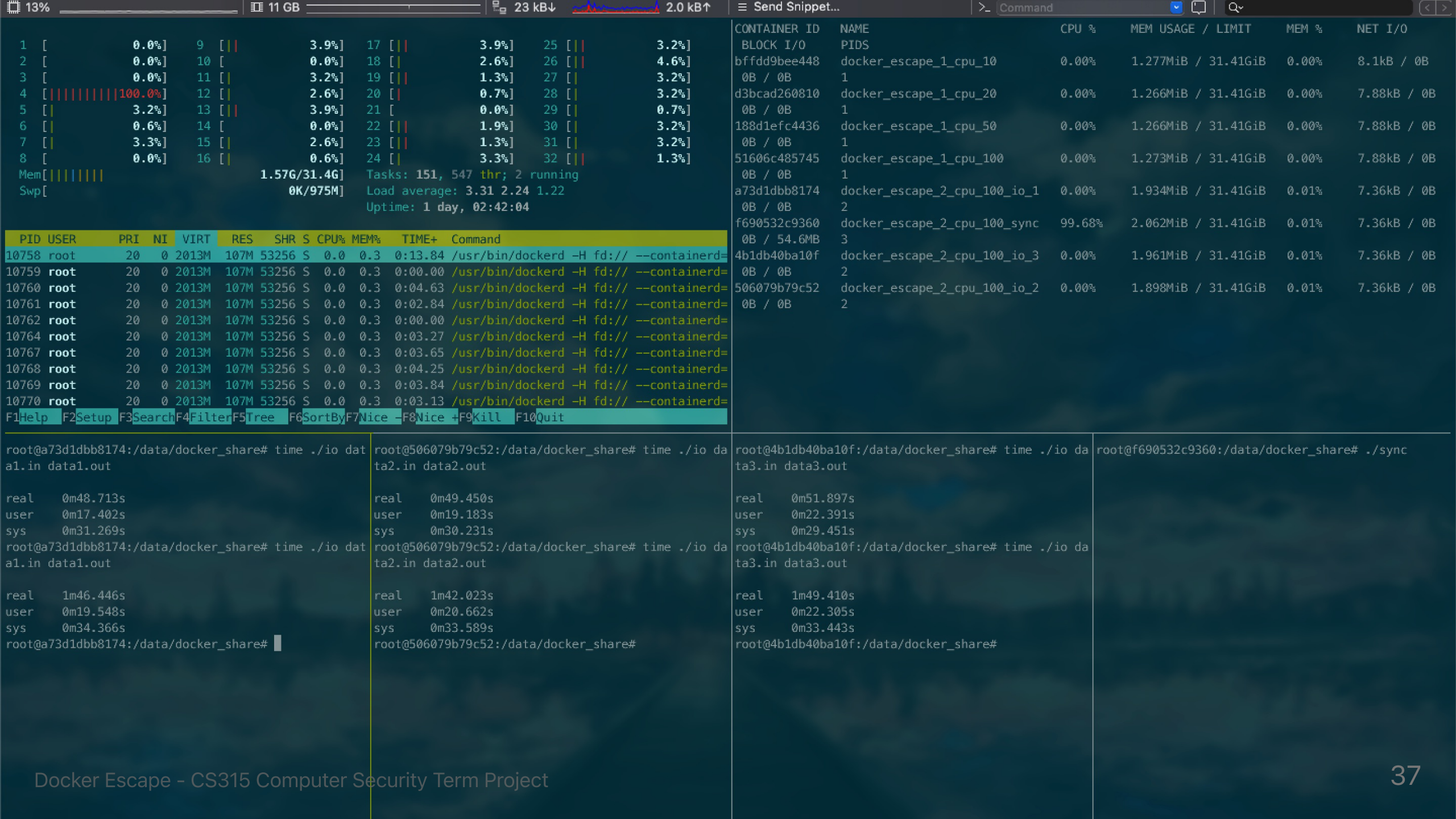
Attack Strategy

- Call `sync` continuously in malicious container
- I/O sentitive tasks will be affected

Experiment

- Three containers performing file I/O
- One malicious container
- Each container restricted to a disparate CPU core
- Measure time consumption and CPU usage with/without the malicious system call (`time`, `htop`)


13% 12 GB 17 kB↓ 6.1 kB↑ Send Snippet... Command
1 [0.0%] 9 [| 2.0%] 17 [|| 5.2%] 25 [| 0.6%]
2 [0.0%] 10 [| 2.0%] 18 [| 0.7%] 26 [0.0%]
3 [0.0%] 11 [| 1.3%] 19 [| 0.7%] 27 [0.0%]
4 [| 1.3%] 12 [0.0%] 20 [|| 2.6%] 28 [| 1.3%]
5 [| 1.3%] 13 [0.0%] 21 [| 1.3%] 29 [| 0.7%]
6 [| 1.3%] 14 [| 0.6%] 22 [| 1.3%] 30 [|| 1.9%]
7 [| 1.3%] 15 [0.0%] 23 [0.0%] 31 [0.0%]
8 [| 0.7%] 16 [| 1.3%] 24 [| 1.3%] 32 [|| 1.3%]
Mem[|||||||| 1.55G/31.4G]
Swp[0K/975M]
Tasks: 152, 526 thr; 2 running
Load average: 2.75 2.19 1.32
Uptime: 1 day, 01:32:41
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
10758 root 20 0 2004M 107M 53256 S 0.7 0.3 0:09.55 /usr/bin/dockerd -H fd:// --containerd=
10759 root 20 0 2004M 107M 53256 S 0.0 0.3 0:00.00 /usr/bin/dockerd -H fd:// --containerd=
10760 root 20 0 2004M 107M 53256 S 0.0 0.3 0:02.88 /usr/bin/dockerd -H fd:// --containerd=
10761 root 20 0 2004M 107M 53256 S 0.0 0.3 0:01.57 /usr/bin/dockerd -H fd:// --containerd=
10762 root 20 0 2004M 107M 53256 S 0.0 0.3 0:00.00 /usr/bin/dockerd -H fd:// --containerd=
10764 root 20 0 2004M 107M 53256 S 0.0 0.3 0:02.42 /usr/bin/dockerd -H fd:// --containerd=
10767 root 20 0 2004M 107M 53256 S 0.0 0.3 0:01.88 /usr/bin/dockerd -H fd:// --containerd=
10768 root 20 0 2004M 107M 53256 S 0.0 0.3 0:02.91 /usr/bin/dockerd -H fd:// --containerd=
10769 root 20 0 2004M 107M 53256 S 0.0 0.3 0:02.89 /usr/bin/dockerd -H fd:// --containerd=
10770 root 20 0 2004M 107M 53256 S 0.0 0.3 0:02.33 /usr/bin/dockerd -H fd:// --containerd=
F1Help F2Setup F3SearchF4FilterF5Tree F6SortByF7Nice -F8Nice +F9Kill F10Quit
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O
BLOCK I/O PIDS
bffdd9bee448 docker_escape_1_cpu_10 0.00% 1.277MiB / 31.41GiB 0.00% 4.88kB / 0B
0B / 0B 1
d3bcad260810 docker_escape_1_cpu_20 0.00% 1.266MiB / 31.41GiB 0.00% 4.66kB / 0B
0B / 0B 1
188d1efc4436 docker_escape_1_cpu_50 0.00% 1.266MiB / 31.41GiB 0.00% 4.73kB / 0B
0B / 0B 1
51606c485745 docker_escape_1_cpu_100 0.00% 1.273MiB / 31.41GiB 0.00% 4.66kB / 0B
0B / 0B 1
a73d1dbb8174 docker_escape_2_cpu_100_io_1 0.00% 1.836MiB / 31.41GiB 0.01% 4.03kB / 0B
0B / 0B 2
f690532c9360 docker_escape_2_cpu_100_sync 0.00% 1.934MiB / 31.41GiB 0.01% 3.93kB / 0B
0B / 0B 2
4b1db40ba10f docker_escape_2_cpu_100_io_3 0.00% 1.828MiB / 31.41GiB 0.01% 3.86kB / 0B
0B / 0B 2
506079b79c52 docker_escape_2_cpu_100_io_2 0.00% 1.797MiB / 31.41GiB 0.01% 3.86kB / 0B
0B / 0B 2
huangweijie@lab-vm ~>
docker exec -it docker_escape_2_cpu_100_io_1 "/bin/bash"
root@a73d1dbb8174:/# cd /data/docker_share/
root@a73d1dbb8174:/data/docker_share# time ./io
real 3m45.704s
user 3m45.554s
sys 0m0.033s
root@a73d1dbb8174:/data/docker_share#
huangweijie@lab-vm ~>
docker exec -it docker_escape_2_cpu_100_io_2 "/bin/bash"
root@506079b79c52:/# cd /data/docker_share/
root@506079b79c52:/data/docker_share# time ./io
real 3m45.299s
user 3m45.190s
sys 0m0.012s
root@506079b79c52:/data/docker_share#
huangweijie@lab-vm ~>
docker exec -it docker_escape_2_cpu_100_io_3 "/bin/bash"
root@4b1db40ba10f:/# cd /data/docker_share/
root@4b1db40ba10f:/data/docker_share# time ./io
real 3m45.445s
user 3m45.396s
sys 0m0.004s
root@4b1db40ba10f:/data/docker_share#
huangweijie@lab-vm ~>
docker exec -it docker_escape_2_cpu_100_sync "/bin/bash"
root@f690532c9360:/# cd /data/docker_share/
root@f690532c9360:/data/docker_share# ./sync


13%
11 GB
23 kB↓
2.0 kB↑
Send Snippet...
Command
1 [0.0%] 9 [|| 3.9%] 17 [|| 3.9%] 25 [|| 3.2%]
2 [0.0%] 10 [0.0%] 18 [| 2.6%] 26 [|| 4.6%]
3 [0.0%] 11 [| 3.2%] 19 [|| 1.3%] 27 [| 3.2%]
4 [|||||||||100.0%] 12 [| 2.6%] 20 [| 0.7%] 28 [| 3.2%]
5 [| 3.2%] 13 [|| 3.9%] 21 [0.0%] 29 [| 0.7%]
6 [| 0.6%] 14 [0.0%] 22 [|| 1.9%] 30 [| 3.2%]
7 [| 3.3%] 15 [| 2.6%] 23 [|| 1.3%] 31 [| 3.2%]
8 [0.0%] 16 [| 0.6%] 24 [| 3.3%] 32 [|| 1.3%]
Mem[|||||||| 1.57G/31.4G]
Swp[0K/975M]
Tasks: 151, 547 thr; 2 running
Load average: 3.31 2.24 1.22
Uptime: 1 day, 02:42:04
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
10758 root 20 0 2013M 107M 53256 S 0.0 0.3 0:13.84 /usr/bin/dockerd -H fd:// --containerd=
10759 root 20 0 2013M 107M 53256 S 0.0 0.3 0:00.00 /usr/bin/dockerd -H fd:// --containerd=
10760 root 20 0 2013M 107M 53256 S 0.0 0.3 0:04.63 /usr/bin/dockerd -H fd:// --containerd=
10761 root 20 0 2013M 107M 53256 S 0.0 0.3 0:02.84 /usr/bin/dockerd -H fd:// --containerd=
10762 root 20 0 2013M 107M 53256 S 0.0 0.3 0:00.00 /usr/bin/dockerd -H fd:// --containerd=
10764 root 20 0 2013M 107M 53256 S 0.0 0.3 0:03.27 /usr/bin/dockerd -H fd:// --containerd=
10767 root 20 0 2013M 107M 53256 S 0.0 0.3 0:03.65 /usr/bin/dockerd -H fd:// --containerd=
10768 root 20 0 2013M 107M 53256 S 0.0 0.3 0:04.25 /usr/bin/dockerd -H fd:// --containerd=
10769 root 20 0 2013M 107M 53256 S 0.0 0.3 0:03.84 /usr/bin/dockerd -H fd:// --containerd=
10770 root 20 0 2013M 107M 53256 S 0.0 0.3 0:03.13 /usr/bin/dockerd -H fd:// --containerd=
F1Help F2Setup F3SearchF4FilterF5Tree F6SortByF7Nice -F8Nice +F9Kill F10Quit
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O
BLOCK I/O PIDS
bffdd9bee448 docker_escape_1_cpu_10 0.00% 1.277MiB / 31.41GiB 0.00% 8.1kB / 0B
0B / 0B 1
d3bcad260810 docker_escape_1_cpu_20 0.00% 1.266MiB / 31.41GiB 0.00% 7.88kB / 0B
0B / 0B 1
188d1efc4436 docker_escape_1_cpu_50 0.00% 1.266MiB / 31.41GiB 0.00% 7.88kB / 0B
0B / 0B 1
51606c485745 docker_escape_1_cpu_100 0.00% 1.273MiB / 31.41GiB 0.00% 7.88kB / 0B
0B / 0B 1
a73d1dbb8174 docker_escape_2_cpu_100_io_1 0.00% 1.934MiB / 31.41GiB 0.01% 7.36kB / 0B
0B / 0B 2
f690532c9360 docker_escape_2_cpu_100_sync 99.68% 2.062MiB / 31.41GiB 0.01% 7.36kB / 0B
0B / 54.6MB 3
4b1db40ba10f docker_escape_2_cpu_100_io_3 0.00% 1.961MiB / 31.41GiB 0.01% 7.36kB / 0B
0B / 0B 2
506079b79c52 docker_escape_2_cpu_100_io_2 0.00% 1.898MiB / 31.41GiB 0.01% 7.36kB / 0B
0B / 0B 2
root@a73d1dbb8174:/data/docker_share# time ./io data1.in data1.out
real 0m48.713s
user 0m17.402s
sys 0m31.269s
root@a73d1dbb8174:/data/docker_share# time ./io data1.in data1.out
real 1m46.446s
user 0m19.548s
sys 0m34.366s
root@a73d1dbb8174:/data/docker_share#
root@506079b79c52:/data/docker_share# time ./io data2.in data2.out
real 0m49.450s
user 0m19.183s
sys 0m30.231s
root@506079b79c52:/data/docker_share# time ./io data2.in data2.out
real 1m42.023s
user 0m20.662s
sys 0m33.589s
root@506079b79c52:/data/docker_share#
root@4b1db40ba10f:/data/docker_share# time ./io data3.in data3.out
real 0m51.897s
user 0m22.391s
sys 0m29.451s
root@4b1db40ba10f:/data/docker_share# time ./io data3.in data3.out
real 1m49.410s
user 0m22.305s
sys 0m33.443s
root@4b1db40ba10f:/data/docker_share#
root@f690532c9360:/data/docker_share# ./sync

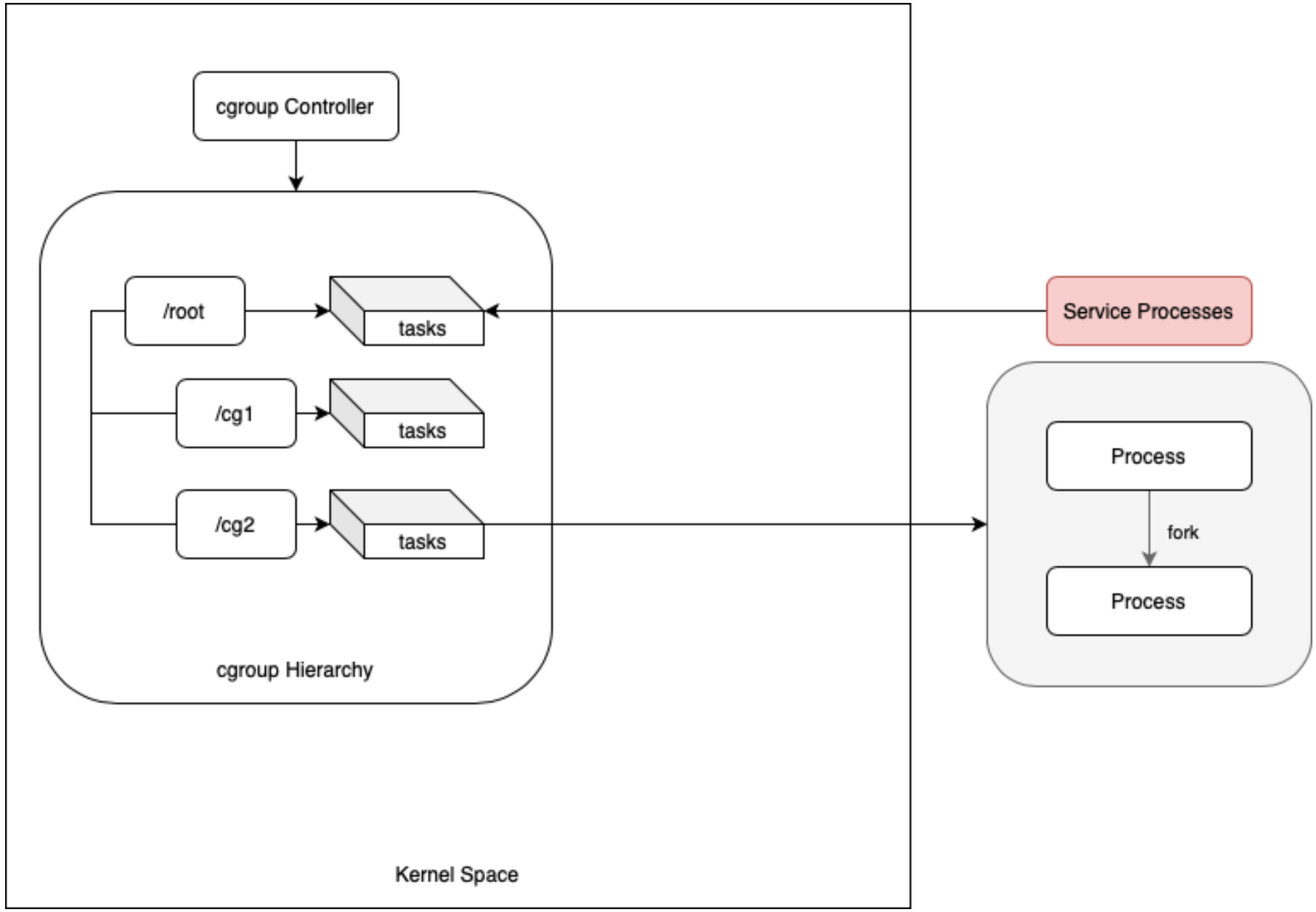
Result

Time consumption: ~200%

CPU usage: ~50%

Part 4: Exploiting Service Processes

System Process Journald


cgroup Controller
/root
tasks
/cg1
tasks
/cg2
tasks
cgroup Hierarchy
Service Processes
Process
fork
Process
Kernel Space

Systemd-Journald Service

- Provides a system service to collect system **logging** data, including kernel log messages, system log messages
- Three categories of operations in a container can **force** the journald process to log

Three categories of operations

- Switch user (su) command
- Add user/group
- Exception

Attack Strategy

- System processes of the host are attached to cgroups that are **different** from the processes within containers
- Workloads inside containers can trigger activities for those system processes will **not** be charged to containers' cgroups

Experiment

- Stop `systemd-journald` service
- Three containers performing file I/O
- Start `systemd-journald` service
- Rerun three I/O containers, and run malicious containers
- Measure time consumotion and CPU usage with/without the malicious system call

tmux /home/huangweijie (ssh) ⌘1 | t@t: ~ (ssh) ⌘2

30% | 11 GB | 4.0 MB↓ 31 kB↑ | Send Snippet... | Command

```
CONTAINER ID        NAME                          CPU %               MEM USAGE / LIMIT     MEM %
        NET I/O             BLOCK I/O           PIDS
8fcd9f63ed56        docker_escape_3_journald      0.00%               33.82MiB / 1.941GiB   1.70%
        37.3MB / 943kB      1.85MB / 7.78GB     1
06c217c2b151        docker_escape_3_io_3          0.00%               1.742MiB / 1.941GiB   0.09%
        1.3kB / 0B          2.56MB / 15.3MB     2
55838dfcf229        docker_escape_3_io_2          0.00%               1.75MiB / 1.941GiB    0.09%
        1.3kB / 0B          2.56MB / 11.5MB     2
362e4dbc14cd        docker_escape_3_io_1          0.00%               1.758MiB / 1.941GiB   0.09%
        1.3kB / 0B          2.56MB / 14.4MB     2
8e2b7fab9926        docker_escape_3_journald_1    104.41%             7.262MiB / 1.941GiB   0.37%
        1.08kB / 0B         311kB / 0B          4
```

```
  1  [||||||||||||||||||||||||||||||||100.0%]   Tasks: 63, 172 thr; 4 running
  2  [||||||||||||||||||||              53.2%]   Load average: 4.15 1.79 0.83
  3  [||||||||||||||||                  44.6%]   Uptime: 04:06:36
  4  [||||||||||||                      33.1%]
Mem[||||||||||||||||||||||||||428M/1.94G]
Swp[|                          1.01M/1.19G]

    PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
  15300 root       20   0  172M  9860  5744 S 104.  0.5  1:22.97 python3 try.py
   7890 t          20   0  9568  5872  3072 S 39.3  0.3  0:43.86 tmux new -s cs315
   2315 root       20   0  974M 97592 47132 S 28.8  4.8  1:55.17 dockerd --group docker --exec-root=/
  14907 root       20   0  106M  6448  4648 S 25.3  0.3  0:22.94 containerd-shim -namespace moby -wor
  15250 root       20   0  459M 58704 34480 S 18.9  2.9  0:17.76 docker exec -it 8e2b /bin/bash
   3268 t          20   0 15024  6536  3820 R 17.5  0.3  0:18.56 sshd: t@pts/0
  15279 root       20   0  459M 58704 34480 S 13.3  2.9  0:04.27 docker exec -it 8e2b /bin/bash
  14915 root       20   0  106M  6448  4648 S  7.0  0.3  0:02.89 containerd-shim -namespace moby -wor
   2382 root       20   0  974M 97592 47132 S  7.0  4.8  0:18.43 dockerd --group docker --exec-root=/
   2388 root       20   0  974M 97592 47132 R  7.0  4.8  0:11.02 dockerd --group docker --exec-root=/
  15274 root       20   0  459M 58704 34480 S  6.3  2.9  0:06.07 docker exec -it 8e2b /bin/bash
  14911 root       20   0  106M  6448  4648 S  6.3  0.3  0:03.07 containerd-shim -namespace moby -wor
   2564 root       20   0  974M 97592 47132 S  4.9  4.8  0:07.39 dockerd --group docker --exec-root=/
  15234 root       20   0  106M  6448  4648 S  4.9  0.3  0:03.09 containerd-shim -namespace moby -wor
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

```
root@362e4dbc14cd:~# time ./io data1.in data1.out

real    0m47.413s
user    0m21.734s
sys     0m25.563s
root@362e4dbc14cd:~# time ./io data1.in data1.out

real    1m6.999s
user    0m21.727s
sys     0m30.747s
root@362e4dbc14cd:~#
```

```
root@55838dfcf229:~# time ./io data2.in data2.out

real    0m45.638s
user    0m21.559s
sys     0m23.971s
root@55838dfcf229:~# time ./io data2.in data2.out

real    1m11.465s
user    0m22.819s
sys     0m34.295s
root@55838dfcf229:~#
```

```
root@06c217c2b151:~# time ./io data3.in data3.out

real    0m48.033s
user    0m21.958s
sys     0m25.977s
root@06c217c2b151:~# time ./io data3.in data3.out

real    1m23.279s
user    0m23.771s
sys     0m33.087s
root@06c217c2b151:~#
```

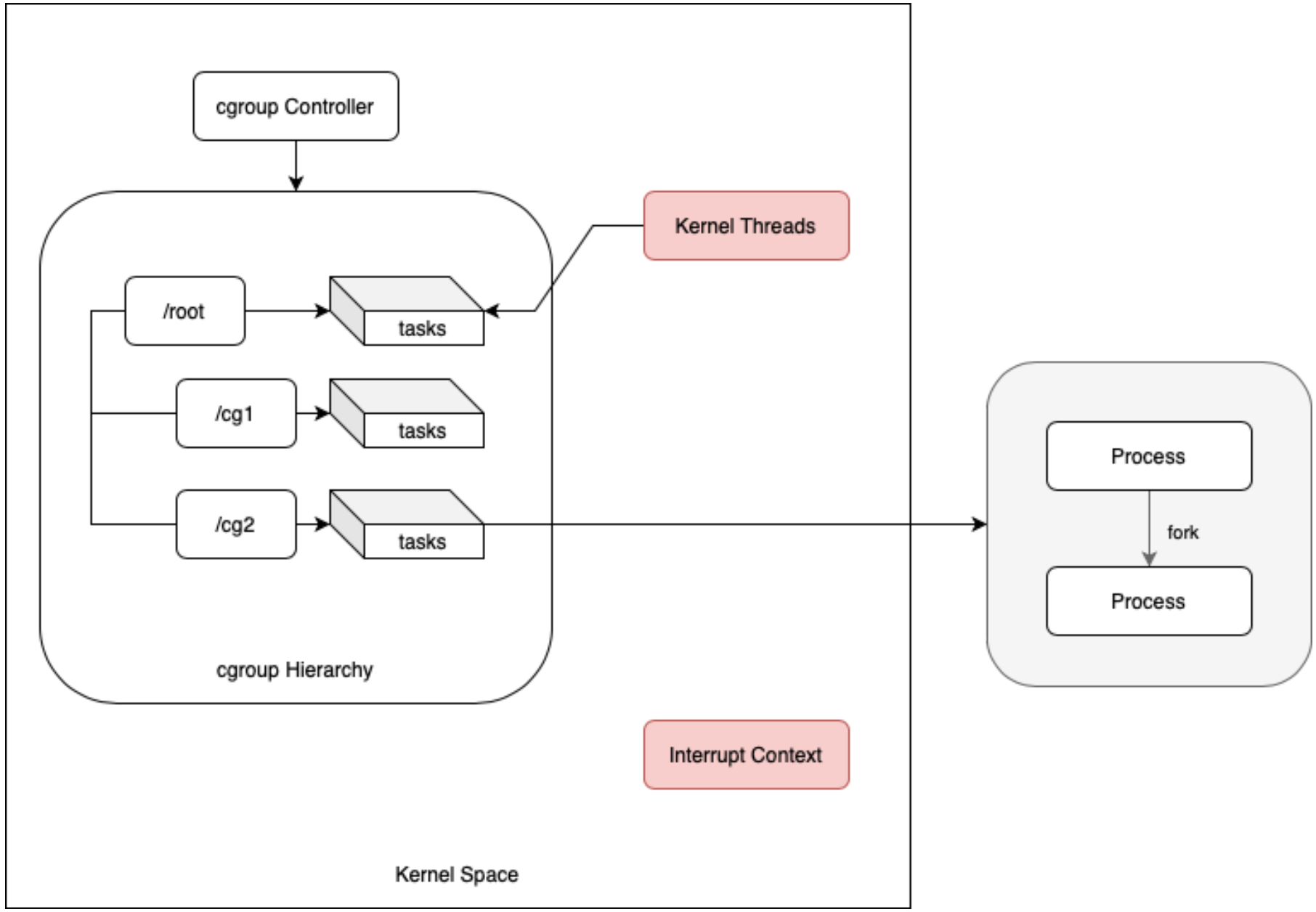
```
  File "/usr/lib/python3.6/threading.py", line 916
, in _bootstrap_inner
    self.run()
  File "try.py", line 14, in run
    run()
  File "try.py", line 19, in run
    tot += 1
UnboundLocalError: local variable 'tot' referenced
 before assignment

Exception in thread Thread-1:
Traceback (most recent call last):
  File "/usr/lib/python3.6/threading.py", line 916
, in _bootstrap_inner
    self.run()
  File "try.py", line 14, in run
    run()
  File "try.py", line 19, in run
    tot += 1
UnboundLocalError: local variable 'tot' referenced
 before assignment
```

Result

- Time consumption: ~157%
- CPU usage: ~43.6%

Part 5: Exploiting Kernel threads & Interrupt Context

Softirq Handling


cgroup Controller
Kernel Threads
/root
tasks
/cg1
tasks
/cg2
tasks
cgroup Hierarchy
Process
fork
Process
Interrupt Context
Kernel Space

Softirq

- Software Interrupt Request
- It is common for `hardware` interrupt handlers to raise softirqs
- While most hardware interrupts might **not** be directly raised by containers
- Container users are able to manipulate workloads on network interface generating NET softirq
- The handling of those softirqs consumes CPU resources on the process context of kernel thread or interrupt context

NET softirq

- Interrupt will be raised once the NIC finishes a packet transmission, and softirqs are responsible for moving packets be-tween the driver’s buffer and the networking stack.
- **1%** overhead for net-working traffic over **1 Gbps**

Attack Strategy

- The overhead incurred by the networking traffic will be greatly amplified by the firewall system (e.g., iptables) on the server.
- On Linux, Docker relies on configuring iptables rules to provide network isolation for containers.
- Particularly, it might set multiple rules for containers that provide web or networking services.
- The rules **exist** even the container is **stopped**.

Experiment

- Limit only one cpu core
- Measure CPU usage with the multithreaded script by `htop`

```
Get:6 http://archive.ubuntu.com/ubuntu bionic/main amd64 libip6tc0 amd64 1.6.1-2ubuntu2 [19.9 kB]
Get:7 http://archive.ubuntu.com/ubuntu bionic/main amd64 libiptc0 amd64 1.6.1-2ubuntu2 [9308 B]
Get:8 http://archive.ubuntu.com/ubuntu bionic/main amd64 libnetfilter-conntrack3 amd64 1.0.6-2 [37.8 k
B]
Get:9 http://archive.ubuntu.com/ubuntu bionic/main amd64 iptables amd64 1.6.1-2ubuntu2 [269 kB]
Fetched 416 kB in 1s (285 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package multiarch-support.
(Reading database ... 16954 files and directories currently installed.)
Preparing to unpack .../multiarch-support_2.27-3ubuntu1.3_amd64.deb ...
Unpacking multiarch-support (2.27-3ubuntu1.3) ...
Setting up multiarch-support (2.27-3ubuntu1.3) ...
Selecting previously unselected package libnfnetlink0:amd64.
(Reading database ... 16957 files and directories currently installed.)
Preparing to unpack .../0-libnfnetlink0_1.0.1-3_amd64.deb ...
Unpacking libnfnetlink0:amd64 (1.0.1-3) ...
Selecting previously unselected package libip4tc0:amd64.
Preparing to unpack .../1-libip4tc0_1.6.1-2ubuntu2_amd64.deb ...
Unpacking libip4tc0:amd64 (1.6.1-2ubuntu2) ...
Selecting previously unselected package libmnl0:amd64.
Preparing to unpack .../2-libmnl0_1.0.4-2_amd64.deb ...
Unpacking libmnl0:amd64 (1.0.4-2) ...
Selecting previously unselected package libxtables12:amd64.
Preparing to unpack .../3-libxtables12_1.6.1-2ubuntu2_amd64.deb ...
Unpacking libxtables12:amd64 (1.6.1-2ubuntu2) ...
Selecting previously unselected package libip6tc0:amd64.
Preparing to unpack .../4-libip6tc0_1.6.1-2ubuntu2_amd64.deb ...
Unpacking libip6tc0:amd64 (1.6.1-2ubuntu2) ...
Selecting previously unselected package libiptc0:amd64.
Preparing to unpack .../5-libiptc0_1.6.1-2ubuntu2_amd64.deb ...
Unpacking libiptc0:amd64 (1.6.1-2ubuntu2) ...
Selecting previously unselected package libnetfilter-conntrack3:amd64.
Preparing to unpack .../6-libnetfilter-conntrack3_1.0.6-2_amd64.deb ...
Unpacking libnetfilter-conntrack3:amd64 (1.0.6-2) ...
Selecting previously unselected package iptables.
Preparing to unpack .../7-iptables_1.6.1-2ubuntu2_amd64.deb ...
Unpacking iptables (1.6.1-2ubuntu2) ...
Setting up libip4tc0:amd64 (1.6.1-2ubuntu2) ...
Setting up libiptc0:amd64 (1.6.1-2ubuntu2) ...
Setting up libxtables12:amd64 (1.6.1-2ubuntu2) ...
Setting up libnfnetlink0:amd64 (1.0.1-3) ...
Setting up libmnl0:amd64 (1.0.4-2) ...
Setting up libip6tc0:amd64 (1.6.1-2ubuntu2) ...
Setting up libnetfilter-conntrack3:amd64 (1.0.6-2) ...
Setting up iptables (1.6.1-2ubuntu2) ...
Processing triggers for libc-bin (2.27-3ubuntu1.3) ...
root@9afb557c2672:/# mkdir /etc/iptables
root@9afb557c2672:/# vim /etc/iptables/rules.v4
root@9afb557c2672:/# iptables-restore < /etc/iptables/rules.v4
```

```
  1  [||||||||||||||||||||||||||||||||94.7%]   Tasks: 2643, 2630 thr; 4 running
  2  [||||||||||||||||||                44.3%]   Load average: 10.08 3.46 1.36
  3  [||||||||||||||||||                40.7%]   Uptime: 2 days, 21:47:56
  4  [||||||||||                        24.6%]
  Mem[||||||||||||||||||||||||||||1.80G/1.94G]
  Swp[||||||||||||||||||||||||||||1.19G/1.19G]

IO Priority:          PID USER       PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
Realtime 0 (High)   35302 root        20   0  177M   596   552 S  0.0  0.0  0:00.01 curl http://sus
Realtime 1          35355 root        20   0  177M   596   548 S  0.0  0.0  0:00.01 curl http://sus
Realtime 2           4757 root        20   0 27812  2868  2084 S  0.0  0.1  0:01.44 tmux new -s ipt
Realtime 3          34629 root        20   0  177M   744   688 S  0.0  0.0  0:00.01 curl http://sus
Realtime 4          34790 root        20   0  177M   712   656 S  0.0  0.0  0:00.01 curl http://sus
Realtime 5          34810 root        20   0  177M   760   712 S  0.0  0.0  0:00.01 curl http://sus
Realtime 6          34820 root        20   0  177M   752   704 S  0.0  0.0  0:00.01 curl http://sus
Realtime 7 (Low)    34083 root        20   0  177M   864   824 S  0.0  0.0  0:00.01 curl http://sus
Best-effort 0 (High) 34088 root       20   0  177M   864   820 S  0.0  0.0  0:00.01 curl http://sus
Best-effort 1       34101 root        20   0  177M   848   800 S  0.0  0.0  0:00.01 curl http://sus
Best-effort 2       34113 root        20   0  177M   860   816 S  0.0  0.0  0:00.01 curl http://sus
Best-effort 3       34339 root        20   0  177M   848   796 S  0.0  0.0  0:00.01 curl http://sus
Best-effort 4       34427 root        20   0  177M   764   720 S  0.0  0.0  0:00.01 curl http://sus
Best-effort 5       33908 root        20   0  177M  1760  1720 S  0.0  0.1  0:00.01 curl http://sus
Best-effort 6        4881 root        20   0 2111M  2720  2720 T  0.0  0.1  0:08.19 python3
Enter Set    Esc Cancel
```

```
root@9afb557c2672:/# python3
Python 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os, threading
>>>
>>> N = 10000
>>>
>>> def foo(x: int):
...  w = os.system('curl http://sustech.edu.cn')
...  return w
...
>>> if __name__ == '__main__':
...  threads = []
...  for i in range(N):
...   threads.append(threading.Thread(target=foo, args=(i,)))
...  for thread in threads:
...   thread.start()
...  for thread in threads:
...   thread.join()
...
```

Result

- CPU usage: ~215.7%

Docker Escape: Future

- Defence? Think about it.

Q&A

11712121 胡玉斌/11811902 顾同舟/11812102 黄炜杰/11812103 张文灏/11812104 石文轩