Container Security

Chih-Hsuan Yang
National Sun-Yet-San University, Taiwan
Bachelor's degree graduation project
Advisor: Chun-I Fan

1. Abstract

A research of container's modern cyber security issue. Many companies use container to run their services.
// FIXME

2. Motivation

The Container is a virtualization technique to package applications and dependencies to run in an isolated environment. Containers are faster to start-up, lighter in memory/storage usage at run time and easier to build up than virtual machines. Because the container shares the kernel with the host OS and other containers.
I often used to run a docker container to host my services. For example: my homework, servers and some services in Information security club at NSYSU. But there are some vulnerabilities about container technique. Like "Dirty CoW[1]" and "Escape vulnerabilities".
"Dirty CoW is a vulnerability in the Linux kernel. It is a local privilege escalation bug that exploits a race condition in the implementation of the copy-on-write mechanism in the kernel's memory-management subsystem"[2]. It founded by Phil Oester. I was 16, the first year I had touched the docker container. I tried to use the Dirty CoW vulnerability to take the root privilege of my Android phone.
Escape vulnerability is a subcategory of sandbox security. At first, security researchers often need sandbox to help they analyze malware, which prevent the malware influence researcher's host OS. Nowadays, the sandbox not only be used in analyzing, but also used to execute a normal application for an isolated environment. But if the application could modify the outside resources without the kernel permission. That loses the purpose of isolation. That might cause the information leaked or the kernel be hacked.
Hence, there is a big problem about: "How to make sure my services isolated and secure?" I am the leader of Information security club. I should maintain all the services working perfectly. Moreover we are information security club. Therefore, the security and performance issue is the top-priority requirement.

3. Papers Review

3.1. Study of the Dirty Copy On Write[3]

In this paper show the race condition, and the mechanism of "copy on write". "Copy on write" is "A resource-management technique used in computer programming to efficiently implement a "duplicate" or "copy" operation on modifiable resources." [4] We often use the CoW while fork() or mmap().

3.1.1. mmap.

// FIXME

3.1.2. race condition. // FIXME

3.1.3. Dirty CoW demo code[1].

Let's analyze the Proof of concept (POC) of dirty CoW.
There are 2 function have race condition, madviseThread and procselfmemThread.

```
pthread_create(&pth1,NULL,
  madviseThread,argv[1]);
pthread_create(&pth2,NULL,
  procselfmemThread,argv[2]);
```

dirtyc0w.c

Create 2 threads and give them argvs.

madviseThread

```
void *madviseThread(void *arg)
{
  char *str;
  str=(char*)arg;
  int i,c=0;
  for(i=0;i<100000000;i++)
  {
    c+=madvise(map,100,MADV_DONTNEED
    );
  }
  printf("madvise %d\n\n",c);
}
```

dirtyc0w.c

The madvise() is a system call, which gives advice about use of memory. The MADV_DONTNEED is a flag to tell kernel "Do not expect access it in the near future." And the manual said: "Note that, when applied to shared mappings, MADV_DONTNEED might not lead to immediate freeing of the pages in the range. The kernel is free to delay freeing the pages until an appropriate moment.[5]"

That is the problem! We ask kernel we would not access it in the near future. The kernel would delay a moments to free it. But, at the meanwhile, our another thread asks for write it.

procselfmemThread

```
void *procselfmemThread(void *arg)
{
  char *str;
  str=(char*)arg;
  int f=open("/proc/self/mem",O_RDWR
  );
  int i,c=0;
  for(i=0;i<100000000;i++) {
    lseek(f,(uintptr_t) map,SEEK_SET
    );
    c+=write(f,str,strlen(str));
  }
  printf("procselfmem %d\n\n", c);
}
```

dirtyc0w.c

If the scheduler context switch from madviseThread to the procselfmemThread, while the madvise() just was having been called. The permission of the page of the mmaped file would not change.

Therefore we could get the permission for I/O.

3.2. Linux Kernel OS Local Root Exploit[6]

4. Methods

5. Expected Outcome

Generate the PoC code and the solution of a container cyber attack.

6. References

References

[1] Phil Oester. *Dirty CoW CVE-2016-5195*. URL: https://dirtycow.ninja/.

[2] Wikipedia. *Dirty CoW*. URL: https://en.wikipedia.org/wiki/Dirty_COW.

[3] Tanjila Farah Delwar Alam Moniruz Zaman. "Study of the Dirty Copy On Write, A Linux Kernel Memory Allocation Vulnerability". In: 2017. URL: https://ieeexplore.ieee.org/abstract/document/7530217.

[4] Wikipedia. *Copy-on-write*. URL: https://en.wikipedia.org/wiki/Copy-on-write.

[5] None. *Manpage of madvise*. URL: https://www.man7.org/linux/man-pages/man2/madvise.2.html.

[6] Babak D. Beheshti A.P. Saleel Mohamed Nazeer. "Linux kernel OS local root exploit". In: 2017. URL: https://ieeexplore.ieee.org/document/8001953.