

The Container Security in Healthcare Data Exchange System

Bachelor's degree graduation project

Chih-Hsuan Yang

National Sun Yat-sen University

February 3, 2021
v1.1

Outline

- 1 Outcome
 - Expected Outcome
 - Current Outcome
- 2 Interfaces
 - Linux feature
 - Exploit
- 3 Related work
 - Study
 - 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

Interfaces

namespace

- Start from 2.4.19(2003)
- Completed in Linux kernel 3.8(2013)
- 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

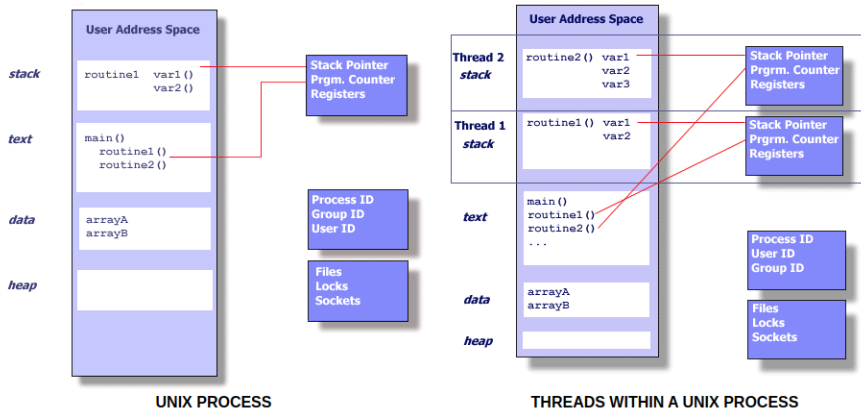
- 6 mechanisms
 - Mount, UTS, IPC, PID, NET, USER
- ps: `mount -t proc proc /proc`
- *PID* = 1, the "init" [1]
 - SIGTERM, SIGKILL
 - The defunct
- Starting in Linux 3.8, unprivileged processes can create user namespaces, ... unprivileged applications now have access to functionality that was formerly limited to root. ... Thus, it may happen that user namespaces have some as-yet unknown security issues. [2]

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.



[3]

Related work

The 3 Big issue

- Concepts
- Container security
- High-performance server

Concepts

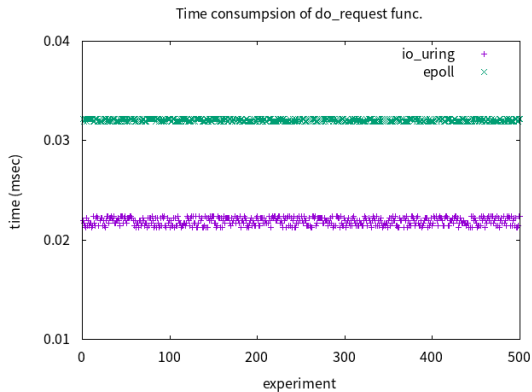
- Virtual machines and containers
- The FHIR system
- Linux kernel features
 - namespaces, cgroups, capabilities, seccomp
 - mmap, copy on write, race condition
 - aio, epoll, io_uring

Container security

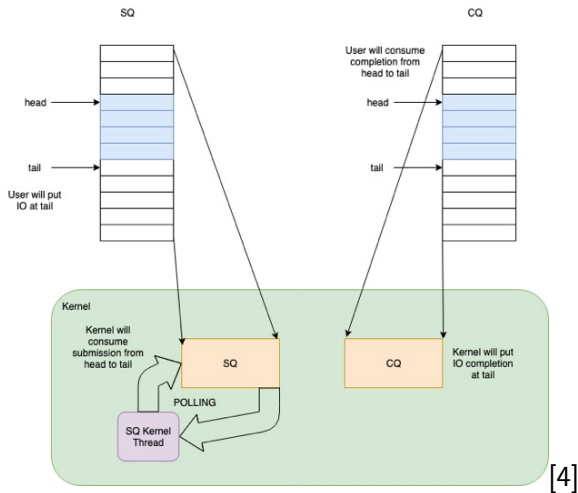
- Study of the Dirty Copy On Write
- Container Security: Issues, Challenges, and the Road Ahead
- Linux kernel exploit

High-performance server

- PINE: Optimizing Performance Isolation in Container Environments
- The epoll vs. io_uring performance comparison



The io_uring



My container

Code review

My container

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

My container

```
51 int cont_start(char *argv[], int do_wait)
52 {
53     c_stkptr = (char *) malloc(STK_SIZE);
54     c_pid = (long) loader(argv);
55     if (c_pid)
56         printf("%s on creating container\n", strerror(errno)
57 );
58     printf("Start container: %s with clone id: %d\n", argv
59 [0], c_pid);
60     if (do_wait)
61         waitpid(c_pid, NULL, 0);
62 }
```

```
45 static inline pid_t loader(char *argv[])
46 {
47     return clone(run, c_stkptr + STK_SIZE,
48                 CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWPID |
49                 SIGCHLD, argv);
50 }
```





My container

```
18 static void isol()  
19 {  
20     unshare(CLONE_FILES | CLONE_FS | CLONE_SYSVSEM |  
CLONE_NEWCGROUP);  
21     sethostname("container", 10);  
22     if (chroot(Stringize_Value_of(ROOTFS)))  
23         perror("chroot error");  
24     printf("In container PID: %ld\n", (long) getpid());  
25 }
```

```
32 static int run(void *argv)  
33 {  
34     char **arg = (char **) argv;  
35     isol();  
36     chdir("/");  
37     int ret = execvp(arg[0], arg);  
38     if (ret)  
39         printf("%s in container\n", strerror(errno));  
40     return ret;  
41 }
```

Current progress

Application of MOST

- Papers:  1.0
- FIXME:  0.9
- Pages:  1.1
- 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/>.



Michael Kerrisk. *Namespaces in operation, part 1: namespaces overview*. site. URL: <https://lwn.net/Articles/531114/>.



Lawrence Livermore National Laboratory Blaise Barney. *POSIX Threads Programming*. site. URL: <https://computing.llnl.gov/tutorials/pthreads/>.



seHTTPd + io_uring. URL: https://hackmd.io/g9YkMb4nRvW_SWeLNkQrow?view.