# Simau: A Dynamic Privilege Management Mechanism for Host in Cloud Datacenters

Lin Wang[1,2], Min Zhu[1,2], Qing Li[1,2](✉), and Bibo Tu[1,2]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
{wanglin1993,zhumin,liqing,tubibo}@iie.ac.cn

[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

**Abstract.** Nowadays, a majority of cyber-attacks are associated with the insider threats owing to improper privileges management. Though a number of access control mechanisms have been carried out, the insider threats are continuously increasing. In cloud, however, the physical machines of datacenters are still exposed to danger. Without the trusted hosts as the foundation, any further measurements for virtual machines are in vain. In this paper, we introduce Simau: a mechanism that constrains the privileges of `root` on each host in the cloud. It deploys a decision engine in user-space to support the variable security policies. The scope of Simau covers both kernel-space and user-space. Under Simau, once a system administrator logs into a host, he has only the least privileges to finish his missions and all his requests for privileged operations are determined by Simau. The experiments at last show good performance of our mechanism.

**Keywords:** Insider threats · Root privileges · Privilege management Host · Cloud · Security

## 1 Introduction

With the popularity of cloud computing, the organizations who have transferred their local services to cloud datacenter are increasing. There is no doubt that the security of cloud datacenter draws a considerable attention. Cloud Security Alliance (CSA) published *Cloud Computing Top Threats in 2016*, pointing out that the breaches caused by improper access control are ranked the second [14]. Usually, the tragedy is caused by a worker who gains the privileges more than what he should have. Additionally, the security of hosts is the central premise of any security-related topic in a cloud datacenter. No protection for the hosts, no security for virtual machines. In conclusion, it is significant to apply appropriate privilege management to physical servers in cloud.

There are several circumstances when the cloud datacenters are posed to danger. Firstly, multifarious work of employees causes privileges abuse inadvertently. Moreover, once attracted by interests, the system administrator who grants the total control rights over the system may copy confidential archives out or remove the database, causing immeasurable damage. Furthermore, the maintenance workers are often delegated with `root` privileges when asked to troubleshoot or upgrade. Finally, there are attacks targeted for the `root` privileges like social engineering against administrator's password.

Existing work has separated privileges in hosts through the combination of Linux Security Module (LSM) [1] and SELinux [11]. LSM is an integrated structure mediating access to the kernel's internal objects through hooks. SELinux is a successful application for LSM that implements a kind of Mandatory Access Control (MAC) in the kernel. However, to protect hosts from insider threats, we need a dynamic privilege management mechanism which assigns the privileges on-demand. The decision logic of SELinux is fully implemented in the kernel. As the result, though SELinux is equipped with several kinds of strategies, they should be perceived as immutable without code recompilation. Furthermore, the hooks of LSM is fixed, which is not enough flexible. Apart from some important objects under the protection of LSM, there are some prominent executions which are independent of any objects, or it is hard to find a clear range of entities attached to them. To manage this kind of operations, we have to define custom hooks. For example, "To install or uninstall software", it seems that the objects are the files related to the software while, in fact, it is not easy to identify all files and directories associated with the software so that it is impossible to bind pertinent data items to the procedure. In that case, we can insert a hook to the installer to block the unauthorized operation, while LSM is unable to support the custom hooks.

This paper introduces *Simau* which actualizes dynamic privileges management for the hosts in cloud datacenters. Simau precludes the unauthorized process from performing privileged operations by inserting hooks that spread over the whole system. Certainly, the hooks of Simau in a host are not something of a novelty. Nevertheless, upon this foundation, Simau provides autonomy—it supports user-defined hooks. It not only supports the reuse of LSM hooks to protect kernel objects but also has the ability to implant tailor-made hooks to satisfy different demands, as well as in user-space. Simau isolates decision-making from enforcement point by deploying a decision engine in user-space so that the security policies can vary as required and lead to the alteration of execution result in hooks timely. These policies can be operated by remote controller or local administrators and are carried into effects immediately.

Our contributions in this paper are:

1. We propose a dynamic privilege management mechanism for hosts in cloud datacenters that takes effect on both kernel-space and user-space.
2. We deploy a decision point in user-space that dominates the decision timely according to varied policies.
3. We devise a method of inserting user-defined hooks to kernel on-the-fly without such great pains by livepatch.

The rest of the paper is arranged as follows: in the next section, we state the background, including an overview of Simau and threat model. Then, we introduce the design principles of Simau in Sect. 3. In Sect. 4 we depict the technical details of our prototype system. Experiment in Sect. 5 reveals the performance of our mechanism. The related work and conclusion are in Sects. 6 and 7 respectively.

## 2 Threat Model

In a typical cloud datacenter environment, the physical servers admit remote users and occasionally local maintenance. Simau that is distributed in each physical machine receives and enforces commands from controller. The controller is deployed in a central machine, keeping connection with each server host.

Trust Computing Base (TCB) is used for Simau to boot all components into a trusted initial state. We assume that Simau runs in a healthy environment with TCB, including secure hardware and operating system, as well as the built-in security mechanisms. The components of Simau are well protected by some process protection measurements [9,10]. For the controller, we assume that the administrator of the controller is not allowed to operate on hosts locally and he will not be in collusion with one who may have the local access to hosts.

Given such a premise, our threat model is a single bad behaved employee who may perform some important operations on the hosts in a cloud datacenter, such as a service manager who has the right to start or stop a couple of crucial services, a system administrator who gains the `root` privileges on a certain host, or storekeeper and cleaner who has the opportunity to access physical machines. The employee may wield his power in wrong time, abuse his right or take the advantages of his job, as a result, cause great loss to an organization.

As a typical instance, a third-party maintenance worker is asked to upgrade software for an organization. In a traditional way, he may gain the administrator password and wield `root` privileges because he has to operate on some important directories. By this way, the worker obtains the additional rights that far more than he needs. He may plant a malware into the kernel, or steal the information asset through portable devices. If Simau, the worker will be assigned the privilege of installer only. He can neither insert kernel modules nor load the devices because Simau will refuse his requests.

## 3 Design

In this section, we introduce the design principles of Simau. Figure 1 shows the main components of Simau and the interactions among them. There are four main parts as the dash denotes: Policy Administration Point (PAP), Policy, Policy Decision Point (PDP) and Policy Enforcement Point (PEP). We will give the anatomy of each of them.

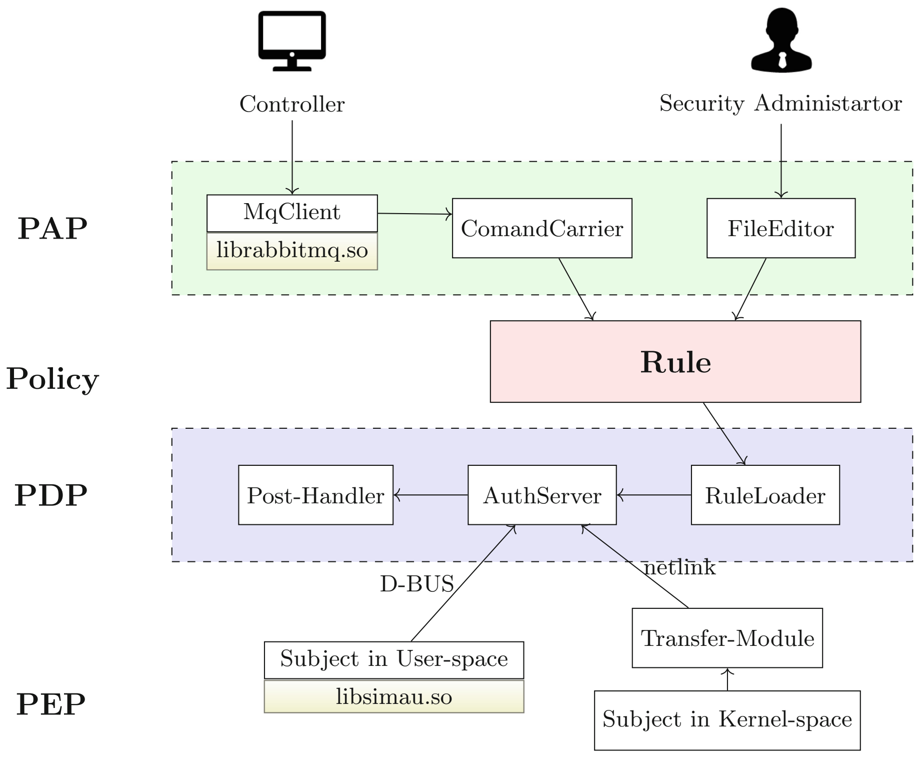

**Fig. 1.** Simau Architecture

### 3.1 Policy Administration Point and Policy

PAP is the interface for remote controller and local administrator to edit Rule files. MqClient which is an implementation of Message Queue is responsible for connection with controller. It receives messages from controller, parses them and passes the instructions to CommandCarrier to enforce the instructions on Rule files. The instructions are usually related to add a new rule, modify an existed one, or delete some rules. FileEditor in PAP refers to the common editor like *vim*. Security Administrators can carry local maintenance work out via editing the Rule files with the editors directly.

Policy is the security strategy we apply to our system. It exists in the form of rules set regulating how Simau performs.

### 3.2 Policy Decision Point

PDP is a central component where the decision logic is fully actualized. It has interactions with both Rule files and PEPs. Post-Handler, AuthServer, and RuleLoader are the main parts. RuleLoader loads the rules from Rule files whenever a modification of files is detected. AuthServer makes a decision on every request according to the rules. The form of request is similar to the rule that can be regarded as the binding of a set of requests plus the effect upon them. For example, "Reading a file that is created on 2017-9-1" is a typical request and "Reading the files that are created after 2017-8-26 is not allowed" is a rule. The primary jobs of decision-making are searching and matching. In this case, the request is matched to the rule and a negative value will be returned.

Post-handler offers an obligation mechanism for Simau. That is, some extra processes are supported after modification detected on Rule files to enforce the new policies. RuleLoader acts as the monitor to stare at the Rule files here. AuthServer becomes conscious immediately whenever new rules are loaded and it invokes a proper post-handler to perform some extra duties if any. For example, a rule reading "Log-in is allowed from 8 am to 5 pm." has an impact on the users who try to visit the system, rather than the online users, which is apparently irrational. The right way is that we kick out the illegal online users when the time is up so that we put all users under the control of Simau. In this case, when AuthServer is aware of the appending or alteration of rules related to log-in, it invokes the corresponding post-handler immediately to check whether the online users are in their valid time. This obligation mechanism can function as a "ruler" to make sure that every entity obeys our policies.

## 3.3 Policy Enforcement Point

PEP acts as the gateway to a privileged operation. It exists in a process in the form of a Simau-hook. *subject* is a process under the control of Simau with a served Simau-hook. Our hook separates the subject into two parts: one is the meaningful portion that we place control on, another is the unprivileged one. The meaningful portion often refers to functions or steps that directly have impacts on the outcome of procedures. If these functions were stepped over, the original procedure would fail. Simau-hook makes the "meaningful portion" be ignored if the subject is unauthorized no matter what other privileges it has been delegated by other mechanisms. Because implementation details of kernel-space and user-space are different, the subjects are segregated into user-space subjects and kernel-space subjects.

The Transfer-Module is designed for subjects in kernel-space and it works as a coordinator between AuthServer and subjects. For the Simau-hook of subjects in kernel-space, since it has to be inserted into kernel, it is inevitable to modify the code. The purpose of Transfer-Module is to share part of the responsibility and decrease the workload of adjustment. As is known to all, the communication with user-space in kernel is not an easy task. Without Transfer-Module, the new code will make the original segment long and convoluted. After alleviating the burden, Simau-hooks in kernel are only responsible for two necessary functions: communication with Transfer-Module which in kernel and collection some information for request constructing. It is obvious that to communicate with Transfer-Module which is written as a Loadable Kernel Module(LKM) is simpler than that with AuthServer.

### 3.4 Workflow

A subject will be trapped in the place where the Simau-hook serves. Under the impact of Simau-hook, the Simau authorization is carried out. If the subject is authorized, it would be allowed to continue, otherwise, it would go to fail. Note that Simau can coexist with other mechanisms like ACL or DAC. The combined policy of them is negative-override which means if any of them gains a negative value, the subject would go to fail.

The arrows in Fig. 1 indicate the workflow during an authentication. When the process arrives at a PEP, it generates an access request including the action identity and other collected elements that help match the right rule. For PEPs in kernel, the request is passed to Transfer-Module by a direct function call. Once the Transfer-Module receives the access request, it reconstructs it in a formal way and sends it to AuthServer through netlink. For PEPs in user-space, the request is forwarded to AuthServer by D-Bus [16]. AuthServer searches the matching rules for the request sequentially and the matching rule's answer will be returned. If there is no outcome for the request after scanning all the rules, a failed value is returned. As soon as the final decision is obtained, AuthServer returns the result to Transfer-Module or to PEPs in user-space directly. Finally, PEP gets the reply and enforces it.

Another flow that has an association with AuthServer is the interaction with a post-handler. Once Rule files are modified through PAP, either alteration or appending, AuthServer will be informed by RuleLoader immediately and post-handler is activated to perform its duty if provided.

## 4 Implement

According to our design principle, we realize a prototype of the system. In this section, we will give the technical details of main components. The last part is a demonstration of our prototype system.

### 4.1 Communication

The method we use in communication with a remote controller is message queue. The message-oriented middleware protocol is Advanced Message Queuing Protocol (AMQP) [17]. MqClient realizes both message queue consumer and publisher based *librabbitmq.so*. *librabbitmq.so* is an open-source C-language AMQP client library.

The interplay between AuthServer and Subjects is of paramount importance during an authentication. To provide service for subjects in both user-space and kernel-space, we apply two Inter-Process Communication (IPC) methods—D-Bus and netlink. D-Bus is for subjects in user-space. AuthServer will expose its authorization API on D-Bus. The subjects can lunch a call to AuthServer and get a reply from it through the bus. Netlink, as well as the Netlink socket family, is a Linux kernel interface used for IPC between user-space and kernel-space. As depicted above, a Transfer-Module is designed for sharing the load of Simau-hooks in subjects of kernel-space. When it is initialized, it registers a special protocol via which Transfer-Module and AuthServer can communicate with each other.

## 4.2 Hooks

PEP is inserted in a process in the form of a Simau-hook. If a program in user-space is going to use Simau, it has to invoke Simau authentication functions explicitly in code. One approach is to add the appropriate Simau-related functions in source code and recompile the program. The other effective way relies on the extensibility of the program. If the program has interfaces for custom binary in some crucial points, such as the point where *Linux-PAM(Pluggable Authentication Modules for Linux)* [18] is deployed, we can realize the Simau-related functions in them too. Otherwise, it is impossible to use Simau. Despite this, for user-space, to insert PEP is not a tough task because coding in user-space is free and easy compared with that in kernel-space. Hence the kernel-space is our main concern.

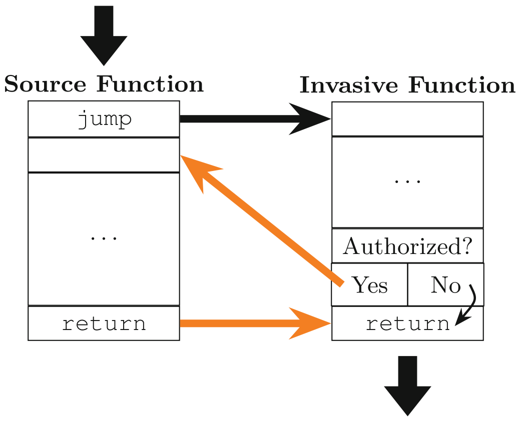

**Fig. 2.** Patch function structure in kernel-space

To plant PEP into kernel code escalates complexity of coding. To recompile or rebuild kernels unsuitable when reconstruction is not allowed. After all, for some operating systems, the source code is not available and it is also unwise to take great pains to install a software. As it is known to all, many patch methods have been actualized in kernel. For example, *Livepatch*, as the name indicates, is a small piece of code "sewn" on kernel to cover the original one. Figure 2 is the principle diagram. To be vivid, combining theory with livepatch, we call the original kernel code *Source Function (SF)*, and the covering code *Invasive Function (IF)*. The black arrows show the mechanism for live-patch itself. The patch plants a `jump` instruction as a tamp at the very beginning of every function, namely SF in our picture, the destination of which is the first instruction of IF. When the process runs into the SF, the `jump` instruction is executed immediately after setting the runtime environments. Then, the next construction to be executed is redirected to the IF. The runtime environments, like stack or heap, however, are that of SF, because this `jump` can be perceived as the conditional branch in a sequential piece of code of `if`. When the process runs into `if`, there are two paths to jump to and has nothing to do with the runtime environments. So that, when the process runs into `return` in the IF, the address that invoked the SF is returned as if the IF were the invoked function.

The orange arrows are the flow of our mechanism. The code of PEP is written in IF. When the process is redirected to the IF under the force of patch, the PEP will perform its duty, to collect some elements etc. The communication method we used in kernel-space is the calling of the external function and synchronization primitive. Then, after receiving a message from Transfer-Module, the PEP will enforce the result commands. If it is authorized, the process will go to the next instruction of `jump` in the SF and the SF is processed as the way it is. Note that, the SF discarding the first instruction is taken as a complete function to be invoked here. That is, a function call happens and the new runtime environment for the SF without `jump` is set. Hence SF will be back to IF under the action of `return` in the last. Otherwise, if it not authorized, the SF is skipped completely and the process goes to the next instruction right after authorization, as the black arrow from "No". Finally, the IF is returned with an error code indicating the failure of this function. The operation will be redirected to error treatment program. In this way, we deploy the PEP into kernel without such much cost.

## 5 Experiment

In this section, we measure the load that brings about by the Simau-hook. The experiments are carried in a test-server (Linux 4.4.0-87-generic, dual-core 64-bit Intel Core i5-3470 at 3.20 GHz, with 6MB/core cache and 2 GB memory).

We examine the running time of an authorized process with Simau-hook in both user-space and kernel-space. Log-in is for the test in user-space. Since Simau-hook has been inserted into PAM authentication management procedure, the running time of PAM authentication is examined. We test the spending time of PAM authentication management with Simau-check and record the average value which is shown in Table 1. As shown in the second column, PAM authentication management with Simau-check spends 6.095e−3 s on average. Likewise, we record the time spent without Simau-check in the third column.

LKM is for the test in kernel-space. We examine the spending time of command `insmod` and `rmmod` respectively. Similarly, we have calculated the average time out. *Time delta* in Table 1 reveals that the cost of Simau-check is negligible because all of them are no more than 1e−3 s which a human being can hardly feel about.

**Table 1.** Performance results of Simau

| Action | with Simau-check(s) | without Simau-check(s) | Time delta(s) |
|---|---|---|---|
| Log in | 6.095e−3 | 5.892e−3 | 2.03e−4 |
| Insert module | 1.924e−4 | 1.540e−4 | 3.84e−5 |
| Remove module | 1.870e−4 | 1.553e−4 | 3.17e−5 |

## 6 Related Work

The popular solutions to defend against insider threats are Access Control Mechanism [2,7,8,20] and the various variants of them [3]. However, all of them pay attention to user-space with complicated policies. SELinux [11] are mechanisms in kernel-space based on LSM [1] but it is not flexible enough because it takes great pains to change security policies. CAP [4] and LandLock [6] implements an explicit function to alter policy but it is limited to a single process. AppArmor [21], known as a simple version of SELinux, aims to constrain a process with limited resources. In contrast, Simau provides global control for each host instead of a single process.

Container [5], Jails [12] and Zone [13] are intended for providing a isolated area to confine privlieges. There is no doubt that they are not suitable for administration because the user has a limited perspective of the whole system.

TOMOYO Linux [22] and other security OS [23] proposed a new kind of OS or OS module to manage the privileges of user and process. Simau, on the other hand, does not require to recompile the kernel or replace OS.

## 7 Conclusion

We propose a dynamic privilege management mechanism for hosts in cloud datacenters. Our mechanism supports user-defined hooks to block processes and takes effects in both kernel-space and user-space. We deploy decision engine in user-space to actualize the variable security policies that can be altered on-demand. We further make a study on some important operations and try to regulate them under Simau. The experiment proves the ability of our mechanism. Finally, since it offers a programmable access control for hosts and realizes the separation of the control panel and action, Simau has great compatibility with the software-defined techniques.

Software-defined infrastructure (SDI), for example, brings many new approaches for managing, monitoring, etc. within clouds [15]. It breaks the restrains of the hardware-centric infrastructure, establishing a flexible and scalable fundamental structure. Simau can be one of the feasible technique that provides the enable support for SDI. We expect that Simau could play his significant role in the future.

**Acknowledgement.** We would like to thank the anonymous reviewers for their insightful comments and suggestions. This work was supported by the National Key Research and Development Plan of China under grant No. 2016YFB0801002.

## References

1. Wright, C., Cowan, C., Morris, J., Smalley, S., Kroah-Hartman, G.: Linux security module framework. In: Ottawa Linux Symposium, vol. 8032, pp. 6–16 (2002)
2. David, F., Richard, K.: Role-based access controls. In: Proceedings of 15th NIST-NCSC National Computer Security Conference, vol. 563 NIST-NCSC, Baltimore (1992)
3. Rajkumar. P.V., Sandhu, R.: POSTER: security enhanced administrative role based access control models, pp. 1802–1804 (2016)
4. Hallyn, S.E., Morgan, A.G.: Linux capabilities: making them work. In: Linux Symposium, vol. 8 (2008)
5. Shalev, N., Keidar, I., Weinsberg, Y., Moatti, Y., Ben-Yehuda, E.: WatchIT: who watches your IT Guy?. In: Proceedings of the 26th Symposium on Operating Systems Principles, pp. 515–530: ACM (2017)
6. Landlock: programmatic access control. https://landlock.io/
7. Lindqvist, H.: Mandatory access control. Master's thesis in Computing Science, Umea University, Department of Computing Science, SE-901, vol. 87 (2006)
8. Li, N.: Discretionary access control. In: van Tilborg, H.C.A., Jajodia, S. (eds.) Encyclopedia of Cryptography and Security, pp. 353–356. Springer, Boston (2011). https://doi.org/10.1007/978-1-4419-5906-5
9. Costan, V., Devadas, S.: Intel SGX explained. IACR Cryptology ePrint Archive, vol. 2016, p. 86 (2016)
10. Azab, A.M., et al.: Hypervision across worlds: real-time kernel protection from the arm trustzone secure world. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 90–102. ACM (2014)
11. McCarty, B.: SELinux: NSA's Open Source Security Enhanced Linux. O'Reilly, Newton (2005)
12. Kamp, P.-H., Watson, R.N.: Jails: confining the omnipotent root. In: Proceedings of 2nd International SANE Conference (2000)
13. Price, D., Tucker, A.: Solaris zones: operating system support for consolidating commercial workloads. In: LISA, vol. 4, pp. 241–254 (2004)
14. Cloud Security Alliance: Cloud Computing Top Threats in 2016, Cloud Security Alliance, Top Threats Working Group, February 2016
15. Gu, G., Hu, H., Keller, E., Lin, Z., Porter, D.E.: Building a security OS with software defined infrastructure. In: Proceedings of the 8th Asia-Pacific Workshop on Systems, p. 4. ACM (2017)
16. D-Bus. https://www.freedesktop.org/wiki/Software/dbus/
17. Vinoski, S.: Advanced message queuing protocol. IEEE Internet Comput. **10**(6), 87–89 (2006)
18. Samar, V.: Unified login with pluggable authentication modules (PAM). In: Proceedings of the 3rd ACM Conference on Computer and Communications Security, pp. 1–10. ACM (1996)
19. Linux Programmer's Manual CAPABILITIES(7). http://man7.org/linux/man-pages/man7/capabilities.7.html

20. Ferraiolo, D., Chandramouli, R., Kuhn, R., Hu, V.: Extensible access control markup language (XACML) and next generation access control (NGAC), pp. 13–24 (2016)
21. AppArmor wiki. https://wiki.ubuntu.com/AppArmor
22. TOMOYO: A Security Module for System Analysis and Protection (2018). http://tomoyo.osdn.jp/
23. Santos, N., Rodrigues, R., Ford, B.: Enhancing the OS against security threats in system administration. In: Narasimhan, P., Triantafillou, P. (eds.) Middleware 2012. LNCS, vol. 7662, pp. 415–435. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35170-9_21