Rapport de Stage

Présenté pour obtenir le diplôme de **DEA**

2001/2002

FILIERE

Informatique

Option

Télécommunications, Réseaux et Services

Détection d'Intrusion :

Etat de l'Art, Critiques et Contributions

Jury :

M. Dominique Méry

M. Didier Galmiche

M^{me} Noëlle Carbonell

Rapporteur :

M. Olivier Festor

Encadrant :

M. Sylvain Gombault

Yacine BOUZIDA

Remerciements

Les travaux ayant fait l'objet de ce stage ont été effectués au sein du département RSM (Réseaux, Services et Multimédia) à l'Ecole Nationale Supérieure des Télécommunications de Bretagne (ENSTB) Campus de Rennes, sous la direction attentionnée de Sylvain Gombault. Ses critiques, ses suggestions et ses encouragements m'ont été très précieux pour la réalisation de ce travail.

Je tiens à remercier le responsable du DEA au LORIA, Dominique Méry, pour m'avoir accepté de poursuivre le DEA bien que j'ai pris du retard au début de l'année universitaire.

Mes remerciements vont aussi aux lecteurs anonymes d'IEEE-RAID2002 (Recent Advances in Intrusion Detection 2002) pour leurs remarques sur les deux papiers que nous avons soumis à cette conférence.

Je remercie également P.Tadonki et J.M Cornilleau, ingénieurs chercheurs au département RSM, pour leur aide et leur disponibilité ainsi que le temps consacré par Patrice pour la lecture de ce rapport.

J'adresse mes vifs remerciements à Francis Dupont qui n'a ménagé ni son temps ni ses efforts à chaque sollicitation de ma part.

Je tiens aussi à remercier Madame Noëlle Carbonell et Messieurs Olivier Festor, Dominique Méry, Didier Galmiche d'avoir accepté d'examiner mon travail.

Table des matières

Liste des figures

Liste des tableaux

Introduction

Bien que beaucoup de progrès ont été faits dans le domaine de la sécurité informatique durant ces deux dernières décennies, il reste un énorme travail pour améliorer la sécurité dans les systèmes informatiques de nos jours.

En effet, des travaux significatifs ont été menés dans ce sens. Cependant, la réalité montre que tous les systèmes informatiques sont vulnérables. Ces systèmes sont vulnérables devant des attaques provenant des utilisateurs non autorisés (attaques externes) et attaques provenant des utilisateurs autorisés (attaques internes) qui abusent de leurs privilèges pour attaquer ou compromettre leurs propres systèmes.

L'infrastructure de la sécurité informatique produit plusieurs services de sécurité, ainsi que de la sûreté de fonctionnement, tels que la disponibilité, l'intégrité, la confidentialité et la fiabilité ;

- le fait d'être prêt à l'utilisation conduit à la **disponibilité** ;
- la continuité de service conduit à la **fiabilité** ;
- la non-occurrence de divulgations non-autorisées de l'information conduit à la **confidentialité** ;
- la non-occurrence d'altérations inappropriées de l'information conduit à l'**intégrité.**

C'est quoi alors l'intrusion ?

Une intrusion est définie par Heady et al. [1] comme "*un ensemble d'actions qui essaye de compromettre l'intégrité, la confidentialité, ou la disponibilité d'une ressource* ".
Le domaine de détection d'intrusion date d'une vingtaine d'années. Le papier le plus cité est celui de J.P Anderson [2] où il définit quatre groupes possibles d'attaquants : "*external penetrator*", "*Masquerador*", "*Misfeasor*", et "*Clandestine user*".

Cette classification est générale et est basée sur les groupes d'utilisateurs qui peuvent apparaître comme attaquants potentiels. Cependant, il existe deux types de détection d'intrusion : approche comportementale (Anomaly Intrusion Detection) et approche par scénario (misuse intrusion detection).

La première partie de ce rapport introduit l'état de l'art et les différentes approches de la détection d'intrusion ainsi qu'une brève présentation de quelques outils d'IDS (Intrusion Detection Systems).
La seconde partie est consacrée à notre contribution qui est l'amélioration d'un IDS basé sur les algorithmes génétiques et de l'autre part la possibilité d'utiliser une nouvelle technique de détection

d'intrusion basée sur la théorie de l'information qui est l'ACP (Analyse en Composantes Principales). Une application de cette méthode sur un fichier web log réel est aussi présentée.

Chapitre 1

Etat de l'art de la détection d'intrusion

Dans ce chapitre, nous présentons quelques notions de base dans le domaine de la sécurité informatique. Nous introduisons, par la suite, les deux grandes familles de détection d'intrusion qui consistent en l'approche comportementale et l'approche par scénario ainsi que quelques systèmes de détection d'intrusion développés dans chacune des deux approches.
La dernière partie, de ce chapitre, est consacrée aux critiques de l'un des IDS ainsi présentés.

1.1 Concepts de base

Nous présentons, dans cette section, les concepts de base et la terminologie utilisée dans le domaine de la sécurité informatique, en général, et dans le domaine de détection d'intrusion en particulier.

1.1.1 Sûreté de fonctionnement

La sûreté de fonctionnement d'un système informatique est la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre.

Le service délivré par un système est son comportement tel que perçu par son, ou ses utilisateurs.
Selon la, ou les applications auxquelles le système est destiné, l'accent peut être mis sur différentes facettes de la sûreté de fonctionnement, ce qui revient à dire que la sûreté de fonctionnement regroupe des propriétés différentes mais complémentaires, qui permettent de définir ses *attributs* :

La sécurité peut être alors vue comme étant l'état dans lequel "*l'information est valide, les infrastructures garantissent l'intégrité, la disponibilité et la confidentialité des données et il est possible de détecter des action malveillantes*". Malheureusement, cet état, dans l'absolu, est un idéal impossible à atteindre.

Bien que plusieurs recherches ont été menées et plusieurs systèmes ont vu le jour, le "*risque zéro*" n'existe pas. Nous ne pouvons agir que sur le niveau résiduel du risque : plus nous voulons le diminuer, plus il faut faire d'effort (et mettre des moyens).

1.1.2 Politique de sécurité

La politique de sécurité peut être définie comme la détermination du bon seuil de sécurité en fonction des risques et des ressources disponibles, et la manière de l'atteindre. Toute politique de sécurité a pour but d'éliminer ou de réduire les vulnérabilités de façon à atteindre le niveau de risque choisi en fixant des règles à appliquer, des mesures à prendre, des structures et l'organisation à mettre en place.

Le rôle de la politique de sécurité consiste alors à surveiller les moyens de protection pour contrôler leurs efficacités, détecter les attaques et les mauvaises configurations en enregistrant les accès aux services sensibles, répondre par des actions correctives et contrôler les défaillances par des alarmes.

1.1.3 Intrusion

Toute violation de la politique de sécurité d'un système informatique est vue comme une intrusion. Cependant, il faut bien faire une distinction entre une attaque et une intrusion. La première consiste en l'exécution d'un scénario qui a comme objectif de compromettre la politique de sécurité du système ciblé et l'intrusion est le résultat d'une attaque réussie.

La mise en place d'un IDS (Intrusion Detection System) demande la surveillance continue de ce qui se passe sur le système ou sur le réseau que l'on veut protéger. Cette surveillance est réalisée via certains mécanismes d'audit de sécurité.

1.1.4 Audit de sécurité

Les mécanismes d'audit génèrent des fichiers d'audit contenant toutes les opérations effectuées sur le système informatique. Les enregistrements d'audit doivent contenir toutes les informations utiles concernant ces opérations à savoir le type d'action, l'horodatage (quand l'action a été réalisée), l'identifiant de l'utilisateur ou du processus sujet de cette action etc.
Deux systèmes de collectes d'événements sont connus :

- Systèmes "*Host Based*" : les informations collectées proviennent des différents hôtes appartenant au réseau.
- Systèmes "*Network Based*" : Dans ce cas, les données d'audit sont collectées au niveau du trafic réseau.

1.1.5 Sources d'attaques

En général, il existe deux catégories d'attaques (intrus) : les attaquants internes qui sont des utilisateurs légitimes qui abusent de leurs privilèges pour contourner la politique de sécurité et les attaquants externes qui sont des utilisateurs non autorisés à utiliser les moyens informatiques mis en place et qui ont pu gagner l'accès aux ressources du système et essayent de le compromettre.
J.P Anderson [2] a défini quatre groupes d'attaquants :

- *External penetrator* : c'est un intrus qui n'a pas le droit d'utiliser les ressources informatiques de l'organisation et qui a pu d'une manière ou d'une autre gagner l'accès à ces moyens informatiques.
- *maquerador* : c'est un intrus, interne ou externe, qui a pu avoir l'accès au système en utilisant l'authentification d'un autre utilisateur autorisé.
- *misfeasor* : c'est un utilisateur légitime qui a des privilèges d'accès à l'information et qui abuse de ses droits pour violer la politique de l'installation.
- *clandestine user* : ce type d'intrus est très difficile à détecter car il a pu accéder au système avec un très haut niveau de privilèges ce qui lui permet, par exemple, d'altérer l'état du système en supprimant les enregistrements d'audit des actions qu'il vient d'accomplir.

1.1.6 Approches de détection d'intrusion

Une présentation complète des différents systèmes et techniques développés dans le domaine de détection d'intrusion n'est pas un objectif principal de ce rapport, cependant, nous essayons dans ce chapitre, de décrire les approches en mentionnant des exemples de systèmes qui sont basés sur ces techniques.

Il existe deux grandes familles de détection d'intrusion : l'approche comportementale (Anomaly Intrusion Detection) et l'approche par scénario (Misuse Intrusion Detection).

Approche Comportementale (Anomaly Intrusion Detection)

Cette approche est basée sur le comportement d'utilisateur, on parle alors de profil utilisateur. Elle a été proposée par Anderson en 1980 et reprise par Denning [3] en 1987.
Anderson a proposé de décrire le profil utilisateur par un ensemble de mesures pertinentes modélisant au mieux son comportement afin de détecter par la suite toute déviation de son comportement habituel ainsi appris. Cette approche cherche alors à répondre à la question : "*le comportement actuel de l'utilisateur est il cohérent avec son comportement passé ?*"

Approche par scénario

Cette approche cherche à retrouver des attaques connues dans le fichier d'audit. Elle nécessite donc une connaissance, a priori, des attaques bien définies. Cette approche cherche alors à répondre à la question : "*le comportement actuel de l'utilisateur contient–il une attaque connue?*".
Dans ce cas, une construction d'une base de données d'attaques ou de scénarios d'attaques est nécessaire.

Une détection d'intrusion hybride qui combine ces deux techniques en même temps peut être ajoutée comme une troisième approche des IDS.

1.2 Approche Comportementale

Cette technique nécessite une phase d'apprentissage qui permet d'apprendre les comportements habituels des utilisateurs et de construire une base de connaissances qui contient les comportements normaux. Vient par la suite, la phase de détection qui à partir du fichier d'audit, un algorithme est mis en place pour vérifier que les utilisateurs audités correspondent bien à des comportements connus dans la base d'apprentissage.
Un algorithme de mise à jour de la base d'apprentissage est nécessaire pour prendre en considération l'évolution du comportement de l'utilisateur. Cependant, un utilisateur interne malicieux peut modifier lentement son comportement afin de parvenir à un comportement intrusif qui, ayant été progressivement appris, ne pourra pas être détecté.

1.2.1 Le modèle statistique de DENNING :

Ce modèle [3] contient six composantes :

- Sujets : ce sont les initiateurs des actions faites sur le système. Un sujet peut être un utilisateur mais peut aussi être un processus.
- Objets : ce sont les récepteurs d'actions et qui peuvent être des fichiers, des programmes …
- Enregistrements d'audit : ils sont générés par le système à chaque action entreprise par les sujets sur les objets. Ces enregistrements contiennent l'identificateur du sujet, l'action entreprise, l'objet, l'horodatage, etc.
- Les profils : ce sont des structures caractérisant le comportement des sujets envers les objets en terme de métriques statistiques observées sur le comportement de l'utilisateur.
- Les enregistrements d'anomalie : ces enregistrements sont générés dans le cas où un comportement anormal est détecté. Chaque enregistrement d'anomalie contient l'événement, l'horodatage et le profil.

- Les règles d'activité : une règle d'activité spécifie l'action à prendre quand un enregistrement d'anomalie est généré. Le meilleur exemple est le cas où le nombre de mots de passe erronés introduit dépasse un certain seuil.

Le but de l'approche statistique de Denning est de déterminer, à partir de n observations $x_1, x_2, \ldots, x_n$ d'une variable aléatoire x, si une nouvelle observation x_{n+1} est anormale par rapport aux n observations précédentes.

1.2.2 Hyperview—Réseaux de neurones pour la détection d'intrusion

Hyperview [4] est un IDS comprenant deux majeurs composants. Le premier est un système expert qui surveille le fichier d'audit pour détecter d'éventuelles intrusions connues à l'époque. Le second est un réseau de neurones qui apprend le comportement d'un utilisateur et déclenche une alarme lors de la déviation de l'audit du comportement ainsi appris.

L'architecture du réseau de neurones choisie

Au départ, les auteurs ont choisi l'architecture d'un réseau de neurones multicouche où les entrées sont les commandes introduites par l'utilisateur et la sortie fournit la prochaine commande prédite.

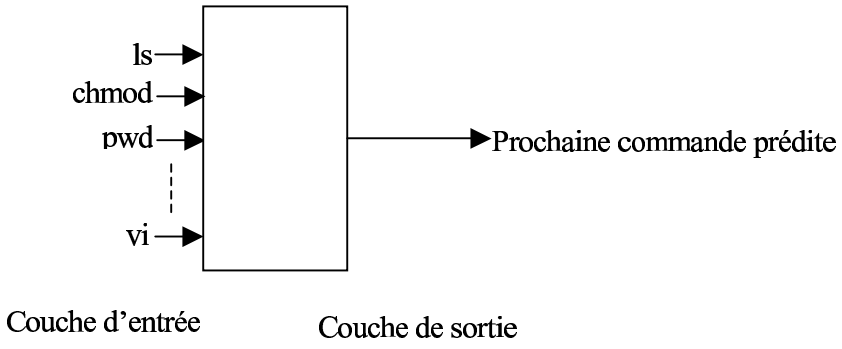

Figure 1.1.: Utilisation des reseaux de neurones dans la detection d'intrusion.

Sur une fenêtre de taille N, les $N-1$ éléments sont utilisés pour prédire le dernier. Dans le cas où la détermination du $N^{ème}$ élément est unique, la prédiction se fait naturellement lorsque les $N-1$ premiers ont été observés. Si les N-1 premiers peuvent être suivis par plusieurs éléments différents, le système prédira le plus probable, c'est à dire celui qui apparaît le plus souvent.
Cependant plusieurs problèmes apparaissent dans cette approche :

- La taille de la fenêtre est statique. Si on veut changer la valeur de N, une nouvelle phase d'apprentissage est nécessaire.
- Si N n'est pas choisi d'une manière adéquate, les performances du réseau seront très réduites.

Ainsi, un réseau récurrent [5] est mis en place au lieu d'un réseau multicouche. Debar et al. [4][6] ont mis en œuvre le système à l'aide d'un réseau de neurones récurrent en auditant un utilisateur anonyme sur une station de travail Sun 3 UNIX. Ils ont utilisé les fichiers d'audit du système où chaque enregistrement contient le nom de la commande, le taux CPU/Mémoire utilisé et le nombre d'E/S effectuées.

Debar [4] a observé une séquence de 6550 et a entraîné le réseau sur la moitié de cette séquence et l'autre partie est utilisée pour la prédiction et les résultats étaient très prometteurs.

1.2.3 Approche immunologique

Cette approche a été proposée par Forrest et al. [7] en 1996, visant à caractériser le comportement normal d'utilisation du système, et non pas le profil utilisateur, à l'aide de courtes séquences d'appels système générés par les applications.

L'analogie avec le système immunitaire humain vient du fait à la distributivité de la détection, la détection est probabiliste et en ligne, et l'utilisation d'un ensemble d'indicateurs qui détectent des organismes étrangers (par rapport au corps humain).

Présentation de la méthode

Un comportement normal d'une application est défini par un ensemble de courtes séquences d'appels système. La longueur utilisée par chaque séquence est de 5, 6 et 11 éléments.
L'idée consiste à construire une base de données d'un comportement normal pour chaque application d'intérêt.
Pour construire cette base de données, une fenêtre de taille $k+1$ parcourt le long de la trace des appels système et enregistre les appels qui succèdent les uns des autres.
Considérons l'exemple suivant, où $k=3$, et la séquence d'appels système suivante :
<open, read, mmap, mmap, open, getrlimit, mmap, close>
soit <O, R, M, M, O, G, M, C> où O pour open, R pour read, etc.
Quand la fenêtre parcourt le long de la séquence, la base de données suivante est ainsi produite :

Appel	Position 1	Position 2	Position 3
O	{R,G}	{M}	{O,C}
R	{M}	{M}	{O}
M	{M,O,C}	{O,G}	{G,M}
G	{M}	{C}	{}
C	{}	{}	{}

Tableau 1.1.: La base de données pour la séquence <O, R, M, M, O, G, M, C>.

Une fois la base de données créée, les nouvelles traces sont testées en utilisant la même méthode moyennant la fenêtre de taille *k+1* et en déterminant si la nouvelle séquence d'appels système diffère de celle enregistrée auparavant dans la base.

Pour illustrer cette phase, considérons la nouvelle séquence qui suit (où open remplace mmap à la quatrième position de la séquence normale) :
<O, R, M, **O**, O, G, M, C>

Cette trace va générer 4 différences (qui ne se coïncident pas) :

- open est suivi par open en troisième position,
- read est suivi par open en deuxième position,
- open est suivi par open en première position, et
- open est suivi par getrlimit en deuxième position.

On enregistre alors le rapport entre le nombre de positions qui ne coïncident pas avec le nombre total de positions qui ne coïncident pas possible. Dans notre exemple, ce rapport est de 22%=4/18.

L'étape de construction de la base est très délicate ainsi il a été proposé [7] de construire la base à partir des spécifications fonctionnelles de l'application :

- Enumérer à partir des spécifications globales de l'application des sources potentielles de variations normales de son comportement,
- Générer des jeux de données provoquant ces variations,
- Construire une base de données à partir des séquences produites.

Cette construction est empirique, cependant, les expériences menées sur les démons de mail et d'impression ont donné lieu à un processus convergent. Au fur et à mesure de l'apprentissage, la taille de la base n'augmente plus, ce qui constitue une bonne heuristique pour considérer que l'on a capturé le comportement normal de l'application.

Cette approche est très simple à mettre en œuvre et très pratique, mais elle reste limitée car elle se concentre sur les applicatifs sensibles d'un système, alors que les sources potentielles d'attaques sont beaucoup plus étendues.

Beaucoup d'autres IDS comportementaux sont développés et vu le jour tels que NIDES [8] qui utilise d'une part une approche statistique pour la modélisation statistique des relations objet-sujet et utilise d'une autre part, un système expert P-BEST [9-10]. La modélisation statistique permet la détection de la déviation du comportement des utilisateurs ainsi que la comparaison de leur comportement relatifs, la décision de déclencher ou non une alerte étant prise par le système expert P-Best.

D'autre techniques sont aussi exploitées dans l'approche comportementale à savoir les systèmes bayésiens, apprentissage chronique [11] et les mécanismes sans apprentissage [12].

1.3 Approche par scénario

Par opposition à l'approche comportementale qui se base sur le comportement utilisateur et/ou application, l'approche par scénario utilise un ensemble d'attaques bien connues susceptibles de violer la politique de sécurité du système. Un algorithme de détection est mis en œuvre pour rechercher ces attaques dans les fichiers d'audit.

1.3.1 USTAT—Unix State Transition Analysis Tool

USTAT [13][14] est une implémentation du prototype de l'approche état-transition dans la détection d'intrusion sous Unix.

L'analyse état-transition considère le système, à l'état initial, dans un état sain. Ensuite, à partir d'une séquence d'actions entreprise par l'attaquant, le système passe dans un état compromis (vulnérable). USTAT lit les spécifications pour les transitions d'états nécessaires pour compléter une intrusion, fournies par l'officier de sécurité, ensuite recherche dans le fichier d'audit des intrusions en se basant sur ces spécifications.

Un exemple de pénétration

Le tableau suivant présente un scénario d'attaque sur UNIX BSD 4.2 qui peut être utilisé d'une manière illégale pour gagner des privilèges de super utilisateur.

Obtention d'un shell super-utilisateur		
1.	%**cp** /bin/sh /usr/spool/mail/root	la boite aux lettres du super utilisateur est vide
2.	%**chmod** 4755 /usr/spool/mail/root	positionner setuid
3.	%**touch** x	créer un fichier vide
4.	%**mail** root <x	envoyer un fichier vide au root.
5.	% /usr/spool/mail/root	exécute setuid au shell root
6.	root #	l'attaque est réussie (intrusion).

Tableau 1.2.: Obtention d'un shell super-utilisateur.

Dans la version UNIX BSD 4.2, il était possible de changer le propriétaire ainsi que les privilèges pour le fichier courrier /usr/spool/mail/nom_utilisateur. C'est ce que permet le scénario précédent. Il suffit d'attendre que la cible n'ait pas de mail, de recopier le fichier /bin/sh dans le répertoire /usr/spool/mail/root, de mettre sur le fichier le privilège du type suid (l'exécution d'un programme ayant un tel privilège donne a celui qui l'exécute les privilèges du propriétaire), de créer un fichier vide puis de l'envoyer en mail à la cible. Le démon va alors concaténer un fichier vide du shell, changer le propriétaire du fichier, mais laisser les privilège suid. L'attaque est ainsi réussie.

Pour modéliser cette attaque à l'aide du schéma étape-transition, quelques assertions doivent être satisfaites : le root ne doit pas avoir un mail, qui est la première condition qui doit être satisfaite. En progressant tout au long des étapes dans l'exemple, nous remarquons aussi que l'attaquant doit avoir aussi l'autorisation pour exécuter les commandes cp, chmod, touch et mail. L'intrusion à l'aide du prototype étape-transition est décrite dans la figure 1.2.

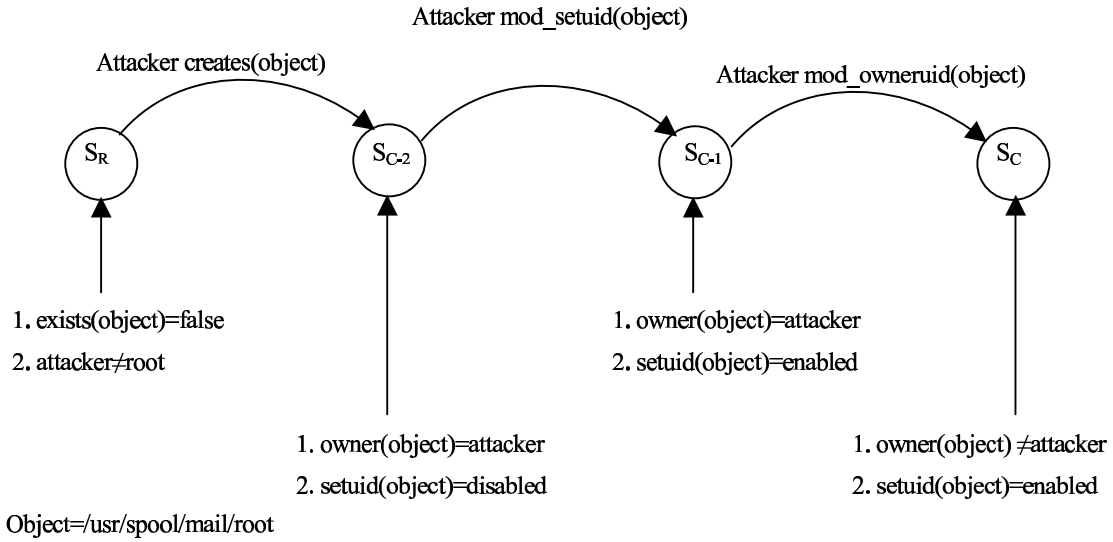

Figure 1.2.: Ustat : Diagramme étape transition [13].

Conclusion sur la méthode

Il est à noter qu'au départ, la réalisation de ce système fut commencée dans le but de représenter les scénarios d'attaque. L'un des problèmes mentionnés par les auteurs est la représentation d'action qui est juste séquentielle. D'ailleurs, pour la signature précédente, une représentation parallèle est préférable. Ceci est réalisé à l'aide des Réseaux de Pétri Colorés qui ont fait l'objet de la thèse de S. Kumar [15] et qui les a utilisés pour détecter la présence de signature d'attaque dans un fichier d'audit.

1.3.2 IDIOT—Les Réseaux de Petri Colorés pour la détection d'intrusion

IDIOT "*Intrusion Detection In Our Time*" [15-18] est un système de détection d'intrusion basé sur les réseaux de Pétri colorés pour détecter les intrusions à base de signatures d'attaque. Chaque signature est représentée à l'aide d'un réseau de Pétri coloré.

La figure 1.3 décrit l'attaque définie dans la section précédente (vol de shell super utilisateur) à l'aide d'un réseau de Pétri coloré. La branche horizontale correspond à l'exécution des deux commandes suivantes :

%**cp** /bin/sh /usr/spool/mail/root
%**chmod** 4755 /usr/spool/mail/root

tandis que la branche diagonale représente l'exécution de la commande

%**touch** x

script	Appel système
%**cp** /bin/sh /usr/spool/mail/root	write
%**chmod** 4755 /usr/spool/mail/root	chmod
%**touch** x	stat ; utime
%**mail** root <x	exec

Tableau 1.3.: Correspondance script-appel système.

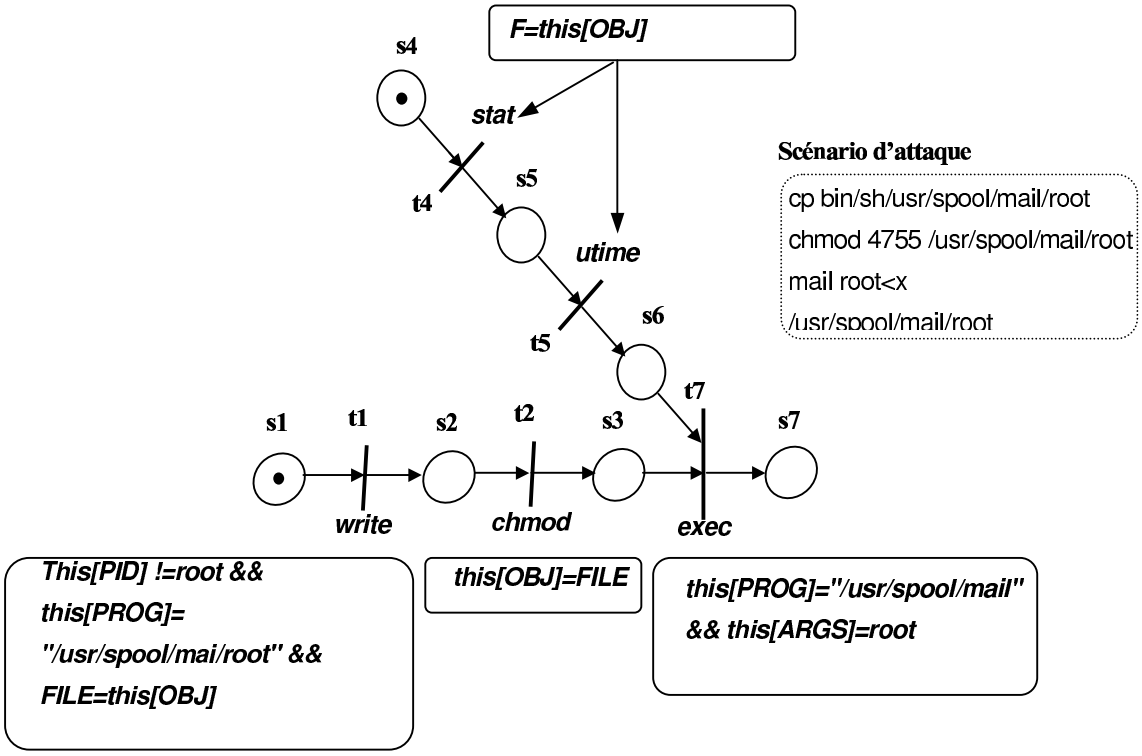

Figure 1.3.: IDIOT – Représentation de la signature avec un RdP Coloré [16].

Chaque signature décrite par un réseau de Pétri peut posséder plus d'une place initiale mais une seule place finale unique. A l'état initial, un jeton est placé dans chaque étape initiale. A chaque transition lui est associé un appel système et les jetons évoluent des places initiales vers la place finale au fur et à mesure que les événements d'audit vérifient les conditions de passage des transitions du RdP coloré.

Chaque jeton possède des variables et chaque couleur associée au jeton correspond à une valuation de ses variables. Ainsi dans notre exemple, les deux appels stat et utime doivent avoir comme objet le même fichier dont on aura le nom lors de la phase d'analyse.

D'une autre part, des gardes sont placées dans les transitions. Ces gardes sont des expressions booléennes. **this** est un opérateur spécial qui est instantié à l'événement de l'audit le plus récent (l'événement courant). Les jetons sont unifiés avant de passer aux gardes pour l'évaluation.

Dans la signature décrite dans la figure 1.3, la transition *t7* ne sera tirée que s'il y a au moins un jeton dans chaque place *s3* et *s6* et l'événement courant doit correspondre à la commande mail (exec) et le destinataire doit être root. Les transitons sont donc étiquetées mais également gardées par des expressions booléennes dont les arguments peuvent correspondre aux champs de l'enregistrement courant de l'audit ou des variables colorant les jetons.

Pour illustrer l'unification, prenons la branche horizontale de la figure précédente. A l'état initial *s1*, lorsque le jeton franchit la transition étiquetée par write, la variable FILE lui est affectée la valeur du nom du fichier de l'enregistrement courant audité (i.e. la valeur de this[OBJ] qui doit être "/usr/spool/mail/root") et quand le jeton s'apprête à franchir la transition étiquetée par chmod, l'appel courant doit avoir le même nom de fichier que la valeur de FILE.

Le choix des réseaux de Pétri colorés se justifie par leur simplicité et la représentation graphique qu'ils fournissent. D'autre part, les grammaires à contexte libre et les expressions régulières ne permettent pas de faire un "matching" conditionnelle sur les valeur des expressions concernées.

Conclusion sur la méthode

Cette approche permet non seulement la détection d'attaques connues mais aussi une représentation graphique claire de ces signatures.

Kumar estime dans sa thèse [15] que les résultats obtenus à l'aide des RdPs Colorés très satisfaisants, d'ailleurs, il a montré qu'il était possible de représenter 88% d'intrusions du CERT [19] connues à l'époque. Cependant, la méthode des signatures n'est pas efficace pour détecter les 12% des signatures et il est plus intéressant d'utiliser une approche comportementale dans ce cas.

1.3.3 GASSATA—Les algorithmes génétiques pour la détection d'intrusion

Les algorithmes génétiques (G.A.), proposés par John Holland dans les années 70, s'inspirent de l'évolution génétique des espèces, plus précisément du principe de sélection naturelle. Il s'agit d'algorithmes de recherche d'optimum construits en s'inspirant de la nature dans laquelle vivent, dans des conditions parfois difficiles, des organismes robustes et adaptables.

La fonction dont on recherche l'optimum est dite fonction objective. On ne fait aucune hypothèse sur cette fonction, en particulier elle n'a pas à être dérivable, ce qui représente un avantage sur certaines méthodes de recherche d'extremum (par exemple les méthodes basées sur le gradient). Pour résoudre un problème quelconque par approche génétique, on devra l'exprimer sous la forme d'une fonction objective. Un algorithme génétique manipule une population de taille constante, formée d'individus représentant chacun le codage d'une solution potentielle au problème à résoudre. Chaque individu prend la forme d'une chaîne de caractères. Dans les cas classiques, l'alphabet chromosomique est

binaire (les deux allèles possibles sont *0* et *1*) mais parfois les spécificités du problème à résoudre amènent à choisir un alphabet de cardinal supérieur à 2.

La population évolue en générations successives. La taille constante de la population induit un phénomène de compétition entre les individus, les plus forts survivant et se reproduisant entre eux pour créer de nouveaux individus, les plus faibles disparaissant petit à petit. De plus, lors des créations d'individus, des mutations génétiques (c'est-à-dire la modification d'un caractère dans la chaîne) se produisent. Cette analogie avec la nature conduit à définir les trois opérateurs génétiques de base : sélection, recombinaison (crossover) et mutation.

La traduction algorithmique des adjectifs *"faible"* et *"fort"* appliqués aux individus conduit à définir une fonction sélective qui permet d'associer une valeur dite sélective à chaque individu de la population.

L'application des opérateurs génétiques sur des individus jugés par une fonction sélective particulière permet d'explorer l'espace des solutions à la recherche d'un extremum. Cette recherche se fait avec un très bon équilibre entre l'exploitation des résultats déjà acquis et l'exploration de nouvelles régions de l'espace, ce qui permet d'obtenir un très faible rapport entre le nombre de points échantillonnés et le nombre de points total dans l'espace de recherche. La structure générale d'un algorithme génétique ainsi que les différents opérateurs génétiques sont représentés dans la figure 1.4 :

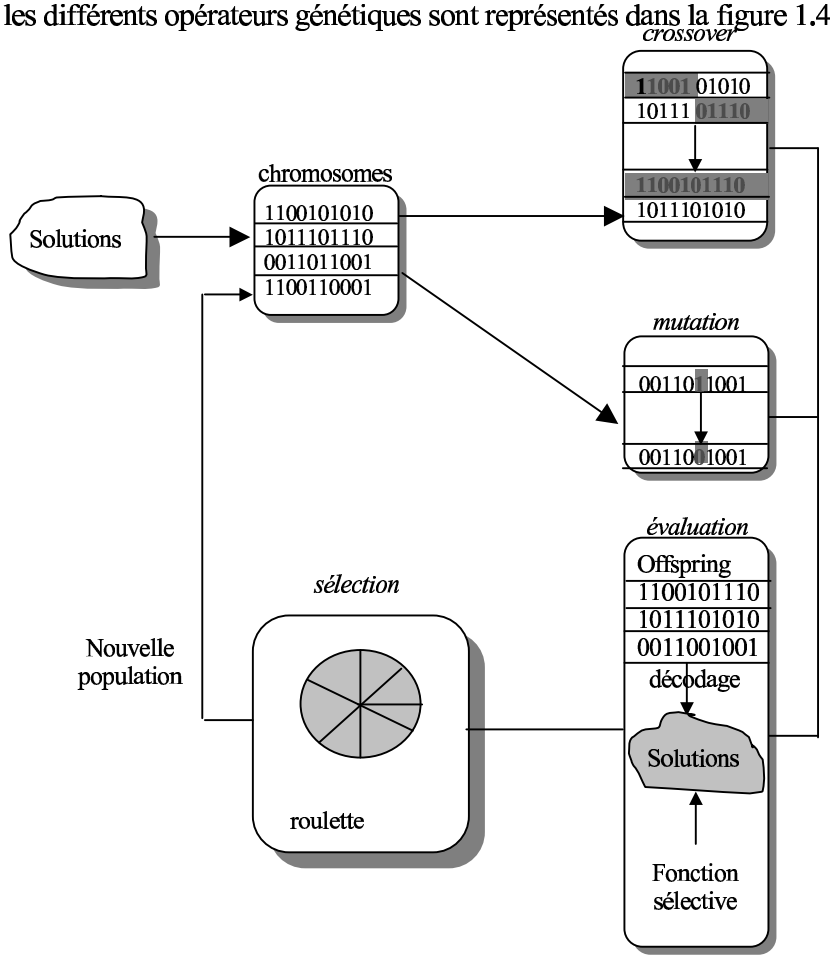

Figure 1.4.: Structure générale d'un algorithme génétique [21].

Mé [20] donne un exemple d'application des algorithmes génétiques à la recherche de scénarios d'attaque prédéfinis dans les traces d'audit, chaque scénario étant défini comme un ensemble d'événements. Tous les entremêlements possibles de ces scénarios doivent être pris en compte. On a là un phénomène d'explosion combinatoire qui interdit l'utilisation d'algorithmes de recherche classiques. L'approche génétique, grâce à son bon équilibre exploration/exploitation, permet d'obtenir en un temps de traitement raisonnable (quelques secondes à quelques minutes selon le nombre d'attaques considérées sur un *IBM RS6000 320*) le sous-ensemble des attaques potentiellement présentes dans les traces d'audit.

Présentation de l'approche proposée par Mé (algorithmes génétiques) :

Dans sa thèse [21], Mé a proposé une approche par scénario qui ne tient pas compte de l'ordre d'apparition des événements dans le fichier d'audit. Les scénarios d'attaque sont exprimés sous forme de couple (E_i,N_i) où E_i représente un type d'événement et N_i représente le nombre d'occurrences de ce type d'événement dans le scénario.

Le fichier d'audit est exprimé sous la forme d'un vecteur colonne d'entiers de dimension N_e, si N_e types d'événement sont auditables. Ce vecteur est appelée matrice d'audit et est notée O (pour Observé). Le $i^{ème}$ élément O_i de O donne le nombre d'occurrences de l'événement i observées durant la session auditée. Chaque attaque lui est associé un poids R_i proportionnel au risque encouru par le système si cette attaque est réalisée.

Le problème de l'analyse simplifiée de fichiers d'audit de sécurité (PASFAS) s'exprime alors de la manière suivante : soit N_e le nombre de types d'événement auditables et N_a le nombre de scénarios d'attaque possibles. Chaque scénario d'attaque est exprimé sous la forme d'un ensemble de N_e couples $(e_i; n_j)$ où e_i est le type d'événement $(1 \leq i \leq Ne)$ et n_j son nombre d'occurrences dans le scénario $(n_j \geq 0)$. Une matrice de dimension N_e x Na *est* construite, notée AE pour Attaque-Evénement, dans laquelle AE_{ij} représente le nombre d'occurrences de l'événement i dans le scénario j $(AE \geq 0)$. Soit R (pour Risque) la matrice ligne de dimension N_a dans laquelle R_i donne le poids de l'attaque i $(R_i > 0)$. Soit O la matrice d'audit de dimension N_e. Soit H (pour Hypothèse) une matrice colonne binaire, de dimension N_a, dans laquelle H_i vaut *1* si on émet l'hypothèse que l'attaque i s'est produite tandis qu'il vaut *0* dans le cas contraire (la matrice H décrit un sous-ensemble d'attaques). L'analyse du fichier d'audit de sécurité consiste à déterminer la matrice H maximisant le produit matriciel $R.H$, tout en respectant les contraintes $(AE.H)_i \leq O_i$, i variant de *1* à N_e.

Le produit matriciel $AE.H$ résulte en une matrice colonne de dimension N_e. L'élément $(AE.H)_i$ $(1 \leq i \leq N_e)$ de cette matrice correspond à la somme des éléments AE_{ij} $(1 \leq j \leq N_a)$ de la matrice Attaque-Evénement, j prenant les valeurs successives des indices de la matrice H pour lesquels $H_j=1$. Il s'agit par conséquent du minimum des nombres d'occurrences de l'événement i, que l'on devrait trouver

dans le fichier d'audit, si la combinaison d'attaques codée dans la matrice H avait effectivement eu lieu. $(AE.H)_i$ est un minimum car des événements de type i peuvent être générés en dehors de tout scénario d'attaque. L'hypothèse codée dans H n'est donc réaliste que si le nombre d'événements de type i enregistrés dans la matrice d'audit est supérieur ou égal au nombre de ces événements donné par $(AE.H)_i$ (voir le système d'équations 1.1).

$$\begin{cases} \begin{pmatrix} 1 & i & Na \\ & R_i & \end{pmatrix} \begin{pmatrix} \\ H_i \\ \\ \end{pmatrix} = maximum \\ 1 \begin{pmatrix} 1 & i & Na \\ & & \\ & AE_{ij} & \\ & & \\ Ne & & \end{pmatrix} \begin{pmatrix} \\ H_i \\ \\ \end{pmatrix} \begin{matrix} 1 & 1 \\ & \\ i \le j & \\ & \\ Na & Ne \end{matrix} \begin{pmatrix} \\ O_i \\ \\ \end{pmatrix} \end{cases} \qquad (1.1)$$

Le temps de traitement nécessaire à l'analyse simplifiée d'un fichier d'audit de sécurité (avec $N_e=50$) [20], sur un *IBM RS6000 320* à l'aide d'une méthode directe en testant tous les cas possibles et en considérant le temps d'exécution nécessaire pour chacune des opérations addition, soustraction, multiplication…, est donné dans le tableau suivant :

N_a	**Temps nécessaire**
1	33.6 µsec.
5	3.12 msec.
10	0.19 sec.
15	8.8 sec.
20	6.16 minutes.
24	1 heure 57 mn.
30	155 heures 20 mn.
40	24 années
50	305 siècles et 8 ans

Tableau 1.4.: Temps nécessaire pour l'exécution de PASFAS [20].

Ce problème (PASFAS) est NP-complet. Les temps de traitement nécessaires à la résolution de PASFAS ne permettent pas d'envisager une solution algorithmique non polynomiale dans les cas réels (gourmand en temps de calcul si le nombre de scénarios d'attaque est élevé) (voir le tableau précédent). C'est pourquoi il convient d'envisager l'utilisation d'une méthode heuristique telle que les algorithmes génétiques.

Codage des individus

Un individu est une chaîne de longueur L codant une solution potentielle au problème à résoudre. Or, résoudre PASFAS consiste à déterminer la matrice colonne H maximisant le produit matriciel $R.H$, tout en respectant la contrainte $(AE.H)_i \leq O_i$, i variant de 1 à N_e. La matrice H est une matrice binaire, de dimension N_a, dans laquelle H_i vaut 1 si H traduit une hypothèse où l'attaque i s'est produite tandis qu'il vaut 0 dans le cas contraire.

Après avoir fixé les différents paramètres nécessaires pour l'utilisation d'un algorithme génétique pour ce problème et après expérimentation sur quatre types d'utilisateurs (d'un utilisateur de mail, d'un novice, d'un développeur et d'un utilisateur expérimenté) sur une période d'une demi-heure (sur un *IBM RS6000 320*), il a été observé des taux qui permettent une excellente discrimination entre attaques présentes et attaques absentes dans la matrice d'audit.

L'outil utilisant cette approche est appelé GASSATA (Genetic Algorithm for Simplified Security Audit Trail Analysis).

Les performances de GASSATA sur un IBM RS6000 320 sont exprimées dans le tableau suivant [20]:

Nombre d'attaques	**Temps de traitement (en secondes.)**
24	18
27	24
40	32
50	38
70	49
100	104
150	300
200	625

Tableau 1.5.: Temps nécessaire pour PASFAS avec GASSATA [20].

Critiques sur la méthode

Cette approche basée sur les algorithmes génétiques présentent les avantages suivants :

- l'utilisation des algorithmes génétiques pour résoudre le problème d'optimisation sous forme de contraintes (1.1) réduit le traitement d'une manière considérable de l'ordre d'années (voir siècles) à l'ordre des minutes et secondes par rapport à une méthode énumérative.
- La définition des scénarios sous forme d'une matrice simplifie à l'officier de sécurité l'ajout de nouveaux où la suppression de scénarios d'attaque dans cette matrice.

D'autre part, elle présente des inconvénients, on peut citer entre autres :

- l'utilisation d'un codage binaire ne permet pas de détecter une réalisation multiple d'une attaque particulière.
- si un intrus réalise, simultanément, plusieurs attaques qui ont un nombre d'actions en commun, GASSATA associe ces événements à l'une ou à l'autre.
- en utilisant les algorithmes génétiques, s'il y a plusieurs solutions optimum, GASSATA fournit l'une d'entre elles d'une manière aléatoire.
- si l'intrus connais a priori la période de la session d'audit, il peut réaliser l'attaque pendant deux ou plusieurs sessions d'audit. Cette dernière ne pourra pas être détectée par l'algorithme. Une solution donnée consiste en l'exécution de l'algorithme dès qu'il est possible et de considérer tout le fichier d'audit.

L'ordre temporel d'apparition des événements n'est pas pris en compte, ce qui signifie que l'algorithme ne fournit qu'une présomption d'attaque ce qui nécessite une analyse ultérieure du fichier d'audit pour localiser précisément les attaques détectées. GASSATA ne se distingue pas sur ce point des approches comportementales.

1.4 Conclusion

Nous avons présenté dans ce chapitre, brièvement, les deux grandes familles dans le domaine de détection d'intrusion ; l'approche comportementale et l'approche par scénario. Le tableau suivant résume les avantages et inconvénients de chacune d'entre elles.

Approche	Avantages	Inconvénients
Comportementale	Détection d'intrusion inconnue possible. Possibilité de détecter des intrus de type masquerador [2].	Choix délicat des mesures à retenir pour un système cible donné. Pour un utilisateur au comportement erratique, toute activité est "normale ". En cas de profonde modification de l'environnement du système cible , déclenchement d'un flot ininterrompu d'alarmes (faux positifs). Utilisateur pouvant changer lentement de comportement dans le but d'habituer le système à un comportement intrusif (faux négatifs).
Par scénario	Prise en compte des comportements exactes des attaquants potentiels.	Bases de règles délicates à construire. Seules les attaques contenues dans la base sont détectées. Mise à jour régulière de la base si l'on veut être protégé des toutes dernières attaques connues.

Tableau 1.6.: Avantages et inconvénients des deux approches.

Bien que plusieurs IDS ont été développés, il reste beaucoup à faire pour améliorer ces systèmes et surmonter leurs inconvénients respectifs. En effet, des études approfondies pourront retrouver leurs vulnérabilités qui peuvent être utilisées par des criminels illégalement et présenter une grande menace aux systèmes les utilisant.

Dans le chapitre suivant, nous présentons dans la première partie une amélioration de GASSATA qui permettra de résoudre trois premiers inconvénients cités pour cet outil. La deuxième partie est consacrée à notre contribution dans le domaine des IDS et qui consiste en l'utilisation de la théorie de l'information et son application dans ce domaine.

Chapitre 2

Critiques et Contributions

Nous avons vu, dans l'état de l'art, les deux approches de détection d'intrusion ainsi quelques outils développés respectivement dans chacune d'elles. Le dernier outil présenté est GASSATA appartenant à l'approche par scénario. Ce dernier fait l'objet de la première partie de ce chapitre dans laquelle nous illustrons avec un simple exemple quelques inconvénients générés par cet algorithme ensuite nous introduisons un nouveau formalisme au problème d'analyse simplifié du fichier d'audit de sécurité (PASFAS) qui apportera des solutions efficaces à ce problème. Puis après avoir introduit rapidement notre formalisme, nous présenterons les résultats expérimentaux de notre nouveau modèle et une comparaison avec ceux de GASSATA.

La deuxième partie introduira une nouvelle méthode de détection d'intrusion qui est basée sur la théorie de l'information. Celle-ci cherche à coder l'information essentielle du profil utilisateur dans un ensemble de profils que nous appelons "*profils propres*". Les différentes étapes de cette méthode ainsi que les différentes formules mathématique nécessaires sont aussi développées. Nous montrons à la fin de cette partie les résultats préliminaires de cette méthode et sa capacité de bien classifier les différents utilisateurs d'un système unix ainsi que son application sur un fichier web log réel.

2.1 Partie 1 : Amélioration de GASSATA

2.1.1 Formalisation

Nous avons gardé les mêmes matrices utilisées pour l'outil GASSATA (*AE*, *I* (pour *H*), et *O*), par contre *I* est un vecteur colonne d'entiers positifs qui détermine non seulement la présence ou l'absence d'une attaque mais le nombre d'occurrences de cette attaque dans la matrice observée *O*. La résolution de ce problème est simple et consiste à comparer chaque scénario d'attaque (i.e. chaque vecteur colonne de la matrice Attaque-Evénement) avec le vecteur contenant les occurrences des événements audités *O*. Notre formalisation est alors la suivante :

$$\begin{cases} Max(\sum_{i=1}^{Na} I_i) \\ s.c.(I_i \times AE^i \leq 0)_{i=1,..,Na} \end{cases} \qquad (2.1.1)$$

Où AE^i est la $i^{ème}$ colonne de AE, représentant le $i^{ème}$ scénario d'attaque.

Il est clair que le système *(2.1.1)* n'est pas un problème NP-Complet et sa résolution est très simple :

$$I_i = \min_{j=1,..,Ne} \left\lfloor \frac{O_j}{AE_{ji}} \right\rfloor, \quad i=1,..,Na \quad tq \quad AE_{ji} \neq 0 \qquad (2.1.2)$$

2.1.2 Comparaison avec GASSATA

Nous montrons à travers l'exemple suivant que notre formalisation trouve des solutions aux trois premiers inconvénients cités dans la dernière section sur GASSATA dans le chapitre précédent.

Appelons ce nouveau modèle PASFAS-G (Problème d'Analyse Simplifiée du Fichier d'Audit de Sécurité Généralisé).

Considérons la matrice Attaque Evénement $AE = \begin{pmatrix} 5 & 0 & 1 & 0 \\ 3 & 6 & 2 & 0 \\ 1 & 2 & 4 & 0 \\ 2 & 1 & 0 & 8 \\ 2 & 0 & 0 & 0 \end{pmatrix}$ *et* $O = \begin{pmatrix} 10 \\ 8 \\ 5 \\ 9 \\ 4 \end{pmatrix}$ la matrice observée.

En utilisant GASSATA, le vecteur optimum H pourra prendre l'une des trois formes suivantes :

$H_{max} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$, $H_{max} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$, ou $H_{max} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$.

Dans notre exemple, bien que trois solutions optimum sont possibles, GASSATA ne fournit que l'une d'entre elles. Nous remarquons aussi que les scénarios d'attaques qui se partagent des événements ne sont pas détectés simultanément. D'ailleurs, si l'attaque *1* est détectée alors les attaques *2* et *4* ne le sont pas et si l'attaque *4* est détectée alors l'attaque *1* ne l'est pas et vice versa.

Par contre la solution générée par notre modèle, non seulement, délivre une seule solution optimale mais aussi générale car elle détecte la duplication d'une attaque comme dans notre exemple, bien que l'attaque *1* est dupliquée mais elle n'est pas détectée par GASSATA. La solution générée par notre modèle est :

$I_{max} = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}$

Nous remarquons qu'avec notre nouveau modèle toutes les attaques sont détectées et les attaques dupliquées sont aussi mentionnées dans le vecteur *I*.

A partir de l'équation 2.1.2, notre problème est résolu d'une manière simple :

$I_1 = \min(O_1 \div AE_{11}, O_2 \div AE_{21}, O_3 \div AE_{31}, O_4 \div AE_{41}, O_5 \div AE_{51})$

$=\min(10\div5,8\div3,5\div1,9\div2,4\div2)=2$

$I_2=\min(O_2\div AE_{22},O_3\div AE_{32},O_4\div AE_{42})\ \{AE_{12}=AE_{52}=0\}$

$=\min(8\div6,5\div2,9\div1)=1$

et ainsi de suite, nous retrouvons $I_{max}=\begin{pmatrix}2\\1\\1\\1\end{pmatrix}$.

2.1.3 Résultats expérimentaux

Pour réaliser des expérimentations réelles, nous avons utilisé la même matrice Attaque Evénement et les mêmes types d'utilisateurs que ceux utilisés dans [20]. (voir annexe pour la matrice attaque événement).

Le graphe et le tableau suivants montrent le temps de traitement en utilisant GASSATA et PASFAS-G.

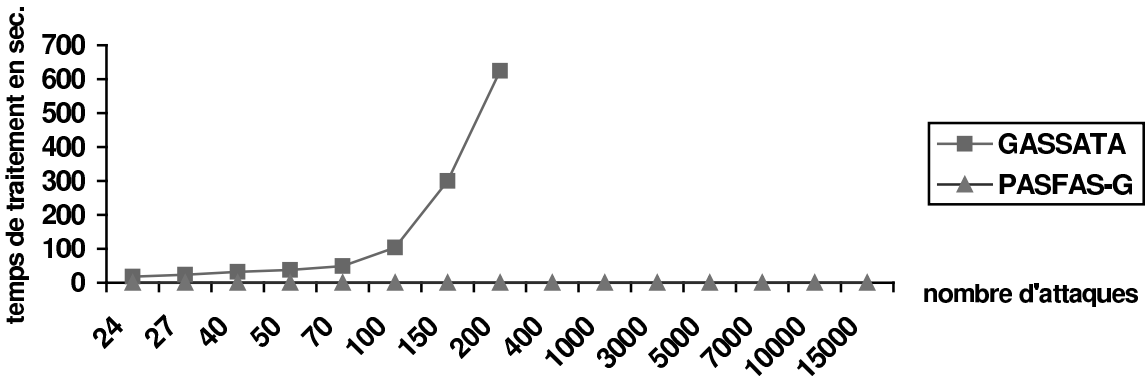

Figure 2.1.1.: Temps de traitement avec GASSATA et PASFAS-G.

Nombre d'attaques	a) Temps d'exécution avec GASSATA	b) Temps d'exécution avec PASFAS-G
24	**18 sec.**	**≈ 0 sec.**
27	**24 sec.**	**≈ 0 sec.**
40	**32 sec.**	**≈ 0 sec.**
50	**38 sec.**	**≈ 0 sec.**
70	**49 sec.**	**≈ 0 sec.**
100	**104 sec.**	**≈ 0 sec.**
150	**300 sec.**	**≈ 0 sec.**
200	**625 sec.**	**≈ 0 sec.**
400	>625 sec.	≈ 0 sec.
1 000	>>625 sec.	0.05 sec.
3 000	>>625 sec.	0.06 sec.
5 000	>>625 sec.	0.11 sec.
7 000	>>625 sec.	0.17 sec.
10 000	>>625 sec.	0.22 sec.
15 000	>>625 sec.	0.38 sec.

Tableau 2.1.1.: Temps de traitement avec GASSATA [20] et PASFAS-G.

2.1.4 Conclusion sur la méthode

Nous avons présenté dans cette première partie une étude critique d'un IDS appelé GASSATA ainsi que ses différents inconvénients à partir d'un simple exemple qui montre les faiblesses de cet algorithme pour détecter la réalisation de multiples attaques et l'impossibilité de détecter les scénarios d'attaque qui se partagent un ensemble d'événements communs. Nous avons par la suite, introduit un nouvel algorithme qui peut détecter toutes les attaques présentes dans un fichier d'audit indépendamment les unes des autres et avec un temps de traitement très réduit de l'ordre de la seconde même en considérant des milliers de scénarios d'attaque.

D'autre part, nous avons aussi développé un autre algorithme que nous n'avons pas présenté dans ce rapport, qui à partir des résultats de PASFAS-G on retrouve en temps très réduit (de l'ordre de la seconde) les résultats de GASSATA. Le passage des résultats de PASFAS-G vers ceux de GASSATA n'est pas nécessaire car cet outil ne donne qu'une présomption d'attaque. Ainsi, il est préférable d'utiliser PASFAS-G car il détecte tous les scénarios d'attaque présents dans le fichier d'audit et il est plus général.

2.2 Partie 2 : Vers l'utilisation de l'ACP dans la Détection d'intrusion

2.2.1 Introduction

Nous présentons, dans cette partie, une nouvelle méthode de détection d'intrusion appartenant à la famille comportementale qui est basée sur l'analyse en composantes principales (ACP). Cette approche fonctionne en projetant les profils utilisateurs sur un espace de traits qui peut décrire de signifiantes variations entre les profils. Ces traits sont connus sous le nom de "*profils propres*" car ils sont les vecteurs propres de l'ensemble des profils. L'opération de projection caractérise un profil utilisateur par une somme pondérée de l'ensemble des profils. Pour détecter si un profil est anormal, il suffit de comparer ces poids à ceux des profils utilisateurs connus.

L'analyse en composantes principales (PCA) [22] est une méthode populaire utilisée d'une manière excessive dans la réduction des dimensions d'espace et elle est utilisée dans plusieurs textes d'analyse multivariée. Elles est appliquée dans plusieurs domaines tels que la compression de données, le traitement d'images et la reconnaissance des formes.

L'ACP consiste à réduire un système complexe de corrélations en un plus petit nombre de dimensions. Ayant un ensemble de vecteurs observés $\{v_i\}, i \in \{1,...,N\}$, les q principaux axes $w_j, j \in \{1,...,q\}$ sont les axes orthonormés sur lesquels la variation sous une projection est importante. Il est prouvé que les vecteurs w_j sont données par les q vecteurs propres dominants (i.e. ceux associés aux plus grandes valeurs propres) de la matrice de covariance $C = \sum_i \frac{(v_i - \bar{v})(v_i - \bar{v})^T}{N}$ tel que $Cw_j = \lambda_j w_j$, où $\bar{v}$ est la moyenne arithmétique simple. $u_i = C^T(v_i - \bar{v})$, de dimension q, est ainsi une représentation réduite du vecteur observé v_i .

2.2.2 Les profils propres

La plupart des travaux effectués sur l'approche comportementale ne se sont intéressés que sur les mesures d'un profil utilisateur qui sont importantes pour les utiliser dans l'algorithme de détection. Ceci nous a suggérés qu'une théorie d'information codant et décodant les comportements des utilisateurs peut donner de nouvelles informations sur ces comportements contenant de signifiantes caractéristiques.

Dans le langage de la théorie de l'information, nous voulons extraire les informations essentielles dans un profil utilisateur, le coder d'une manière très efficace, ensuite comparer un comportement ainsi codé avec une base de données des comportements utilisateurs codés de la même façon. Une méthode

simple pour extraire les informations contenues dans un profil consiste à capturer la variation dans une collection de comportements des utilisateurs et utiliser ces informations pour coder et comparer les profils des comportements utilisateurs.

Dans le langage mathématique, nous souhaitons trouver les composantes principales de la distribution des comportements, ou bien trouver les vecteurs propres de la matrice de covariance de l'ensemble des profils utilisateurs, en considérant un comportement comme un point (ou vecteur) dans un espace de dimension égale au nombre des différentes métriques utilisées. Ces métriques sont des mesures qui peuvent être le temps CPU utilisé, le nombre de commandes introduites par l'utilisateur durant une session, le nombre de chaque type d'événement audité durant un intervalle de temps, …

Ces vecteurs propres peuvent être considérés comme un ensemble de traits qui caractérisent la variation entre les comportements des utilisateurs. Chaque position d'un comportement contribue plus ou moins pour chaque vecteur propre que nous appelons profil propre.

Chaque profile peut être représenté par une combinaison linéaire des profils propres. Chaque profil peut être aussi *approximé* en utilisant seulement les meilleurs profils —ceux correspondant au plus grandes valeurs propres— et qui comptent pour la plus grande variation dans l'ensemble des profils des utilisateurs.

Un profil normal d'un utilisateur, dans notre système, consiste en un ensemble de mesures statistiques telles que celles proposées par Denning [3] :

- le temps CPU utilisé
- le nombre de mots de passe erronés introduits durant un intervalle de temps,
- le nombre de chaque type de commande introduite par l'utilisateur pendant un intervalle de temps,
- nombre de fichiers ouverts pendant une période,
- nombre d'applications exécutées durant une session…

L'ensemble de ces mesures sont collectées dans un vecteur de dimension N appelé vecteur du profil utilisateur représentant le profile de l'utilisateur durant une session ou un intervalle de temps choisie, où N est le nombre de mesures statistiques mentionnées ci dessus.
Si Γ est le profil qui correspond à un comportement d'un certain utilisateur, alors nous pouvons écrire

$$\Gamma = \begin{pmatrix} m_1 \\ m_2 \\ \cdot \\ \cdot \\ \cdot \\ m_N \end{pmatrix} \qquad (2.2.1)$$

où $m_i, i=1..N$ sont les métriques caractérisant le profil utilisateur.

2.2.3 Les différentes étapes de la méthode

Le schéma général de notre système est décrit dans la figure *(2.2.1)*

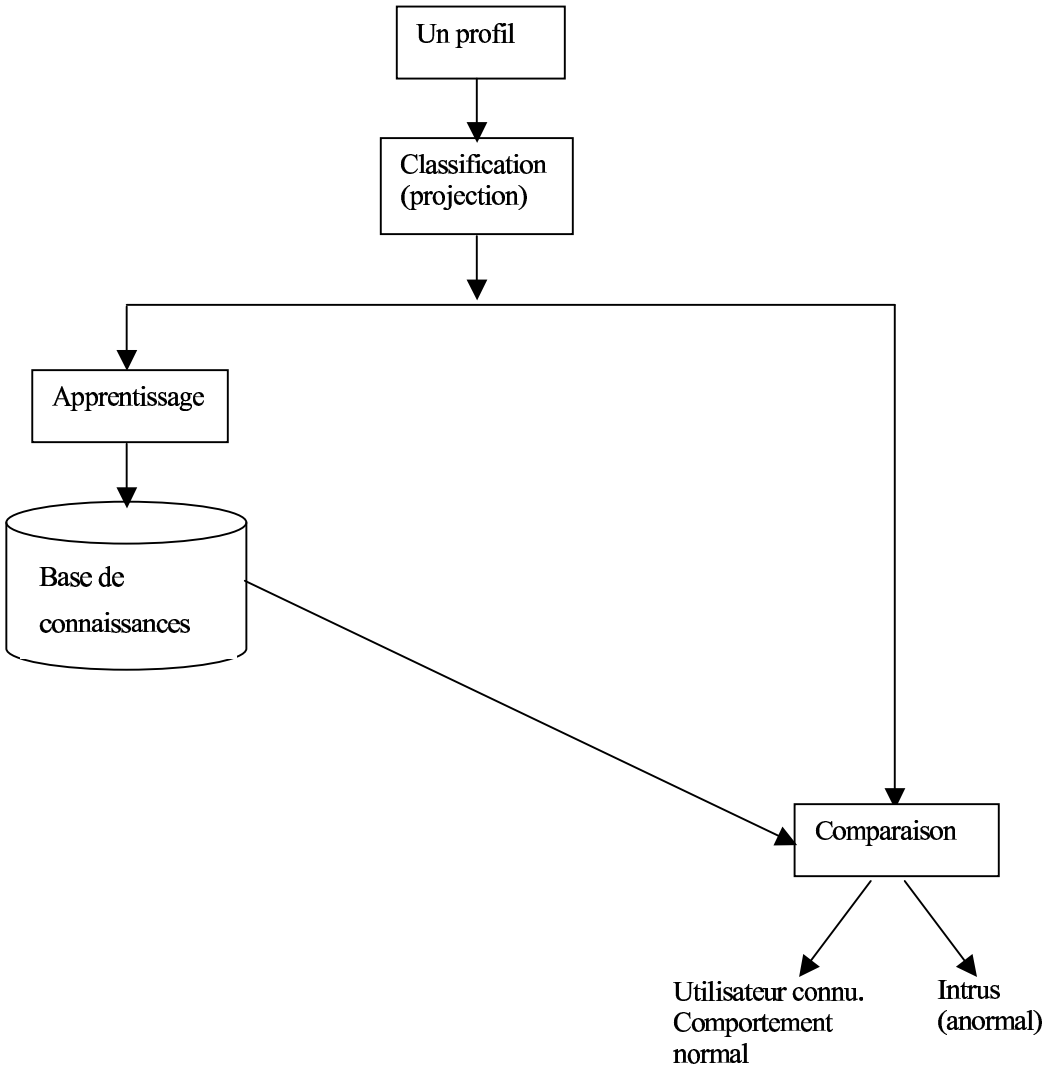

Figure 2.2.1.: Structure générale de notre système.

Nous pouvons résumer les différentes étapes de cette méthode comme suit :

1. Un processus d'initialisation est nécessaire, il consiste à :

- Auditer l'ensemble des profils des utilisateurs du réseau.
- Calculer les vecteurs propres de cet ensemble
- Extraire les profils traits des profils connus
- Pour chaque classe (utilisateur), déterminer le vecteur trait de référence
- Déterminer le seuil de chaque classe.

2. Un processus de classification est ensuite utilisé pour détecter si un nouveau comportement ainsi audité est normal ou non. Il comporte les étapes suivantes :

- La projection du profil audité sur l'espace des profils propres,
- Comparaison de son vecteur trait avec ceux des profils connus pour l'identification et la détection.

2.2.3.1 Le processus d'initialisation

Considérons un système comportant *N* utilisateurs. Chaque utilisateur est audité *NU* fois pendant des périodes différentes. Alors le nombre de profils est de *N*x*NU*.

Profil moyen des utilisateurs

Soit l'ensemble des profils audités $\Gamma_1, \Gamma_2, ..., \Gamma_M$. Le profil moyen de cet ensemble est défini comme étant le centre de gravité des métriques des profils utilisés pour l'apprentissage.

$$\Psi=\frac{1}{M}\sum_{i=1}^{M}\Gamma_i \qquad (2.2.2)$$

Profil caricature d'un utilisateur

Le profil caricature Φ_i est défini comme la différence entre le profil connu Γ_i et le profil moyen Ψ :

$$\Phi_i=\Gamma_i-\Psi,\ \ i=1..M \qquad (2.2.3)$$

Calcul des profils propres

Les profils propres sont les vecteurs propres de la matrice de covariance *C*

$$\text{Où } C=\frac{1}{M}\sum_{i=1}^{M}\Phi_i\Phi_i^T=A.A^T \qquad (2.2.4)$$

$$\text{et } A=\frac{1}{\sqrt{M}}\left[\Phi_1\ \Phi_2 ... \Phi_M\right] \qquad (2.2.5)$$

soit : U_k le $k^{ème}$ vecteur propre de C associé à la valeur propre λ_k et $U=\left[U_1\ U_2 ... U_M\right]$ est la matrice de ces vecteurs propres (profils propres). Alors

$$CU_k=\lambda_k U_k \qquad (2.2.6)$$

tel que

$$U_k^t U_n=\begin{pmatrix}1 & si\ k=n\\ 0 & si\ k\neq n\end{pmatrix} \qquad (2.2.7)$$

le vecteur trait du profil est :

$$\Omega_i = U^T \times \Phi_i = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_M \end{pmatrix} \qquad (2.2.8)$$

Les poids $\omega_i, i=1..M$ décrivent la contribution de chaque profil propre dans la représentation du profil introduit, traitant les vecteurs propres comme l'ensemble de base des comportements des utilisateurs. Ce vecteur trait peut être par la suite utilisé dans un algorithme de classification standard pour trouver la classe dans les comportements établis dans la phase d'apprentissage qui décrit au mieux ce nouveau comportement. La première idée qui vient à l'esprit pour déterminer quelle classe produit la meilleure description d'un vecteur en entrée est de trouver une classe dans l'espace des profils propres qui minimise la distance euclidienne avec le profil en entrée décrite dans le paragraphe suivant.

Si la taille d'un vecteur de profil est *N* (nombre de métriques considérées), la matrice *C* est de taille *N* par *N* et le calcul de *N* valeurs propres et vecteurs propres est une tâche très difficile à accomplir si on considère des centaines de métriques. Ainsi, une méthode faisable qui à partir des équations *(2.2.6)* et *(2.2.4)*, on peut obtenir

$$A.A^T U_k = \lambda_k U_k \qquad (2.2.9)$$

$$A^T.A(A^T.U_k) = \lambda_k (A^T.U_k) \qquad (2.2.10)$$

soit

$$Y_k = A^T U_k \qquad (2.2.11)$$

alors

$$A^T.AY_k = \lambda_k Y_k \qquad (2.2.12)$$

d'après l'équation *(2.2.12)* Y_k est un vecteur propre de la matrice $A^T.A$ associé à la valeur propre λ_k.

Soit $X_k = \alpha_k Y_k$

Donc X_k est aussi un vecteur propre de la matrice $A^T.A$.

D'après l'équation *(2.2.11)*

$$X_k^T X_k = (\alpha_k A^T U_k)^T (\alpha_k A^T U_k) = \alpha_k^2 \lambda_k U_k^T U_k \qquad (2.2.13)$$

Comme $U_k^T U_k = 1$

Alors $X_k^T X_k = \alpha_k^2 \lambda_k \qquad (2.2.14)$

Pour obtenir un vecteur X_k normé (i.e. $X_k^T X_k = 1$) il faut avoir

$$\alpha_k^2 \lambda_k = 1 \Rightarrow \alpha_k = \frac{1}{\sqrt{\lambda_k}} \qquad (2.2.15)$$

d'après les équations *(2.2.11)* et *(2.2.9)*, nous avons :

$$AY_k = AA^T.U_k \qquad (2.2.16)$$

$$AY_k = \lambda_k U_k \qquad (2.2.17)$$

$$U_k = \frac{1}{\lambda_k} AY_k = \frac{1}{\lambda_k \alpha_k} AX_k = \frac{1}{\sqrt{\lambda_k}} AX_k \qquad (2.2.18)$$

D'où

$$\boxed{U_k = \frac{1}{\sqrt{\lambda_k}} AX_k} \qquad (2.2.19)$$

Avec cette analyse, les calculs sont réduits d'une manière considérable, de l'ordre du nombre de métriques utilisées jusqu'à l'ordre du nombre de profils utilisés dans la phase d'apprentissage, car X_k est un vecteur propre de $A^T A$ qui est de dimension très réduite *M*.

Calcul des vecteurs traits

Le vecteur trait Ω_i d'un profil Γ_i est obtenu en projetant son vecteur caricature Φ_i sur l'espace des profils propres

$$\Omega_i = U^T \times \Phi_i = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_M \end{pmatrix} \text{ où } U = [U_1\ U_2 \ldots U_M]$$

Ainsi, chaque profil est représenté par un ensemble de *M'* points (le vecteur trait).

Organisation en classes

A chaque utilisateur lui correspond une classe composée des vecteurs traits obtenus en projetant les *NU* profils audités dans l'espace des profils propres. Chaque classe *k* est représentée par un vecteur trait de référence Ω^k :

$\Omega^k = \frac{1}{NU} \sum_{i=1}^{NU} \Omega_i^k$ où Ω_i^k est le $i^{ème}$ vecteur trait du $k^{ème}$ utilisateur (classe).

La méthode la plus simple qui détermine la classe la plus proche du profil introduit (Γ_i) consiste à déterminer la classe *k* qui minimise la distance Euclidienne

$$\varepsilon_k = \left\| (\Omega - \Omega^k) \right\|^2 \qquad (2.2.20)$$

En plus, ce profil qui vient d'être audité sera affecté à la classe la plus proche si la condition suivante est vérifiée :

$\varepsilon_k < seuil\,\theta_k$ (θ_k : seuil choisi par expérience)

dans le cas contraire, le profil est classé comme intrus!

2.2.3.2 Le Processus d'identification et de détection

Le processus de détection comporte les étapes suivantes

- Auditer un nouveau profil Γ_i à vérifier.
- Calculer son profil caricature $\Phi_i = \Gamma_i - \Psi$.
- Projeter le profil caricature Φ_i sur l'espace des profils. On obtient alors le vecteur trait :

$$\Omega_i = U^T \times \Phi_i = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_M \end{pmatrix}$$ avec U matrice des profils propres U_i.

- Déterminer la classe k qui minimise la distance $\varepsilon_k = \left\| \Omega_i - \Omega^k \right\|^2$
- **si** $\varepsilon_k < \theta_k$ **alors**

  le nouveau profil observé est celui du $k^{ème}$ utilisateur.
- **sinon**

  le nouveau profil observé est anormal.
- **fsi**

2.2.4 Résultats expérimentaux préliminaires

Afin de montrer la rapidité, la robustesse et la simplicité de notre approche, nous avons considéré un simple exemple (qui peut être considéré comme une petite partie de notre système car on ne prend en compte que les occurrences des événements et quelques commandes) qui consiste en quatre types d'utilisateurs qui ont été utilisés dans [20] : l'utilisateur de mail, le novice, le développeur et l'utilisateur inexpérimenté. Les enchaînements qui suivent correspondent à des activités possibles pendant une courte période, de l'ordre de *30* minutes. Les métriques que nous considérons dans notre première expérimentation sont les différentes commandes[1] introduites par l'utilisateur durant un intervalle de temps.

---

[1] Ces commandes sont traduites en événements d'audit AIX.

2.2.4.1 Définitions de ces types de comportements

L'utilisateur de mail

Il s'agit par exemple d'une secrétaire ou d'un employé de service administratif. Ses activités peuvent être surtout l'utilisation du courrier électronique, le traitement de texte etc. Cet utilisateur peut faire plusieurs connexions par jour ou rester connecté toute la journée avec de longues périodes d'inactivité. Ce comportement est caractérisé comme suit :

[x@ host]/u/perso/si/x>lpr mail1
[x@ host]/u/perso/si/x>lpr mail2
[x@ host]/u/perso/si/x>ls
[x@ host]/u/perso/si/x>rm mail0
[x@ host]/u/perso/si/x> rm mail00
[x@ host]/u/perso/si/x>mail
[x@ host]/u/perso/si/x>lpr mail1
[x@ host]/u/perso/si/x>lpr mail2
[x@ host]/u/perso/si/x>ls
[x@ host]/u/perso/si/x>rm mail1
[x@ host]/u/perso/si/x> rm mail2

Le novice

Ce type d'utilisateur travaille dans un environnement multi-fenêtres, il est limité à l'utilisation du courrier électronique et au développement de petites applications. Ce comportment est caractérisé par l'enchaînement de commandes suivantes :

[x@ host]/u/perso/si/x>xterm
[x@ host]/u/perso/si/x>who
[x@ host]/u/perso/si/x>mail
[x@ host]/u/perso/si/x>ls
[x@ host]/u/perso/si/x>cd /perso/si/x/repertoire
[x@ host]/u/perso/si/x>ls
[x@ host]/u/perso/si/x>cp serveur.c serveur_old.c
[x@ host]/u/perso/si/x>vi serveur.c
[x@ host]/u/perso/si/x>make serveur
[x@ host]/u/perso/si/x>make serveur
[x@ host]/u/perso/si/x>make serveur

[x@ host]/u/perso/si/x>serveur

[x@ host]/u/perso/si/x>make serveur

[x@ host]/u/perso/si/x>serveur

[x@ host]/u/perso/si/x>serveur

[x@ host]/u/perso/si/x>lpr serveur.c

[x@ host]/u/perso/si/x>who

[x@ host]/u/perso/si/x>mail

Le développeur

Celui-ci développe des applications, les outils informatiques utilisés sont essentiellement un éditeur de texte, un compilateur, un debugger, etc. Il utilise aussi des commandes plus développées telles que grep, ftp, make, etc. Ce comportement est caractérisé comme suit :

[x@ host]/u/perso/si/x>xterm

[x@ host]/u/perso/si/x>xterm

[x@ host]/u/perso/si/x>xterm

[x@ host]/u/perso/si/x>who

[x@ host]/u/perso/si/x>mail

[x@ host]/u/perso/si/x>ls

[x@ host]/u/perso/si/x>cd /u/perso/si/x/rep1

[x@ host]/u/perso/si/x>ls

[x@ host]/u/perso/si/x>cd /u/perso/si/x/rep2

[x@ host]/u/perso/si/x>ls

[x@ host]/u/perso/si/x>cp serveur.c serveur_old.c

[x@ host]/u/perso/si/x>vi serveur.c

[x@ host]/u/perso/si/x>vi client.c

[x@ host]/u/perso/si/x>cat serveur.make

[x@ host]/u/perso/si/x>env

[x@ host]/u/perso/si/x>grep –F NBMAXCONNECT –f serveur.c

[x@ host]/u/perso/si/x>make serveur

[x@ host]/u/perso/si/x>make serveur

[x@ host]/u/perso/si/x>make client

[x@ host]/u/perso/si/x>xde serveur

[x@ host]/u/perso/si/x>make serveur

[x@ host]/u/perso/si/x>xde serveur

[x@ host]/u/perso/si/x>client

[x@ host]/u/perso/si/x>make serveurps – edf | grep x

[x@ host]/u/perso/si/x>lpr serveur.c

[x@ host]/u/perso/si/x>lpstat

[x@ host]/u/perso/si/x>ftp

[x@ host]/u/perso/si/x>who

[x@ host]/u/perso/si/x>mail

[x@ host]/u/perso/si/x>xlock

L'utilisateur expérimenté

Cet utilisateur a une très longue expérience du système UNIX. Il communique beaucoup avec l'extérieur et a un compte sur plusieurs machines l'emmenant à utiliser les commandes suivantes : rlogin, rcp, rsh, etc. Il est caractérisé comme suit :

[x@ host]/u/perso/si/x>who

[x@ host]/u/perso/si/x>find . –type f –size +1000 –print

[x@ host]/u/perso/si/x>more serveur.c

[x@ host]/u/perso/si/x>ftp

[x@ host]/u/perso/si/x>chmod 700 toto.c

[x@ host]/u/perso/si/x>vi toto.c

[x@ host]/u/perso/si/x>lpr toto.c

[x@ host]/u/perso/si/x>ps –edf | grep x

[x@ host]/u/perso/si/x>finger @remotehost.xxx.fr

[x@ host]/u/perso/si/x>mail

[x@ host]/u/perso/si/x>who

[x@ host]/u/perso/si/x>rlogin guest @remotehost.xxx.fr

[x@ host]/u/perso/si/x>cd /u/perso/si/x/rep1

[x@ host]/u/perso/si/x>vi memoire.tex

[x@ host]/u/perso/si/x>latex memoire

[x@ host]/u/perso/si/x>xdvi memoire

2.2.4.2 Application de la méthode des profils propres

Appliquons les différentes étapes citées ci-dessus à ces différents utilisateurs :

Le processus d'initialisation

1- les profils générés par ces comportement d'utilisateurs (la base d'apprentissage) sont les suivants :

	L'utilisateur de mail Γ_1	Le Novice Γ_2	Le développeur Γ_3	L'utilisateur expérimenté Γ_4
uscr_login fail				
uscr log (23h to 6h)				
short_Session				
usc_SU OK				
user_SU fail				
who,w,finger,…		2	3	4
more,pg,cat,…		1	3	3
ls OK	2	2	3	
ls fail				
df,hostname,uname				
arp,netstat,ping				
ypcat				
lpr	4	1	1	1
rm, mv	4			
ln				
whoami, id				
rexec,rlogin,rsh			1	2
proc_Execute	16	18	55	17
proc_SetPetri				
file_open fail		4	2	4
file_open fail cp				
file_open .netrc				
file_read lpr				
file_read passwd...				
file_write				
passwd,… fail				
file_write cp ok		1	1	
file_unlink rm	4			
file_mode				1

Tableau 2.2.1.: les profils générés des différents utilisateurs [20].

2- le profil moyen de ces comportements est le suivant (en utilisant l'équation *(2.2.2)*) :

$$\Psi^T=\frac{1}{4}\sum_{i=1}^{4}\Gamma_i^T=(0\ 0\ 0\ 0\ 0\ 2.25\ 1.75\ 1.75\ 0\ 0\ 0\ 0\ 1.75\ 1\ 0\ 0\ 0.75\ 26.5\ 0\ 2.5\ 0\ 0\ 0\ 0\ 0\ 0.5\ 1\ 0.25)$$

3- calculer le vecteur trait (équation *(2.2.3)*) pour chaque utilisateur *(*$\Phi_1,\Phi_2,\Phi_3,\Phi_4$ *)*.

4- calculer la matrice *A* à partir de l'équation *(2.2.5)*.

5- calculer les vecteurs propres de la matrice de covariance $C=\frac{1}{M}\sum_{i=1}^{M}\Phi_i\Phi_i^T=A.A^T$:

ceci peut être fait en utilisant le résultat défini dans l'équation *(2.2.19)*.

avec $A^T.A=\begin{bmatrix} 37.15 & 20.03 & -77.40 & 20.21 \\ 20.03 & 19.65 & -60.28 & 20.59 \\ -77.40 & -60.28 & 204.78 & -67.09 \\ 20.21 & 20.59 & -67.09 & 26.28 \end{bmatrix}$

Les valeurs propres de cette matrice sont $\begin{bmatrix} 273.792 \\ 12.249 \\ 1.833 \\ 0 \end{bmatrix}$

Leurs vecteurs propres correspondants sont

$X_1(273.792)=\begin{bmatrix} -0.329 \\ -0.254 \\ 0.865 \\ -0.282 \end{bmatrix}$, $X_2(12.249)=\begin{bmatrix} -0.786 \\ 0.268 \\ -0.038 \\ 0.556 \end{bmatrix}$, $X_3(1.83)=\begin{bmatrix} 0.157 \\ -0.783 \\ 0.026 \\ 0.601 \end{bmatrix}$

Les vecteurs propres U_k $k=1,..,3$ de la matrice de covariance $C=\frac{1}{M}\sum_{i=1}^{M}\Phi_i\Phi_i^T=A.A^T$ sont obtenus à partir de l'équation *(2.2.19)*.

Dans cet exemple, nous avons considéré un seul profil pour chaque utilisateur. Ainsi, chaque profil correspond exactement à la classe qu'il forme $\Omega^k, k=1,..,4$ (car $NU=1$).

Le tableau suivant présente la distance euclidienne entre les différents utilisateurs après projection (à partir de l'équation *(2.2.20)*).

	User1	**User2**	**User3**	**User4**
User1	0	4.095	19.927	4.795
User2		0	18.577	2.179
User3			0	19.111
User4				0

Tableau 2.2.2.: la distance euclidienne entre les différentes classes.

Si nous essayons de représenter les quatre profils dans le nouvel espace engendré par les profils propres, leur projection est représentée dans la figure *(2.2.2)* suivante.

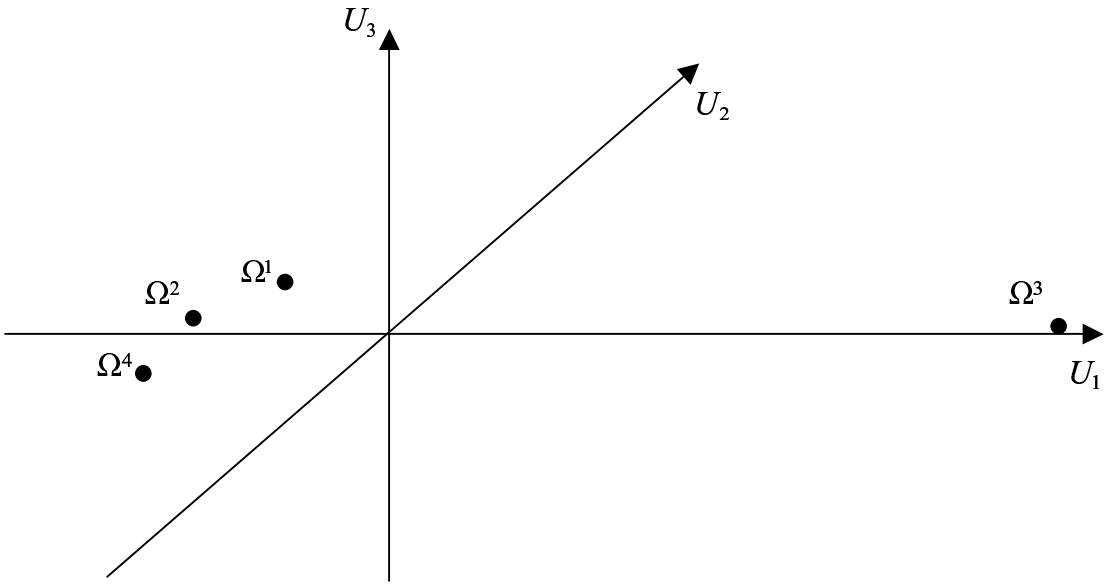

Figure 2.2.2.: La projection des profils des utilisateurs sur le nouvel espace des profils propres.

2.2.5 Application de cette méthode sur un fichier web log

Ce paragraphe présente l'ensemble des tests pour mettre en œuvre le modèle de l'ACP et son application sur un fichier web log réel.

Nous avons utilisé un fichier log (access.log) généré par SQUID qui est un logiciel de cache (web proxy cache) gratuit et open source [23]. Ce software est installé sur un pare-feu (firewall) reliant un réseau local contenant 6 machines d'utilisateurs qui sont des stagiaires à l'ENST-Bretagne.

Nous avons audité ces utilisateurs pendant un mois et effectué des tests sur ce le fichier access.log ainsi généré qui est d'une taille d'environ 43 Méga Octets. Chaque entrée dans ce fichier a dix champs ayant le format suivant :

[Timestamp] [Elapsed-Time] [Client-IP] [Action]/[Code] [Size] [Method] [URL] [Hierarchy]/[Content Type]

Avec:

- Timestamp : L'horodatage indiquant l'heure de la requête, exprimé en seconde à partir du 1er janvier 1970 et avec une résolution en millisecondes.
- Elapsed Time : Ce champs indique le temps écoulé de la requête en millisecondes.
- Client-IP : L'adresse IP du client.
- Action : Ce champs décrit comment la requête a été traitée localement dans le serveur proxy.
- Code : Le code d'état envoyé au client comme réponse à sa requête.

- Size : Ce champs enregistre le nombre d'octets transférés du proxy vers le client pour une requête donnée.
- Method : La méthode http demandé par le client.
- URL : L'URL du document demandé.
- Hierarchy : Ce champs décrit où et comment le document est cherché.
- Content : Le type du document spécifié.

Exemple:

985796546.734 1 192.168.xx.xx TCP_MISS/200 207 GET http://good.niagara.sightpath.com/images/homepage/smb-job-npm.gif - NONE/-image/gif

Avant de passer à la phase d'identification et de détection, le système doit acquérir un ensemble d'informations pour accomplir cette tâche. C'est ce qu'on appelle "*la phase d'apprentissage*". Son rôle est de fixer les paramètres du système pour fournir les meilleurs résultats. C'est la tâche la plus importante et la plus délicate.

Elle est importante car les performances du système dépendent directement du choix de ces paramètres. Elle est délicate car il n'existe ni de formules ni de méthodes directes permettant la détermination de leurs valeurs, il faut les calculer par expérience.

Afin d'utiliser l'ACP, la matrice de corrélation doit être calculée. Pour ce faire, chaque élément du vecteur de profil Γ_i détermine le nombre de requêtes pour une URL donnée pendant une session d'audit (dans notre cas, une session consiste en une journée pendant la période d'activité des utilisateur (de 08h00 à 19h00)). Ainsi, la taille du vecteur profil Γ_i dépend du nombre d'URL visitées par l'ensemble des utilisateurs choisis pour l'apprentissage durant une certaine période.

Dans notre expérimentation, nous avons choisis trois clients pour tester notre méthode. Le nombre de profils audités choisis pendant un mois était de 48 profils, soit 16 profils pour chaque utilisateur. Le reste des profils n'est pas sélectionné car ils correspondent à des journées de non activité (tel que les week- end et jours d'absences de ces utilisateurs).

Au départ, nous avons divisé cette base de profils en deux classes. La première, utilisée pour la phase d'apprentissage, est composée de 24 profils de trois utilisateurs représentant leurs profils durant les deux premières semaines, soit 8 profils par utilisateur.

Les 24 profils restants de ces utilisateurs constituent la seconde classe. Cette dernière sert à mesurer la capacité de généralisation du système.

Le tableau 2.2.3 récapitule les résultats obtenus :

	Ensemble appris	Ensemble non appris
Identification avec succès	17/24(70.83%)	10/24 (41,67%)
Mauvais rejet (faux positif)	7/24(29.17%)	14/24(58,33%)

Tableau 2.2.3.: Les performances du système avec la première expérimentation.

Ce tableau montre que les résultats obtenus ne sont pas intéressants, d'ailleurs 29,17% des profils de l'ensemble familier ne sont pas reconnus. La cause de ces faux positifs est dû en réalité aux changements des comportements des utilisateurs avec le temps.

Ceci nous a poussé à utiliser une autre stratégie qui consiste à auditer les utilisateurs pendant une période (nous avons choisi cette période égale à 4 jours successifs), ensuite nous introduisons les profils des utilisateurs de la journée qui suit pour la détection. Nous avons gardé trois utilisateurs pour la phase d'apprentissage. Au fur et à mesure que nous avançons d'une journée, la base d'apprentissage est mise à jour en considérant les profils des 3 utilisateurs pendant les quatre derniers jours d'audit choisis précédant la détection sauf dans le cas où le profil réel de l'utilisateur est détectée comme anormal. Ceci nous rappelle l'approche statistique de DENNING qui détermine, à partir de n observations $x_1, x_2, \ldots, x_n$ d'une variable aléatoire x, si une nouvelle observation x_{n+1} est anormale par rapport aux n observations précédentes.

Dans notre cas, chaque observation représente le profil d'un utilisateur audité pendant une journée.

Le tableau 2.2.4 résume les différents résultats obtenus en utilisant cette stratégie.

	Ensemble appris	Ensemble non appris
Identification avec succès	(100%)	34/36 (94,44%)
Mauvais rejet (faux positif)	0%	2/36(5,56%)

Tableau 2.2.4.: Les performances du système avec la deuxième stratégie.

Dans ce deuxième tableau, nous avons testé notre système sur 36 profils, les douze premiers profils sont utilisés pour le premier apprentissage (4 profils par utilisateur).

Les capacités du système sur le fichier log

En utilisant un apprentissage régulier au fur et à mesure de l'évolution des profils des utilisateurs, la méthode n'a trouvé aucun problème pour reconnaître les profils qui lui ont été présentés. Concernant sa généralisation, elle a pu identifié 34 nouveaux profils sur 36.

Il est à noter que cette méthode peut facilement détecter les intrus de type masquerador [2]. D'ailleurs, si un utilisateur change de poste et garde toujours le même profil alors son vecteur trait sera proche de

sa vraie classe. Dans ce cas, il suffit juste de vérifier l'adresse IP de cette classe pour savoir de quel utilisateur il s'agit. Ainsi, cet utilisateur est détecté comme masquerador.

Cependant, le temps de traitement pour la phase d'apprentissage varie selon la taille du fichier de log audité pendant les 4 jours d'apprentissage. Dans notre cas, il varie entre 2 à 3 secondes en moyenne. Le temps de détection et d'identification est de l'ordre de la dixième de seconde car la détection consiste juste en la projection du nouveau profil sur la base des profils propres et le calcul de distance entre son vecteur trait et les différentes classes (3 classes pour les 3 utilisateurs).

2.2.6 Conclusion sur la méthode des profils propres (eigenprofiles)

En résumé, cette nouvelle méthode apporte les idées suivantes :

- Elle introduit une nouvelle méthode de détection d'intrusion dans l'approche comportementale qui permet une bonne classification des différents utilisateurs sous Unix et elle a démontré sa capacité d'apprentissage de profils et de généralisation en utilisant les fichiers log.
- Elle présente une solution simple à exploiter pour le problème de détection d'intrusion en utilisant l'analyse en composantes principales
- Les premiers résultats expérimentaux réalisés sur un fichier log généré par SQUID sont très intéressants.

Conclusion et Perspectives

Ce rapport présente un travail préliminaire dans le domaine de la détection d'intrusion. L'étude des différentes méthodes de détection d'intrusion nous a fait découvrir un nouveau domaine qui suscite beaucoup de curiosités scientifiques.

La première contribution de ce rapport consiste à l'amélioration de GASSATA qui était utilisée dans le projet MIRADOR[1] avec sa première version.

La seconde contribution est une nouvelle méthode basée sur la théorie de l'information qui est introduite dans le domaine de détection d'intrusion. Les premiers résultats expérimentaux de cette méthode ont montré qu'elle est très utile pour la classification des différents utilisateurs d'un système unix et elle est très efficace pour l'apprentissage et ensuite la généralisation des différents profils utilisateurs d'un fichier log réel généré par SQUID. Cependant, elle présente un inconvénient dans la phase d'apprentissage qui consiste à définir, par expérience, un seuil pour chaque classe comme présenté dans le processus de détection et d'identification. Ainsi, pour chaque apprentissage, qui est d'ailleurs effectué d'une manière régulière, il faut définir un seuil pour chaque client (classe) ce qui sera une tâche délicate dans le cas où on a une centaine de machines. Notre système reste alors ouvert et enrichissable. Ainsi, comme solution au problème du seuil pour chaque classe, nous nous proposons l'utilisation des réseaux de neurones pour déterminer à quelle classe appartient le nouvel utilisateur audité. La couche d'entrée de ce réseau contient autant d'unités que de profils propres. Cette couche reçoit les vecteur traits des utilisateurs et la couche de sortie contient autant d'unités que de classes (utilisateurs ou clients). Pour la détection, l'utilisateur qui vient d'être audité sera affecté à la classe correspondante au neurone de sortie ayant la plus grande valeur d'activation.

La méthode des profils propres est en cours de mise en œuvre pour l'ajouter comme un nouveau composant dans l'architecture du projet MIRADOR[1].

La capacité des profils propres à apprendre les comportements des utilisateurs d'une manière régulière est prometteuse pour l'avenir. Nous espérons qu'elle pourra être étendue plus largement à des domaines divers et nous souhaitons la tester sur des bases de plusieurs centaines de méga octets.

---

[1] PEA de la DGA/CELAR/CASSI piloté par Alcatel et impliquant l'ENSTB, l'ONERA et Supelec

Bibliographie

1. R. Heady, G. Luger, A. Maccabe, and M. Servilla. An Architecture of a Network Level Intrusion Detection System. Technical Report, Department of Computer Science, University of New Mexico, August 1990.

2. J. P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, James. P. Anderson Co., Fort Washington, Pennsylvania, April 1980.

3. D. Denning. An Intrusion Detection Model, IEEE Transactions on Software Engineering, Vol. 13 (2), 1987, pp. 222,232.

4. H. Debar, M. Becker, and D. Siboni. A neural network component for an intrusion detection system. Proceedings of the 1992 IEEE Symposium. On Research in Computer Security and Privacy, Oakland, CA, May 1992, pp. 240,250.

5. H. Debar. Application des réseaux de neurones à la détection d'intrusion sur les systèmes informatiques. Université de Paris 6, Thèse de Doctorat, June 22nd 1993.

6. H. Debar and B. Dorizzi. An application of a Recurrent Network ton an Intrusion Detection System, Proc. of the International Joint Conference on Neural Networks, Baltimore, MD, June 1992, 478 – 483.

7. S. Forrest, S.A Hofmeyr, A. Somayagi, and T. A. Longstaff. A Sense of Self for Unix Processes. In Proceedings of the IEEE Symposium on Security and Privacy, pp. 120,128, 1996.

8. D. Anderson, T. Frivord, A. Tamaru, A.Valdes. NIDES: Next Generation Intrusion Detection Expert System. Software Users Manual, Beta-Update Release, SRI International, December 1st 1994.

9. U. Lindqvist. On the Fundamentals of Analysis and Detection of Computer Misuse. PhD Thesis, Chalmers University of Technology, Göteborg, Sweden 1999.

10. U. Lindqvist and P.A. Porras. Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (B-BEST). Proceeding of the IEEE Symposium on Security and Privacy, 1999.

11.C. Dousson . Suivi d'évolutions et reconnaissance de chroniques. Université Paul Sabatier, Thèse de Doctorat, 1994.

12. Y. Debroise, NePoSe. Détection d'intrusion base sur la politique de sécurité. Rapport de Stage de DESS ENST-Bretagne/IFSIC, 2001.

13. K. Ilgun, R. A. Kemmerer, and P. A. Porras. State transition analysis: A rule-based intrusion detection approach. IEEE Transactions on Software Engineering, 21(3) 181-199, March 1995.

14. K. Ilgun. USTAT: A Real-Time Intrusion Detection System for UNIX. Master's Thesis, Computer Science Department, University of California, Santa Barbara, July 1992.

15. S. Kumar. Classification and Detection of Computer Intrusions. PhD Thesis, Purdue University, West Lafayette, Indiana, August 1995.

16. S. Kumar and E. H. Spafford. An application of pattern matching in intrusion detection. Technical Report CSD-TR-94-013, the COAST Project, Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907-1398, June 17th 1994.

17. S. Kumar and E. H. Spafford. A pattern matching model for misuse intrusion detection. In Proceedings of the 17th National Computer Security Conference, Baltimore MD, 1994, pp. 11,21. NIST, National Institute of Standards and Technology/ National Computer Security Center.

18. S. Kumar and E. H. Spafford. A Software architecture to support misuse intrusion detection. Technical Report CSD-TR-95-009, the COAST Project, Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907-1398, March 17th 1995.

19. CERT Adivisories. http://www.cert.org

20. L. Mé. Audit de Sécurité par Algorithmes Génétiques, Université de Rennes 1. Thèse de doctorat, N° d'Ordre 1069, 7 Juillet 1994.

21. M. Gen, R. W. Cheng: Genetic Algorithms and engineering design, John Wiley and Sons, Inc., 1997.

22. I. T. Jolliffe. Principal Component Analysis. New York: Springer Verlag, 1986.

23. http://www.squid-cache.org/

Annexe

La matrice attaque événements utilisée est la suivante [20] :

	Type d'attaque																						
	a_2	l_1	l_2	l_3	l_4	a_3	a_4	a_5	l_5	l_6	l_7	l_8	a_6	a_7	a_8	a_9	l_9	l_{10}	l_{11}	a_{10}	l_{12}	l_{13}	l_{14}	l_{15}
user_login fail	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
user log (23h to 6h)	.	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
short_Session	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
usc_SU OK	.	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
user_SU fail	.	.	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
who,w,finger,…	.	3	.	.	.	.	.	.	8	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
more,pg,cat,…		.	.	.	.	5	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	5	.	.
ls OK	.	.	.	.	.	30	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
ls fail	.	.	.	.	.	.	5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
df,hostname,uname	.	.	.	.	.	.	.	.	.	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.
arp,netstat,ping	.	.	.	.	.	.	.	.	.	.	2	.	.	.	.	.	.	.	.	.	.	.	.	.
ypcat	.	.	.	.	.	.	.	.	.	.	.	3	.	.	.	.	.	.	.	.	.	.	.	.
lpr	.	.	.	.	.	.	.	.	.	.	.	.	.	.	10	1	.	.	.	.	.	.	.	.
rm, mv	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.
ln	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.
whoami, id	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	4
rexec,rlogin,rsh	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.
proc_Execute	.	3	.	.	.	35	5	.	8	3	2	3	.	.	10	3	.	300	.	2	.	5	.	4
proc_SetPetri	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	100	.	.	.	.	.
file_Open fail	.	.	.	.	.	.	.	5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
file_Open fail cp	.	.	.	.	.	.	.	.	.	.	.	.	.	10	.	.	.	.	.	.	.	.	.	.
file_Open .netrc	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.	.	.	.
file_Read lpr	.	.	.	.	.	.	.	.	.	.	.	.	.	.	10	.	.	.	..	.	.	.	.	.
file_Read passwd...	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	5	.	.
file_write passwd,… fail	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.
file_write cp Ok	.	.	.	.	.	.	.	.	.	.	.	.	30	.	.	.	.	.	.	.	.	.	.	.
file_Unlink rm	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	50	.	.	.	.	.	.	.
file_mode	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3	.	.	.