StemJail : cloisonnement dynamique d'activités pour la protection des données utilisateur

Mickaël Salaün
mickael.salaun@ssi.gouv.fr

ANSSI

Résumé StemJail est une architecture de cloisonnement dynamique des activités utilisateur. Cette nouvelle approche du cloisonnement simplifie l'utilisation du contrôle d'accès en l'intégrant dans le *workflow* de l'utilisateur. Celui-ci devient capable de limiter les droits d'accès de ses instances d'applications et donc de circonscrire les modifications et les fuites potentielles d'information. La solution proposée par StemJail se veut pragmatique : l'intervention de l'utilisateur est minimisée, l'impact sur les performances est faible tout en préservant la compatibilité logicielle et sans augmenter la quantité de code privilégié qui s'exécute sur le système. Il s'agit de la version courte d'un article dont une version plus détaillée est disponible sur le site de la conférence et sur GitHub [1].

1 Introduction

Le modèle de contrôle d'accès discrétionnaire classiquement employé sur les systèmes GNU/Linux veut que l'ensemble des processus exécutés par le compte d'un utilisateur dispose des mêmes privilèges vis-à-vis des données auxquelles l'utilisateur a accès. En particulier, selon ce modèle, un programme légitime et un programme malveillant (résultant par exemple de l'exploitation d'une vulnérabilité au sein d'une application, ou d'une porte dérobée) disposent des mêmes droits. Ce modèle de contrôle d'accès est donc inadapté à une séparation des privilèges des applications en fonction de différentes activités que peut avoir un utilisateur.

Les mécanismes de contrôle d'accès obligatoire (*Mandatory Access Control*), tels que SELinux ou AppArmor, visent notamment à pallier les limites du modèle de contrôle d'accès discrétionnaire (*Discretionary Access Control*), en permettant à l'administrateur d'appliquer une politique de sécurité globale sur le système, qui affine et restreint les permissions accordées aux programmes. Toutefois, cette politique de sécurité est appliquée de manière statique et s'avère complexe à élaborer en pratique.

Cet article présente StemJail, une solution qui permet de répondre au besoin de cloisonnement d'activités utilisateur de manière dynamique.

1. https://github.com/stemjail

Cette nouvelle approche du cloisonnement simplifie l'utilisation du contrôle d'accès pour l'utilisateur. Notre preuve de concept permet de maîtriser les accès aux données utilisateur afin d'éviter des fuites ou modifications malveillantes, avec un impact négligeable sur les performances.

StemJail tire son nom des cellules souches (*stem cells*) capables de se multiplier mais surtout de se spécialiser. StemJail s'utilise sur un système GNU/Linux, sans modification intrusive, par des utilisateurs non privilégiés et en complément des contrôles d'accès déjà présents. L'intégralité du code source est disponible sur GitHub [1] sous licence libre.

2 Objectif de sécurité et fonctionnalités de StemJail

Contrôle d'accès : StemJail fournit un contrôle d'accès complémentaire sur les données accessibles à l'utilisateur en fonction de son activité, ce qui permet d'améliorer leur protection en confidentialité et en intégrité. L'utilisateur peut ainsi gérer les accès de ses instances d'applications en fonction de leur contexte d'utilisation. On s'intéresse uniquement au contrôle des accès légitimes pour le système aux fichiers utilisateur, ce qui exclut par exemple les attaques sur la *Trusted Computing Base* (p. ex. noyau, services du système, matériel...) ou encore les attaques par canaux auxiliaires.

Fonctionnement adaptatif : Notre solution est capable de s'adapter aux changements d'activité utilisateur de manière dynamique.

Accessible à tout utilisateur : StemJail offre à chaque utilisateur du système le contrôle des accès à ses données, mais sans nécessiter de droits d'administration [2].

3 Modèle de contrôle d'accès

3.1 Politique de sécurité définie par l'utilisateur

Lorsque l'utilisateur est la source de l'information, nommée *objet*, il est le premier capable d'identifier son niveau de confidentialité. Il peut par exemple déterminer le caractère public ou privé d'une information, ou si elle relève de la sphère professionnelle ou personnelle. L'utilisateur va donc devoir définir une politique de sécurité pour chacune de ses activités.

2. Les systèmes d'exploitation des *smartphones* (p. ex. Android, iOS) apportent une forme de contrôle d'accès par application. Ces systèmes sont conçus pour un utilisateur unique qui est également, dans une certaine mesure, administrateur de la machine.

Chacune des instances d'application (c.-à-d. processus) d'un utilisateur est appelée *sujet*. On caractérise un *domaine* par un ensemble d'*accès* symbolisés par des triplets *(sujet, action, objet)* autorisés dans le cadre d'une activité. Les différents domaines de la politique définie par l'utilisateur sont appelés *domaines finaux*.

3.2 Cloisonnement adaptatif par transition

Pour prendre en compte la politique configurée, on souhaite être capable d'inférer le domaine correspondant le mieux à l'activité en cours en fonction des droits d'accès de tous les domaines atteignables. Les recouvrements des domaines finaux forment des intersections qu'on nomme *domaines intermédiaires*. Chaque domaine intermédiaire correspond donc à un sous-ensemble de domaines finaux. Afin de respecter les droits d'accès aux données précédemment acquis, le type de transition envisageable consiste à transiter d'un domaine moins permissif vers un domaine plus permissif. Cette spécialisation de domaine permet donc de gagner des droits d'accès tout en respectant ceux du domaine source. L'objectif est qu'en configuration donnée, l'utilisateur n'ait pas d'action supplémentaire à effectuer pour que ses activités soient correctement prises en compte par la politique qu'il a initialement définie. Ceci permet également d'éviter des oublis ou erreurs de l'utilisateur et de gagner en fluidité.

La découverte d'une (future) activité commence donc par un domaine intermédiaire correspondant à un des sous-ensembles du domaine final de cette activité. Au fur et à mesure des demandes d'accès faites par un sujet, le système peut être amené à faire transiter le domaine de ce sujet pour autoriser ses accès. Cette spécialisation du domaine est fonction des accès possibles et demandés (p. ex. demande d'ouverture d'un fichier) par un sujet. On autorise donc des transitions d'un domaine intermédiaire vers un autre domaine (intermédiaire ou final) seulement si le domaine destination inclut le domaine source. On s'assure ainsi de toujours respecter la politique d'accès définie par un domaine final, autrement dit, par l'utilisateur.

Les ensembles à gauche de la figure 1 représentent trois domaines finaux : A, B et C. Chacun de ces domaines correspond à un ensemble d'accès qui peuvent être communs pour des domaines superposés. Les domaines intermédiaires respectifs sont ab, bc, ac et abc. Comme le montre le treillis à droite de la figure 1, il est possible de changer de domaine tant que le domaine destination contient le domaine source. Le déroulement du scénario de transitions par état de la figure 1 est le suivant :

1. Le premier état représente le domaine initial abc qui est l'intersection de trois domaines finaux. Tous les domaines précédemment

cités sont accessibles par transition, ce qui signifie que le sujet qui y est présent dispose potentiellement des droits d'accès des domaines A, B et C. Le point symbolise un triplet $(sujet, action, objet)$ correspondant à la demande du sujet.

2. Après une demande d'accès à une ressource uniquement accessible par le domaine A et B, afin de pouvoir y accéder, le sujet transite vers le domaine ab. Dans le deuxième état, il est maintenant uniquement possible de transiter vers le domaine A ou B. Le système a donc déduit que l'activité n'était pas celle du domaine C. Toute demande d'accès à une ressource uniquement disponible dans le domaine C sera désormais refusée.
3. Enfin, lors d'une demande d'accès vers une ressource uniquement accessible par le domaine B, le domaine du sujet se spécialise en B. Le domaine final, troisième état, est maintenant atteint. Les accès précédemment accordés sont conservés mais il n'y a plus aucune transition possible. Durant toute la vie du sujet, tous les accès qui lui sont accordés sont donc ceux définis par B ; c'est comme s'il avait toujours été dans B.

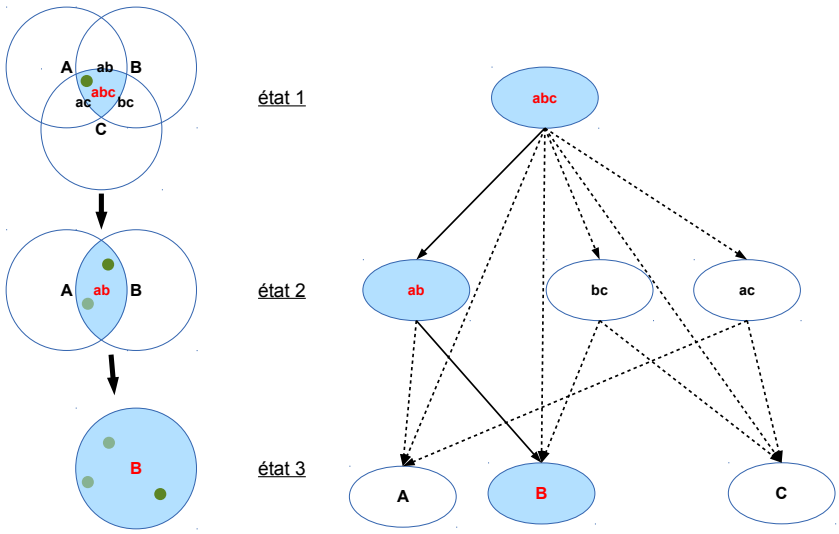

Figure 1. Spécialisation de domaine sous forme d'ensembles et de treillis

Afin que cette découverte de domaine fonctionne, il n'est pas nécessaire que tous les domaines aient un sous-ensemble commun. Il faut cependant

que la demande d'accès initiale corresponde à un de ces sous-ensembles minimaux. Dans le cas contraire, les accès sont refusés. Les demandes suivantes seront traitées séquentiellement.

Dans le cas où les demandes d'accès ne peuvent pas être délivrées par un des domaines atteignables, elles sont simplement refusées. Le sujet reste alors dans son domaine intermédiaire jusqu'à ce qu'une autre demande d'accès puisse éventuellement le pousser dans un domaine qui se rapproche plus d'un domaine final. La politique de sécurité définie est bien respectée et permet de circonscrire au domaine de l'activité une éventuelle compromission. Les données dont l'accès n'est pas autorisé par ce domaine ne seront donc pas accessibles à l'attaquant.

4 Mise en œuvre

Afin de permettre à chaque utilisateur de mettre en œuvre des fonctionnalités de cloisonnement natives à GNU/Linux, StemJail tire parti des espaces de nom utilisateur (*user namespaces*), offrant ainsi aux utilisateurs non-privilégié des fonctionnalités jusque là uniquement accessibles à l'administrateur de la machine. StemJail peut ainsi utiliser des *bind mounts* pour exposer les fichiers autorisés à être visibles, accessibles en lecture-seule ou bien en lecture-écriture. Par ailleurs, l'utilisation de mécanismes de sécurité déjà présents sur le système évite d'en augmenter la surface d'attaque. StemJail est développé en Rust pour ses bonnes propriétés : un typage fort, de bonnes garanties sur l'utilisation de la mémoire (sans *garbage collector*) ou encore la gestion sûre de la concurrence.

Comme le montre la figure 2, l'architecture de StemJail est composée d'une instance de moniteur par domaine. Chaque moniteur gère son domaine et peut créer un nouveau domaine (avec un moniteur dédié) inclus dans son propre domaine. Ceci peut être utile dans un but de sécurité en profondeur par l'ajout de restrictions supplémentaires.

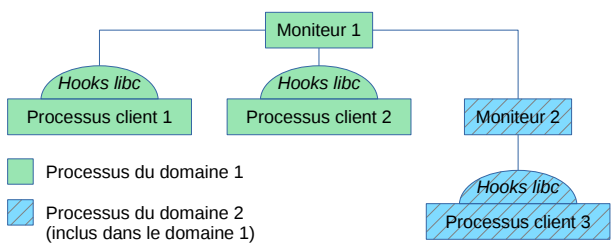

Figure 2. Exemple d'instantiation de l'architecture

Les processus de chaque domaine peuvent communiquer par socket UNIX avec leur moniteur afin de demander des droits d'accès supplémentaires. L'obtention de ces accès est effectuée par transition de domaine suivant les règles établies dans la section 3.2. Comme le montre la figure 3, nous dissocions la demande d'accès et l'accès effectif. Le client effectue d'abord une requête vers le moniteur pour demander l'accès à une ressource. Si la demande est légitime, contrairement au client, le moniteur est capable de faire évoluer le domaine avec une mise à jour des *namespaces* par le noyau. Enfin, le client peut accéder à la ressource si le noyau le lui autorise.

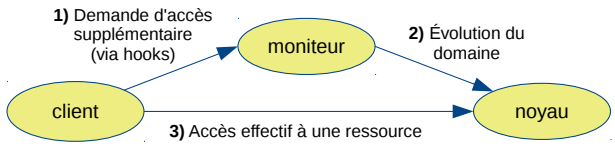

Figure 3. Communications entre les trois acteurs du contrôle d'accès

Afin de s'intégrer dans les systèmes existants sans modification (statique) des applications, nous mettons en place des *hooks* sur certaines fonctions de la *libc* (p. ex. `open`) grâce à la fonctionnalité `LD_PRELOAD` (via un fichier `/etc/ld.so.preload` dans chaque domaine). Cette approche a comme avantage de simplifier le maintien des *hooks* sans affecter les mises à jour du système. De plus, l'impact sur les performances est négligeable par rapport à des techniques comme `ptrace` qui impliquent de nombreux changements de contexte. Étant donné que les décisions d'accorder les droits d'accès demandés ne sont pas prises par les clients mais bien par le moniteur puis vérifiées par le noyau, **la sécurité du système ne repose donc absolument pas sur les *hooks* de la *libc***. Cette fonctionnalité permet cependant de rendre la majorité des programmes compatibles avec StemJail, le rendant utilisable dans des contextes concrets.

5 Conclusion

Nous avons présenté StemJail, une solution qui complète le contrôle d'accès existant en offrant la capacité à chaque utilisateur de maîtriser, par activité, les accès à ses données. La prise en compte des autres ressources du système (p. ex. réseau) est en cours. La conception d'une IHM de confiance prenant en compte les domaines des processus, afin de permettre aux utilisateurs de les distinguer, fait également partie des travaux futurs.