Implementing Secure Isolated Containers in an Operating System Kernel

Aleksandar Andrejevic
Chair for Applied Computer Science
Faculty of Technical Sciences
University of Novi Sad
Novi Sad, Serbia
E-mail: andrejevic@uns.ac.rs

***Abstract*—This paper describes some of the common methods of operating system-level virtualization, as well as a PhD research proposal to implement such a system in an operating system which manages resources through a centralized object manager with namespace support.**

I. Introduction

The Monolithium Operating System is a new monolithic hobby operating system [1]. It is currently developed only for x86-architecture computers. It aims to be as simple as possible while still providing most of the features of modern operating systems. It's still very early in development, and is, therefore, subject to major changes. One of the goals of Monolithium is to see if a very simple but modular design, along with a very simple implementation, could lead to a reduced number of defects. It is important to note that Monolithium is not a Unix-based system [1].

II. Secure Isolated Containers

Containers are an operating system-level virtualization method, that use the kernel's own facilities to separate logically independent user-space systems from each other as well as from the host system. They are becoming increasingly popular as a method of separating an application from the operating system and the physical infrastructure it uses to connect to the network. The container is instantiated within the kernel of the operating system and virtualizes the instance of the application. The popularity of containers and their use is discussed in [2].

Many current implementations of operating system-level virtualization exist. The simplest one, available in virtually all Unix-based operating systems, is *chroot* [2].

Chroot is a Unix system call that changes the root directory of the process that calls it, as well as all processes cloned from that process. By doing this, it restricts filesystem access to a certain subtree of the global filesystem, thus presenting a separate, virtual filesystem tree. If a process drops privileges between calling *chroot* and executing a different program, the newly executed program will not be able to call *chroot* itself, and will therefore be "trapped" inside the subtree. A major issue with *chroot* is that it is often unacceptable to run programs without superuser privileges [2].

Linux Containers is a widely used form of operating system-level virtualization on GNU/Linux. It uses *cgroups*, a feature of the Linux kernel, to separate the userspace instance. A project called *Docker* aims to provide automation for the deployment of applications inside containers, and it is already widely used today [2].

Nevertheless, there are several issues with Linux Containers that remain to be resolved, such as efficiency, sharing resources and the complexity of the resource limits configuration [3].

Another implementation of virtualization, developed only for the FreeBSD operating system, is known as the *FreeBSD jail*. It is basically an extension of the *chroot* system, that provides a more secure environment than *chroot* alone, while allowing superuser privileges inside the jail. Still, jails are limited, and it's not possible to mount or unmount disks, change the network interfaces, nor modify the kernel from within the jail [4].

III. Implementation in Monolithium

Monolithium manages certain resources such as processes, threads, memory sections, synchronization objects, and files, using an object-based model [1]. This system could be modified to group objects into namespaces to facilitate the separation of certain groups of objects. Currently the object management system is incomplete and not all resources are managed through it, but this issue will be resolved in the near future.

In general, each process would only be able to enumerate and access objects in its own namespace. Creating, deleting, and switching to a different namespace are privileged operations, which means access to them is controlled by Monolithium's privilege system. In other words, they can only be executed by processes running as a user with the appropriate privilege.

Since processes in Monolithium "open" objects to work with them, handles to already opened objects will not be lost after a namespace switch. This could be used to implement resource sharing between containers.

This system could also be made recursive, if namespaces had their own hierarchy. Each namespace would have its own set of users, and therefore, its own superuser which could

create namespaces but only under the restriction of the current namespace. Only the superuser of the root namespace can access all the objects in the system.

Theoretically, it should be possible for programs running inside containers to privately mount devices in their own namespace. Many problems associated with mounting devices inside containers on Unix-like systems are avoided thanks to the robust privilege system, as well as the lack of *setuid*-related security risks.

The major difference between this approach and other implementations of containers is that an object-based system with namespaces works in the lowest level of the operating system kernel, and doesn't represent an extension of the kernel's feature set, but rather an innate ability of the kernel to split the user-space into multiple, and possibly nested, domains.

IV. Conclusion

The hypothesis of the proposed research is that the object-based model of Monolithium could provide a better foundation for operating system-level virtualization than the typical file system abstraction provided by Unix-like operating systems. This is the main idea upon which the PhD research proposal described in this paper is founded.

The expected advantages to implementing containers in the object manager of an operating system kernel are simplicity of the design, improved code readability, and enhanced performance.

The biggest obstacle for achieving this goal is the incompleteness of Monolithium, and its constantly shifting design. The first issue that needs to be resolved on the research road map is the management of objects outside of the object system, such as user accounts.

References

[1] Aleksandar Andrejevic, *An Operating System Kernel for x86-Architecture Computers*, 2016 (in Serbian)

[2] Scott Hogg, *Software Containers: Used More Frequently than Most Realize*, http://www.networkworld.com/article/2226996/cisco-subnet/software-containers--used-more-frequently-than-most-realize.html (Accessed: 01 Feb 2017)

[3] Nathan Willis, *Seven problems with Linux containers*, 2014, https://lwn.net/Articles/588309/ (Accessed: 01 Feb 2017)

[4] Poul-Henning Kamp and Robert N. M. Watson, *Jails: Confining the omnipotent root*, 2000, http://phk.freebsd.dk/pubs/sane2000-jail.pdf (Accessed: 01 Feb 2017)