tinyPM - leveraging the application-level resource control in the Linux environment

Kirill Belyaev
Colorado State University. Department of Computer Science.
Fort Collins, CO, USA

March 12, 2015

1 Project Description

1.1 Introduction

The application runtime access control to the underlying OS resources in the Linux environment has been traditionally regulated via the super user account privileges.

Within the system, a particular user's property is generally contained in files which are annotated with a numerical ownership user-identifier (UID). OS manages and abstracts the computer hardware, offering applications an environment in which to execute.

The classic UNIX model for wielding privilege is to assign a special UID the right to do anything. Applications running in the context of this super-user are not bound by the normal AC rules. They can read, modify, and change any user's data. In UNIX, UID=0 is the special context assigned to this administrative identity.

According to Unix security conventions, there are certain privileges reserved only for the root account. Privileged actions include things like binding a process to a privileged port, loading kernel modules, mounting and unmounting file systems, and a variety of other system activities.

While the simple UNIX privilege mechanism has more or less sufficed for decades, it has long been observed that it has a significant shortcoming: that applications that require only some privilege must in fact run with unnecessary full privilege. This leads to the possibility of programming errors in privileged programs that can be leveraged by hostile users to lead to full system compromise.

In version 2.1 of the Linux kernel, work was started to add what are known as capabilities. The goal of this work was to eliminate the dependence on the root account for certain actions. As of the 2.6 and 3.0+ kernels, this functionality has been successfully incorporated into mainstream Linux kernel and distributions.

The new proposed privilege model introduced a separation of root privilege into a set of capabilities. These capabilities break the super-user's privilege into a set of meaningfully separable privileges. In Linux, for instance, the ability to switch UIDs

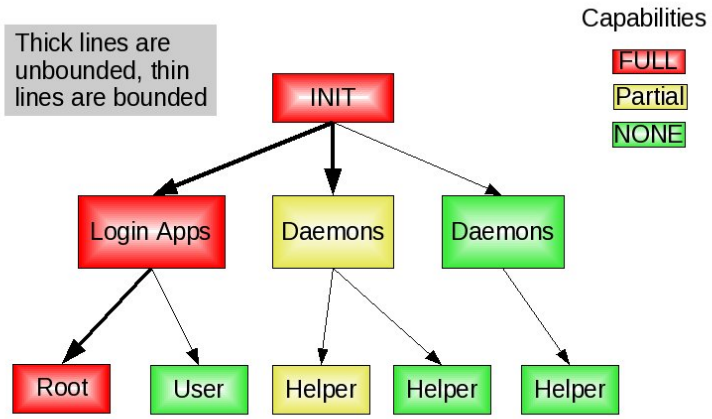

Figure 1: Linux Capabilities Model

is enabled by CAP_SETUID while the ability to change the ownership of an object is enabled by CAP_CHOWN.

A key insight is the observation that applications, not people, exercise privilege. That is, everything done in a computer is via agents—programs —and only if these programs know what to do with privilege can they be trusted to wield it. The UID=0 model of privilege makes privilege a feature of the super-user context, and means that it is arbitrary which programs can do privileged things. Capabilities, however, limit which applications can wield any privilege to only those applecations marked with filesystem-capabilities. This feature is especially important in the aftermath of a hostile user exploiting a flaw in a setuid-root program to gain super-user context in the system.

1.2 Application containers applicability

The new emerging concept of Linux application containers such as Docker service heavily leverages the Linux capabilities model. In the VM model each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.

In the application containers model such as the Docker Engine the container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

By default, Docker starts containers with a restricted set of capabilities. Capabilities turn the binary "root/non-root" dichotomy into a fine-grained access control system. This way we don't have to hand out full root permissions to some processes. For example

a web server normally runs at port 80. To start listening on one of the lower ports (below 1024), you need root permissions. Now this web server daemon needs to be able to listen to port 80, however it does not need access to kernel modules (that would be a serious threat to the integrity of the system!). Instead of giving this daemon all root permissions, we can set a capability on the web server binary, like CAP_NET_BIND_SERVICE. With this specific capability it can open up port 80 (and 443 for HTTPS) and listen to it, like intended.

1.3 tinyPM motivation and architecture

tinyPM is the lightweight management subsystem (written in Java) developed with the goal of making the management and tracking of application-level Linux capabilities easier. As we have already outlined it is best suited for server-based applications and container deployments where various types of applications are constantly deployed in isolated environments within containers with each container dedicated to servicing a single type of application in a security constrained environment. Most common examples include server-based containers to serve various system and user oriented services such as HTTP web/application servers, DNS server, NTP server, NFS server etc. tinyPM essentially allows to keep track and change/update the application privileges for the desired applications in easy manner. tinyPM consists of the following components:

- Database persistence layer - stores the capabilities privilege models on disk and allows to perform CRUD (Create/Read/Update/Delete) operations on them.
- Parsing engine - parses and processes the models given at the command-line-interface
- Shell - provides the command-line-interface to input the policy commands
- Enforcer - libcap wrapper module that is invoked by the parser to call the libcap system functions to manage Linux capabilities on a particular application

The following commands are currently implemented to manage and enforce the Linux capabilities model from tinyPM command-line shell interface:

- EXIT
- HELP
- SHOW_APP_POLICIES
- ADD_APP_POLICY
- DELETE_APP_POLICY
- COUNT_APP_POLICIES

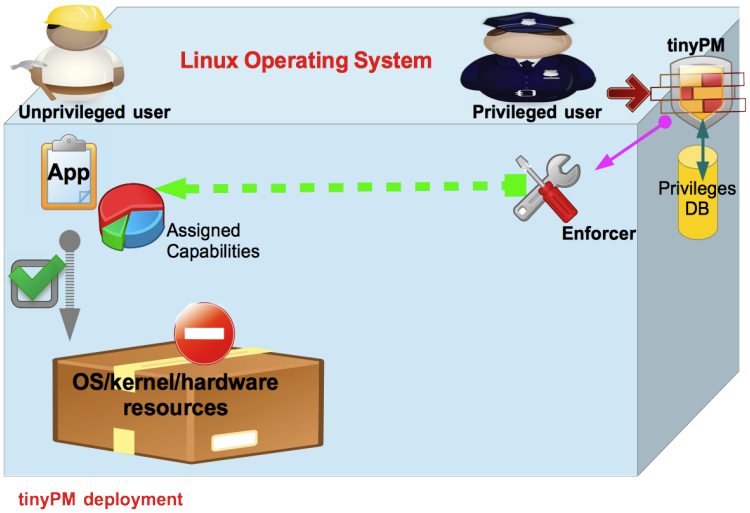

Figure 2: tinyPM deployment

1.4 deployment guidelines

tinyPM as a management utility must be run by the privileged user account. It is owned and invoked by the root account or a special account (created by root) that has the privilege to change the CAP_SETFCAP capability added to the tinyPM package binary. This capability allows the executable to change the capabilities of other applications binaries. So no other account except the privileged user is allowed to execute tinyPM to manage the capabilities. The rogue application will not be able to change the capabilities to gain root privileges and compromise the system due to underlying kernel level restrictions. Certain privileges can be bestowed upon it (in a transparent manner) by the tinyPM utility executed by the tinyPM owner (usually root) but without the explicit policy the application has no way of exploiting the Linux capabilities.

1.5 tinyPM versus Policy Machine

Unlike Policy Machine that is exclusively targeted for Microsoft Windows Server environments and works only through Active Directory tinyPM is geared towards Linux server deployments where it controls access to the OS resources for the specified application service either through application containers abstraction or through conventional installation. tinyPM does not directly manage users and is not dependent on user authorisation/authentication schemes using Microsoft Active Directory framework. It adheres to the key observation that applications, not people, exercise privilege (especially highly relevant in the server deployments). Therefore it is deployed in the transparent manner without the application even being aware it is actually regulated in terms of outlined capabilities privilege model.

Policy Machine model of unification of access control and data services gave us the idea of developing tinyPM prototype for a subset of a problem domain. Policy Machine has emphasized the importance of dealing with processes and applications as consumers of operating system resources. A subset of controlled delivery of data services that is a primary focus of PM is carried out in the Linux server container environment by the tinyPM concept. Linux capabilities model offers the basic data services operations executed by the application (such as read/write) on the data objects as well as more advanced privileges centered at the Linux kernel/OS operating environment resources.

1.6 Future directions

One of the immediate goals of the tinyPM project for the future is the addition of the distributed capabilities model where application capabilities could be deployed, managed and tracked on a cloud like installation with Linux servers located on the network. This is a very common real-world scenario seen in many data centers worldwide on a regular basis. Unlike the local host deployment this problem introduces a set of complex questions related to scalability, authentication and Access Control aspects imminent in the case of a large cloud-based (server-farm) deployments. A distributed coordination protocol should be designed and implemented to allow the orchestration and coordination of a multitude of tinyPM installations across a server farm. The model representation semantics is also expected to change since the representation of non-local policies are required to be expressed and enforced in a distributed manner. In the future deployments we also anticipate the requirements to manage and control application capabilities for a set of applications acting as a group with the incorporation of the time-based predicates.

2 References

1. http://man7.org/linux/man-pages/man7/capabilities.7.html

2. Hallyn, Serge E., and Andrew G. Morgan. "Linux capabilities: Making them work." In Linux Symposium, p. 163. 2008.

3. https://www.docker.com/whatisdocker/

4. https://docs.docker.com/articles/security/#linux-kernel-capabilities

5. http://www.friedhoff.org/posixfilecaps.html

6. Ferraiolo, David, Vijayalakshmi Atluri, and Serban Gavrila. "The Policy Machine: A novel architecture and framework for access control policy specification and enforcement." Journal of Systems Architecture 57, no. 4 (2011): 412-424.