Est-il difficile de respecter le principe de moindre privilège sur Linux ?

Eddie Billoir, Romain Laborde, Ahmad Samer Wazan, Yves Rütschlé, Abdelmalek Benzekri

Est-il difficile de respecter le principe de moindre privilège sur Linux ?

Eddie Billoir
Airbus Protect
Toulouse, France
eddie.billoir@airbus.com

Romain Laborde
Paul Sabatier University
Toulouse, France
laborde@irit.fr

Ahmad Samer Wazan
Zayed University
Abu Dhabi
ahmad.wazan@zu.ac.ae

Yves Rütschlé
Airbus Protect
Toulouse, France
yves.rutschle@airbus.com

Abdelmalek Benzekri
Paul Sabatier University
Toulouse, France
benzekri@irit.fr

***Abstract*—Le principe de moindre privilège est un processus d'ingénierie qui consiste à comprendre les responsabilités des personnes pour leur attribuer le strict minimum des permissions nécessaires à la réalisation de leurs tâches. Historiquement et par défaut, Linux octroie l'ensemble des privilèges pour réaliser une tâche administrative, ce qui ne respecte pas ce principe. Avec les nouvelles réglementations liées à la protection des données personnelles, le principe des moindres privilèges doit être appliqué y compris pour les administrateurs des systèmes. Par exemple, avec les stratégies Bring Your Own Device ou le télétravail, l'utilisation hybride professionnelle/privée des équipements informatiques est de plus en plus répandue. Dans ce cadre, les administrateurs des parcs informatiques doivent gérer ces postes sans toutefois pouvoir accéder aux données personnelles des utilisateurs. Cet article analyse les outils disponibles à l'implémentation du principe de moindre privilège et leurs limites pour les droits administratifs de Linux.**

***Index Terms*—Linux, Principe de Moindre privilège, Security by design, Noyau Linux, Permissions, Capacités, Bring your own device, Contrôle d'accès, LSM, BPF**

I. Introduction

Le principe de moindre privilège (ou POLP pour Principle Of Least Privileges) est un processus d'ingénierie qui consiste à comprendre les responsabilités des utilisateurs pour pouvoir leur attribuer le strict minimum des permissions nécessaires à la réalisation de leur tâches au niveau des systèmes informatiques [1]. Le POLP doit s'appliquer à tous les utilisateurs et en particulier aux personnes qui administrent le système et qui usuellement possèdent tous les privilèges directement ou indirectement. En effet, avec les nombreuses réglementations liées à la protection des données personnelles et l'utilisation hybride professionnelle/personnelle des ordinateurs de travail, les privilèges administrateurs doivent être également plus rigoureusement restreints et contrôlés [2].

Le POLP peut s'appliquer à de nombreux critères qui correspondent généralement à un besoin métier. Dans cet article, nous limitons les critères du POLP au sens des fonctionnalités qu'offre le système Linux aux administrateurs et utilisateurs du système. Il y a donc plusieurs critères que nous prenons en compte qui vont d'une configuration simple, manipulant un à deux critères, jusqu'à une solution générique où les critères se limitent aux ressources que possède le système [3].

D'autre part, la fonction d'administration dans les sociétés et les organisations est généralement répartie dans plusieurs domaines. Par exemple, une organisation peut déléguer à une société tierce la gestion de la configuration réseau des postes d'un parc informatique. Il peut arriver qu'un équipement soit partagé entre administrateurs système et réseau. Le système est alors co-administré par deux organisations différentes. Par défaut sur Linux, les programmes de configuration réseau sont exécutés avec l'ensemble des privilèges même s'ils n'en ont pas le besoin, ce qui ne respecte pas le principe de moindre privilège et présente des risques nécessitant une confiance entre les deux organisations.

Dans cet article, nous présentons les solutions proposées par Linux pour les besoins du POLP. Ces solutions sont catégorisées en deux ensembles complémentaires : octroyer et restreindre les privilèges. Il faut s'assurer que l'utilisateur-administrateur ait suffisament de privilèges pour ensuite les restreindre sur des critères plus précis à chaque cas d'utilisation.

II. Le mécanisme *setuid*

La première solution fournie par le système Linux est de changer temporairement l'utilisateur qui exécute un programme par le propriétaire du fichier exécutable [4]. Par exemple, les utilisateurs qui ont le droit d'exécuter un programme avec 'setuid' détenu par 'root', peuvent détenir son contexte privilégié. Cette solution suppose que les droits du fichier exécutable soient correctement positionés, ce qui peut poser des problèmes de sûreté [3].

Le premier outil mettant en place ce mécanisme est 'su'. Celui-ci permet à un utilisateur de démarrer une session en tant qu'un autre utilisateur avec les identifiants de celui-ci. Cet outil requiert donc de connaître un mot de passe qui n'est pas censé être partagé. 'sudo' est un outil largement utilisé sur les systèmes Linux. Il permet de mettre en place un système de contrôle d'accès obligatoire pour le changement d'utilisateur [5]. Ses très nombreuses fonctionnalitées permettent à un administrateur de résoudre la plupart des difficultés d'attribution de droit d'exécution et de contrôle d'accès sur des programmes. Cependant, cet outil permet de changer l'utilisateur en tant que 'root' par défaut ou tout autre spécifié. Par conséquent, tous les privilèges de 'root' ou de l'utilisateur sont donnés au processus ainsi que tous les sous-processus créés. Par exemple, l'éditeur de texte 'vi' permet de lancer un shell. Lancer un processus 'vi' en tant que 'root' avec la commande sudo, permet donc de créer un shell avec les mêmes droits ainsi que tous les processus créés à partir de ce shell.

Le mécanisme setuid et les outils associés permettent de mettre en oeuvre le POLP partiellement en accordant les privilèges pour un programme donné, dans un temps donné qui est celui de la durée de vie d'une session ou l'exécution d'un processus. Cependant, ils ne permettent pas de contrôler complètement l'exécution du processus à qui on a octroyé les privilèges.

III. Les capabilities sous Linux

Les *capabilities* (ou *capacités*) sous Linux sont une solution de répartition des privilèges administrateurs détenus par le 'root' en un ensemble de privilèges détenus par les processus. Avec cette solution, il est possible de restreindre les privilèges pour chaque processus du système, permettant au principe de moindre privilège d'être plus finement appliqué pour les programmes, sous-programmes et processus enfants qui en ont le besoin. Avec cette solution, il est par exemple possible d'obliger 'root' à respecter les droits d'accès aux fichiers. Pour attribuer des capabilities à des programmes, un équivalent du setuid existe pour les capabilities. Linux a ajouté dans les attributs étendus des fichiers d'autres ensembles qui indiquent les capacités à octroyer au le programme lorsqu'il est exécuté [6]. Cependant, l'utilisation des attributs étendus pose un problème car à chaque mise à jour de fichier, les attributs étendus sont automatiquement supprimés. Si cela peut-être une bonne mesure de protection d'intégrité et de sécurité du système, cela impose aussi un coût de maintenance supplémentaire non négligeable.

Pour chaque processus, le noyau va conserver en mémoire 5 ensembles de capacités définissant l'état de la politique discrétionnaire et obligatoire des privilèges. Ces ensembles permettent de manipuler les privilèges pour un critère de temps d'utilisation, mais également de les répartir à travers ses processus fils (politique discrétionnaire). Si un programme exécute un autre binaire, c'est l'autorité du système qui définit les privilèges qui seront hérités (politique obligatoire). Par défaut sur Linux, une unique politique est appliquée concernant l'héritage des privilèges Linux. Celle-ci est intrinsèque au noyau [6].

Pour strictement mettre en place la solution, il est nécessaire d'inclure le code source des programmes et de leur dépendances dans le périmètre d'un projet. Ce prérequis est rarement atteint pour les systèmes car sur Linux, les logiciels libres et d'organisations tierces sont majoritaires dans la plupart des infrastructures. Même dans le cas où ce prérequis est atteint, la quantité de modification et de maintenance de la solution peut facilement dissuader de la choisir.

En admettant qu'un projet exprime le besoin d'implémenter cette solution sans inclure le code source des programmes utilisés dans son périmètre, le projet doit connaître les privilèges obligatoires et facultatifs par les programmes tiers pour chaque cas d'utilisation. Cette information devrait être donnée par les développeurs. Cependant, même les développeurs n'ont pas forcément connaissance de ces privilèges car l'utilisation des capabilities n'est pas explicite, c'est-à-dire qu'un développeur peut utiliser des appels système qui nécessitent des privilèges dont il n'a pas connaissance. Par exemple, l'appel système 'fork()' qui permet de créer des processus fils peut demander un privilège pour outrepasser les limites de création des fils [7], mais cette information n'est pas facilement accessible même dans la documentation, et le noyau ne donne aucune information quant aux capacités qui sont demandées pour exécuter un programme, ce qui ne facilite pas la remontée d'information vers l'administrateur. Le deuxième problème est que certains programmes ne gèrent pas du tout les privilèges. Par exemple, le programme 'newgrp' oblige que l'utilisateur soit 'root' pour fonctionner. D'autres logiciels décident à la place de l'administrateur les privilèges à octroyer sur le système pour leur programme. Par exemple, 'ping' s'installe en configurant les privilèges qui lui sont nécessaires. Sauf que le noyau possède une configuration qui permet à 'ping' de fonctionner sans aucun privilège. 'ping' s'octroie alors un privilège sans considérer le système correctement.

L'outil principal de gestion des capabilities proposé par Linux est 'setcap'. Il permet de manipuler les ensembles de capacités sur les fichiers. Le deuxième outil est un module de PAM nommé 'pam_cap' [8] . Ce module permet d'octroyer des capacités lorsqu'un utilisateur démarre une session, ce qui permet de créer des utilisateurs qui peuvent être privilégiés. Le problème de cette solution est qu'on ne respecte plus le principe de moindre privilège associé à un programme.

Le projet "RootAsRole" [9] propose d'appliquer un modèle de contrôle d'accès par rôle pour octroyer des capacités selon une commande et l'utilisateur assigné au rôle. Cet outil est plus respecteux du moindre privilège car dans sa dernière version de développement, il réduit les capacités de façon permanente pour un programme. Autrement dit, le processus créé par cet outil ne peut pas exécuter d'autres programmes pour s'élever d'autres privilèges. Il propose également un outil nommé 'capable' qui permet de connaître les capacités demandées pour un programme spécifique. Avec cet outil, il peut être plus simple de connaître les capabilities et de les octroyer via des rôles. Cependant, le projet est encore à l'état de preuve de concept et donc ne répond pas pour l'instant à des besoins de production.

Les capabilities sous linux permettent d'appliquer le principe de moindre privilège sur les cas d'administration d'un programme, un processus et ses sous-processus. De la même manière que pour le mécanisme setuid, les droits d'accès peuvent être configurés de manière discrétionnaire ou obligatoire selon les utilisateurs et/ou leurs droits associés. Cependant l'administrateur n'a pas possibilité de directement agir sur les processus, et doit déléguer cette tâche aux responsables de développement des logiciels utilisés.

IV. Les Linux Security Modules

Les Linux Security Modules sont des modules intégrés au noyau Linux permettant d'appliquer une politique de sécurité obligatoire sur le système dans son ensemble. Contrairement aux outils précédents qui permettent d'accorder des privilèges sur le système, les LSM restreignent les privilèges des processus. Le LSM est donc complémentaire aux technologies

précédentes. Par exemple on peut utiliser le programme 'sudo' pour lancer un programme avec tous les privilèges et confiner celui-ci via un LSM. C'est la méthode la plus répandue dans l'industrie pour durcir un système. Le problème avec cette méthode est que le processus possède quand même des privilèges. Avec cette méthode, 'sudo' outrepasse les mécanismes de droit d'accès des fichiers et ne se réfère qu'aux droits d'accès indiqués par le LSM. Et si le programme réussit avec une vulnérabilité à manipuler son environnement de confinement, il pourra user de tous ses privilèges librement.

Les LSM les plus répandus sont SELinux et AppArmor, ils permettent de confiner les droits d'un programme à sa stricte utilisation dans les cas d'utilisation identifiés [11]. Ces outils vont agir sur toutes les ressources systèmes et grâce à leur configuration générique, ils peuvent appliquer un modèle de contrôle d'accès tel que Bell-Lapadula [12] ou Biba [13].

Pour l'implémenter, SELinux propose un modèle de contrôle d'accès labelisé (LBAC) et avec des niveaux. Cet outil utilise les attributs étendus sur les fichiers. Ce qui n'est pas forcément une solution idéale comme expliqué pour les capabilities Linux. D'un point de vue sécurité, cet outil est suffisamment générique pour créer et manipuler des critères systèmes comme des critères associés au métier. Mais les coûts de cette solution peuvent rapidement devenir très élevés, que ce soit pour l'implémenter ou le maintenir. Le principal problème de ce module est qu'il est difficile de comprendre tous les critères sur lesquels agir. Et prendre une décision pour un critère sans le maîtriser par manque de temps ou d'expertise est un risque à ne pas ignorer. Nous évoquons le temps d'implémentation car la liste des critères est générique, et amène à manipuler des critères trop nombreux et précis vis-à-vis du besoin. Par exemple, pour confiner l'exécution d'un programme avec SELinux, il est nécessaire d'accorder le droit d'allocation de mémoire pour permettre l'exécution du programme. Ce détail peut être difficile à maintenir pour l'ensemble du système.

Concernant AppArmor, ce module propose un modèle MAC similaire mais plus simple que SELinux. Néanmoins, sa mise en oeuvre peut ne pas respecter le principe de moindre privilège. Par exemple, AppArmor préfère auto-configurer des ensembles de règles plutôt qu'une règle unitaire, ce qui l'amène à autoriser des actions qui ne sont pas forcément dans le cas d'utilisation du programme [10]. AppArmor et SELinux sont de bons exemples de la problématique d'utilisabilité des critères pour respecter le POLP : Comment équilibrer complexité et conformité vis-à-vis du POLP ?

Une contrainte de ces modules de sécurité est qu'ils doivent être déjà compilés avec le noyau. Si les LSM actuels ne répondent pas au besoin de simplicité, il faut recompiler le noyau Linux pour ajouter son propre module. Cela peut être une tâche difficile à maîtriser. Pour résoudre ce problème, depuis Linux 5.7, il existe les LSM BPF. Le BPF (Berkley Packet Filter) est un code injecté dans le noyau Linux qui écoute les appels pour exécuter un code venant de l'espace utilisateur, et le LSM BPF permet d'imposer une politique de sécurité à la volée. Le projet "KubeArmor" [14] qui date de fin 2021 utilise cette solution pour renforcer une politique de sécurité dynamique pour les solutions de confinement avec Kubernetes. Ainsi, la politique de sécurité est plus simple à déployer car elle répond au besoin de chaque environnement de confinement.

V. Conclusion

Sur Linux, restreindre les privilèges selon les critères choisis d'une politique de sécurité est un processus faisable mais fastidieux. Il est encore trop complexe d'accorder les privilèges qui sont strictement nécessaires aux besoins. Les capabilities Linux ne sont pas suffisamment développées pour être utilisables par un administrateur, même expérimenté. Pour résoudre ce problème à court terme, il faut modifier le noyau afin de remonter l'information des capacités demandées pour que les administrateurs et développeurs appréhendent le fonctionnement de cette technologie. À moyen terme, il est intéressant d'intégrer dans les processus de développement une documentation des privilèges requis et facultatifs par un programme.

Nous avons montré plusieurs mécanismes pour répondre au besoin du POLP. Nous avons montré qu'avec setuid, le contrôle du POLP est limité à la session. Les capabilities sous Linux répondent partiellement au besoin mais cela nécessite une simplicité et une appréhension sur son utilisation. Les LSM sont plus précis et complets, mais ils posent des problèmes d'utilisabilité. Il est donc nécessaire de trouver un moyen de combiner et équilibrer l'octroie des privilèges avec les moyens de les restreindre afin d'atteindre l'objectif du moindre privilège. Sans solution à ce besoin, il restera difficile de respecter le POLP sur un système d'exploitation Linux.

References

[1] "Least Privilege — CISA." https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/least-privilege.

[2] "CNIL - guide securite personnelle". Available: https://www.cnil.fr/sites/default/files/atoms/files/cnil_guide_securite_personnelle.pdf

[3] P. Samarati and S. C. de Vimercati, "Access Control: Policies, Models, and Mechanisms," in Foundations of Security Analysis and Design, Berlin, Heidelberg, 2001, pp. 137–196.

[4] "sys/stat.h" https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/sys_stat.h.html.

[5] "Sudo," Sudo. https://www.sudo.ws/.

[6] "capabilities(7) - Linux manual page." https://man7.org/linux/man-pages/man7/capabilities.7.html.

[7] "fork.c - kernel/fork.c - Linux source code (v6.1.6) - Bootlin." https://elixir.bootlin.com/linux/latest/source/kernel/fork.c#L2123.

[8] "Fully Capable - pam_cap.so." https://sites.google.com/site/fullycapable/pam_cap-so.

[9] A. S. Wazan et al., "RootAsRole: a security module to manage the administrative privileges for Linux," Computers & Security, p. 102983, 2022, doi: https://doi.org/10.1016/j.cose.2022.102983.

[10] A. S. Wazan, D. W. Chadwick, R. Venant, R. Laborde, and A. Benzekri, "RootAsRole: Towards a Secure Alternative to sudo/su Commands for Home Users and SME Administrators," in ICT Systems Security and Privacy Protection, Cham, 2021, pp. 196–209.

[11] "Linux Security Module Usage — The Linux Kernel documentation." https://www.kernel.org/doc/html/v5.0/admin-guide/LSM/index.html.

[12] D. E. Bell and L. J. L. Padula, Secure Computer System: Unified Exposition and Multics Interpretation, 1976.

[13] K. Biba, Integrity Considerations for Secure Computer Systems, p. 68, Apr. 1977.

[14] "Runtime protection for Kubernetes." https://kubearmor.io/.