MASTER EN SCIENCES INFORMATIQUES

Analyse des fichiers de sécurité sur réseau de stations SUN

Abdelmounaim, Guedira

Award date:
1993

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
rue Grandgagnage, 21, 5000 Namur

Analyse des fichiers de sécurité sur réseau de stations SUN

Guedira Abdelmounaim

Promoteur: Monsieur NAJI HABRA

Mémoire présenté en vue de l'obtention du grade de
Licencié et Maître en Informatique

Année académique 1992-1993

Résumé

Les systèmes à haut niveau de sécurité sont munis d'un mécanisme d'audit permettant la génération de fichiers appelés "Audit Trails" qui retracent toute interaction des utilisateurs avec le système. En effet, ces fichiers encapsulent des informations caractérisant les événements ayant eu lieu dans le système sous forme d'enregistrements appelés "Audit Record". L'analyse automatique de tels fichiers constitue un moyen efficace pour pallier à la violation de la sécurité des systèmes informatiques. Cette analyse peut s'effectuer de deux manières :

1.une analyse off-line : celle-ci permet de détecter les actions malicieuses et d'empêcher leur reproduction;

2. une analyse on-line : elle permet d'avorter en temps réel toute tentative d'intrusion.

ASAX (Advanced Security Audit trail on uniX) est un outil efficace pour analyser tout fichier séquentiel d'un format bien précis appelé "NADF (Normalized Audit Data Format). Il dispose d'un langage de règles "RUSSEL" (RUle-baSed Sequential Evaluation Language) permettant à l'utilisateur d'exprimer ses requêtes d'analyse.

Dans le cadre du mémoire, ASAX a été opérationnel pour l'analyse des fichiers de sécurité générés par le système SunOS. Tout fichier natif SunOS est converti en format NADF avant d'être évalué. La conversion est assurée par un format adaptateur réalisé au cours de ce mémoire. On a pu ainsi vérifier concrètement l'applicabilité de l'évaluateur ASAX à la détection d'utilisations malveillantes du système par l'élaboration de règles permettant de telles détections.

Abstract

High level security computer systems require auditing mechanisms allowing generation of so called "Audit Trail" that record all user interaction with the operating system. These files are a collection of audit records representing information about events initiated in the operating system. Preventing security violation of computer systems can be efficiently achieved by means of automatic audit trail analysis. Such analysis can be carried out :

1. off-line : to detect malicious actions and prevent them from recuring;

2. on-line : to achieve real-time abortion of attempted break-ins.

ASAX (Advanced Security Audit trail on uniX) is a universal, powerful and efficient tool for analysing sequential files in the so called NADF (Normalised Audit Data Format). Its related rule-based language RUSSEL (RUle-baSed Sequential Evaluation Language) allows to express queries on audit trails. The purpose of this thesis is to use ASAX for analysing security audit trails generated on a SunOS workstations. During this thesis, a Format Adaptor was implemented in order to translate native SunOS audit trails to NADF format. The applicability of the ASAX evaluator to the detection of security breaches and misuses was then proven by implementing rules in RUSSEL for such detections.

Remerciements

Je tiens à exprimer ma gratitude envers Monsieur Naji Habra, promoteur de ce mémoire, pour son aide constante et ses remarques judicieuses nécessaires à l'amélioration de ce travail.

Je remercie pour sa disponibilité et sa sympathie Monsieur Baudouin Le Charlier, concepteur d'ASAX, sans lui ce mémoire n'aurait pas de raison d'être.

Je suis également reconnaissant envers mon ami Aziz Mounji pour ses explications pertinentes, ses lectures vigilantes et son soutien.

Merci aussi à mon épouse pour sa patiente lecture, ses conseils et ses encouragements.

Je tiens à remercier ma famille et ma belle-famille pour toute leur attention à mon égard;

ainsi que tous les autres qui m'ont aidé.

Table des matières

Introduction

Ce mémoire porte sur la sécurité logique des systèmes informatiques (protection de l'information, contrôle d'accès, ...) et en particulier celle du système SunOS 4.1. Ce dernier est muni d'un mécanisme d'audit qui permet la génération de fichiers liés à la sécurité (appelé : Audit Trail). Ces fichiers d'audit renferment la trace de toute interaction avec le système, sous forme d'enregistrements appelés : "Audit Record". Chaque audit record représente un événement.

ASAX est un évaluateur qui permet d'analyser des fichiers d'un format bien précis: NADF (Normalized Audit Data Format). L'objectif de ce mémoire est double :

1. l'implémentation d'un format adaptateur assure la conversion d'un fichier généré par SunOS 4.1 en un format NADF;

2. l'application d'ASAX pour l'analyse des fichiers d'audit SunOS 4.1.

Le chapitre 1 présente les différents types d'attaques (sans être exhaustifs) et illustre l'audit de la sécurité qui est une contre-mesure, parmi d'autres, pour limiter ces menaces. Un modèle de détection d'intrusion, développé par Denning, est exposé pour mettre en oeuvre des concepts abstraits (Sujet, Objet, Action, ...) utilisé au cours de ce mémoire.

Le chapitre 2 traite des problèmes et approches portant sur l'analyse des audits trails. Dans ce chapitre figure aussi une brève description de l'évaluateur ASAX.

Le traitement des fichiers natifs (audit trails SunOS 4.1) demanderait une bonne spécification de leur contenu. Ainsi le chapitre 3 décrit la structure logique et physique des fichiers d'audit SunOS 4.1.

Pour être analysé, un fichier natif doit d'abord être converti en un format approprié à l'outil d'analyse : dans le cadre du mémoire, il s'agit du format NADF. La conversion est accomplie par l'adaptateur de format faisant l'objet du chapitre 4.

Enfin, le chapitre 5 nous fournit une brève description de la syntaxe et de la sémantique du langage RUSSEL. Quelques règles d'analyse, implémentées dans le langage RUSSEL, permettent entre autre la détection de certains scénarii au sein du réseau Sun.

CHAPITRE 1

LA SECURITE ET L'AUDIT

1. LA SECURITE ET L'AUDIT

1.1. INTRODUCTION

La rapidité d'évolution de l'informatique, tant du point de vue matériel que logiciel, impose une contrainte de sécurité non négligée des systèmes informatiques. En effet, ces derniers envahissent quasi tous les domaines (entreprises, secteur économique, vie privée, ...), leurs défaillances pourraient impliquer des dégâts d'ordre majeur (destruction de l'information, détournement de logiciels, ...).

Les systèmes actuels sont loin d'être totalement maîtrisés. En fait, ils sont de plus en plus complexes, automatisés, distribués et intégrés entre eux. C'est ainsi que leur sécurité pose des problèmes aigus et difficiles à résoudre : aucun système de sécurité ne garantit une protection totale. Malgré la suppression ou la diminution de certains risques, d'autres subsistent. Le caractère complexe des systèmes ne permet pas de prouver formellement leur sûreté absolue.

L'objectif de base de la sécurité informatique est de maintenir trois qualités fondamentales exposées ainsi :

1. **la confidentialité :** désigne la protection de l'information. Toute information doit rester secrète pour un utilisateur non autorisé à y accèder;

2. **l'intégrité :** désigne la conservation de l'information. Toute information doit rester intacte et ne subit une destruction ou une transformation que par l'intervention des personnes autorisées à le faire;

3. **la disponibilité :** désigne le bon fonctionnement d'un système informatique. Les informations et les ressources de tels systèmes doivent être accessibles, dans des délais convenables, par des personnes autorisées dès qu'elles le souhaitent. Les fonctions vitales du système doivent être protégées tout en garantissant un équililibre entre la disponibilité et le degré de protection du système. Le système UNIX, par vocation un

système ouvert, est un exemple d'une grande disponibilité. Néanmoins, il offre des outils de sécurité.

La sécurité informatique peut être classée en deux catégories : la sécurité physique (incendie, panne de matériel, conditionnement d'air, ...) et la sécurité logique (contrôle d'accès, protection de l'information, copies de sauvegarde, ...).

Ce mémoire ne tiendra pas compte de la sécurité physique, mais notre attention se concentrera plutôt sur la sécurité logique.

Ce chapitre traitera des différents types d'attaques (1.2.) et des contre-mesures pour les défier (1.4). Une des contre_mesures, objet de ce mémoire, est l'audit de la sécurité : elle sera traitée en détail en (1.7.).

1.2. LES DIFFÉRENTS TYPES D'ATTAQUES

Différents types d'attaques menacent les systèmes informatiques. Dans la littérature, des chercheurs, par exemple, [Anderson 80], [Lunt 88b], ont classifié ces formes d'attaques dans le but de définir des catégories de scénarios.

Trois classes majeures déterminant les différents types de menaces sont présentées ainsi :

1. **les intrusions externes :** Ce sont des pénétrations effectuées par des utilisateurs non autorisés à utiliser le système : par exemple, un utilisateur tente de pénétrer dans le système à l'aide d'un générateur de mots de passe.

2. **les intrusions internes :** Ce sont des pénétrations d'utilisateurs autorisés à utiliser le système. On peut distinguer parmi eux :

   2.1. ceux qui cherchent à outrepasser leurs privilèges afin d'atteindre les ressources ne leur étant pas accordées : par exemple, des personnes essaient de pirater des programmes, d'avoir l'oeil sur des informations confidentielles, etc.;

   2.2. ceux qui travaillent sous une fausse identité : ces derniers, masquent leur identité espérant ne pas être décelés lors d'un acte malveillant;

   2.3. ceux qui abusent de leurs pouvoirs et font de leurs droits d'accès un mauvais usage : l'administrateur du système, qui en a souvent le plein pouvoir, en est une bonne illustration. En particulier,

si ce dernier est un utilisateur habituel du système, il ne doit utiliser l'identité de **root** qu'en cas de nécessité absolue (maintenance, anomalie, ...).

3. **les virus informatiques :** les systèmes informatiques sont aussi cibles d'infections virales. Ces virus sont caractérisés par une séquence d'instructions destinées à détruire et altérer les données. Il existe plusieurs formes de virus dont les plus courants sont : les virus logiques, les chevaux de Troie, les bombes logiques, les vers et les bactéries. Voici ci-dessous, une description non exhaustive de ces virus :

1. **les virus logiques** : un virus logique est une partie de code conçu de façon à détruire un logiciel cible tout en s'auto-copiant. Ce mode de reproduction permet au virus d'infecter ainsi tous les types de support, disques, disquettes, mémoire, réseau, ...

Un virus bien conçu peut même aller jusqu'à supprimer les preuves de sa présence;

2. **les chevaux de Troie** : un cheval de Troie est un programme conçu pour le vol d'informations relatives à la sécurité d'un réseau, modification ou destruction de documents, etc. A la différence des virus informatiques, les chevaux de Troie ne se reproduisent pas.

Exemple classique d'un cheval de Troie : on l'appelle "*mule de Troie*", il est représenté par une procédure de login truquée, destinée à intercepter le mot de passe de l'utilisateur en le copiant dans un fichier ou en utilisant un programme de courrier électronique;

3. **les bombes logiques** : l'image d'une bombe logique est celle d'un cocon porteur d'un virus ou de toute autre bestiole désagréable. Elle est conçue pour accomplir des dégâts lorsqu'elle est déclenchée par la satisfaction d'une condition : date, présence d'une donnée particulière, etc.;

4. **les vers** : un ver (**worm**), tout comme un virus logique, est destiné à détériorer les données d'un système. Le ver se reproduit d'une façon exponentielle d'un ordinateur à un autre : il se propage en empruntant comme support de transmission les liaisons entre ordinateurs et entre utilisateurs (courrier électronique, ...);

L'objectif principal du ver est d'immobiliser le maximum de ressources de la machine qui demeure dès lors difficilement utilisable;

5. **les bactéries** : une bactérie est un programme qui n'a pas l'intention directe d'endommager le système dans lequel elle s'est infiltrée. Elle se multiplie d'elle-même de façon exponentielle jusqu'à surcharger le système et dégrader sa performance.

1.3. DES SYMPTÔMES D'INTRUSION

Pour les méthodes de détection a posteriori, il est importe de connaître les symptômes des différentes menaces afin d'empêcher la reproduction d'actions illégales. Chaque type d'attaque se caractérisé par un symptôme détectable lors d'une analyse adéquate. En voici quelques exemples :

1. pour les utilisateurs non autorisés à utiliser le système, un symptôme est, par exemple, une répétition de commandes login infructueuses. Cela permet de déceler la tentative de violation du droit d'accès au système;

2. pour les utilisateurs autorisés à utiliser le système, l'exécution répétitive de commandes échouant tels que : la copie d'un fichier, le changement d'accès à une ressource, ..., en est un symptôme. A propos des personnes agissant sur le système sous une fausse identité, la différence entre leur profil observé et leur profil standard détermine un symptôme, un profil étant la description d'un comportement supposé comme "normal";

3. concernant les virus, leur détection demanderait une description détaillée du comportement normal du systèmes. Un comportement anormal (diminution de la mémoire disponible, accroissement des fichiers exécutables, dégradation de la performance du système, ...) présente alors un symptôme.

1.4. LES CONTRE-MESURES

Plusieurs contre-mesures existent pour pallier à la violation de la sécurité des systèmes informatiques. En voici quelques-unes :

I. des contrôles d'accès : il s'agit de contrôler l'accès aux systèmes, aux machines ainsi qu'aux terminaux. Les objectifs principaux du contrôle d'accès sont :

1. empêcher l'intrus de pénétrer dans le système en mettant une "barrière". Autrement dit, le système doit authentifier l'utilisateur désireux de se connecter, et ceci grâce à une procédure **login/passwrd**.

Tout utilisateur reçoit de l'administrateur un **nom** (*username*), qui l'identifie parmi tous les autres uitilisateurs, un **mot de passe**, ainsi qu'un **répertoire personnel**. Ces renseignements sont groupés dans un **compte** créé par l'administrateur du système, dans un fichier spécial (par exemple, le fichier /etc/passwd pour le système UNIX);

2. contrôler l'accès aux données et aux informations sensibles : deux types de contrôles d'accès se pésentent :

a. contrôle d'accès discrétionnaire (DAC : Discretionnary Access Control) : dans ce cas, l'utilisateur est le seul responsable et possède la liberté totale du choix de protection de ces fichiers;

b. contrôle d'accès mandaté (MAC : Mandatory Access Control) : ici, les systèmes se chargent de la protection des fichiers. Ce mode de contrôle est adopté par des ordinateurs qui manipulent des données très sensibles;

II. la cryptographie : elle consiste à chiffrer les données au moyen d'un jeu de clés, de manière à les rendre illisibles à celui qui ne possède pas la clé. En général, le cryptage est effectué suivant un algorithme de codification; un algorithme inverse, permettra ensuite de reconstituer l'information originale. Parmi les algorithmes les plus répandus en cryptographie, on peut citer l'algorithme DES (Data Encryption Standard);

III. D'autres méthodes divers de sécurité sont utilisées. Par exemple : la sauvegarde des copies, le contrôle d'accès physique, des procédures antivirus, etc.;

IV. l'audit de la sécurité : cette technique consiste à enregistrer dans un ou plusieurs fichiers des informations sur l'activité des utilisateurs. (cfr 1.7.)

1.5. UN MODÈLE GÉNÉRIQUE DE DÉTECTION D'INTRUSION

Il s'agit d'un modèle développé par Denning [D.DENING 87] qui est devenu un standard dans la description des systèmes de sécurité. C'est un modèle conçu pour la protection contre la violation de la sécurité d'un système informatique. Il est indépendant de tout système particulier, de tout

environnement d'application ou type d'intrusion. C'est aussi un modèle de base pour la description de système expert de détection d'intrusion en temps réel. Un système expert appelé IDES (Intrusion Detection Expert System) a été développé sur base de ce modèle.

Le modèle est basé sur deux approches : une approche statistique et une approche basée sur des règles. La figure 1.1 représente une architecture du modèle IDES :

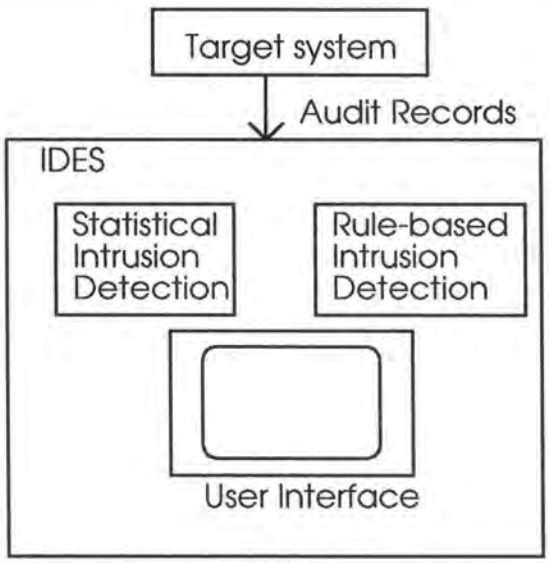


Figure 1.1 : Architecture du modèle IDES

Les composantes principales du "modèle" sont :

1. Sujets;
2. Objets;
3. Audit records;
4. Profiles;
5. Anomaly records;
6. Activity rules.

1.5.1. Sujets

Le sujet est une "*entité active*", c'est l'auteur de l'action sur le système cible. Il prend généralement la forme d'un utilisateur, toutefois il peut s'agir d'un programme, processus, ...

1.5.2. Objets

L'objet est une "*entité passive*", c'est le récepteur de l'action. Un objet est par exemple : fichier, répertoire, terminal, imprimante, ... Les objets sont souvent représentés d'une façon granulaire : par exemple, les records qui sont groupés dans des fichiers et les fichiers dans des répertoires.

1.5.3. Audit records

L'audit record représente l'action effectuée par le sujet sur l'objet. Il comprend six composantes :

<Sujet, Action, Objet, Exeption_condition, Resource_usage,
Time_stamp>.

En voici la description :

1. **Sujet** : voir (1.5.1.);

2. **Action** : C'est l'opération effectuée par le sujet sur l'objet. Elle se caractérise par un résultat déterminant son succès ou son échec. Une action est par exemple : login , logout, read, execute, etc;

3. **Objet** : voir (1.5.2.);

4. **Exception_condition** : c'est une valeur retournée par le système indiquant éventuellement la violation du mode d'accès à une ressource.

5. **Resource_usage** : C'est une liste quantitative fournissant le nombre utilisé de certaines ressources. Par exemple : le nombre de pages imprimées, le nombre de records lus ou écrits; le nombre d'entrée/sortie, le temps CPU consommé, etc;

6. **Time_stamp** : représente la date et l'heure pendant lesquelles l'action s'est déroulée.

La plupart des opérations sur le système impliquent plusieurs objets. Par exemple, la compilation d'un programme met en jeu le compilateur, le programme source, le programme objet et des fichiers inclus (par l'instruction "include").

Par souci de simplicité, le modèle décompose toute action en plusieurs actions élémentaires de façon à ce que chaque audit record fasse référence à un seul objet.

Exemple : **la copie d'un fichier**

La commande **copy** est décomposée en trois opérations :

1. **execute** : pour l'exécution du programme de copie;
2. **read** : pour la lecture du fichier source;
3. **write** : pour l'écriture dans le fichier de destination.

La figure 1.2 ci-dessous illustre les audit records générés en réponse à la commande :

> copy file_name <répertoire>file_name

effectuée par l'utilisateur TOTO.

(TOTO, execute, <répertoire>copy.exe, 0, CPU=00002, 110693225323)

(TOTO, read, <TOTO>file, 0, RECORDS=0, 110693225324)

(TOTO, write,<répertoire>file,write_viol,RECORDS=0, 110693225325)

Figure 1.2

La valeur "**write_viol**" du champ *Exception_condition*, de la figure ci-dessus, retournée par le système indique que l'utilisateur TOTO n'a pas le droit d'accès en écriture au fichier *file_name*, par conséquent l'opération d'écriture a échoué.

Notons que le système cible est responsable de la génération des audit records et de leur transmission au système expert de détection d'intrusion pour leur analyse.

1.5.4. Profiles

L'idée de profil est d'avoir une description du comportement considéré comme "normal" d'une manière à détecter tout écartement significatif de ce comportement comme "anomalie". Il est caractérisé par un ensemble de mesures (metric) et un modèle statistique.

- la mesure : elle est une variable aléatoire x représentative d'une mesure quantitative accumulée sur une période. Le modèle définit trois types de mesure :

1. event counter : x représente le nombre d'événements satisfaisants à une certaine condition pendant une période (nombre de login en une heure, nombre de password échouant, ...);

2. interval timer : x représente le temps écoulé entre deux événements (intervalle de temps entre des login successifs, ...);

3. resource measure : x représente la quantité de ressources consommées pendant une période (temps CPU, ...). La période est spécifiée dans le champ Resource_usage de l'audit record (voir 1.5.3.).

- modèle statistique : il se base sur la variable aléatoire x citée ci-dessus. Etant donné n observations x_1, ..., x_n, le but du modèle statistique est de déterminer si l'observation x_{n+1} est anormale par rapport aux observations précédentes. Plusieurs modèles statistiques peuvent être utilisés (Mean and Standard deviation model, Multivariate model, Marcov process model , exponentiel model, ...) [D.DENING 87] [MIDAS 88]. A titre d'exemple, voici l'illustration d'un de ces modèles :

- **Mean and Standard deviation model** : cette approche se base sur la connaissance des observations x_1, ..., x_n. Une nouvelle observation x_{n+1} est définie comme étant anormale si elle se trouve en dehors de l'intervalle :

[moyenne-d*stdev, moyenne+d*stdev]

Avec :

d : représente le seuil (**treshold**); son choix est très important;

somme = x_1+ ... +x_n;

carré_somme = $x_1^2 + ... + x_n^2$;

moyenne = somme / n;

stdev = $\sqrt{(\text{carré\_somme} / n-1) - \text{moyenne}^2}$.

Une grande déviation du profil standard de l'utilisateur fait de lui un suspect. Exemple : un utilisateur travaille brusquement à une heure inhabituelle (jour de fête, week-end, ...). Néanmoins, cette approche (profil) est vulnérable et peut être violée par des utilisateurs futés. En effet, sachant que le profil est mis à jour périodiquement, l'intrus changera légèrement de profil jusqu'au moment où, inaperçu, il peut passer à l'attaque.

1.5.5. Anomaly Records

Ce sont des records présentant une certaine anomalie ou un comportement anormal vis-à-vis du système ou des objets mis en jeu. Ces records sont détectés par le déclenchement de règles (voir section suivante). Un record anormal se constitue de trois composantes :

- **Event** : représente l'événement causé par une action anormale (login, read, ...);

- **Time_stamp :** voir 1.5.3;

- **profile** : voir 1.5.4;

1.5.6. Activity Rules

Ce sont des règles actives. Une règle active spécifie une action à effectuer quand certaines conditions sont remplies . Une condition porte sur les éléments d'un événement (sujet, objet, date, ...). Le rôle des règles est essentiellement de :

- détecter des audit records qui représentent une anomalie et de générer en conséquent un rapport pour l'officier de sécurité;

- vérifier le profil des utilisateurs et celui du système. S'il y a présence d'anomalies, un rapport est généré; dans le cas contraire, le profil est mis à jour.

1.6. LES CRITÈRES D'ÉVALUATION DE LA SÉCURITÉ DES SYSTÈMES INFORMATIQUES

La sécurité des systèmes informatiques est un sujet en pleine croissance. Une raison majeure de cet accroissement réside dans les actions malicieuses sur

le système et les mauvaises manipulations (volontaires ou involontaires) de l'information.

Les critères d'évaluation de la sécurité des systèmes informatiques définissent des moyens afin de contrer et de limiter ces menaces au bon fonctionnement des systèmes : ceux-ci ont parfois des conséquences considérables.

Il existe différentes normes de référence qui définissent des critères pour le classement en niveau de la sécurité d'un système informatique.

Par exemple :

- "Department of Defense (DOD) Trusted Computer System Evaluation Criteria" [DOD 85] aux Etats Unis;

- "IT_Security Criteria" [ITSC 89] en Allemagne;

- "Information Technology Security Evaluation Criteria" [ITSEC 91] pour la Communauté Européenne .

- "The Canadian Trusted Computer Product Evaluation Criteria" [CTCPEC 92] au Canada.

Tous les documents susmentionnés font référence à un document de base du nom d'"Orange Book"[1] [DOD 83].

Classement du DOD

Les niveaux de sécurité définis par le DOD sont découpés en quatre divisions. Chaque division contient une ou plusieurs classe(s) Les niveaux supérieurs doivent respecter les contraintes des niveaux inférieurs. Le découpage défini dans l'"Orange Book" se présente ainsi :

Division D : est le niveau de protection minimal. Il n'y a pas de notion de protection entre utilisateur dans les systèmes de classe D (exemple : les micro_ordinateurs).

Division C : représente la sécurité avec accès discrétionnaire (voir 1.4.). Cette division ce découpe en deux classes:

---

[1] Porte ce nom, tout simplement parce que sa couverture est orange. Il a été publié en 1983 et mis à jour en 1985 [DOD 85].

1. Classe C1 : la sécurité à ce niveau est assez réduite. Les systèmes classés C1 ne sont pas suffisamment protégés contre les intrusions extérieures. Les utilisateurs peuvent protéger leurs données. Ceci suppose une authentification (procédure login/passwd) et un outil de protection fondé sur la notion de propriété (permission d'accès par utilisateur, groupe ou autres utilisateurs). Par exemple : les premières versions des systèmes IBM/MVS et d'Unix;

2. Classe C2 : dans laquelle l'authentification est plus stricte. Un système de cette classe est muni aussi d'un mécanisme d'audit. Les utilisateurs sont responsables de leurs actions. De plus, les ressources du système sont soumises à un contrôle d'accès et la documentation doit être complète. (exemple : le système SunOS Release 4.1).

Remarquons à ce stade que le niveau C2 exige simplement un mécanisme de génération d'information audit sans rien mentionner à propos de l'analyse de cette information.

Division B : La sécurité est mandatée (voir 1.4.). Cette division est découpée en trois classes :

1. Classe B1 : dans laquelle les objets sont étiquettés, le système doit assurer le respect des étiquettes. Celles-ci ne sont pas modifiables par les utilisateurs. L'accès aux objets se base sur l'étiquette, celle-ci en détermine le droit d'accès (exemple : SunOS MLS Release 1.0);

2. Classe B2 : dans laquelle : 1) la sécurité doit être représentée par un modèle; et 2) il doit être possible de tester les outils de sécurité disponibles. Le système doit être associé à une notion de privilège minimum. Un des rares systèmes classés au niveau B2 est le système MULTICS de Honeywell Information System;

3. Classe B3 : est appelée "Domaines de sécurité". La structure interne du système doit être complétement documentée et prouvée informellement. Les listes de contrôle d'accès sont obligatoires. L'administration de la sécurité doit être une tâche séparée de l'administration du système lui-même. Le seul système, actuellement, évalué B3 est le système XTS-200 de Honeywell Federal System.

Division A : La seule classe de la division A est la classe A1. Une preuve *formelle* est demandée. Il faut un modèle pour la protection du système et une preuve pour sa consistance et son adéquation. Le seul système agréé au niveau

A1 est le système SCOMP (Secure Communication Processor) de Honeywell Information System.

Remarque : je n'ai pas eu plus d'information sur ce système. Peut-être s'agirait-il d'un petit protocole de communication et pas d'un O.S. complet...?

1.7. QU'EST-CE QUE L'AUDIT DE LA SÉCURITÉ ?

L'audit de la sécurité est une contre-mesure qui tente essentiellement de détecter toute activité pouvant nuire à la sécurité pour éventuellement empêcher sa reproduction. Cette technique permet de rassembler des informations tels que:

- qui a effectué telles opérations;

- quelles sont les opérations effèctuées sur le système;

- qui a effectué des opérations anormales.

Les systèmes équipés d'un mécanisme d'audit génèrent des fichiers appelés "Audit Trails" relatifs à la sécurité. Ces derniers forment une collection d'informations permettant de retracer certaines activités sélectionnées du système. Chaque fichier audit trail est composé d'enregistrements appelés "audit records". Chaque audit record représente un événement.

1.7.1. Les événements auditables

Tout événement lié à la sécurité du système est un événement auditable. D'après les recommandations du "National Computer Security Center" [NCSC 87], les événements à auditer dépendent du niveau de sécurité du sytème et sont présentés dans la figure ci-dessous :

Level Security	Auditable Events
C2	use of identification and authentication mechanism
	introduction of objects into user's adress space
	deletion of objects from a user's address space
	action taken by computer operators and system administrations and/or security administrators
	all security-relevant events
B1	any override of human readable output markings on paged, hard copy output devices
	change of designation of any communication channel or I/O device change of sensitivity level(s) associated with a single-level communication channel or I/O device
	change of range designation of any multi-level communication channel or I/O device
B2	add to B1 auditable events , events that may exercice covert storage channels
B3	add to B2 auditable events, events that may indicate an imminent violation of the system 's security policy
A1	no new requirements have been added at the A1 class

Figure 1.3 : Evénements auditables

Que représente un événement ?

Il représente une action effectuée par un sujet sur un objet avec un certain résultat (échec ou succès de l'action). Voir la figure 1.4 ci-dessous.

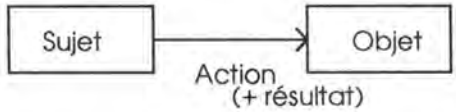


Figure 1.4 : Représentation d'un événement

1.7.2. Le contenu d'un événement

Les composantes d'un événement varient d'un système à un autre, en fonction du générateur d'audit de celui-ci . Néanmoins, suivant le modèle de Dening, un événement contient au moins les informations suivantes :

- la date et l'heure de l'action : appelées "Time-stamp";

- le type de l'événement : les événements sont généralement typés en classes;

- un identifiant du sujet auteur de l'événement;

- un identifiant de l'objet (répertoire, fichier,...);

- le résultat de l'action (échec ou succès) : parfois divers codes d'échec en fonction de la cause de l'échec ;

- L'origine de l'action (type du terminal, ...).

1.7.3. La sélection des événements

Les systèmes munis d'un mécanisme d'audit peuvent générer une quantité énorme d'informations. Ceci pose des problèmes d'espace disque et rend la tâche difficile à l'officier de sécurité. Cette tâche consiste à parcourir la masse de données afin de détecter la présence éventuelle d'anomalies. Pour remédier à cet inconvénient, le système générateur d'audit trails offre des possibilités de faire des présélections, via des paramètres, pour ne produire que certains événements en fonction des désirs de l'officier de sécurité. Toutefois, une présélection très stricte a le désavantage d'ignorer certains événements intéressants pour la sécurité. Les fichiers résultants de cette présélection portent le nom de "**collection files**". La présélection peut porter par exemple sur :

- le type d'événement;

- l'identité de l'utilisateur;

- l'identité de l'objet;

- ....

Une post-sélection appliquée aux fichiers d'audits existants produira des fichiers appelés "**reduction files**". Lors de la post-sélection, les audits records sont filtrés au moyen, par exemple, d'opérateurs logiques AND, OR, NOT portant sur les conditions de séléction.

La figure suivante illustre la présélection et la post-sélection des événements.

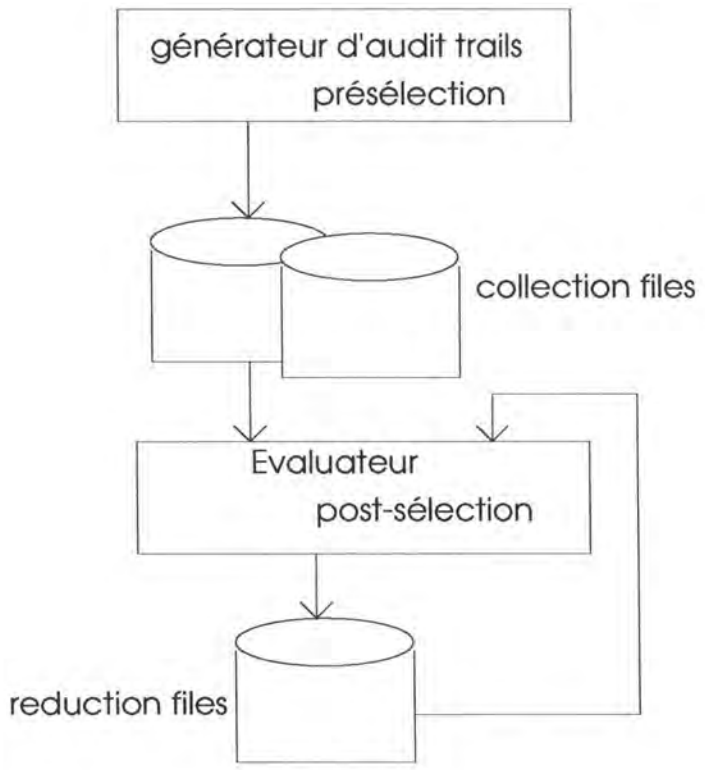


Figure 1.5 : Pré/Post-sélection d'événements

1.7.4. Protection des audit trails

Des mesures très strictes vis-à-vis des fichiers d'audit doivent être prises en considération . Le droit d'accès à ces fichiers ne peut être qu'en lecture seulement et par des personnes autorisées, cela afin de préserver l'intégrité des informations "audit data" qu'ils abritent. A ce propos, le "National Computer Security Center" [NCSC 87/2] déclare :

" At the minimum, the data on the audit trail should be considered to be sensitive, and the audit trail itself shall be considered to be as sensitive as the most sensitive data contained in the system".

L'accès en écriture aux audit trails n'est permis que par le mécanisme d'audit. Ce dernier est responsable de l'enregistrement de toute activité liée à la sécurité.

1.8. CONCLUSION

Parmi les contre-mesures relatives à toute atteinte à la sécurité, l'audit est un moyen efficace pour mettre en garde les indésirables (ou les intrus).

Le but principal de l'audit est de garder la trace de toute interaction avec le système, permettant ainsi de découvrir toute action malicieuse et alors d'empêcher sa reproduction. Les activités des utilisateurs sur le système sont enregistrées sous forme de records par le mécanisme d'audit. Ces derniers doivent faire l'objet d'une analyse judicieuse et adéquate. Le chapitre suivant présente un outil d'analyse nouveau dans le domaine de la sécurité logique, il porte le nom d'"ASAX".

CHAPITRE 2

ASAX : UN OUTIL DE SECURITE

2. ASAX : UN OUTIL DE SECURITE

2.1. INTRODUCTION

De nos jours, les systèmes informatiques sont vulnérables aux différents types d'attaques (intrusions, virus, ...) . Il est impossible d'obtenir une sécurité absolue, tout simplement parce qu'on ne sait pas prévoir toutes les menaces pouvant se présenter. L'idéal est de concevoir et réaliser un système démontré fiable contre toutes les attaques . Un tel système satisferait au niveau de sécurité "A"[2] . Ce qui est très difficilement réalisable, du moins actuellement. Comme l'explique le professeur Baudouin Le Charlier : " la réalisation d'un système "parfait" demanderait une formalisation de tous ces mécanismes d'accès, permettant de prouver mathématiquement l'impossibilité d'une utilisation non permise, malveillante ou accidentelle. Ce type de formalisation est loin d'être possible, dans l'état de l'art actuel, étant donné la complexité des systèmes d'exploitation et des protocoles de communication, par exemple" [ESORICS 92].

Une façon de couvrir cet handicap consiste à détecter les attaques a posteriori. Pour ceci, les systèmes munis d'un haut niveau de sécurité comportent des mécanismes de stockage d'informations relatives à la sécurité, dans les fichiers audit trails.

L'analyse de ces fichiers permet de déceler des scénarii d'utilisations incorrectes ou malicieuses, des tentatives d'intrusions, des virus..., après leurs productions ou même en cours de productions.

ASAX[3] (Advanced Security Audit trail on uniX) est un outil de sécurité dont la tâche est d'évaluer les fichiers audit trails. C'est un analyseur entièrement nouveau

---

[2] Niveau "A" dans l'"us Evaluation Criteria" [TCSEC]; "G7" dans le "German National Criteria [GISA-EC]; "E6" dans le "standardised European Information Technology Security Evaluation Criteria" [ITS-EC].

[3] Projet de sécurité, résultat d'une collaboration entre Siemens Nixdorf software Namur et l'institut d'informatique de Namur.

dans le domaine de la sécurité logique, permettant à l'utilisateur de décrire des requêtes relatives à n'importe quel scénario. ASAX analyse ces requêtes et les exécute sur le fichier audit trail demandé en une seule passe et dans un temps très court.

Nous traitons dans ce chapitre l'analyse des audit trails et les qualités requises par ASAX.

2.1. L'ANALYSE DES AUDIT TRAILS

Un fichier audit trail est un fichier séquentiel généralement très grand[4] et comprenant des audit records. Ces derniers représentent l'histoire ou le passé du système. Ils sont ordonnés sur une base chronologique : chaque record contient un champ indiquant la date (Time_stamp) à laquelle il a été généré. L'objectif de leur stockage dans un ou plusieurs fichier(s) audit trail(s) est de pouvoir les analyser dans un deuxième temps. Cette analyse peut avoir plusieurs buts :

1. déceler les actions spécifiques des utilisateurs, par exemple :

- une tentative de login, réussie ou non;
- une demande d'ouverture de fichier;
- une requête de lecture ou d'écriture dans un fichier;
- un accès à une ressource éloignée via un réseau;
- ...

et l'utilisation hostile ou douteuse du système. Une utilisation hostile peut se caractériser par un type de succession d'actions spécifiques, c-à-d une séquence non nécessairement contiguë d'actions spécifiques. Parmi ces utilisations, citons, par exemple, la tentative d'accès en lecture à un fichier "filename" par un utilisateur non autorisé. Cette tentative ce caractérise par la séquence d'actions suivantes :

- examen des droits d'accès du fichier, en utilisant, par exemple, la commande UNIX : **ls -l.** Un fichier est accessible en *lecture* (r), en *écriture* (w) ou en *exécution* (x);
- changement de ces droits d'accès par la commande : **chmod +r filename;**

[4] Peut atteindre 4 MB pour un système de charge moyenne sur une journée de travail

- lecture effective du fichier.

2. faire des études statistiques sur , par exemple :

- le nombre de commandes login effectuées par un utilisateur pendant une certaine période (heure, journée, ...), ou par un groupe d'utilisateurs;

- le nombre de commandes exécutées par un utilisateur (copy, open, execute, ...);

- le nombre d'accès illégaux à des objets (rlogin, print, read, ...)

- ...

L'analyse d'un fichier audit (détection d'anomalies d'utilisation ou étude statistique) peut parfois nécessiter une pré-sélection ou un filtrage préalable de certains records. Ceci permet de réduire la taille du fichier et de mieux centrer l'analyse.

Par rapport au moment où l'analyse est effectuée, on peut distinguer deux types d'analyses :

1. une analyse effectuée *après* que la génération du fichier d'audit soit terminée . On l'appelle *analyse off-line*. Cette technique permet de détecter et de dissuader les actions non autorisées pour empêcher leur reproduction;

2. une l'analyse ayant eu lieu au *moment même* de la création du fichier d'audit, c-à-d lorsque les audits records sont analysés au fur et à mesure de leur production par le mécanisme d'audit, on parlera alors d'*analyse on-line*. Elle permet d'empêcher les actions non autorisées. En effet, l'outil d'analyse réagira instantanément contre l'attaque en cours en envoyant, par exemple, un message à l'officier de sécurité. Ce dernier prendra alors les mesures appropriées (message, alarme, ...) contre l'intrus afin de l'empêcher de continuer l'action. Ce type d'analyse exige une grande efficacité de l'outil d'analyse, d'une part, pour ne pas être "noyé" par le flux des records générés par le système d'exploitation et, d'autre part, pour ne pas dégrader la performance de celui-ci.

2.1.1.Problèmes et approches

Accomplir une analyse adéquate requiert un outil, à la fois intelligent et efficace, d'analyse automatique conçu à ce propos. La conception d'un tel outil poserait quelques problèmes particuliers liés à :

1. la taille énorme que peuvent atteindre les audit trails;
2. la variété de brèches dans la sécurité;
3. la réutilisabilité (reusability) de l'outil d'analyse;
4. l'interface utilisateur.

2.2.1.1. La taille des audit trails

Le problème lié à la taille des audit trails découle de la sémantique même de tels fichiers : ils sont sensés contenir tous les événements ayant eu lieu dans le système et pas seulement les événements pertinents pour la sécurité. Cependant, les fichiers audits peuvent être utilisés à des fins qui ne sont pas nécessairement liées à la sécurité : ainsi ils peuvent être utilisés dans le cadre d'une activité de monitoring des utilisateurs (vitesse de frappe d'une secrétaire, taux de réussite de commandes d'une nouvelle recrue, ...) et de certaines applications (performance, ...). Cette contrainte (qu'est la taille) a un impact sur l'espace disque pour le stockage des audit trails et le temps CPU requis pour leur enregistrement. D'un autre côté, l'analyse manuelle de ces audit trails peut s'avérer difficile pour l'officier de sécurité. Afin de faire face à cet inconvénient, du moins pour le limiter, deux approches possibles sont à retenir : la présélection et la compression des données.

1. **La présélection :** elle a été définie dans le chapitre 1, section 1.7.3.
2. **La compression des données :** une autre solution pour limiter la taille des données est de les comprimer. En effet, plusieurs mécanismes d'audit enregistrent les audit datas sous une forme codée afin de réduire l'occupation de l'espace disque.

Cependant, malgré la réduction de la taille des fichiers audit trails (cfr une des approches précitées), celle-ci reste suffisamment importante : on peut jamais dire a priori quelles sont les informations qui vont être utilisées pour l'analyse. Ceci oblige donc à envisager une analyse en une seule passe.

2.2.1.2. La variété de brèches dans la sécurité

Les brèches dans la sécurité des systèmes informatiques prennent diverses formes (intrusions, virus, ...). La classification des différentes formes de menaces par des chercheurs éminents (voir chapitre 1, section 1.2.) a pour but de définir des catégories de scénarios détectables par l'évaluation des audit trails.

Néanmoins, dans chaque classe, beaucoup de scénarii d'attaques sont possibles dont certains restent toujours imprévus. La difficulté provient donc du fait que l'on doit connaître a priori un scénario d'attaque et interroger le fichier d'audit. Cela suppose une expertise non seulement des scénarii d'attaques mais aussi de la façon dont ils sont représentés dans le fichier audit.

2.2.1.3. La réutilisabilté : généricité et universalité

La réutilisabilité d'un logiciel est une qualité très recommandée dans le monde du génie logiciel : un logiciel peut être réutilisé dans différents environnements, matériels et logiciels, avec un effort d'adaptation minimum. Dans le cas d'outil d'analyse d'audit trail cette qualité est très importante vue la diversité de O.S. et de versions multiples de ces O.S.

Deux approches sont à envisager pour atteindre la réutilisabilité :

1. la conception d'un outil d'analyse paramétrisé et dans lequel les formats des différents audit trails peuvent être instanciés. Il s'agit d'un *outil générique*;

2. ou, un outil plus général permettant d'analyser tout type d'audit trail après conversion préalable en un format approprié à l'outil d'analyse : on parlera d'un *outil universel.*

2.2.1.4. L'interface utilisateur

L'analyse des audit trails présente un autre problème lié au confort d'utilisation de l'évaluateur : en effet, l'ergonomie de l'interface utilisateur[5] est primordiale. L'outil d'analyse doit être convivial dans le sens où il permet à l'officier de sécurité d'effectuer ses analyses sans aucune difficulté.

2.3. LES QUALITÉS D'ASAX

A l'origine, ASAX est un outil de sécurité. Cependant, il permet généralement d'analyser tout fichier séquentiel. Ce produit a été conçu pour réaliser les objectifs suivants :

- analyser en une seule passe de longs fichiers séquentiels et principalement les fichiers audit trails liés à la sécurité des systèmes, mais aussi le

[5] User interface, onl'appelle également interface Homme-machine

traitement d'autres applications tels que : la comptabilité, les systèmes bancaires, l'analyse de flux de données via un réseau, ...

- accepter toutes requêtes d'analyse possible y compris les requêtes qui portent sur la relation entre plusieurs records.

Les caractéristiques principales de l'évaluateur ASAX sont : *l'universalité,* la *puissance,* l'*efficacité* et la *portabilité.* Elles sont développées dans les sections suivantes.

2.3.1. L'universalité

Les systèmes d'exploitation munis d'un mécanisme d'audit génèrent des fichiers audit trails de formats différents (quelques exemples sont exposés dans la section 2.2.1.1). Les outils d'aide à l'analyse de ces audit trails sont rares et ne sont conçus que pour un usage interne. On peut citer entre autre l'évaluateur SATUT (Security Audit Trail Utility) pour les audit trails générés par BS2000 de Siemens, MIDAS (Multics Intrusion Detection and Alerting System) pour Multics system.

ASAX se veut un évaluateur universel (voir section 2.1.1.3.). Cette universalité est réalisée par le choix et la définition d'une structure de fichier souple et standardisée, appelé NADF. L'analyse s'effectue uniquement sur des fichiers aux formats standardisées. Les fichiers d'origine (appelé fichier Natif) doivent donc être traduits en fichie de format NADF. Un fichier NADF est une séquence de records ayant le format NADF. (Pour plus de détails voir le chapitre 4, section 4.3).

Vu la flexibilité du format NADF, la conversion de tout type d'audit trail en un tel format se ferait relativement aisément avec la réalisation d'un adaptateur de format. Un des objectifs de ce mémoire est d'implémenter un adaptateur de format pour les audit trails générés par le système SunOS. Le chapitre IV traitera de la mise en oeuvre d'un tel adaptateur de format.

Une architecture simplifiée d'ASAX est présentée dans la figure 2.1.

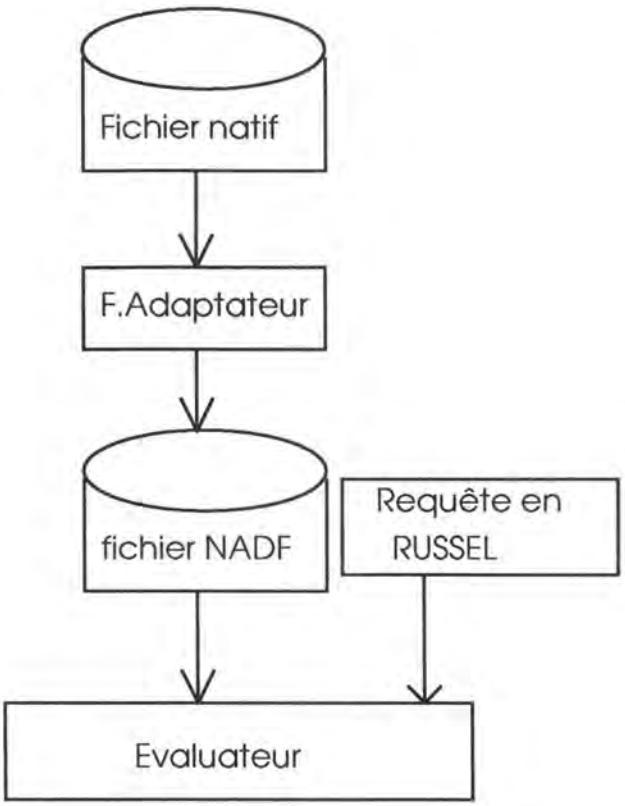


Figure 2.1 : Une simple architecture d'ASAX

2.3.2. La puissance

La puissance d'ASAX repose sur le langage de règles appelé RUSSEL (RUle-baSed Sequential Evaluation Language) qui permet à l'utilisateur d'exprimer ses requêtes d'analyse. Il s'inspire du principe des langages d'intelligence artificielle basé sur des règles avec chaînage avant (à la OPS5[1]). Plusieurs systèmes experts de sécurité utilisent l'approche "langage de règles" (rule-based language), comme par exemple :

- le système IDES [Lunt 90], système basé sur le modèle IDES développé dans le chapitre 1, utilise PBEST : un outil expert général à base de règles;

[1]Système à base de règles, un des premiers systèmes essentiels (ou générateur pour systèmes experts), diffusé commercialement sur de nombreux matériels (y compris des micro-ordinateurs) compatibles IBM ou Macintoch). Il dispose d'un compilateur de règles efficace.

- le système expert de détection de virus de l'université de Hamburg [Brunstain 91] utilise des règles à la OPS5;

- le système OSIRIS [Baur 88] utilise des règles à la Prolog.

RUSSEL est un langage de règles conçu spécifiquement pour l'analyse des audit trails, et d'une façon plus générale, pour celle de n'importe quel fichier séquentiel. La programmation d'une analyse, en RUSSEL, se fait au moyen d'un ensemble de règles. Une règle est considérée comme un procédure, au sens classique, avec des paramètres et des variables locales. Les paramètres permettent le passage d'informations d'un record à un autre. Trois types de règles sont à distinguer :

1. **des règles d'initialisation :** elles sont activées au début du traitement, pour le premier record. Leur but est d'initialiser le processeur d'analyse en activant des règles pour le premier record;

2. **des règles de sélection :** ce sont des règles actives pendant l'évaluation d'une record donné. A tout moment, cet ensemble de règles peut être modifié;

3. **des règles de terminaison :** elles sont évaluées après la fin du traitement de l'entièreté du fichier audit trail. Leur déclenchement peut activer l'impression d'un rapport sur les résultats de l'analyse, de la fermeture des fichiers, ...

Une règle sert à détecter un événement dans un scénario, elle renferme un ensemble de conditions et une séquence d'actions. Les actions se présentent sous deux formes :

1. des actions spéciales basées sur un mécanisme de déclenchement de règles (**trigger off**). Le déclenchement d'une nouvelle règle consiste à donner son nom, les valeurs des paramètres et un paramètre spécial indiquant si la règle sera activée pour le record courant (**trigger off for current**), le record suivant (**trigger of for next**) ou la fin de l'analyse (**trigger off at completion**).

   Après avoir évalué un record, la règle est automatiquement désactivée, à moins d'être réactivée explicitement par elle-même ou à l'aide d'une autre règle;

2. des actions classiques sous forme d'appels de procédure pré-définie permettant d'implémenter des fonctions de haut niveau (envoyer un message, déclencher une alarme, générer des fichiers temporaires, ...).

2.3.3. L'efficacité

L'efficacité d'ASAX repose sur une implémentation [ASAX 2] sophistiquée. Cette dernière est d'une nécessité majeure vu la fameuse taille des audit trails à analyser. Trois points essentiels déterminent l'efficacité de l'évaluateur :

1. L'analyse du fichier d'audit se fait en un seul parcours séquentiel. A un certain moment, toute information sur le passé de l'analyse (c-à-d les records déjà analysés) est encapsulée dans l'ensemble des règles actives. Cette information est représentée par la valeur des paramètres effectifs de ces règles;

2. Le traitement d'un record courant est précédé d'un prétraitement (pour une raison d'optimisation du temps d'accès) permettant un accès direct (via une table d'indirections) à ses champs lors de l'évaluation de la condition de sélection. Après avoir évalué le record courant, un ensemble de règles actives est appliqué au record suivant;

3. Evaluation efficace des conditions : les conditions qui figurent dans les règles sont évaluées de façon très efficace. Ainsi, l'évaluateur utilise une technique permettant de tronquer l'évaluation d'une condition dès que la valeur de vérité de celle-ci est connue, sans devoir évaluer l'entièreté de la condition. Une telle efficacité d'évaluation des conditions est indispensable: cela correspond à une grande partie du temps CPU nécessaire à l'analyse.

2.3.4. La portabilité

L'évaluateur est implémenté dans un langage de haut niveau (le langage C), ce qui le rend facile à porter sur d'autres machines. En effet, la portabilité de cet analyseur est effective sur les systèmes d'exploitation Sinix, Unix et Dos. Actuellement, ASAX tourne avec succès sur MX2, MX300, Sun Sparc (dans le cadre du mémoire, la portabilité d'ASAX sur les Sun s'est accomplie sans difficulté majeure) et aussi bien sur PC DOS.

2.4. QUELQUES EXEMPLES DE RÈGLES

Des exemples, non sophistiqués, sont développés ci-dessous pour illustrer l'utilisation du langage RUSSEL.

Notons que les mots-clés du langage sont écrits en gras, tandis que ceux en italique représentent des appels de procédures.

Exemple 1 : **Détection d'une tentative d'intrusion externe.**

Le scénario à détecter consiste en une succession de "*maxtimes*" fois de commandes login qui échouent durant une période de "*duration*" secondes. Deux règles sont utilisées et sont présentées aux figures 2.2 et 2.3. Une première règle **Failed_login** détecte un premier login qui échoue et déclenche une deuxième règle **Count_rule1**. Cette dernière compte le nombre de login ayant échoués, dans la période de temps déterminée.

```
rule Failed_login (maxtimes, duration : integer);
    begin
      if event = 'login' and result = 'failure'
                --> trigger off for next Count_rule1(maxtimes-1,
                                        timestamp + duration, usr_id);
    fi
    trigger off for next Failed_login
    end
```

Figure 2.2 : La règle Failed_login

```
if event = 'login' and result = 'failure' and timestamp<expiration
                                        and usr_id = usr_par
    --> if countdown>1
        --> trigger off for next Count_rule (countdown-1,
                                    expiration, usr_par);
        countdown = 1
        -----> SendMessage('too much failed login's for', usr_id)
        fi;
timestamp>= expiration -----> skip;
true ----> trigger off for next Count_rule ( countdown, expiration,
                                    usr_id)
fi
```

Figure 2.3 : La règle Count_rule1

Exemple 2 : **Détection d'intrusion par un utilisateur abusant de son privilège**

Le symptôme à détecter consiste en une succession de "*maxtimes*" fois un événement douteux qui échoue durant une période de "*duration*" secondes. Comme par exemple :

- chown : pour le changement de propriété d'un fichier UNIX;
- chmod : pour le changement du mode d'accès à un objet (fichier ou directorie);
- cd : pour changer de directory;
- ...

Deux règles sont utilisées et sont représentées aux figures 2.4 et 2.5.

La première règle **Abusive_user** détecte une première commande qui échoue et déclenche la règle **Count_rule2**. Cette dernière compte le nombre de commandes qui échouent pour l'utilisateur suspect. Elle est active jusqu'à expiration du temps spécifié.

```
rule Abusive_user (maxtimes, duration : integer);
    begin
      if ((event = 'chown' and result = 'failure') or (event = 'chmod' and
          result = 'failure') or (event = 'cd' and  result = failure')
    or (event = 'open' and result = failure) or ...)
          ----> trigger off for next Count_rule2 ( maxtimes-1,
                                                timestamp + duration, user_id)
    fi;
    trigger off for next Abusive_user ( maxtimes, duration)
    end
```

Figure 2.4 : La règle Abusive_user

```
rule Count_rule2 (countdown, expiration, suspect_id : integer);
    if ((event = 'chown' and result = 'failure') or (event = 'chmod' and
    result = 'failure') or (event = 'cd' and  result = failure') or (event = 'open' and
    result = failure) or ...)
        and   user_id = suspect_id
        and   timestamp < expiration
        ----> if countdown > 1
                ----> trigger off for next Count_rule2 (count_down-1,
                                                expiration, suspect_id);
                countdown = 1
                ----> SendMessage ('too much access break attempts
                                                for : ', suspect_id)
              fi
      timestamp >= expiration ----> skip;
    true ----> trigger off for next Count_rule (count_down, expiration,
                                                suspect_id)
    fi
```

Figure 2.5 : La règle Count_rule2

CHAPITRE 3

LES FICHIERS DE SECURITE "AUDIT TRAILS" D'UNE PLATE FORME SUN

3. LES FICHIERS DE SECURITE "AUDIT TRAILS" D'UNE PLATE FORME SUN

3.1. INTRODUCTION

Plusieurs systèmes d'exploitation (BS2000, Multics, ...) sont dotés d'un mécanisme d'audit, moyen efficace pour retracer toute interaction (normale ou anormale) avec le système. Le système faisant l'objet de ce mémoire est le système SunOS 4.1 pour les stations de travail SUNmycrosystem. Ce dernier offre une option de sécurité satisfaisante au niveau C2 défini par le DOD (voir chapitre 1).

L'objectif de ce chapitre, d'aspect plutôt technique, est de présenter la structure logique et physique des fichiers de sécurité "audit trail" générés par le système SunOS 4.1.

3.2. STRUCTURE D'UN FICHIER AUDIT TRAIL SUNOS 4.1

Un fichier audit trail généré par le système SunOS 4.1 se présente sous la forme d'une séquence de records :

I. un record *"file header"* : représente l'en-tête du fichier l'audit. Il est composé de trois champs :

1. ah_magic : représente un nombre dit "magic number", sa valeur dans le fichier */etc/magic* permet de reconnaître le format d'un fichier (fichier text, fichier postscript, ...).

Exemple : les audit trails ont un magic number égal à 58761;

2. ah_time : détermine la date et l'heure du déclenchement du mécanisme d'audit pour un fichier audit;

3. ah_namelen : représente la longueur du fichier audit précédent.

La structure physique du record "*file header*" défini en langage C est la suivante:

```
struct audit_header {
        int         ah_magic;
        time_t      ah_time;
        short       ah_namelen
};
```

Avec :

int est représenté sur quatre bytes;

time_t est un *entier long* représenté sur quatre bytes;

short est représenté sur deux bytes.

Le record "*file header*" est suivi par le nom du fichier audit précédent. En cas d'initialisation (boot) par le processus "audit daemon", ce nom vaut "NULL".

II. une séquence de records appelés "audit records" : chaque record représente un événement élémentaire, il a une longueur variable suivant le type d'événement. Un audit record est une séquence de champs appelées "audit data".

Un audit data constitue une chaîne de caractères terminée par le caractère "NULL" (\0).

III. un record "*file trailer*" : représente le "trailer" du fichier d'audit. Il est composé de quatre champs :

1. at_record_size : représente la taille du record "file trailer";

2. at_record_type : détermine le type du record "file trailer" (TRAILER). Les différents types possibles sont exposés en *annexe A*.

3. at_time : détermine la date et l'heure pendant lesquelles le mécanisme d'audit s'est arrêté pour un audit trail. Pour une date ultérieure, le mécanisme peut éventuellement redémarrer pour un nouveau enregistrement, mais sur un autre audit trail.

La structure physique du record "file trailer" défini en langage C est la suivante :

```
struct audit_trailer {
        short       at_record_size;
        short       at_record_type;
        time_t      at_time;
        short       at_namelen
};
```

La figure suivante illustre le format d'un tel fichier audit trail.

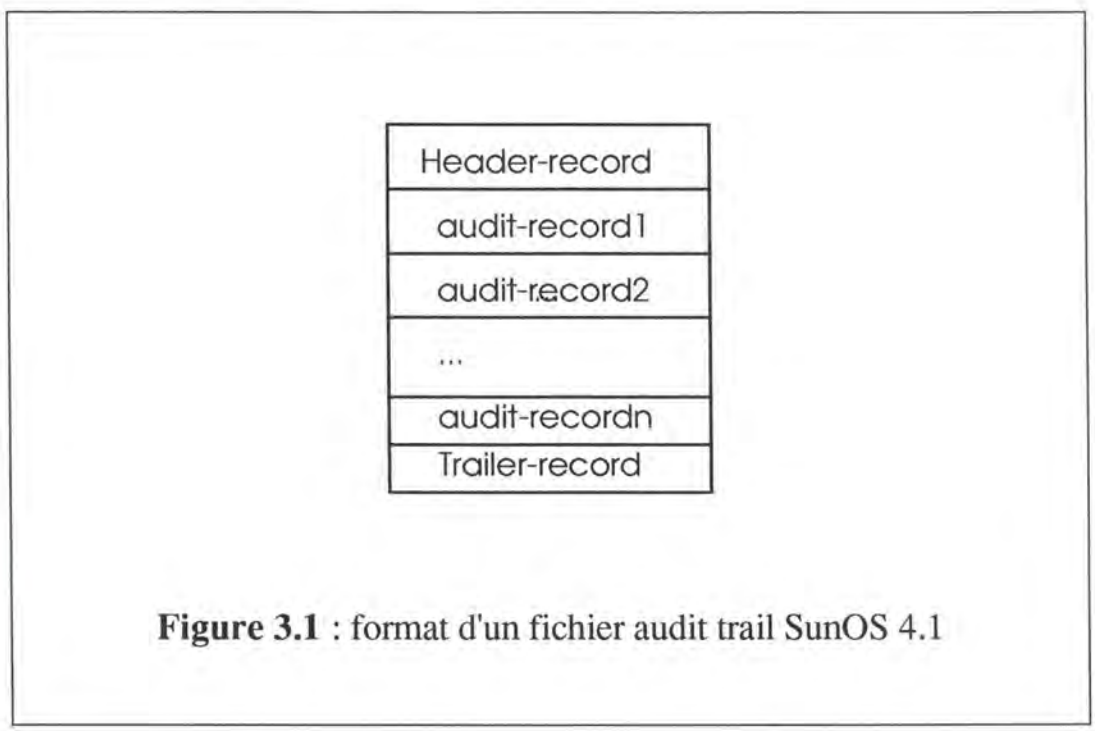


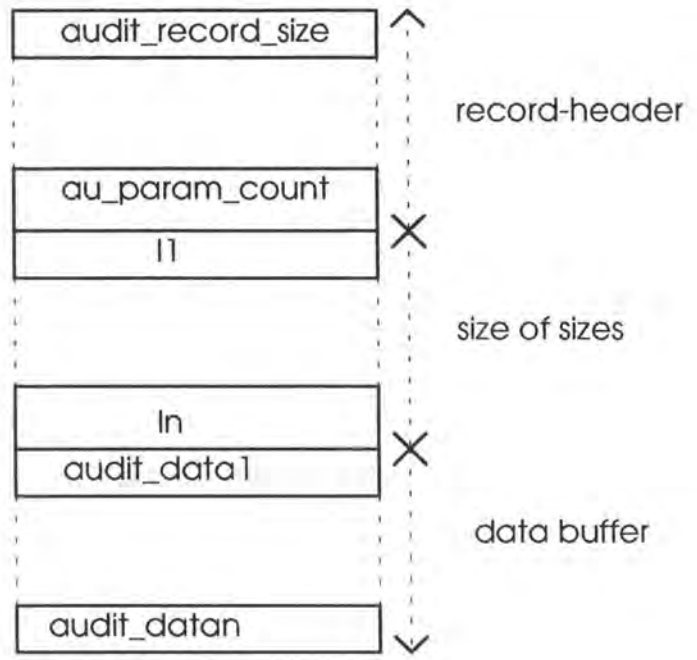
Figure 3.1 : format d'un fichier audit trail SunOS 4.1

3.3. STRUCTURE DES AUDIT RECORDS SUNOS 4.1

Les audit records constituant le corps d'un fichier audit sont appelés : audit records. Un audit record représente l'ensemble des informations concernant un événement ayant lieu dans le système.

Un audit record SunOS 4.1 est constitué de trois parties : un *header record*, une zone appelée "*size of sizes*" et une séquence de données encapsulé dans une zone

appelée "*data buffer*" (voir la figure suivante). Ces trois parties sont décrites par les trois paragraphes suivants :



Avec :

1<= i <= n : audit_datai audit data de longueur li

Figure 3.2 : structure d'un audit record SunOS 4.1

3.3.1. Le header du record

Le header du record représente une partie fixe commune à tous les audit records et ce quelque soit le type de l'événement, il contient les informations suivantes :

1. au_record_size : représente la taille de l'audit record;

2. au_record_type : représente le type de l'audit record. On distingue deux catégories d'audit records :

2.1. des audit records représentant des appels systèmes dont par exemple :

- **access** : access(filename, mode) vérifie si le processus appelant a les permissions de lecture, d'écriture ou d'exécution sur le fichier *filename*, selon la valeur du mode d'accès. Cette dernière vaut 4 (lecture), 2 (écriture), 1 (exécution) ou une combinaison de ces différentes valeurs;

- **open** : open(filename, mode) ouvre le fichier spécifié selon la valeur de mode. Cette valeur est 0 (lecture), 1(écriture) ou 2 (lecture/écriture). Open retourne un descripteur de fichier *fd* pour une utilisation dans d'autres appels systèmes. fd vaut -1 en cas d'erreur.

- **link** : link(filename1, filename2) donne un autre nom *filename2* au fichier *filename1*. Le fichier devient alors accessible par l'un ou l'autre des noms;

- **mount** : mount(filename, dir, mode) monte le système de fichiers spécifié par *filename* sur le répertoire *dir*. *Mode* spécifie si le système de fichiers est monté en lecture seulement ou non. Mount renvoie -1 en cas d'erreur, 0 sinon;

- ...

2.2. des audit records composés de textes arbitraires. Ces derniers sont générés par des programmes ne faisant pas appel au noyau du système. Néanmoins, quelques événements contenus dans ces textes sont fort liés à la sécurité. Par exemple :

- **login** : la commande *login* valide le nom d'usager et le mot de passe. En cas de réussite de la procédure de connexion, l'utilisateur est invité à lancer ses commandes. Sinon, un message erreur s'imprime.

Exemple :

$ login : agu

passwrd : ********

Login incorrect

$ login :

passwrd :

Bienvenue sur les stations SUN

$

- **su** : la commande *su* permet à un utilisateur d'en devenir un autre sans être obligé de se déconnecter, puis de se reconnecter.

Exemple : soit l'utilisateur "toto" qui veut accéder temporairement aux fichiers de l'utilisateur "titi". Toto lance la commande : **su titi**, et est invité à entrer le mot de passe de titi. En cas de succès, toto prend l'identité de titi comme s'il s'était connecté au système sous le nom de titi.

L'utilitaire su permet aussi de prendre l'identité **root** (statut du super-user), il suffit pour cela d'omettre le nom d'utilisateur dans la ligne de commande;

- ...

3. au_event : représente le type d'événement. Un événement appartient à une ou plusieurs classes. Une classe corespond à un ensemble d'occurences d'événements à auditer. Les différentes classes sont listées ci-dessous :

- data_read (dr) : lecture de données, ouverture pour une lecture, etc.;

- data_write (dw) : écriture ou modification de données;

- data_create (dc) : création ou suppression d'un objet;

- data_access_change (da) : changer le mode d'accès à un objet (modes, propriétaire);

- login_logout (lo) : login, logout, création par *at* (at : est un utilitaire qui exécute à un moment spécifique la séquence d'instructions fournie à son entrée standard);

- administrative (ad) : opération administrative normale;

- minor_privilege (p0) : opération privilégiée;

- major_privilege (p1) : opération privilégiée inhabituelle.

4. au_time : détermine la date et l'heure pendant lesquelles un événement s'est produit;

5. au_uid (**u**ser **id**entifier) : chaque utilisateur du système reçoit un identifiant numérique appelé *uid*. C'est cette valeur qui permet au système d'identifier un utilisateur et non pas son nom de connexion. En particulier, tout utilisateur ayant une valeur uid égale à zéro est super-utilisateur (root);

6. au_auid (**au**dit **u**ser **id**entifier) : ce champ est utilisé dans un but d'audit. Il permet de garder la trace de l'utilisateur ayant effectué le premier *login*. Cela aide l'officier de sécurité de savoir "qui se déguise en qui ?"

Lors de l'analyse des audit trails, c'est l'auid qui est pris en compte;

7. au_euid (**e**ffective **u**ser **id**entifier) : un utilisateur Unix identifié par uid, peut durant l'exécution d'un programme particulier prendre les privilèges du propriétaire de ce programme (uniquement durant cette exécution, et seulement si le mode de ce programme l'indique). Ainsi, son identifiant uid reste inchangé et son identifiant "effectif" devient euid, l'identifiant du propriétaire du programme en cours d'exécution.

Parmi les informations contenues dans le fichier */etc/passwd*, on trouve celles relatives au uid;

8. au_gid (**g**roup **id**entifier) : chaque utilisateur du système peut appartenir à un ou plusieurs groupes. Ces groupes ont un numéro de groupe "gid" et en général un nom. Des informations relatives aux groupes sont définies dans un fichier nommé */etc/group*. En particulier, le groupe du super-utilisateur possède un gid égale à zéro, il porte aussi le nom de **wheel**.

9. au_pid (**p**rocess **id**entifier) : un process (processus) représente l'état dynamique d'un programme. Chaque processus actif est identifié par un numéro unique, appelé "pid";

10. errno : chaque appel système renvoie une valeur (status code), indiquant si l'appel s'est bien déroulé ou pas. En cas d'erreur, la valeur retourné vaut -1, et la variable externe "errno" (définie dans le fichier /sys/errno.h) précise l'erreur.

11. return_value : joue le même rôle que errno;

12. au_label : représente la catégorie et le niveau de sécurité de l'objet dans l'audit record;

13. au_param_count : représente le nombre d'audit data dans le data buffer.

La structure physique du "record header" de chaque record est la suivante :

```
struct audit_record {
        short           au_record_size;
        short           au_record_type;
```

```
        unsigned int    au_event;
        time_t  au_time;
        uid_t           au_uid;
        uid_t           au_auid;
        uid_t           au_euid;
        gid_t           au_gid;
        short           au_pid;
        int             au_errno;
        int             au_return
        blabel_t        au_label;
        short           au_param_count;
};
```

Avec :

uid_t : est un short représenté sur deux bytes;

gid_t : est unsigned short représenté sur deux bytes;

blabel_t : est une structure qui se présente comme suit :

```
struct binary_label {
        short   sl_level;
        char    sl_categories[16];
        char    sl_unused[14];
};
```

3.3.2. Size of sizes

Ces champs représentent un nombre d'"au-param-count" zones de deux bytes. Le contenu de chaque zone détermine la taille de l'audit data lui correspondant dans l'ordre des audit datas.

3.3.3. Data buffer

Le data buffer d'un audit record est une séquence d'un nombre "au-param-count" de données. Cette séquence représente l'information spécifique au type de l'audit record, donc à l'événement audité. Il renferme en général les valeurs des attributs spécifiques à un événement : directorie courante, nom d'un fichier, ...

J'ai constaté que pour tous les audit records, l'attribut du premier audit data est un champ " NULL" représenté sur quatre bytes.

Exemples :

1. type de record : access. Les attributs du data buffer sont :

   - (null);
   - current root (string);
   - current working directory (string);
   - file name (string).

2. type de record (text) : su. Les attributs (tous des string) du data buffer sont :

   - (null);
   - su;
   - message text (bad password, successful su, ...);
   - user name ;
   - user environnement;
   - home directory;
   - shell;

- path.

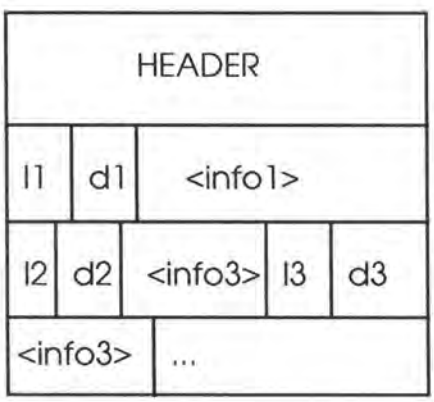
3.4. QUELQUES EXEMPLES DE FORMATS DES AUDIT TRAILS D'AUTRES O.S.

Exemple 1 : **Audit records BS2000 de Siemens**

Le format se présente sous une forme unique et très flexible, il consiste en :

1. un header : est une partie fixe pour tous les audit records et se compose des champs : user_id , task seq. number, event, result, time et date.

2. une partie variable contenant un nombre VARIABLE d'audit data. Chaque audit data est représenté par son identifiant, sa longueur et sa valeur. Les audit datas sont triés sur l'identifiant pour optimiser leur traitement.



Avec : l_i= longueur du ième audit data

id_i = identifiant du ième audit data

$<info_i>$ = valeur du ième audit data

$id_1 <= ..... <= id_n$

Exemple 2 : **Audit records IBM/RACF[7] (Resources Access Control Facility).**

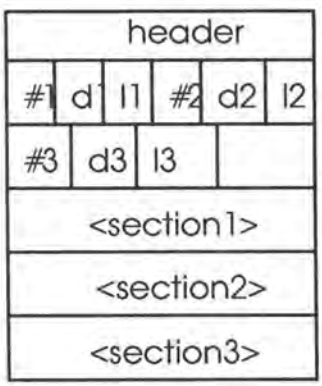
[7] RACF Installation Reference Manuel; sc-28-0734-5, 82

Les événements auditables sont contenus dans des audit records appelés : SMF (System Management Facility) data. Ils ont un des formats suivants : SMF records types 20, 30, 80, 81, 83. [RACF, 82].

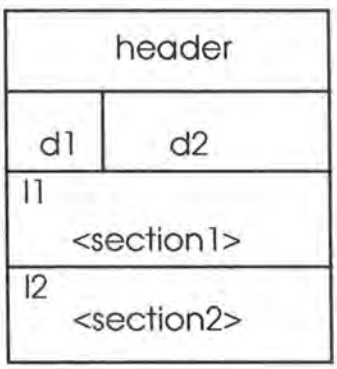
chaque record type consiste en :

1. un header contenant le même type d'informations pour tous les types de records: record length, segment descriptor, system indicator, record type, time, date, system identification;
2. une deuxième partie qui dépend fortement du type du record . Elle est formée souvent d'une description des sections contenues dans le record, suivie des sections elles-mêmes. Le nombre de sections est fixé pour chaque record type.

Record type 20



Record type 30



Avec :

#i = nombre de fois que la section de type i apparaît (#1=2, #2= 0, #3=1)

d_i = offset de la première section de type i en ce qui concerne le début du record

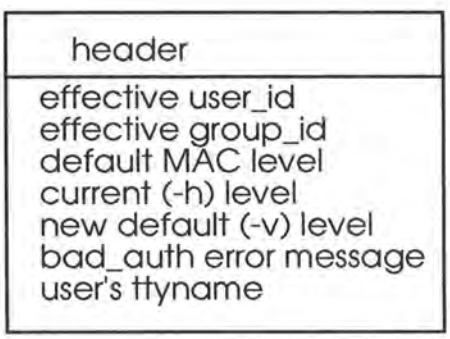
l_i = longueur de la section de type i

Exemple 3 : **Audit records AT&T Unix system 5.4**

Là aussi, à chaque événement correspond un format d'audit record différent. Chaque audit record est constitué d' :

1. un header qui représente une structure commune pour tous les audit records. Il contient les informations suivantes : record type, event subtype, record size, sequence_number, process_id, start time, result (success or failure)

2. un ensemble de données variables suivant le type d'événement. Par exemple, l'événement "login" est représenté par l'audit record suivant :



CHAPITRE 4

L'ADAPTATEUR DE FORMAT POUR SUNOS

4. L'ADAPTATEUR DE FORMAT POUR SUNOS

4.1. INTRODUCTION

Le format des audit trails générés par le système SunOS est non conforme à l'évaluateur ASAX . En effet, comme nous l'avons exposé dans le chapitre 2, ASAX ne reconnaît que des fichiers de format standard nommé NADF (voir section 4.2.1.). Par conséquent, l'implémentation d'un adaptateur de format pour le système SunOS se révèle indispensable. Un tel adaptateur a pour objectif la conversion de tout audit trail produit par le mécanisme d'audit en un fichier de format NADF.

4.2. GÉNÉRALITÉS SUR LES FORMATS ADAPTATEURS

Un adaptateur de format est un module assurant la conversion d'un ou plusieurs fichier(s) structuré(s) en un format approprié à l'outil d'analyse. Dans le cadre de ce mémoire, le format d'origine est le format natif SunOS et le format cible est le NADF. En général, cette translation ne se résume pas à la génération du seul fichier converti, mais consiste aussi à produire divers fichiers auxiliaires. Ces derniers garderont des informations nécessaires à l'analyse (exemple : table de conversion de noms, ...).

L'adaptateur de format s'exécutant sur un fichier natif audit trail, génère en général les fichiers suivants [E.Van Meerbeck, I. Mathieu 92] :

1. un fichier NADF;

2. un fichier de description des audit datas;

3. un fichier de description de conversion, avec éventuellement une librairie contenant les routines de conversion;

4. un fichier de description des règles de conversion.

Le contenu des trois derniers fichiers ne dépend que du format du fichier d'audit natif (donc de l'O.S. qui le génère) et pas de son contenu. Par conséquent, pour toute opération de conversion vers le format standard, ces trois fichiers sont identiques et peuvent être générés une fois pour toute.

Le diagramme des flux d'un adaptateur de format est donné dans la figure suivante :

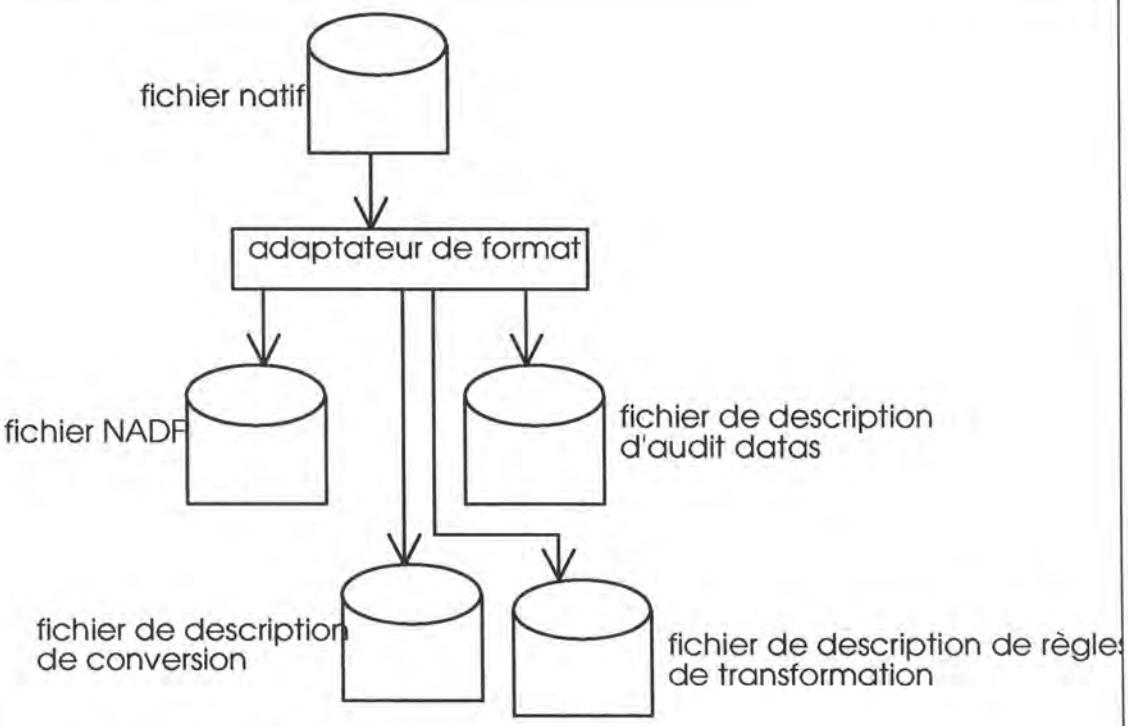


Figure 4.1 : diagramme des flux d'un adaptateur de format

4.3. STRUCTURE D'UN FICHIER NADF

Un fichier NADF est un fichier séquentiel composé de records ayant un format standard appelé NADF (Normalized Audit Data Format). Ce dernier a été inspiré du format des audit records générés par BS2000 vu la flexibilité que celui-ci présente. Le header d'un fichier NADF commence par une séquence de caractères : "__NADF__1N0", ce qui l'identifie parmi d'autres types de fichiers, le reste du fichier est une séquence de records.

4.3.1. Qu'est-ce que le format NADF?

Un record de format NADF possède la structure suivante :

1. real length : un champ de quatre bytes contenant la longueur (en bytes) réelle du record, à laquelle est ajoutée quatre bytes, longueur de la zone contenant le champ de la longueur;

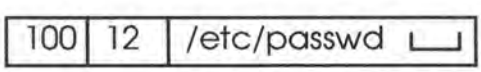
2. le contenu du record : représente une séquence de champs appelés "audit data". Chaque audit data est composé de :

   2.a. son identifiant : un entier représenté sur *deux bytes* et identifiant cet audit data parmi d'autres;

   2.b. sa longueur : est un entier représenté sur *deux bytes* ;

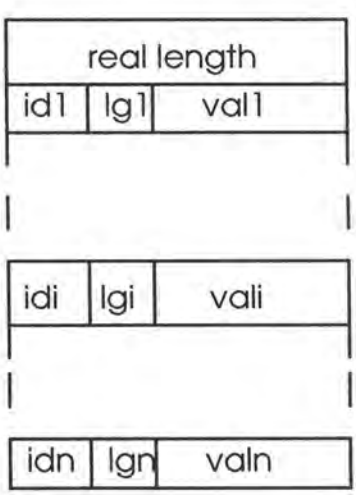
   2.c. sa valeur : c'est la valeur de l'audit data, elle est *alignée* sur un demi-mot.

Exemple : supposons que l'entier '100' soit l'identifiant de l'audit data 'filename' dont la valeur est /etc/passwd. Cette valeur est alors représentée comme suit :



Les deux caractères "espace" sont ajoutés pour aligner la chaîne de caractères "/etc/passwd" sur deux bytes.

Voici ci-dessous l'illustration du format NADF d'un record :



Avec :

idi : est l'identifiant du ième audit data;

lgi : est la longueur de la valeur vali;

vali : est la valeur du ième audit data (la longueur des valeurs est variable).

Les audit datas sont triés sur l'identifiant afin d'optimiser leur traitement :

pour tout i : 1<=i<=n id1< id2< ...< idi< ...< idn.

4.3.2. Transformation d'un record natif en un format NADF

Tout record natif peut être transformé assez facilement en un record NADF. Cependant, un seul cas pose une petite complication de transformation : lorsqu'un ou plusieurs record(s) d'un fichier audit trail contien(nent) des audit datas répétitifs. La conversion d'un tel type de record est un peu plus compliquée. Dans ce genre de situation, le record est décomposé en plusieurs records NADF, tout simplement parce que le langage de règles RUSSEL ne tient pas compte du concept de variables indexées ou du concept liste. Il existe plusieurs façons de faire. En voici une présentée ci-dessous.

Exemple de conversion :

Supposons qu'un audit record contienne k audit data répétitifs, n(k) étant le nombre de fois que l'audit data k est répété. La conversion de cet audit record produit m NADF records où :

m = maximum {n(1), n(2), ..., n(k)};

soit '3' l'identifiant de l'audit data 'filename';

soit '4' l'identifiant de l'audit data 'group';

Voici ci-dessous la structure du record original : il contient 2 audit datas répétitifs (k=2) 'filename' et 'group' dont les valeurs sont répétées respectivement 2 et 3 fois (n(1) = 2, n(2) = 3).

id=1	lg=3	HDR
id=3	lg=4	TOTO
id=3	lg=4	TUTU
id=4	lg=3	GRP
id=4	lg=2	AA
id=4	lg=2	BB

Pour convertir ce record en un format NADF, un nouveau audit data "rec-split" est introduit : soit '5' l'identifiant de cet audit data. La valeur de cet audit data est exprimée en fonction de m et vaut : m - 1. Cette dernière valeur dans le record1 (voir la figure ci-dessous) permet, lors de l'analyse du fichier NADF, de reconnaître que la séquence des trois records (record1, record2, record3) provient de la décompositon d'un même audit record dans le fichier natif.

Résultat de la transformation

Le record est transformé en m records :

Avec :

$$m = \text{maximum}\ \{n(1), n(2)\}$$

$$= \text{maximum}\ \{2, 3\} = 3;$$

Les trois records se présentent ainsi :

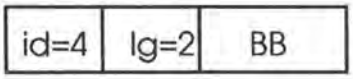
id=1	lg=3	HDR
id=3	lg=4	TOTO
id=4	lg=3	GRP
id=5	lg=2	2

record1

m - 1= 2
(m=3)

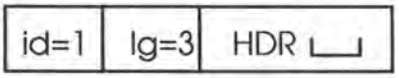
id=3	lg=4	TUTU
id=4	lg=2	AA

record2



record3

En réalité, puisque les valeurs doivent être alignées sur deux bytes, l'audit data du record1 se présente ainsi :



4.4. LES FICHIERS AUXILIAIRES

4.4.1. Fichier de description d'audit datas

Ce fichier correspond à un fichier séquentiel de type texte contenant des informations décrivant les audit datas du fichier converti.

La description de chaque audit data est constituée d'une séquence de cinq lignes : chacune commençe par un numéro allant de 1 à 5. La sémantique de chaque ligne est décrite ci-dessous :

1. identifiant : l'identifiant d'un audit data constitue une contrainte essentielle : c'est un entier qui permet de repérer univoquement chaque audit data parmi tous les audit datas existants. L'identifiant est représenté sur une ligne commençant par le chiffre "1";

2. type d'origine : il indique la représentation interne des audit datas dans le fichier natif. Dans le but de réduire la taille des audit trails, le mécanisme d'audit enregistre quelques audit datas sous leur forme interne (appelée : forme codée). Le type d'origine est représenté sur une ligne commençant par le chiffre "2";

3. type de destination : il indique la représentation interne des audit datas dans le fichier NADF. Actuellement, ASAX a un seul type, il interprète chaque audit data comme étant une chaîne de bytes (string of bytes).Le type de destination est représenté sur une ligne commençant par le chiffre "3";

4. nom externe : à la forme interne est associée une forme externe (une forme lisible), c'est le nom externe de l'audit data. Ce dernier est utilisé lors de la programmation d'une requête. Le nom externe est représenté sur une ligne commençant par le chiffre "4";

5. description sémantique : elle sert à illustrer brièvement la signification de l'audit data. La description sémantique est représentée sur une ligne commençant par le chiffre "5";

L'ensemble de description des audit datas est précédé par une séquence de six lignes spéciales constituant l'en-tête du fichier de description des audit datas. En voici la description :

1. la première ligne commence par la lettre "A" et est suivie du nom de la version du système d'exploitation générateur des audit trails;

2. la première ligne commence par la lettre "B" et est suivie par un ensemble d'informations déterminant les caractéristiques de la machine productrices des audit trails;

3. la première ligne commence par la lettre "C" et est suivie par la date et l'heure auxquelles l'audit trail à été créé;

4. la première ligne commence par la lettre "D" et est suivi par le nom de la version du programme (adaptateur) générateur du fichier NADF;

5. la première ligne commence par la lettre "E" et est suivie par des informations sur les caractéristiques de la machine productrice du fichier NADF;

6. la première ligne commence par la lettre "F" et est suivie par la date et l'heure pendant lesquelles le fichier NADF a été généré.

Voici ci-dessous la syntaxe BNF du fichier de description d'audit datas :

<data description file> ::= <header lines> { <data lines> }

<header lines> ::= <empty_lines> | <A_lines> | <B_lines> | <C_lines>
| <D_lines> | <E_lines> | <F_lines>

for a = A, ..., F :
<a_lines> ::= <a_line> {<a_line>}
<a_line> ::= <empty_lines> | <a> <blank> <any_seq> <eln>
<a> ::= a

<data lines> ::= <1_line> <2_line> <3_line> <4_line> [<5_line>]

<1_line> ::= <empty_lines> 1 <blank> <integer_s> <spaces>
<eln>

<integer_s> ::= 0 | 1 | 2 | ... | 65536
for x = 2, 3, 4 :
<x_line> ::= <empty_lines> <x> <blank> <token> <spaces> <eln>
<x> ::= x

<token> ::= any sequence of letters, underscores and digits beginning with a letter or an underscore

<data_line5> ::= <space> 5 <spaces> <free text>

<5_lines> ::= <5_line> { <5_line> }

<5_line> ::= <empty_lines> 5 <blank> <any_seq> <eln>

<empty lines> ::= any sequence of blanks tabs or end of lines

<blank> ::=

<spaces> ::= any sequence of blanks and tabs

<any_seq> ::= any sequence of characters excluding end of line

<eln> ::= the end of line character

4.4.2. Fichier de description de conversion

Certains audit datas sont représentés dans le fichier natif sous leur forme codée. Pour faciliter la manipulation de ces audit datas lors de la programmation d'une requête, il convient de les présenter sous une forme lisible dans le fichier NADF. La conversion d'une forme à l'autre se fera via des règles de conversion. Le fichier de description de conversion, fichier séquentiel de type texte, renferme des informations sur la conversion de chaque audit data codé.

Le corps de ce fichier est une séquence de six lignes dont voici la description :

1. une ligne contenant l'identifiant de l'audit data précédé par l'indicateur "1";

2. une ligne contenant le type de conversion par l'indicateur "2";

3. une ligne contenant l'information à convertir précédée par l'indicateur "3";

4. une ligne contenant l'information sous sa forme convertie précédé par l'indicateur "4";

5. une ligne contenant le nom de la routine de conversion précédé par l'indicateur "5";

6. une ligne contenant le nom de la routine inverse de conversion précédé par l'indicateur "6".

Exemple :

1 5

2 long int

3 time

4 string

5 ctime

4.4.3. Fichier de description de règles de conversion

C'est un fichier séquentiel de type texte, dans lequel se trouve la description générale concernant l'opération de conversion : c'est-à-dire, des informations sur la règle de conversion, sur le type d'origine, sur le type de destination, et la manière dont s'est déroulée la conversion (du type d'origine au type de destination).

4.5. DESCRIPTION DE L'ADAPTATEUR POUR SUNOS 4.1

Dans le cadre de ce mémoire, l'analyse des audit trails générés par le système SunOS 4.1 demanderait de passer par une première étape : la conversion de l'audit trail en un format NADF. Cette conversion est réalisée par un programme "adaptateur". La réalisation de ce programme à première vue paraît relativement facile : il s'agit de lire dans le fichier natif, de transformer les records natifs en un format NADF et d'enregistrer les records convertis dans le fichier NADF, et ceci jusqu'au "record trailer" qui détermine la fin du fichier natif. Néanmoins, comme la réalisation de n'importe quel programme, des problèmes surviennent. En voici une description.:

1. la structure du fichier : en général, pour traiter un fichier, il faut bien connaître la spécification des records qu'il renferme. Ce qui n'était pas le cas, du moins au début. En faisant appel à BIM, j'ai reçu par mail, une explication non exhaustive mais qui m'a permis de démmarrer.

2. une documentation incomplète :

   a. des appels systèmes dont la spécification est incomplète. Par exemple, l'appel système "*shmget*" (annexe 1) est constitué de trois champs dans le manuel des Sun, alors qu'effectivement il est formé de quatre champs comme exprimé par l'utilitaire *praudit*[10];

   b. l'appel système "*chdir*" ne figure pas parmi les appels systèmes dans le manuel. Par conséquent, j'ai dû interprêter sa structure en me basant sur sa sémantique;

   c. les records texte "*cron*" et "*shutdown*" ne sont pas spécifiés, alors qu'ils sont audités;

   d. des appels systèmes dont la spécification n'est pas claire. Par exemple, "mount nfs", ...;

   e. parfois le manuel des SUN ne pécise pas si un appel système est une structure ou une forme plate.

4.5.1. Les fichiers générés par l'adaptateur pour SunOS 4.1

L'adaptateur de format reçoit en entrée un fichier natif SunOS 4.1 et génère en sortie le fichier converti NADF et un fichier auxiliaire (fichier de description des audit datas), voir la figure 4.2. Les fichiers de conversion cités en 4.5. et en 4.6. ne sont pas générés tout simplement parce que actuellement, ASAX ne possède qu'un seul type string of bytes, qui est le type de destination.

[10] praudit est un utilitaire fournit avec l'O.S. qui permet de visualiser le contenu d'un fichier d'audit.

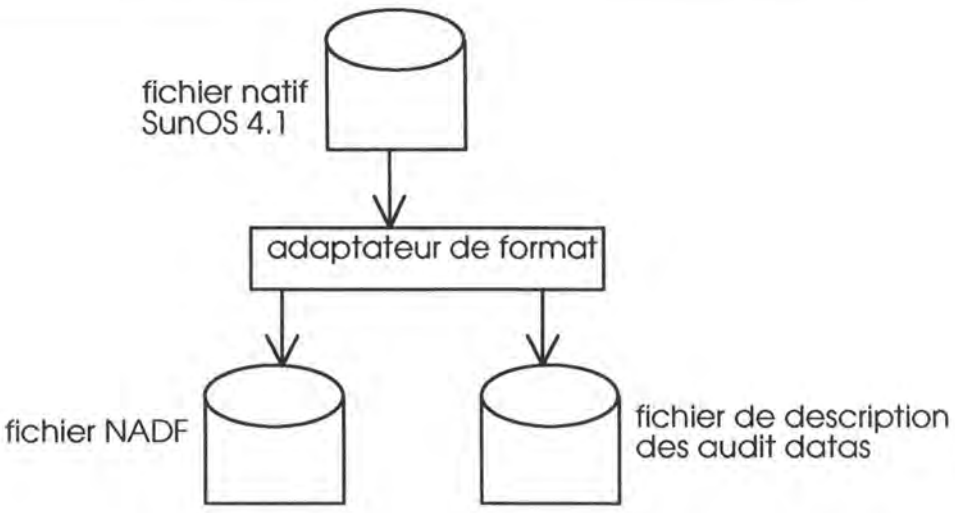


Figure 4.2 : Digramme de flux de l'adaptateur de format pour SunOS 4.1

4.5.1. Implémentation de l'adaptateur

Cette section présente l'algorithme abstrait qui illustre le fonctionnement de l'adaptateur et la spécification des rutines utilisées.

Algorithme :

```
création_en_écriture_fichier_nadf;
ouverture_en_lecture_fichier_natif;
tant que pas_fin_fichier_natif faire
        {
                lire_record_natif;
                convertir_record_natif;
                enregistrer_record_natif.
        }
fermer_fichier_natif;
```

fermer_fichier_nadf;

Convertir_record_natif : cette fonction est un ensemble de "*case*" sur le type du record :

case record_type : **traiter_record**;

break;

Il existe a autant de *case* que de types de records (access, open, chmod, chown, ...).

traiter_record : cette fonction fait appel à des fonctions tels que : charger_I, charger_STR, charger_L, ..., suivant le type (int, long int, string, ...) des audit datas dans le record. La spécification de quelques-unes de ces fonctions figure ci-dessous.

Spécification des routines utilisées :

creat_NADF(filename) : (fonction prédéfinie, implémentée par Siemens)

VARIABLES GLOBALES : /

PRECONDITION :

- filename : nom du fichier NADF a créer

POSTCONDITION:

- Si l'opération de création du fichier réussit, la valeur retournée est positive (>= 0) et représente le file descriptor du fichier NADF créé. Sinon, c'est une valeur négative représentant le code de l'erreur ayant eu lieu.

OPEN_AUDIT(filename)

VARIABLES GLOBALES : /

PRECONDITION :

- filename : nom du fichier à ouvrir

POSTCONDITION :

- Si l'opération d'ouverture du fichier réussit, la valeur retournée est positive (>= 0) et représente le file descriptor du fichier qui est alors ouvert en lecture. Sinon, la valeur -1 est retournée.

READ_AUDIT(fd1, fd2)

VARIABLES GLOBALES :

- char *buffAddr; adresse du buffer contenant le record natif à convertir en format NADF;

- char *tempo; adresse du buffer contenant le record converti en format NADF.

PRECONDITION :

- fd1: file descriptor du fichier natif ouvert en lecture;

- fd2 : file descriptor du fichier NADF ouvert en écriture.

POSTCONDITION:

- le fichier fd2 contient tous les records du fichier fd1 converti sous format NADF.

- l'ordre d'occurrence des records est conservé

charger_db_I(sh, pdf,id)

VARIABLES GLOBALES : /

PRECONDITION :

- pdf est l'adresse d'un entier;

- id est un entier court non signé;

- sh est un pointeur vers l'adresse d'une zone alignée sur un demi-mot;

- soit sh0 la valeur de (*sh).

POSTCONDITION :

- Soit lg = la taille en byte d'un entier;

  v = la valeur pointée par pdf;

- alors sh0 est l'adresse de début de trois zones contiguës :

1. la première zone de 2 bytes contenant la valeur id;

2. la seconde zone de 2 bytes contenant la valeur lg;

3. la troisième zone de 'lg' bytes contenant la valeur v;

- *sh est l'adresse du byte suivant immédiatement la 3-ième zone;

- la valeur retournée est 'lg'.

charger_I(sh, id, val)

VARIABLES GLOBALES : /

PRECONDITION:

- val est un entier;

- id est un entier court non signé;

- sh est un pointeur vers l'adresse d'une zone alignée sur un demi-mot;

- soit sh0 la valeur de (*sh);

POSTCONDITION:

- Soit lg = la taille en byte d'un entier;

- alors sh0 est l'adresse de début de trois zones contigues :

1. la première zone de 2 bytes contenant la valeur id;

2. la seconde zone de 2 bytes contenant la valeur lg;

3. la troisième zone de 'lg' bytes contenant la valeur val;

- *sh est l'adresse du byte suivant immédiatement la 3-ième zone;

- la valeur retournée est 'lg'.

charger_db_S(sh, id, pdf, val)

VARIABLES GLOBALES : /

PRECONDITION:

- val est une valeur entière;

- pdf est un pointeur vers l'adresse d'une zone de val bytes;

- id est un entier court non signé;

- sh est un pointeur vers l'adresse d'une zone alignée sur un demi-mot;

- soit h0 la valeur de (*sh);

POSTCONDITION :

- soit lgr le plus petit multiple de ALIGN_REC supérieur ou égal à val;

- alors sh0 est l'adresse de début de trois zones contiguës :

1. la première zone de 2 bytes contenant la valeur id;

2. la seconde zone de 2 bytes contenant la valeur lgr;

3. la troisième zone de 'lgr' bytes dont les val premiers qui sont pointés par *pdf; le reste étant des blancs.

- *sh est l'adresse du byte suivant la 3-ième zone et étant aligné sur un demi-mot;

- la valeur retournée est 'lgr'.

align(shaux, sh)

VARIABLES GLOBALES : /

PRECONDITION :

- shaux est une valeur entière;

- sh est un pointeur vers l'adresse d'une zone de bytes;

- soit sh0 la valeur de (*sh).

POSTCONDITION :

- Soit al = shaux*(ALIGN_REC - 1))%ALIGN_REC;

- *sh pointe 'al' bytes au-delà de sh0;

- ces bytes sont égaux au caractère blanc;

- la valeur retournée est al.

write_NADF(fd, rec_addr, flush) (fonction prédéfinie, implémentée par Siemens)

VARIABLES GLOBALES : /

PRECONDITION :

- fd est le file descriptor d'un fichier NADF ouvert avec "creat_NADF";

- rec_add est l'adresse du record NADF à écrire;

- flush un flag indiquant si le buffer doit être écrit sur le disque;

POSTCONDITION:

- Si l'opération d'écriture s'est bien déroulée, la valeur reournée est >= 0; sinon une valeur négative est retournée pour signaler le type de l'erreur.

close_NADF(fd) (fonction prédéfinie, implémentée par Siemens)

VARIABLES GLOBALES : /

PRECONDITION :

- fd est le file descriptor d'un fichier NADF ouvert avec "creat_NADF";

POSTCONDITION:

- Si l'opération de fermeture s'est bien déroulée, la valeur retournée est >= 0 sinon une valeur négative est retournée pour signaler le type de l'erreur.

CLOSE_AUDIT(fd)

VARIABLES GLOBALES: /

PRECONDITION:

- fd est le file descriptor d'un fichier NADF ouvert avec "OPEN_AUDIT";

POSTCONDITION:

- Si l'opération de fermeture s'est bien déroulée, la valeur retournée est 0, le buffer est libéré. Sinon une valeur négative est retournée pour signaler le type de l'erreur.

4.6. UNE ANALYSE ON-LINE

Comme nous l'avons exposé dans le chapitre 2, section 2.2., une analyse on-line est possible : plutôt que de générer le fichier NADF et de l'évaluer par après (analyse off-line), chaque recod du fichier natif une fois généré est converti et évalué. Ce processus continue dans l'ordre de création des audit records.

L'adaptateur de format doit lire dans l'input standard (stdin) plutôt que dans un fichier. Dans ASAX, il a été prévu de rendre le processus d'évaluation tout à fait indépendant du mécanisme de saisie des audit records. En effet, il incombe à l'utilisateur d'implémenter ses propres routines d'entrée/sortie pour gérer lui-même la façon dont les audit records sont soumis à l'évaluateur :

- analyse on-line : lecture dans l'input standard;

- analyse dans le mode inversion/évaluation où le fichier natif est directement évalué : un record est lu sur le fichier natif, converti en format NADF et puis transmis à l'évaluateur (pas de fichier NADF intermédiaire).

Pour cacher ces divers modes de saisie des audit records, dans ASAX, il a introduit la notion de fichier "virtuel" ainsi que des opérations de manipulation de tels fichiers : vopen, vread, vclose. Le mot "virtuel" exprime le fait que les opérations précitées agissent sur un fichier d'audit virtuel. Ce dernier peut être un simple fichier d'audit, mais en général, il correspond à un flot séquentiel d'audit records résultant d'un ensemble d'opérations arbitraires (collection, sélection, conversion, fusion, ...) sur un ensemble de fichiers d'audits.

En résumé, l'utilisateur doit fournir un seul module objet implémentant trois routines : vopen, vread, vclose. Ce module est ensuite intégré avec le noyau d'ASAX. En réalité, ce module est soumis à un outil automatisé (**masax**) faisant partie de l'environnement ASAX. Cet outil permet de générer une version exécutable d'ASAX correspondant à ces routines.

Pour accomplir cette analyse, le programme source de l'adaptateur va se réduire en un module ayant trois routines appelées : vopen (virtual open), vread (virtual read) et vclose (virtual close), pour être intégrées avec le noyau d'ASAX.

Spécification des routines vopen, vread et vclose

vopen(arg(c), arg(v))

VARIABLE GLOBALE :

- fdesc : file descriptor du fichier virtuel.

PRECONDITION:

- argv[ ] est un tableau de 'argc' pointeurs vers des strings.

POSTCONDITION :

- le fichier virtuel est ouvert en lecture. Son file descriptor est affecté à la variable global 'fdesc'.

- valeur retournée :

- la valeur 0 est retournée dans le cas où l'opération d'ouverture a réussi;

- une valeur négative est retournée dans le cas contraire. Cette valeur identifie l'erreur commise.

vread(recAddr)

VARIABLE GLOBALE :

- fdesc : file descriptor d'un fichier virtuel préalablement ouvert par 'vopen()'.

PRECONDITION:

- recAddr est l'adresse d'un pointeur de caractère.

POSTCONDITION:

- Si l'opération de lecture s'est bien déroulée :

1. *recAddr est l'adresse du record lu dans le fichier virtuel;

2. la valeur retournée est 0.

- Si une erreur quelconque a eu lieu lors de l'opération de lecture :

1. une valeur négative est retournée. Cette valeur identifie l'erreur commise;

2. *recAddr est indéfini.

vclose( )

VARIABLE GLOBALE:

- fdesc: file descriptor du fichier virtuel;

PRECONDITION:

- 'fdesc' est le file descriptor d'un fichier virtuel prealablement ouvert par "vopen()";

POSTCONDITION:

- si l'opération de fermeture du fichier virtuel identifié par 'fdes' s'est bien deroulée, la valeur retournée est 0; dans le cas contraire, une valeur négative est retournée. Cette valeur identifie l'erreur commise.

4.7. SOLUTION ALTERNATIVE : ADAPTATEUR GÉNÉRIQUE

On peut envisager, comme extension à ce memoire, la réalisation d'un adaptateur générique. autrement dit, plutôt que de produire manuellement un adaptateur pour chaque système particulier, l'idée est de réaliser un adaptateur générique qui serait intelligent et capable de produire automatiquement le code C du programme adaptateur à partir d'une description adéquate du format natif.

La réalisation d'un tel adaptateur demanderait l'implémentation d'un langage de description du format natif recevant la structure adéquate des audit records.

Les routines d'écriture dans le record NADF d'une valeur d'audit data d'un type donné (int, long, string, ...) peuvent être programmées une fois pour toute. En effet, cela consiste à écrire une valeur d'un type donné dans un format bien connu (identifiant | longueur | valeur). Par contre, l'accès au record natif pour exploiter ces valeurs dépend complétement de la structure de ce record (donc de l'O.S. qui le génère). On peut ainsi concevoir des fonctions paramétrisées (des fonctions d'accès aux champs du record) qui exploiteront une description de haut niveau de la structure du record.

CHAPITRE 5

EXEMPLES DE REGLES

5. EXEMPLES DE REGLES

5.1. INTRODUCTION

L'objectif de ce chapitre est d'exposer quelques règles implémentées dans le langage "RUSSEL" (RUle-baSed Sequentiel Evaluation Langage). Ces règles permettent d'illustrer l'utilité de l'évaluateur pour l'analyse des fichiers de sécurité SunOS 4.1.

Avant de présenter quelques règles, et pour plus de clarté, je me pemets de rappeler ce qu'est le langage RUSSEL et de dresser un bref aperçu de la syntaxe et la sémantique de ce langage.

5.2. RAPPEL

D'une façon générale, RUSSEL est un langage de règles permettant d'analyser tout fichier séquentiel structuré, et en particulier les fichiers de sécurité "audit trail", voir (2.3.2).

5.3. SYNTAXE ABSTRAITE DU LANGAGE RUSSEL

<module> ::= <usage list>
<global var decl>
<rule decl list>
<init action>.

<usage list> ::= <empty> |
<module usage> ; ... ; <module usage> ;

<module usage> ::= **uses** <list of modules>

<list of modules> ::= <module name> , ... , <module name>

<module name> ::= identifier

<global var decl> ::= <empty> |
<global var group> ; ... ; <global var group> ;

<global var group> ::= global <variable class>
<variable name> , ... , <variable name> : <type>

<variable class> ::= <empty>
| **internal**
| **external**

<rule decl list> ::= <empty> |
<rule declaration> ; ... ; <rule declaration>

<rule declaration> ::= <rule heading>
<variable declaration part>
<action part>

<rule heading> ::= <rule class>
rule <rule name> (<parameter list>)

<rule class> ::= <empty>
| **internal**
| **external**

<rule name> ::= <identifier>

<parameter list> ::= <empty>
| <parameter group> ; ... ; <parameter group>

<parameter group> ::=
<parameter name> , ... , <parameter name> : <type>

<parameter name> ::= <identifier>

<type> ::= **string**
| **integer**

<variable declaration part> ::= <empty>
| **var** <variable declaration> ; ... ; <variable declaration>

<variable declaration> ::=
< variable name> , ... , <variable name> : <type>

<variable name> ::= <identifier>

<action part > ::= <action>

<action> ::= **skip**
| <assignment>
| <conditional action>
| <repetitive action>
| <compound action>
| <rule triggering>
| <predefined procedure call>

<assignment> ::= < left expression> **:=** <right expression>

<left expression > ::= <parameter name>
| <variable name>
| <global variable name>

<right expression> ::= <expression>

<expression> ::= <constant>
| <field name>
| <left expression>
| <expression> <binary arithmetic operator> <expression>
| <unary arithmetic operator> <expression>

<field name> ::= <identifier>

<conditional action> ::=
if <guarded action> **;** ... **;** <guarded action> **fi**

<guarded action> ::= <condition> -> <action>

<condition> ::= **true**
| **false**
| **present** <field name>
| <expression> <relational operator> <expression>
| <condition> <logical operator> <condition>
| **not** <condition>

<relational operator> ::= < | > | **!=** | <= | >= | %=

<logical operator> ::= **and** | **or**

<repetitive action> ::=
do <guarded action> **;** ... **;** <guarded action> **od**

<compound action> ::=
begin <action> ; ... ; <action> **end**

<rule triggering> ::= **trigger off** <triggering mode> <rule call>

<triggering mode > ::= **for_next**
| **for_current**
| **at_completion**

<predefined procedure call> ::=
<predefined procedure name>
(<expression>, ... ,<expression>)

<predefined procedure name> ::= <identifier>

<rule call> ::= <rule name> (<expression> , ... , <expression>)

<rule name> ::= <identifier>

<init action> ::= **init_action ;**
<variable declaration part> <action part>

5.4. SYNTAXE CONCRÈTE DU LANGAGE RUSSEL

<condition> ::= <disjunction>

<disjunction> ::= <conjunction>
| <disjunction> **or** <conjunction>

<conjunction> ::= <simple condition>
| <conjunction> **and** <simple condition>

<simple condition> ::= **true** | **false** | **present** <field name>
| <relational expression>
| (<condition>)
| **not** <condition>

<relational expression> ::=
<arithmetic expression> <relational operator> <arithmetic expression>

<relational operator> ::= < | > | ≠ | = | ≤ | ≥

<arithmetic expression> ::=
<term>

| <arithmetic expression> <additive operator> <term>

<term> ::= <factor>
| <term> <multiplicative operator> <factor>

<factor> ::= <simple expression>

<simple expression> ::= <global variable name>
| <parameter name>
| <variable name>
| <integer constant>
| (<arithmetic expression>)
| <pre-defined procedure call>

<multiplicative operator> ::= * | **div** | **mod**

<additive operator> ::= + | -

Priorité des opérations définies dans la syntaxe concrète ci-dessus

priorité	opérateurs
1	* , **mod** , **div**
2	+ , –
3	<, > ,≤ ,≥ , =,!=, %=
4	**not**, **present**
5	**and**
6	**or**

Exemple : l'expression suivante :

a + b * c < c + d

est interprétée par l'expression "parenthésée" suivante :

(a + (b * c)) < (c + d) .

5.5. BREF APERÇU SUR LA SÉMANTIQUE DU LANGAGE RUSSEL

Le but ici n'est pas de décrire dans le détail la sémantique opérationnelle du langage RUSSEL. Nous nous contentons de rappeler brièvement les points suivants :

1. évaluation des expressions;

2. exécution des actions;

3. exécution d'un déclenchement de règles;

4. exécution d'un appel de procédure prédéfinie;

5. algorithme général de l'évaluateur.

En effet, ce rappel est utile pour comprendre les exemples de règles présentées dans ce chapitre. Le lecteur peut évidemment consulter [ASAX1] et [ASAX2] pour examiner dans le détail cette sémantique.

Avant d'aborder les points précités, voici ci-dessous les concepts de base introduits :

1. Current Record(CR) : le record courant (dans le format NADF), il se compose d'un(e) :

   - identifiant;

   - longueur;

   - valeur.

2. Current Environment (CE) : l'environnement courant est un ensemble d'informations décrivant l'état d'exécution d'un programme. Il consiste en :

a. le record courant;

b. l'environnement local : ensemble de variables (globales, locales et les paramètres des règles);

c. DStrig : l'ensemble des règles actives ou déclenchées pour le record courant;

d. DSnext : l'ensemble des règles actives pour le record suivant;

e. DScompl : l'ensemble des règles actives après l'analyse de l'entièreté du fichier.

5.5.1. Evaluation des expressions

1. Une constante littérale (C-littéral ou X-littéral) est évaluée au string de bytes qu'elle représente.

2. Une expression de la forme :

present *fieldname*

est évaluée à la valeur VRAI si le record courant contient le champ *fieldname*; sinon elle est évaluée à FAUX.

3. L'évaluation des expressions arithmétiques et relationnelles est tout à fait semblable à celle des langages de programmation comme le C, PASCAL et FORTRAN.

4. Il en est de même pour les expressions conditionnelles simples et composées.

5.5.2. Exécution des actions

1. La construction **skip** : elle correspond à l'action vide. Son exécution n'a aucun effet. Son utilisation permet de clarifier le code d'une règle. **skip** est similaire à l'instruction vide en pascal : "; ;".

2. Application : l'exécution de l'action :

lexpr := *rexpr*

par rapport à l'environnement courant revient à :

- évaluer l'expression *rexpr* par rapport à l'environnement courant. Soit *v* sa valeur;

- affecter la valeur *v* à la variable *lexpr*.

3. Action conditionnelle : l'exécution de l'action :

if
$cond_1$ -> $action_1$;
...
$cond_n$ -> $action_n$
fi

par rapport à l'environnement courant, est effectuée comme suit :

$cond_1$ est évaluée à v

- si v = TRUE l'action $cond_1$ est exécutée par rapport à l'environnement courant et l'exécution est terminée;

- sinon on ne fait rien.

4. Actions répétitives : l'exécution de l'action répétitive :

do
$cond_1$ -> $action_1$;
...
$cond_n$ -> $action_n$
od

par rapport à l'environnement courant, est effectuée de la façon suivante :

- les conditions $cond_1$, ...,$cond_n$ sont successivement évaluées par rapport à l'environnement courant jusqu'à ce que l'une d'entre elles soit évaluée à VRAI. Soit $cond_i$ cette condition. Dans ce cas, l'action $action_i$ est exécutée et ensuite l'action répétitive de départ est exécutée à nouveau;

- si aucune des conditions n'est évaluée à VRAI, l'exécution de l'action répétitive est terminée.

5. Actions composées : l'exécution de l'action :

begin *action1*; ...; $action_n$ **end**

par rapport à l'environnement courant consiste à exécuter successivement les $action_i$ par rapport à cet environnement.

5.5.3. Exécution d'un déclenchement de règle

Soit E un environnement courant et *rule_name* une règle. L'exécution de l'action:

trigger off *tr_mode rule_name* (*expr*$_1$, ..., *expr*$_n$)

avec $n \geq 0$, est effectué comme suit :

1. Les expressions *expr*$_1$, ..., *expr*$_n$ sont évaluées par rapport à l'environnement courant. Soit L = {v_1, ..., v_n} la liste de leurs valeurs respectives;

2. L'effet de l'exécution est d'ajouter cette règle à l'ensemble des règles actives:

- pour le record courant si *tr_mode* = **for_current**;

- pour le record suivant si *tr_mode* = **for_next**;

- à la fin de l'analyse si *tr_mode* = **at_completion**.

5.5.4. Exécution d'un appel de procédure prédéfinie

Soit *proc_name* une procédure prédéfinie, E l'environnement courant. L'exécution de l'appel de procédure prédéfinie :

proc_name (expr$_1$, ..., expr$_n$)

avec n ≥ 0, est effectué comme suit :

1. les expressions expr$_1$, ..., expr$_n$ sont évaluées par rapport à E. Soit v_1, ...,v_n leurs valeurs respectives.

2. l'appel de procédure :

proc_name (v_1, ..., v_n)

est exécuté par rapport à E.

5.5.5. Algorithme général de l'évaluateur

Initialisation :

Scurr := init_action;
Scompl := { };
Ouvrir le fichier natif;

en effet, au départ, la seule règle active est la règle de nom réservée **init_action** alors que l'ensemble des règles actives à la fin de l'analyse est vide.

condition de fin :
fin du fichier NADF

Itération :
lire le record suivant;

exécuter les règles actives pour le record courant en produisant NSnext et NScompl;

Scurr := NSnext;
Scompl := NScompl ∪ Scopml;

en effet, en général, l'exécution des règles actives pour le record courant peut activer d'autres règles pour le record suivant et peut aussi activer d'autres règles pour la fin de l'analyse. Ces derniers viennent s'ajouter à l'ensemble des règles actives à la fin et accumulées jusqu'à maintenant.

Clôture :
fermer le fichier NADF;
exécuter les règles actives à la fin de l'analyse.

On obtient ainsi l'algorithme abstrait de l'évaluateur :

```
Scurr   := init_action;
Scompl := { };
Ouvrir le fichier NADF;

Tant que pas fin du fichier NADF Faire
        lire le record suivant;
        exécuter Scurr donnant NSnext et NScompl;
        Scurr   := NSnext;
        Scompl := NScompl ∪ Scompl;

Fermer le fichier NADF;
Exécuter Scompl.
```

5.6. EXEMPLES DE RÈGLES

5.6.1. Détection d'attaque du mot de passe

La procédure de **login** exige de l'utilisateur, en général, de fournir un mot de passe pour authentifier son identité. Ce mécanisme constitue la première ligne de défense d'un système Unix. Cependant, certains utilisateurs choisissent mal leur mot de passe et rendent leur compte vulnérable aux attaques externes effectuées par des utilisateurs désireux de deviner leur mot de passe.

Dans ce contexte, la règle qui suit implémente la fonctionnalité suivante : "Si un utilisateur effectue plus de *max_times* fois une tentative de **login** infructueuse, endéant *duration* secondes, un message est affiché à l'écran pour prévenir l'officier de sécurité qu'une tentative d'attaque est en cours".

Cette fonctionnalité est implémentée en utilisant deux règles : **failed_login** et **count_rule**.

La première règle détecte un audit record représentant un événement **login** ayant échoué pour un utilisateur donné et à un instant donné; la seconde règle est alors déclenchée par la première et sert à rechercher dans le restant du fichier *max_times* - 1 records représentant un événement **login** infructueux pour le même utilisateur et avant l'expiration de l'intervalle de temps fixé. Aucun événement **login** réussi pour cet utilisateur ne doit avoir lieu entre ces audit records.

Dans le système SunOS 4.1, l'événement de login est représenté par un audit record dont le champs *event* à la valeur '**login_logout**'. Tandis que l'échec de cet événement est représenté par le fait que le champs *'au_text_4'* a la valeur 'incorrect passwd' ou 'ROOT LOGIN REFUSED'.

```
global v1: integer;
rule failed_login(max_times, duration: integer);
begin
    if (event %= 'login_logout')
        and ( (au_text_4 %= 'incorrect password')
        or ( au_text_4 %= 'ROOT LOGIN REFUSED'))

        —> begin
            trigger off for_next
            count_rule(auid), strToInt(time)+duration,
             max_times-1)
           end
    fi;
  trigger off for_next failed_login(max_times, duration)
end;
```

```
rule count_rule(suspect_auid : string; expiration, count_down : integer);
begin
  if auid = suspect_auid
    and (event %= 'login_logout')
          and ( (au_text_4 %= 'incorrect password')
          or ( au_text_4 %= 'ROOT LOGIN REFUSED'))
       and strToInt(time) < expiration
          —> if count_down > 1
                —>
                   trigger off for_next
                   count_rule(suspect_auid, expiration, count_down-1);
                  count_down = 1

                —> begin
                         v1 := v1 + 1;
                         println('too mutch bad login for', strToInt(auid))
                    end
               fi;
        strToInt(time) ≥ expiration  —> skip;

   true —> trigger off for_next
               count_rule(auid,  expiration, count_down)
fi
end;
```

```
rule echo_result;
begin
       println('       ');
       println('*****************************************');
       print('v1');
       println(' sequence(s) of 3 failed logins');
       println('performed during less than 3 seconds');
       println('*****************************************');
       println(' ')
end;
```

```
init_action;
begin
   v1 := 0;
   trigger off for_next failed_login(3,300);
   trigger off at_completion echo_result
end.
```

Remarques :

1. Au lieu d'envoyer un simple message à l'écran, lorsqu'une tentative d'attaque du mot de passe est détectée, on peut imaginer l'exécution d'une procédure prédéfinie dont le but est de désactiver (lock) le compte faisant l'objet de cette attaque;

2. Il importe de bien choisir les valeurs des paramètres de la règle **failed_login** (figurant dans la déclaration de la règle **init_action**). Des valeurs trop petites risquent de générer de fausses alarmes tandis que des valeurs trop larges pourraient laisser inaperçues des attaques réelles du système.

5.6.2. Affichage du contenu d'un fichier audit

Etant donné que les fichiers audit SunOS 4.1 sont sous un format binaire, il est souvent utile de disposer d'un module de règles permettant l'affichage du contenu de ce fichier sous une forme lisible. En effet, cela peut aider à "débuguer" des règles de détection : on a besoin d'afficher le record ayant satisfait à une condition donnée et de vérifier directement qu'il n'y a pas eu d'erreur. Inversément, on peut se baser sur ce format lisible pour construire des règles de détection de scénarii. On simule ainsi un tel scénario et ensuite on peut examiner le contenu des audit records générés par la suite des événements correspondants pour en déduire la règle à programmer.

Dans ce module, on utilise deux règles : la première '**SHOW_REC**' permet d'afficher le record courant et de se réactiver elle-même pour le record suivant; la seconde '**echo_result**' permet simplement d'afficher le nombre de records dans le fichier traité. La procédure prédéfinie '**display_current**' reçoit l'adresse du record courant et affiche son contenu sous le format suivant :

field_name [identifiant longueur] = valeur

```
global v1 : integer;
rule SHOW_REC;

begin
        print('***************************');
       display_current;
       v1 := v1 + 1;
       trigger off for_next SHOW_REC
end;
```

```
rule echo_result;
begin
   println(' ');
   println('*******************************');
   print(v1);
   println(' record(s) ');
   println('*******************************');
   println('')
end;
```

```
init_action;
 begin
   v1 := 0;
   trigger off for_next SHOW_REC;
   trigger off at_completion echo_result
 end.
```

5.6.3. Filtrage du fichier audit sur une condition

Le filtrage (réduction ou présélection) du fichier audit a été évoqué dans le chapitre 2. On peut rappeler que cela permet de réduire la taille du fichier à analyser en ne gardant que certains audit records représentant des événements pertinents pour l'analyse à effectuer. On peut, par exemple, sélectionner tous les

événements provoqués par un utilisateur dont le comportement devient douteux, ou sélectionner les événements **login** pour les regarder de très près au moyen du module **failed_login** exposé pécédemment, ...

Dans ce module, les records satisfaisants une certaine condition de présélection (ou de filtrage) sont écrits sur un fichier (reduction file), les autres sont simplement ignorés. Une variable globale est utilisée pour compter le nombre des records filtrés. La règle d'initialisation **init_action** permet d'ouvrir le fichier de sélection pour le record suivant et d'activer la règle de sélection pour le record qui fermera le fichier de sélection et affichera le nombre de records sélectionnés.

```
global v1: integer;
rule rule_type(fd: integer);
var rc: integer;

begin
        if  condition  %= 'valeur'
              —> begin
                    rc := writeNADF (fd);
                    if rc < 0 --> println ('read error') fi;
                    v1 := v1+1
                  end
       fi;
      trigger off for_next rule_type(fd)
end;
```

```
rule echo_result(fd: integer);
begin

        if closeNADF(fd) < 0 --> println('close error') fi;
       println('**************************************');
       print(v1);
       println(' records trouvés ');
       println('**************************************')
 end;
```

```
init_action;
var n, fd: integer;
 begin
        fd:= creatNADF('NADF.out');
        if fd<0 —> println('pas possible de creer NADF.out ');
              true —> begin
                            v1 := 0;
                             trigger off for_next rule_type(fd);
                             trigger off at_completion echo_result(fd)
                        end
        fi
end.
```

La condition portera sur, par exemple :

- l'identifiant de l'utilisateur : exemple : strToInt (auid) = 312;
- le type du record : exemple : record_type %= 'OPEN'. L'opérateur "%=" sert à comparer deux strings : str1 et str2; str1 et str2 de même préfixe, ayant éventuellement l'un ou l'autre un suffixe composé de caractères blancs.
- le type d'événement : exemple : event %= 'data_write';
- ...

5.6.4. Détection de modification des fichiers système

Le but est d'écrire un module de règles permettant de détecter toute tentative de modification des fichiers système. Une fois qu'un tel événement est découvert dans le fichier audit, un message est affiché à l'écran pour signaler l'identité de l'utilisateur ayant effectué cette modification, le fichier système ayant été modifié ainsi que d'autres informations telles que la date et l'heure de l'événement. De plus, les records représentant cet événement sont écrits sur un fichier de sortie.

Sur SunOS, il s'agit des fichiers sous les directory système suivantes :

- /usr/etc;

- /usr/lib;

- /usr/bin;

- /bin;

- /etc.

La condition de filtrage porte sur le type de l'événement *data_write* et sur l'objet modifié *au_file_name*. On ajoute dans RUSSEL la fonction prédéfinie :

IsPref (*str1*, *str2*)

Elle renvoie la valeur 1 si *str1 est* préfixe de *str2*; sinon elle renvoie 0.

Comme dans les exemples précédents, on utilise une règle qui examine le record courant et se réactive elle-même pour le record suivant. La règle d'initialisation ouvre le fichier de sortie et active la règle de recherche des records satisfaisant la condition. Elle déclenche également une règle **echo-result** qui ferme le fichier de sortie et affiche le nombre total des records trouvés à la fin de l'analyse.

```
global v : integer;
rule corrupted_system_file(fd : integer);
        begin
                if event %= 'data_write'
                and (IsPref('/usr/etc', au_filename) = 1
                        or IsPref('/usr/lib', au_filename) = 1
                        or IsPref('/usr/ucb', au_filename) = 1
                        or IsPref('/usr/bin', au_filename) = 1
                        or IsPref('/bin', au_filename) = 1
                        or IsPref('/etc', au_filename) = 1
                        )
                —> begin
                        println('the system file : ', au_filename);
                        println('was corrupted by the user_id : ', strToInt(auid));
                        println('at the time and the date : ', time(time));
                    end
end
```

Remarques

1. tous les audit records détectés par ce module ne correspondent pas nécessairement à des actions malicieuses. En effet, l'administrateur du système peut très bien installer une nouvelle application dans, par exemple, le répertoire '/bin' ou '/usr/bin';

2. cette règle peut détecter une tentative d'attaque par cheval de Troie. Un utilisateur peut en effet remplacer l'exécutable '/bin/ls' par un exécutable servant à un but malicieux quelconque.

5.6.5 Règle de génération de mesures statistiques

Le but poursuivi ici est d'écrire un module de règles permettant la génération de rapport statistique sur l'utilisation du système (restreint évidemment au contenu du fichier audit analysé).

Le contenu de ce rapport peut, par exemple, reprendre la liste de tous les utilisateurs, le nombre total des processus que chacun d'eux aurait créés, ainsi que le taux d'erreurs de ces processus. Si un utilisateur posséde un taux d'erreurs trop élevé par rapport aux statistiques précédentes, on peut soupçonner une masquarade.

L'implémentation de ce module se base principalement sur trois procédures prédéfinies et sans argument :

1. **initStat**( ) : elle est exécutée par la règle d'initialisation. Elle permet d'initialiser une structure de donnée globale (une table en l'occurrence);

2. **updateStat**( ) : elle est exécutée pour chaque audit record. Elle permet de mettre à jour cette table globale en fonction du contenu du record courant;

3. **printStat**( ) : elle est exécutée à la fin de l'analyse pour afficher le contenu de la table sous forme d'un rapport statistique.

```
rule statistics;
        begin
                updateStat;
                trigger off for_next statistics
        end;
```

```
rule printStatReport;
        begin
                printStat
        end;
```

```
init_action;
        begin
                initStat;
                trigger off for_next statistics;
                trigger off for at_completion printStatReport
        end.
```

6. CONCLUSION

Pendant de nombreuses années, les ordinateurs fonctionnaient sans faire appel à un système particulier de sécurité. Cependant, ceux-ci étaient, au mieux, pourvus d'un mot de passe. De nos jours, ils manipulent des données sensibles qui sont véhiculées la plupart du temps à travers des réseaux informatiques. Il n'est plus question de considérer la sécurité des systèmes informatiques comme un sujet secondaire. A cet égard, plusieurs organisations se sont penchées sur le problème de la sécurité, à leur tête, le département de la Défense des Etats-Unis. Ce dernier a défini des règles de sécurité que doivent satisfaire les systèmes d'exploitation pour être certifiés conformes à différents niveaux, allant du niveau de sécurité minimal jusqu'au niveau auquel la sécurité du système doit être prouvée formellement.

Le système SunOS des stations Sun satisfait à une option de sécurité de niveau C2. Il est doté alors d'un mécanisme d'audit qui consiste à enregistrer dans des fichiers "Audit Trail" des informations retraçant toute interaction avec le système. Ces fichiers d'audit doivent faire l'objet d'une analyse adéquate pour déceler toute tentative d'intrusion. Cette analyse demanderait la présence d'un outil automatisé, efficace et "intelligent". ASAX en est un (peut-être le premier/seul) : il permet d'évaluer tout fichier séquentiel structuré de format normalisé NADF et il dispose d'un langage de règles efficace "RUSSEL" permettant l'expression de requêtes d'analyse.

Avant d'être évalué, un fichier natif SunOS est converti en format NADF. La conversion est assurée par un adaptateur de format réalisé au cours de ce mémoire. La détection de scénarios d'attaques est réalisable par l'implémentation de règles en RUSSEL. L'application de l'évaluateur ASAX pour la détection d'utilisations malveillantes du système a pu être vérifiée concrètement par l'élaboration de règles permettant de telles détections.

On peut envisager la réalisation d'un adaptateur de format générique capable de produire automatiquement le code C du programme adaptateur. Cette réalisation s'effectuera à l'aide d'une description de haut niveau des enregistrements du fichier natif.

Il serait également intéressant de pouvoir analyser, en temps réel, des fichiers audit provenant de plusieurs machines au sein d'un réseau et en particulier du réseau SUN. Ainsi, on pourrait détecter des actions paraissant, a priori, normales pour une machine isolée, mais qui peuvent révéler un scénario d'attaques lorsqu'elles sont considérées à l'échelle du réseau.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[ANDERSON 80]
J.P. Anderson, "Computer Security Threat Monitoring and Surveillance", J.P. Anderson CO, Fort Washington, PA., April 1980.

[ASAX 1]
N. Habra , B. Le Charlier, A. Mounji & I. Mathieu, "Preliminary Report on Advenced Security Audit Trail on UniX", Research Report 1/92, Institut d'informatique, University of Namur, January 1992.

[ASAX 2]
N. Habra , B. Le Charlier, A. Mounji, "Advanced Security Audit Trail Analysis on UniX: Implementation Design of the NADF Evaluator", Research Report 7/92, Institut d'Informatique, University of Namur, March 1992.

[BRUNNSTEIN 91]
K. BRUNNSTEIN, ; FISCHER-HÜBNER, M. SWIMMER, "Concepts of an Expert System for Virus Detection", Proceedings of the 7th IFIP International Conference and Exhibition on Information Security, Brigthon, UK, May 1991.

[DENNING et AL.87]
D. E. DENNING, EDWARDS, JAGANNATHAN, LUNT, PETER, NEUMANN,"A Prototype Ides : a Real-Time Intrusion-Detection Expert System", SRI Project ECU 7508, Menlo Park, August 1987.

[ESORICS 92]
proceedings, Toulouse, France, Nov. 23/25, N. HABRA, B. LE CHARLIER, A. MOUNJI, I. MATHIEU, p. 435-450.

[GISA-EC 90]
"Manual for the Evaluation of Trustworthiness of Information Technology Systems", the German Information Security Agency (GISA), February 1990.

[ITSEC 91]
"Information Technology Security Evaluation Criteria", European Community Advisory Group SOG-Is, June 1991.

[LIBION 91]
F. LIBION, "Towards ' Intelligent' Security Audit Trail Analysis Tools", MS Thesis, FUNDP, Institut d'Informatique, Namur, 1990/1991.

[LUNT et AL. 88a]
T.F. LUNT and R. JAGANNATHAN, "A Prototype Real-Time Intrusion Detection Expert System", Proceedings of the 1988 IEEE Symposium on Security and Privacy,Menlo Park, California, April 1988.

[LUNT 88b]
T.F. LUNT, "Automated Audit Trail Analysis and Intrusion Detection : A Survey", Proceedings of the 11th National Security Conference, Baltimore, MD, October 1988.

[LUNT et AL. 89a]
T.F. LUNT, R. JAGANNATHAN, R. LEE, A. WHITEHURST, S. LISTGARTEN "Knowledge-Based Intrusion Detection", in Proceedings of the 1989 AI Systems in Government Conference, Washington, DC, March 1989.

[LUNT et AL. 89b]
T.F. LUNT, "Real-Time Intrusion Detection", in Proceedings, COMPCON, San Francisco, CA 2/27/89, Spring 1989.

[RACF 82]
"RACF Installation Reference Manual", SC28-0734-5, 1982.

[SEBRING et AL.]
M.M.SEBRING, E. SHELLHOUSE, M. E. HANNA et R. A. WHITEHURST, "Expert System in Intrusion Detection : a case Study", in Proceedings of the 11th National Computer Security Conference, Baltimore, MD, October 1988.

[STOECKEL 93]
R. STOECKEL, "Sécurité et Performances des Systèmes Unix", Ed. Eyrolles, Paris, 1993.

[TCSEC 85]
"Trusted Computer System Evaluation Criteria", in "The Orange Book", Department of Defense, NCSC, National Computer Security Centre, DOD 52OO.28-STD, December 1985.

ANNEXES

Annexe A : Format des audit records générés par SunOS 4.1

1.1.Audit records for system calls

This section describes the audit format for system calls :

access

Record Type = access
Event Types = [dr]
Fields = 3
string = current root
string = current working directory
string = file name

adjtime

Record Type = adjtime
Event Types = [p0]
Fields = 2 types/4 values
long = seconds - adjust by
long = microseconds - and adjust by
long = seconds - old adjustment value
long = microseconds - old adjustment value

chmod

Record Type = chmod
Event Types = [da]
Fields = 4
string = current root
string = current working directory
string = file name
int = new mode

chown

Record Type = chown
Event Types = [da]
Fields = 5
string = current root
string = current working directory
string = file name
int = new user ID (-1 if no change)
int = new group ID (-1 if no change)

chroot

Record Type = chroot
Event Types = [p0]
Fields = 3
string = current root
string = current working directory
string = new root name

core

Record Type = core
Event Types = [dw]
Fields = 3
string = current root
string = current working directory
string = corefile "core" or "more.core"

creat

Record Type = creat
Event Types = [dcldw]
Fields = 3
string = current root
string = current working directory
string = file name

execv

Record Type = execv
Event Types = [dr]
Fields = 3
string = current root
string = current working directory
string = path name

execve

Record Type = execve
Event Types = [dr]
Fields = 3
string = current root
string = current working directory
string = path name

fchmod

Record Type = fchmod
Event Types = [da]
Fields = 2
int = file descriptor
int = new mode

fchown

Record Type = chown
Event Types = [da]
Fields = 3
int = file descriptor
int = new user ID (-1 if no change)
int = new group ID (-1 if no change)

ftruncate

Record Type = ftruncate
Event Types = [dw]
Fields = 2
int = file descriptor
long = length to truncate to

kill

Record Type = kill
Event Types = [dw]
Fields = 2
int = process ID
int = signal number

killpg

Record Type = killpg
Event Types = [dw]
Fields = 2
int = process group ID
int = signal number

link

Record Type = link
Event Types = [dc]
Fields = 4
string = current root
string = current working directory
string = name of file to link to
string = hard link name

mkdir

Record Type = mkdir
Event Types = [dc]
Fields = 3
string = current root
string = current working directory
string = directory name

mknod

Record Type = mknod
Event Types = [dc, p0(only if regular user and not a FIFO)]
Fields = 4
string = current root
string = current working directory
string = file name
string = mode

mount

Record Type = mount
Event Types = [p1]
Fields = 6
string = current root
string = current working directory
int = mount type0:MOUNT_UFS - defined in `mount.h`
string = local mount directory
int = flags - file system read/write and suid, etc - defined in `mount.h`
int = mount type specific arguments

mount ufs

Record Type = mount_ufs
Event Types = [p1]
Fields = 6
string = current root
string = current working directory
int = mount type0:MOUNT_UFS - defined in `mount.h`
string = local mount directory
int = flags - file system read/write and suid, etc - defined in `mount.h`
string = block special file to mount type specific argument

mount nfs

Record Type = mount_nfs
Event Types = [p1]
Fields = 13 types/26 values
string = current root
string = current working directory
int = mount type(1):MOUNT_NFS - defined in `mount.h`
string = local mount directory name
int = flags - file system attributes - defined in `mount.h`
sockaddr_in - file server address consists of the following fields:
    short = family AF_INET=2; defined in socket.h
    unsigned short = port number
    hex = IP address may consist of 1 long, 2 shorts or 4 chars
    char = 8 chars
fhandle_t - file handle to be mounted (differs between client and server)
    client:
        NFS_FHSIXE (32) chars = clients view of fhandle
        (opaque data displayed as four hexadecimal values)
    nfs server:
        2 ints = file system ID
        unsigned short = file number
        char = only used to pad structure
int = flags
int = write size in bytes
int = read size in bytes
int = initial timeout in .1 seconds
int = times to retry send
string = remote host name

msgctl

Record Type = msgctl
Event Types = [dc(IPC_RMID), dw(IPC_SET), dr(IPC_STAT)]
Fields = 1
int = message queue ID

msgget

Record Type = msgget
Event Types = [drldw] { ldc if a new message queue is created}
Fields = 2
int = message queue ID or -1 on error
int = key - type of message queue as defined in ipc.h

msgrcv

Record Type = msgrcv
Event Types = [dr]
Fields = 1
int = message queue ID

msgsnd

Record Type = msgsnd
Event Types = [dw]
Fields = 1
int = message queue ID

open

Record Type = open
Event Types = [dwldr, dw, dr] [ldc]
Fields = 3
string = current root
string = current working directory
string = file name

ptrace

Record Type = ptrace
Event Types = [dw]
Fields = 5
int = request as defined in ptrace.h
int = process ID
int = depends on type of request - see ptrace man page
int = depends on type of request - see ptrace man page
int = depends on type of request - see ptrace man page

quotactl

This is broken down as follows:

quota on

Record Type = quota_on
Event Types = [p0]
Fields = 6
string = current root
string = current working directory
int = quota command - as defined in quota.h
string = device - character or block
int = uid
string = quota file name

quota off

Record Type = quota_off
Event Types = [p0]
Fields = 5
string = current root
string = current working directory
int = quota command - as defined in quota.h
string = device - character or block
int = uid

quota set

Record Type = quota_set
Event Types = [p0]
Fields = 6 types/13 values
string = current root
string = current working directory
int = quota command - as defined in quota.h
string = device - character or block
int = uid
dqblk = defines the format of the disk quota file.
  unsigned int = absolute limit on disk blks alloc
  unsigned int = preferred limit on disk blks
  unsigned int = current block count
  unsigned int = maximum # allocated files + 1
  unsigned int = preferred file limit
  unsigned int = current # allocated files
  unsigned int = time limit for excessive disk use
  unsigned int = time limit for excessive files

quota lim

Record Type = quota_lim
Event Types = [p0]
Fields = 6 types/13 values
string = current root
string = current working directory
int = quota command - as defined in quota.h
string = device - character or block
int = uid
dqblk = defines the format of the disk quota file.
unsigned int = absolute limit on disk blks alloc
unsigned int = preferred limit on disk blks
unsigned int = current block count
unsigned int = maximum # allocated files + 1
unsigned int = preferred file limit
unsigned int = current # allocated files
unsigned int = time limit for excessive disk use
unsigned int = time limit for excessive files

quota sync

Record Type = quota_sync
Event Types = [p0]
Fields = 5
string = current root
string = current working directory
int = quota command - as defined in quota.h
string = device - character or block
int = uid

quota

Record Type = quota
Event Types = [p0]
Fields = 6
string = current root
string = current working directory
int = quota command - as defined in quota.h
string = device - character or block
int = uid
string = argument supplied to quota

readlink

Record Type = readlink
Event Types = [dr]
Fields = 5
string = current root
string = current working directory
string = link name
string = buffer which contain contents of link
int = count of characters in buffer

reboot

Record Type = reboot - always generated before reboot call
Event Types = [p1]
Return Value = successful - AU_EITHER(-1) defined in audit.h
Fields = 1
int = argument to reboot - see reboot2 man page

rebootfailure

Record Type = rebootfail - only generated if reboot fails
Event Types = [p1]
Fields = 1
int = argument to reboot - see reboot2 man page

rename

Record Type = rename
Event Types = [dc]
Fields = 4
string = current root
string = current working directory
string = rename from file name
string = rename to file name

rmdir

Record Type = rmdir
Event Types = [dc]
Fields = 3
string = current root
string = current working directory
string = directory name

semctl

The possible commands are:

GETVAL SETVAL GETPID GETNCNT GETZCNT
GETALL SETALL
IPC_STAT IPC_SET IPC_RMID

The event type value for all commands except the one listed below is zero (0).

Record Type = semctl
Event Types = [dc(IPC_RMID), dw(SETALL), dr(SETZCNT)]
Fields = 1
int = semaphore ID

semget

Record Type = semget
Event Types = [drldw] [ldc if a new semaphore is created]
Fields = 2
int = semaphore ID
int = key

semop

Record Type = semop
Event Types = [drldw]
Fields = 1
int = semaphore ID

setdomainname

Record Type = setdomainname
Event Types = [p1]
Fields = 2
string = old domain name
string = new domain name

sethostname

Record Type = sethostname
Event Types = [p1]
Fields = 2
string = old host name
string = new host name

settimeofday

Record Type = settimeofday
Event Types = [p0]
Fields = 2 types/4 values
long = seconds
long = microseconds
int = minutes west of Greenwich
int = type of dst correction

shmat

Record Type = shmat
Event Types = [drldw(if attach is not read only)ldc(if first attach, create)]
Fields = 1
int = shared memory ID

shmctl

The three shmctl commands are:

IPC_STAT IPC_SET IPC_RMID

The event type for IPC_SET is zero(0).

Record Type = shmctl
Event Types = [dc(IPC_RMID, dr(IPC_STAT)]
Fields = 1
int = shared memory ID

shmdt

Record Type = shmdt
Event Types = [dc]
Fields = 1
int = shared memory ID

shmget

Record Type = shmget
Event Types = [drldw] [ldc - if creating memory segment]
Fields = 2
int = shared memory segment ID or EINVAL if error
int = key

socket

Record Type = socket
Event Types = [dwldrldc]
Fields = 3
int = domain address family; defined in socket.h
int = socket type as defined in socket.h
int = protocols - see socket(2), services(5) and protocols(5) man pages

stat

Record Type = stat
Event Types = [dr]
Fields = 3
string = current root
string = current working directory
string = file name

statfs

Record Type = statfs
Event Types = [dr]
Fields = 4 types/19 values
string = current root
string = current working directory
string = directory name fs is mounted on
statfs = file system statistics - consists of the following fields:
long = type of info, zero for now
long = fundamental file system block size
long = total blocks in file system
long = free block in fs
long = free blocks avail to non-superuser
long = total file nodes in file system
long = free file nodes in fs
long = first part of file system ID
long = second part of file system ID
7 longs = spare for later

symlink

Record Type = symlink
Event Types = [dc]
Fields = 4
string = current root
string = current working directory
string = file name to link to
string = link name

sysacct

Record Type = sysacct
Event Types = [p1ldw]
Fields = 1
string = accounting file name

truncate

Record Type = truncate
Event Types = [dw]
Fields = 4
string = current root
string = current working directory
string = file name
long = length to truncate to

unlink

Record Type = unlink
Event Types = [dc]
Fields = 3
string = current root
string = current working directory
string = file name

unmount

Record Type = unmount
Event Types = [p1]
Fields = 3
string = current root
string = current working directory
string = local mount directory

utimes

Record Type = utimes
Event Types = [dw]
Fields = 4 types/7 values
string = current root
string = current working directory
string = file name to set times on
times consists of:
long = time of last access in seconds
long = and microseconds
long = time of last modification in seconds
long = and microseconds

1.2. Audit Records for Arbitrary Text

This section describes the audit format for arbitrary text. Many programs write text audit records because they perform tasks that don't use system calls, but nevertheless have an impact on security.

text

Record Type = text
Event Types = depends on program
Fields = number depends on program
string = content depends on program

In addition to kernel generated audit messages, several programs generate text audit records. The event type, return values, error codes, and data fields depend on the program that generates the audit record. These fields are described below. The data field lines are proceeded by the data field position number. The first data field is the name of the command.

audit - audit trail maintenance

Event Types = [ad]

1-string = audit
2-string = Invalid input
2-string = Successful

COMMAND = audit -d usernames
2-string = Invalid user name
2-string = Error in getting event state.
2-string = [system error message]
3-string = -d
4-number of users-string = usernames
" " " "

[maximum of ten user names]
" " " "

COMMAND = audit -u username
2-string = Invalid user name
2-string = Error in converting audit flags.
2-string = [system error message]

3-string = -u
4-string = username

COMMAND = audit -s
2-string = Error in audit_data open.
2-string = {system error message}
3-string = -s

COMMAND = audit -n
2-string = Error in audit_data open.
2-string = {system error message}
3-string = -n

clri

Event Types = [ad]
1-string = clri
additional strings = parameters to clri

dcheck

Event Types = [ad]

1-string = dcheck
additional strings = parameters to dcheck

dump

Event Types = [ad]

1-string = dump
additional strings = parameters to dump

fsck

Event Types = [ad]

1-string = fsck
additional strings = parameters to fsck

icheck

Event Types = [ad]

1-string = icheck
additional strings = parameters to icheck

login - sign on

Event Types = [lo]

Return Value = successful 0 {user authenticated}
Return Value = incorrect password 1
Return Value = logins disabled and uid != 0 2
Return Value = ROOT LOGIN REFUSED 3
Return Value = No shell 5

Error Code = same as return values

1-string = login
2-string = login name
3-string = message text, as return value
4-string = terminal name
5-string = host name

ncheck

Event Types = [ad]

1-string = ncheck
additional strings = parameters to ncheck

pwdauthd - server for authenticating passwords

Event Types = [ad]

1-string = pwdauthd

2-string = "user" if authentication was for a user
2-string = "group" if authentication was for a group

3-string = remote user's uid
4-string = name of user or group to authenticate
5-string = password to authenticate

6-string = not using passwd.adjunct
6-string = not using group.adjunct
6-string = valid
6-string = invalid

restore

Event Types = [ad]

1-string = restore
additional strings = parameters to restore

rexd - remote execution daemon

Event Types = [lo]

1-string = in.rexd
2-string = command
3-string = remote machine name
4-string =

5-string = rexd: service rpc register: error
5-string = rexd: svctcp_create: error
5-string = rexd: service rpc register: error

5-string = user authorizated

rexecd - remote execution daemon

Event Types = [lo]

1-string = in.rexecd
2-string = user name
3-string = remote host name
4-string = command to execute

5-string = Login incorrect
5-string = Password incorrect
5-string = No remote directory

5-string = user authorizated

rshd - remote shell daemon

Event Types = [ad]

1-string = in.rshd
2-string = remote user name
3-string = local user name
4-string = host name
5-string = command to execute

6-string = Login incorrect.
6-string = Permission denied
6-string = Can't make pipe
6-string = Error in fork

su - super-user, temporarily switch effective user ID

Event Types = [ad]

Return Value = successful	0 (user authenticated)
Return Value = Unknown login:	1
Return Value = bad password	2
Return Value = setgid	3
Return Value = initgroups failed	4
Return Value = setuid	5
Return Value = no directory	6
Return Value = no shell	7

Error Code = same as return values

1-string = su
2-string = message text
3-string = user name
4-string = user environment; user
5-string = home directory
6-string = shell
7-string = path

yppasswdd - server for modifying NIS password file

Event Types = [ad]

1-string = yppasswdd
2-string = remote user name

3-string = Not in passwd.adjunct.
3-string = Bad password in passwd.adjunct.
3-string = Inconsistency between passwd files.
3-string = Successful.

passwd - Change user password, full name, or shell

Event Types = [ad]

Return Value = Password changed 0
Return Value = Full name changed 0
Return Value = Shell changed 0
Return Value = Program error 1
Return Value = Password file busy 1
Return Value = User not found in passwd file 1
Return Value = Permission denied 1

Error Code = same as return values

1-string = username
1-string = "", when user name is not known

ANNEXE B : Programme du Format Adaptateur pour SunOS 4.1

```
#include<stdio.h>           /* standard I.O library */
#include<sys/label.h>       /* declararation of some data structures */
#include<sys/types.h>       /* declararation of useful data types */
#include<sys/stdtypes.h>
#include <sys/fcntl.h>       /* definition of system I/O constants */
#include<sys/audit.h>        /* declaration of audit record structures */
#include<sys/vfs.h>
#include "NADF_flag.h"       /* file containing the NADF flags    */
#include "SATX_sys.h"        /* definition of system calls */
#include "SATX_error.h"      /* file containing the error symbols */
#define ALIGN_REC   4        /* alignment number */
#define LENGTH_SIZ  4        /* size of the NADF length field */
#include "record.type.tab"   /* table of mapping record types to their names */
#include "event.type.tab"    /* table of mapping event classes to their names */
#include "id_audit"          /* #define's for audit data id's   */
static char *buff_add[MAX_FILES];     /* beginning of the buffers */
static char *buff_end[MAX_FILES];     /* end of the buffers */
static char *buff_ptr[MAX_FILES];     /* current position in the buffers */
static short open_mode[MAX_FILES];    /* flag indicating the open mode  */
                                      /* 'WRITE' or 'READ'              */
static int buff_len[MAX_FILES];       /* real size of the buffers */
static int flush_len[MAX_FILES];      /* flush size to synchonize the */
                                      /* writing of the buffers       */


/* declaration of NADF files header record. This header identifies the format */
/* of an NADF file. */

struct
    {
    int len;
    char val[12];
    } TypeNADF= {15,"__NADF__1|\0"};


#include<sys/user.h>

             /* type definition for time representation */

typedef struct timeval  au_timeval;
typedef struct timezone  au_timezone;

typedef struct audit_record aud_rec;
typedef struct statfs db_stat;

char *buffAddr;      /* address of the buffer holding the current native  */
                     /* record to be converted to NADF format             */

char *tempo;         /* address of the buffer holding the audit record in */
                     /* NADF format                                       */
/*************************************************************************/
/*                                                                       */
/* VARIABLES GLOBALES: /                                                 */
/* PRECONDITION:                                                         */
/*               - argc = 3                                              */
/*               - argv[0] = nom de ce fichier                           */
/*               - argv[1] = nom du fichier natif a convertir            */
/*               - argv[2] = nom du fichier NADF a generer               */
/* POSCONDITION:                                                         */
```

```
/*              - un fichier de nom argv[2] est genere. Ce fichier        */
/*                correspond au fichier de nom argv[1] sous format NADF    */
/*                                                                         */
/*************************************************************************/
main(argc,argv)
int argc;
int *argv[];

    {
    int fd1;   /* file descriptor of the NADF file                */
    int fd2;   /* file descriptor of the native file              */
    int rc1;   /* returned code of the close NADF file operation */


    fd1 = creat_NADF(argv[2]);    /* create an NADF file */
    printf("fd1 = %d\n" , fd1);
    if (fd1<1)
        {
             /* create operation fails */

        printf("%s:cannot create %s\n",argv[0],argv[2]);
        exit(1);
        }

      /* open the native file to be converted to NADF format */

    fd2 = OPEN_AUDIT(argv[1]);
    printf("fd2 = %d\n" , fd2);
    if (fd2<1)
        {
            /* open operation fails */

        printf("%s:cannot open %s\n",argv[0],argv[1]);
        exit(1);
        }

    READ_AUDIT(fd1, fd2);  /* adapt the native file  fd1 to the NADF file fd2 */
    CLOSE_AUDIT(fd2);      /* close the native file */
    rc1=close_NADF(fd1);   /* close the NADF file  and return code */
    printf("rc1 = %d\n", rc1);
    }
/*************************************************************************/
/*                                                                       */
/* VARIABLES GLOBALES: /                                                 */
/* PRECONDITION:                                                         */
/*               - filename: nom du fichier NADF a creer                 */
/* POSCONDITION:                                                         */
/*               - Si l'operation de creation du fichier reussi, la valeur */
/*                 retournee est positive ( >= 0) et represente le file  */
/*                 descriptor du fichier NADF creee. Sinon, c'est une    */
/*                 valeur negative representant representant le code de  */
/*                 l'erreur ayant eu lieu                                */
/*                                                                       */
/*************************************************************************/

creat_NADF(filename)
char *filename;
    {
    int fd;                  /* file descriptor */
    int rc;                  /* return code of 'write_NADF' */
    int buffLen = BUF_SIZE;  /* size of the buffer */
```

```
char *buffAdd;           /* pointer to the memory area */
char cmd[MAXPATHLEN+4];  /* command line to erase the file */
char *malloc();

fd = open(filename, O_RDONLY);  /* Checking of the existence of */
                                /* the file                     */

if (fd >= 0)                    /* If the file already exists   */
    {
    close(fd);
    return(ESFILEEX);              /* error : the file exists */
    }
else
    {
    fd = creat(filename, PROT_NADF); /* 'PROT_NADF' is the protection */
                                     /* of the NADF file (by default) */

    if ((fd >= 0) && (fd < MAX_FILES)) /* If no problem during 'create'*/
        {
        /* Try to allocate a dynamic area as great as possible */

        buffAdd = NULL;   /* no memory area */

        while ((buffAdd == NULL) && (buffLen >= BUFSIZ))
          {
          buffAdd = malloc(buffLen);   /* Ask for a dynamic area */

          if (buffAdd == NULL)          /* Allocation problem */
              {
              /* decrease the size of the buffer required */
              buffLen = buffLen - BUFSIZ;
              }
          }
        if (buffAdd == NULL)
            {
            return(ESMALLOC);          /* Allocation error */
            }
        open_mode[fd] = WRITE;       /* to store the open mode */

        /* storage of the buffer information */
        buff_add[fd] = buffAdd;
        buff_end[fd] = buffAdd+buffLen;
        buff_ptr[fd] = buffAdd;
        buff_len[fd] = buffLen;

        rc = write_NADF(fd, &TypeNADF, NO_FLUSH);
                                   /* 'TypeNADF' is a NADF record */
                                   /* indicating the NADF format  */
        if (rc < 0)
            {
            return(rc);     /* error during the 'write_NADF' */
            }

        return(fd);                /* returning file descriptor */
        }
    else
        {
        /* if the maximum of open files is reached */
        if (fd >= MAX_FILES)
            {
            /* closing of the file */
```

```
                close(fd);

                /* erase the file */
                sprintf(cmd, "rm %s", filename);
                system(cmd);

                return(ESOVERFLOW);  /* error: limit of open files */
                }

            return(ESBADCREAT);     /* error: no create executed */
            }
        }
    }

/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES: /                                                     */
/* PRECONDITION:                                                             */
/*                - filename: nom du fichier a ouvrir                        */
/* POSCONDITION:                                                             */
/*                - Si l'operation d'ouverture du fichier reussi, la valeur  */
/*                  retournee est positive ( >= 0) et represente le file     */
/*                  descriptor du fichier qui est alors ouvert en lecture.   */
/*                  Sinon, la valeur -1 est retournee                        */
/*                                                                           */
/*****************************************************************************/


OPEN_AUDIT(filename)
char *filename;
    {
    int  fd;                    /* file descriptor */

    fd = open(filename,O_RDONLY);
    if (fd >= 0)    /* if the open succeeded */
        {

        return(fd);

        }
    else
        {
        return(-1);  /* the open operation failed */
        }
}

/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES:                                                       */
/*                                                                           */
/*                - char *buffAddr; adresse du buffer contenant le record    */
/*                                  natif a convertir en format NADF         */
/*                - char *tempo;    adresse du buffer contenant le record    */
/*                                  converti en format NADF                  */
/* PRECONDITION:                                                             */
/*                - fd1: file descriptor du fichier natif ouvert en lecture  */
/*                - fd2: file descriptor du fichier NADF ouvert en ecriture  */
/* POSCONDITION:                                                             */
/*                - le fichier fd2 contient tous les records du ficher fd1   */
/*                  convertis sous foemat NADF.                              */
```

```
/*              - l'ordre d'occurrence des records est conserve            */
/*                                                                         */
/*************************************************************************/


READ_AUDIT(fd1,fd)
int fd1,fd;
    {
    int nb_read;      /* number of actually read bytes                 */
    int dbsz;         /* data buffer size                              */
    int temp;         /* number of data in the data buffer             */
    int rc;           /* returned code from write NADF operation       */
    int i;            /* loop counter in the data buffer               */
    int len_trail;    /* audit trail file name length                  */
    int rlen;         /* auxiliary to count the length of NADF record  */
    int *ih;          /* pointer to the record header                  */
    int *ent;         /* auxiliary variable                            */

         /* miscelanuous auxilary variables */

    unsigned int *uent;
    short *sh,*sz;
    long *lg;
    char *pdf,*rlen_ptr, *pdf1;
    aud_rec *ph;
    audit_trailer_t *atrail;
    char *pci, *pcl;
    db_stat  *pstatfs;
    au_timeval *ptimeval;
    au_timezone *ptimezone;
    audit_header_t *ahead;
    int len_header, var=0;
    int  buffLen =2*BUF_SIZE;   /* Length of the buffer */

        buffAddr = malloc(buffLen);     /* Ask for a dynamic area */
        nb_read = read(fd, buffAddr, sizeof(audit_header_t));

              if (nb_read  < 0)
                  {
                  printf("read_error\n");
                  exit(1);
                  }
        ahead=(audit_header_t *) buffAddr;
        len_header=ahead->ah_namelen;

        rlen_ptr=malloc(buffLen);
        tempo=rlen_ptr;
        rlen=sizeof(int)+6*sizeof(short);
        rlen_ptr=rlen_ptr+sizeof(int);
        sh=(short *) rlen_ptr;

        rlen+=charger_I(&sh,magic_number,ahead->ah_magic);
        rlen+=charger_L(&sh,ahtime,ahead->ah_time, 24);
        rlen+=charger_SH(&sh,namelen,ahead->ah_namelen);
        ih=(int *) tempo;
        *ih=rlen;

        nb_read= read(fd,buffAddr,len_header);
        if(nb_read<0){
                printf("read error");
                exit(1);
```

```
    }
    if ((rc=write_NADF(fd1,tempo,FLUSH))<0){
            printf("wirte error\n");
            exit(1);
            }

    while (var == 0 )
    {

rlen_ptr=tempo;
nb_read = read(fd, buffAddr, sizeof(aud_rec));
 ph = (aud_rec *) buffAddr;
if(nb_read<0)
    {
    printf("read error\n");
    exit(1);
    }
    else if (ph->au_record_type == 1000)
             {
             atrail=(audit_trailer_t *) buffAddr;
             len_trail = atrail->at_namelen;
             rlen=sizeof(int)+6*sizeof(short);
             sh=(short *) (rlen_ptr + sizeof(int) );
             rlen+=charger_L(&sh,attime,atrail->at_time,24);
             rlen+=charger_SH(&sh,atrecord_type,atrail->at_record_type);
             rlen+=charger_SH(&sh,atnamelen,atrail->at_namelen);
            ih=(int *) tempo;
            *ih=rlen;
            var=1;
            if ((rc=write_NADF(fd1,tempo,FLUSH))<0)
                 {
                 printf("wirte error\n");
                 exit(1);
                 }
            return(-1);
            }
    else  if (nb_read== sizeof(aud_rec))
              {
              rlen=sizeof(int)+26*sizeof(short);
              rlen_ptr=rlen_ptr+sizeof(int);
              sh=(short *) rlen_ptr;


              rlen+=class_event(&sh,ph->au_event);
              rlen+=charger_L(&sh,time,ph->au_time);
              rlen+=charger_STR(&sh,record_type,
              record_type_tab[ph->au_record_type].record_name,13);
              rlen+=charger_SH(&sh,uid,ph->au_uid);
              rlen+=charger_SH(&sh,auid,ph->au_auid);
              rlen+=charger_SH(&sh,euid,ph->au_euid);
              rlen+=charger_USH(&sh,gid,ph->au_gid);
              rlen+=charger_SH(&sh,pid,ph->au_pid);
              rlen+=charger_I(&sh,errnb,ph->au_errno);
              rlen+=charger_I(&sh,return_value,ph->au_return);
              rlen+=charger_SH(&sh,label_level,ph->au_label.sl_level);
              rlen+=charger_STR(&sh,label_categorie,
                                ph->au_label.sl_categories,16);
              rlen+=charger_STR(&sh,label_unused,ph->au_label.sl_unused,
                                                          14);
              dbsz=ph->au_record_size-sizeof(aud_rec);
              temp=ph->au_param_count;
```

```
switch (ph->au_record_type)
        {
                case AUR_ACCESS:

                        nb_read=read(fd, buffAddr, dbsz);
                        if (nb_read<0)
                                {
                                printf("read error\n");
                                exit(1);
                                }

                        sz=(short *) buffAddr;
                        pdf=buffAddr+sizeof(short)*temp;
                        rlen=rlen+temp*2*sizeof(short);

                        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                        pdf=pdf+*sz;
                        sz++;

                        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
                        pdf=pdf+*sz;
                        sz++;

                       rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
                       pdf=pdf+*sz;
                       sz++;

                       rlen+=charger_db_S(&sh,file_name,&pdf,*sz);
                       ih=(int *) tempo;
                       *ih=rlen;
                       break;

               case AUR_ADJTIME:

                        nb_read=read(fd, buffAddr, dbsz);
                        if (nb_read<0)
                                {
                                printf("read error\n");
                                exit(1);
                                }

                        sz=(short *) buffAddr;
                        pdf=buffAddr+sizeof(short)*temp;
                        rlen=rlen+temp*2*sizeof(short);

                        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                        pdf=pdf+*sz;
                        sz++;

                        pcl = (char *) malloc(sizeof(long));
                        copy_bytes(pdf, pcl, sizeof(long));
                        lg=(long *) pcl;
                        rlen+=charger_db_L(&sh,pdf,s_adj_by,*lg);
                        free(pcl);
                        pdf=pdf+*sz;
                        sz++;


                        pcl = (char *) malloc(sizeof(long));
                        copy_bytes(pdf, pcl, sizeof(long));
```

```
                lg=(long *) pcl;
                rlen+=charger_db_L(&sh,pdf,ms_adj_by,*lg);
                free(pcl);
                pdf=pdf+*sz;
                sz++;


                pcl = (char *) malloc(sizeof(long));
                copy_bytes(pdf, pcl, sizeof(long));
                lg=(long *) pcl;
                rlen+=charger_db_L(&sh,pdf,s_old,*lg);
                free(pcl);
                pdf=pdf+*sz;
                sz++;

                pcl = (char *) malloc(sizeof(long));
                copy_bytes(pdf, pcl, sizeof(long));
                lg=(long *) pcl;
                rlen+=charger_db_L(&sh,pdf,ms_old,*lg);
                ih=(int *) tempo;
                *ih=rlen;
                free(pcl);
                break;

        case AUR_CHMOD:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

               rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,file_name,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                pci=(char *) malloc(sizeof(int));
                copy_bytes(pdf,pci,sizeof(int));
                rlen+=charger_db_I(&sh,pdf,new_mode);
                ih=(int *) tempo;
                *ih=rlen;
                free(pci);
                break;
```

```
case AUR_CHOWN:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

       rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,file_name,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,new_usr,*ent);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,new_group,*ent);
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_CHROOT:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;
```

```
                rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,nr_name,&pdf,*sz);
                ih=(int *) tempo;
                *ih=rlen;
                break;

        case AUR_CHDIR:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,new_dir,&pdf,*sz);
                ih=(int *) tempo;
                *ih=rlen;
                break;

        case AUR_CORE:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
```

```
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,corefile,&pdf,*sz);
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_CREAT:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

       rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,file_name,&pdf,*sz);
        ih=(int *) tempo;
        *ih=rlen;
         break;

case AUR_EXECV:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

       rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
        pdf=pdf+*sz;
```

```
        sz++;

        rlen+=charger_db_S(&sh,path_name,&pdf,*sz);
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_EXECVE:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

       rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,path_name,&pdf,*sz);
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_FCHMOD:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        pci = (char *) malloc(sizeof(int));
        copy_bytes(pdf, pci, sizeof(int));
        rlen+= charger_db_I(&sh,pdf,file_descrp);
        free(pci);
        pdf=pdf+*sz;
        sz++;
```

```
        pci = (char *) malloc(sizeof(int));
        copy_bytes(pdf, pci, sizeof(int));
        rlen+= charger_db_I(&sh,pdf,new_mode);
        ih=(int *) tempo;
        *ih=rlen;
        free(pci);
        break;

case AUR_FCHOWN:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,file_descrp,*ent);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,new_usr,*ent);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,new_group,*ent);
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_FTRUNCATE:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        pci = (char *) malloc(sizeof(int));
        pcl = (char *) malloc(sizeof(long));
        copy_bytes(pdf, pci, sizeof(int));
```

```
                rlen+= charger_db_I(&sh,pdf,file_descrp);
                free(pci);
                pdf=pdf+*sz;
                sz++;

                copy_bytes(pdf, pcl, sizeof(long));
                lg=(long *) pcl;
                rlen+= charger_db_L(&sh,pdf,l_trun_to,*lg);
                ih=(int *) tempo;
                *ih=rlen;
                free(pci);
                free(pcl);
                break;

        case AUR_KILL:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                pci= (char *) malloc(2*sizeof(int));
                copy_bytes(pdf, pci, 2*sizeof(int));
                pdf=pci;
                ent=(int *) pdf;
                rlen+=charger_I(&sh,process_id,*ent);
                pdf=pdf+*sz;
                sz++;

                ent=(int *) pdf;
                rlen+=charger_I(&sh,signal_nb,*ent);
                ih=(int *) tempo;
                *ih=rlen;
                free(pci);
                break;

        case AUR_KILLPG:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
```

```
        sz++;

        pci = (char *) malloc(sizeof(int));
        copy_bytes(pdf, pci, sizeof(int));
        rlen+= charger_db_I(&sh,pdf,process_gr);
        free(pci);
        pdf=pdf+*sz;
        sz++;

        pci = (char *) malloc(sizeof(int));
        copy_bytes(pdf, pci, sizeof(int));
        rlen+= charger_db_I(&sh,pdf,signal_nb);
        ih=(int *) tempo;
        *ih=rlen;
        free(pci);
        break;

case AUR_LINK:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;


       rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,nf_link,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,hd_linkn,&pdf,*sz);
        ih=(int *) tempo;
        *ih=rlen;
        break;


case AUR_MKDIR:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }
```

```
                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

               rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,name_dir,&pdf,*sz);
                ih=(int *) tempo;
                *ih=rlen;
                break;

        case AUR_MKNOD:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

               rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,file_name,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,mode,&pdf,*sz);
                ih=(int *) tempo;
                *ih=rlen;
                break;

        case AUR_MOUNT:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
```

```
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);


        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;


        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;


       rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,type_mount,*ent);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,loc_mount_dir,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,mount_flag,*ent);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,mount_arg,*ent);
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_MOUNT_UFS:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
```

```
            pdf=pdf+*sz;
            sz++;

           rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;

            ent=(int *) pdf;
            rlen+=charger_I(&sh,type_mount,*ent);
            pdf=pdf+*sz;
            sz++;

          rlen+=charger_db_S(&sh,loc_mount_dir,&pdf,*sz);
            pdf=pdf+*sz;
                      sz++;

            ent=(int *) pdf;
            rlen+=charger_I(&sh,mount_flag,*ent);
            pdf=pdf+*sz;
            sz++;

            ent=(int *) pdf;
            rlen+=charger_I(&sh,mount_block,*ent);
            ih=(int *) tempo;
            *ih=rlen;
            break;

/* case AUR_MOUNT_NFS : */

case AUR_MSGCTL:

            nb_read=read(fd, buffAddr, dbsz);
            if (nb_read<0)
                    {
                    printf("read error\n");
                    exit(1);
                    }

            sz=(short *) buffAddr;
            pdf=buffAddr+sizeof(short)*temp;
            rlen=rlen+temp*2*sizeof(short);

            rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;

            ent=(int *) pdf;
            rlen+=charger_I(&sh,mesg_queue,*ent);
            ih=(int *) tempo;
            *ih=rlen;
            break;

case AUR_MSGGET:

            nb_read=read(fd, buffAddr, dbsz);
            if (nb_read<0)
                    {
                    printf("read error\n");
                    exit(1);
                    }
```

```
                sz=(short *) buffAddr;
                        pdf=buffAddr+sizeof(short)*temp;
                        rlen=rlen+temp*2*sizeof(short);


                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                        pdf=pdf+*sz;
                        sz++;

                pci = (char *) malloc(2*sizeof(int));
                copy_bytes(pdf,pci,2*sizeof(int));
                pdf=pci;
                ent=(int *) pdf;
                rlen+=charger_I(&sh,mesg_queue,*ent);
                pdf=pdf+*sz;
                sz++;

                ent=(int *) pdf;
                rlen+=charger_I(&sh,mesg_key,*ent);
                ih=(int *) tempo;
                *ih=rlen;
                break;

        case AUR_MSGRCV:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                pci = (char *) malloc(sizeof(int));
                copy_bytes(pdf, pci, sizeof(int));
                rlen+= charger_db_I(&sh,pdf,mesg_queue);
                ih=(int *) tempo;
                *ih=rlen;
                free(pci);
                break;

        case AUR_MSGSND:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);
```

```
                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                ent=(int *) pdf;
                rlen+=charger_I(&sh,mesg_queue,*ent);
                ih=(int *) tempo;
                *ih=rlen;
                break;

        case AUR_OPEN:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;


                rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

               rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,file_name,&pdf,*sz);
                ih=(int *) tempo;
                *ih=rlen;
                break;

        case AUR_PTRACE:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                ent=(int *) pdf;
                rlen+=charger_I(&sh,request,*ent);
                pdf=pdf+*sz;
```

```
                sz++;

                ent=(int *) pdf;
                rlen+=charger_I(&sh,proc_ptrace,*ent);
                pdf=pdf+*sz;
                sz++;

                ent=(int *) pdf;
                rlen+=charger_I(&sh,ptrace_1,*ent);
                pdf=pdf+*sz;
                sz++;

                ent=(int *) pdf;
                rlen+=charger_I(&sh,ptrace_2,*ent);
                pdf=pdf+*sz;
                sz++;

                ent=(int *) pdf;
                rlen+=charger_I(&sh,ptrace_3,*ent);
                ih=(int *) tempo;
                *ih=rlen;
                break;

        case AUR_QUOTA_ON:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0){
                        printf("read error\n");
                        exit(1);
                }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;


               rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                ent = (int *) pdf;
                rlen+=charger_I(&sh,quota_cmd,*ent);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,device,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                ent = (int *) pdf;
                rlen+=charger_I(&sh,quota,*ent);
                pdf=pdf+*sz;
                sz++;
```

```
        rlen+=charger_db_S(&sh,quota_filename,&pdf,*sz);
           ih=(int *) tempo;
           *ih=rlen;
           break;

    case AUR_QUOTA_OFF:

           nb_read=read(fd, buffAddr, dbsz);
           if (nb_read<0)
                   {
                   printf("read error\n");
                   exit(1);
                   }

           sz=(short *) buffAddr;
           pdf=buffAddr+sizeof(short)*temp;
           rlen=rlen+temp*2*sizeof(short);

           rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
           pdf=pdf+*sz;
           sz++;


           rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
           pdf=pdf+*sz;
           sz++;


          rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
           pdf=pdf+*sz;
           sz++;

           ent = (int *) pdf;
           rlen+=charger_I(&sh,quota_cmd,*ent);
           pdf=pdf+*sz;
           sz++;

           rlen+=charger_db_S(&sh,device,&pdf,*sz);
           pdf=pdf+*sz;
           sz++;

           ent = (int *) pdf;
           rlen+=charger_I(&sh,quota,*ent);
           ih=(int *) tempo;
           *ih=rlen;
           break;

    case AUR_QUOTA_SET:

           nb_read=read(fd, buffAddr, dbsz);
           if (nb_read<0)
                   {
                   printf("read error\n");
                   exit(1);
                   }

           sz=(short *) buffAddr;
           pdf=buffAddr+sizeof(short)*temp;
           rlen=rlen+temp*2*sizeof(short);
```

```
 rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
 pdf=pdf+*sz;
 sz++;

 rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
 pdf=pdf+*sz;
 sz++;


rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
 pdf=pdf+*sz;
 sz++;

 ent = (int *) pdf;
 rlen+=charger_I(&sh,quota_cmd,*ent);
 pdf=pdf+*sz;
 sz++;

 rlen+=charger_db_S(&sh,device,&pdf,*sz);
 pdf=pdf+*sz;
 sz++;

 ent = (int *) pdf;
 rlen+=charger_I(&sh,quota,*ent);
 pdf=pdf+*sz;
 sz++;

 uent=( unsigned int *) pdf;
 rlen+=charger_UI(&sh,dblk_1,*uent);
 pdf=pdf+*sz;
 sz++;

 uent=( unsigned int *) pdf;
 rlen+=charger_UI(&sh,dblk_2,*uent);
 pdf=pdf+*sz;
          sz++;

 uent=( unsigned int *) pdf;
 rlen+=charger_UI(&sh,dblk_3,*uent);
 pdf=pdf+*sz;
 sz++;

 uent=( unsigned int *) pdf;
 rlen+=charger_UI(&sh,dblk_4,*uent);
 pdf=pdf+*sz;
 sz++;

 uent=( unsigned int *) pdf;
 rlen+=charger_UI(&sh,dblk_5,*uent);
 pdf=pdf+*sz;
          sz++;

 uent=( unsigned int *) pdf;
 rlen+=charger_UI(&sh,dblk_6,*uent);
 pdf=pdf+*sz;
 sz++;

 uent=( unsigned int *) pdf;
 rlen+=charger_UI(&sh,dblk_7,*uent);
 pdf=pdf+*sz;
 sz++;
```

```
                uent=( unsigned int *) pdf;
                rlen+=charger_UI(&sh,dblk_8,*uent);
                ih=(int *) tempo;
                *ih=rlen;
                break;

        case AUR_QUOTA_LIM:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

               rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                ent = (int *) pdf;
                rlen+=charger_I(&sh,quota_cmd,*ent);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,device,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                ent = (int *) pdf;
                rlen+=charger_I(&sh,quota,*ent);
                pdf=pdf+*sz;
                sz++;

                uent=( unsigned int *) pdf;
                rlen+=charger_UI(&sh,dblk_1,*uent);
                pdf=pdf+*sz;
                sz++;

                uent=( unsigned int *) pdf;
                rlen+=charger_UI(&sh,dblk_2,*uent);
                pdf=pdf+*sz;
                sz++;

                uent=( unsigned int *) pdf;
                rlen+=charger_UI(&sh,dblk_3,*uent);
                pdf=pdf+*sz;
                sz++;
```

```
                uent=( unsigned int *) pdf;
                rlen+=charger_UI(&sh,dblk_4,*uent);
                pdf=pdf+*sz;
                sz++;

                uent=( unsigned int *) pdf;
                rlen+=charger_UI(&sh,dblk_5,*uent);
                pdf=pdf+*sz;
                sz++;

                uent=( unsigned int *) pdf;
                rlen+=charger_UI(&sh,dblk_6,*uent);
                pdf=pdf+*sz;
                sz++;

                uent=( unsigned int *) pdf;
                rlen+=charger_UI(&sh,dblk_7,*uent);
                pdf=pdf+*sz;
                sz++;

                uent=( unsigned int *) pdf;
                rlen+=charger_UI(&sh,dblk_8,*uent);
                ih=(int *) tempo;
                *ih=rlen;
                break;

        case AUR_QUOTA_SYNC:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;


                rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;


               rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                ent = (int *) pdf;
                rlen+=charger_I(&sh,quota_cmd,*ent);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,device,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;
```

```
        ent = (int *) pdf;
        rlen+=charger_I(&sh,quota,*ent);
        ih=(int *) tempo;
        *ih=rlen;
        break;


case AUR_QUOTA:


        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

      rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        ent = (int *) pdf;
        rlen+=charger_I(&sh,quota_cmd,*ent);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,device,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        ent = (int *) pdf;
        rlen+=charger_I(&sh,quota,*ent);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,quota_arg,&pdf,*sz);
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_READLINK:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }
```

```
                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

               rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,link_name,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,buff_link,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                pci = (char *) malloc(sizeof(int));
                copy_bytes(pdf, pci, sizeof(int));
                rlen+= charger_db_I(&sh,pdf,count_buff);
                ih=(int *) tempo;
                *ih=rlen;
                free(pci);
                break;

        case AUR_REBOOT:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                ent=(int *) pdf;
                rlen+=charger_I(&sh,arg_reb,*ent);
                ih=(int *) tempo;
                *ih=rlen;
                break;


        case AUR_REBOOTFAIL:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
```

```
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);


        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,arg_reb,*ent);
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_RENAME:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

       rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,from_fname,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,to_fname,&pdf,*sz);
        ih=(int *) tempo;
        *ih=rlen;
        break;


case AUR_RMDIR:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
```

```
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

       rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,name_dir,&pdf,*sz);
        ih=(int *) tempo;
        *ih=rlen;
        break;


case AUR_SEMCTL:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,sem_id,*ent);
        ih=(int *) tempo;
        *ih=rlen;
        break;


case AUR_SEMGET:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

          sz=(short *) buffAddr;
          pdf=buffAddr+sizeof(short)*temp;
          rlen=rlen+temp*2*sizeof(short);
```

```
                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                pci = (char *) malloc(sizeof(int));
                copy_bytes(pdf, pci, sizeof(int));
                rlen+= charger_db_I(&sh,pdf,sem_id);
                pdf=pdf+*sz;
                sz++;

                pci = (char *) malloc(sizeof(int));
                copy_bytes(pdf, pci, sizeof(int));
                rlen+= charger_db_I(&sh,pdf,sem_key);
                ih=(int *) tempo;
                *ih=rlen;
                free(pci);
                break;

        case AUR_SEMOP:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                ent=(int *) pdf;
                rlen+=charger_I(&sh,sem_id,*ent);
                ih=(int *) tempo;
                *ih=rlen;
                break;

        case AUR_SETDOMAINNAME:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,old_dname,&pdf,*sz);
                pdf=pdf+*sz;
```

```
                sz++;

                rlen+=charger_db_S(&sh,new_dname,&pdf,*sz);
                ih=(int *) tempo;
                *ih=rlen;
                break;


        case AUR_SETHOSTNAME:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,old_hname,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,new_hname,&pdf,*sz);
                ih=(int *) tempo;
                *ih=rlen;
                break;

        case AUR_SETTIMEOFDAY:


                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                pcl = (char *) malloc(2*sizeof(long));
                copy_bytes(pdf, pcl,2*sizeof(long));
                ptimeval=(au_timeval *) pcl;
                rlen+=charger_L(&sh,second,ptimeval->tv_sec);
                rlen+=charger_L(&sh,msecond,ptimeval->tv_usec);
                pdf =pdf+*sz;

                pci = (char *) malloc(2*sizeof(int));
                copy_bytes(pdf, pci,2*sizeof(int));
```

```
        ptimezone=(au_timezone *) pci;
rlen+=charger_I(&sh,minute,ptimezone->tz_minuteswest );
rlen+=charger_I(&sh,type_dest,ptimezone->tz_dsttime );
        ih=(int *) tempo;
        *ih=rlen+8;
        free(pci);
        free(pcl);
        break;

case AUR_SHMAT:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,shm_id,*ent );
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_SHMCTL:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,shm_id,*ent );
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_SHMDT:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
```

```
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,shm_id,*ent );
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_SHMGET:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+3*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,shm_seg,*ent );
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,shm_key,*ent );
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_SOCKET:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;
```

```
        ent=(int *) pdf;
        rlen+=charger_I(&sh,adressf,*ent);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,socket_type,*ent);
        pdf=pdf+*sz;
        sz++;

        ent=(int *) pdf;
        rlen+=charger_I(&sh,protocols,*ent );
        ih=(int *) tempo;
        *ih=rlen;
        break;


case AUR_STAT:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

       rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

        rlen+=charger_db_S(&sh,file_name,&pdf,*sz);
        ih=(int *) tempo;
        *ih=rlen;
        break;

case AUR_STATFS:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);
```

```
            rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;

            rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;

            rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;

            rlen+=charger_db_S(&sh,name_dir,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;


            pcl = (char *) malloc(16*sizeof(long));
            copy_bytes(pdf,pcl,16*sizeof(long));
            pstatfs = (db_stat *) pcl;
            rlen+=charger_L(&sh,type_info,pstatfs->f_type);
            rlen+=charger_L(&sh,fsB_S,pstatfs->f_bsize);
           rlen+=charger_L(&sh,totB_fs,pstatfs->f_blocks);
            rlen+=charger_L(&sh,free_Bfs,pstatfs->f_bfree);
           rlen+=charger_L(&sh,free_Bnsu,pstatfs->f_bavail);
            rlen+=charger_L(&sh,totf_nfs,pstatfs->f_files);
            rlen+=charger_L(&sh,free_fnfs,pstatfs->f_ffree);
     rlen+=charger_L(&sh,first_pfs,pstatfs->f_fsid.val[0]);
    rlen+=charger_L(&sh,second_pfs,pstatfs->f_fsid.val[1]);
          rlen+=charger_L(&sh,spare_1,pstatfs->f_spare[0]);
          rlen+=charger_L(&sh,spare_2,pstatfs->f_spare[1]);
          rlen+=charger_L(&sh,spare_3,pstatfs->f_spare[2]);
          rlen+=charger_L(&sh,spare_4,pstatfs->f_spare[3]);
          rlen+=charger_L(&sh,spare_5,pstatfs->f_spare[4]);
          rlen+=charger_L(&sh,spare_6,pstatfs->f_spare[5]);
          rlen+=charger_L(&sh,spare_7,pstatfs->f_spare[6]);

            free(pcl);
            ih=(int *) tempo;
            *ih=rlen+60;
            break;

    case AUR_SYMLINK:

            nb_read=read(fd, buffAddr, dbsz);
            if (nb_read<0)
                    {
                    printf("read error\n");
                    exit(1);
                    }

            sz=(short *) buffAddr;
            pdf=buffAddr+sizeof(short)*temp;
            rlen=rlen+temp*2*sizeof(short);

            rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;

            rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
            pdf=pdf+*sz;
```

```
            sz++;

           rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;

            rlen+=charger_db_S(&sh,nf_link,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;

            rlen+=charger_db_S(&sh,link_name,&pdf,*sz);
            ih=(int *) tempo;
            *ih=rlen;
            break;

    case AUR_SYSACCT:

            nb_read=read(fd, buffAddr, dbsz);
            if (nb_read<0)
                    {
                    printf("read error\n");
                    exit(1);
                    }

            sz=(short *) buffAddr;
            pdf=buffAddr+sizeof(short)*temp;
            rlen=rlen+temp*2*sizeof(short);

            rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;

         rlen+=charger_db_S(&sh,account_fname,&pdf,*sz);
            ih=(int *) tempo;
            *ih=rlen;
            break;

    case AUR_TRUNCATE:

            nb_read=read(fd, buffAddr, dbsz);
            if (nb_read<0)
                    {
                    printf("read error\n");
                    exit(1);
                    }

            sz=(short *) buffAddr;
            pdf=buffAddr+sizeof(short)*temp;
            rlen=rlen+temp*2*sizeof(short);

            rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;

            rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;

           rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
            pdf=pdf+*sz;
            sz++;
```

```
                rlen+=charger_db_S(&sh,file_name,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                pcl=(char *) malloc(sizeof(long));
                copy_bytes(pdf,pcl,sizeof(long));
                lg=(long *) pcl;
                rlen+=charger_db_L(&sh,pdf,l_trun_to,*lg);
                ih=(int *) tempo;
                *ih=rlen;
                free(pcl);
                break;

        case AUR_UNLINK:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

                rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;

               rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
               pdf=pdf+*sz;
               sz++;

               rlen+=charger_db_S(&sh,file_name,&pdf,*sz);
               ih=(int *) tempo;
               *ih=rlen;
               break;

        case AUR_UNMOUNT:

                nb_read=read(fd, buffAddr, dbsz);
                if (nb_read<0)
                        {
                        printf("read error\n");
                        exit(1);
                        }

                sz=(short *) buffAddr;
                pdf=buffAddr+sizeof(short)*temp;
                rlen=rlen+temp*2*sizeof(short);

                rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
                pdf=pdf+*sz;
                sz++;
```

```
        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

       rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
       pdf=pdf+*sz;
       sz++;

      rlen+=charger_db_S(&sh,loc_mount_dir,&pdf,*sz);
       ih=(int *) tempo;
       *ih=rlen;
       break;

case AUR_UTIMES:

        nb_read=read(fd, buffAddr, dbsz);
        if (nb_read<0)
                {
                printf("read error\n");
                exit(1);
                }

        sz=(short *) buffAddr;
        pdf=buffAddr+sizeof(short)*temp;
        rlen=rlen+temp*2*sizeof(short);

        rlen+=charger_db_S(&sh,vide_value,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;


        rlen+=charger_db_S(&sh,cur_root,&pdf,*sz);
        pdf=pdf+*sz;
        sz++;

       rlen+=charger_db_S(&sh,curw_directory,&pdf,*sz);
       pdf=pdf+*sz;
        sz++;

       rlen+=charger_db_S(&sh,fname_time,&pdf,*sz);
       pdf=pdf+*sz;
        sz++;

        pcl = (char *) malloc(sizeof(long));
        copy_bytes(pdf, pcl,sizeof(long));
        pdf=pcl;
        lg =(long *) pdf;
rlen+=charger_db_L(&sh,pdf,sec_last_access,*lg);
        free(pcl);
        pdf=pdf+*sz;
        sz++;


        pcl = (char *) malloc(sizeof(long));
        copy_bytes(pdf, pcl, sizeof(long));
        pdf=pcl;
        lg=(long *) pdf;
      rlen+=charger_db_L(&sh,pdf,ms_last_access,*lg);
        free(pcl);
        pdf=pdf+*sz;
        sz++;
```

```
                        pcl = (char *) malloc(sizeof(long));
                        copy_bytes(pdf, pcl, sizeof(long));
                        pdf=pcl;
                        lg=(long *) pdf;
                      rlen+=charger_db_L(&sh,pdf,sec_last_modif,*lg);
                        free(pcl);
                        pdf=pdf+*sz;
                        sz++;

                        pcl = (char *) malloc(sizeof(long));
                        copy_bytes(pdf, pcl, sizeof(long));
                        pdf=pcl;
                        lg=(long *) pdf;
                       rlen+=charger_db_L(&sh,pdf,ms_last_modif,*lg);
                        ih=(int *) tempo;
                        *ih=rlen;
                        free(pcl);
                        break;


                case AUR_TEXT:


                        nb_read=read(fd, buffAddr, dbsz);
                        if (nb_read<0)
                                {
                                printf("read error\n");
                                exit(1);
                                }

                        sz=(short *) buffAddr;
                        pdf=buffAddr+sizeof(short)*temp;
                        rlen=rlen+temp*2*sizeof(short);

                        i=0;
                        while(i++<temp)
                                {
                                rlen+=charger_db_S(&sh,i+25,&pdf,*sz);
                                pdf=pdf+*sz;
                                sz++;
                                }
                        ih=(int *) tempo;
                        *ih=rlen;
                        break;

                default :
                        ih=(int *) tempo;
                        *ih=rlen;
                        nb_read = read(fd, buffAddr, dbsz);
                        if (nb_read<0)
                                {
                                  printf("read error\n");
                                exit(1);
                                }

        } /* end of switch */
if (rc=write_NADF(fd1,tempo,FLUSH)<0)
        {
        printf("wirte error\n");
```

```
                exit(1);
                }

        }
    }
}

/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES:   /                                                   */
/* PRECONDITION:                                                             */
/*              - pdf est l'adresse d'un entier                              */
/*              - id est un entier court non signe                           */
/*              - sh est un pointeur vers l'adresse d'une zone alignee sur   */
/*                un demi-mot                                                */
/*              - soit sh0 la valeur de (*sh)                                */
/*                                                                           */
/* POSTCONDITION:                                                            */
/*              - Soit lg = la taille en byte d'un entier;                   */
/*                      v = la valeur pointee par pdf                        */
/*              - alors sh0 est l'adresse de debut de trois zones contigues: */
/*                   - la premiere de 2 bytes contenant la valeur id         */
/*                   - la seconde de 2 bytes contenant la valeur lg          */
/*                   - la troisieme de 'lg' bytes contenant la valeur v      */
/*              - *sh est l'adresse du byte qui suit immediatement la 3-ieme */
/*                zone                                                       */
/*              - la valeur retournee est 'lg'                               */
/*                                                                           */
/*****************************************************************************/

charger_db_I(sh, pdf, id)
short  **sh, id;
char *pdf;
        {
        int  rlen=0;
        char *pc;
        short shaux;

        **sh=id;
        (*sh)++;
        shaux=**sh=sizeof(int);
        rlen=rlen+**sh;
        (*sh)++;
        pc= (char *) *sh;
        copy_bytes(pdf, pc, sizeof(int));
        *sh += sizeof(int);
        rlen += align(shaux, sh);
        return (rlen);
        }

/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES:   /                                                   */
/* PRECONDITION:                                                             */
/*              - val est un entier                                          */
/*              - id est un entier court non signe                           */
/*              - sh est un pointeur vers l'adresse d'une zone alignee sur   */
/*                un demi-mot                                                */
/*              - soit sh0 la valeur de (*sh)                                */
/*                                                                           */
/* POSTCONDITION:                                                            */
```

```
/*              - Soit lg = la taille en byte d'un entier;                    */
/*              - alors sh0 est l'adresse de debut de trois zones contigues:  */
/*                   - la premiere de 2 bytes contenant la valeur id          */
/*                   - la seconde de 2 bytes contenant la valeur lg           */
/*                   - la troisieme de 'lg' bytes contenant la valeur val     */
/*              - *sh est l'adresse du byte qui suit immediatement la 3-ieme  */
/*                zone                                                        */
/*              - la valeur retournee est 'lg'                                */
/*                                                                            */
/******************************************************************************/


charger_I(sh, id, val)
short  **sh, id;
int  val;
        {
        int  rlen=0,*ih;
        int shaux;


        **sh=id;
        (*sh)++;
        shaux=**sh=sizeof(int);
        rlen=rlen+**sh;
        (*sh)++;
         ih=(int *) *sh;
        *ih=val;
        ih++;
        *sh=(short *) ih;
        rlen += align(shaux, sh);
        return (rlen);
        }

/******************************************************************************/
/*                                                                            */
/* VARIABLES GLOBALES:   /                                                    */
/* PRECONDITION:                                                              */
/*              - val est un entier court non signe                           */
/*              - id est un entier court non signe                            */
/*              - sh est un pointeur vers l'adresse d'une zone alignee sur    */
/*                un demi-mot                                                 */
/*              - soit sh0 la valeur de (*sh)                                 */
/*                                                                            */
/* POSTCONDITION:                                                             */
/*              - Soit lg = la taille en byte d'un entier;                    */
/*              - alors sh0 est l'adresse de debut de trois zones contigues:  */
/*                   - la premiere de 2 bytes contenant la valeur id          */
/*                   - la seconde de 2 bytes contenant la valeur lg           */
/*                   - la troisieme de 'lg' bytes contenant la valeur val     */
/*              - *sh est l'adresse du byte qui suit immediatement la 3-ieme  */
/*                zone                                                        */
/*              - la valeur retournee est 'lg'                                */
/*                                                                            */
/******************************************************************************/


charger_UI(sh, id, val)
short  **sh, id;
unsigned int  val;
        {
        int  rlen=0;
```

```
        unsigned int *uih;
        int shaux;


        **sh=id;
        (*sh)++;
        shaux=**sh=sizeof(int);
        rlen=rlen+**sh;
        (*sh)++;
        uih=(unsigned int *) *sh;
        *uih=val;
        uih++;
        *sh=(short *) uih;
        rlen += align(shaux, sh);
        return (rlen);
        }

/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES:   /                                                   */
/* PRECONDITION:                                                             */
/*              - pdf est l'adresse d'un entier long                         */
/*              - id est un entier court non signe                           */
/*              - sh est un pointeur vers l'adresse d'une zone alignee sur   */
/*                un demi-mot                                                */
/*              - soit sh0 la valeur de (*sh)                                */
/*                                                                           */
/* POSTCONDITION:                                                            */
/*              - Soit lg = la taille en byte d'un entier;                   */
/*                      v = la valeur pointee par pdf;                       */
/*              - alors sh0 est l'adresse de debut de trois zones contigues: */
/*                   - la premiere de 2 bytes contenant la valeur id         */
/*                   - la seconde de 2 bytes contenant la valeur lg          */
/*                   - la troisieme de 'lg' bytes contenant la valeur v      */
/*              - *sh est l'adresse du byte qui suit immediatement la 3-ieme */
/*                zone                                                       */
/*              - la valeur retournee est 'lg'                               */
/*                                                                           */
/*****************************************************************************/

charger_db_L(sh,pdf, id, val)
short  **sh, id;
long  val;
char *pdf;
        {
        int  rlen=0;
        long *lh;
        char *pc;
        int shaux;


        **sh=id;
        (*sh)++;
        shaux=**sh=sizeof(long);
        rlen=rlen+**sh;
        (*sh)++;
        pc= (char *) *sh;
        copy_bytes(pdf, pc, sizeof(long));
```

```
        *sh+=sizeof(long);
        rlen += align(shaux, sh);
        return (rlen);
        }

/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES:   /                                                   */
/* PRECONDITION:                                                             */
/*              - val est un entier long                                     */
/*              - id est un entier court non signe                           */
/*              - sh est un pointeur vers l'adresse d'une zone alignee sur   */
/*                un demi-mot                                                */
/*              - soit sh0 la valeur de (*sh)                                */
/*                                                                           */
/* POSTCONDITION:                                                            */
/*              - Soit lg = la taille en byte d'un entier long;              */
/*              - alors sh0 est l'adresse de debut de trois zones contigues: */
/*                   - la premiere de 2 bytes contenant la valeur id         */
/*                   - la seconde de 2 bytes contenant la valeur lg          */
/*                   - la troisieme de 'lg' bytes contenant la valeur val    */
/*              - *sh est l'adresse du byte qui suit immediatement la 3-ieme */
/*                zone                                                       */
/*              - la valeur retournee est 'lg'                               */
/*                                                                           */
/*****************************************************************************/


charger_L(sh, id, val)
short  **sh, id;
long  val;
        {
        int  rlen=0;
        long *lh;
        int shaux;



        **sh=id;
        (*sh)++;
        shaux=**sh=sizeof(long);
        rlen=rlen+**sh;
        (*sh)++;
        lh=(long *) *sh;
        *lh=val;
        lh++;
        *sh=(short *) lh;
        rlen += align(shaux, sh);
        return (rlen);
        }


/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES:   /                                                   */
/* PRECONDITION:                                                             */
/*              - val est un entier court                                    */
/*              - id est un entier court non signe                           */
/*              - sh est un pointeur vers l'adresse d'une zone alignee sur   */
/*                un demi-mot                                                */
/*              - soit sh0 la valeur de (*sh)                                */
```

```
/*                                                                          */
/* POSTCONDITION:                                                           */
/*              - Soit lg = la taille en byte d'un entier court;            */
/*              - alors sh0 est l'adresse de debut de trois zones contigues: */
/*                   - la premiere de 2 bytes contenant la valeur id        */
/*                   - la seconde de 2 bytes contenant la valeur lg         */
/*                   - la troisieme de 'lg' bytes contenant la valeur val   */
/*              - *sh est l'adresse du byte qui suit la 3-ieme zone et qui  */
/*                soit aligne sur un demi-mot                               */
/*              - la valeur retournee est 'lg'                              */
/*                                                                          */
/**************************************************************************/


charger_SH(sh, id, val)
short  **sh,id;
short val;
        {
        int rlen=0;
        int shaux;



        **sh=id;
        (*sh)++;
        shaux=**sh=sizeof(short);
        rlen=rlen+**sh;
        (*sh)++;
        **sh=val;
        (*sh)++;
        rlen += align(shaux, sh);
        return(rlen);
        }

/**************************************************************************/
/*                                                                        */
/* VARIABLES GLOBALES:   /                                                */
/* PRECONDITION:                                                          */
/*              - val est un entier court non signe                       */
/*              - id est un entier court non signe                        */
/*              - sh est un pointeur vers l'adresse d'une zone alignee sur */
/*                un demi-mot                                             */
/*              - soit sh0 la valeur de (*sh)                             */
/*                                                                        */
/* POSTCONDITION:                                                         */
/*              - Soit lg = la taille en byte d'un entier court non signe; */
/*              - alors sh0 est l'adresse de debut de trois zones contigues: */
/*                   - la premiere de 2 bytes contenant la valeur id      */
/*                   - la seconde de 2 bytes contenant la valeur lg       */
/*                   - la troisieme de 'lg' bytes contenant la valeur val */
/*              - *sh est l'adresse du byte qui suit la 3-ieme zone et qui */
/*                soit aligne sur un demi-mot                             */
/*              - la valeur retournee est 'lg'                            */
/*                                                                        */
/**************************************************************************/


charger_USH(sh, id, val)
short  **sh,id;
unsigned short val;
        {
```

```
        int rlen=0;
        unsigned short *ush;
        int shaux;



        **sh=id;
        (*sh)++;
        shaux=**sh=sizeof(unsigned short);
        rlen=rlen+**sh;
        (*sh)++;
        ush=(unsigned short *) *sh;
        *ush=val;
        ush++;
        *sh=(short *) ush;
        rlen += align(shaux, sh);
        return(rlen);
        }

/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES:   /                                                   */
/* PRECONDITION:                                                             */
/*              - val1 est l'adresse d'une zone de longueur val2 bytes       */
/*              - id est un entier court non signe                           */
/*              - sh est un pointeur vers l'adresse d'une zone alignee sur   */
/*                un demi-mot                                                */
/*              - soit sh0 la valeur de (*sh)                                */
/*                                                                           */
/* POSTCONDITION:                                                            */
/*              - Soit lgr le plus petit multiple de ALIGN_REC superieur ou  */
/*                egal a val                                                 */
/*              - alors sh0 est l'adresse de debut de trois zones contigues: */
/*                   - la premiere de 2 bytes contenant la valeur id         */
/*                   - la seconde de 2 bytes contenant la valeur lgr         */
/*                   - la troisieme de 'lgr' bytes dont les val2 premiers    */
/*                     sont ceux pointes par val1; le reste etant des blancs */
/*              - *sh est l'adresse du byte qui suit immediatement la 3-ieme */
/*                zone                                                       */
/*              - la valeur retournee est lgr                                */
/*                                                                           */
/*****************************************************************************/

int  charger_STR(sh, id, val1, val2)
short  **sh,id;
char *val1;
short val2;
        {
        char *ch;
        int rlen=0;
        int shaux;

        **sh=id;
        (*sh)++;
        shaux=**sh=val2;
        rlen=rlen+**sh;
        (*sh)++;
        ch=(char *) *sh;
        copy_bytes(val1, ch, val2);
        ch=ch+val2;
        *sh=(short *) ch;
```

```
        rlen += align(shaux, sh);
        return(rlen);
        }
/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES:   /                                                   */
/* PRECONDITION:                                                             */
/*              - val est une valeur entiere                                 */
/*              - pdf est un pointeur vers l'adresse d'une zone de val bytes */
/*              - id est un entier court non signe                           */
/*              - sh est un pointeur vers l'adresse d'une zone alignee sur   */
/*                un demi-mot                                                */
/*              - soit sh0 la valeur de (*sh)                                */
/*                                                                           */
/* POSTCONDITION:                                                            */
/*              - Soit lgr le plus petit multiple de ALIGN_REC superieur ou  */
/*                egal a val                                                 */
/*              - alors sh0 est l'adresse de debut de trois zones contigues: */
/*                   - la premiere de 2 bytes contenant la valeur id         */
/*                   - la seconde de 2 bytes contenant la valeur lgr         */
/*                   - la troisieme de 'lgr' bytes dont les val premiers     */
/*                     sont ceux pointes par *pdf; le reste etant des blancs */
/*              - *sh est l'adresse du byte qui suit la 3-ieme zone et qui   */
/*                soit aligne sur un demi-mot                                */
/*              - la valeur retournee est 'lgr'                              */
/*                                                                           */
/*****************************************************************************/


int  charger_db_S(sh, id, pdf,val)
short  **sh, id;
char **pdf;
short val;
        {
        char *ch;
        int rlen=0;
        int shaux;

        **sh=id;
        (*sh)++;
        shaux=**sh=val;
        rlen=rlen+(**sh);
        (*sh)++;
        ch=(char *) *sh;
        copy(*pdf,ch,val);
        ch=ch+val;
        *sh=(short *) ch;
        rlen += align(shaux, sh);
        return(rlen);
        }
/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES:   /                                                   */
/* PRECONDITION:                                                             */
/*              - shaux est une valeur entiere                               */
/*              - sh est un pointeur vers l'adresse d'une zone  de bytes     */
/*              - soit sh0 la valeur de (*sh)                                */
/*                                                                           */
/* POSTCONDITION:                                                            */
/*              - Soit al = shaux*(ALIGN_REC - 1))%ALIGN_REC;                */
/*              - *sh pointe 'al' bytes au dela de sh0;                      */
```

```
/*              - ces bytes sont egaux au caractere blanc                    */
/*              - la valeur retournee est al                                 */
/*                                                                           */
/*****************************************************************************/


align(shaux, sh)
int shaux;
short **sh;
{
        int i;
        char *ch;

        ch = (char *) (*sh);
        for (i=0; i<((shaux*(ALIGN_REC - 1))%ALIGN_REC); i++) {
                *ch= 32;
                ch++;
        }
        *sh = (short *) ch;
        return(i);
}


/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES:   /                                                   */
/* PRECONDITION:                                                             */
/*              - fd est le file descriptor d'un fichier NADF ouvert avec    */
/*                "creat_NADF";                                              */
/*              - rec_add adresse du record NADF a ecrire                    */
/*              - flush un flag indicant si le buffer doit etre ecrit sur le */
/*                disk                                                       */
/*                                                                           */
/* POSTCONDITION:                                                            */
/*              - Si l'operation d'ecriture s'est bien deroulee, la valeur   */
/*                reournee est >= 0 sinon une valeur negative est retournee  */
/*                pour signaler le type de l'erreur                          */
/*                                                                           */
/*****************************************************************************/


write_NADF(fd, rec_add, flush)
int fd;
char *rec_add;
short flush;
    {
    int  i;         /* loop counter */
    int  nb;        /* number of bytes remaining in the buffer */
    int  nb_to_wrt; /* number of bytes to be written */
    int  nb_trans;  /* number of bytes written effectively */
    int  *rec_len;  /* length of the record specified in the first field */
    int  len_rec;   /* length of the record specified in the first field */
    char *rec_ptr;  /* address of the current position in the record */

    if ((fd > -1) && (fd < MAX_FILES))   /* fd is in the valid range */
        {

        if (open_mode[fd] == WRITE)
            {
            if (rec_add == NULL)
```

```
            {
            return(ESNORECORD);  /* no record to be written */
            }
        rec_ptr = rec_add;     /* current position = beginning */
        rec_len = (int *) rec_add;
        len_rec = *rec_len;
        /* To treat the records with a length > free space in buffer */
        while (len_rec > (buff_end[fd] - buff_ptr[fd] - flush_len[fd]))
          {
          nb = buff_end[fd] - buff_ptr[fd];
          copy_bytes(rec_ptr, buff_ptr[fd], nb);
          nb_trans = write(fd, buff_add[fd], buff_len[fd]);
          if (nb_trans != buff_len[fd])
              {
              return(ESBADWRITE);   /* Writing error */
              }
          buff_ptr[fd] = buff_add[fd];
          rec_ptr = rec_ptr + nb;
          len_rec = len_rec - nb;
          flush_len[fd] = 0;
          }

        /* CASE length of the record <= free space in the buffer */
        copy_bytes(rec_ptr, buff_ptr[fd], len_rec);
        buff_ptr[fd] = buff_ptr[fd] + len_rec;

        /* alignment */
        for(i=0; i<((len_rec*(ALIGN_REC-1))%ALIGN_REC); i++)
            {
            *buff_ptr[fd]='\0';
            buff_ptr[fd]++;
            }


        /* CASE flush required */
        if (flush == FLUSH)
            {
            /* flush */
            nb_to_wrt = buff_ptr[fd]-buff_add[fd];
            nb_trans  = write(fd, buff_add[fd], nb_to_wrt);

            if (nb_trans != nb_to_wrt)
                {
                return(ESBADWRITE);   /* Writing error */
                }
            flush_len[fd] = (buff_ptr[fd]-buff_add[fd]) % BUFSIZ;
                        /* the '%' is the modulo operator */
            buff_ptr[fd] = buff_add[fd];

            }
        return(0);        /* success */
        }
    if (open_mode[fd] == READ)      /* fd is a file descriptor of    */
                                    /* a file open with 'open_NADF'  */
        {
        return(ESBADMODE);           /* error : the file is not open in */
                                     /*         the 'write' mode        */
        }
    }
return(ESBADFID);            /* the file descriptor is invalid */
}
```

```
/*****************************************************************************/
/*                                                                           */
/* VARIABLE GLOBALE:   \                                                     */
/* PRECONDITION:                                                             */
/*               - from est un pointeur vers une zone de 'n' bytes           */
/*               - to est un pointeur vers une zone alouee d'au moins n bytes */
/*                                                                           */
/* POSTCONDITION:                                                            */
/*               - 'n' bytes sont copies (dans l'ordre) a partir de l'adresse */
/*                 from vers l'adresse 'to'                                  */
/*                                                                           */
/*****************************************************************************/

copy_bytes(from, to, n)
char *from, *to;
int n;
     {
     while(n>0)
         {
         *to++ = *from++;
         n--;
         }
     }

/*****************************************************************************/
/*                                                                           */
/* VARIABLE GLOBALE:   \                                                     */
/* PRECONDITION:                                                             */
/*               - from est un pointeur vers une zone de 'n' bytes           */
/*               - to est un pointeur vers une zone alouee d'au moins n bytes */
/*                                                                           */
/* POSTCONDITION:                                                            */
/*               - posons p = le nombre de bytes avant le premier caractere  */
/*                 '\0'  dans la zone pointee par 'from'  (p <= n)           */
/*               - 'p' bytes sont copies (dans l'ordre) a partir de l'adresse */
/*                 from vers l'adresse 'to'                                  */
/*               - (n-p) caracteres blancs sont copies a partir de l'adresse */
/*                 from + p vers l'adresse  (to + p)                         */
/*                                                                           */
/*****************************************************************************/


copy(from, to, n)
char *from, *to;
int n;
     {
     while((n>0) && (*from!='\0'))
        {
        *to++ = *from++;
        n--;
        }
    while(n>0)
        {
        *to++=32;
        n--;
        }
}
class_event(sh, ent)
  short **sh;
 unsigned int ent;
```

```
{
        int rlen = 0;
switch (ent)
        {
        case 1 :
                rlen+=charger_STR(sh,event,event_class[1].type_name,32);
                break;
        case 2 : rlen+=charger_STR(sh,event,event_class[2].type_name,32);
                break;
        case 4 : rlen+=charger_STR(sh,event,event_class[3].type_name,32);
                break;
        case 8 : rlen+=charger_STR(sh,event,event_class[4].type_name,32);
                break;
        case 16 : rlen+=charger_STR(sh,event,event_class[5].type_name,32);
                break;
        case 32 : rlen+=charger_STR(sh,event,event_class[6].type_name,32);
                        break;
        case 64 : rlen+=charger_STR(sh,event,event_class[7].type_name,32);
                break;
        case 128 : rlen+=charger_STR(sh,event,event_class[8].type_name,32);
                break;
        case 256 : rlen+=charger_STR(sh,event,event_class[9].type_name,32);
                break;
        case 512 : rlen+=charger_STR(sh,event,event_class[10].type_name,32);
                break;
        case 1024 : rlen+=charger_STR(sh,event,event_class[11].type_name,32);
                break;
        case 10 : rlen+=charger_STR(sh,event,event_class[12].type_name,32);
                break;
        case 3 : rlen+=charger_STR(sh,event,event_class[13].type_name,32);
                break;
        case 11 : rlen+=charger_STR(sh,event,event_class[14].type_name,32);
                break;
        case 258 : rlen+=charger_STR(sh,event,event_class[15].type_name,32);
                break;
        }
        return(rlen);
}

/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES:   /                                                   */
/* PRECONDITION:                                                             */
/*              - fd est le file descriptor d'un fichier NADF ouvert avec    */
/*                "creat_NADF";                                              */
/*                                                                           */
/* POSTCONDITION:                                                            */
/*              - Si l'operation de fermeture s'est bien deroulee, la valeur */
/*                retournee est >= 0 sinon une valeur negative est retournee */
/*                pour signaler le type de l'erreur                          */
/*                                                                           */
/*****************************************************************************/


close_NADF(fd)
int fd;
    {
    int  nb;        /* number of bytes to be written */
```

```
    int  nb_trans;  /* number of bytes written effectively */

    if ((fd >-1) && (fd < MAX_FILES))  /* if the file descriptor is valid */
        {
        if (open_mode[fd] == WRITE)   /* if the open mode is 'WRITE' */
            {
            /* flush */
            nb = buff_ptr[fd]-buff_add[fd];
            nb_trans = write(fd, buff_add[fd], nb);

            if (nb_trans != nb)
                {
                return(ESBADWRITE);   /* Writing error */
                }
            free(buff_add[fd]);      /* release the memory area */
            close(fd);
            open_mode[fd] = NONE;
            return(0);               /* success */
            }
        else
            {
            if (open_mode[fd] == READ)  /* if the open mode is 'READ' */
                {
                free(buff_add[fd]);   /* release the memory area */
                close(fd);
                open_mode[fd] = NONE;
                return(0);            /* success */
                }
            }
        }
    open_mode[fd] = NONE;
    return(ESBADFID); /* error: invalid 'fd' value */
    }

/*****************************************************************************/
/*                                                                           */
/* VARIABLES GLOBALES:   /                                                   */
/* PRECONDITION:                                                             */
/*              - fd est le file descriptor d'un fichier NADF ouvert avec    */
/*                "OPEN_AUDIT";                                              */
/*                                                                           */
/* POSTCONDITION:                                                            */
/*              - Si l'operation de fermeture s'est bien deroulee, la valeur */
/*                retournee est 0, le buffer est libere. Sinon une valeur    */
/*                negative est retourne, pour signaler le type de l'erreur   */
/*                                                                           */
/*****************************************************************************/


CLOSE_AUDIT(fd)
int fd;
    {
    if (fd >-1)  /* if the file descriptor is valid */
        {
        free(buffAddr);   /* release the memory area */
        close(fd);
        open_mode[fd] = NONE;
        return(0);            /* success */
        }
    open_mode[fd] = NONE;
    return(ESBADFID);        /* error: invalid 'fd' value */
```

```
}
```