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 some kernel vulnerabilities, which could inspire in the container and influence the host or the other containers. This project would aim and patch a kernel vulnerability. Hope this research would make the technique of container more secure.

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?" The author 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.

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. 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 calling process. Some libiraries are also mapped into the virtual address space to handle the function call.

Why use mmap

How to use mmap

3.1.2. Copy on write. // FIXME: Many callers request same modifiable resources, and the kernel could use this technique to enhance the performance of these callers.

3.1.3. Race condition. // FIXME: Processes or threads are racing the same modifiable resources.

3.1.4. Dirty CoW demo code[1]. Let's analyze the proof of concept(PoC) of dirty CoW.(Oester, 2016)
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.

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

dirtyc0w.c

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

threads in main

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

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.[5]” 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.[5]

madviseThread

```
33 void *madviseThread(void *arg)
34 {
35   char *str;
36   str=(char*)arg;
37   int i,c=0;
38   for(i=0;i<100000000;i++)
39   {
40     c+=madvise(map,100,MADV_DONTNEED
       );
41   }
42   printf("madvise %d\n\n",c);
43 }
```

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[6]. procselfmemThread

```
50 void *procselfmemThread(void *arg)
51 {
52   char *str;
53   str=(char*)arg;
54   int f=open("/proc/self/mem",O_RDWR
       );
55   int i,c=0;
56   for(i=0;i<100000000;i++) {
57     lseek(f,(uintptr_t) map,SEEK_SET
       );
58     c+=write(f,str,strlen(str));
59   }
60   printf("procselfmem %d\n\n", c);
61 }
```

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.

3.2. Container Security: Issues, Challenges, and the Road Ahead[7]

This paper 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.[7]

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).

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

3.2.2. 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.[7]

cgroups
// FIXME: Limits, accounts for, and isolates the resource usage of a collection of processes. [8]

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

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

4. Methods

4.1. Study CVEs about the Linux kernel

This project will study several kernel vulnerabilities were discover in packet socket: CVE-2016-8655 [9], CVE-2017-7308[10], and CVE-2020-14386[11].

And study some kernel exploit techniques[12], 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.

4.2. Study 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.

4.3. Implement a simple container

4.4. List secure issues of the simple container

4.5. Aim a vulnerability

After studying the CVEs and related mechanisms. This project would find a vulnerability of privilege escalation in the container by attacking the kernel.

4.6. Implement the PoC

Implement the proof of concept of the aimed vulnerability.

4.7. Implement the patch and pull request

Being a security researcher, we cannot just only exploit the software, but also give patches to the maintainer. The Linux kernel is an open source project under GPL-2.0 license in GitHub. I would pull request to the repository. If my patch could be merged into the kernel to solve the container vulnerability.

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] 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] GNU. *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.

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

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

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

[10] MITRE Corporation. *CVE-2017-7308*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7308.

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

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

7. Academic Advisor

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