Securing Your Containers

An Exercise in Secure High Performance Virtual Containers

Adam Miller

Department of Computer Science
Sam Houston State University
Huntsville, TX 77341
ajm023@shsu.edu

Lei Chen

Department of Computer Science
Sam Houston State University
Huntsville, TX 77341
LXC008@shsu.edu

Abstract – ***In this research we introduce a new way for container based virtualization to be used in a highly secure fashion. As the industry requires for condensing enterprise infrastructure, the need to use virtualization technologies is becoming a necessity. In the realm of virtualization technologies there are many popular hypervisors (also called Virtual Machine Manager, or VMM), but they all fall short in terms of performance when compared to container based virtualization technologies. However, the virtualization technologies that utilize containers often become victims to security vulnerabilities in ways that hypervisors are abstracted away from. In this paper we introduce a security mechanism that can be used to thwart the shortcomings of popular container based virtualization technologies. We incorporate SELinux along with Linux Containers (LXC) that make use of a recent Linux kernel feature called cgroups. It shows that by incorporating these two technologies we are able to achieve both high level of security and high performance environment where container based virtual servers can run and be utilized for the enterprise.***

Keywords: Containers, Virtual Machine Manager, SELinux, virtualization, security, hypervisors

1 Introduction

Virtualization is the creation of the virtual version of system and network resources, including hardware, operating system, storage devices, and network resources [11]. The essential benefit of virtualization is that computer processing power is treated as a utility and service that clients can buy and subscribe. Cloud computing is considered a natural evolution based on virtualization. Virtualization aims to centralize administrative tasks while improving scalability and workloads.

In 1967 IBM introduced the System/360 Model 67 which contained the first recorded instance of virtualization. Nowadays companies such as VMware, Microsoft, Citrix (formerly XenSource), Red Hat, and Oracle all offer virtualization products of their own in order to try and win their place in the industry's datacenters. One thing all of these technologies have in common is that they rely on hypervisor technologies.

Hypervisors are an abstraction layer between the virtual machine kernel and the actual hardware allowing for "fake" hardware resources to be presented to each virtual operating system while leaving the management of the actual hardware resource to the hypervisor. This abstraction layer helps in many ways but at the same time adds overhead to the system as resource management is no longer solely dependent on the operating system but upon the hypervisor. If this overhead could be removed and containers within the operating system can be created to allow other "contained" instances of the operating system, it would thwart the speed impact of hypervisors [1].

The rest of the paper is structured as follows. Next in Section 2, we conduct background study of container based virtualization especially focusing its security concerns. Section 3 discusses the details of Secure Linux Containers, including Linux Containers, Research Environment, Implementation, Process Isolation, Test Design, and Research Contribution. Simulation, experiments and results are discussed in Section 4. Conclusion is drawn in Section 5 and references are listed in Section 6.

2 Background

While container based virtualization thwarts the speed impact of hypervisors, it introduces a new concern about security allowing different systems to exist within the same file system hierarchy. There will be an added layer of security needed in order to enforce this new virtualization construct. Lowering the number of context switches can be achieved by reducing the number of system calls required for the applications running within these containers [2]. Once the performance gains are introduced there will be a need to enhance the security measures by implementing a custom SELinux context per container.

The security advancements found in SELinux will bring the further isolation needed in order to keep the separate running instances of guest operating systems apart from one another by utilizing the SELinux Mandatory Access Control mechanisms [3][4][5][6][7].

In the industry there are a number of virtualization technologies clouding the landscape. The large commonplace issue is that all of these technologies are reliant on a hypervisor technology to manage the resources on the back end. This consequently introduces a heavy overhead that has proven to impact up to 28.1% higher CPU utilization [1] when compared to a much lighter weight solution such as the container based technologies. The main issue this new paradigm introduces is the concern of keeping compromised processes from exiting their directory structure and entering into a neighboring virtual environment. Our research targets this security concern and provides a solution to it.

3 Secure virtual containers

3.1 Linux Containers

There are a number of container based virtualization technologies today in the open source world, among which the more prominent ones are OpenVZ and BSD Jails. Our research focuses on an emerging technology, Linux Container (LXC), because of its deep roots in reliance on a Linux kernel level construct that breaks resources into “control groups” referred to cgroups.

LXC is based around cgroups which are implemented and executed via a virtual filesystem. This filesystem is mounted and its files are modified traditionally using shell scripts and GNU coreutils as will be demonstrated throughout the course of the research. The general hierarchy of the filesystem layout for cgroups is as follows in Figure 1.

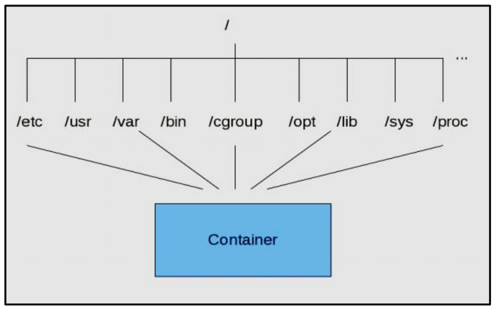


Figure 1. General hierarchy of filesystem layout for cgroups

Beyond the virtualization technology itself we will need to bring in the security aspect. SELinux is the de facto standard of high security in the Linux environment. The SELinux concepts revolve around using context for files, processes, and transactions and using policy to police these transactions by enforcing mandatory access control. The basic overview of this in practice is outlined in Figure 2.

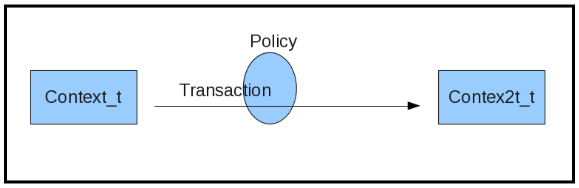


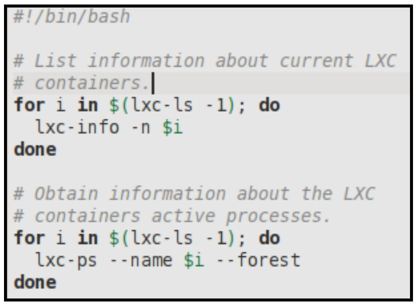
Figure 2. Mandatory Access Control

3.2 Research Environment

In this section we discuss the configuration and setup of the environment in which the research was performed. We used a system of modest resources in order to further highlight the performance gains. The system used in this research and development is Fedora GNU/Linux Distribution version 14 (codename Laughlin), and the version of LXC being used is 0.7.2 as per the release currently in the stable distribution repository at the time of this writing. The version of SELinux Utils that was used is 2.0.96 and the version of SELinux Targeted Policy is 3.9.7 also as per the releases available in the stable repositories. The LXC configuration and test setup was implemented using the example configuration files found in the LXC man (7) page for simplicity and ease of standardization.

3.3 Implementation

One of the most powerful elements of any UNIX-styled operating system is its shell environment combined with a special set of Core Utilities (conventionally known as coreutils). Code listing 1 shows a small example obtained from the LXC man (7) page that offers a glimpse into the powerful system level utilities that we have at our disposal.



```
#!/bin/bash

# List information about current LXC
# containers.
for i in $(lxc-ls -1); do
  lxc-info -n $i
done

# Obtain information about the LXC
# containers active processes.
for i in $(lxc-ls -1); do
  lxc-ps --name $i --forest
done
```

Code Listing 1. Example from LXC man page

This example offers insight into the abilities of the scripting environment built into the shell of the system that we used to administer the testing environment.

Next we need to create our SELinux context and policy in order to apply to our container. This was done using a combination of constructs, one being the new

SELinux type and the other being the policy as defined by code listing 2.

```
module my_container 1.0.0;

require {
  type unconfined_t;
  class process transition;
}

type my_container_t;
type my_container_exec_t;

role unconfined_r types my_container_t;

type_transition unconfined_t my_container_exec_t : process my_container_t;
```

Code Listing 2. SELinx policy module

This code listing is the SELinux policy module that provides the simplest configuration needed. We have created two new types and allowed unconfined types operating within our container's context to continue to do so but nothing else. All non-allowed actions will be denied so when we create the directory structure that will contain the root filesystem for the container based virtual machine it will be necessary to label the entire directory structure with our my_context_t type as well as set the default context of the highest level parent directory with this type, such that its children and processes executed from within it will inherit these context attributes. It is this premise that keeps our container secure [3][4][5][6][7].

3.4 Process Isolation

Now that the core security is in place by isolating an entire container based virtual environment confined within its own SELinux context and the policy has been created to keep the context transactions isolated within itself. The following is a fraction of the upstream Linux kernel documentation listed at the time of this writing for the 2.6.35 kernel from kernel.org:

"A *cgroup* associates a set of tasks with a set of parameters for one or more subsystems.

A *subsystem* is a module that makes use of the task grouping facilities provided by cgroups to treat groups of tasks in particular ways. A subsystem is typically a "resource controller" that schedules a resource or applies per-cgroup limits, but it may be anything that wants to act on a group of processes, e.g. a virtualization subsystem.

… " [8]

This is relevant as it describes a "subsystem" such as the LXC system that is being used in the course of this research. Without this ground work in which we can build upon, very little of this would be possible.

3.5 Test Design

With the LXC containers setup and configured, the SELinux policy and contexts in place, and the subsystem running we are ready to start to run some services within the environment in order to test the implementation. The easiest way to do this is to run an old outdated version of some server software with known vulnerabilities so that we can identify if our SELinux enforcement will actually protect the environment. The likely candidate is to run an older version of a web service program like Apache but to make it simple we chose to run a version of an interpreted module based language within the apache environment. Here we use php because of its wide popularity. It is target to many exploits and one of which we can use is a simple directory traversal exploit which will afford us the ability to determine if the SELinux containers are truly functional and can be done without causing any heavy amount of damage to our testing environment. Another reason why this is a solid example to show the security measures being enforced is that directory traversal vulnerabilities are often not due to an exploit in the implementation language such as php but of the code itself that was used to develop the web aspplication. The following listing (Code Listing 3) is an example of php code that leaves the server it is run on open to directory traversal. (NOTE: This code was obtained from wikipedia.org on 05/04/2011, author unknown.)

```
<?php
$template = 'red.php';
if ( isset( $_COOKIE['TEMPLATE'] ) )
   $template = $_COOKIE['TEMPLATE'];
include ( "/home/users/phpguru/templates/" . $template );
?>
```

Code Listing 3. Example of php code with traversal vulnerabilities

This could easily be exploited by an attacker using a properly constructed HTTP GET request to the server which would then warrant a response of either a directory listing or the contents of a file pending the permissions and SELinux context in place. The expected result of our research is that the potential attacker would not be able to obtain system level information by exploiting this aspect of our system regardless of the bugs introduced by novice web developers.

3.6 Research Contribution

The publication landscape in this field focuses on the architecture of the entire virtual stack with enhancements proposed in the area of hypervisor algorithms, resource management, and performance enhancements. These topics range from publications that cover the top to bottom architectures that are used in virtual environments [9] to those of performance characteristics based on the differences between various virtualization approaches [10].

With the introduction of the new research we can utilize the emerging technology based on highly developmental frameworks that are reliant on constructs which are now formally part of the upstream Operating System kernel. The system that has formed around this kernel by using the GNU userspace has been the basis for the currently most widely used Enterprise UNIX-styled Operating System as produced by Red Hat. By proposing research based upon already existing proven technologies that are heavily used in the industry, it is hoped that the methods could be quickly implemented in such markets. Another aspect of this research is built upon SELinux which is an enhancement originally introduced by the United States of America National Security Agency and is largely advocated as a default inclusion in the Red Hat Enterprise Linux Operating System. We used these two primary elements to combine for a more powerful secured virtualization construct.

The formal outline is simply that the performance gains of container based virtualization combined with the mandatory access control of SELinux will offer both enhanced performance and high level of security for virtual environments.

4 Simulation, experiments, and results

In this section we walk through the simulation environment that was utilized, the experiments performed upon the environment and present the results that were found in order to show that the research performed enforces the proposed result.

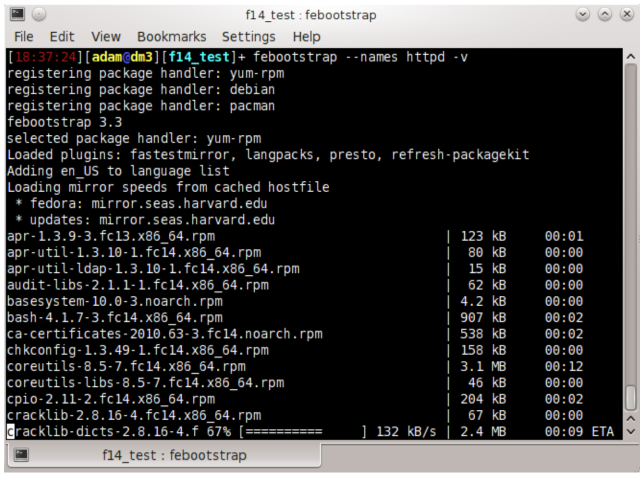
4.1 Simulation Preparation

We use the LXC constructs in order to define the container in which to run our virtual GNU/Linux environment. At the time of the writing the current stable release of Fedora GNU/Linux is 14 which is used in this research. We first configure an LXC environment using the example files provided by the LXC utilities manual pages in order to create a base LXC container. Then a utility called febootstrap is made for creating miminal bootstrapped operating systems within our previously constructed container. Figure 3 shows a screenshot of the creation of that contained virtual instance.

From here we initialize the apache instance and run a piece of known vulnerable php code in order to attempt the exploit against files from an adjacent container.

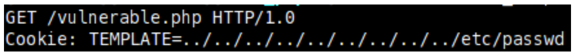
4.2 LXC Without SELinux Context

In this scenario we assume that the file permissions in an adjacent container were accidentally left in the state of "chmod 777" which allows all users all access to the files. In a situation where someone was running a virtual private server hosting company or similar business unit this is not an uncommon occurrence. In the following output we see that this directory traversal was easily obtained. Figures 4 and 5 show the HTTP GET request and successful response from the server respectively.



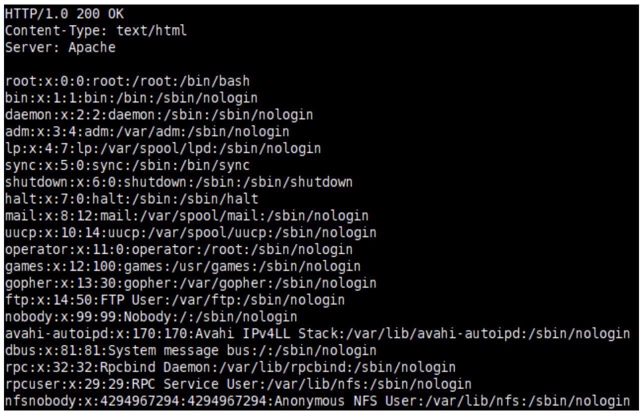
```
[18:37:24][adam@dm3][f14_test]+ febootstrap --names httpd -v
registering package handler: yum-rpm
registering package handler: debian
registering package handler: pacman
febootstrap 3.3
selected package handler: yum-rpm
Loaded plugins: fastestmirror, langpacks, presto, refresh-packagekit
Adding en_US to language list
Loading mirror speeds from cached hostfile
 * fedora: mirror.seas.harvard.edu
 * updates: mirror.seas.harvard.edu
apr-1.3.9-3.fc13.x86_64.rpm                               | 123 kB     00:01
apr-util-1.3.10-1.fc14.x86_64.rpm                         |  80 kB     00:00
apr-util-ldap-1.3.10-1.fc14.x86_64.rpm                    |  15 kB     00:00
audit-libs-2.1.1-1.fc14.x86_64.rpm                        |  62 kB     00:00
basesystem-10.0-3.noarch.rpm                              | 4.2 kB     00:00
bash-4.1.7-3.fc14.x86_64.rpm                              | 907 kB     00:02
ca-certificates-2010.63-3.fc14.noarch.rpm                 | 538 kB     00:02
chkconfig-1.3.49-1.fc14.x86_64.rpm                        | 158 kB     00:00
coreutils-8.5-7.fc14.x86_64.rpm                           | 3.1 MB     00:12
coreutils-libs-8.5-7.fc14.x86_64.rpm                      |  46 kB     00:00
cpio-2.11-2.fc14.x86_64.rpm                               | 204 kB     00:02
cracklib-2.8.16-4.fc14.x86_64.rpm                         |  67 kB     00:00
cracklib-dicts-2.8.16-4.f 67% [==========      ] 132 kB/s | 2.4 MB     00:09 ETA
```

Figure 3. Creating contained virtual instance



```
GET /vulnerable.php HTTP/1.0
Cookie: TEMPLATE=../../../../../../../../../etc/passwd
```

Figure 4. HTTP GET request



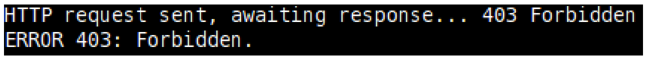
```
HTTP/1.0 200 OK
Content-Type: text/html
Server: Apache

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
avahi-autoipd:x:170:170:Avahi IPv4LL Stack:/var/lib/avahi-autoipd:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:4294967294:4294967294:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
```

Figure 5. Successful HTTP response

The output here shows that we are able to obtain the output from the vulnerable containers. This was the expected result and shows how this can be problematic in practice.

4.3 SELinux Enforcing

In the second scenario we have a configuration identical to the first in respect to the virtual container, but on the back end we are enforcing SELinux policy and we can see that the attempt to perform a directory traversal is thwarted by our SELinux enforcement. Figure 6 shows the screenshot.



```
HTTP request sent, awaiting response... 403 Forbidden
ERROR 403: Forbidden.
```

Figure 6. Attempt to perform directory traversal forbidden

5 Conclusion and future work

In this research we have shown that not only can we obtain highly available, high performance, and highly scalable virtualization infrastructure using container based virtualization [1] but we can also provide a high level of security inside these containers using the new paradigm enforced by SELinux. These concepts combined have proven to alleviate the host server administration needs concerned with the virtual containers from impeding upon one another. Future work will be to explore options of this application and to fine tune the approach for more sophisticated architectures.

6 References

[1] Stephen Soltesz, Herbert Potzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. "Container-base operating system virtualization: a scalable, high-performance alternative to hypervisors." *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (EuroSys '07), ACM, New York, NY, USA, 275-287.

[2] Yih Huang, Angelos Stavrou, Anup K. Ghosh, and Sushil Jajodia. "Efficiently tracking application interactions using lightweight virtualization." *Proceedings of the 1st ACM workshop on Virtual machine security* (VMSec '08). ACM, New York, NY, USA, 19-28.

[3] Giorgio Zanin and Luigi Vincenzo Mancini. "Towards a formal model for security policies specification and validation in the SELinux system." *Proceedings of the ninth ACM symposium on Access control models and technologies* (SACMAT '04). ACM, New York, NY, USA, 136-145.

[4] Gaoshou Zhai, Wenlin Ma, Minli Tian, Na Yang, Chengyu Liu, and Hengsheng Yang. "Design and implementation of a tool for analyzing SELinux secure policy." *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human* (ICIS '09). ACM, New York, NY, USA, 446-451.

[5] Bjorn Vogel and Bernd Steinke. "Using SELinux security enforcement in Linux-based embedded devices." *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications* (MOBILWARE '08). ICST, Brussels, Belgium, Belgium, Article 15 , 5 pages.

[6] Fabrizio Baiardi, Daniele Sgandurra. "Securing a Community Cloud." *Distributed Computing Systems Workshops, International Conference on*, pp. 32-41, IEEE 30th International Conference on Distributed Computing Systems Workshops, 2010.

[7] Gaoshou Zhai, Yaodong Li. "Analysis and Study of Security Mechanisms inside Linux Kernel." *Security Technology, International Conference on*, pp. 58-61, International Conference on Security Technology, 2008

[8] Kernel Docmentation on cgroups maintained by kernel developers abroad, the following were listed or original authors/modifiers: Written by Paul Menage <menage@google.com> based on Documentation/cgroups/ cpusets.txt

[9] Jeff Daniels. "Server virtualization architecture and implementation." *Crossroads* 16, 1 (September 2009), 8-12.

[10] Jeanna Neefe Matthews, Wenjin Hu, Madhujith Hapuarachchi, Todd Deshane, Demetrios Dimatos, Gary Hamilton, Michael McCabe, and James Owens. "Quantifying the performance isolation properties of virtualization systems." *Proceedings of the 2007 workshop on Experimental computer science* (ExpCS '07). ACM, New York, NY, USA , Article 6.

[11] Virtualization, retrieved on June 20th from *http://en.wikipedia.org/wiki/Virtualization*