

# Container Security

Bachelor's degree graduation project

Chih-Hsuan Yang

National Sun Yat-sen University

January 20, 2021  
v1.0

# Outline

- 1 Outcome
  - Expected Outcome
  - Current Outcome
- 2 FIXME
  - Linux feature
  - Exploit
- 3 Paper review
  - Papers
  - Simple container
- 4 Current progress
- 5 Reference

# Outcome

# Medical cloud

- Container
- Privacy, Security
- Load balanceability, Portability, Manageability

# Current outcome

- An easy container with Linux namespace.

```
→ container git:(main) X gcc *.c -o c
→ container git:(main) X sudo ./c "bash"
Success on creating container
Start container: bash with clone id: 193761
In container PID: 1
bash-5.0# ./test.sh
This is the self test script in container!
Support bash cat echo ls rm hostname, 5 commands.
./test.sh
-----FILE: test.sh -----
1  #!/bin/bash
2
3  echo "This is the self test script in container!"
4  echo "Support bash cat echo ls rm hostname, 5 commands."
5
6  echo $0
7
8  echo "-----FILE: test.sh -----"
9  cat -n test.sh
10 echo "-----"
11
12 echo $(hostname) >天竺鼠車車
13 cat 天竺鼠車車
14 rm 天竺鼠車車
15 ls
-----
container
bin dev etc home lib lib64 mnt opt proc root run sbin sys test.sh tmp usr var
bash-5.0# exit
```

# List of attack surface

- cgroups with race condition
- namespace
  - wrong privileges
  - Cannot cross namespace? Really?
- init.
  - stack overflow(thread)?
  - fork and CoW?
  - defunct processing
- lib/syscall/kernel exploit

FIXME

# namespace

- From Linux kernel 3.8
- System calls
  - clone, unshare, setns
- Nested, scope

```
→ container git:(main) X sudo unshare --fork --pid --mount-proc bash
[sudo] password for scc:
root@scc-lab:/media/d/git/nsysu/cs/try/lc/container# ps -a
  PID TTY          TIME CMD
    1 pts/1        00:00:00 bash
    8 pts/1        00:00:00 ps
root@scc-lab:/media/d/git/nsysu/cs/try/lc/container# exit
exit
→ container git:(main) X
```



# namespace

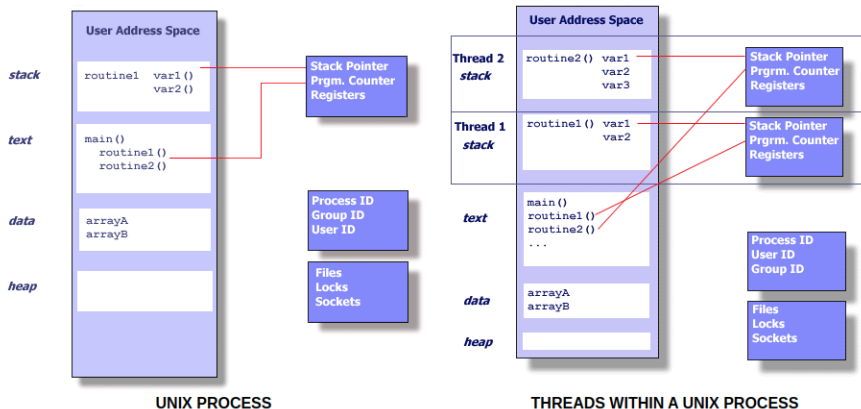
- 6 mechanisms
  - Mount, UTS, IPC, PID, NET, USER
- ps: `mount -t proc proc /proc`
- *PID* = 1, the "init" [1]
  - SIGTERM, SIGKILL
  - The defunct

# cgroups

- Access controller
  - Resource limiting: CPU, Mem, IO...
  - Prioritization: CPU, IO...
  - Accounting: evaluate
  - Control: freeze, check, and resume
- The OOM killer 4.19
  - Guarantee the integrity of the workload.

# Stack overflow

- Default: 8MB
- The init of container, confused here.



[2]

# Paper review

# Have been read papers

- Linux Kernel OS Local Root Exploit[3]
- PINE: Optimizing Performance Isolation in Container Environments[4]
- Study of Security Flaws in the Linux Kernel by Fuzzing[5]

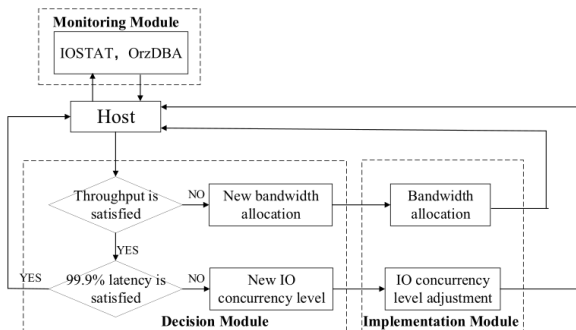
# Linux Kernel OS Local Root Exploit

- Dirty CoW
- race condition with mmap
- Counteract
  - Comparing the size of the binary against the size of the original binary[3]
  - systemtap module
  - update && upgrade

# Optimizing Performance Isolation

## Microservices

- latency-sensitive services
- throughput-first services



**FIGURE 7.** The overall flow of PINE.

[4]

# Fuzzing

## Syzkaller

- Stack overflow
  - Canary
  - KSLR
  - Shadow stack
- Integer overflow
  - Options to detected and SIGKILL
- Heap overflow
  - Check size of the variable in comparison with the size `copy_to_user`, `copy_from_user`
  - Guard pages
  - Check functions and glibc's heap protections



# Fuzzing

## Syzkaller

- Format string injection
  - Detect non-constant format string
- Kernel pointer leak
  - Remove visibility for kernel symbols
  - Block the use of %p
- Uninitialized variables
  - RAIL
- Use-after-free
  - RAIL too

# My container

Code review

# My container

```
4 int cont_start(char *argv[], int do_wait);  
5 int cont_stop();
```

# My container

```
44 int cont_start(char *argv[], int do_wait)
45 {
46     c_stkptr = (char *) malloc(STK_SIZE);
47     c_pid = (long) loader(argv);
48     if (c_pid)
49         printf("%s on creating container\n", strerror(errno)
50 );
51     printf("Start container: %s with clone id: %d\n", argv
52 [0], c_pid);
53     if (do_wait)
54         waitpid(c_pid, NULL, 0);
55 }
```

```
38 static inline pid_t loader(char *argv[])
39 {
40     return clone(run, c_stkptr + STK_SIZE,
41                 CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWPID |
42                 SIGCHLD, argv);
43 }
```





# My container

```
25 static int run(void *argv)
26 {
27     char **arg = (char **) argv;
28     isol();
29     chdir("/");
30     int ret = execvp(arg[0], arg);
31     if (ret)
32         printf("%s in container\n", strerror(errno));
33     return ret;
34 }
```

```
15 static void isol()
16 {
17     unshare(CLONE_FILES | CLONE_FS | CLONE_SYSVSEM |
18     CLONE_NEWCGROUP);
19     sethostname("container", 10);
20     if (chroot("./rootfs"))
21         perror("chroot error");
22     printf("In container PID: %ld\n", (long) getpid());
23 }
```

## Current progress

# Application of MOST

- Papers:  1.0
- FIXME:  0.308
- Pages:  0.380
- My Exp.:  Maybe...

# Demo

Live demo.







# List of attack surface

- cgroups with race condition
- namespace
  - wrong privileges
  - Cannot cross namespace? Really?
- init.
  - stack overflow(thread)?
  - fork and CoW?
  - defunct processing
- lib/syscall/kernel exploit

## Reference

# References I

-  HONGLI LAI. *Docker and the PID 1 zombie reaping problem*. blog. 2015. URL: <https://blog.phusion.nl/2015/01/20/docker-and-the-pid-1-zombie-reaping-problem/>.
-  Lawrence Livermore National Laboratory Blaise Barney. *POSIX Threads Programming*. site. URL: <https://computing.llnl.gov/tutorials/pthreads/>.
-  Babak D. Beheshti A.P. Saleel Mohamed Nazeer. “Linux kernel OS local root exploit”. In: 2017. URL: <https://ieeexplore.ieee.org/document/8001953>.
-  Congfeng Jiang Youhuizi Li Jiancheng Zhang. “PINE: Optimizing Performance Isolation in Container Environments”. In: *IEEE Access* 7.18526707 (2019).

# References II



E.V. Sharlaev P.A. Teplyuk A.G. Yakunin. “Study of Security Flaws in the Linux Kernel by Fuzzing”. In: 2020. URL: <https://ieeexplore.ieee.org/document/9271516>.