Container Security

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

1. Abstract

Recently, many companies use containers to run their microservices, since containers could make their hardware resources be used efficiently. For example, GCP(Google Cloud Platform), AWS(Amazon Web Services), and Microsoft Azure are using this technique to separate subscribers' resources and services. However, if the hacker attacks the kernel or gets privilege escalation of containers, then such attacks would influence the host or the other containers. Therefore, this research would analyze, implement, and protect the container escalation. The container escalation could inspire in a container and influence the host or the other containers. This project would use the medical system with FHIR(Fast Healthcare Interoperability Resources)[1] to simulate the real world threat and purpose a secure solution to protect patient's privacy.

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 deploy than virtual machines. Because the container shares the kernel with the host OS and other containers, and deploys by a configure file.
First, we often used to run a docker container to host our services. For example: assignments, servers and some services in Information security club at NSYSU(National Sun Yat-sen University). But there are some threats about container technique. Like "Dirty CoW[2]" 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"[3]. It founded by Phil Oester. We were 16, the first year we had touched the docker container. We 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. However 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?" The author of this paper is the leader of Information security club. He should maintain all the services working perfectly. Moreover we are information security club. Therefore, the security and performance issue is the top-

priority requirement.
Second, in order to present the container security, we would take the medical system for example. The medical system is the most famous part internationally. Including face the COVID-19 in Taiwan, we not have the local COVID-19 case in more than 250 days.[4] However, the medical profession needs to renew the exchanging EHR(electronic health records) system these years. In order to protect the privacy of patients, and producing the high performance system to exchange the EHR, we need a easy deployment, effective runtime, and secure system. We have to do this research in this project.

3. Related works

In this section will introduce why we choose these papers, some what did the paper say, and some related mechanisms about that issue. In this project will focus on (I) security, and (II) high performance.

3.1. Security

3.1.1. Study of the Dirty Copy On Write. In this paper[5] 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." [6] It often be inspire when fork() or mmap().

mmap This is a system call of mapping files or devices in to memory, which creates a new mapping in the virtual address space of the caller process. Such that the process could operate the instance of file in memory directly. And some libraries are also mapped into the virtual address space to share and handle the function call. Therefore, the processes could take the same view of libraries in it's memory space.

Copy on write This mechanism purposes of a resource being duplicated but not modified, it is not necessary to create a new entry. Therefore the kernel can make callers share the same memory resources. The mmap is a system call could inspire this mechanism in above paragraph. When some processes request same memory resource, the kernel supplies the same memory page to callers.

Race condition That is processes or threads are racing the same mutable resource. For example: There is a accessible and mutable shared memory which be initialized as 0. And there are 2 threads or processes sharing that page. Consider one of the tasks is assigning the page full of character 'A'. In the meanwhile, the schedular context switches to the other task, which assigns that page full of 'B'. Then the schedular context switches again to the first task. There is a problem now. What is the page for the first task looked like? Obviously, it does not meet the expectation for the first task. This is race condition.

3.1.2. Dirty CoW demo code. Let's analyze the proof of concept(PoC) of dirty CoW.(Oester, 2016)[2] The key of inspiring this vulnerability is the mmaped memory space, which is mapped with the PROT_READ flag. The PROT_READ flag declares the page is read only.

```
f=open(argv[1],O_RDONLY);
fstat(f,&st);
name=argv[1];
map=mmap(NULL,st.st_size,PROT_READ
  ,MAP_PRIVATE,f,0);
```

src/dirtyc0w.c

It creates 2 threads, which would have a race condition of the mmaped memory space, madviseThread and procselfmemThread.
threads in main

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

```c
  pthread_create(&pth2,NULL,
   procselfmemThread,argv[2]);
```

src/dirtyc0w.c

In one thread, call a system call "madvise", would make the user thread gain the root privilege to operate the protected page temporary. And the flag MADV_DONTNEED would tell the kernel: "Do not Expected access it in the near future.[7]" Moreover, this flag might not lead to immediate freeing of pages in the range. The kernel is free to delay free the pages until an appropriate moment.[7]

madviseThread

```c
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);
}
```

src/dirtyc0w.c

In another thread, open its memory resource file. This file is a special file, which allow the process reads its memory by itself. Than, we move the printer of file descriptor of the memory resource file to the mmaped space. And try to write it. But the mmaped space is a read only space. We expected the kernel would create a copy of the this space and write the copy[8].

procselfmemThread

```c
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);
}
```

src/dirtyc0w.c

But there is a problem! There is an another thread is racing this page with root privilege. If the schedular context switches the madviseThread to procselfmemThread, while the adviseThread is calling the "madvise" system call. It would cause the procselfmemThread gain the root privilege from madviseThread to control the mmaped file.

3.1.3. Container Security: Issues, Challenges, and the Road Ahead. This paper[9] has derived 4 generalized container security issues: (I) protecting a container from applications inside it, (II) inter-container protection, (III) protecting the host from containers, and (IV) protecting containers from a malicious or semi-honest host.[9]

The Dirty CoW vulnerability is a exploit from kernel. But the benefit of container and host OS are share the same kernel. This vulnerability can be used in container to attack the kernel, and gives this application root privilege, changes this containers as a privileged container or supervises the other containers. Therefore, we should protect the host form the container(which belongs to type (III) threat in this paper).

Virtual machine and container // FIXME: Draw the architecture of VM and container.

Linux kernel features // FIXME: Introduce these features for isolating processes in Linux.

namespaces

// FIXME: Namespaces perform the job of isolation and virtualization of system resources for a collection of processes.[9]

cgroups
// FIXME: Limits, accounts for, and isolates the resource usage of a collection of processes. [10]

capabilities
// FIXME: Divide the privileges traditionally associated with superuser into distinct units.

seccomp
// FIXME: Only some specified process could call some specified system calls.

3.1.4. Kernel fuzzing.

3.2. High performance

This section will study some IO performance and caching issues. Because the medical data exchange system demands stringent specification of the response time. The IO is the most often causing the bottleneck in the low latency required system. In order to support a high performance system, we can design a module to control the throughput intelligently, and use the cache friendly architecture to minimal the latency.

3.2.1. PINE: Optimizing Performance Isolation in Container Environments. This paper[11] introduce a high throughput and low latency module to control the IO streams.

4. Methods

This project would use the MapReduce model. As shown in Figure 1.

4.1. Study

4.1.1. Study CVEs and related mechanisms. The Linux kernel is a monolithic kernel, which is over 28 million lines of codes now(2020). There are many mechanisms to solve the real life situations. Study those CVEs' related mechanism in the kernel, might have more chance to find new vulnerabilities.

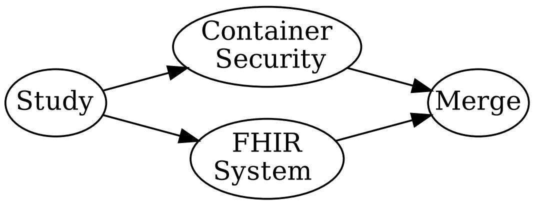

Figure 1. The mapReduce model in this project

This project will study several container vulnerabilities for example: CVE-2016-8655 [12], CVE-2016-9962[13], and CVE-2020-14386[14].

And study some kernel exploit techniques[15], because the container shares the kernel. If I could exploit the kernel in the suffering container, it might have more chance to influence the other containers or host.

4.1.2. Study FHIR and related standards. This project will implement the FHIR[1] data exchange system to demonstrate the container security risk. Hence, we should study FHIR standard, JSON format(RFC7159), XML format(RFC4825), and RESTful APIs.

4.1.3. Efficient IO with io_uring. [16] This is a new asynchronous I/O API in Linux kernel 5.7. POSIX has aio_read() and aio_write() to satisfy asynchronous IO, however the implementation of those is most often lackluster and performance is poor. We will study and implement the new asynchronous I/O API: io_uring in this project to optimize performance of the FHIR data exchange system.

4.2. Container Security

4.2.1. Implement a simple container. The Linux kernel supply some system calls to clone a process(also in thread) in their own namespace and group. We could implement a simple container by ourself, so that we can make a list of vulnerabilities may happen.

```
âđIJ  container git:(main) âIJŮ sudo ./c bash
Success on creating container
Start container: bash with clone id: 375696
In container PID: 1
bash-5.0# ./test.sh
This is the self test script in container!
Support bash cat echo ls rm hostname tree, 7 commands.
/bin/bash
./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 tree, 7 commands."
     5
     6  echo "$SHELL"
     7  echo $0
     8
     9  echo "-------FILE: $0 --------"
    10  cat -n $0
    11  echo "----------------------"
    12
    13  echo $(hostname) > åd'l'çńžéi jăèżŁèżŁ
    14  cat åd'l'çńžéi jăèżŁèżŁ
    15  rm åd'l'çńžéi jăèżŁèżŁ
    16  ls
----------------------
container
bin  dev  etc  home  lib  lib64  mnt  opt  proc  root  run  sbin  sys  test.sh  tmp  usr  var
bash-5.0# exit
exit
âđIJ  container git:(main) âIJŮ
```

src/lc_out.txt

4.2.2. List secure details of the simple container. In my rough opinion, there are 5 types of container security risks, (I) Host OS risks, (II) Orchestration system risks, (III) Container runtime risks, (IV) Registry risks, (V) Images risks. In this stage we should research the details of those risk, and purpose some solutions.

Host OS risks

- Improper user permission
- Kernel vulnerabilities

Orchestration system risks

- Unbounded domain access
- Weak credentials
- Mismanaged inter-container network traffic
- Mixed of workload sensitivity levels

Container runtime risks

- Runtime software vulnerabilities
- Unbounded network access from containers
- Insecure container runtime configurations

Registry risks

- Insecure connections to registries
- Old images in registries

Images risks

- Image vulnerabilities
- Embedded malware or secrets

4.2.3. Aim a vulnerability and implement the PoC.
After listing the risks. This project would find a vulnerability of privilege escalation in the container and affect with other containers.

4.2.4. Implement the patch and pull request. Being a security researcher, we cannot just only exploit the software, but also give patches to the maintainer. Make the container technique more secure.

4.3. FHIR system

4.3.1. Front-end. This project would designed a user friendly interface. Make it easy to get data for the patient and patient's family. And make the exchange of patient's data between different medical center confidential, integral, and available.

The interface would be designed as a website. Which make every user could access in different platform.

4.3.2. Back-end.

4.3.3. Database. ,

4.3.4. Access controller.

4.3.5. High performance server.

4.4. Merge

5. Expected Outcome

Would research some related vulnerabilities, and implement the PoC code. Moreover this project will generate the patch of the vulnerability(s) to protect these attack(s).

6. References

References

[1] HL7. *FHIR homepage*. URL: https://www.hl7.org/fhir/.

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

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

[4] James Griffiths Chandler Thornton. *Taiwan reports first local Covid-19 case in more than 250 days*. URL: https://edition.cnn.com/2020/12/22/asia/taiwan-coronavirus-intl-hnk/index.html.

[5] 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.

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

[7] GNU. *Manpage of madvise*. URL: https://www.man7.org/linux/man-pages/man2/madvise.2.html.

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

[9] Tassos Dimitriou Sari Sultan Imtiaz Ahmad. "Container Security: Issues, Challenges, and the Road Ahead". In: *IEEE Access* 7.18620110 (2019).

[10] Wikipedia. *cgroups*. URL: https://en.wikipedia.org/wiki/Cgroups.

[11] Congfeng Jiang Youhuizi Li Jiancheng Zhang. "PINE: Optimizing Performance Isolation in Container Environments". In: *IEEE Access* 7.18526707 (2019).

[12] Inc. Red Hat. *CVE-2016-8655*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-8655.

[13] MITRE Corporation. *CVE-2016-9962*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-9962.

[14] Inc. Red Hat. *CVE-2020-14386*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-14386.

[15] xairy. *Linux Kernel Exploitation*. URL: https://github.com/xairy/linux-kernel-exploitation.

[16] Jonathan Corbet. *Ringing in a new asynchronous I/O API*. URL: https://lwn.net/Articles/776703/.

7. Academic Advisor

- Organize to a complete structure.
- Extend to a formal paper, and publish.