Stash your Cache - Cross-Container Linux Page Cache Covert Channel

Naor Radami
Ben Gurion University

Novak Boškov
Boston University

Trishita Tiwari
Cornell University

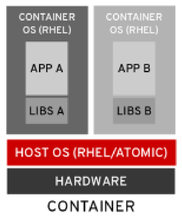
Ari Trachtenberg
Boston University

Abstract

Recently software containers have become a popular unit of light-weight virtualization. As opposed to virtual machines, containers offer faster boot times and lower resource usage by utilizing the services and resources of a host operating system. Consequently, container isolation is handled through enhanced operating system process isolation techniques. In this work, we demonstrate how the isolation of containers can be undermined in the various cloud settings including the top commercial environments. Our attacks make use of union file systems, one of the key distinctive features of software containers. The attacks include leakage of sensitive data out of the running process context, and creating a covert channel which may be used to transfer data and commands, which will enable us to perform data mulling out of the organization. Our attacks affect the most recent Linux kernels, even those with the patches for the side-channel mitigation against page cache attacks (CVE-2019-5489). This highlights the need for rethinking the page cache design in the context of multi-tenant container clouds.

Container based virtualization

- Package software with all its dependencies
- Maximize performance
  - Save on boot time, memory usage, amount of storage, etc.
  - Make orchestration easier (service scaling, migration, etc.)



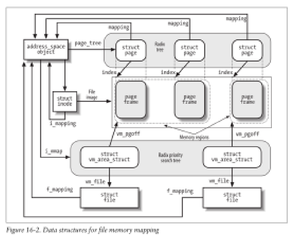
https://www.redhat.com/en/blog/virtual-machines-or-containers-maybe-both

Containers Reuse The Page Cache For Shared Files

- Page Cache is indexed using inode objects
- Both containers point to the same inode, despite using different mount targets

C1$ ls -i /.../overlay2/abc/merged/mysqld
2497575

C2$ ls -i /.../overlay2/1b3/merged/mysqld
2497575



"Understanding the Linux Kernel", Bovet & Cesati

Mincore

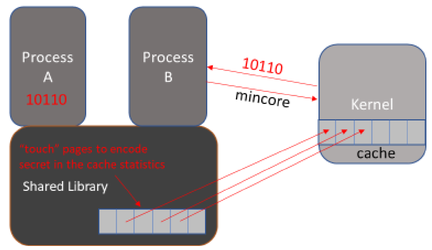
```
#include <sys/mman.h>

int mincore(void *addr, size_t length, unsigned char *vec);
```

mincore() returns a vector that indicates whether pages of the calling process's virtual memory are resident in core (RAM), and so will not cause a disk access (page fault) if referenced. The kernel returns residency information about the pages starting at the address *addr*, and continuing for *length* bytes.

https://man7.org/linux/man-pages/man2/mincore.2.html

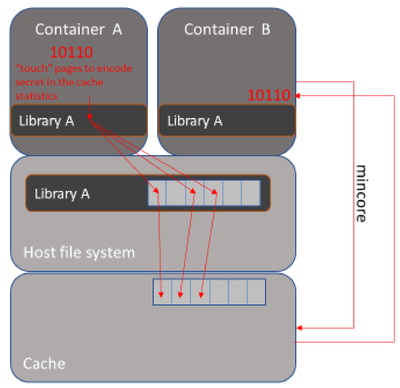
Sensitive data leakage from a process context

- Infecting a common library with our malicious code.
- Identifying sensitive data from within the process context.
- "touching" pages within a common file or library to encode the sensitive data.
- Identifying the message from a different runtime context using cache information (mincore, or timing).



Covert channel between containers

- Infecting a common library or executable.
- Container A – "touches" pages of a common file or library to encode the message.
- Container B – Identifying the message using cache information (mincore, or timing).



Demonstration on a major cloud provider

Container A

```
radaminaor@Naors-MBP Downloads % kubectl exec --stdin --tty nginx-admin-deployment-68f9cfb9b9-knr62 -- /bin/bash
root@nginx-admin-deployment-68f9cfb9b9-knr62:/#
root@nginx-admin-deployment-68f9cfb9b9-knr62:/#
root@nginx-admin-deployment-68f9cfb9b9-knr62:/# ./spy_on /lib/x86_64-linux-gnu/libhistory.so.7.0
00110000011
root@nginx-admin-deployment-68f9cfb9b9-knr62:/# ./page_cache_timing.s /lib/x86_64-linux-gnu/libhistory.so.7.0 2
argv[1] = /lib/x86_64-linux-gnu/libhistory.so.7.0
file includes 10 pages
47,
root@nginx-admin-deployment-68f9cfb9b9-knr62:/# ./spy_on /lib/x86_64-linux-gnu/libhistory.so.7.0
00110001111
```

Container B

```
radaminaor@Naors-MBP Downloads % kubectl exec --stdin --tty nginx-deployment-5f8bd88c54-9vfp7 -- /bin/bash
root@nginx-deployment-5f8bd88c54-9vfp7:/#
root@nginx-deployment-5f8bd88c54-9vfp7:/#
root@nginx-deployment-5f8bd88c54-9vfp7:/# ./spy_on /lib/x86_64-linux-gnu/libhistory.so.7.0
00110000011
root@nginx-deployment-5f8bd88c54-9vfp7:/# ./spy_on /lib/x86_64-linux-gnu/libhistory.so.7.0
00110001111
```

CVE-2019-5489 patch

- Exploits demonstrated in the "Page Cache Attacks" paper by Gruss et al. lead to a patch in the Linux kernel on the mincore syscall. This patch introduces an additional permission-checking routine, can_do_mincore, that is invoked before mincore. If the user passes the permission checks, they may obtain the output for mincore as before. However, if the permission checks fail, the user cannot obtain the information that mincore would otherwise disclose.
- This patch is insufficient
  - The container management engine shares common files across different containers through the copy on write mechanism.
  - This means that each container gets the illusion of exclusive access even though the underlying file is shared.

Why is this a fundamental problem?

- Since each container runs from a base image, Docker also optimizes the usage of images.
  - Instead of copying around all the image files for each newly created container, the container engine utilizes union filesystems. By doing so, the container engine achieves that all the containers share the files derived from the common base image as long as the containers do not change them. This is a footprint and performance enhancement. Removing this enhancement would significantly impact memory and performance.
- Removing the mincore functionality was not successful in the past and this was reverted by Linus.
  - "For Netflix, losing accurate information from the mincore syscall would lengthen database cluster maintenance operations from days to months. We rely on cross-process mincore to migrate the contents of a page cache from machine to machine, and across reboots."
- Even if we remove the mincore functionality, we would still be able to infer which pages are in the cache by measuring the time it takes to load the pages.

Threat Model

Threat	Vulnerability/Use Case	Asset	Impact 1-5 (High)	Likelihood 1- 5 (Frequent)	Risk	Control Recommendations
Malicious code in application (source)	Injecting Malicious code in build environment	Build environment	5	1	5	Static tool analysis and checking with DB for open source libs. Sandbox for checking in run time
Malicious code in application (binaries)	Injecting Malicious code in build environment	Build environment	5	1	5	Integrity checking tools. Comparisons and signatures.
Vulnerable code in application	Vulnerable code in build environment from imported packages, and in dev code.	Code	5	1	5	Static tool analysis and checking with DB for open source libs. Sandbox for checking in run time
Vulnerable libraries in the OS	Vulnerable code in build environment from imported packages, and in dev code.	Code	5	1	5	Static tool analysis and checking with DB for open source libs. Sandbox for checking in run time
Content injection	Injecting or replacing existing image in Container repository	Container repository	5	3	15	Signing every image before storing in repository and write protection on existing data
Repository compromise	Modification and tampering with the whole repository (e.g., modifying the image, signature or location)	Container repository	5	2	10	Protection and access control on the repository
Content tampering	Tampering with Docker image content that is stored in the Container repository	Container repository	5	2	10	Save the images content in an encrypted and signed fashion.
Content theft	Stealing the sensitive content from Docker image	Container repository	4	3	12	Save the images content in an encrypted fashion. Enable pulling of images only to production environment only.
Deploy malicious container	Deploying malicious container to the same run time environment.	Production environment	3	4	12	Allow only signed images to be deployed. Allow deployment only from specific container repo. e.g. ip based.

Exploiting: From weakness to vulnerability

- We were able to prove that mincore can be used to get cache information of files and libraries within the same container and between containers on the major cloud service providers.
- Attacking an organization or a cloud deployment would be most affective if one would be able to infect a common library code or a base image, since it will enable us to create a covert network that will be used for data mulling and data transfer out of the organization.
- The diagram in this slide illustrates how this vulnerability may be used to create a covert network of sensitive data.