

VI. Sécurité.

1. Introduction.

Nous avons déjà expliqué la nécessité de protéger les processus des erreurs commises par d'autres processus. La protection mémoire et les modes (superviseur ou utilisateur) sont des mécanismes de base mis en œuvre pour assurer l'intégrité d'un système. Ces mécanismes de protection doivent être employés dans le cadre d'une politique générale et cohérente de protection qui fait l'objet de ce chapitre.

A. Motivations.

Les motivations de l'existence des mécanismes de protection dans un système multi-utilisateurs peuvent être résumées comme suit:

a) Protection contre l'erreur.

Les processus doivent être protégés des erreurs introduites par d'autres processus, aucune interférence non maîtrisée ne peut survenir entre deux processus. Que penser d'un système d'exploitation qui permet la modification accidentelle par un processus fautif d'une zone mémoire ne lui appartenant pas:

- les résultats du processus impactés seraient rendus aléatoires sans que ce dernier ne soit fautif;
- la détection de l'erreur serait rendue quasi impossible, il est difficile de remonter au processus responsable de l'erreur initiale.

De même, les utilisateurs doivent être protégés les uns des autres ou individuellement de leurs erreurs de manipulation; même un système d'exploitation mono-utilisateur doit se protéger des erreurs de manipulation de son utilisateur.

b) Protection contre la malveillance.

Les informations stockées et traitées par les ordinateurs constituent de plus en plus le seul patrimoine d'une entreprise: société de software, fichiers de clients..., il est donc essentiel d'en protéger l'accès tant en modification qu'en lecture sous peine de préjudice financier irréversible. Cette protection serait relativement simple si chaque processus ne devait partager les ressources du système avec d'autres, or nous avons vu précédemment que la majorité des ressources étaient partagées.

c) Protection contre les désastres ou les défaillances.

La duplication des éléments techniques constitue la protection de choix contre les désastres ou les défaillances techniques que peuvent subir les équipements informatiques. Les grands centres informatiques actuels disposent de ce qu'il est convenu d'appeler un site de back-up dans lequel on trouve les mêmes équipements et surtout les mêmes données que dans le site principal.

Cette duplication des équipements permet d'assurer la pérennité de l'entreprise en cas de désastre (explosion, écrasement d'un avion) touchant le site principal.

B. Protections.

Une sécurité optimale ne peut être basée uniquement sur des protections techniques offertes par un système d'exploitation. Le patrimoine d'une entreprise doit être protégé par différentes mesures:

- multiplication des sites informatiques basée soit sur une distribution des informations au sein d'un réseau soit par la maintenance d'une installation complète capable de reprendre la charge de travail en cas de dommages physiques subis par l'installation principale;
- protection physique des installations pour prévenir l'accès aux matériels, aux consoles...
- politique de back-up: prise de copies régulières, mise en lieu sûr d'une des copies permettant le déchargement des informations dans un autre site...
- redondance des équipements (double alimentation électrique, groupe électrogène, batteries...) ou des informations (disques RAID) ;
- attribution des privilèges d'accès et d'emploi des systèmes et de leurs ressources: listes de contrôle d'accès, mots de passe...
- protections incorporées dans le système d'exploitation: protection mémoire, instructions privilégiées...
- protections applicatives: validation des données présentées, vérification de signatures, checksum, contrôle de flux...

Toutes ces mesures doivent être mises en œuvre pour assurer une protection optimale; la moindre porte ouverte à un échelon quelconque de la liste précédente alors que toutes les mesures ont été prises à d'autres échelons, peut mener à la catastrophe. L'accès physique à une console peut annuler toutes les mesures prises en matière d'attribution de privilèges et de protection au sein du système d'exploitation. La sécurité globale d'un système est celle de son maillon le plus faible !

Nous nous contenterons d'aborder dans ce chapitre les techniques mises en place au sein d'un système d'exploitation pour assurer un niveau de sécurité correct. Toutes les autres mesures en matière de sécurité relèvent de l'organisation dans l'entreprise et de la conception des systèmes et réseaux. Elles ne doivent cependant en aucun cas être sous-évaluées.

La sécurité informatique impose la mise en place de procédures qui assurent l'intégrité des données. Cette intégrité peut être mise en péril soit par erreur soit par malveillance si les systèmes sont insuffisamment protégés. L'intégrité doit être également assurée par le système d'exploitation, les programmes qui le constituent doivent être non seulement corrects (c.à.d. délivrer le résultat attendu dans des conditions connues et maîtrisées) mais aussi fiables: délivrer des résultats corrects en cas de problème ou à tout le moins ne jamais passer sous silence un résultat incorrect. Un raisonnement valable pour un système d'exploitation peut l'être pour du code applicatif: des tests plus ou moins poussés prouvent que les différents composants d'une application fonctionnent correctement mais n'impliquent aucune conclusion sur la fiabilité de cette application. Seule une programmation défensive permet de rendre une application fiable: validation, redondance d'information,....

Le niveau de fiabilité dépend naturellement des exigences des utilisateurs. A l'optimum, on trouve des systèmes de contrôle d'engins ou de surveillance médicale. L'énergie à déployer pour produire un système fiable doit être concentrée sur les domaines suivants:

- production d'un système correct: techniques de programmation (encapsulation, programmation structurée...), tests...
- détection rapide de la faute: redondance des informations, interruptions en cas d'overflow, d'accès mémoire incorrect...
- élimination de la faute; cette élimination doit forcément être précédée d'une localisation de la faute (logging, dump mémoire...) ;
- réparation du dommage causé par l'erreur: remplacement de matériel, bug fix si possible sans interrompre le service.

C. Tiger Teams

L'histoire de l'informatique est pleine d'anecdotes racontant avec force détails l'intrusion de personnes bien ou mal intentionnées dans les systèmes. Ces intrusions ont pu avoir lieu soit grâce à une mauvaise conception du système (Unix, Windows, OS/360...) soit grâce à la négligence des utilisateurs (réseau interministériel belge, piratage de sites internet...).

Quelques consultants se sont fait une spécialité de s'attaquer aux défenses d'un système avec l'accord du propriétaire de ce système en vue d'en vérifier la robustesse. Ces professionnels sont souvent affublés du vocable *Tiger teams* ou *penetration teams*. Les moyens qu'ils emploient pour ouvrir une brèche dans un système sont :

- lire des informations en mémoire;
- tenter d'exécuter des SVC (Supervisor ou System, Calls) avec de mauvais paramètres;
- tenter d'exécuter des SVC, d'en comprendre la logique en lisant la mémoire et de surcharger certaines zones non protégées;
- tenter de perturber le déroulement de la procédure de log-in;
- tenter d'exploiter les faiblesses de certains softwares en exécutant des opérations non conseillées.

2. Principes élémentaires

Afin de sécuriser valablement un système, il est souvent nécessaire de mettre en place un composant (ou un logiciel) spécialisé qui met en œuvre quelques principes élémentaires:

- la conception du composant ne doit pas être publique: quelle est la valeur d'un algorithme d'encryption de mots de passe dont le code serait rendu public (sauf s'il est basé sur d'autres éléments secrets comme les systèmes à clef publique et clef privée)?
- l'absence d'autorisation implique l'absence de privilège ; autrement dit, les accès ne doivent être accordés que sur autorisation explicite, tout doit être interdit par défaut ;
- les autorisations doivent être vérifiées à chaque accès ou utilisation d'une ressource ;
- le minimum de privilèges doit être accordé, les privilèges généraux doivent être limités;

- ce composant doit soit faire partie intégrante du système d'exploitation soit en constituer une extension.

Les concepteurs du système vont mettre en œuvre une série de techniques:

- authentification (*authentication*),
- autorisation (*authorisation*),
 - protection des ressources,
 - profils ou rôles,
 - listes de contrôle d'accès,

afin de respecter quelques-unes des contraintes citées plus haut.

La forme la plus classique d'authentification d'un utilisateur consiste à lui demander l'introduction d'un mot de passe (*password*) plus ou moins complexe. Les procédures de login rendent obligatoires l'introduction de ces passwords qui sont encryptés¹⁰ et comparés au contenu d'un dictionnaire de password. En cas de correspondance, l'accès au système est accordé. Le composant chargé de la sécurité d'accès est responsable de la maintenance du dictionnaire, il peut exiger le changement d'un password à intervalles réguliers et compter le nombre de tentatives infructueuses d'entrer dans le système afin de désactiver le compte de l'utilisateur sujet de l'attaque.

A. Protection des ressources.

Comme nous l'avons déjà expliqué dans le cadre du chapitre consacré aux performances, le système d'exploitation gère des ressources de types fort divers: processeurs, mémoire, terminaux, fichiers, postes de travail dans un réseau... Chacune de ces ressources possède un nom unique et peut subir une série d'opérations: par exemple définition, lecture, écriture, exécution pour un fichier.

Ces ressources peuvent être rassemblées en domaines pour en faciliter la gestion, un processus ayant accès à un domaine aura accès à tous les objets inclus dans ce domaine.

Dans le même ordre d'idées, les utilisateurs peuvent être rassemblés sous des profils, ou rôles génériques s'ils possèdent des privilèges équivalents. On parle alors de « *role based access control (RBAC)* »

B. Listes de contrôle d'accès

La liste des utilisateurs et leurs privilèges peuvent être représentés sous forme d'une matrice dont les cellules contiennent le ou les privilèges d'un utilisateur (ou profil d'utilisateur) sur un objet. Cette matrice est normalement composée d'une majorité de cases vides. Afin d'en faciliter la manipulation, elles sont représentées sous forme de listes de contrôle d'accès (*Access Control List, ACL*). Nous reviendrons ultérieurement sur la manière dont ces ACL sont implémentées en UNIX.

A l'inverse, cette matrice peut être représentée comme une liste d'accès associée à un processus, on parle de *Capability Lists*.

¹⁰Un mot de passe n'est jamais sauvé en clair ! Seule une version encryptée est conservée, et comparée à l'encryption du mot de passe entré par l'utilisateur.

3. Comparaison Unix/MVS

Afin d'établir une base de comparaison valable, nous supposons qu'aucune intervention *hardware* ou provoquant un IPL du système n'est possible. MVS ne possède pas de sécurité en mode natif, à l'exception des mécanismes de protection de la mémoire (segmentation et protection *hardware*), on ne trouve que la possibilité de protéger un fichier via un *password*. En fait, aucun site au monde n'utilise MVS sans produit de contrôle d'accès. Les plus courants sont RACF (IBM), ACF-2 ou Top Secret (Computer Associates).

A. Intégrité de l'O/S

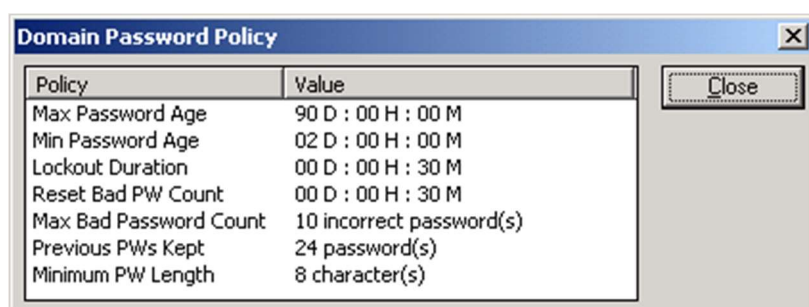
Tous les O/S de type *multi-users* doivent être bâtis afin d'éviter les interférences entre les processus ou entre un processus et le système d'exploitation. Ce dernier doit pouvoir faire une distinction entre un processus en *Supervisor State* et un processus s'exécutant en *Problem State*.

Normalement, un processus en *mode Supervisor* doit pouvoir lire n'importe quelle zone mémoire y compris des informations relatives à la sécurité. Si un utilisateur quelconque peut se faire passer pour le superviseur toutes les règles de sécurité sont inutiles. Les seuls moyens disponibles pour passer dans le mode superviseur sont les SVC ou l'exécution à partir d'une librairie ou directory autorisée.

Un système d'exploitation est intègre si son fournisseur s'engage à corriger tout problème d'intégrité et a construit les mécanismes assurant cette intégrité. L'Internet Worm est un exemple de non-respect de cette intégrité: la procédure *Finger* permettait d'exécuter une procédure passée en paramètre en cas d'*overflow* sur son *buffer*. En 1990, une mesure formelle de ce type de danger fût réalisée: sur 90 utilitaires de 7 versions UNIX différentes, 24% menaient à une possibilité d'intrusion suite au passage de mauvais paramètres. Un SVC doit être considéré comme un cheval de Troie.

B. Password

Un utilisateur doit être identifié de manière univoque par un *user identifier* (UID) ne serait-ce que pour des raisons de comptabilité d'emploi du système (*accounting*). Un *password* est associé à cet UID, il ne peut être lu par personne (y compris le *system administrator*). Des règles d'attribution de *password* doivent imposer la non réutilisation de *password*, une longueur minimum, la composition, la fréquence de changement.



Policy	Value
Max Password Age	90 D : 00 H : 00 M
Min Password Age	02 D : 00 H : 00 M
Lockout Duration	00 D : 00 H : 30 M
Reset Bad PW Count	00 D : 00 H : 30 M
Max Bad Password Count	10 incorrect password(s)
Previous PWs Kept	24 password(s)
Minimum PW Length	8 character(s)

Fig. 66. Une stratégie de mots de passe en Windows.

Il est excessivement dangereux de permettre un accès non sécurisé via des UID ne demandant pas de *password*, c'est le cas en Unix. En MVS, les seuls UID sans *password* sont de type *Started Tasks*, il est impossible de les utiliser à partir d'un terminal.

C. Partage des Uid

A côté de mauvaises pratiques présentes dans tous les systèmes, on trouve en Unix la possibilité de partager un UID : *Anonymous FTP*, partage du *Root Account (Superuser Uid)* en cas de partage des tâches de maintenance du système. En MVS, le produit de sécurisation permet d'attribuer de manière discrète les responsabilités sans devoir partager les UID. Les *Group Account* ou les fichiers *.rhosts* sont d'autres exemples de partage de UID. Ces derniers permettent à des utilisateurs distants (*remote*) d'accéder un account sans vérification de password. Les fichiers de type *.netrc* permettent à un utilisateur de se *logger* au *machine remote* en spécifiant son *user id* et son *password* sur cette machine.

D. Password Guessing

La tentative de lecture des *fichiers password* est une des activités les plus prisées par les pirates. En MVS, toute tentative répétée de violation d'un *password* conduit automatiquement à l'inactivation du Uid associé.

```
ACF82003 ACF2, ENTER LOGON ID - G36749
ACF82004 ACF2, ENTER PASSWORD -
ACF82000 ACF2, LOGON IN PROGRESS
ACF01012 PASSWORD NOT MATCHED
ACF82004 ACF2, ENTER PASSWORD -
ACF82000 ACF2, LOGON IN PROGRESS
ACF01012 PASSWORD NOT MATCHED
ACF82904 ACF2, SESSION TERMINATED.
***

ACF82000 ACF2, LOGON IN PROGRESS
ACF01013 LOGONID G36749 SUSPENDED BECAUSE OF PASSWORD VIOLATIONS
ACF82904 ACF2, SESSION TERMINATED.
***
```

Fig. 67. Réaction d'ACF2 en cas de password guessing.

Si le *fichier des passwords* (encryptés) peut être lu, une comparaison entre le *password* et une série de mots encryptés peut être rapidement menée, d'autant que dans un système ouvert, l'algorithme d'encryptage est connu et que le fichier des passwords doit être lisible suite à la conception du système.

MVS résout ce problème en ne permettant aucun accès au *fichier des passwords*. Afin de parvenir au niveau de sécurité C2¹¹, les *fichiers password shadow* ont été introduits en Unix.

¹¹ Niveau fixé par le Department of Defense.

Unix permet l'introduction de *password* « *case sensitive* ». Dans la même veine, un nombre de tentatives répétées d'introduction d'un password provoque une désactivation du UID.

Un problème commun aux différents OS est l'interception des *passwords* entrés à l'écran. Cette information peut circuler en clair dans un LAN. Des écrans simulant la procédure de *log-in* (ou *logon*) permettent l'interception aisée de ce type d'information. Ce problème s'est généralisé aux sites internet qui imitent un autre site demandant des mots de passe...

E. Administrator/Superuser

Chaque compte Unix possède un UID qui est utilisé par le système. Le UID 0 identifie le *Supersuser account* auquel toutes les actions sont permises.

En MVS, un processus peut utiliser certaines instructions dites privilégiées s'il est autorisé. Ce niveau d'autorisation est atteint si le programme exécuté est issu d'une librairie dite *APF-Authorised*. Les définitions systèmes déterminent quelle librairie est APF ou non. Une vérification soigneuse du code présent dans ces librairies fait partie des consignes de sécurité présentes dans toute installation *MVS*.

Un administrateur de la sécurité est bien défini en MVS, mais tous les accès de ce dernier sont enregistrés (*logged*). Une politique saine de la sécurité repose sur l'attribution de privilèges juste nécessaires pour exécuter les tâches, sans plus.

F. ACL

ACF-2 permet à un utilisateur ou à un groupe d'utilisateur de lire, écrire, allouer ou exécuter le contenu d'un fichier. Ces activités peuvent être permises, refusées ou *loggées*. Toutes les combinaisons sont permises pour n'importe quel fichier

Les privilèges d'accès en Unix étaient initialement spécifiés par 9 bits associés au fichier (Read/Write/eXecute), pour un *user* particulier (le possesseur du fichier), pour un groupe ou pour le monde entier (World). Ce système était fort limité, et pour y remédier, les ACL ont été introduites.

	Page/Namespace	User/Group	Permissions ¹⁾
#1	*	@ALL	<input type="radio"/> None <input type="radio"/> Read <input type="radio"/> Edit <input checked="" type="radio"/> Create <input type="radio"/> Upload <input type="radio"/> Delete
#2	*	bigboss	<input type="radio"/> None <input type="radio"/> Read <input type="radio"/> Edit <input type="radio"/> Create <input type="radio"/> Upload <input checked="" type="radio"/> Delete
#3	devel: *	@ALL	<input checked="" type="radio"/> None <input type="radio"/> Read <input type="radio"/> Edit <input type="radio"/> Create <input type="radio"/> Upload <input type="radio"/> Delete
#4	devel: *	@devel	<input type="radio"/> None <input type="radio"/> Read <input type="radio"/> Edit <input type="radio"/> Create <input checked="" type="radio"/> Upload <input type="radio"/> Delete
#5	devel: *	bigboss	<input type="radio"/> None <input type="radio"/> Read <input type="radio"/> Edit <input type="radio"/> Create <input type="radio"/> Upload <input checked="" type="radio"/> Delete
#6	devel: *	@marketing	<input type="radio"/> None <input checked="" type="radio"/> Read <input type="radio"/> Edit <input type="radio"/> Create <input type="radio"/> Upload <input type="radio"/> Delete
#7	devel: funstuff	bigboss	<input checked="" type="radio"/> None <input type="radio"/> Read <input type="radio"/> Edit <input type="radio"/> Create <input type="radio"/> Upload <input type="radio"/> Delete
#8	devel: marketing	@marketing	<input type="radio"/> None <input type="radio"/> Read <input checked="" type="radio"/> Edit <input type="radio"/> Create <input type="radio"/> Upload <input type="radio"/> Delete
#9	marketing: *	@marketing	<input type="radio"/> None <input type="radio"/> Read <input type="radio"/> Edit <input type="radio"/> Create <input checked="" type="radio"/> Upload <input type="radio"/> Delete
#10	start	@ALL	<input type="radio"/> None <input checked="" type="radio"/> Read <input type="radio"/> Edit <input type="radio"/> Create <input type="radio"/> Upload <input type="radio"/> Delete

Fig. 68. Exemple d'ACL.

G. Protection des Ressources

Chaque installation peut en MVS définir les ressources à protéger: transactions, commandes, classe d'exécution. L'utilisation de ces ressources est alors permise sur base de ces règles définies comme des règles d'accès à des fichiers. Chaque ressource potentielle s'identifie dans la cadre d'un appel au produit de sécurité.

```
SKEY(ACTIVATE) TYPE(DLA)
$USERDATA(DELTA IMS : ACCESS TO MAIN MENU)
UID(I**IEXE5423*G35101) SERVICE(READ) ALLOW
UID(I**ISUE10**DELTA) SERVICE(READ) ALLOW
UID(I**ISUE3103*G38758) SERVICE(READ) ALLOW
UID(I**ISYE1103) SERVICE(READ) ALLOW
UID(I**ISYE8303*G39849) SERVICE(READ) ALLOW
UID(I**R0*CSY01*G32093) SERVICE(READ) ALLOW
UID(I**R0*PLN) SERVICE(READ) ALLOW
UID(*****IOA) SERVICE(READ) ALLOW
UID(*****LOI) SERVICE(READ) ALLOW
UID(*) PREVENT
```

Fig. 69. Protection des ressources en ACF-2.

Le produit en charge de la gestion des ressources (DBMS s'il s'agit de bases de données, système s'il s'agit de fichiers...) doit faire appel au composant de sécurité afin de déterminer si l'utilisateur a le droit d'utiliser ou non la ressource. Cette vérification est effectuée en employant un interface standard dans le monde MVS: SAF pour System Access Facility.

H. Audit

SMF est un composant du système d'exploitation MVS dont la fonction est de collecter un grand nombre d'enregistrements signalant un événement dans le système : *Log-in*, *open* d'un fichier, *jobs steps*, ... Les *packages de sécurité* utilisent ce composant pour créer des enregistrements relatifs à un événement qui a été défini comme auditable: l'accès à un fichier par un certain utilisateur... Les données produites par SMF sont importantes dans la gestion d'une installation. Elles permettent entre autres de réaliser le suivi du système (gestion des ressources).

Les systèmes Unix conçus pour atteindre le niveau de sécurité C2 possèdent une fonction d'audit, les événements sont, comme en MVS enregistrés dans des fichiers particuliers qu'il est de bonne pratique de protéger.