MASTER EN SCIENCES INFORMATIQUES

La détection des intrusions
Une application pour DECnet Phase IV

Parmentier, Laurence

Award date:
1994

Awarding institution:
Universite de Namur

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

La détection des intrusions : Une application pour DECnet Phase IV

Laurence Parmentier

Promoteur : Philippe van Bastelaer

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

Année académique 1993-1994

Résumé

Les problèmes relatifs à la sécurité des systèmes informatiques se trouvent de plus en plus sous les feux de l'actualité. Un nombre croissant de personnes prennent conscience de la nécessité de se protéger contre les violations de sécurité et les menaces qu'elles représentent. Dans ce travail, nous aborderons les problèmes de détection d'intrusions. L'audit de sécurité semble être une solution attrayante à ces problèmes. Etant donné la quantité d'informations rassemblées lors de l'audit, il est intéressant de disposer d'outils automatisés qui facilitent l'analyse des ces données. Nous étudierons plusieurs outils d'analyse automatisés. Nous nous pencherons ensuite plus particulièrement sur une solution qui a été retenue pour la détection des intrusions au CERN, le Laboratoire Européen pour la physique des particules.

Abstract

Computer security problems receive a growing attention. More people are getting aware of the necessity to protect themselves from security violations and the associated threats. In this paper we will approach the intrusion detection's problems. Security auditing seems to be an attractive solution for these problems. Given the amount of data collected by auditing, it is helpful to have automated tools to assist in analysing it. We will study some automated tools. Then, we will concentrate on a CERN's solution for detection of intrusions.

Remerciements

Nous tenons à remercier

Monsieur Philippe van Bastelaer
qui nous a donné la possibilité de réaliser notre stage au CERN
et qui a enrichi ce mémoire par ses précieux conseils ;

Monsieur John Gamble du CERN
qui a guidé notre travail pendant les six mois de notre stage ;

Toutes les personnes rencontrées au CERN
qui nous ont si gentiment aidés et conseillés ;

Toutes les personnes qui ont participé à la réalisation de ce mémoire,
entre autres, par leurs nombreuses relectures.

Table des matières

Introduction

La sécurité des systèmes informatiques devient de plus en plus importante. Le nombre d'incidents relatés dans la presse en témoigne. Il est donc nécessaire de se protéger contre les violations de sécurité et les menaces qu'elles représentent.

Dans ce travail, nous aborderons les problèmes de détection d'intrusions. Nous verrons ce qu'est l'audit de sécurité et pourquoi cela semble être une solution attrayante pour la détection des intrusions. Nous nous pencherons ensuite plus particulièrement sur une solution qui a été retenue au CERN, le Laboratoire Européen pour la physique des particules.

Dans le premier chapitre, nous présenterons le CERN et nous positionnerons le problème, à savoir la détection des activités suspectes observées sur les lignes DECnet qui entrent et sortent du CERN.

Dans le deuxième chapitre, nous nous intéresserons à l'audit de sécurité, principalement dans le domaine de la détection des intrusions, ainsi qu'à certains systèmes de détection d'intrusions existants.

Dans les chapitres suivants, nous aborderons une solution retenue par CERN.

Dans le troisième chapitre, nous décrirons l'environnement de travail, à savoir DECnet, et en particulier DECnet Phase IV. Nous consacrerons également quelques pages à la plus récente version de DECnet, DECnet Phase V.

Dans le quatrième chapitre, nous présenterons le travail réalisé dans le cadre de notre stage au CERN. Nous expliquerons la solution en détail.

Enfin, nous terminerons ce travail en suggérant quelques généralisations qu'il serait intéressant d'apporter au système mis en place. Peut-être pourrions-nous envisager l'application du programme à d'autres protocoles que DECnet ?

Mais commençons par présenter le CERN ainsi que le problème qui nous fut posé.

Chapitre 1

Positionnement du problème

Nous allons commencer ce travail par une description du contexte dans lequel nous avons été plongés lors de notre stage. Ainsi, nous décrirons brièvement les activités du CERN ainsi que ses importants moyens de communication. Nous exposerons ensuite plus en détail le travail qu'il nous a été demandé de réaliser et le projet dans lequel s'intègre ce dernier. Nous terminerons ce premier chapitre par une évaluation des intérêts du projet.

1.1. Le CERN

Après avoir exposé brièvement l'histoire et les fonctions du CERN, nous nous attarderons quelques instants sur la structure de son réseau informatique.

1.1.1. Qu'est-ce que le CERN ?

Le CERN, le Laboratoire Européen pour la physique des particules, est l'un des plus grands laboratoires scientifiques du monde. Créé en 1954 par douze Etats d'Europe Occidentale, il rassemble l'essentiel du potentiel européen de recherche dans le domaine de la physique des particules. C'était d'ailleurs là l'une des principales raisons de sa création : après la guerre, beaucoup de chercheurs européens ont émigré vers les Etats-Unis où les laboratoires étaient très modernes et bien équipés, contrairement aux laboratoires européens. C'est essentiellement pour les retenir en Europe que les douze Etats ont réunis leur ressources et créé le CERN. Dès ses débuts, le CERN a contribué à rétablir des liens amicaux entre des pays que la guerre avait dressés les uns contre les autres.

Les pays membres

Situé à cheval sur la frontière franco-suisse, comme le montre la figure n°1.1, le CERN comprend aujourd'hui dix-neuf états membres : l'Allemagne, l'Autriche, la Belgique, le Danemark, l'Espagne, la Finlande, la France, la Grèce, la Hongrie, l'Italie, la Norvège, les Pays-Bas, la Pologne, le Portugal, la Slovaquie, la République Tchèque, la Suède, la Suisse et le Royaume Uni. Ces pays participent au financement du CERN en fonction de leur produit national brut.

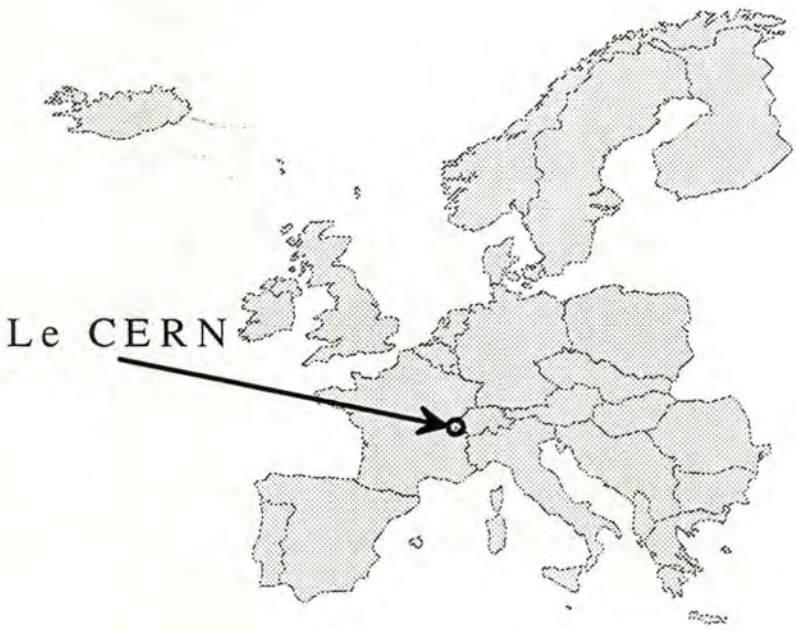

Figure n°1.1 : Position du CERN en Europe

Ses activités

Le CERN a pour but l'étude de la métamorphose de l'énergie en matière et de tout ce que cette métamorphose met en évidence, par exemple la structure de la matière dans l'infiniment petit et les forces les plus fondamentales qui agissent sur la Nature. Aujourd'hui, plus de 4000 chercheurs d'Europe et d'autres continents travaillent au CERN et, groupés en équipes multinationales, y conduisent des dizaines d'expériences, toutes non-militaires, auxquelles sont aussi associés de nombreux laboratoires nationaux.

Le CERN lui-même emploie relativement peu de chercheurs en physique, mais il met des installations d'expérimentation à la disposition des chercheurs de tous pays. La principale de ces installations est le collisionneur électron-positon LEP (*Large Electron Positron*), cet anneau souterrain de 27 kilomètres de circonférence dans lequel des faisceaux d'électrons et des faisceaux de positons sont lancés en sens opposés à une vitesse proche de

celle de la lumière et sont projetés les uns contre les autres. Le LEP est la plus grande machine de ce genre au monde. Quatre puissants détecteurs sont disposés sur l'anneau pour arrêter, enregistrer et analyser les fragments de matière émergeant des collisions. La plupart des personnes employées par le CERN, venant des 19 pays membres, sont donc chargées de fournir aux physiciens le support dont ils ont besoin.

Cette coopération internationale requiert des installations de réseaux informatiques pour permettre notamment le développement d'applications distribuées, l'analyse des données par les chercheurs qui sont retournés dans leur pays ou le contrôle à distance des expériences. Nous allons donc maintenant nous intéresser aux réseaux du CERN.

1.1.2. Les réseaux du CERN

Nous pouvons distinguer deux réseaux au CERN : le réseau externe et le réseau interne. Le réseau externe relie le CERN à l'extérieur alors que le réseau interne relie entre elles les différentes machines du CERN.

Nous présenterons d'abord la structure du réseau externe puis nous aborderons la structure du réseau interne. Enfin, nous indiquerons le lien qui existe entre le réseau externe et le réseau interne.

1. Le réseau externe

Nous avons déjà vu que l'installation d'un réseau qui permet la communication entre le CERN et l'extérieur est nécessaire pour assurer la coopération entre les différents partenaires. Il est aussi indispensable que ce réseau utilise des technologies très performantes. Illustrons notre propos par un exemple. Les expériences du LEP génèrent chaque année 25 térabytes de données physiques qui doivent être analysées à distance. Les transmettre via une ligne à un mégabit par seconde qui travaille à pleine vitesse nécessiterait sept années. Il est donc nécessaire d'avoir des lignes plus rapides. Signalons toutefois que seule une petite partie de ces 25 térabytes est transmise via le réseau externe. Cette partie représente environ 2 térabytes par an. L'autre partie des données est envoyée sur bandes magnétiques par camion ou par avion.

Les deux lignes de conduite pour le développement du réseau externe

Le CERN a donc développé son réseau externe selon deux lignes de conduite complémentaires :

- En collaboration avec d'autres instituts de physique, un ensemble de lignes dédiées ont été installées vers certaines destinations clés, cela dans le but de répondre à des exigences spécifiques. Il s'agit du réseau HEPnet (*High Energy Physics Network*).

- En plus de cela, des connexions performantes reliant le CERN à des réseaux généraux d'éducation et de recherche ont été mises en oeuvre.

a) HEPnet

La configuration actuelle de HEPnet est basée sur des lignes louées, formant principalement une étoile autour du CERN. Ces lignes sont financées par les organisations reliées à ce réseau. Le tableau n°1.1 présente les différentes lignes du réseau HEPnet qui passent par le CERN, en précisant la destination de ces lignes, leurs vitesses et les protocoles de communication qu'elles supportent.

Destination	Vitesse	Protocole(s)
Barcelone	64 Kbps	DECnet
Bologne	2 Mbps	DECnet, IP, X.25
Bombay	9,6 Kbps	IP
Bombay	64 Kbps	NJE sur IP
Budapest	9,6 Kbps	IP
Cracovie	9,6 Kbps	IP
Le laboratoire DESY en Allemagne	768 Kbps	IP, DECnet, X.25
Lyon	2 Mbps	DECnet, IP, X.25
MIT	256 Kbps	DECnet, IP
Villigen	64 Kbps	DECnet
Saclay	256 Kbps	DECnet, IP, X.25
Zurich	64 Kbps	SNA
Le laboratoire RAL en Angleterre	256 Kbps	SNA, DECnet, X.25

Tableau n°1.1 : Les lignes du réseau HEPnet

Les exigences auxquelles le réseau HEPnet est appelé à répondre, particulièrement grâce à ses lignes rapides, sont :

- une large bande passante garantie et déterminée entre deux sites spécifiques ;
- des temps de latence courts et déterminés (typiquement 10/30 ms en Europe) ;
- l'indépendance des protocoles de communication ;

- une pleine flexibilité pour diviser au mieux la bande passante disponible, en particulier pour les services innovateurs (par exemple, les applications multimédia, les expériences pilotes d'ATM,...).

b) Participation aux réseaux généraux d'éducation et de recherche

La participation du CERN à des réseaux généraux d'éducation et de recherche est justifiée par le fait que le CERN, en tant que centre scientifique et technique, doit avoir d'excellentes connexions avec le reste de la communauté scientifique. Cela permet aussi à des instituts de recherche en physique plus petits ou plus éloignés, ne faisant donc pas partie de HEPnet, d'accéder au CERN.

La participation du CERN

Historiquement, le CERN a été un noeud important de EARN ; c'est aussi l'ancien noeud principal de EASInet, le réseau résultant de l'initiative EASI d'IBM, qui fournit une connexion directe à une vitesse de 1,5 Mbps entre le CERN et le réseau NSFnet des Etats Unis. Le CERN fait également partie des dorsales européennes Ebone et EuropaNET. Par ailleurs, il est directement connecté aux réseaux nationaux de ses deux pays hôtes, RENATER en France et SWITCH en Suisse. De plus, le CERN est relié à Internet. Enfin, le CERN dispose également d'une ligne louée de 1 Mbps le reliant à Berne. Le tableau n°1.2 reprend l'ensemble des lignes louées qui relient le CERN à ces réseaux. Pour chacune, nous mentionnerons le réseau, la destination, la vitesse et les protocoles de communication.

Réseau	Destination	Vitesse	Protocole(s)
Ebone	Athènes	9,6 Kbps	IP
EuropaNET	Berne	1 Mbps	IP sur PPP
SWITCH	Genève	2 Mbps	IP
SWITCH	Lausanne	2 Mbps	IP
EuropaNET	Washington	1,5 Mbps	IP
Ebone	Paris	2 Mbps	IP
Ebone	Amsterdam	1,5 Mbps	IP
Ebone	Tel-Aviv	64 Kbps	NJE sur IP, IP
EARN	ONU à Genève	9,6 Kbps	NJE sur BSC

Réseau	Destination	Vitesse	Protocole(s)
Ebone	Vienne	256 Kbps	IP
Internet	O.M.S. à Genève	64 Kbps	IP

Tableau n°1.2 : Le CERN et les réseaux généraux d'éducation et de recherche

c) HEPnet et les réseaux généraux d'éducation et de recherche

La figure n°1.2 donne une vue globale des endroits vers lesquels le CERN possède des lignes louées.

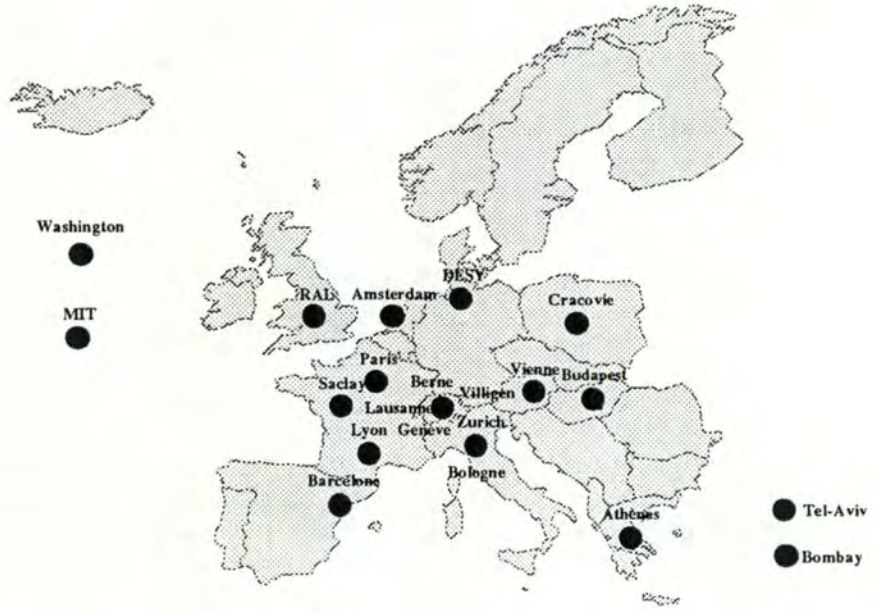

Figure n°1.2 : Les destinations des lignes louées du CERN

Au total, le réseau HEPnet et les connexions aux réseaux d'ordre général fournissent une bande passante agrégée de 15 Mbps autour du CERN ; le trafic mensuel moyen transitant sur ces connexions s'élève à 800 gigabytes.

Remarquons que ces lignes utilisent différents protocoles de communication, principalement IP et DECnet.

La figure n°1.3 nous montre la richesse du réseau externe du CERN. Les informations présentées sur cette figure ne correspondent pas exactement à celles proposées précédemment. Ceci est dû à la rapide évolution des lignes du CERN. Les informations exposées auparavant tiennent compte des dernières évolutions du réseau, alors que les informations de la figure présentent la situation telle qu'elle était en octobre 1993.

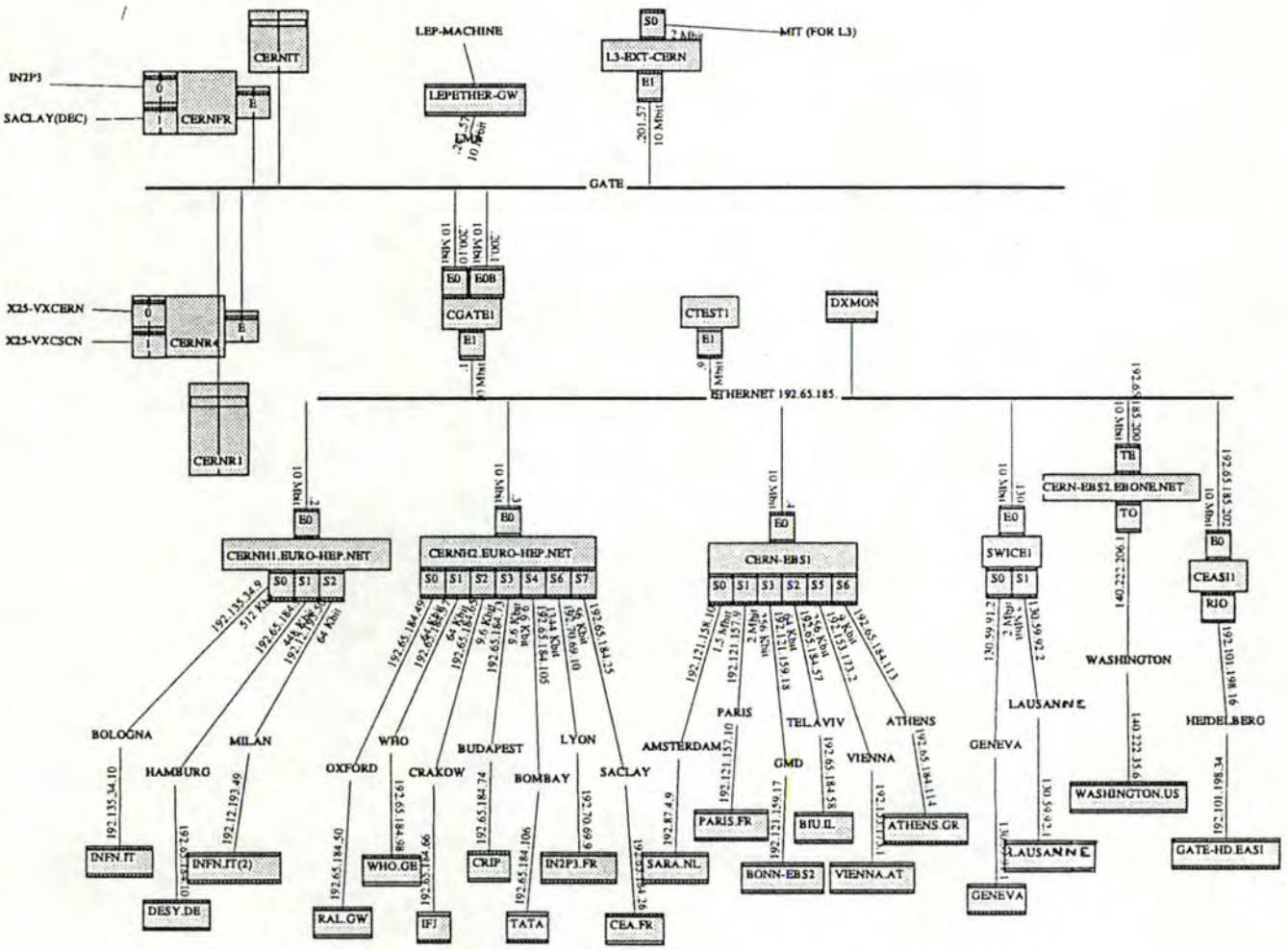

Figure n°1.3 : Réseau externe du CERN

2. Le réseau interne

Alors que le réseau externe permet la communication entre le CERN et l'extérieur, le réseau interne rend possible la communication au sein du CERN. Voyons comment se caractérise ce réseau interne.

Globalement, le réseau interne du CERN est composé d'un ensemble de réseaux Ethernet reliés par une dorsale en fibre optique utilisant le standard FDDI. Les réseaux Ethernet atteignent des vitesses de 10 Mbps alors que la vitesse de la dorsale peut s'élever à 100 Mbps. Actuellement, plus de 4000 machines sont connectées aux réseaux Ethernet et ce nombre continue à croître.

La figure n°1.4 donne une schématisation de la structure du réseau interne.

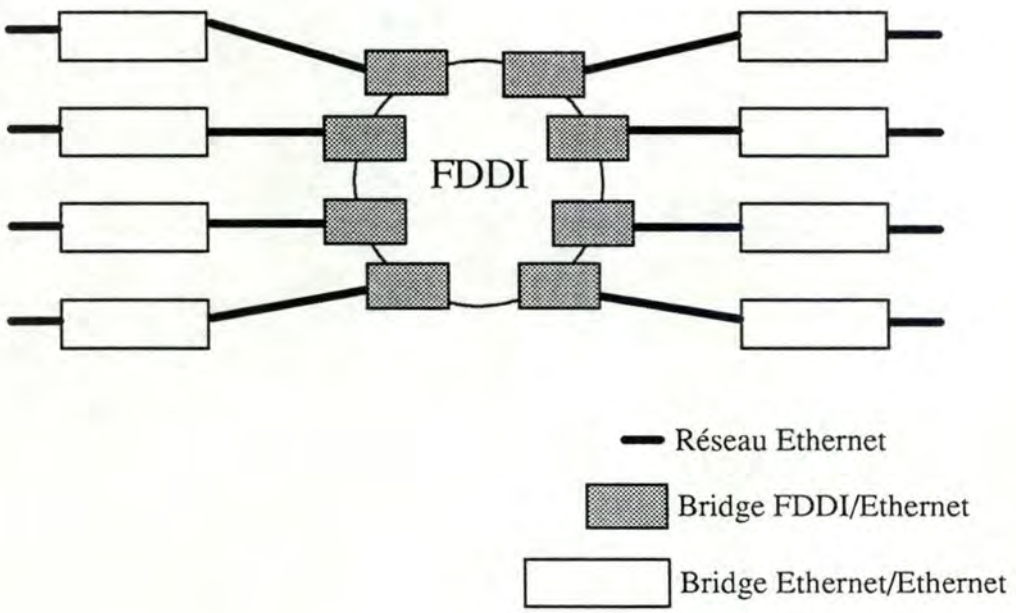

Figure n°1.4 : Structure du réseau interne du CERN

Les différents réseaux Ethernet sont connectés par des bridges ; il n'y a donc pas de restriction sur le flux du trafic du réseau, chaque machine pouvant, en principe, communiquer avec n'importe quelle autre machine.

Plusieurs protocoles de communication sont utilisés sur ces réseaux locaux, notamment TCP/IP, DECnet, SNA, X.25, Appletalk, Domain. De même, différents types de machines sont interconnectés ; citons par exemple des stations Vax, des Macintosh, des stations DEC, des stations SUN, ... Et ces machines utilisent différents systèmes d'exploitation.

Les réseaux locaux du CERN sont donc caractérisés par leur hétérogénéité.

La figure n°1.5 nous montre la complexité du réseau interne du CERN. La structure du réseau est beaucoup plus compliquée en réalité que la structure simplifiée que nous avons présentée précédemment. En fait, plusieurs anneaux en fibre optique permettent de relier plusieurs réseaux Ethernet.

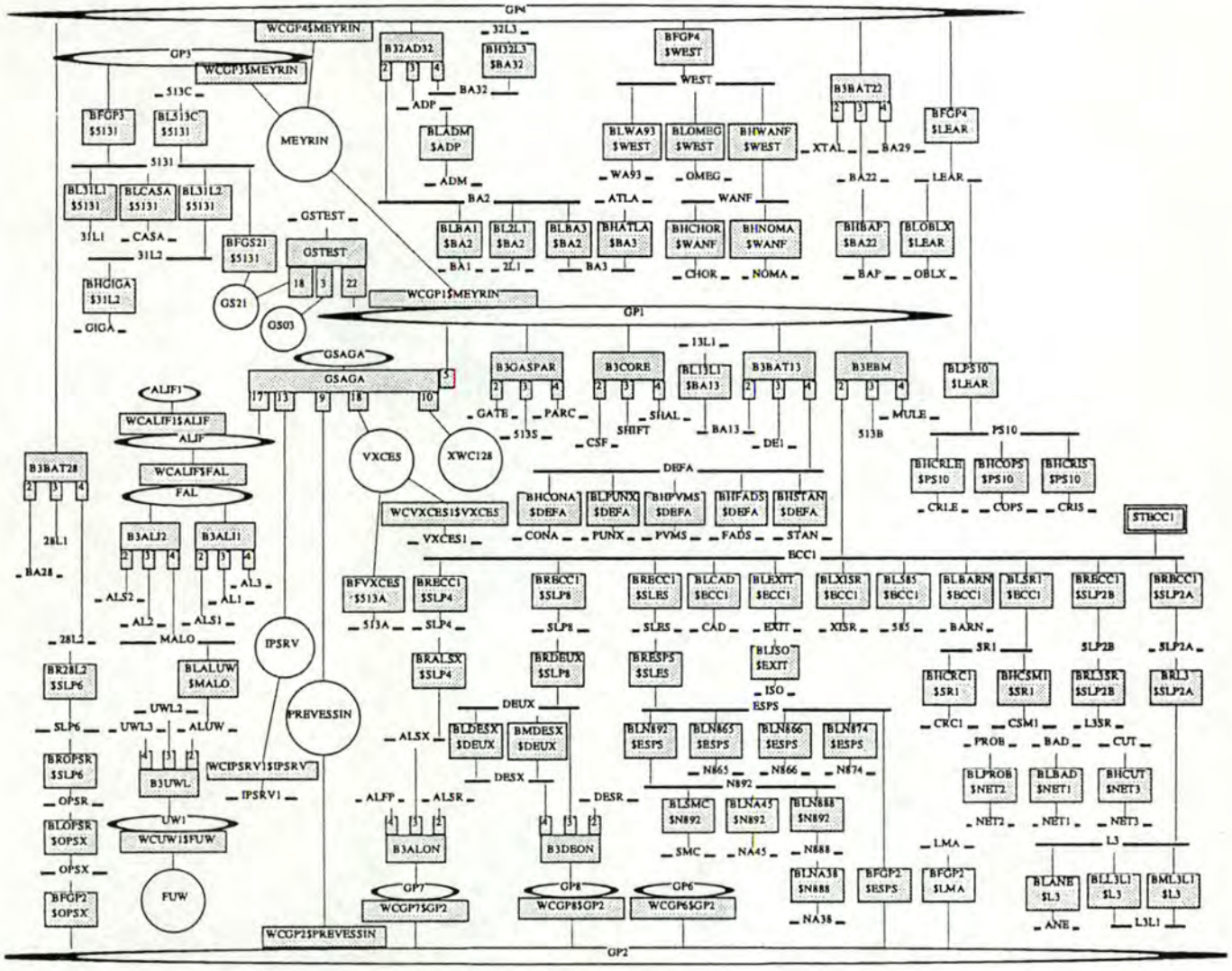

Figure n°1.5 : Réseau interne du CERN

3. Le lien entre le réseau interne et le réseau externe

Ces deux réseaux ne sont pas isolés. Ainsi, le trafic en provenance du CERN et destiné à l'extérieur passe d'abord par le réseau interne. De même, le trafic en provenance de l'extérieur à destination du CERN termine son trajet sur le réseau interne. Voyons comment cela se passe.

Après être passé via des routeurs, le trafic transitant sur les lignes externes aboutit au CERN sur un réseau Ethernet interne. Dorénavant, nous appellerons ce réseau Ethernet "dorsale du trafic externe". Cette dorsale regroupe tous les routeurs qui envoient ou reçoivent du trafic de l'extérieur.

La figure n°1.6 nous présente le lien qui existe entre le réseau externe et le réseau interne. Nous n'avons représenté ici que les routeurs DECnet, ceci dans un but de clarté.

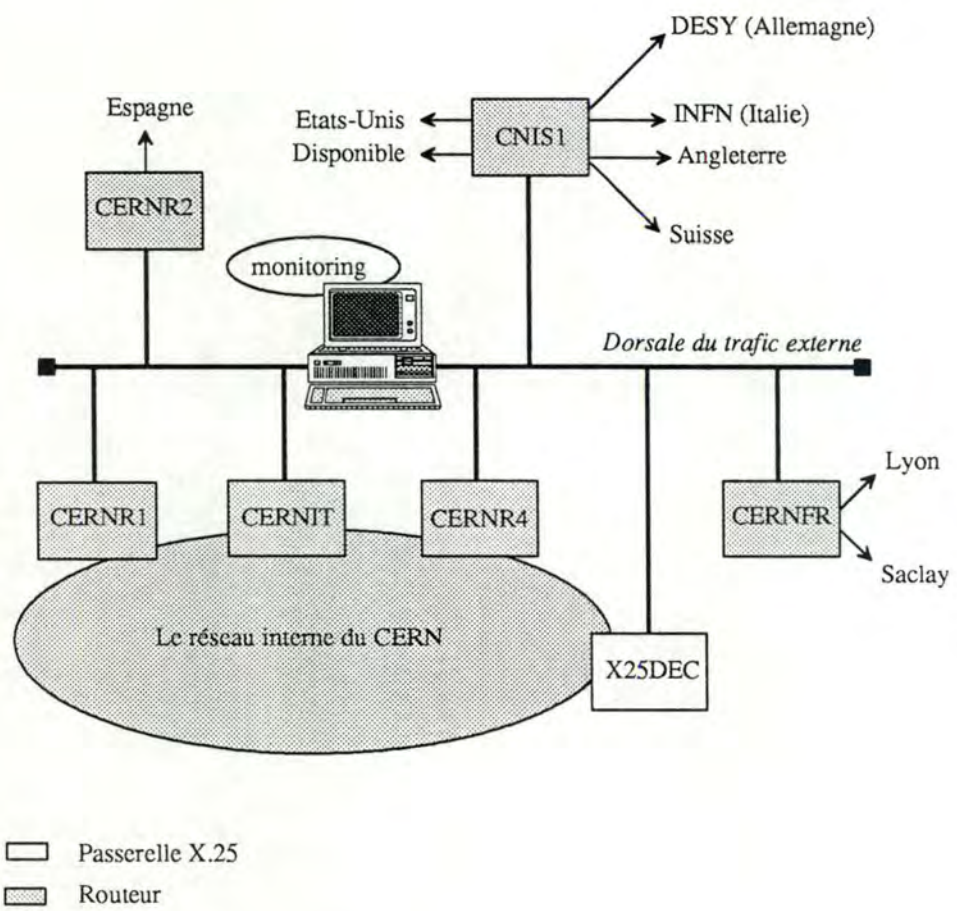

Figure n°1.6 : Lien entre le réseau externe et le réseau interne du CERN

1.2. Le projet

Le contexte étant posé, nous pouvons à présent présenter le projet auquel nous avons participé. Dans un premier temps, nous exposerons l'objectif du projet et la place de notre travail au sein de ce projet. Nous verrons ensuite les principes que nous avons décidé de suivre au moment de la réalisation de notre travail.

1.2.1. L'objectif du projet

L'objectif du projet auquel nous avons participé est d'écouter les lignes DECnet qui entrent et qui sortent du CERN afin de repérer les connexions douteuses. Une connexion est un lien créé temporairement par une application pour un utilisateur afin que ce dernier puisse accéder à une machine sur laquelle il désire travailler. Lorsque son travail est terminé, l'utilisateur se déconnecte de la machine ; le lien est alors détruit par l'application. Nous appellerons connexions douteuses les connexions qui ont **peut-être** été initialisées par des personnes qui essaient de se connecter sur des machines du CERN sans en avoir l'autorisation ou des personnes qui, à partir d'une machine du CERN, se connectent illégalement sur une ou plusieurs machines extérieures. Ce projet est donc motivé par une volonté de détection des intrusions. Comme nous le verrons ultérieurement, les intrus peuvent représenter une menace pour le CERN.

DECnet Phase IV

Le projet est destiné à être utilisé uniquement avec DECnet Phase IV. Plusieurs arguments motivent ce choix. D'une part, un projet similaire existe déjà pour TCP/IP ; d'autre part, au moment de notre stage, le CERN n'était pas encore passé à DECnet Phase V ; enfin, le temps nous était limité.

Le programme et le projet

Le programme réalisé présente à l'utilisateur les informations concernant les connexions douteuses. Il n'est cependant pas certain que toutes ces connexions soient des intrusions ; c'est à l'utilisateur, en l'occurrence le responsable de la sécurité, de décider, après avoir pris connaissance des données fournies par le programme, s'il y a lieu de s'inquiéter au sujet d'une connexion.

Si le responsable de la sécurité considère qu'il y a une intrusion, il place un analyseur de réseau afin de surveiller de plus près la connexion et les activités qui ont lieu sur le réseau et d'évaluer la taille des dégâts.

Après avoir vu ce qui se passe exactement, s'il juge encore que la connexion est une intrusion, le responsable de la sécurité peut prendre les mesures de protection qu'il juge nécessaires. Ainsi, selon les circonstances, il peut notamment prendre des mesures permettant d'augmenter la sécurité des systèmes attaqués. Il peut aussi décider de fermer le compte sur lequel l'intrus s'est introduit. Il peut également, si l'intrusion a eu de graves conséquences, entamer une poursuite judiciaire.

1.2.2. Principe d'une solution

Nous avons déjà pu constater l'hétérogénéité des réseaux du CERN : différents types de machines, différents systèmes d'exploitation, ... Nous essaierons donc de rester les plus indépendants possible de l'environnement dans lequel nous travaillerons. Ainsi, le retrait ou l'ajout d'une machine n'aura pas d'influence sur le programme et le programme ne dépendra pas du type des machines interconnectées par le réseau. Enfin, les machines ne se rendront pas compte de l'existence du programme.

L'écoute du trafic externe

Nous écouterons le trafic qui passe sur la dorsale du trafic externe du CERN et ne retiendrons que les paquets transmis en utilisant le protocole DECnet phase IV. Le trafic arrivant de l'extérieur aboutit sur ce réseau et le trafic en provenance du CERN destiné à l'extérieur passe aussi sur ce réseau. Nous sommes donc sûrs d'observer tout le trafic échangé entre le CERN et l'extérieur. Toutefois, étant donné le volume des informations traversant le réseau, ce choix peut nous amener à recueillir des informations moins précises.

Une détection et non une prévention

Le programme n'offrira qu'une détection des violations de sécurité et non leur prévention. Pour ce faire, il rassemblera un certain nombre d'informations pertinentes relatives aux différentes connexions DECnet en provenance ou à destination du CERN. Les informations à retenir seront définies au moment de la conception. Ces informations seront ensuite analysées dans le but de découvrir les connexions douteuses.

Les mots clés

Cette analyse sera basée sur la présence de mots clés dans les commandes envoyées par l'utilisateur à un système ou dans les réponses du système. Par exemple, on peut considérer que des connexions sur lesquelles passe un grand nombre de fois le groupe de mots "user authorization failure" sont douteuses. La chasse aux connexions douteuses reposera donc uniquement sur une analyse des commandes envoyées par un utilisateur et des réponses envoyées par le système.

Cette méthode nous permettra d'apprendre des choses sur les méthodes de travail des intrus, puisque nous conserverons une partie des informations concernant les connexions initialisées par ceux-ci. Pour la même raison, cette méthode d'écoute du réseau pourra également fournir des preuves qui permettront d'établir la culpabilité ou l'innocence d'individus suspects.

1.3. Evaluation des intérêts du projet

Afin de répondre aux éventuelles questions concernant l'utilité du projet, nous allons montrer qu'il est possible d'attaquer un système informatique. Nous verrons les menaces que présentent ces intrusions. Nous terminerons en montrant que le risque est réel et qu'il est nécessaire de prendre des mesures de protection contre les violations de sécurité.

1.3.1. Comment attaquer une machine ?

De nombreuses stratégies peuvent être utilisées par les personnes qui désirent se connecter sur une machine sans en avoir l'autorisation. Nous allons en signaler quelques-unes, tout en étant conscients du fait qu'il en existe beaucoup d'autres. Nous mentionnerons celles auxquelles nous avons pensé. Signalons toutefois que, d'une part, tous les scénarios d'attaques ne sont pas connus et d'autre part, pour des raisons de sécurité évidentes, les méthodes utilisées par un intrus ne sont pas dévoilées au grand jour, ceci afin de ne pas donner aux gens intéressés les moyens de s'introduire illégalement dans un système.

L'utilisation d'un compte connu

Pour s'introduire illégalement sur une machine, il est possible d'utiliser un compte connu dont on a découvert le nom d'utilisateur et le mot de passe. Ceci est réalisable en faisant plusieurs essais. C'est dans le but de rendre ce genre d'attaques plus compliqué qu'il est recommandé d'avoir un mot de passe différent de son nom d'utilisateur et de ne pas avoir de mot de passe trop simple. Remarquons que, dans ce cas, le système enverra une réponse du genre "Login incorrect" chaque fois qu'une tentative de connexion se solde par un échec. Donc, si nous considérons notre solution, à condition que les mots clés à rechercher aient bien été choisis, il est probable que l'intrusion soit détectée.

L'utilisation d'une lacune du système

Il est également possible d'utiliser une lacune du système pour s'introduire sur une machine. La détection de ce genre d'intrusion, en se basant sur notre solution, est moins probable, excepté si l'intrus envoie des commandes qui pourraient le trahir en étant repérées par le programme.

L'octroi de privilèges

Après s'être introduit sur une machine, l'intrus peut essayer de se donner des privilèges supplémentaires ou de lire des fichiers auxquels il n'a pas le droit d'accéder. De nouveau, en supposant que les mots clés aient été bien choisis, notre solution pourrait détecter ce genre d'attaques.

L'abus par des personnes internes à l'organisation

Enfin, il est possible que des personnes internes à l'organisation abusent des droits qui leur sont donnés. Une nouvelle fois, avec des mots clés appropriés, notre solution pourrait permettre de détecter ce genre d'abus. Toutefois, notre travail n'a pas pour objectif de détecter ces intrusions par les personnes internes à l'organisation.

1.3.2. Menaces et évaluation de ces menaces

Nous venons de voir quelques manières de s'introduire illégalement dans un système informatique. Voyons maintenant quelles sont les menaces que présentent ces intrusions pour le système attaqué.

La menace pour la confidentialité

Un certain nombre d'informations sont réservées à certaines personnes. Pourtant, en cas d'intrusion, certaines informations confidentielles pourraient être lues sans autorisation. C'est notamment ce qui se passe dans le cas de l'espionnage industriel.

En ce qui concerne le CERN, cette menace n'est pas cruciale. En effet, le CERN est transparent en ce sens que tous les résultats des expériences sont publiés et accessibles aux chercheurs du monde entier. De même, les innovations technologiques que le CERN peut mettre au point pour la construction d'une de ses machines sont également du domaine public : ces technologies ne sont pas brevetées et chacun peut s'en servir librement.

Remarquons cependant qu'une lecture non autorisée du courrier électronique d'autrui est une atteinte à la vie privée.

La menace pour l'intégrité du système

La menace pour l'intégrité du système concerne le changement et le retrait non autorisé d'informations. Le système n'est plus alors dans son état initial, supposé correct. Les données modifiées peuvent être totalement inutilisables et des modifications plus subtiles peuvent produire une chaîne d'erreurs dans tous les travaux basés sur ces données.

Puisque l'on ne peut pas identifier avec certitude les données affectées, une conséquence importante de la violation d'intégrité est une perte de confiance dans toutes les données du système. Cette attaque provoque également une perte de temps due à la nécessité de reconstruire le travail effectué depuis la dernière copie de sauvegarde intègre.

La menace pour la disponibilité du système

La disponibilité du système peut également être menacée, notamment par l'utilisation illégitime des ressources et le dérangement dans les services que le système devrait offrir. Un intrus peut rendre un service indisponible en surchargeant le système, en détruisant des données cruciales ou encore en introduisant un virus, un cheval de Troie, une bombe logique,... dans le système.

Dans le cas du CERN, nous pouvons particulièrement mentionner l'utilisation illégitime de ses ressources réseaux. En effet, il n'est pas rare de constater que des personnes se connectent au CERN afin d'initialiser une nouvelle connexion vers l'extérieur. Il se pourrait donc qu'elles fassent un mauvais usage des ressources. Ces personnes devraient plutôt se connecter directement vers leur destination depuis leur institution d'origine.

La menace pour la crédibilité du CERN au niveau de la sécurité informatique

La menace contre la crédibilité du CERN au niveau de la sécurité informatique est plus particulièrement représentée par les personnes qui, depuis une machine du CERN, se connectent illégalement sur une ou plusieurs machines externes.

Si le CERN est utilisé comme tête de pont pour attaquer d'autres sites, cela peut lui donner une mauvaise réputation en ce qui concerne les mesures de sécurité qu'il met en oeuvre. Vu la situation du CERN, ce point est assez important.

La menace pour la sécurité du personnel

L'altération des systèmes contrôlant des machines peut mettre des vies humaines en danger. Pour ce qui est du CERN, les accélérateurs sont contrôlés par des systèmes informatiques. Des attaques contre ces systèmes mettraient donc en danger la vie des personnes en relation avec ces accélérateurs.

1.3.3. Evaluation de la démarche et des intérêts du projet

Nous venons de voir qu'il existe plusieurs façons de s'introduire sur une machine et que ces intrusions peuvent provoquer des dommages pour la victime de l'intrusion. Toutefois, nous pouvons nous demander si ces intrusions représentent des menaces réelles et s'il est vraiment nécessaire de se protéger contre ce genre d'attaques. Nous allons aborder ce problème dans les quelques lignes qui suivent.

Les menaces

Le nombre croissant d'incidents se rapportant à des violations de sécurité et relatés dans la presse témoigne de la vulnérabilité des systèmes informatiques face aux intrusions perpétrées par des personnes extérieures et aux abus commis par des personnes autorisées à accéder à ces systèmes. En voici un exemple[1] :

En septembre 1993, la succursale belge de la Banque Nationale de Paris a déclaré avoir été victime d'une importante fraude informatique en juin de la même année. Selon la presse, une somme de 245 millions de francs belges aurait été détournée. Les deux fraudeurs utilisaient leur accès direct aux ordinateurs pour demander le débit des comptes de la BNP et le transfert des bénéfices vers leurs propres comptes ouverts dans d'autres banques. Selon les sources de la BNP, les commissaires aux comptes ont découvert la fraude lors de vérifications de routine sur les transactions interbancaires.
Dès que la fraude a été découverte, les autres banques ont été contactées et l'argent a été récupéré. Par suite de la fraude, la BNP s'interroge sur la façon dont les fraudes se sont produites et sur les éventuels renforcements possibles de ses systèmes de sécurité, ceci dans le but d'éviter une récidive.

De plus en plus de personnes prennent donc conscience de la nécessité d'instaurer des mesures de protection. Mais quel type de mesures prendre ?

Les mesures formelles

On peut penser aux mesures formelles et à la restriction d'accès aux systèmes informatiques ainsi qu'aux fichiers qu'ils contiennent. Cependant, plusieurs facteurs limitent l'efficacité de ces mesures.

Le premier est de nature humaine : les utilisateurs considèrent souvent les procédures de sécurité comme des obstacles diminuant l'efficacité de leur travail et donc ne les appliquent pas.

---

[1] *Brussels Branch Of BNP Hit By Computer Fraud*, dans Software Engineering Notes, Vol. 19, N°1, Janvier 1994, pages 6-7.

Deuxièmement, les responsables de systèmes doivent réaliser un compromis entre des intérêts conflictuels ; tandis que la sécurité pourrait requérir la centralisation des ressources, les utilisateurs demandent souvent l'accès aux ressources distribuées.

Remarquons toutefois que la décentralisation ne s'oppose pas nécessairement à la sécurité des systèmes. En effet, lorsque les ressources sont centralisées, un intrus qui s'introduit dans un système obtient plus facilement l'accès aux autres ressources du système puisque les ressources sont regroupées à un seul endroit. Si les ressources sont décentralisées, l'accès à l'une d'elles ne facilite pas l'accès aux autres.

D'une part, la centralisation réduit le nombre de points de contrôle sous la surveillance du responsable de la sécurité et d'autre part, la décentralisation empêche l'accès simultané à toutes les ressources en cas d'intrusion. Il faut donc trouver un équilibre entre centralisation et décentralisation.

Troisièmement, les systèmes contiennent souvent des points faibles qui ne sont pas connus.

Enfin, il est toujours possible que les personnes internes à l'organisation abusent des privilèges qui leur sont accordés.

L'audit

De ces limitations est née l'idée des méthodes de détection d'intrusions basées sur l'audit. L'audit permet de garder une trace de tous les événements qui se produisent sur un système. Ces informations sont rassemblées dans le fichier d'audit. Il est alors possible d'analyser ce fichier afin de découvrir les éventuelles intrusions. Nous allons présenter plus longuement l'audit de sécurité dans la chapitre suivant.

Chapitre 2

L'audit de sécurité

Un programme d'audit de sécurité écoute l'activité d'un système informatique et enregistre certains événements qui permettent de retracer l'activité de ce système et de découvrir d'éventuels problèmes liés à la sécurité. Toutes ces informations sont placées dans un ou plusieurs fichiers d'audit. Les fichiers d'audit sont ensuite analysés par une personne, généralement le responsable de la sécurité, ou par un programme, comme nous le verrons ultérieurement, afin de détecter, comprendre et/ou contrer les éventuelles attaques.

Dans ce chapitre, nous nous centrerons sur l'utilisation de l'audit pour la détection des intrusions. Nous verrons ce qui motive le développement de systèmes de détection d'intrusions basés sur l'audit de sécurité. Nous envisagerons ensuite successivement la réalisation de l'audit, l'analyse des fichiers d'audit, les qualités des outils d'analyse de fichiers d'audit et les problèmes suscités par l'audit. Nous présenterons ensuite un modèle de détection d'intrusions et le rôle que peut jouer l'expertise dans la détection des intrusions. Nous terminerons en présentant quelques outils d'analyse de fichiers d'audit.

2.1. Motivations

Comme le soulignent [LUNT, 93][1] et [DENNING, 87][2], le développement de systèmes de détection d'intrusions basés sur l'audit est motivé par plusieurs facteurs :

- La plupart des systèmes ont des points faibles au niveau de la sécurité, ce qui les rend vulnérables face aux intrusions et à d'autres formes d'abus. La détection et la correction de tous ces points faibles ne sont pas possibles pour des raisons techniques et économiques.

---

[1] Teresa F. LUNT, *A survey of intrusion detection techniques*, Computers & Security, Vol. 12, n°4, 1993, pages 405-418.

[2] Dorothy E. DENNING, *An Intrusion-Detection Model*, IEEE Transactions on Software Engineering, Vol. 13, n°2, Février 1987, pages 222-232.

- Les systèmes existants qui ont des points faibles connus ne peuvent pas facilement être remplacés par des systèmes plus sûrs, principalement pour deux raisons : d'abord, les systèmes existants ont souvent des caractéristiques attrayantes qui manquent aux systèmes plus sûrs ; ensuite, ces remplacements peuvent être impossibles pour des raisons économiques.
- Développer des systèmes absolument sûrs est extrêmement difficile, si pas généralement impossible.
- Même les systèmes les plus sûrs sont vulnérables face aux abus commis par les personnes de l'organisation qui emploient à tort leurs privilèges. L'audit est peut-être le seul moyen de détecter ce genre d'abus.
- Puisqu'ils gardent une trace des activités des utilisateurs, les fichiers d'audit peuvent fournir des preuves qui permettent d'établir la culpabilité ou l'innocence d'un suspect.
- L'audit n'entrave pas le travail des personnes autorisées à accéder au système qui n'abusent pas des droits qui leur sont octroyés.

L'audit représente donc une solution attrayante pour la détection des intrusions. Mais comment réaliser cet audit ?

2.2. La réalisation de l'audit

Les deux sections qui suivent sont inspirées de [LUNT, 93][1], [MARSHALL, 91][2] et [HABRA, 92][3].

Nombre d'applications, notamment le système d'exploitation, créent leur propre fichier d'audit. Toutefois, les informations contenues dans ces fichiers sont principalement destinées à des mesures de performances, à des statistiques ou à des tests. Il en résulte que seule une infime partie de ces informations peut être utile pour la détection des intrusions. De plus, le volume des informations à considérer rend toute analyse compliquée. Le responsable ne peut généralement analyser qu'une toute petite partie des données, manquant peut-être les menaces les plus importantes.

---

1 Teresa F. LUNT, op cit., pages 405-406.

2 Victor H. MARSHALL, *Intrusion Detection in Computers, Summary of the Trusted Information Systems (TIS) Report on Intrusion Detection Systems*, Janvier 1991.

3 N. HABRA, B. LE CHARLIER, I. MATHIEU et A. MOUNJI, *ASAX : Software Architecture and Rule-Based Language for Universal Audit Trail Analysis*, Proceedings of the Second European Symposium on Research in Computer Security (ESORICS), Toulouse, France, Novembre 1992, pages 435-450.

Les informations pertinentes

Plusieurs solutions sont envisageables afin de réduire le volume d'informations inutiles :

- Soit le responsable de la sécurité opère une sélection parmi toutes les informations d'audit fournies, par exemple, par le système d'exploitation. Il analyse alors uniquement les informations qu'il considère pertinentes. Remarquons qu'il est indispensable que la sélection soit adéquate. Faute de cela, l'audit de sécurité peut se révéler tout à fait inutile.
- Soit le responsable dispose d'un outil d'audit propre à la sécurité qui rassemble des informations pertinentes pour la détection des intrusions. Il est alors nécessaire de se demander quels sont les événements pertinents que l'on désire enregistrer.
- Soit le responsable d'audit modifie les utilitaires offerts par le système d'exploitation afin d'adapter l'audit au besoin de la sécurité. C'est notamment ce qui a été réalisé à l'université de Stanford dans le cadre du programme Swatch (Simple WATCHer) [HANSEN, 92][1]. Pour réaliser cela, il faut toutefois disposer du code source du système d'exploitation.

La protection des fichiers d'audit

Dès que les fichiers d'audit sont constitués, il est crucial qu'ils soient protégés contre toute corruption ou tout accès non autorisé. En effet, il n'est pas question que des intrus très habiles soient capables d'altérer ces fichiers afin de faire disparaître toute trace de leur intrusion.

Ces fichiers doivent donc se trouver sur un espace du système particulièrement sûr disposant d'un contrôle d'accès sévère. Seules quelques personnes ont l'autorisation d'accéder à ces fichiers. Ainsi, il est peut-être préférable de transférer les fichiers d'audit sur un système différent du système surveillé afin qu'on ne puisse pas y accéder en utilisant une des lacunes du système surveillé.

De plus, si l'analyse des fichiers d'audit se fait sur une machine différente de celle sur laquelle les informations ont été rassemblées, la ligne sur laquelle sont transmis les fichiers doit être protégée contre toute attaque.

---

[1] S. E. HANSEN et E. Todd ATKINS, *Centralized System Monitoring With Swatch*, Stanford University, Juillet 1992.

2.3. L'analyse des fichiers d'audit

Nous avons déjà mentionné que la quantité d'informations rassemblées lors de l'audit est importante. Même après une sélection adéquate, les informations pertinentes sont nombreuses. De plus, l'analyse de ces informations est difficile. Il est en effet nécessaire d'identifier ce que constitue un comportement suspect et de rechercher, parmi toutes les données d'audit, des indices qui pourraient révéler un tel comportement. Ceci est encore plus vrai lorsqu'il faut considérer des fichiers d'audit en provenance de plusieurs systèmes. Il est donc intéressant de disposer d'outils automatisés qui analysent les données d'audit et assistent la détection des événements suspects.

Les outils automatisés

Nous pouvons distinguer deux types d'outils automatisés d'analyse de fichiers d'audit.

D'un côté, les outils, que nous appellerons "passifs", réduisent la quantité d'informations que doit considérer le responsable de la sécurité. Ces outils ne détectent aucune tentative d'intrusion, mais ils allègent la tâche du responsable.

De l'autre côté, les outils, que nous appellerons "intelligents", essayent de repérer les tentatives d'intrusion en effectuant une analyse plus ou moins complexe des données d'audit. Deux types d'analyses sont possibles :

- une analyse a posteriori des données afin de détecter les intrusions ;
- une analyse en temps réel des données afin de détecter les intrusions et les tentatives d'intrusion.

Souvent, dans leur phase expérimentale, les outils offrent une analyse a posteriori des fichiers d'audit. Certains évoluent ensuite vers une analyse en temps réel. Cette évolution dépend du type de problème que doit résoudre l'outil ; tous les problèmes ne nécessitent pas une réponse en temps réel.

Les outils intelligents

Les outils intelligents profitent de la capacité et de la vitesse de traitement des ordinateurs. En effet, les ordinateurs modernes sont capables d'analyser rapidement un grand nombre de données. Lorsque le responsable est averti que des activités suspectes sont en train de se dérouler, il peut prendre les mesures qui s'imposent. Par ailleurs, un système peut être programmé afin de prendre lui-même des mesures défensives, par exemple interrompre la session d'un intrus. Ces outils sont donc de véritables assistants à la détection des intrusions.

Actuellement, plusieurs outils intelligents existent. Ces outils reposent sur des modèles et de l'expertise. Nous y reviendrons dans un instant. Auparavant, nous allons voir les qualités requises pour de tels outils et les réactions que leur utilisation ont pu susciter chez certaines personnes.

2.4. Les qualités des outils automatisés d'analyse

Tous les outils automatisés d'analyse de fichiers d'audit doivent présenter certaines qualités : d'une part, au niveau de leur conception et de leur interface, d'autre part, au niveau des résultats qu'ils fournissent.

La conception

Une des qualités principales d'un logiciel est sa capacité à être utilisé dans un environnement différent de celui dans lequel il a été conçu avec un minimum d'adaptation. Ceci vaut également pour les outils d'analyse de fichiers d'audit [HABRA, 92][1].

Ainsi, un tel outil devrait être utilisable dans différents environnements avec un minimum d'adaptation. Par exemple, un outil prévu pour surveiller une machine ayant un système d'exploitation UNIX devrait également pouvoir surveiller une machine possédant un système d'exploitation de type VMS. Il est également possible qu'un outil soit amené à analyser des fichiers d'audit en provenance de plusieurs systèmes qui se trouvent sur un réseau hétérogène. Il est donc nécessaire que l'outil puisse traiter sans problème ces différents fichiers.

L'interface

L'interface d'un logiciel est aussi très importante. C'est à travers l'interface que l'utilisateur dialogue avec le système. Cette interface doit être conviviale. Elle conditionne la facilité et l'efficacité d'utilisation du système ainsi que la facilité d'apprentissage du logiciel [MEINADIER, 91][2].

L'interface d'un outil d'analyse de fichiers d'audit est tout aussi importante. Elle doit permettre à l'utilisateur, en l'occurrence le responsable de la sécurité, de converser facilement avec le système et d'utiliser toutes ses fonctionnalités. Elle doit également permettre un apprentissage rapide du logiciel.

---

[1] N. HABRA, B. LE CHARLIER, I. MATHIEU et A. MOUNJI, op cit., page 438.

[2] J.P. MEINADIER, *L'interface utilisateur - Pour une informatique plus conviviale*, DUNOD, 1991.

Les résultats

Enfin, un outil intelligent d'analyse de fichiers d'audit doit fournir de bons résultats. Ainsi, il doit déclencher peu de fausses alarmes tout en laissant un minimum d'intrusions non détectées.

Si l'outil déclenche souvent des alarmes injustifiées, le responsable pourrait se retrouver surchargé et relâcher une partie de sa surveillance, laissant ainsi la porte ouverte aux abus. Le danger des intrusions non détectées est évident.

Toutefois, il est plus facile pour le responsable de vérifier l'occurrence des fausses alarmes que les intrusions non détectées.

En conclusion

Il est donc important de disposer d'outils paramètrables pour permettre, au moins théoriquement, au responsable de la sécurité de construire un système aussi adapté que possible à son environnement.

2.5. Et le respect de la vie privée ?

Cette section est inspirée de [LUNT, 93][1].

L'utilisation de l'audit et des systèmes de détection d'intrusions a suscité, chez certaines personnes, des inquiétudes concernant le respect de la vie privée. En effet, les systèmes de détection d'intrusions en temps réel permettent de surveiller les employés et leur rythme de travail. Il est donc nécessaire d'utiliser ces systèmes de façon adéquate.

Il n'existe pas, à l'heure actuelle, de règle ou de législation concernant l'usage de l'audit. Cependant, certaines personnes pensent qu'il faudrait que les employés soient avertis que leurs activités sont surveillées et qu'ils soient informés des informations qui sont rassemblées ainsi que de l'utilisation qui en sera faite. Par exemple, lorsque l'on se connecte sur certains serveurs Internet aux Etats-Unis, un message nous avertit que toutes nos opérations seront enregistrées dans un fichier. Si nous n'acceptons pas ce principe, nous pouvons nous déconnecter.

A l'opposé, d'autres personnes considèrent que le simple fait d'avoir connaissance de l'existence d'un système de détection d'intrusions pourrait représenter une aide à un intrus potentiel.

---

[1] Teresa F. LUNT, op cit., pages 416-417.

De plus, le respect de la vie privée des utilisateurs devrait inciter les responsables de systèmes de détection d'intrusions à veiller à la protection des fichiers d'audit et des résultats de la surveillance. Les responsables devraient également assurer que les données et les résultats seront uniquement utilisés par la sécurité.

2.6. Un modèle de détection d'intrusions

Les modèles permettent une approche systématique de la détection des intrusions. Nous allons présenter le modèle IDES (*Intrusion-Detection expert System*) développé au SRI International. Ce modèle a donné lieu à de nombreuses recherches dans le domaine de la détection des intrusions et forme la base conceptuelle de bon nombre d'outils. C'est pourquoi nous avons décidé de le présenter.

2.6.1. Le modèle

Cette description est basée sur [DENNING, 87][1].

Le modèle IDES est basé sur l'hypothèse que les intrusions impliquent une utilisation anormale du système. Par conséquent, les violations de sécurité peuvent être détectées en recherchant les comportements inhabituels lors de l'utilisation du système. Par exemple, une personne qui se connecte sur un système par l'intermédiaire d'un compte auquel elle ne peut normalement pas accéder pourrait se connecter à des heures différentes et depuis des endroits différents de ceux de l'utilisateur légitime de ce compte. De plus, le comportement de l'intrus peut être assez différent de celui de l'utilisateur légitime. Ainsi, l'intrus pourrait passer beaucoup de temps à parcourir les différents répertoires et à copier des fichiers alors que l'activité principale de l'utilisateur légitime est d'éditer ou de compiler des programmes.

Ce modèle est indépendant de tout système particulier, ainsi que de tout environnement d'applications, des points faibles du système ou des types d'intrusions. Il permet donc de décrire un système de détection d'intrusions de façon générale. En fait, il est presque devenu un standard de facto dans le domaine car il fournit une ossature générale qui permet de décrire d'autres outils d'analyse de façon générale et abstraite.

---

[1] Dorothy E. DENNING, op cit., pages 222-232.

Les composants

Le modèle comprend six composants :

- les sujets *(Subjects)* exécutent des actions sur le système surveillé ; généralement, il s'agit de l'utilisateur ;
- les objets (*Objects*) sont les ressources gérées par le système ; ce sont les fichiers, les commandes,... Les sujets réalisent des actions sur les objets ;
- les enregistrements d'audit (*Audit Records*) sont générés par le système surveillé en réponse aux actions exécutées ou tentées par des sujets sur des objets ;
- les profils (*Profiles*) caractérisent le comportement normal des sujets vis-à-vis des objets. Un comportement observé est exprimé par une mesure et un modèle statistique ;
- les enregistrements d'anomalie (*Anomaly Records*) sont générés lorsqu'une activité suspecte est détectée ;
- les règles d'activités (*Activity Rules*) spécifient les actions à entreprendre lorsque certaines conditions sont vérifiées. Ces règles mettent à jour les profils, détectent les comportements anormaux et produisent les rapports.

Le principe

Le système surveillé rassemble lui-même les données d'audit. Il doit également transmettre les enregistrements d'audit vers le système de détection d'intrusions où ils sont analysés.

Quand un enregistrement d'audit est reçu, il est comparé aux profils. Les observations obtenues à partir des données d'audit sont utilisées avec le modèle statistique déterminé par le profil afin de déterminer si la nouvelle observation est anormale. En même temps, les profils sont mis à jour. Si un comportement anormal est détecté, un enregistrement d'anomalie est généré.

Il est important de bien choisir les activités que l'on désire enregistrer ainsi que les modèles statistiques que l'on veut utiliser afin de détecter le plus d'intrusions possible et d'éviter les fausses alarmes.

2.6.2. Une évaluation

Ce modèle n'a aucune connaissance concernant les mécanismes de sécurité ou les points faibles du système surveillé. Si le modèle était basé sur des connaissances concernant les points faibles du système, il ne pourrait pas détecter les intrusions dues à des déficiences qui ne sont pas connues. Ici, il est possible que la détection des intrusions permette au responsable de la sécurité de localiser certains points faibles du système.

Cependant, il est possible qu'une personne échappe à la détection en modifiant progressivement son comportement jusqu'à ce qu'un nouveau modèle de son comportement normal ait été établi, ce modèle permettant d'attaquer en toute sécurité le système. Cette attaque est donc compliquée pour l'intrus, mais il se peut que cela se produise.

De plus, certains utilisateurs peuvent avoir un comportement bien établi, se connectant toujours aux mêmes heures et à partir des mêmes endroits, réalisant toujours le même type d'activités, ... Cependant, d'autres utilisateurs peuvent avoir un comportement beaucoup plus variable au fil des jours ; ils se connectent toujours à des heures différentes et depuis des endroits différents, chaque jour ils changent totalement leurs activités, ... Pour ces derniers, pratiquement tous les comportements peuvent être considérés comme normaux et un intrus qui utiliserait le compte d'un tel utilisateur pourrait passer inaperçu.

Ce modèle représente donc une bonne base pour un système de détection d'intrusions. Mais, comme nous l'avons vu, il est possible que certaines intrusions ne soient pas détectées.

2.7. Les systèmes experts

Les systèmes experts peuvent également être utiles pour la détection des intrusions. Les règles représentent un ensemble de faits qui sont directement utiles pour la détection des intrusions. Ainsi, des informations concernant les vulnérabilités connues du système surveillé, les scénarios d'attaques connus mais aussi les intuitions concernant les comportements suspects sont encodées sous la forme de règles dans le système expert. Ces règles sont ensuite utilisées pour analyser les données d'audit à la recherche d'activités suspectes.

Les systèmes experts présentent toutefois un désavantage. En effet, ils ne s'intéressent qu'aux vulnérabilités et aux attaques qui sont connues. Un scénario d'intrusion qui ne déclenche aucune règle ne sera pas détecté. Or, il est possible que la plus grande menace provienne des vulnérabilités qui ne sont pas encore connues et des attaques qui n'ont pas encore été essayées.

2.8. Les outils

Plusieurs outils ont été conçus afin d'implémenter efficacement ces modèles et cette expertise. Nous allons en présenter quelques-uns. Nous commencerons par décrire le prototype IDES. Nous mentionnerons ensuite l'existence du projet Minos. Nous terminerons par une présentation des systèmes NADIR et ASAX.

2.8.1. Le prototype IDES

Cette description est basée sur l'article [LUNT, 88][1].

Le prototype IDES a été développé au SRI International sur base du modèle IDES. Ce prototype surveille les utilisateurs sur un système éloigné à travers les données d'audit générées par ce système et détecte les comportements anormaux.

Pour ce faire, IDES enregistre, pour chaque utilisateur du système, différentes mesures. Chaque mesure représente un aspect du comportement d'un utilisateur. Ainsi, IDES enregistre des informations telles que le moment où l'utilisateur se connecte, l'endroit depuis lequel il se connecte, la durée de sa session,... Pour chacune de ces mesures, le prototype conserve un profil. Le profil est une description du comportement normal de l'utilisateur pour une certaine mesure.

Les données d'audit sont générées par le système d'exploitation du système surveillé. Après leur génération, les données sont mises dans le format IDES, indépendant du système ; elles sont chiffrées et transmises au système IDES selon un protocole de transmission défini dans le cadre du projet. Aucune donnée d'audit ne circule en clair.

Lorsqu'il reçoit les données d'audit, IDES regarde si l'activité décrite par ces données est anormale par rapport au profil de l'utilisateur et les profils sont mis à jour sur base de l'activité observée. Quand un comportement anormal est détecté, un enregistrement d'anomalie est généré et placé dans une base de données.

La détection se fait en temps réel.

IDES offre également une interface graphique, permettant au responsable de la sécurité de visualiser graphiquement l'activité du système surveillé et les comportements anormaux détectés.

---

[1] Teresa F. LUNT et R. JAGANNATHAN, *A Prototype Real-Time Intrusion-Detection Expert System*, Proceedings of 1988 IEEE Symposium on Security and Privacy, 1988, pages 59-66.

IDES est implémenté sur une machine différente de la machine surveillée. Cela a plusieurs avantages :

- au niveau des performances, IDES ne dégrade pas le temps de réponse du système surveillé ;
- au niveau de la sécurité, les points faibles du système surveillé ne mettent pas la sécurité de l'IDES en danger ;
- au niveau de l'intégration, IDES peut être adapté à différents environnements et peut être intégré avec différents systèmes d'exploitation dans les systèmes surveillés.

La version la plus récente de l'IDES intègre également des informations concernant les points faibles connus du système, les scénarios d'attaques connus et les intuitions au sujet des comportements suspects. Des règles sont fixées ; elles ne dépendent pas du passé de l'utilisateur.

L'expertise combinée au modèle IDES pourrait offrir une meilleure détection puisqu'elle réunit les points forts de ses deux composants.

2.8.2. Minos

Des travaux d'un autre style ont été entrepris en Australie, avec le projet Minos. Cette présentation est inspirée de [NEWBERRY, 90][1].

Tout comme IDES, Minos utilise des informations sur la façon dont l'utilisateur se comporte avec l'ordinateur afin d'identifier les intrusions. Les intrus sont identifiés sur base de leurs déviations par rapport au comportement de l'utilisateur normal.

Minos examine toutes les commandes tapées par l'utilisateur et recherche les comportements non caractéristiques qui pourraient révéler un intrus. L'originalité réside dans les mesures que Minos utilise pour identifier les différents utilisateurs. Ces mesures sont les suivantes :

- le choix des commandes de l'utilisateur. Les utilisateurs se servent toujours du même ensemble de commandes ; cela est employé pour distinguer les utilisateurs légitimes des intrus ;
- l'ordre dans lequel les commandes sont utilisées ;
- le positionnement de l'utilisateur dans l'ordinateur à savoir, dans quel répertoire se trouve l'utilisateur, à quel fichier accède-t-il, ... ? Si un utilisateur accède souvent à des sections auxquelles il n'avait pas l'habitude d'accéder, cela peut révéler un intrus ;

---

[1] Michael NEWBERRY, *Minos : Extended User Authentiçation*, Proceedings of Advances in Cryptology (Auscrypt), 1990, pages 410-423.

- le style de frappe de l'utilisateur. Celui-ci est mesuré par l'intervalle de temps qui sépare deux frappes consécutives de l'utilisateur.

En 1990, au moment de la parution de l'article, seule la première des mesures mentionnées ci-dessus était implémentée. Toutefois, la fin de l'implémentation était prévue pour le mois de décembre de la même année.

2.8.3. Le système NADIR

Cette présentation est basée sur [HOCHBERG, 93][1].

Le système expert NADIR (*Network Anomaly Detection and Intrusion Reporter*) a été développé pour détecter les abus sur le réseau du Laboratoire National de Los Alamos, aux Etats-Unis.

NADIR reçoit des données d'audit en provenance d'un noeud particulier du réseau. Ces données reflètent l'activité du réseau. Chaque semaine, NADIR synthétise les différentes données d'audit qu'il a reçues sous la forme de profils. Les règles du système sont appliquées à ces profils.

La formulation des règles

Les règles du système codifient des scénarios de violation de la politique de sécurité mais également des informations concernant les activités suspectes ou inhabituelles.

Afin de formuler ces règles, les concepteurs du système se sont entretenus avec des experts, à savoir le personnel responsable de la sécurité. De même, les personnes responsables de l'établissement et de l'application de la politique de sécurité ont été interrogées. De plus, les connaissances au sujet des abus et des intrusions passées ainsi que l'intuition des responsables de la sécurité ont permis de formuler des règles.

Un premier ensemble de règles a donc été implémenté puis testé. Ces tests ont permis de découvrir de nouveaux scénarios d'attaques et des vulnérabilités inconnues du système. Ces découvertes ont permis à leur tour de compléter l'ensemble de règles. Ces règles continuent à être complétées au fur et à mesure que de nouvelles découvertes ont lieu. Malheureusement, nous ne disposons pas d'exemple de règles utilisées par NADIR.

Actuellement, NADIR effectue une analyse a posteriori des données. Toutefois, les concepteurs espèrent atteindre prochainement une détection en temps réel.

---

[1] J. HOCHBERG, K. JACKSON, C. STALLINGS, J.F. McCLARY, D. DUBOIS et J. FORD, *NADIR : An Automated System for Detecting Network Intrusion and Misuse*, Computers & Security, Vol. 12, n°3, 1993, pages 235-248.

2.8.4. Le système ASAX

Cette description est basée sur [HABRA, 92][1] et [HABRA, 94][2].

Le projet ASAX (*Advanced Security Audit-trail Analysis on uniX*), entrepris conjointement par les Facultés de Namur et la société Siemens Nixdorf Software, vise le développement d'un outil d'analyse de données d'audit universel, efficace et puissant.

ASAX n'est basé sur aucun modèle particulier ni sur aucune expertise. Il est utilisable avec tout modèle ou toute expertise.

Un outil universel

ASAX est capable d'analyser n'importe quel fichier d'audit. Pour ce faire, tout fichier doit être traduit, avant l'analyse, dans un format approprié que l'outil comprend. Ce format, défini dans le cadre du projet, est suffisamment flexible pour que tout fichier d'audit puisse être traduit assez facilement dans ce format. Des adaptateurs automatiques de format pourraient même être fournis pour traduire chaque fichier d'audit dans le format normalisé. Actuellement, il existe de tels adaptateurs, mais ils ne sont pas génériques.

Les fichiers d'audit générés par les systèmes d'exploitation des différents systèmes surveillés sont donc mis dans le format normalisé. L'analyse est ensuite effectuée sur ces fichiers en format normalisé. Ceci assure le caractère universel de l'outil.

Un outil efficace et puissant

Afin d'obtenir un outil puissant, un langage particulier a été mis au point. Ce langage, RUSSEL (*RUle-baSed Sequence Evaluation Language*), est spécialement conçu pour l'analyse efficace de grands fichiers séquentiels, tels que les fichiers d'audit. Ainsi, la recherche d'une séquence d'enregistrements dans un fichier séquentiel se fait en une seule passe. Etant donné la taille des fichiers à considérer, cette possibilité est très précieuse et rend l'outil très puissant.

Les règles que le responsable de la sécurité désire appliquer au moment de l'analyse des fichiers d'audit sont exprimées dans ce langage. Le responsable doit donc apprendre ce langage.

---

[1] N. HABRA, B. LE CHARLIER, I. MATHIEU et A. MOUNJI, op cit., pages 435-450.

[2] N. HABRA, B. LE CHARLIER, A. MOUNJI et D. ZAMPUNIERIS, *Prototype Distributed System for On-line Network Security Monitoring*, Juillet 1994.

Un outil distribué

Les dernières recherches sur le projet sont orientées vers le développement d'un système distribué pour la surveillance de la sécurité des réseaux en temps réel. En effet, une analyse globale des événements qui surviennent sur l'ensemble des systèmes peut révéler des activités suspectes alors que ces mêmes événements considérés séparément ne témoignent d'aucun danger. Ainsi, une tentative de connexion qui échoue sur un système ne représente aucun danger. Par contre, si plusieurs systèmes sont témoins d'un tel échec en un laps de temps assez court, cela peut indiquer une tentative d'intrusion.

Au niveau de chaque système surveillé, une première sélection est opérée sur les données d'audit générées par le système d'exploitation. Ainsi, les données d'audit sont converties dans le format normalisé et analysées. Les données retenues sont ensuite envoyées vers un système central qui réalise une analyse globale des données. Cette analyse peut produire une alarme, des statistiques ou encore un rapport sur le niveau de sécurité du réseau.

Un exemple

Supposons, par exemple, que l'on désire surveiller le nombre de tentatives de connexion qui échouent sur l'ensemble des systèmes d'un réseau. Au niveau local, le système analyse les données d'audit et n'envoie au système central que les données indiquant un tel échec. Le système central analyse alors l'ensemble des données qu'il reçoit des différents systèmes afin de découvrir des séquences de tentatives de connexion qui ont échoué.

La figure n°2.1 présente un exemple de règle. Cette règle recherche l'occurrence d'un échec de connexion et, après en avoir trouvé un, déclenche une règle qui vérifie qu'il n'y a pas plus de "max" occurrences d'un tel échec en "durée" secondes. Si cette deuxième règle est enfreinte, une alarme sera envoyée au responsable de la sécurité.

```
rule Failed_login (max, durée : integer)
#Cette règle détecte un premier échec de connexion et
#déclenche une règle de comptage avec un temps d'expiration

begin

if evt = 'login' and res = 'failure' and
is_unsecure(terminal)
  ->Trigger off for next Count_rule1 (max-1, temps+durée)
fi

Trigger off for next Failed_login (max, durée)

end
```

Figure n°2.1 : Exemple de règle

2.9. Conclusions

L'audit semble être une technique intéressante pour la détection des intrusions, particulièrement si le responsable dispose d'un outil automatisé qui lui permet de réaliser une analyse des données assez complexe.

Plusieurs techniques de détection existent, chacune présentant des points forts et des points faibles. Un système de détection d'intrusions devrait combiner plusieurs de ces techniques afin de réunir les points forts et d'éliminer les points faibles de chacune. Ainsi, un système combinant l'expertise et le modèle IDES, tel que la version la plus récente du prototype IDES, pourrait permettre de détecter une intrusion qui ne s'était encore jamais produite. En effet, bien que le comportement ne déclenche aucune règle du système expert, il pourrait être détecté sur base de son caractère anormal. D'autre part, un comportement pourrait déclencher une règle du système expert indépendamment de toute considération relative au caractère normal ou non du comportement.

Chapitre 3

Une présentation de DECnet

Nous avons présenté l'audit de sécurité et quelques solutions qui ont déjà été proposées pour la détection des intrusions. Nous pouvons à présent aborder la solution qui a été retenue au CERN pour essayer de détecter les connexions DECnet douteuses. Nous commencerons par présenter, dans ce chapitre, notre environnement de travail, à savoir DECnet.

DECnet est une implémentation de l'architecture de réseaux de Digital Equipment Corporation. Etant donné que notre projet est motivé par une volonté de détection des connexions douteuses qui utilisent cette implémentation, nous allons consacrer un chapitre à la présentation de DECnet.

Nous aborderons d'abord la notion d'architecture de réseaux. Nous retracerons ensuite rapidement l'évolution de DECnet. Après cela, nous présenterons les objectifs de l'architecture de réseaux de Digital. Notre programme étant destiné à analyser les connexions initialisées avec DECnet Phase IV, nous présenterons cette phase. Puis, nous donnerons les raisons qui motivent l'apparition de DECnet Phase V, la dernière version de DECnet, et nous exposerons les principes de cette version.

Nous sommes conscients que cette description n'est pas complète. Cependant, le but de notre travail n'est pas une étude détaillée de DECnet. Cette description est motivée par le rapport direct qui existe entre DECnet Phase IV et notre travail.

3.1. Architecture de réseaux

Qu'est-ce qu'une architecture de réseaux ? Quels sont les avantages et les inconvénients d'une architecture en couches ? Telles sont les questions auxquelles nous essaierons de répondre au cours de cette section. Nous terminerons en présentant l'architecture de réseaux de Digital.

3.1.1. Définition

Une architecture de réseaux définit les mécanismes de communication et les interfaces que des systèmes informatiques de différents types doivent adopter afin de pouvoir s'échanger des données.

Nécessité d'une architecture de réseaux

Selon Digital[1], une architecture de réseaux est indispensable pour plusieurs raisons :

- il existe une grande variété d'équipements informatiques, de moyens de communication et de systèmes d'exploitation. Ceux-ci sont mis au point par des personnes différentes. Aussi, la communication entre ces différents équipements n'est possible que si les constructeurs utilisent un ensemble de spécifications communes ;
- l'architecture de réseaux définit, d'une part, comment utiliser, dans le réseau, des standards tels qu'OSI (*Open System Interconnection*) et d'autre part, comment y intégrer des standards individuels ;
- les technologies de communication sont en constante évolution et les réseaux doivent être capables d'évoluer pour inclure ces nouvelles technologies. Dans une architecture modulaire, les changements dans un domaine n'ont pas d'influence sur les autres domaines. Cette évolution est donc possible ;
- la structure modulaire permet d'implémenter certaines fonctions hautement performantes au niveau du matériel, cette implémentation étant indépendante du système. Ainsi, les protocoles de liaison de données des réseaux locaux sont implémentés sous la forme d'adaptateurs pour ces réseaux.

3.1.2. Principes et avantages d'une architecture en couches

Attardons nous quelques instants sur un type particulier d'architecture de réseaux, l'architecture en couches.

Dans une architecture en couches, les différents services offerts pour permettre la communication sont divisés en groupes logiquement cohérents appelés couches. Les couches sont construites les unes au-dessus des autres et chaque couche utilise les services fournis par la couche inférieure. Chaque système du réseau contient son propre élément de la couche, appelé entité.

---

[1] *DECnet DIGITAL Network Architecture (Phase V) - General Description*, Digital Equipment Corporation, Maynard Massachusetts, Septembre 1987.

Deux types de relations

L'architecture spécifie deux types de relations entre les entités :

- les interfaces, c'est-à-dire les relations entre entités qui se trouvent généralement dans le même système. Typiquement, une entité d'une couche fournit un service à une entité de la couche supérieure en demandant un service à une entité de la couche inférieure. L'architecture spécifie les services offerts comme des appels de procédures, mais ces spécifications sont fonctionnelles et ne doivent pas nécessairement être implémentées sous la forme d'appels de procédures ;
- les protocoles, c'est-à-dire les relations entre entités équivalentes qui se trouvent généralement dans des systèmes différents. Ces protocoles définissent très précisément le format des messages à échanger au moment de la communication et les règles à suivre lors de l'échange de ces messages.

La figure n°3.1 illustre le principe d'une architecture en couches.

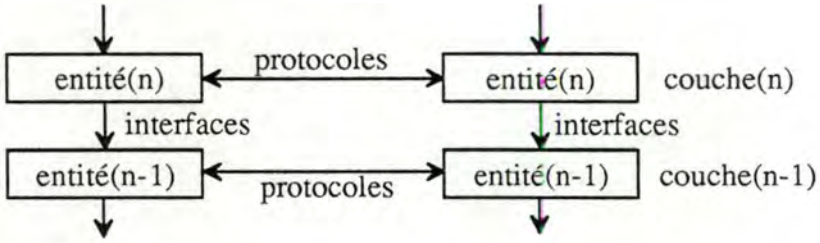

Figure n°3.1 : Principe d'une architecture en couches

Indépendance des couches

Les protocoles et les mécanismes particuliers mis en oeuvre par une couche sont cachés aux autres couches. Une couche ne connaît donc que les services offerts par la couche inférieure. Cela conduit à l'indépendance des couches. En effet, les mécanismes utilisés dans une couche ne sont pas connus des autres couches. Des changements très importants peuvent donc être réalisés dans une couche sans affecter les autres couches.

3.1.3. DNA, l'architecture de réseaux de Digital

L'architecture de réseaux de Digital (*DNA - Digital Network Architecture*) définit les moyens par lesquels les systèmes informatiques de Digital peuvent communiquer avec d'autres systèmes informatiques fournis par Digital ou par d'autres vendeurs afin de former un système distribué. C'est une architecture en couches ; nous y reviendrons lorsque nous aborderons les objectifs de DNA.

Cette architecture spécifie :

- les protocoles de communication qui permettent aux systèmes d'échanger de l'information ; elle précise donc les règles de construction et d'interprétation des messages ;
- les interfaces internes qui permettent de décrire un protocole en termes de services fournis par la couche inférieure ;
- les mécanismes par lesquels les systèmes s'adaptent aux variations de charge et de configuration, ceci afin de permettre le meilleur usage des ressources disponibles ;
- les moyens de gérer un réseau distribué à la fois localement et à distance.

DNA semble être une architecture sans limite fixe. Ainsi, des phases ultérieures de DNA pourraient ajouter des couches, des entités dans une couche existante ou des modèles alternatifs pour certaines couches.

DECnet est un ensemble spécifique de produits qui implémente cette architecture. Il conviendrait donc de parler de l'implémentation DECnet de la Phase 1 de DNA. Toutefois, pour plus de concision, nous désignerons cela par DECnet Phase 1.

3.2. Historique de DECnet

Le développement de DECnet s'est déroulé par phases. Chacune de ces phases représente un pas important dans l'évolution des produits DECnet. Le bref historique qui va suivre est inspiré de [HARPER, 1993][1].

Phase 1

En 1974, Digital a lancé son premier produit pour l'informatique distribuée, DECnet. Avant cela, les produits permettant la communication à travers un réseau étaient souvent destinés à résoudre des problèmes spécifiques et étroitement liés à une application

---

[1] John HARPER, *Overview of Digital's Open Networking*, Digital Technical Journal, Vol. 5, N°1, Hiver 1993, pages 12-20.

particulière. DECnet, quant à lui, permettait à une application de partager des données avec n'importe quelle autre application.

Ce produit initial, plus tard appelé Phase 1, a révélé quelques problèmes inattendus. En effet, la communication et la collaboration entre plusieurs systèmes, ainsi que l'élaboration des produits que ces derniers utilisaient, n'étaient pas une chose aisée.

Un groupe a donc été chargé de produire des spécifications détaillées des protocoles et interfaces à utiliser. Toutefois, les constructeurs ne devaient pas être contraints à implémenter ces spécifications d'une façon particulière.

Phase II

La Phase II de DECnet, introduite en 1978, offrait, grâce à l'adhésion à une architecture rigoureusement spécifiée, une interopérabilité complète entre les différentes implémentations.

Toutefois, les systèmes devaient encore être directement connectés pour communiquer, le routage n'existait pas.

Phase III

Avec la Phase III, lancée en 1981, est apparue la capacité de router des messages à travers un certain nombre de systèmes intermédiaires pour atteindre une destination. La Phase III permettait également la gestion des réseaux à distance et l'utilisation des réseaux X.25 pour connecter des systèmes. Ceci reflétait l'émergence graduelle de standards pour les télécommunications.

La taille maximale d'un réseau utilisant DECnet Phase III était de 255 noeuds.

Phase IV

L'apparition des réseaux locaux, en particulier Ethernet, a eu un grand impact sur les télécommunications. Pour la première fois, on pouvait connecter un système à un réseau simplement et à bas prix. DECnet Phase IV, apparu en 1984, supporte Ethernet.

Un réseau utilisant DECnet Phase IV peut relier au maximum 65536 noeuds.

Phase V

Lorsque DECnet est apparu en 1974, tous ses protocoles étaient propriétaires ; ils étaient développés par Digital et restaient sous le contrôle de Digital. A ce moment, les standards et les protocoles définis publiquement n'existaient pas encore. C'est en 1978 qu'est apparue l'idée d'interconnexion des systèmes ouverts (*OSI - Open Systems Interconnection*). Ce projet a été entrepris par l'organisation internationale pour la standardisation (*ISO - International Organization for Standardization*). Il visait le développement

d'un ensemble complet de protocoles standards qui permettraient aux ordinateurs, non seulement d'échanger des données, mais également d'utiliser ces données de façon significative dans leurs applications.

Lorsque DECnet Phase IV est devenu disponible en 1984, les travaux sur OSI avaient fait d'importants progrès. Le modèle d'architecture avait été publié comme standard international et la standardisation de nombreux protocoles était bien avancée. Il était également clairement apparu que les communications entre des systèmes provenant de divers fournisseurs joueraient un rôle important dans l'avenir des réseaux informatiques.

Une analyse détaillée des protocoles d'OSI montra qu'ils formaient une base convenable pour l'évolution de DECnet. La décision a donc été prise d'utiliser autant que possible les standards OSI dans la prochaine phase de DECnet, DECnet Phase V. Les protocoles propriétaires existants ont été maintenus uniquement dans le but de permettre la compatibilité avec les phases antérieures.

Etant donné que notre programme est destiné à être utilisé avec DECnet Phase IV, nous allons nous intéresser de plus près à cette phase. Cependant, pour tenir compte de l'état de l'art en la matière, nous présenterons également la dernière version de DECnet, DECnet Phase V. Mais avant de considérer ces implémentations, penchons nous sur les buts de l'architecture de réseaux de Digital.

3.3. Les objectifs de DNA

L'ensemble des buts que s'était fixés Digital pour DNA a évolué dans le temps pour inclure aujourd'hui les objectifs suivants[1] :

- la transparence des opérations du réseau : les opérations internes d'un grand réseau sont inévitablement complexes. Ces opérations doivent être cachées à l'utilisateur ;
- le support d'une large gamme d'applications ;
- le support d'une grande variété de moyens de communication : les technologies de communication sont en constante évolution ; les vitesses augmentent et la nature des services change. La structure de DNA permet d'intégrer ces nouvelles technologies sans apporter de changements majeurs au reste de l'architecture ;
- le support d'un grand éventail de topologies de réseaux ;
- l'utilisation des standards chaque fois que c'est possible. Si l'élaboration de certains standards n'est pas entièrement terminée, il faut s'assurer que la transition future vers ces standards sera la plus douce possible ;
- un minimum de gestion : quand c'est faisable, DNA permet à un réseau de fonctionner sans l'intervention d'un opérateur. Ceci est particulièrement important

[1] *DECnet DIGITAL Network Architecture (Phase V) - General Description*, op cit., pages 2-3.

pour les systèmes orientés vers les utilisateurs tels que les stations de travail personnelles ;

- la facilité de gestion : toutes les fonctions de gestion ne peuvent pas être automatisées. Par exemple, certaines de ces fonctions dépendent de la politique organisationnelle de l'utilisateur. Dans de tels cas, la gestion devrait être aussi simple que possible et l'architecture ne devrait pas imposer un style particulier de gestion ;
- une possibilité de croissance sans perturbation ou reconfiguration importante : ainsi, un réseau reliant deux systèmes peut facilement évoluer vers un réseau contenant des centaines de systèmes ; pour les deux systèmes initiaux, cette évolution est transparente ;
- la possibilité de transition entre les différentes versions : chaque phase de DECnet est compatible avec la phase précédente. En conséquence, les systèmes dans un réseau peuvent être mis à jour sur une longue période. Il serait en effet gênant de devoir mettre à jour des milliers de systèmes en une nuit ;
- la possibilité d'extension à de nouveaux développements technologiques ;
- la grande disponibilité face aux pannes des systèmes et des lignes ou même, dans la mesure du possible, face aux erreurs des opérateurs ;
- la distribution : les principales fonctions de DNA, telles que le routage et la gestion de réseau, ne sont pas centralisées dans un seul système du réseau, ce qui augmente aussi la disponibilité de ce dernier;
- la mise à disposition de procédures de sécurité telles que l'authentification des utilisateurs à distance et le contrôle d'accès ;
- la segmentation : tous les systèmes ne doivent pas nécessairement implémenter toutes les fonctions proposées par l'architecture. Il est possible de communiquer avec d'autres systèmes grâce à un ensemble minimal de fonctions.

Architecture en couches

Afin de répondre aux objectifs de DNA, particulièrement ceux relatifs à la flexibilité, une architecture en couches est essentielle. La propriété d'indépendance des couches offerte par cette architecture a été largement exploitée au cours du développement de DECnet, permettant la modification ou le remplacement relativement aisé de protocoles.

3.4. DECnet Phase IV

Intéressons-nous d'abord à la Phase IV de l'architecture de réseaux de Digital. Au cours de cette section, nous présenterons le principe et les couches de cette phase. Puis, nous comparerons cette architecture au modèle OSI. Ensuite, nous examinerons de plus près les différentes couches définies par DNA. Nous terminerons en analysant un type particulier de message DECnet. Cette section est principalement basée sur [DIGITAL, 82][1].

3.4.1. Présentation

DNA Phase IV définit les couches suivantes :

- la couche utilisateur (*User Layer*) regroupe principalement des programmes écrits par les utilisateurs. Elle contient également le programme de contrôle du réseau qui permet aux gestionnaires des systèmes d'accéder, via un terminal, aux couches inférieures ;
- la couche de gestion de réseaux (*Network Management Layer*) permet aux utilisateurs de contrôler et de surveiller les activités du réseau. La gestion de réseau donne également des informations utiles pour la planification de l'évolution du réseau et la résolution des problèmes du réseau. Cette couche est la seule qui peut accéder directement à chacune des couches inférieures.
  Etant donné que notre programme ne s'intéresse pas aux activités de cette couche, nous ne développerons pas plus en détail les fonctions de celle-ci ;
- la couche application réseau (*Network Application Layer*) fournit des services génériques à la couche utilisateur. Ces services concernent notamment l'accès à des fichiers qui se trouvent sur d'autres machines, le transfert de ces fichiers, l'accès interactif, via un terminal, à un système éloigné, ... Cette couche contient à la fois des modules fournis par Digital et des modules écrits par l'utilisateur.
  Puisque notre programme nous a conduits à examiner plus en détail le protocole CTERM, nous nous intéresserons uniquement à ce dernier. CTERM permet à un terminal d'accéder à un système hôte éloigné en vue d'y lancer des commandes interactives et d'y exécuter des programmes d'application ;
- la couche de contrôle de session (*Session Control Layer*) définit la communication entre programmes via une liaison logique tout en intégrant le système utilisé. Elle s'occupe de la conversion des noms des noeuds en adresses, de l'adressage et parfois du contrôle d'accès. Cette couche utilise le protocole de contrôle de session (*Session Control Protocol*) ;

---

[1] *DECnet DIGITAL Network Architecture (Phase IV) - General Description*, Digital Equipment Corporation, Maynard Massachusetts, Mai 1982.

- la couche de communication de bout en bout (*End Communication Layer*) est responsable de la création et de la gestion des liaisons logiques indépendamment du système utilisé. Elle s'occupe du contrôle de flux, de la gestion des erreurs en bout à bout ainsi que de la segmentation et la reconstitution des messages. Cette couche utilise le protocole de services de réseau (*Network Services Protocol*) ;
- la couche de routage (*Routing Layer*) transmet les données de l'utilisateur vers leur destination. Cette couche utilise le protocole de routage (*Routing Protocol*) ;
- la couche de liaison de données (*Data Link Layer*) crée un chemin de communication entre deux noeuds adjacents et assure l'intégrité des données transmises sur ce chemin. Plusieurs protocoles sont utilisés simultanément et indépendamment les uns des autres à ce niveau : le protocole de messages de communication des données de Digital (*DDCMP - Digital Data Communications Message Protocol*), les niveaux 2 et 3 de X.25 et Ethernet ;
- la couche de liaison physique (*Physical Link Layer*) s'occupe de la transmission physique des données sur une liaison physique. Elle est notamment chargée de surveiller les signaux sur cette ligne, de gérer les interruptions provenant du matériel et d'avertir la couche de liaison de données quand une transmission est terminée. Cette couche n'est pas définie par DNA mais par un ensemble de standards issus de l'industrie tels que EIA RS-232C ou la couche physique d'Ethernet. Nous ne nous y arrêterons pas.

Relations entre les couches

Chaque couche utilise les services de la couche qui lui est inférieure. La couche utilisateur utilise également les services de la couche application réseau. Les trois couches supérieures peuvent aussi directement faire appel aux services de la couche de contrôle de session. Par ailleurs, la couche de gestion de réseau utilise les services de la couche de liaison de données pour les fonctions qui ne nécessitent pas la création d'une liaison logique.

La figure n°3.2 montre les relations entre les différentes couches définies par DNA.

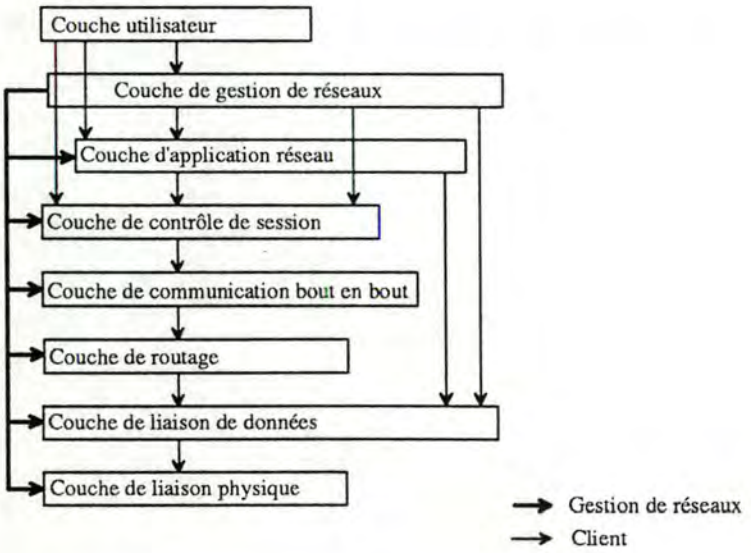

Figure n°3.2 : DNA Phase IV

Flux des données

Les données que l'utilisateur désire envoyer traversent ces différentes couches avant d'être transmises sur la liaison physique. Chaque couche ajoute aux données qu'elle reçoit de la couche supérieure, des informations de contrôle qui lui permettent de réaliser les tâches qui lui sont assignées. Ce sont les données de l'utilisateur et les informations de contrôle qui sont transmises sur la ligne.

Lorsque les données arrivent au noeud de destination, les informations de contrôle sont enlevées au fur et à mesure que le message traverse les différentes couches le conduisant au processus destinataire.

La figure n°3.3 illustre ce processus. Dans cet exemple, la couche de gestion de réseau n'intervient pas. Les données utilisateur sont les données que le processus source désire envoyer à un processus destinataire. Les informations qu'une couche reçoit de la couche supérieure sont représentées par des carrés vides alors que les informations de contrôle qu'elle ajoute sont symbolisées par un carré intitulé "Info Contrôle".

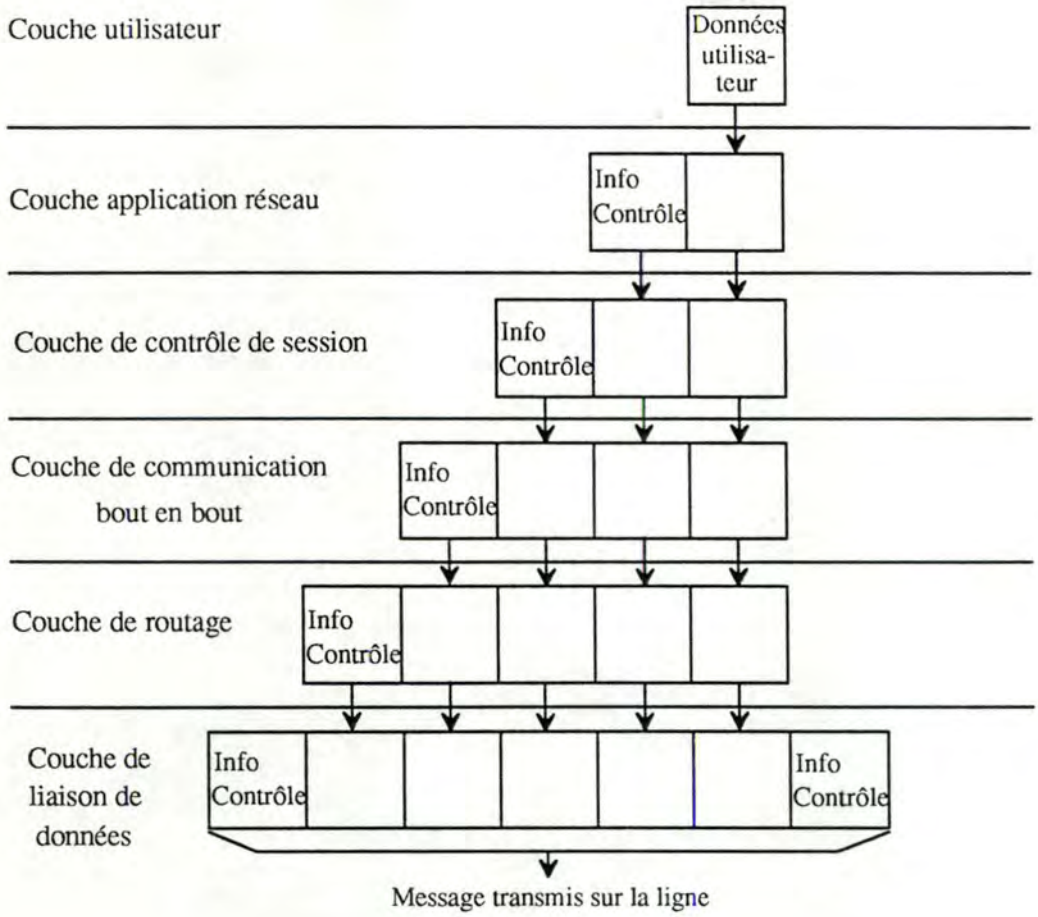

Figure n°3.3 : Flux des données au travers des couches

3.4.2. Comparaison avec OSI

L'architecture de DNA comprend plusieurs similitudes avec le modèle OSI. La couche utilisateur de DNA reprend un ensemble de fonctions qui appartiennent à la couche application d'OSI. La couche application réseau de DNA rassemble des fonctions qui appartiennent à la couche application et des fonctions qui relèvent de la couche présentation du modèle OSI. La couche de contrôle de session correspond à la couche session du modèle OSI et la couche de communication de bout en bout a les mêmes fonctions que la couche transport d'OSI. La couche de routage correspond à la couche réseau d'OSI. Les couches de liaison de données et physique ont toutes deux leur équivalent en OSI.

La couche de gestion de réseau, quant à elle, n'intervient pas explicitement dans le modèle OSI.

La figure n°3.4 présente les couches de DNA Phase IV par rapport à celles d'OSI.

<table>
<tr><th>OSI</th><th colspan="2">DNA</th></tr>
<tr><td>Application</td><td colspan="2">Utilisateur</td></tr>
<tr><td>Présentation</td><td rowspan="6">Application réseau</td><td rowspan="7">Gestion de réseaux</td></tr>
<tr><td>Session</td><td>Contrôle de session</td></tr>
<tr><td>Transport</td><td>Communication bout en bout</td></tr>
<tr><td>Réseau</td><td>Routage</td></tr>
<tr><td>Liaison de données</td><td>Liaison de données</td></tr>
<tr><td>Physique</td><td>Liaison physique</td></tr>
</table>

Figure n°3.4 : Comparaison entre DNA Phase IV et OSI

3.4.3. La couche de liaison de données

Nous l'avons déjà dit, la couche de liaison de données est responsable de la création d'un chemin de communication entre deux noeuds adjacents.

Les fonctions de cette couche sont les suivantes :

- assembler les bits reçus de la couche physique afin de former les messages ;
- détecter le début et la fin des messages reçus de la couche physique ;
- découper les messages reçus de la couche cliente afin de les envoyer sur la liaison physique reliant les deux noeuds qui communiquent ;
- vérifier l'intégrité des messages reçus de la couche physique ;
- gérer l'utilisation des ressources de la liaison physique ;
- éventuellement, garantir que les données transmises ne soient pas erronées et arrivent dans l'ordre où elles ont été envoyées.

Plusieurs protocoles résident dans la couche de liaison de données de DNA : Ethernet, DDCMP et les niveaux 2 et 3 de X.25 ; nous l'avons déjà mentionné. Nous allons donner les grandes caractéristiques de ces protocoles. Notons que les messages DECnet sont interceptés par notre programme sur un réseau Ethernet.

1. Le protocole DDCMP

DDCMP est un protocole de Digital spécialement conçu pour DNA en 1974. Ce protocole assure que les données transmises arriveront dans l'ordre dans lequel elles ont été envoyées et qu'elles seront correctes. Il garantit également la transparence des données ; n'importe quelle suite de bits peut être envoyée par le client de la liaison de données. Puisqu'elle n'a pas de signification pour la couche de liaison de données, cette suite ne sera pas mal interprétée par cette couche.

Respect de la séquence

Afin de respecter l'ordre dans lequel les données ont été transmises, la couche de liaison de données attribue aux différents messages à envoyer un numéro de séquence. Ce numéro est augmenté de un pour chaque nouveau message.

Contrôle d'erreurs

Dans le but d'assurer l'intégrité des données, la couche de liaison de données ajoute à chaque message transmis deux codes de détection d'erreurs de 16 bits ; le premier porte sur l'en-tête du message et la valeur du second est calculée à partir des données présentes dans le message. Lorsque le destinataire reçoit un message, il recalcule la valeur des codes et compare ces valeurs à celles qui sont présentes dans le message ; si les valeurs sont identiques, il en déduit que les données ne contiennent pas d'erreur, les transmet à la couche de routage et envoie un accusé de réception positif à l'émetteur. Cet accusé de réception contient le numéro de séquence du message qui a bien été reçu. Si les valeurs sont différentes, le récepteur demande la retransmission du message. Cela peut se passer de la façon suivante :

Chaque fois que l'émetteur envoie un message, il enclenche un chronomètre. Si, après un certain laps de temps, l'émetteur n'a pas reçu d'accusé de réception, il envoie au récepteur un message qui le force à envoyer un accusé de réception. Si ce dernier est négatif, l'émetteur envoie de nouveau le message, sinon, il envoie le message suivant.

Amélioration des performances

Plusieurs techniques peuvent être utilisées dans le but d'améliorer les performances lors de l'échange de messages.

D'une part, le récepteur peut envoyer un accusé de réception en même temps qu'un message de données destiné à l'émetteur ; c'est la méthode du piggybacking. Un champ du message de données contient alors le numéro de séquence du message dont on accuse réception.

D'autre part, l'émetteur peut envoyer plusieurs messages avant de recevoir un accusé de réception ; le récepteur ne doit pas nécessairement accuser réception de tous les messages. Lorsque l'émetteur reçoit un accusé de réception positif, cela signifie que tous les messages, jusques et y compris celui dont le numéro se trouve dans cet accusé, ont été bien reçus. Un accusé de réception négatif signifie que tous les messages, dont le numéro de séquence est inférieur à celui mentionné dans l'accusé de réception, ont été bien reçus. Seuls les messages dont le numéro de séquence est supérieur ou égal à celui présent dans l'accusé de réception doivent être retransmis.

Ce protocole fonctionne aussi bien avec des lignes synchrones qu'asynchrones.

2. Le protocole X.25

La recommandation X.25 du Comité Consultatif International de Télégraphie et Téléphonie (*CCITT - Consutative Committee for International Telegraph and Telephone*) définit une interface standard entre un système informatique et un point d'accès à un réseau public à commutation par paquets.

Ces réseaux publics sont disponibles dans beaucoup de pays et représentent, pour les organisations qui désirent disposer d'un réseau, une solution plus économique que les lignes louées.

Trois niveaux

La recommandation du CCITT structure X.25 selon trois niveaux :

- le niveau physique (*Physical Level*) définit les caractéristiques mécaniques, électriques, fonctionnelles et procédurales de la liaison physique reliant le système au point d'entrée du réseau ;
- le niveau des trames (*Frame Level*) définit le protocole de liaison de données entre le système et le point d'entrée du réseau ;
- le niveau des paquets (*Packet Level*) définit le format des paquets et les procédures de contrôle pour l'échange des paquets de contrôle et de données entre le système et le point d'entrée du réseau.

Dans le cas de DNA, le niveau physique réside dans la couche physique et les deux couches supérieures se retrouvent dans la couche de liaison de données.

Service de circuit virtuel

Ce protocole offre un service de circuit virtuel entre deux systèmes. Ce service a les propriétés suivantes :

- le transfert des données est full-duplex, c'est-à-dire que des messages peuvent être transmis simultanément dans les deux directions ;
- les messages arriveront dans l'ordre dans lequel ils ont été transmis ;
- ce service est fiable : les messages reçus ne contiennent pas d'erreur et aucun message n'arrive en double.

Avant tout échange de données, il est nécessaire d'établir un circuit virtuel. Plusieurs circuits virtuels peuvent être établis sur une même liaison physique ; c'est le multiplexage.

Contrôle de flux

Ce protocole assure également un contrôle de flux grâce à la technique assez courante de la fenêtre. La fenêtre détermine le nombre de messages que l'émetteur peut envoyer sans avoir reçu d'accusé de réception. Elle spécifie les numéros des messages que l'on peut émettre ; la fenêtre se déplace au fur et à mesure que l'émetteur reçoit des accusés de réception pour les messages qu'il a envoyés. Le déplacement de la fenêtre est donc contrôlé par le récepteur. Ce contrôle de flux se fait indépendamment pour chaque direction de transmission sur chaque circuit virtuel.

3. Le protocole Ethernet

Le protocole Ethernet trouve ses origines au Centre de Recherche Palo Alto de Xerox. Il a ensuite été standardisé par la norme de réseaux locaux 802.3 de l'IEEE.

Ce protocole définit une couche physique et une couche de liaison de données. Ces deux couches se retrouvent dans la Phase IV de DNA.

Les messages sont interceptés par notre programme sur la dorsale du trafic interne du CERN. Cette dorsale implémente le protocole Ethernet.

Carrier-Sense Multiple-Access with Collision-Detection

La configuration de base de tout Ethernet est un bus partagé, généralement un câble coaxial. Toutefois, la paire torsadée et la fibre optique sont de plus en plus utilisées. La méthode d'accès au bus est basée sur l'écoute du bus et la détection des collisions (*CSMA/CD - Carrier-Sense Multiple-Access with Collision-Detection*).

Lorsqu'un système désire envoyer des données sur le bus, il écoute si ce dernier est libre. Si un autre noeud est en train d'émettre, le système attend. Sinon, le système commence à envoyer ses données. Cependant, il est possible que deux noeuds aient écouté le support au même moment. Ayant tous deux observé que celui-ci était libre, ils commencent à émettre simultanément. Il en résulte une collision, semblable à ce qui se passe lorsque plusieurs personnes parlent en même temps. Lorsque les deux noeuds détectent la collision, ils arrêtent d'émettre. Ils attendent alors un temps aléatoire avant d'écouter de nouveau le support. Lorsque la liaison de données a pu envoyer les données sans collision, elle en avertit la couche supérieure.

Afin de détecter les éventuelles collisions, il est nécessaire que les messages envoyés restent suffisamment longtemps sur le support. C'est pour cette raison que la longueur minimale d'un message Ethernet est de 64 bytes. Si nous enlevons les quelques 18 bytes d'informations de contrôle, nous obtenons des messages contenant au minimum 46 bytes de données.

Afin de donner l'occasion à tous les noeuds d'envoyer des données, le protocole s'assure qu'un noeud ne garde pas la main pendant une période trop longue. C'est pour cette raison que les messages Ethernet ont une longueur maximale ; ils ne peuvent pas contenir plus de 1500 bytes de données.

Service à datagramme

Ce protocole offre un service à datagramme ; il est possible que des messages, aussi appelés trames, se perdent. Le contrôle des erreurs sera assuré par une couche supérieure. Cela se justifie par le faible taux d'erreurs des liaisons Ethernet.

Chaque message transmis contient l'adresse du système qui l'a envoyé ainsi que l'adresse du système à qui il est destiné.

La liaison de données traite les données de façon transparente ; les données peuvent contenir n'importe quelle séquence de bits.

3.4.4. La couche de routage

Nous l'avons déjà mentionné, la couche de routage achemine les messages vers leur destination. Avant d'atteindre leur destination, les messages, aussi appelés paquets à ce niveau, peuvent transiter par un certain nombre de noeuds intermédiaires. La couche de routage choisit le chemin que doivent suivre les paquets pour atteindre leur destination.

Le routage implémente un service à datagramme ; il est donc possible que certains paquets se perdent, arrivent en double ou encore que l'ordre dans lequel les paquets arrivent ne corresponde pas à l'ordre dans lequel ils ont été envoyés.

Fonctions

Les fonctions de la couche de routage sont les suivantes :

- déterminer le chemin que doivent suivre les paquets entre leur noeud source et leur noeud de destination. Un chemin est une séquence de liaisons de données et de noeuds. Pour choisir le meilleur chemin, la couche de routage se base sur les coûts que les responsables des réseaux attribuent aux différents circuits. Ainsi, s'il existe plus d'un chemin entre deux noeuds, la couche de routage choisit celui dont le coût est minimum ;
- faire suivre les paquets. Si le paquet qui arrive à un noeud est destiné à ce noeud, la couche de routage transmet ce paquet à la couche supérieure ; si le paquet est destiné à un autre noeud, la couche de routage l'envoie vers le noeud suivant du chemin ;
- s'adapter à la topologie du réseau. Si un noeud ou une ligne tombe en panne, la couche de routage trouve un autre chemin par lequel envoyer un paquet, si un tel chemin existe ;
- s'adapter aux différents types de liaisons de données. La couche de routage supporte des chemins constitués de liaisons de données de différents types ;
- informer les autres entités de routage des changements observés dans la topologie du réseau ;
- renvoyer à l'émetteur les paquets adressés à des noeuds inaccessibles ;
- limiter le nombre de noeuds qu'un paquet peut visiter et empêcher ainsi que des paquets bouclent indéfiniment dans le réseau.

Il est possible que certains noeuds n'implémentent qu'une partie des fonctionnalités offertes par la couche de routage. Ces noeuds sont capables d'envoyer et de recevoir des messages mais ils ne sont pas aptes à router les messages de leur source vers leur destination. A l'opposé, les routeurs implémentent toutes les fonctions de la couche de routage et sont donc capables d'acheminer les messages vers leur noeud de destination.

Adaptation à la topologie du réseau

La façon dont la couche de routage prend connaissance de la topologie du réseau et s'adapte aux changements de topologie dépend du type de la liaison de données. Voyons comment cela se passe sur un réseau Ethernet.

Régulièrement, les routeurs envoient un message (*Ethernet Router Hello Message*) à tous les autres noeuds, routeurs ou non. Ce message contient la liste de tous les routeurs sur le réseau Ethernet qui ont récemment envoyé un message au noeud envoyant le présent

message. En s'échangeant de tels messages, les routeurs restent informés du statut des autres routeurs sur le réseau.

Les noeuds finaux (*End Nodes*) envoient régulièrement des messages (*Ethernet Endnode Hello Message*) à tous les routeurs. Ces messages permettent aux routeurs de connaître le statut des noeuds finaux sur l'Ethernet.

Grâce aux messages reçus, les routeurs établissent une base de données indiquant le meilleur chemin à suivre pour atteindre chaque noeud. Cette base de données est mise à jour au fur et à mesure que de nouveaux messages arrivent. Lorsqu'un noeud reçoit un paquet, il consulte sa base de données pour déterminer le noeud vers lequel il doit envoyer ce paquet.

De plus, la couche de routage choisit un routeur parmi tous les routeurs du réseau. Ce routeur est appelé le routeur désigné (*Designated Router*). Il reçoit les messages envoyés par les noeuds du réseau, en particulier les demandes de connexion, et les envoie vers leur destination. Lorsque ce routeur reçoit d'un noeud un message de demande de connexion adressé à un autre noeud qui se trouve sur le même Ethernet, le routeur désigné signale au noeud destinataire qu'il se trouve sur le même réseau que le noeud émetteur. Suite à cela, la communication entre les deux noeuds se fait directement.

3.4.5. La couche de communication de bout en bout

La couche de communication de bout en bout permet à deux processus de s'échanger des données de façon fiable indépendamment de leur localisation dans le réseau. Nous l'avons déjà dit, cette couche implémente le protocole de service réseau (*NSP - Network Service Protocol*).

Liaison logique

La communication entre deux processus nécessite la création d'une connexion. Cette connexion entre processus est appelée liaison logique. Une liaison logique permet à un processus situé à un bout de la liaison d'envoyer des données au processus situé à l'autre bout de la liaison. La couche de communication de bout en bout crée, maintient et détruit les liaisons logiques à la demande de la couche de contrôle de session décrite dans la section suivante.

Respect de la séquence et contrôle d'erreurs

Le protocole NSP garantit que les messages arriveront chez le destinataire dans le même ordre que celui dans lequel ils ont été émis. Il utilise pour cela un mécanisme de contrôle d'erreurs. Les messages se voient attribuer un numéro de séquence et chacun des processus communiquant via la liaison logique envoie un accusé de réception pour les messages qu'il a bien reçus. Si l'accusé de réception est négatif, l'émetteur retransmet la

donnée. De même, si l'émetteur a envoyé un message et n'a pas reçu d'accusé de réception après un certain laps de temps, il retransmet la donnée. Les accusés de réception peuvent être combinés avec des messages de données destinés à l'émetteur, c'est la méthode du piggybacking dont nous avons déjà parlé.

Segmentation des données

La longueur des messages que la couche de communication de bout en bout peut transmettre à la couche de routage est limitée. Les spécifications ne précisent pas la longueur maximale d'un message. Pourtant, les messages que la couche de contrôle de session donne à la couche de communication de bout en bout peuvent être très longs. NSP découpe donc, en segments, les messages reçus de la couche de contrôle de session. Pour chaque segment, on indique sa place dans le message, à savoir, début, milieu ou fin. Chaque segment est numéroté et envoyé à l'entité NSP réceptrice. Celle-ci utilise les numéros de séquence et les indications sur la place du segment dans le message pour reconstituer le message à partir des différents segments reçus.

Contrôle de flux

L'entité NSP réceptrice s'occupe également du contrôle de flux. Il existe trois types de contrôle de flux :

- aucun ;
- le récepteur indique à l'émetteur le nombre de segments qu'il peut recevoir ;
- le récepteur indique à l'émetteur le nombre de messages qu'il peut recevoir en provenance de la couche de contrôle de session. Selon Digital[1], cette méthode est obsolète et tend à être abandonnée.

En plus de cela, le récepteur peut toujours interrompre ou reprendre la transmission à tout moment.

Notons que les mécanismes d'établissement de la liaison logique, de contrôle d'erreurs et de contrôle de flux mis en oeuvre par le protocole NSP sont transparents aux processus utilisateurs de la liaison logique.

3.4.6. La couche de contrôle de session

La couche de contrôle de session permet aux applications, qui s'exécutent avec un certain système d'exploitation, d'utiliser les services de la couche de communication de bout en bout. Elle fournit également aux processus utilisateurs une interface unique pour envoyer ou recevoir des données ainsi qu'établir ou détruire une liaison logique. Nous l'avons

---

[1] *DECnet DIGITAL Network Architecture (Phase IV) - General Description*, op cit. page 4.7.

déjà mentionné, cette couche implémente le protocole de contrôle de session (*SCP - Session Control Protocol*).

Réception d'une demande de connexion d'un processus utilisateur

La couche de contrôle de session demande la création de liaisons logiques chaque fois que des processus utilisateurs le désirent. Lorsqu'elle reçoit une demande de connexion d'un processus utilisateur, la couche de contrôle de session accomplit les opérations suivantes :

- elle établit la correspondance entre le nom du noeud de destination, fourni par le processus dans sa demande de connexion, et l'adresse de ce noeud. L'entité de contrôle de session dispose, à cet effet, d'une table mettant en relation les noms des noeuds et leur adresse. Elle peut alors donner à la couche de communication de bout en bout l'adresse du noeud de destination ;
- la couche de contrôle de session met la demande de connexion dans le format adéquat pour la couche de communication de bout en bout ;
- elle transmet ensuite la demande de connexion à la couche de communication de bout en bout ;
- éventuellement, la couche de contrôle de session lance un chronomètre au moment où elle transmet la demande à la couche inférieure. Si, après un certain laps de temps, la demande de connexion n'a toujours pas été acceptée ou rejetée par le destinataire, la couche de contrôle de session annonce au processus source une déconnexion.

Réception d'une demande de connexion de la couche de communication de bout en bout

Lorsque la couche de communication de bout en bout transmet à la couche de contrôle de session une demande de connexion provenant d'un autre noeud, la couche de contrôle de session exécute les opérations suivantes :

- elle analyse la demande de connexion reçue afin de dégager des informations telles que les noms des processus utilisateurs à la source et à la destination ainsi que des informations de contrôle d'accès ;
- elle valide toutes les informations de contrôle d'accès ;
- elle identifie, crée ou active le processus utilisateur à la destination ;
- elle remplace l'adresse du noeud source par le nom de ce noeud. Elle utilise à cet effet sa table de correspondance ;
- elle transmet ensuite, au processus utilisateur à la destination, la demande de connexion qu'elle vient de recevoir ;

- éventuellement, au moment où elle transmet la demande de connexion au processus utilisateur, la couche de contrôle de session démarre un chronomètre. Si, après un certain laps de temps, le processus n'a pas accepté la connexion, la session envoie une demande de déconnexion à la couche de communication de bout en bout.

Envoi des données

La couche de contrôle de session est également chargée d'envoyer et de recevoir des données. Ces données sont passées directement à la couche de communication de bout en bout.

Demande de déconnexion

De plus, la couche de contrôle de session, d'une part, transmet les demandes de déconnexion reçues des processus utilisateurs à la couche de communication de bout en bout et, d'autre part, indique aux processus utilisateurs que le processus, avec lequel ils communiquent, a demandé une déconnexion.

Surveillance de la liaison logique

Enfin, la couche de contrôle de session peut éventuellement surveiller la liaison logique dans le but de :

- détecter les déconnexions qui peuvent survenir au niveau réseau, interrompant la communication entre les processus reliés par la liaison logique ;
- détecter les inaptitudes de la couche de communication de bout en bout à passer les données transmises de manière opportune.

Si un tel événement se produit, la couche de contrôle de session met fin à la liaison logique.

3.4.7. La couche application réseau

Comme nous l'avons déjà mentionné, la couche application réseau offre aux utilisateurs des services génériques d'accès aux données et de communication. Puisque notre travail nous a amenés à mieux connaître le protocole CTERM, nous allons exclusivement décrire ce dernier. Ce protocole permet à un terminal de se connecter sur un système éloigné (*Remote System*) afin d'y lancer des commandes interactives et d'y exécuter des programmes. Le système éloigné traite tous les terminaux de la même façon, que ces terminaux soient directement reliés au système ou qu'ils soient connectés au système par l'intermédiaire d'un réseau.

Le terme CTERM est une abréviation pour *Command Terminal Module*, un des composants du protocole.

On distingue deux sous-couches :

- la couche de fondation (*Foundation Layer*) est la couche inférieure. Elle est responsable de la création, du maintien et de la destruction des connexions entre les applications et les terminaux. Une telle connexion est appelée lien (*Binding*). Cette couche permet également à un terminal d'envoyer et de recevoir des données du système éloigné ;
- le module du terminal de commandes (*Command Terminal Module*) fournit des fonctions qui permettent aux terminaux de communiquer avec l'interpréteur de commandes et les programmes d'applications du système éloigné. Il fournit par exemple les fonctions qui permettent de lire une ligne en entrée, d'écrire une ligne de résultat, ...

3.4.8. Analyse d'un type particulier de message

Nous allons à présent analyser la structure d'un message de données envoyé en utilisant le protocole CTERM. Les informations qui nous ont permis d'écrire cette section sont notamment issues d'un analyseur de réseaux que nous avons eu l'occasion d'utiliser lors de notre stage.

Nous indiquerons la signification de la plupart des bytes de ce message, montrant ainsi le type d'analyse que notre programme est amené à réaliser sur les différents messages afin de dégager les informations qui lui sont utiles.

a) Vision globale du message

La figure n°3.5 donne une vision globale de la structure du message passant au niveau physique. Nous trouvons d'abord les informations de contrôle des différentes couches suivies des données de l'utilisateur qui se trouvent dans le message CTERM. Le message se termine par des informations de contrôle ajoutées par la couche Ethernet.

En-tête Ethernet	En-tête Routage	En-tête Communication bout en bout	En-tête Session	En-tête Fondation	Message CTERM	Terminaison Ethernet

Figure n°3.5 : Vision globale du message passant au niveau physique

Nous allons maintenant examiner ces informations en considérant tour à tour les différentes couches. Toutes les valeurs de bytes mentionnées lors de l'analyse du message sont exprimées en hexadécimal.

b) Au niveau de l'en-tête et de la terminaison Ethernet

La figure n°3.6 présente le format de l'en-tête et de la terminaison de la trame Ethernet.

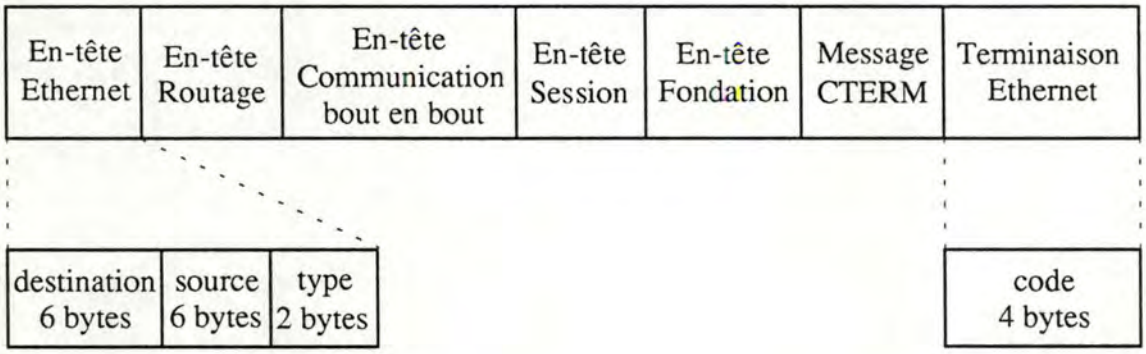

Figure n°3.6 : Format de l'en-tête et de la terminaison de la trame Ethernet

Dans l'en-tête de la trame, nous trouvons les informations suivantes :

- "destination" et "source" indiquent respectivement les adresses du noeud de destination et du noeud source de la liaison de données ;
- "type" permet d'identifier le protocole de niveau supérieur associé à la trame. Cela permet à plusieurs protocoles du niveau de la couche de routage de coexister sur le même Ethernet. La valeur 6003 correspond au protocole de routage de DECnet Phase IV.

Dans la terminaison de la trame, nous trouvons l'information suivante :

- "code" permet de détecter les erreurs de transmission.

c) Au niveau de l'en-tête de la couche de routage

La figure n°3.7 présente le format de l'en-tête du paquet. Dorénavant, nous désignerons un byte par B.

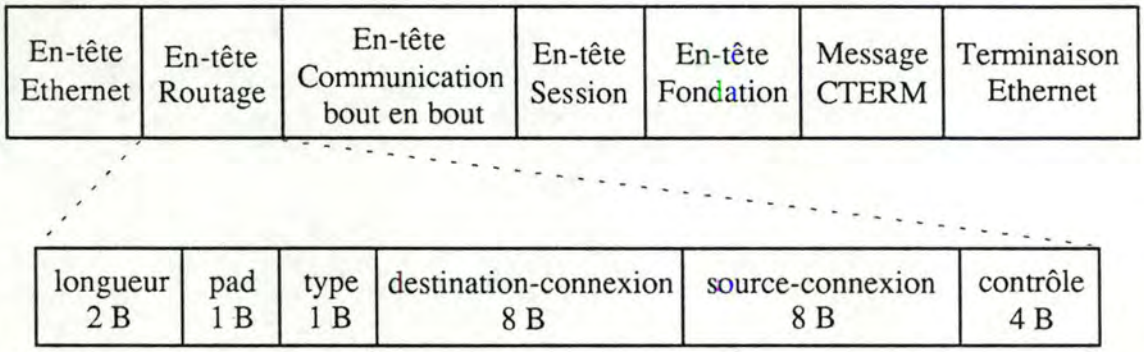

Figure n°3.7 : Format de l'en-tête du paquet

On y trouve les informations suivantes :

- "longueur" indique la longueur de la suite du paquet ;
- "pad" est un byte de remplissage, il n'est pas toujours présent. Si ce byte est présent, sa valeur est 81 ;
- "type" indique le type et le format du message ;
- "destination-connexion" et "source-connexion" sont respectivement les adresses du noeud de destination et du noeud source de la connexion ;
- "contrôle" regroupe un ensemble d'autres informations de contrôle, notamment un compteur du nombre de noeuds visités par un paquet. Ce compteur permet de jeter les paquets qui ont visité trop de noeuds. Ces informations ne sont pas utiles pour notre programme.

d) Au niveau de l'en-tête de la couche de communication de bout en bout

La figure n°3.8 présente l'en-tête du message au niveau de la couche de communication de bout en bout.

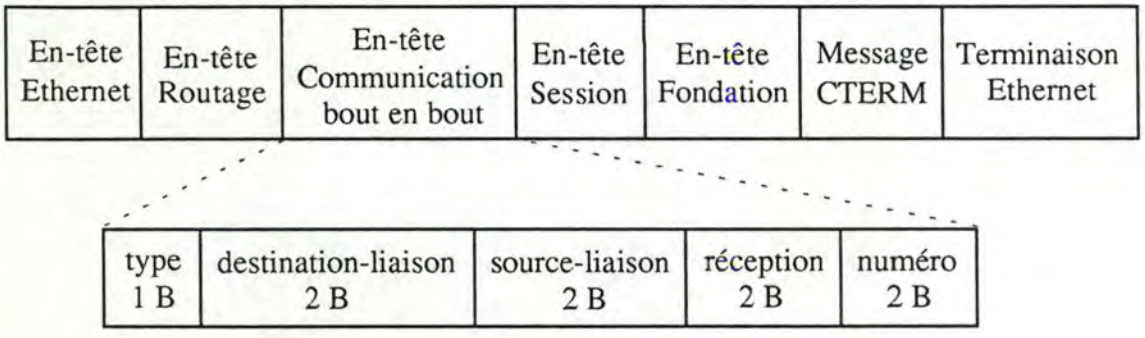

Figure n°3.8 : Format de l'en-tête du message au niveau de la couche de communication de bout en bout

On y trouve les informations suivantes :

- "type" indique le type de message. Voici quelques exemples : la valeur 18 indique une demande de connexion, la valeur 28 indique une confirmation de connexion, la valeur 60 indique un message de données, la valeur 38 indique une demande de déconnexion,... ;
- "destination-liaison" et "source-liaison" représentent respectivement les adresses source et destination de la liaison logique créée entre les processus qui communiquent ;
- "réception" est un champ facultatif. Il contient le numéro du message dont on accuse réception. Si ce champ est présent, le bit le plus significatif du champ est mis à 1 ; dans la cas contraire, il est mis à 0 ;
- "numéro" contient le numéro de séquence du message.

e) Au niveau de l'en-tête de la couche de contrôle de session

Aucune information de contrôle n'est ajoutée au niveau de la couche de contrôle de session pour un tel message. Cette couche ajoute des informations uniquement pour les messages de demande et de confirmation de connexion ainsi que les messages de demande de déconnexion.

f) Au niveau de l'en-tête de la couche de fondation et du message CTERM

La figure n°3.9 présente le format du message au niveau de la couche application réseau. Nous trouvons les informations ajoutées par la couche de fondation suivies du message CTERM.

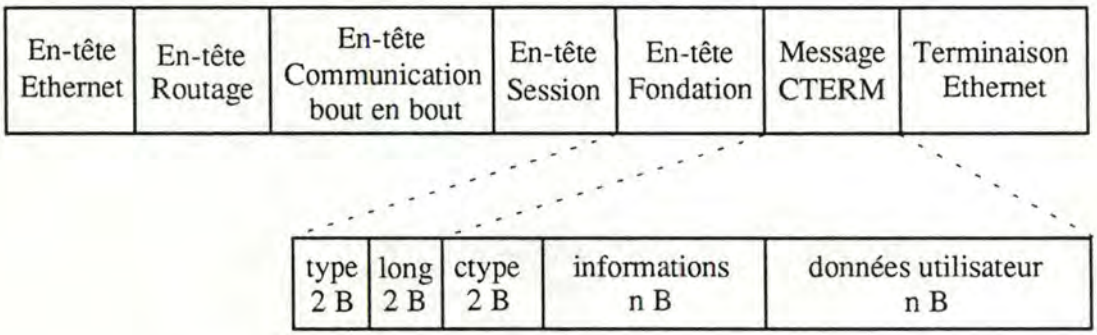

Figure n°3.9 : Format du message au niveau de la couche application réseau

Dans l'en-tête ajouté par la couche de fondation, nous trouvons les informations suivantes :

- "type" indique le type de message. La valeur 09 indique un message de données ;
- "long" donne la longueur du message CTERM qui se trouve immédiatement après cet en-tête. Plusieurs messages CTERM peuvent se trouver dans le même message. Après le premier message, on retrouve alors un nouveau champ ajouté par la couche de fondation qui donne la longueur du message CTERM suivant ;

Dans le message CTERM, nous trouvons les informations suivantes :

- "ctype" indique le type de message CTERM. La valeur 03 indique qu'il s'agit d'un message envoyé du terminal vers le système éloigné ; ce message contient donc une commande lancée par l'utilisateur depuis le terminal. La valeur 07 indique qu'il s'agit d'un message contenant une réponse envoyée par le système éloigné vers le terminal ;
- "informations" reprend un ensemble d'informations échangées entre le système éloigné et le terminal. La longueur et le contenu de cet ensemble d'informations dépendent du type du message CTERM. On y trouve des informations telles que le nombre de caractères présents dans les données, ...
  Notre programme s'intéresse principalement à la longueur de ce champ afin de localiser le début des données de l'utilisateur. Cette longueur peut être déduite du type du message CTERM ;
- "données utilisateur" contient les données qui ont été envoyées par l'utilisateur ou le système éloigné.

3.5. Pourquoi une nouvelle architecture ?

DECnet Phase IV a été conçu au début des années 80. Il s'est donc avéré intéressant, après quelques années, de repenser ses idées. En effet, plusieurs évolutions ont eu lieu.

Intégration des standards

D'une part, les standards OSI, dont le but est de permettre l'interconnexion de systèmes provenant de différents vendeurs, ont fait d'importants progrès. Fin des années 80, les standards concernant les couches inférieures sont assez complets. Digital décide donc d'intégrer autant que possible ces standards dans DECnet afin de permettre la communication entre des systèmes de Digital et d'autres systèmes.

Taille des réseaux

D'autre part, les réseaux sont de plus en plus grands ; il n'est pas rare d'avoir un réseau qui relie plus de 100.000 systèmes. DECnet Phase IV ne convient pas pour les grands réseaux ; la taille maximale d'un réseau qui implémente DECnet Phase IV est en effet de 65536 noeuds. Il est donc nécessaire d'étendre l'espace d'adressage afin de supporter ces réseaux de plus en plus importants.

Etant donné cette volonté d'étendre l'espace d'adressage et d'intégrer les standards, une nouvelle architecture est nécessaire.

Avantage d'une architecture en couches

L'architecture en couches de DECnet a permis de localiser les changements à réaliser. En effet, dans ce type d'architecture, les différentes couches subissent des modifications qui n'affectent ni les autres couches ni les applications qui utilisent DECnet.

Remarquons toutefois que Digital a sous-estimé le travail à faire. Bien que les premières spécifications de DECnet Phase V datent de 1987, les premiers logiciels ne sont apparus qu'en 1992.

3.6. DECnet Phase V

Nous allons maintenant présenter les grandes lignes de DECnet Phase V. Nous nous attarderons plus particulièrement sur les éléments qui distinguent DECnet Phase IV de DECnet Phase V ainsi que la compatibilité entre les systèmes qui implémentent DECnet Phase IV et DECnet Phase V. Nous terminerons en indiquant le lien qui existe entre DECnet Phase V et OSI.

3.6.1. Présentation

DNA Phase V définit les couches suivantes :

- la couche physique (*Physical Layer*) s'occupe principalement des aspects électriques de la communication. Les opérations réalisées par cette couche sont détaillées dans des standards tels que EIA RS-232-D, CCITT V.24 et X.21 ou ISO 8802-3. Le standard utilisé dépend du type de la communication ; ainsi, ISO 8802-3 s'applique aux réseaux locaux. DNA ne s'occupe donc que de la gestion de ces composants et leur interface avec les autres éléments de l'architecture ;
- la couche de liaison de données (*Data Link Layer*) fournit, comme nous l'avons mentionné précédemment, un chemin de communication fiable entre deux systèmes directement connectés. DECnet Phase V supporte plusieurs protocoles de liaison de données ; mentionnons DDCMP, le protocole de Digital dont nous avons déjà parlé, X.25, HDLC (*High-level Data Link Control*) qui est un protocole international et les protocoles de réseaux locaux de type CSMA/CD ;
- la couche réseau (*Network Layer*), nous l'avons déjà dit, permet de transmettre des données entre deux noeuds quelconques d'un réseau. Cette couche utilise les protocoles de réseaux et de routage OSI, offrant ainsi un service non connecté ; nous y reviendrons ultérieurement. Signalons que DNA offre également un service orienté connexion dans des systèmes reliés à des réseaux tels que X.25 ;
- la couche transport (*Transport Layer*) offre un service fiable de bout en bout entre deux systèmes qui communiquent, comme nous l'avons vu précédemment. Contrairement aux couches inférieures, cette couche et les couches supérieures ne sont présentes que dans les systèmes finaux (*End Nodes*). DECnet Phase V supporte deux protocoles au niveau de la couche transport : NSP, défini pour les phases précédentes de DECnet, le protocole de transport OSI.

Couches supérieures

Deux types d'accès à la couche transport sont possibles :

D'une part, la couche de contrôle de session (*Session Control Layer*) de DNA, que nous avons présentée au début de ce chapitre, est utilisée par les applications de DNA pour accéder aux services offerts par la couche transport.

D'autre part, OSI définit trois couches au-dessus de la couche transport. Ces trois couches sont présentes dans DECnet Phase V et sont utilisées par les applications OSI afin d'accéder aux services offerts par la couche transport.

- La couche session (*Session Layer*) organise la structure des échanges de messages ;
- La couche présentation (*Presentation Layer*) gère l'existence de plusieurs représentations des données dans différents systèmes. Ainsi, elle permet aux différents systèmes impliqués dans la communication de choisir une syntaxe commune de transfert que chacun comprend ;
- La couche application (*Application Layer*) est divisée en un certain nombre d'éléments de service (*Application Service Elements*), chacun offrant un ensemble spécifique de fonctions aux applications.

DNA Phase IV, qui est également supporté par DNA Phase V, ne possède pas cette structure. Si cela s'avère nécessaire, certaines fonctions de la couche session OSI et les fonctions de la couche présentation OSI sont intégrées dans les applications de DNA.

Au-dessus de ces couches, on trouve les applications de Digital, qui reposent sur des protocoles tels que CTERM, et les applications OSI, telles que FTAM et VTP. L'utilisateur peut également développer ses propres applications.

Relations entre les couches

La figure n°3.10 ci-dessous présente les différentes couches de DNA Phase V.

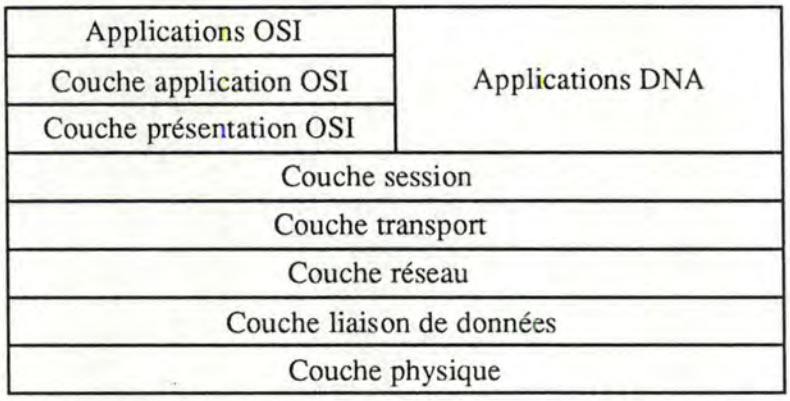

Figure n°3.10 : DNA Phase V

3.6.2. Les changements clés par rapport à DECnet Phase IV

Plusieurs changements distinguent DECnet Phase IV de DECnet Phase V. Nous allons mentionner les principaux.

Signalons immédiatement que le modèle de gestion de réseaux a changé. En effet, le modèle de gestion de réseaux de DECnet Phase V est basé sur les spécifications du modèle d'OSI, à savoir CMIP (*Common Management Information Protocol*). De plus, DECnet Phase V supporte également SNMP (*Simple Network Management Protocol*), le protocole de gestion de réseaux de TCP/IP. Nous ne reviendrons plus sur ce point.

1. La terminologie

Notons d'abord que le nom de certaines couches a changé, ceci dans le but de s'aligner sur les noms d'OSI.

Ainsi, la couche réseau (*Network Layer*) était auparavant appelée couche de routage (*Routing Layer*). De même, la couche transport (*Transport Layer*) portait le nom de couche de communication de bout en bout (*End Communication Layer*). Enfin, la couche physique (*Physical Layer*) était appelée couche de liaison physique (*Physical Link Layer*).

Notons également que les couches utilisateur (*User Layer*) et d'application réseau (*Network Application Layer*) de DECnet Phase IV se retrouvent regroupées, dans DECnet Phase V, au niveau des applications.

2. La couche de liaison de données

Au niveau de la couche de liaison de données, DNA Phase V supporte HDLC en plus de DDCMP, X.25 et des protocoles de réseaux locaux. HDLC est défini dans plusieurs standards ISO :

- ISO 3309 - HDLC frame structure
- ISO 4335 - HDLC elements of procedure
- ISO 7809 - HDLC classes of procedure
- ISO 8471 - HDLC data link address resolution
- ISO 8885 - HDLC general purpose XID information field content and format

Ce protocole de liaison de données offre un service fiable assurant, entre autres, le respect de la séquence et la détection des erreurs, le tout étant transparent pour l'utilisateur.

3. La couche réseau

Deux changements importants apparaissent au niveau de la couche réseau ; l'un concerne les adresses, l'autre, les protocoles de routage.

Les adresses

Premièrement, la structure et la taille des adresses sont modifiées. Pour la structure des adresses, DNA a adopté le standard ISO 8348 Addendum. Le format des adresses est maintenant hiérarchique.

De plus, alors que les adresses de DECnet Phase IV ont une taille fixe de 16 bits, la taille des adresses de DECnet Phase V est variable. La taille maximale d'une adresse de DECnet Phase V est de 20 bytes, autorisant ainsi la présence d'un nombre beaucoup plus important de machines sur le réseau.

Les protocoles de routage

Secondement, les protocoles de routage utilisés dans la Phase V sont des standards internationaux, en particulier les standards End System to Intermediate System Routing Protocol - ISO 9542 - et Intermediate System to Intermediate System Protocol. L'algorithme utilisé pour la détermination du chemin le plus court est celui de Dijkstra.

Avant de présenter la technique utilisée pour le routage, nous allons définir quelques concepts essentiels pour le routage de DECnet Phase V. Un ensemble de systèmes gérés par une même organisation est appelé un domaine administratif (*Administrative Domain*). Un domaine administratif peut être divisé en un certain nombre de domaines de routage (*Routing Domains*). Un domaine de routage est un ensemble de systèmes qui utilisent tous les mêmes procédures de routage. Un domaine de routage peut lui-même être partitionné en plusieurs régions (*Areas*). Chaque système appartient à une seule région. Le routage au sein d'une région est appelé routage de niveau 1 (*Level 1 Routing*). Les routeurs responsables du routage de niveau 1 sont les routeurs de niveau 1 (*Level 1 Router*). Le routage entre régions est appelé routage de niveau 2 (*Level 2 Routing*). Les routeurs responsables du routage de niveau 2 sont les routeurs de niveau 2 (*Level 2 Router*).

Le routage entre les domaines de routage est statique ; le responsable du réseau doit lui-même communiquer les informations de routage aux différents systèmes. Ces informations sont rassemblées dans des tables maintenues par le responsable de chaque domaine.

Pour le routage au sein d'un domaine de routage, les routeurs s'échangent des informations de routage.

Régulièrement, les routeurs s'envoient des messages (*Link State Packets*) contenant des informations relatives à l'état de leurs liaisons et aux coûts de ces liaisons. Recevant des messages en provenance de tous les autres systèmes, chaque système est en possession des informations concernant les liaisons de tous les autres systèmes. Cette information est utilisée par les systèmes pour déterminer la topologie du réseau et calculer le meilleur chemin que doit suivre un paquet pour atteindre une certaine destination. Contrairement à ce qui se passe dans DECnet Phase IV, chaque routeur garde une trace de l'état global du réseau et chaque fois qu'un paquet arrive, il calcule le meilleur chemin à suivre.

Si un paquet est destiné à un noeud d'une autre région, le routeur de niveau 1 qui a reçu le paquet l'envoie au routeur de niveau 2 le plus proche, sans tenir compte de la destination du paquet. Le paquet est ensuite transmis vers sa région de destination, en passant éventuellement par plusieurs routeurs de niveau 2. Lorsqu'il arrive au routeur de niveau 2 de sa région de destination, le paquet est de nouveau véhiculé via le routage de niveau 1 jusqu'au système de destination.

La figure n°3.11 illustre ce processus. Dans ce dessin, A désire envoyer un message à B.

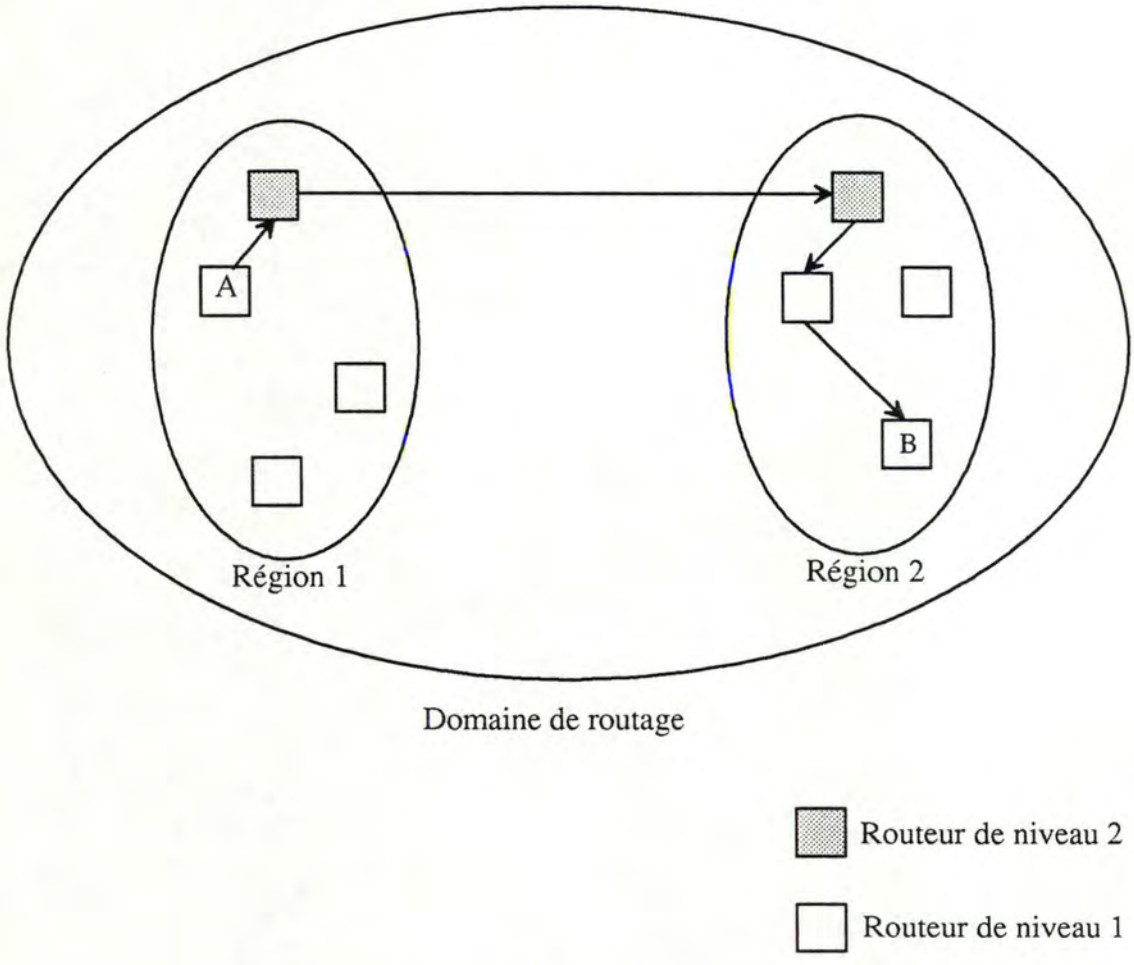

Figure n°3.11 : Routage entre régions

4. La couche transport

Nous avons déjà vu que plusieurs protocoles, à savoir OSI et NSP, coexistent au niveau de la couche transport. Attardons-nous quelque peu sur les services offerts par le protocole OSI.

Le standard OSI, spécifié par le standard ISO 8073, définit 5 classes de protocoles, numérotées de 0 à 4. DNA offre les classes 4, 2 et 0, la classe 4 étant celle qui offre le plus de services.

La présence de ce protocole OSI permet non seulement la communication entre des systèmes DNA Phase V mais aussi la communication entre des systèmes DNA Phase V et des systèmes non DNA qui implémentent ce protocole.

Le choix du protocole à utiliser lors d'une communication particulière - NSP, OSI classe 0, 2 ou 4 - se fait lors de l'établissement de la connexion. Une interface unique est proposée aux utilisateurs de la couche transport.

5. Le répertoire distribué

Les protocoles des couches inférieures travaillent avec des adresses. Le format de ces adresses est imposé par les protocoles et leur valeur est imposée par la topologie du réseau ou par le matériel. Ces adresses ne peuvent donc pas être facilement mémorisées par les utilisateurs du réseau. Heureusement, ces derniers peuvent accéder aux objets du réseau sans savoir où ils se trouvent, en les désignant par leur nom. Par conséquent, les ordinateurs dans le réseau doivent être capables, étant donné un nom, de déterminer l'adresse qui lui est associée et inversement.

La solution proposée par DECnet Phase IV est très simple : chaque système conserve la totalité de la table qui établit la correspondance entre les noms et les adresses. Les réseaux qui implémentent DECnet Phase IV étant de taille relativement petite, cette solution est possible. Cependant, DECnet Phase V est destiné à être utilisé avec des réseaux de taille beaucoup plus grande. La solution adoptée par la Phase IV n'est donc plus envisageable.

Digital a donc entrepris des travaux pour un service de répertoire distribué. Ce service a reçu le nom de DECdns (*Digital Distributed Name Service*).

Ce service fournit :

- la distribution : toutes les informations concernant la correspondance entre les noms et les adresses ne doivent pas être regroupées à un seul endroit du système ;
- la réplication : une information peut se retrouver à plusieurs endroits. Ceci permet une plus grande disponibilité de l'information face aux pannes ;
- une mise à jour dynamique : une information peut être changée à tout moment ;
- une mise à jour automatique : les changements et les nouvelles informations sont automatiquement propagés dans le réseau ;
- la possibilité d'avoir des noms hiérarchiques : un nom peut avoir plusieurs composants afin de refléter la structure d'une organisation ou d'une administration.

DECnet supporte également le service de noms d'OSI, X.500.

Ce service est particulièrement utilisé par la couche de contrôle de session afin de traduire les noms en adresses, mais ces fonctions sont également directement disponibles aux applications.

3.6.3. La compatibilité avec DECnet Phase IV

Conformément aux objectifs de DNA, un système qui implémente DNA Phase V est entièrement capable de communiquer avec un système qui implémente DNA Phase IV.

Un réseau Phase IV peut donc migrer graduellement vers un réseau Phase V. Il n'est pas nécessaire d'organiser la transition d'un grand nombre de systèmes simultanément. Pour l'utilisateur d'un réseau DECnet, la transition de la Phase IV à la Phase V se fait de façon transparente. La seule différence perceptible pour ce dernier est la disponibilité de nouvelles fonctions.

Remarquons une nouvelle fois que pour des protocoles tels que CTERM, rien ne change. Toutes les applications qui tournaient avec DECnet Phase IV continuent à tourner avec DECnet Phase V.

3.6.4. La relation avec OSI

Nous avons pu remarquer que les couches du modèle de DNA Phase V correspondent relativement bien aux sept couches du modèle OSI. Ceci est encore visible sur la figure n°3.12. Selon Digital[1], cette correspondance reflète, d'une part, l'origine d'OSI qui s'inspire des architectures de réseaux existantes y compris DNA et, d'autre part, la volonté de DNA de s'aligner sur les standards internationaux.

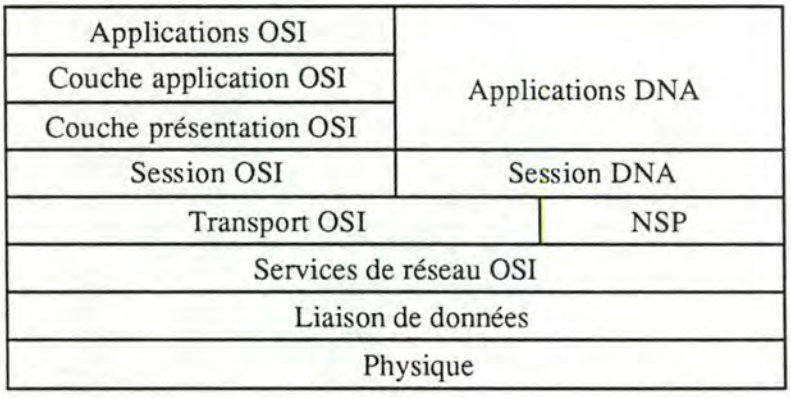

Figure n°3.12 : Relation entre DNA Phase V et OSI

[1] *DECnet DIGITAL Network Architecture (Phase V) - General Description*, op cit., page 11.

Tous les protocoles utilisés jusqu'à la couche transport sont des protocoles OSI, les protocoles propriétaires étant maintenus pour des raisons de compatibilité.

Au-dessus de la couche transport, les protocoles OSI peuvent être utilisés. Toutefois, ils ne sont pas utilisés par toutes les applications. Digital[1] explique ce phénomène par plusieurs éléments :

- seules quelques-unes des applications proposées par DNA sont étudiées dans le but d'une standardisation OSI ;
- au moment de la conception de DECnet Phase V, le seul standard de couche supérieure qui avait été approuvé comme standard international était celui de la couche session. Les autres standards étaient sujets à des modifications avant l'approbation finale ;
- la communication avec des implémentations précédentes de DNA est fondamentale. Or, cela ne peut se faire qu'avec les protocoles propriétaires.

Cependant, les nouveaux standards internationaux sont intégrés dans la Phase V de DNA aussitôt qu'ils deviennent disponibles.

---

[1] *DECnet DIGITAL Network Architecture (Phase V) - General Description*, op cit., page 12.

Chapitre 4

Le travail au CERN

Le problème et l'environnement de travail étant posés, nous pouvons exposer la solution qui a été implémentée au CERN. Nous expliquerons le principe des deux programmes réalisés.

4.1. Introduction

Le but du travail qu'on nous a demandé de réaliser au CERN est de repérer les connexions douteuses. Pour rappel, une connexion est un lien créé temporairement par une application pour un utilisateur afin que ce dernier puisse accéder à une machine sur laquelle il désire travailler. Lorsque son travail est terminé, l'utilisateur se déconnecte de la machine ; ce lien est alors détruit par l'application. Nous appellerons connexions douteuses les connexions qui ont **peut-être** été initialisées par des personnes qui essaient de se connecter sur des machines du CERN sans en avoir l'autorisation ou des personnes qui, à partir d'une machine du CERN, se connectent illégalement sur une ou plusieurs machines extérieures. Comme nous l'avons mentionné précédemment, les personnes qui se connectent sur des machines du CERN sans en avoir l'autorisation peuvent provoquer des dégâts considérables à l'intérieur du CERN alors que celles qui se connectent illégalement depuis une machine du CERN sur une machine extérieure peuvent nuire à la réputation de ce dernier.

Ce travail a comporté deux parties.

Dans un premier temps, il a été question de concevoir le programme de monitoring qui rassemble les informations pertinentes concernant les connexions observées sur les lignes DECnet.

Ensuite, s'est posé le problème du traitement des informations réunies par ce premier programme. En effet, ces dernières représentent en effet une importante quantité de données et traiter toutes ces informations constitue un travail considérable. Afin d'alléger la tâche du responsable de la sécurité, nous avons donc décidé de reporter une partie de ce

travail sur un programme de traitement. Ce programme analyse les données et n'en présente qu'une partie à l'utilisateur : celles qui pourraient indiquer une intrusion.

La figure n°4.1 montre l'enchaînement des programmes.

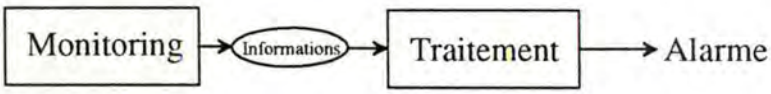

Figure n°4.1 : Enchaînement des programmes

Dans ce chapitre, nous allons présenter ces deux programmes.

4.2. Le programme de monitoring

Dans cette partie, nous allons étudier de plus près le programme de monitoring. Nous exposerons d'abord les spécifications du programme ; nous verrons ensuite comment le programme réalise ce qui lui est demandé. Dans un troisième temps, nous situerons plus précisément, sur le réseau du CERN, l'endroit où tourne le programme. Après avoir développé la méthode de travail qui a été suivie, nous présenterons une évaluation du programme au niveau des performances et une évaluation par l'utilisateur.

4.2.1. Spécifications : ce que fait le programme

Comme nous l'avons mentionné précédemment, le but du programme est de rassembler des informations au sujet de toutes les connexions en provenance ou à destination du CERN afin de permettre, ultérieurement, l'identification de connexions douteuses. Pour ce faire, le programme intercepte certains messages DECnet qui passent sur les lignes qui entrent et qui sortent du CERN. Les messages qui nous intéressent sont ceux d'ouverture de connexion, de confirmation d'ouverture de connexion, de données et de demande de déconnexion. En effet, c'est dans ces messages que nous pourrons trouver les informations intéressantes, c'est-à-dire celles qui nous seront utiles au moment de la recherche des connexions douteuses.

Sur base de ces messages, le programme enregistre certaines informations concernant les connexions surveillées, telles que l'adresse du noeud source et l'adresse du noeud de destination. De plus, il reconstitue, d'une part, les lignes de commande tapées par les utilisateurs qui se sont connectés à un système du CERN en utilisant le protocole CTERM, et, d'autre part, les réponses renvoyées par le système à ces utilisateurs. Il est nécessaire de reconstituer ces lignes de commande et ces réponses puisqu'il est possible qu'une ligne soit décomposée et transmise sur le réseau en plusieurs messages. Il convient donc de rassembler toutes les informations qui constituent une ligne avant l'analyse.

La recherche de mots clés

La chasse aux connexions douteuses est basée sur la recherche de mots clés dans les messages interceptés ou dans les lignes de commande reconstituées. Le programme de monitoring s'intéresse donc à la présence de certains mots dans les messages et les lignes de commande reconstituées. Des mots tels que "password" ou "user authorization failure" sont recherchés dans les messages et on enregistre le nombre d'occurrences de ces mots clés observées par connexion.

Les informations enregistrées

Pour chaque connexion, le programme enregistre, dans un fichier que nous appellerons "fichier des informations", les informations suivantes :

- l'adresse du noeud source, c'est-à-dire l'adresse du noeud qui a initialisé la connexion,
- l'adresse du noeud de destination, c'est-à-dire l'adresse du noeud qui reçoit la connexion,
- le sens de la connexion qui indique, si on le connaît, quel est le noeud qui a effectivement initialisé la connexion.
  Comme nous le verrons plus tard, le programme perd des messages ; il est donc possible qu'il n'intercepte pas le message d'ouverture d'une connexion. Dans ce cas, il est impossible de savoir lequel des deux noeuds a réellement initialisé la connexion ; l'adresse enregistrée comme étant celle du noeud source est alors l'adresse du noeud qui a envoyé le premier message intercepté, tandis que l'adresse enregistrée comme étant celle du noeud de destination est l'adresse du noeud qui a reçu ce message.
- un identifiant de la connexion au sein de la machine d'origine,
- un identifiant de la connexion au sein de la machine de destination.
  Ces deux identifiants nous permettent de distinguer deux connexions différentes entre deux mêmes machines.
- le nom des processus utilisateurs à la source et à la destination,
- le nombre de messages observés sur la connexion,
- la valeur des compteurs d'occurrences pour les différents mots clés,
- la raison de la déconnexion,
- l'heure de création de la connexion, ou, si le message de demande de connexion n'a pas été intercepté, l'heure à laquelle est passé le premier message observé sur la connexion,

- l'heure de déconnexion ou, si le message de demande de déconnexion n'a pas été intercepté, l'heure à laquelle est passé le dernier message observé sur la connexion,
- un identifiant de la connexion donné par le programme afin de distinguer une connexion de toutes les autres connexions.

Les lignes de commande suspectes observées sur une connexion, c'est-à-dire celles qui contiennent un des mots clés, sont enregistrées dans un autre fichier, le "fichier de commandes", avec l'identifiant de la connexion attribué par le programme.

Les fichiers

Les fichiers créés sont des fichiers binaires. En effet, étant donné le volume des informations à enregistrer, le choix s'est porté sur le type de fichiers qui occupait le moins d'espace disque possible.

Afin de permettre une plus grande flexibilité du programme, les mots clés à rechercher dans les messages et les lignes de commande reconstituées sont lus dans un fichier écrit par l'utilisateur. Ceci permet non seulement de combler des oublis dans le choix des mots à rechercher mais encore d'orienter la recherche en fonction de nouvelles découvertes d'activités suspectes et ce, sans modifier le code. Ainsi, si nous décidons de rechercher principalement les personnes qui utilisent abusivement "TELNET" ou "FTP" à partir du CERN, nous pouvons ajouter ces mots dans le fichier afin d'en tenir compte lors de la recherche. Cette modification peut facilement être réalisée par l'utilisateur.

Les options offertes

Il est possible d'ignorer les messages qui entrent via un routeur bien défini et ressortent immédiatement via un autre routeur bien défini. En effet, ces messages ne sont pas utiles pour notre travail, vu qu'ils transitent de façon transparente par le site. De nouveau, afin de permettre une plus grande flexibilité, les adresses de ces routeurs sont lues à partir d'un fichier écrit par l'utilisateur. Ceci permet d'ajouter des routeurs ou de modifier les adresses des routeurs sans changer le code.

Il est également possible de demander une recherche faisant la distinction entre majuscules et minuscules.

Enfin, il est possible d'ignorer, lors de la reconstitution des lignes de commande et de la recherche des mots clés, les connexions dont l'adresse source se situe à l'intérieur du site. Ceci permet de diminuer la quantité de messages à analyser. Cette option est utile puisque nous pouvons présumer que la plus grande menace vient des personnes extérieures qui essaient de se connecter sur des machines du CERN. Les personnes qui, depuis le CERN, se connectent ailleurs, mettent moins en péril la sécurité du site.

4.2.2. Comment travaille le programme ?

Nous étudierons ici les fichiers qui sont nécessaires à l'exécution du programme, de même que les fichiers créés par le programme lors de son exécution. Nous présenterons ensuite la structure globale du programme, avant d'analyser plus en profondeur certains principes de conception.

1. Les fichiers de données

Deux fichiers peuvent être donnés en entrée lors de l'exécution du programme. L'un d'entre eux, le fichier de mots clés, doit obligatoirement être fourni ; la présence obligatoire de l'autre dépend du choix des options : il s'agit du fichier contenant les adresses des routeurs.

a) Le fichier de mots clés

Le fichier de mots clés contient les mots clés à rechercher dans les messages et les lignes de commande reconstituées. Ce fichier est écrit par l'utilisateur avant de lancer l'exécution du programme de monitoring. Le choix des mots clés placés dans le fichier est fait par l'utilisateur, en fonction de ses besoins et de ses connaissances sur les activités des intrus. Ainsi, s'il recherche notamment les personnes qui essaient de modifier des fichiers auxquels elles n'ont pas le droit d'accéder, l'utilisateur peut faire figurer le mot "Insufficient privilege" dans le fichier.

Les contraintes

Nous avons formulé certaines contraintes au sujet du contenu de ce fichier afin d'en permettre l'utilisation par le programme de monitoring. Celles-ci ne devraient pas entraver la façon dont l'utilisateur veut travailler. Voici ces contraintes :

- Ce fichier ne peut contenir plus d'un certain nombre de mots, ce nombre étant défini comme une constante. Si le nombre de mots dans le fichier excède le nombre prévu, les mots excédentaires sont ignorés lors de la recherche. La constante a été définie de façon à tenir compte des besoins actuels et il serait assez simple de la changer au cas où cela s'avérerait nécessaire.
- Il y a un seul mot par ligne et les mots ont une longueur maximum, également définie comme une constante. Les mots trop longs sont tronqués. La longueur maximale est actuellement égale au nombre de caractères qu'il est possible d'écrire sur une ligne, soit quatre-vingts. Cela paraît raisonnable. De plus, toute

modification peut se faire sans difficulté vu que cette longueur est définie comme une constante.

- Les caractères blancs figurant après un mot sont ignorés lors de la recherche ; il n'en est pas de même pour les caractères blancs situés avant un mot. En effet, les caractères blancs se trouvant avant un mot sont aperçus par l'utilisateur au moment où il écrit son fichier ; nous pouvons donc penser qu'il les a placés intentionnellement. Par contre, les caractères blancs introduits après un mot ne sont pas visibles par l'utilisateur ; nous pouvons donc nous dire que ce dernier les a introduits par erreur. Les lignes ne contenant que des caractères blancs sont ignorées.
- Il est possible d'insérer des commentaires dans ce fichier.

Les avertissements

Les avertissements qui auraient pu survenir lors de la lecture de ce fichier, par exemple l'indication que le fichier contient trop de mots, sont enregistrés dans un fichier rassemblant la totalité des mises en garde destinées à l'utilisateur, afin que ce dernier puisse en prendre connaissance et éventuellement y remédier.

Le fichier de mots clés ne peut normalement pas être vide. Si tel est le cas, le programme s'arrête en affichant un avertissement à l'écran. Le but de ce programme est en effet de détecter des connexions douteuses en se basant sur la recherche de mots clés. L'absence de mots clés rend donc le programme sans intérêt.

Signalons encore qu'au moment de l'exécution du programme, le contenu de ce fichier sera mis en mémoire. Cela permet de diminuer le temps d'accès aux mots présents dans le fichier. Vu que le programme doit y accéder chaque fois qu'il recherche un des ces termes dans un message ou une ligne de commande reconstituée, cela améliore la vitesse d'exécution.

Un exemple

La figure n°4.2 donne un exemple de fichier contenant quelques mots clés.

```
! Ce fichier contient la liste des mots clés à rechercher

Password
User authorization failure
Insufficient privilege
Login incorrect
Telnet
Ftp
```

Figure n°4.2 : Exemple de fichier de mots clés

b) Le fichier d'adresses des routeurs

Un autre fichier de données contient les adresses Ethernet des routeurs, celles-ci étant écrites en hexadécimal. Ce fichier est également créé par l'utilisateur avant de lancer l'exécution du programme et permet au programme de monitoring d'ignorer, si l'utilisateur l'a demandé, les messages qui entrent via un routeur et ressortent immédiatement via un autre routeur.

Les contraintes

De nouveau, il existe certaines contraintes au sujet du contenu de ce fichier afin d'en permettre l'utilisation par le programme de monitoring. Celles-ci ne devraient pas entraver la façon dont l'utilisateur veut travailler. Voici ces contraintes :

- Il y a un nombre maximum d'adresses qui peuvent figurer dans le fichier. Ce nombre est défini comme une constante et peut facilement être modifié. Les adresses supplémentaires sont ignorées par le programme.
- Une adresse contient six bytes. Elle est donc représentée en hexadécimal par douze caractères compris entre 0 - 9 et a - f. Si une adresse contient plus de six bytes, les bytes excédentaires sont tronqués tandis que si une adresse contient moins de six bytes, l'adresse est complétée à droite avec des 0. Cela permet une certaine vérification des adresses introduites par l'utilisateur puisqu'en cas d'erreur, une mise en garde apparaît dans le fichier d'avertissements.
- Les bytes peuvent être séparés par un tiret, ce qui peut rendre plus facile l'écriture des adresses.
- Les caractères blancs avant ou après une adresse sont ignorés.

Les avertissements

De nouveau, les avertissements qui auraient pu survenir au cours de la lecture du fichier, par exemple la présence d'une adresse incomplète, sont rassemblés dans le fichier réunissant l'ensemble des mises en garde destinées à l'utilisateur. En prenant connaissance de ce fichier, l'utilisateur peut corriger ses erreurs.

Si le fichier d'adresses des routeurs est vide ou n'existe pas et que l'utilisateur a demandé d'ignorer les messages qui entrent via un routeur et ressortent immédiatement via un autre, le programme s'arrête en affichant à l'écran un avertissement. Nous pouvons en effet présumer que l'utilisateur a oublié de créer ou de remplir le fichier.

Au moment de l'exécution du programme, si l'utilisateur a demandé d'ignorer les messages transitant de façon transparente via le CERN, le contenu de ce fichier sera placé en mémoire. Cela permet de diminuer le temps d'accès aux adresses présentes dans le fichier.

Vu que le programme doit y accéder chaque fois que passe un message DECnet, cela améliore la vitesse d'exécution.

Un exemple

La figure n°4.3 donne un exemple de fichier contenant des adresses de routeurs.

```
! Ce fichier contient les adresses des routeurs

aa-00-04-00-0b-5a
aa-00-04-00-0c-5a
aa-00-04-00-f4-5d
```

Figure n°4.3 : Exemple de fichier d'adresses de routeurs

2. Les fichiers résultats

Trois fichiers résultats sont créés. Deux d'entre eux sont utilisés par le programme de traitement ; le troisième est destiné à l'utilisateur.

a) Le fichier des informations et le fichier de commandes

Le premier fichier créé contient les informations générales sur les connexions, le deuxième, les lignes de commande qui pourraient être suspectes. Comme nous l'avons déjà mentionné, ces fichiers sont des fichiers binaires. C'est le programme de traitement qui permet à l'utilisateur de visualiser le contenu de ces fichiers.

Un exemple

La figure n°4.4 donne un exemple de présentation des informations contenues dans les fichiers des informations et de commandes. La présentation utilisée est identique à celle du programme de traitement.

Les informations présentées dans cet exemple sont les suivantes :

- l'adresse du noeud source : 22,44 ;
- l'adresse du noeud de destination : 22,16 ;
- le sens de la connexion : ? indique que l'on ne connaît pas le noeud qui a effectivement initialisé la connexion ;
- le nom du processus utilisateur à la source : LPA ;
- le nom du processus utilisateur à la destination : CTERM ;

- la valeur des compteurs d'occurrences des différents mots clés qui nous apprend que trois mots clés, parmi les quatre qui étaient recherchés, ont été découverts ;
- l'heure de création de la connexion : 12 heures 1 minute et 23 secondes ;
- la durée de la connexion exprimée en secondes: 75 ;
- la raison de la déconnexion exprimée en hexadécimal : 9, la connexion a été détruite à la demande de l'utilisateur ;
- les trois lignes de commande suspectes observées sur cette connexion, précédées de l'heure à laquelle elles ont été observées et de l'origine de la ligne de commande. Deux lignes de commande proviennent d'une réponse envoyée par le système : elles sont précédées d'un 'S'. La troisième provient de l'utilisateur : elle est précédée d'un 'U'.

```
22,44   22,16   ? LPA  CTERM   1 1  0  1  12:01:23  75  9
12:01:23   S    Username:
12:01:30   S    Password:
12:02:05   U    Copy *.*
```

Figure n°4.4 : Présentation des renseignements tirés des fichiers des informations et de commandes

Chaque heure, de nouveaux fichiers sont créés. Ceci permet de limiter les pertes d'informations dues aux éventuels problèmes en cours d'exécution. Grâce à cette technique, seules les données de la dernière heure sont alors perdues.

Le nom des fichiers est construit par le programme lui-même. Le nom des fichiers est composé, entre autres, des date et heure de leur création.

Chaque jour à minuit, toutes les informations qui concernent des connexions ouvertes au cours de la journée et ne se retrouvent dans aucun des fichiers déjà créés, sont enregistrées dans le dernier fichier de la journée. Cela permet de rassembler les informations concernant toutes les connexions de la journée dans les fichiers adéquats.

b) Le fichier destiné à l'utilisateur

Le troisième fichier créé rassemble les avertissements survenus lors de la lecture des fichiers de données, comme, par exemple, l'indication que le cinquième mot du fichier de mots clés a dû être tronqué parce qu'il était trop long. Ce fichier est un fichier texte qui peut directement être lu par l'utilisateur.

Nous avons choisi de rassembler les avertissements dans un fichier plutôt que de les afficher à l'écran parce que le programme est destiné à être exécuté en tâche de fond. La

présence de l'utilisateur au moment de l'affichage d'éventuels avertissements n'est donc pas assurée. De plus, le programme pouvant s'exécuter malgré les problèmes qui peuvent survenir, il n'est pas nécessaire de l'interrompre.

Comme nous l'avons déjà signalé, les seuls cas dans lesquels nous avons décidé d'arrêter l'exécution du programme sont l'absence de mots clés à rechercher et, en fonction des options choisies, l'absence des adresses des routeurs. En effet, comme dans de telles circonstances, le programme ne dispose pas de toutes les informations qui lui sont nécessaires en entrée, il est préférable de l'interrompre.

3. Vue d'ensemble sur les fichiers

La figure n°4.5 donne une vision globale des fichiers de données et des fichiers résultats.

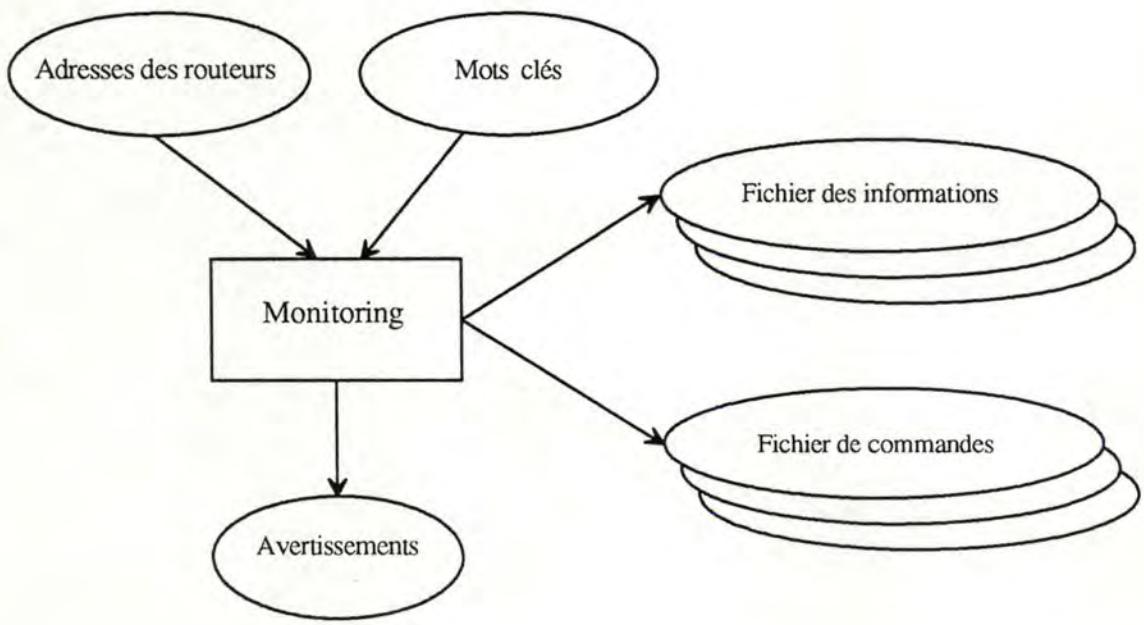

Figure n°4.5 : Vue globale des fichiers

4. Structure du programme

La figure n°4.6 donne une première présentation, simplifiée, de la structure du programme. Toutes les options offertes par le programme sont ignorées ici. Nous supposerons que la recherche des mots clés fait la distinction entre les majuscules et les minuscules et que l'on traite tous les messages. La structure du programme ne mentionne donc pas la

mise en majuscules des lignes reconstituées et des messages, ni les tests sur les adresses source et destination des messages. Nous expliquerons les principes de conception plus en détail dans le point suivant. Le code du programme se trouve en annexe.

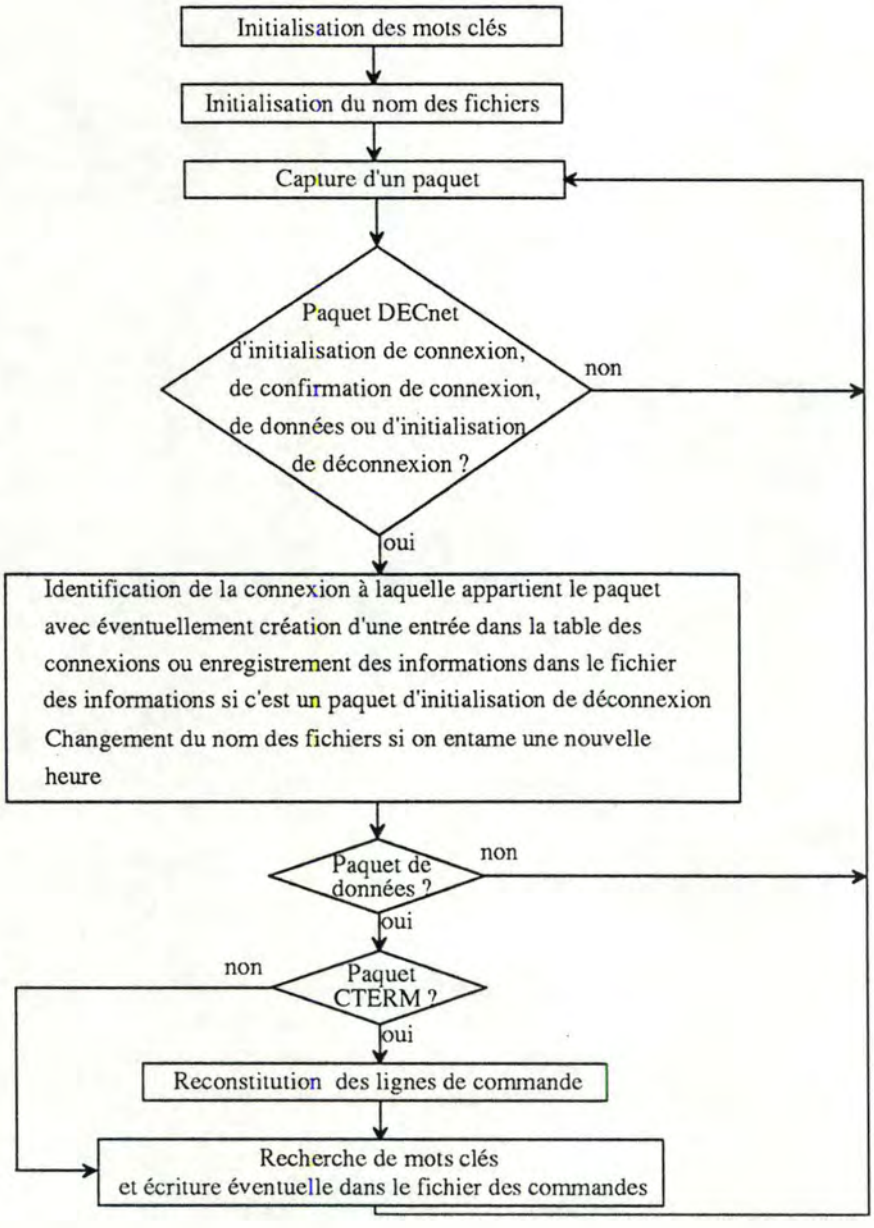

Figure n°4.6 : Structure du programme de monitoring

5. Certains principes de conception

Le programme de monitoring se compose d'un programme principal et de trois procédures importantes. Chacune de ces procédures appelle à son tour d'autres procédures pour effectuer des tâches spécifiques.

Le principe du programme

Voyons d'abord le principe du programme.

Le programme garde en mémoire une table des connexions en cours, ou, en tout cas, des connexions pour lesquelles il n'a pas vu le message de demande de déconnexion. Pour chacune de ces connexions, la table contient toutes les informations la concernant. Cette table peut gérer 30250 connexions. Bien que nous ne connaissions pas le nombre typique de connexions ouvertes à un moment donné, ce nombre nous paraît suffisamment élevé.

Chaque fois que le programme intercepte un message appartenant à une connexion qu'il ne connaît pas encore, c'est-à-dire pour laquelle il n'a pas encore d'entrée dans sa table, le programme crée une entrée. Les informations dans la table sont tenues à jour lors du passage d'un message et sont sauvées dans le fichier binaire contenant les informations au sujet des connexions au moment où passe le message de demande de déconnexion ou lorsque la table est remplie.

En effet, il est alors nécessaire de libérer une place dans la table pour la nouvelle connexion. Le choix de la connexion à retirer de la table, lorsque celle-ci est complète et qu'il est nécessaire de libérer une place, est fait de façon à minimiser la probabilité de retirer de la table des informations concernant une connexion qui n'est pas encore terminée. Le programme enlève de la table les informations relatives à une connexion sur laquelle il n'a plus observé de messages depuis le temps le plus long, ce qui peut indiquer que la connexion est terminée bien qu'il n'ait pas intercepté le message d'initialisation de la déconnexion.

Le programme, après avoir reconstitué une ligne de commande complète envoyée en utilisant le protocole CTERM, y cherche un des mots clés. Si un de ces mots y est présent, la ligne de commande est enregistrée dans le fichier de commandes ainsi que l'heure à laquelle cette ligne de commande a été interceptée sur le réseau. De même, si un message de données, envoyé par un protocole autre que CTERM, contient un mot clé, ce mot et l'heure de son passage sont enregistrés dans le fichier de commandes.

Afin de permettre la mise en relation des lignes de commande et des informations concernant les connexions, le programme attribue à chaque connexion un identifiant qui est repris dans les informations concernant la connexion et avec chaque ligne de commande observée sur cette connexion.

La table de hachage

Signalons encore l'utilisation d'une table de hachage afin d'accéder aux connexions présentes dans la table en mémoire. Sur base de l'adresse du noeud source et l'adresse du noeud de destination de la connexion qui sont présentes dans tous les messages, nous calculons un code de hachage correspondant à un indice dans la table de hachage. A cet endroit de la table de hachage, nous trouvons un pointeur vers une connexion dans la table des connexions. Toutes les connexions ayant le même code de hachage sont ensuite reliées entre elles. Ceci permet de retrouver une connexion sans devoir parcourir l'ensemble de la table des connexions. Nous ne devons parcourir que la liste de toutes les connexions ayant le même code de hachage, ce qui permet donc d'améliorer le temps d'accès à une connexion.

La figure n°4.7 illustre ce principe.

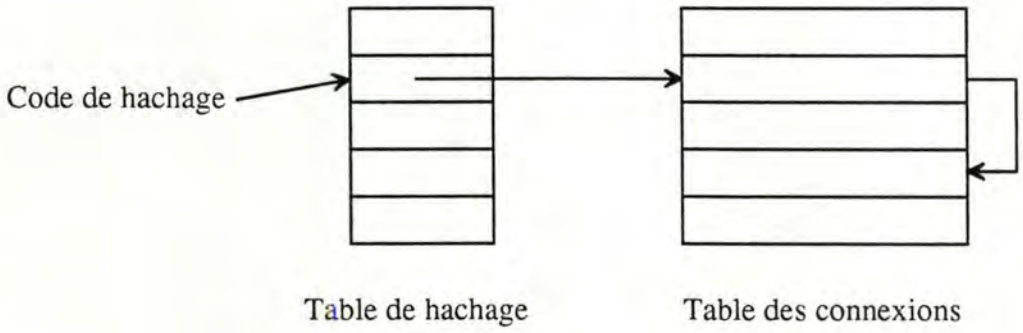

Figure n°4.7 : Principe d'une table de hachage

Voyons maintenant plus en détail le programme et les procédures.

Le programme principal

Le programme principal s'occupe de l'initialisation de la table de hachage de façon à ce qu'elle contienne des nombres qui ne correspondent pas à des indices de la table des connexions puisque cette dernière est encore vide. Il s'occupe également de l'initialisation de la structure en mémoire qui contient les mots clés et, éventuellement, de celle qui contient les adresses des routeurs ; il utilise pour cela les fichiers qui lui sont donnés. De plus, il initialise le nom des premiers fichiers qui seront créés et traite les arguments. Rappelons que les options proposées à l'utilisateur sont les suivantes : faut-il ou non faire la distinction entre les majuscules et les minuscules lors de la recherche des mots clés, faut-il ou non ignorer les messages qui ne font que transiter via le CERN et faut-il ou non, lors de la reconstitution des lignes de commande et la recherche des mots clés, tenir compte des connexions dont la source se situe à l'intérieur du site ? Si la recherche des mots clés ne

doit pas faire la distinction entre majuscules et minuscules, les mots clés sont convertis en majuscules dans la structure en mémoire. Lorsque cela est fait, la lecture et le traitement des messages peuvent commencer.

La sélection des messages

Lorsqu'un message est intercepté, une procédure détermine s'il est intéressant pour le traitement. Cette procédure de sélection teste donc le type des messages et ne retient que les messages de type DECnet qui sont des messages d'initialisation de connexion, de confirmation de connexion, de données ou de demande de déconnexion. Si l'utilisateur l'a demandé, cette procédure rejette aussi les messages qui passent entre deux routeurs.

Pour les messages retenus, la procédure de sélection appelle la procédure responsable de la gestion et de la mise à jour des informations dans la table des connexions en mémoire. Elle appellera ensuite, uniquement pour les messages de données et en ignorant éventuellement les messages passant sur une connexion dont l'adresse source se situe à l'intérieur du CERN, la procédure responsable de la reconstitution des lignes de commande et de la recherche des mots clés. Enfin, elle demandera la lecture d'un nouveau message.

La gestion et la mise à jour de la table

La procédure de gestion et de mise à jour de la table regarde si le message intercepté appartient à une connexion existante. Pour ce faire, elle fait appel à une procédure qui calcule le code de hachage à partir des adresses présentes dans le message et consulte la table de hachage afin de savoir s'il existe au moins une connexion dans la table en mémoire ayant ce code. Si tel est le cas, c'est-à-dire si la table de hachage renvoie un indice de la table des connexions, la procédure vérifie si les adresses de la connexion trouvée correspondent aux adresses présentes dans le message. Si c'est le cas, elle contrôle encore que les identifiants de la connexion au sein des machines à la source et à la destination sont identiques à ceux présents dans le message. Si les adresses ou les identifiants ne correspondent pas, la procédure examine la connexion suivante dans la chaîne et fait les mêmes tests. Si aucune connexion ne répond aux critères, la procédure informe la procédure de gestion et de mise à jour de la table que la connexion n'existe pas.

Si la connexion existe, la procédure de gestion de la table met à jour les informations concernant cette connexion, à savoir le nombre de messages observés sur la connexion et l'heure à laquelle est passé le dernier message appartenant à la connexion.

Si la connexion n'existe pas, la procédure crée, dans la table en mémoire, une entrée pour cette nouvelle connexion et initialise toutes les informations en fonction des données présentes dans le message ou de ce qu'elle peut en déduire. Ainsi, si le premier message observé contient un message d'initialisation de connexion, nous trouvons dans le message des renseignements tels que le nom des processus utilisateurs à la source et à la destination et nous pouvons également déterminer quel est le noeud qui a initialisé la connexion. Cependant, comme nous le verrons ultérieurement, le programme n'intercepte pas tous les

messages ; il lui arrive d'en rater. Il est donc possible que le premier message observé sur une connexion ne soit pas le message d'initialisation de la connexion. Si le premier message observé sur la connexion contient un message de confirmation de connexion, nous pouvons déterminer quel est le noeud qui a initialisé la connexion, mais nous ne connaissons pas les noms des processus utilisateurs. Enfin, si le premier message observé sur la connexion contient un message de données, il n'est pas possible de connaître le nom des processus utilisateurs à la source ou à la destination ; il est également impossible de savoir lequel des deux noeuds a initialisé la connexion.

Si le message intercepté contient un message de demande de déconnexion, les informations concernant la connexion sont sauvées sur le fichier des informations et une nouvelle place devient libre dans la table en mémoire. Notons également que si le premier message intercepté sur une connexion contient un message de demande de déconnexion, la connexion est ignorée puisque nous ne pouvons en retirer aucun renseignement utile.

Si la procédure doit créer une entrée dans la table en mémoire pour une nouvelle connexion, il est possible qu'elle doive faire appel à une autre procédure afin de libérer une place dans cette table. Cette procédure se base sur le moment où est passé le dernier message sur une connexion pour choisir la connexion à retirer. Ainsi, c'est la connexion sur laquelle on n'a plus vu passer de message depuis le moment le plus long qui est enregistrée sur le fichier et retirée de la table.

Remarquons que le retrait d'une connexion de la table en mémoire, que ce soit dû au passage du message de demande de déconnexion ou à la nécessité de libérer une place, peut modifier la table de hachage. En effet, il faut retirer la connexion de la chaîne des connexions ayant le même code de hachage et, éventuellement, supprimer le pointeur vers la chaîne si la connexion enlevée est la seule de la chaîne. De plus, la table des connexions est gérée de façon à ne jamais contenir d'espace libre entre deux connexions, ce qui permet de rassembler les places libres en fin de table. Il peut donc être nécessaire de déplacer des connexions au sein de la table. Etant donné cela, il est possible que des valeurs changent dans la table de hachage pour pointer vers d'autres endroits de la table des connexions. Des procédures gèrent ce problème.

Le changement de nom des fichiers

Chaque fois qu'elle crée une nouvelle entrée dans la table, la procédure appelle une procédure chargée de vérifier l'heure. Si la procédure constate que l'heure vient de changer, elle ferme les fichiers de l'heure précédente, change le nom des fichiers et ouvre les nouveaux fichiers. Si, en plus, il est minuit, avant de fermer le fichier des informations, cette procédure y enregistre toutes les données présentes dans la table des connexions. Cela permet de rassembler dans le fichier adéquat les informations concernant les connexions ouvertes au cours de la journée mais qui ne sont pas encore sauvées. Les données restent toutefois présentes dans la table des connexions afin de permettre la poursuite de leur traitement. Nous avons constaté que la création d'une nouvelle entrée dans la table est un événement assez fréquent. En effet, il se produit plusieurs fois par seconde. Nous avons donc

choisi de consulter l'horloge chaque fois qu'une nouvelle connexion apparaît plutôt que d'utiliser un chronomètre qui demande régulièrement au programme de consulter l'horloge. Et l'expérience nous a montré que les fichiers sont correctement créés au début de chaque heure.

La reconstitution des lignes et la recherche des mots clés

La procédure responsable de la reconstitution des lignes de commande et de la recherche des mots clés calcule d'abord le numéro de séquence du message intercepté. Si le message n'est pas un message retransmis, c'est-à-dire si le numéro de séquence est différent du numéro de séquence du message de données précédent, le message est analysé, sinon, il est rejeté. Si la recherche des mots clés ne fait pas la distinction entre les majuscules et les minuscules, le contenu du message est converti en majuscules.

Les messages CTERM sont transmis à la procédure de décodage du protocole CTERM qui rassemble les informations contenues dans tous les messages qui constituent une ligne de commande afin de reconstituer cette ligne.

Les lignes reconstituées et les messages qui n'utilisent pas le protocole CTERM sont fouillés pour y trouver un mot clé. Si un mot est découvert dans une ligne de commande, cette dernière est enregistrée dans le fichier de commandes avec l'heure de son passage et l'identifiant de la connexion sur laquelle elle a été observée. Si un mot clé est découvert dans un message, le mot clé, l'heure à laquelle il est passé et l'identifiant de la connexion à laquelle il appartient sont sauvés dans le fichier de commandes.

4.2.3. Où tourne le programme ?

Le programme écoute les lignes DECnet qui arrivent de l'extérieur du CERN et celles qui sortent du CERN. Les messages sont interceptés sur la dorsale du trafic externe.

La figure n°1.6 du chapitre 1 nous présente cette dorsale ainsi que les réseaux externe et interne du CERN. Sur cette figure, la machine sur laquelle tourne le programme est dénommée *monitoring*. Le programme se situe juste au-dessus de la couche Ethernet de cette machine. Chaque fois qu'elle reçoit une trame, la couche Ethernet la transmet au programme.

4.2.4. Méthode de travail

Pour réaliser ce programme, nous avons suivi une approche par prototypage.

En nous basant sur un programme écrit en FORTRAN qui intercepte les messages DECnet et les traite dans le but de recréer et suivre les sessions entre deux systèmes hôtes, c'est-à-dire écouter ce que fait un utilisateur, nous avons obtenu une première version du programme. Il a été nécessaire d'adapter cette version afin de cerner correctement le problème qui nous intéressait.

Au vu de cette première version qui se contentait de fouiller les messages sans reconstituer les lignes de commande, de nouvelles fonctionnalités ont été ajoutées, entre autres, la reconstitution des lignes de commande tapées par les utilisateurs connectés via le protocole CTERM et la reconstruction des réponses renvoyées par le système aux utilisateurs, afin de pouvoir connaître plus précisément l'activité de ces utilisateurs.

Les spécifications ont donc été modifiées au fur et à mesure que le programme se développait.

Quelques difficultés rencontrées

De plus, au cours du développement, il a été nécessaire de faire des retours en arrière. Il nous est ainsi arrivé de constater que le byte du message que nous regardions ne correspondait pas à ce que nous croyions ; il était donc nécessaire de modifier le code et d'examiner la valeur d'un autre byte. Mais, étant donné que nous n'avons trouvé, dans notre environnement de travail, aucune documentation complète décrivant bit par bit, ou byte par byte, le format des messages DECnet, nous n'étions pas surs que notre nouveau choix serait le bon. Ce n'était qu'au moment du test du programme que nous pouvions décider si la solution retenue était ou non correcte.

Nous avons également utilisé un analyseur de messages qui, bien que ne décodant pas tous les bytes de tous les messages, pour le protocole CTERM notamment, nous a permis de mieux comprendre le format des messages.

Le déchiffrement des messages a donc été la tâche la plus compliquée, particulièrement le décodage de ceux transmis en utilisant le protocole CTERM.

L'approche par prototypage

Remarquons que l'approche par prototypage est souvent utilisée. En effet, le client donne une première spécification, pas toujours très précise, de ce qu'il attend. Le développeur réalise alors une première version du programme qu'il propose au client. Au vu de cette première version, le client demande certains changements. Le développeur modifie

alors le programme selon les désirs du client et cela continue jusqu'au moment où le client est satisfait de ce qui lui est proposé.

4.2.5. Evaluation

Dans un premier temps, nous allons évaluer les performances du programme. Nous expliquerons alors certaines modifications qui ont été apportées au programme dans le but d'en améliorer les performances. Dans un second temps, nous présenterons une évaluation du programme par l'utilisateur.

1. Evaluation des performances

Actuellement, le programme perd des messages, ce qui empêche une reconstitution instructive des lignes de commande et des réponses du système. Ceci est dû à un manque de puissance de la machine par rapport au trafic qui passe sur le réseau.

Environ 600 messages passent par seconde sur la dorsale du trafic externe. Comme la configuration du réseau a récemment été changée, le trafic IP passe maintenant par un autre segment. La situation est donc meilleure. En effet, auparavant, on observait 1200 messages par seconde sur la dorsale.

Pourquoi perd-on des messages ?

La machine sur laquelle tourne le programme lit les messages en mode promiscuous, c'est-à-dire qu'elle intercepte tous les messages qui passent sur le réseau. Les messages qui sont lus sont placés dans un tampon où le programme vient les chercher pour les analyser. Lorsque le tampon est plein, la machine ne lit plus de message. Ainsi, si le programme ne peut pas traiter les messages suffisamment vite, le tampon se remplit et on perd des messages. Il faut donc que les messages soient analysés à un rythme moyen plus élevé que celui de leur passage.

Quelques améliorations proposées

Plusieurs solutions ont été envisagées, d'une part, pour diminuer le nombre de messages que devait traiter le programme et, d'autre part, pour diminuer le temps de traitement d'un message. En conséquence, plusieurs options ont été ajoutées au programme. Ainsi, comme nous l'avons mentionné précédemment, l'utilisateur peut choisir d'ignorer, pour la reconstitution des lignes de commande et la recherche des mots clés, les messages appartenant à des connexions initialisées à l'intérieur du CERN. Cela permet de diminuer le nombre de messages qui demandent un traitement complet. L'utilisateur peut également décider de ne pas traiter les messages qui entrent via un routeur et ressortent immédiatement via un autre routeur. De plus, une recherche faisant la distinction entre les majuscules et les

minuscules a également été offerte. Cela évite de convertir tous les messages en majuscules et diminue le temps de traitement d'un message.

L'efficacité de la table de hachage a aussi été étudiée. Ainsi, plusieurs codes de hachage ont été testés et le plus efficace a été retenu. Un code de hachage est efficace si les longueurs des chaînes créées sont plus ou moins identiques et si les chaînes ne sont pas trop longues. Ainsi, le parcours des chaînes ne prend pas trop de temps et la connexion est retrouvée plus vite. Nous avons retenu un code de hachage basé sur les adresses du noeud source et du noeud de destination. Ce code peut prendre des valeurs comprises entre 0 et 1023.

L'arrivée d'une machine plus puissante est aussi prévue pour les mois qui suivent.

2. Evaluation par l'utilisateur

Voici l'avis de l'utilisateur du programme.

Le programme de monitoring est utile, non seulement pour identifier les activités suspectes, mais également pour découvrir les utilisateurs qui se connectent au CERN en utilisant les lignes DECnet et utilisent ensuite TELNET, le protocole de terminalisation TCP/IP, pour se rendre ailleurs. Il se pourrait que ces personnes fassent un mauvais usage des ressources du CERN ; elles devraient se connecter directement via TELNET depuis leur institution sur la machine de destination.

Cependant, le problème d'identifier réellement les intrus subsiste toujours. Il arrive trop souvent que nous ne voyions pas les lignes de commande tapées par les utilisateurs. En effet, nous avons déjà signalé que le programme de monitoring perd des messages. Cela empêche souvent une reconstitution complète des lignes de commande tapées par l'utilisateur. Par conséquent, il est difficile de décider si une personne fait quelque chose qu'elle ne devrait pas.

Jusqu'à présent, le programme a permis de découvrir l'existence d'activités suspectes. Cependant, les informations rassemblées sont insuffisantes pour permettre la détection des intrusions. De plus, plusieurs intrusions se produisent chaque semaine. Bien que la plupart de ces intrusions ne présentent aucun danger, la quantité d'informations à analyser est importante. Il est donc difficile de traiter ces informations pendant le temps dont dispose le responsable de la sécurité.

4.3. Le programme de traitement

Dans cette partie, nous allons examiner plus en détail le programme qui traite les données rassemblées par le programme de monitoring et ne présente à l'utilisateur que celles qui pourraient indiquer une intrusion. Nous commencerons en essayant de définir plus précisément ce qu'est une connexion douteuse : quels sont les éléments qui nous permettent d'affirmer qu'une connexion est douteuse ? Ensuite, nous présenterons les spécifications du programme : en fonction de ce qui a été dégagé précédemment, quelle solution a été retenue ? Enfin, nous verrons comment travaille le programme pour réaliser l'objectif qui lui a été fixé.

4.3.1. Qu'est-ce qu'une connexion douteuse ?

Lorsque nous avons abordé le problème du traitement des données, il a été nécessaire de se demander ce qu'est une connexion douteuse. Nous avions déjà choisi de baser la recherche de ces connexions sur la présence de mots clés, mais presque chaque connexion contient une fois au moins le mot "password" ; la plupart de ces connexions ne sont pourtant pas des intrusions. Il a donc été nécessaire de trouver des critères qui nous permettent de déduire, à partir des informations rassemblées, qu'une connexion est douteuse.

En nous plongeant dans la littérature et dans la lecture de faits réels[1], nous avons pu rassembler quelques renseignements.

Les connexions contenant certains mots

Une connexion sur laquelle l'utilisateur se voit de nombreuses fois refuser l'accès à la machine suite à son identification, peut indiquer qu'il s'agit d'un intrus qui tente de s'introduire sur cette machine en essayant plusieurs noms d'utilisateur et plusieurs mots de passe.

Une connexion avec plusieurs violations de protection peut aussi être considérée comme douteuse. Il peut s'agir d'un intrus qui, après s'être introduit sur une machine, essaie de copier des fichiers auxquels il n'a pas le droit d'accéder ou essaie de modifier des fichiers dans le but de se donner des privilèges qui ne lui avaient pas été accordés. Une telle connexion peut aussi révéler un utilisateur légitime qui essaie de faire des choses auxquelles il n'est pas autorisé.

---

[1] Dorothy E. DENNING, *Intrusion-Detection Model*, dans IEEE Transactions on Software Engineering, Vol. 13, n°2, Février 1987, pages 222-232.

Alain BROSSARD, (brossard@sic.epfl.ch), *Discussion d'une attaque*, Usenet news, Forum epfl.silicon/1, 22 Octobre 1992.

Les connexions inhabituelles

Les connexions inhabituelles sont aussi douteuses. Toutes les choses qui montrent un comportement étrange de l'utilisateur par rapport à son comportement habituel peuvent révéler une tentative d'intrusion. Parmi toutes ces choses, nous pouvons citer, par exemple, des temps de connexion plus longs que d'habitude, des heures de connexion inhabituelles, des activités inhabituelles sur la connexion,...

Le moment qui s'écoule entre deux connexions ayant les mêmes noms d'utilisateur est également important. Si le temps écoulé depuis la dernière connexion est très long, il se pourrait que la personne qui s'est connectée soit un intrus utilisant un compte qui n'est plus utilisé.

Une connexion entre deux noeuds qui n'avaient jamais été connectés auparavant peut aussi paraître douteuse.

Plusieurs connexions ayant toutes la même adresse d'origine mais éventuellement avec des adresses de destination différentes situées dans le même site peuvent être considérées comme douteuses, principalement si ces connexions sont de brève durée et si elles contiennent de nombreuses fois le mot "password". Elles pourraient révéler une tentative d'intrusion d'un individu qui, à partir d'une machine, essaie de s'introduire sur une machine ou un compte d'un autre endroit. Tant qu'il échoue, il essaie de s'introduire sur une autre machine ou un autre compte, mais toujours dans le même site. Donc, ces connexions sont de courte durée et échouent. Toutes ces connexions apparaissent dans un laps de temps assez court.

4.3.2. Spécifications : ce que nous avons décidé d'implémenter

Dans la solution retenue pour l'implémentation, l'utilisateur exprime lui-même les règles qu'il désire appliquer pour décider si une connexion est douteuse. Ainsi, dans le fichier contenant les mots clés, l'utilisateur précise, en plus des mots clés à rechercher, les règles à appliquer lors de l'affichage des informations. Cela permet à l'utilisateur de choisir lui-même sa définition de connexion douteuse, cette définition pouvant varier en fonction des événements et de l'utilisateur. Ce choix n'est pas fixé une fois pour toutes par le programme.

Les règles à appliquer

Les règles permettent de préciser le nombre de fois qu'il faut avoir observé un mot clé sur une connexion pour la considérer comme douteuse et afficher les informations la concernant. Il est également possible de spécifier l'intervalle de temps maximum qui doit séparer les différentes occurrences du mot clé, mais cette information est optionnelle.

Ainsi, un utilisateur peut décider que, si le mot "password" est passé 3 fois sur une connexion en moins de 30 secondes, la connexion est considérée douteuse et qu'il désire prendre connaissance des informations relatives à cette connexion.

Etant donné que les règles portent sur le nombre d'occurrences des différents mots clés, nous avons décidé de placer les règles dans le même fichier que les mots clés.

Le programme de traitement reprend donc le fichier des informations et le fichier de commandes créés par le programme de monitoring, ainsi que le fichier créé par l'utilisateur contenant les mots clés et les règles. Il décide, en fonction des règles, si les informations concernant une connexion doivent ou non être affichées à l'écran et, dans l'affirmative, affiche ces informations ainsi que les lignes de commande observées sur cette connexion.

Les options offertes

Plusieurs options sont offertes par le programme.

Il est possible que le programme affiche les informations concernant toutes les connexions présentes dans le fichier, indépendamment des règles exprimées par l'utilisateur. De plus, si aucune règle n'est énoncée par l'utilisateur, le programme propose à l'écran les informations concernant toutes les connexions. Cela peut être utile pour diverses raisons, notamment des raisons de statistique.

Par défaut, le programme arrête l'affichage des informations après chaque écran ; l'utilisateur doit alors presser une touche pour poursuivre la présentation des données. Il est cependant possible que le programme laisse défiler les informations sans interruption. C'est utile si l'utilisateur veut rediriger le résultat du programme vers un fichier. Il est donc possible de mettre l'ensemble des informations concernant les connexions dans un fichier texte si cela s'avère nécessaire.

4.3.3. Comment travaille le programme ?

Nous étudierons ici les fichiers de données qui doivent être fournis au moment de l'exécution du programme. Nous présenterons ensuite la structure du programme, avant d'analyser plus en profondeur certains principes de conception.

1. Les fichiers de données

Le programme de traitement reprend en entrée les fichiers binaires créés par le programme de monitoring, à savoir le fichier des informations et le fichier de commandes.

Il doit également disposer du fichier contenant les mots clés et les règles. Les restrictions liées au contenu du fichier de mots clés et mentionnées dans le cadre du programme de monitoring restent valables ; nous allons maintenant nous pencher sur les contraintes relatives à la formulation des règles.

Les contraintes relatives à la formulation des règles

N'importe quelle ligne du fichier peut contenir une règle. Si une ligne contient une règle et non un mot clé, elle doit débuter par les caractères "R ". Il est en effet indispensable que le programme puisse faire la différence entre les mots clés et les règles. De plus, étant donné qu'il est possible qu'un mot commence par un "R", un espace doit être présent entre le "R" et le début de la règle.

Chaque règle peut contenir un et un seul "AND" (et logique) ou un et un seul "OR" (ou non exclusif). Si une règle contient plusieurs "AND", plusieurs "OR" ou une combinaison de "AND" et de "OR", un avertissement apparaît à l'écran et le programme s'arrête après la lecture de cette règle.

Les règles simples sont exprimées de la façon suivante :

R mot-clé >= nombre-d'occurrences [(intervalle-de-temps)]

Ceci signifie que si le nombre d'occurrences de "mot-clé" observées sur une connexion est plus grand que "nombre-d'occurrences" et que le laps de temps écoulé entre le passage de la première occurrence et le passage de l'occurrence numéro "nombre-d'occurrences" est plus petit que "intervalle-de-temps", les informations concernant la connexion seront affichées. Le paramètre "intervalle-de-temps" est exprimé en secondes. De plus, il est facultatif. S'il n'est pas mentionné, le programme ne tiendra compte que du nombre d'occurrences du mot clé.

Nous pouvons également avoir :

- R mot-clé1 >= nombre-d'occurrences1 [(intervalle-de-temps1)] AND mot-clé2 >= nombre-d'occurrences2 [(intervalle-de-temps2)]
- R mot-clé1 >= nombre-d'occurrences1 [(intervalle-de-temps1)] OR mot-clé2 >= nombre-d'occurrences2 [(intervalle-de-temps2)]

Si le nombre d'occurrences minimal signalé dans la règle est plus petit ou égal à zéro et/ou si l'intervalle de temps est plus petit que zéro, une erreur est signalée à l'utilisateur et le programme s'arrête après la lecture de cette règle. De même, si le signe ">=" n'est pas présent dans l'une des règles, le programme affiche un avertissement à l'écran et s'arrête après la lecture de cette règle.

Un mot clé ne peut apparaître que dans une seule règle. Si ce n'est pas le cas, le programme s'arrête dès qu'il a lu la règle violant la contrainte et indique une erreur à l'utilisateur. D'une manière analogue, si un mot est orthographié différemment dans une règle que dans la liste des mots clés ou si un mot présent dans une règle ne figure pas dans la liste des mots clés, le programme s'arrête après avoir lu la règle et donne une indication à l'utilisateur. Toutefois, la correspondance n'est pas sensible au fait que les mots soient écrits en majuscules ou en minuscules.

Si une règle est trop longue, c'est-à-dire si elle s'étend sur plus d'une ligne, le programme s'arrête immédiatement en le mentionnant à l'utilisateur. Toutefois, la longueur maximale d'une règle, tout comme la longueur maximale d'un mot clé, est définie comme une constante. Il est donc assez aisé de modifier la valeur de cette constante et d'admettre des règles plus longues si cela s'avère utile.

Les avertissements

Lorsqu'une contrainte n'est pas respectée, nous avons choisi d'arrêter le programme immédiatement avec une indication sur le type d'erreur plutôt que de rassembler la totalité des avertissements dans un fichier. En effet, ce programme, contrairement au programme de monitoring, est destiné à être exécuté de façon interactive. L'utilisateur peut donc corriger immédiatement les erreurs. Permettre à l'utilisateur de prendre tout de suite connaissance des problèmes et lui donner la possibilité de les corriger sans délai nous a paru être la meilleure solution.

Si aucun mot clé n'est présent dans le fichier, le programme s'arrête. Cela se justifie par le fait que la chasse aux connexions douteuses se base sur la recherche de mots clés. Par contre, comme nous l'avons déjà signalé, si aucune règle n'est exprimée dans le fichier, le programme affiche les informations concernant toutes les connexions.

Un exemple

La figure n°4.8 donne un exemple de contenu du fichier contenant les mots clés et les règles.

```
! Ce fichier contient la liste des mots clés à rechercher
! et des règles à appliquer au moment de l'affichage des
! informations

Password

R password >= 3 (30)

Insufficient privilege
Login incorrect

R login incorrect >= 5 AND insufficient privilege >= 6 (50)

User authorization failure
Ftp

R ftp >= 5 OR user authorization failure >=10
```

Figure n°4.8 : Exemple de fichier contenant les mots clés et les règles

Remarquons qu'il est important que le programme de traitement soit exécuté avec un fichier qui contient les mêmes mots clés que le fichier avec lequel le programme de monitoring a été exécuté et que ces mots apparaissent dans le même ordre. Si ce n'est pas le cas, les résultats risquent d'être faussés. Il est impossible pour le programme de traitement de tester l'identité des mots clés puisque le fichier des informations ne contient aucune trace des mots clés utilisés pour la recherche ; il ne comprend que la valeur des compteurs d'occurrences pour les différents mots clés, dans le même ordre que celui dans lequel les mot clés figurent dans le fichier. Il n'est donc pas possible de comparer les mots clés utilisés par le programme de monotoring et les mots clés présents dans le fichier employé par le programme de traitement.

Une vue d'ensemble

La figure n°4.9 donne une vue d'ensemble des fichiers utilisés par le programme de traitement.

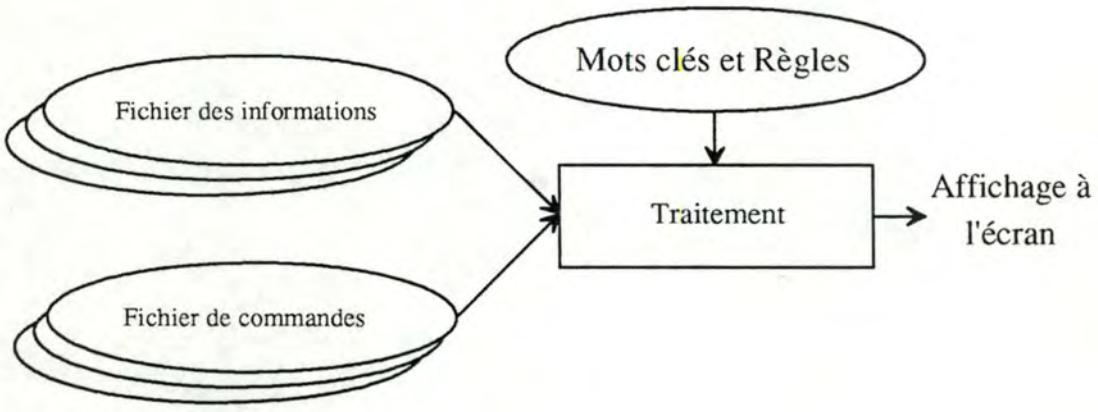

Figure n°4.9 : Vue globale des fichiers

Remarquons que le programme de traitement affiche les données à l'écran ; aucun fichier de résultats n'est donc créé.

2. La structure du programme

La figure n°4.10 présente une vision simplifiée de la structure du programme. Nous ignorons ici le traitement des paramètres et nous supposons que le programme affiche les informations en fonction des règles et qu'il ne se soucie pas de la présentation écran par écran. Nous expliquerons les principes de conception plus en détail dans le point suivant. Le code du programme se trouve en annexe.

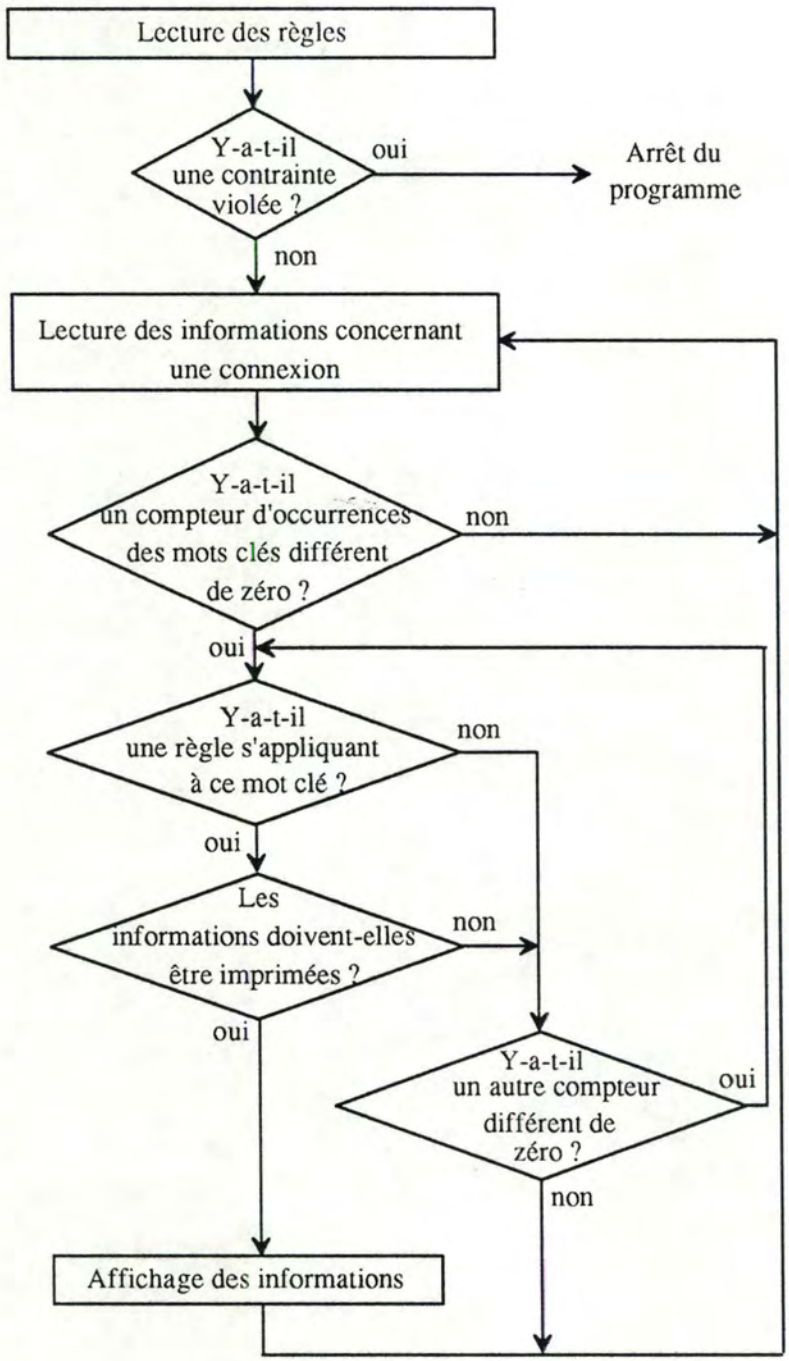

Figure n°4.10 : Structure du programme de traitement

3. Certains principes de conception

Nous allons d'abord voir le principe du programme. Ensuite, nous étudierons plus en détail la représentation des règles en mémoire. C'est en effet cette représentation qui constitue la partie la plus intéressante du programme.

Le principe du programme

Le programme de traitement lit les fichiers binaires créés par le programme de monitoring et affiche à l'écran les informations contenues dans ces fichiers tout en respectant les règles exprimées par l'utilisateur dans le fichier des mots clés. Pour réaliser cela, le programme doit faire le lien entre les informations contenues dans les deux fichiers, à savoir le fichier des informations et le fichier de commandes. Ce lien est possible grâce à l'identifiant des connexions qui se retrouve dans les deux fichiers.

Pour chaque connexion considérée comme douteuse, en fonction des règles exprimées par l'utilisateur, le programme doit afficher les informations relatives à cette connexion ainsi que les lignes de commande et les réponses du système observées sur cette dernière. Cela se fait de la façon suivante :

Après avoir pris connaissance des règles à appliquer, le programme lit les informations concernant une connexion, décide, en fonction des règles, s'il faut afficher ces informations et, dans l'affirmative, parcourt l'ensemble des lignes de commande du fichier de commandes afin de retrouver celles qui ont été observées sur cette connexion. Il affiche ensuite les informations et les lignes de commande de cette connexion puis lit les informations concernant la connexion suivante. Tant que le programme n'est pas arrivé à la fin du fichier des informations, il réitère ces opérations.

Afin d'accélérer le parcours du fichier de commandes, le contenu de ce fichier est placé en mémoire centrale. Ce fichier est en effet parcouru entièrement chaque fois que l'on décide que les informations concernant une connexion doivent être affichées.

Au moins un paramètre doit être donné au programme de traitement ; il s'agit du nom du fichier des informations que l'on veut traiter. A partir du nom de ce fichier, le programme construit lui-même le nom du fichier de commandes associé.

La représentation des règles en mémoire

Voyons maintenant comment le programme prend connaissance des règles à appliquer et les représente en mémoire.

Les règles sont représentées en mémoire sous la forme d'un tableau composé de trois colonnes. Chaque ligne du tableau correspond à un mot clé. Ainsi, la ligne d'indice i dans la structure contenant les règles se rapporte au mot clé qui occupe la case d'indice i dans la structure contenant les mots clés. Avant d'initialiser la structure contenant les règles, il est donc nécessaire d'initialiser la structure contenant les mots clés à partir du contenu du fichier.

La première colonne du tableau contient le nombre minimum de fois que le mot clé doit être apparu sur la connexion, la deuxième colonne du tableau indique l'intervalle de temps maximal entre les occurrences et la dernière colonne indique si la règle relative à ce mot contient un "AND". Une règle contenant un "AND" ou un "OR" peut être décomposée en deux règles distinctes. En effet, une règle contenant un "AND" peut être exprimée comme deux règles qui doivent être satisfaites en même temps ; après avoir vérifié que la première partie de la règle est vérifiée, il est nécessaire de s'assurer que la deuxième partie de la règle l'est également. En revanche, une règle contenant un "OR" peut être exprimée comme deux règles indépendantes ; dès que l'une est vérifiée, l'information peut être affichée. La troisième colonne du tableau indique donc, dans le cas d'une règle contenant un "AND", l'indice de la ligne contenant l'autre partie de la règle ; cette deuxième partie de la règle doit également être vérifiée par le programme. Dans les autres cas, cette colonne contient une valeur ne correspondant pas à un indice valide de ligne.

La figure n°4.11 illustre la manière de représenter les règles en mémoire.

```
Soit le fichier de mots clés et de règles suivant :

Password
User authorization failure
Insufficient privilege
Login incorrect
Telnet
Ftp

R password >= 3 (30)
R ftp >= 5 OR user authorization failure >=10
R login incorrect >= 5 AND insufficient privilege >= 6 (50)

Ces règles sont représentées par :
```

password
user authorization failure
insufficient privilege
login incorrect
telnet
ftp

Tables des mots clés

3	30	0
10	0	0
6	50	4
5	0	3
0	0	0
5	0	0

Table des règles

Figure n°4.11 : Représentation des règles en mémoire

Cette structure est remplie au fur et à mesure que le programme lit le fichier. Comme les compteurs d'occurrences de mots clés se présentent dans le même ordre que les mots clés dans le fichier, il suffit de regarder quels compteurs ont une valeur différente de zéro pour déterminer les mots clés qui ont été observés sur une connexion et donc les indices des règles à vérifier.

Chapitre 5

Généralisations possibles

Beaucoup de modifications peuvent être apportées à la solution présentée dans le chapitre précédent afin de la généraliser. Nous allons en mentionner quelques-unes. Nous examinerons d'abord les modifications qui permettraient de considérer des protocoles autres que DECnet Phase IV et des réseaux autres qu'Ethernet. Nous envisagerons ensuite le problème de la détection des intrusions et de l'alarme.

5.1. Les protocoles et les réseaux

Nous avons vu, dans le deuxième chapitre, qu'une des qualités d'un logiciel de détection d'intrusions est sa capacité à être utilisé dans un environnement différent de celui dans lequel il a été conçu, ce avec un minimum d'adaptation. La solution présentée n'offre pas cette qualité. En effet, comme nous l'avons vu, le programme qui rassemble les informations utiles à la détection des intrusions est destiné à analyser les messages échangés sur un réseau Ethernet via le protocole DECnet Phase IV.

Il serait donc intéressant de généraliser ce programme de façon à ce qu'il puisse être utilisé avec n'importe quel protocole sur n'importe quel réseau, cela sans imposer d'importantes modifications au code du programme.

5.1.1. Les protocoles

Afin de pouvoir considérer différents protocoles, le programme devrait connaître le format des différents messages échangés selon ce protocole.

Le format de ces messages pourrait être décrit dans un fichier texte. Ce fichier comprendrait, pour chaque protocole examiné, une liste des messages utiles pour le programme ainsi qu'une description du format de ces messages.

Dans chaque message, un ou plusieurs bytes indiquent le protocole utilisé. Ainsi, pour DECnet Phase IV, le 13ème byte d'une trame Ethernet a toujours la valeur hexadécimale 60 et le 14ème byte, la valeur 03. Grâce à ces bytes, le programme pourrait identifier le protocole qu'il doit considérer. En effet, il suffirait d'examiner le fichier de description de façon séquentielle jusqu'au moment où les valeurs des bytes identifiant le protocole correspondent à celles présentes dans la trame.

Connaissant le protocole, le programme pourrait déterminer, en consultant le fichier, le type du message qu'il vient d'intercepter. En examinant la description du format de ce message, le programme pourrait en extraire les informations utiles.

Cette solution nécessiterait la mise au point d'une syntaxe permettant de décrire un protocole par un fichier texte. Il faudrait définir de façon très précise où se trouvent, dans les trames Ethernet, les informations utiles pour le programme. Plusieurs solutions sont possibles ; nous allons en proposer une à travers un exemple.

La figure n°5.1 donne un exemple d'une partie d'un fichier décrivant des protocoles. Cette partie du fichier décrit quelques-unes des informations nécessaires au traitement des messages transmis par DECnet Phase IV sur un réseau Ethernet.

La première ligne indique que la description des messages de DECnet Phase IV commence. Les deux lignes suivantes donnent la place des bytes qui permettent d'identifier le protocole qu'utilise le message ainsi que leur valeur. Dans les messages DECnet Phase IV, les 13ème et 14ème bytes ont respectivement les valeurs hexadécimales 60 et 03. La quatrième ligne spécifie que le programme ne doit traiter que les trames Ethernet dont le 39ème byte a une des valeurs mentionnées. Ensuite, la réelle description des messages commence. Ainsi, l'adresse de destination se trouve dans les 25ème et 26ème bytes et l'adresse source, dans les 33ème et 34ème bytes. La dernière ligne indique que, s'il s'agit d'un message dont le 24ème byte a la valeur 38, l'information utile se trouve dans le 43ème byte. Il s'agit de la raison de la déconnexion.

```
[DECnet IV]
13 = 60
14 = 03
OK -> 39 = 60,38,28,18,00,20,40
25, 26 = dest
33, 34 = src
if 39 = 38 then 43 = disconnect reason
...
```

Figure n°5.1 : Exemple d'un fichier de description de formats de messages

Toutefois, il faut remarquer que le nombre de messages que doit traiter le programme est proportionnel au nombre de protocoles qu'il doit considérer. Nous avons déjà vu que notre programme, qui considère uniquement DECnet Phase IV, est incapable de traiter tous les messages. Si le nombre de protocoles à envisager est plus élevé, le risque de

manquer des messages est également plus élevé. Les informations rassemblées pourraient donc être moins précises. Il y a donc un compromis à trouver entre efficacité et généralité.

5.1.2. Les réseaux

Afin de pouvoir être utilisé sur différents réseaux, le programme doit connaître le format des différentes trames qui circulent sur les différents types de réseaux.

Les trames ne contiennent pas de byte indiquant sur quel type de réseau elles circulent. La solution proposée pour les protocoles n'est donc plus applicable. Il faudrait avertir le programme, au début de son exécution, du type de réseau qui est utilisé, ce type restant inchangé jusqu'au moment où le programme est arrêté.

Cette solution ne présente pas d'inconvénient puisque le programme, une fois lancé, tourne toujours sur la même machine et écoute toujours le même réseau. Le type de réseau écouté ne change donc pas au cours d'une exécution.

Il serait donc possible de décrire le format des trames en fonction du type de réseau dans un fichier. Un paramètre donné au programme lors de son lancement permettrait au programme de prendre connaissance, une fois pour toutes, du format des trames qu'il doit considérer.

5.2. La détection des intrusions et l'alarme

Dans la solution présentée, le responsable de la sécurité joue encore une rôle important dans la détection des intrusions. En effet, le programme présente à l'écran les informations concernant les connexions douteuses. Le responsable de la sécurité doit alors décider si la connexion représente un danger et prendre les mesures qui s'imposent. Il serait intéressant de perfectionner le programme de façon à diminuer l'intervention du responsable de la sécurité.

5.2.1. La détection des intrusions

Ayant pris connaissance de l'existence d'une connexion qu'il juge dangereuse, le responsable de la sécurité utilisera certainement un analyseur de réseau lui permettant d'apprendre plus précisément ce qui se passe.

On pourrait toutefois concevoir que le programme détecte lui-même qu'il y a un risque d'intrusion et se transforme en analyseur de réseau.

Le programme devrait à cet effet disposer de règles qui lui permettent de décider qu'une connexion présente un réel danger, tout comme le fait un système expert. Par exemple, une règle pourrait préciser que toute connexion initialisée depuis un certain endroit et

sur laquelle on observe beaucoup de mots de passe erronés doit être surveillée plus attentivement. Le programme se concentre alors particulièrement sur la connexion qui présente un risque, enregistrant par exemple toutes les opérations qui sont réalisées lors de cette session. En fonction de cette surveillance plus rapprochée, il pourrait alors déterminer si la connexion est ou non une intrusion.

5.2.2. L'alarme

Ayant découvert une intrusion, le système pourrait réagir de différentes façons.

D'une part, on pourrait imaginer que le système, lorsqu'il a détecté une intrusion, soit capable de réagir seul, par exemple en coupant la connexion.

D'autre part, le système pourrait avertir le responsable de la sécurité qu'une intrusion vient d'avoir lieu, par exemple en lui envoyant un message via le courrier électronique.

En conclusion, le programme que nous avons réalisé constitue un premier pas vers un véritable système de détection d'intrusions. Toutefois, de nombreuses améliorations peuvent y être apportées.

Conclusions

Les problèmes relatifs à la détection des intrusions ne sont pas simples. Nous avons vu que l'audit semble être une solution attrayante, surtout si le responsable de la sécurité dispose d'un outil automatisé qui lui permet de réaliser une analyse assez complexe des données d'audit.

Il existe plusieurs techniques de détection d'intrusions basées sur l'audit. Chacune présente des points forts et des points faibles. Il est donc intéressant de disposer d'un outil qui combine plusieurs de ces techniques, conservant ainsi les points forts de chacune et éliminant leurs points faibles.

La solution retenue au CERN implique deux programmes.

Le premier programme réunit un ensemble de données considérées comme pertinentes pour la détection des intrusions.

Le deuxième programme analyse les données recueillies par le premier. Il s'agit d'un système basé sur des règles dont le rôle est de diminuer la quantité d'informations que doit considérer le responsable de la sécurité. En effet, le responsable exprime les règles qui permettent de déterminer, parmi toutes les informations rassemblées, celles qui doivent être affichées.

Plusieurs améliorations pourraient être apportées au système. Ainsi, il serait intéressant de le rendre utilisable sur plusieurs types de réseaux, avec plusieurs protocoles différents. De même, le système pourrait détecter en temps réel les intrusions et en informer le responsable de la sécurité, par exemple via le courrier électronique.

La solution que nous avons implémentée est donc un premier pas vers l'établissement d'un véritable système de détection d'intrusions.

Bibliographie

[BROSSARD, 92] Alain BROSSARD, (brossard@sic.epfl.ch), *Discussion d'une attaque*, Usenet news, Forum epfl.silicon/1, 22 Octobre 1992.

[DENNING, 87] Dorothy E. DENNING, *An Intrusion-Detection Model*, IEEE Transactions on Software Engineering, Vol. 13, n°2, Février 1987, pages 222-232.

[DIGITAL, 82] *DECnet DIGITAL Network Architecture (Phase IV) - General Description*, Digital Equipment Corporation, Maynard Massachusetts, Mai 1982.

[DIGITAL, 87] *DECnet DIGITAL Network Architecture (Phase V) - General Description*, Digital Equipment Corporation, Maynard Massachusetts, Septembre 1987.

[GUEDIRA, 93] Abdelmounaim GUEDIRA, *Analyse des fichiers de sécurité sur réseau de stations SUN*, mémoire de l'Institut d'Informatique, 1993.

[HABRA, 92] N. HABRA, B. LE CHARLIER, I. MATHIEU et A. MOUNJI, *ASAX : Software Architecture and Rule-Based Language for Universal Audit Trail Analysis*, Proceedings of the Second European Symposium on Research in Computer Security (ESORICS), Toulouse, France, Novembre 1992, pages 435-450.

[HABRA, 94] N. HABRA, B. LE CHARLIER, A. MOUNJI et D. ZAMPUNIERIS, *Prototype Distributed System for On-line Network Security Monitoring*, Juillet 1994.

[HANSEN, 92] S. E. HANSEN et E. Todd ATKINS, *Centralized System Monitoring With Swatch*, Stanford University, Juillet 1992.

[HARPER, 93] John HARPER, *Overview of Digital's Open Networking*, Digital Technical Journal, Vol. 5, N°1, Hiver 1993, pages 12-20.

[HAULOTTE, 93] Vincent HAULOTTE, *Etude des virus informatiques et utilisation d'un langage d'analyse d'audit-trail pour leur détection*, mémoire de l'Institut d'Informatique, 1993.

[HOCHBERG, 93] J. HOCHBERG, K. JACKSON, C. STALLINGS, J.F. McCLARY, D. DUBOIS et J. FORD, *NADIR : An Automated System for Detecting Network Intrusion and Misuse*, Computers & Security, Vol. 12, n°3, 1993, pages 235-248.

[LIBION, 91] Fabian LIBION, *Towards "Intelligent" Security Audit Trail Analysis Tools*, mémoire de l'Institut d'Informatique, 1991.

[LUNT, 93] Teresa F. LUNT, *A survey of intrusion detection techniques*, Computers & Security, Vol. 12, n°4, 1993, pages 405-418.

[LUNT, 88] Teresa F. LUNT et R. JAGANNATHAN, *A Prototype Real-Time Intrusion-Detection Expert System*, Proceedings of 1988 IEEE Symposium on Security and Privacy, 1988, pages 59-66.

[MALAMUD, 91] Carl MALAMUD, *Analysing DECnet/OSI Phase V*, Van Nostrand Reinhold, 1991.

[MARSHALL, 91] Victor H. MARSHALL, *Intrusion Detection in Computers, Summary of the Trusted Information Systems (TIS) Report on Intrusion Detection Systems*, Janvier 1991.

[MEINADIER, 91] J.P. MEINADIER, *L'interface utilisateur - Pour une informatique plus conviviale*, DUNOD, 1991.

[MERCER, 90] Lindsay C. J. MERCER, FCA MBCS, *Tailor-made Auditing of Information Systems for the Dectection of Fraud, Computers & Security*, Vol. 9, n°1, 1990, pages 59-66.

[NEWBERRY, 90] Michael NEWBERRY, *Minos : Extended User Authentication*, Proceedings of Advances in Cryptology (Auscrypt), 1990, pages 410-423.

[NOTES, 94] *Brussels Branch Of BNP Hit By Computer Fraud*, Software Engineering Notes, Vol. 19, N°1, Janvier 1994, pages 6-7.

Entretien avec Denise HEAGERTY, Responsable DECnet du CERN.

Annexe 1
Les programmes

Le programme de monitoring

```
/*****************************************************************
* This file contains the data structure, constant and type's definitions *
* used in the programs monitor.c and treatment.c.                *
*****************************************************************/


#define MAX_CONNECTIONS     550              /* maximum number of connections */

#define MAX_LINKS           55       /* maximum number of links in a connection */

#define MAXHASH             1024         /* number of entries in the hash table */

#define MAXSIZE             82                  /* maximum size of the keywords
                                                   ( including '\n' and '\0' ) */
#define MAXKEY              20                   /* maximum number of keywords */

#define MAXROUTERS          10          /* maximum number of routers' addresses */

#define MAXINT              4294967295                  /* maximum unsigned int */

#define MAXPACK             24192000     /* number of packets seen on a connection
                                            during 2 weeks, according that there
                                            are 20 packets a sec */
enum bool {false, true};
typedef enum bool  bool;

typedef unsigned char  add[6];            /* contains the address of a router */


/* contains the keywords to search, read from a file */

typedef struct
{
   unsigned char  val[MAXSIZE];                                    /* keyword */

   int            length;                              /* length of the keyword */

} key;


/* information about a logical link */

typedef struct
{
   unsigned int   src_link;                           /* source of the link */
   unsigned int   dest_link;                     /* destination of the link */
   unsigned int   c_packets;                /* number of packets of the link */
   unsigned int   last_packet;                                  /* timestamp */
   unsigned int   data_seg[2];   /* data segment number seen in the last packet,
                                    one for each direction */
   unsigned int   id2;           /* second part of the identifier of the link */
                                  /* it is increased by one for each new link */
   time_t         id1;            /* first part of the identifier of the link */
                                  /* it will contain the starting hour of the
                                     program, thus we can restart the program
                                     during a day and concatanate all the files
                                     of the day because the identifiers will be
                                     unique ( even after the restart of the
                                     program ) */

   int            count[MAXKEY];       /* contains the counter for the keywords,
                                          in the same order as in the file */
   int            last_host;           /* contains the first place free to insert
                                          the next character in 'host' */
```

```
   int            last_server;        /* contains the first place free to insert
                                         the next character in 'server' */

   char           src_uname[16];                    /* username of the source */
   char           dest_uname[16];                /* username of the destination */
   char           order;     /* contains the sens of the link between the node :
                                'r' = right , 'i' = inversed ( the source node
                                is in fact the destination node and the
                                destination node is in fact the source node ),
                                '?' = unknown */
   char           direction;       /* indicates whether a link is going out from
                                      CERN ( = 'o' ), is entering CERN ( = 'i' )
                                      or we don't know ( = '?' ) */

   unsigned char  disc_reason;    /* contains the reason of the disconnection :
                                     0x99 if we haven't see the disconnect
                                     initiate packet, 0x98 if the data have been
                                     saved because it is midnight, 0x97 if the
                                     data have been saved because the timestamp
                                     became too great, 0x00 (= no error ),
                                     0x04 (= destination end user does not
                                     exist ) or 0x09 (= logical link aborted by
                                     end user ) or 0x26 (= connect request
                                     aborted by end user ) */

   unsigned char  host[MAXSIZE];          /* contains a line sent by the host */
   unsigned char  server[MAXSIZE];       /* contains a line typed by the user */
   unsigned char  old_server[MAXSIZE];      /* contains the previous line typed
                                               by the user */

   time_t         begin;                          /* time of the first packet */
   time_t         end;                             /* time of the last packet */

   bool           ip;          /* says if we have seen an IP address on the link
                                  and copied it in the commands' file.
                                  It concerns the IP packet encapsulated in
                                  Decnet packet */
} logical_link;


/* information about a connection */

typedef struct
{
   int            next; /* link between connections having the same hash code */
   int            num_links;            /* number of links of the connection */

   unsigned char  dest_node[2];           /* address of the destination node */
   unsigned char  src_node[2];                 /* address of the source node */

   unsigned int   c_packets;          /* number of packets of the connection */

   logical_link   links[MAX_LINKS];               /* links of the connection */

} connection;


/* information about a logical link which will be saved in a file */

typedef struct
{
   unsigned char  src_node[2];                 /* address of the source node */
   unsigned char  dest_node[2];           /* address of the destination node */
   unsigned char  disc_reason;    /* contains the reason of the disconnection :
```

```
                                0x99 if we haven't see the disconnect
                                initiate packet, 0x98 if the data have been
                                saved because it is midnight, 0x97 if the
                                data have been saved because the timestamp
                                became too great, 0x00 (= no error ),
                                0x04 (= destination end user does not
                                exist ) or 0x09 (=logical link aborted by
                                end user ) or 0x26 (= connect request
                                aborted by end user ) */


   unsigned int   src_link;                              /* source of the link */
   unsigned int   dest_link;                        /* destination of the link */
   unsigned int   c_packets;                   /* number of packets of the line */
   unsigned int   id2;          /* second part of the identifier of the link */
                                 /* it is increased by one for each new link */

   time_t         id1;           /* first part of the identifier of the link */
                                 /* it will contain the hour of starting of the
                                    program, thus we can restart the program
                                    during a day and concatanate all the files
                                    of the day because the identifiers will be
                                    unique ( even after the restart of the
                                    program ) */

   int            count[MAXKEY];       /* contains the counter for the keywords,
                                          in the same order as in the file */

   char           src_uname[16];                  /* username of the source */
   char           dest_uname[16];            /* username of the destination */
   char           order;     /* contains the sens of the link between the node :
                                'r' = right , 'i' = inversed ( the source node
                                is in fact the destination node and the
                                destination node is in fact the source node ),
                                '?' = unknown */

   time_t         begin;                          /* time of the first packet */
   time_t         end;                             /* time of the last packet */

} call_record;


/* information about a command */

typedef struct
{
  unsigned int   id_link2;      /* contains the second part of the identifier of
                                   the link on which the command was seen */

  time_t         id_link1;       /* contains the first part of the identifier of
                                    the link on which the command was seen */

  char           origin;          /* says who sent the line : 's' = the server,
                                                              'h' = the host
                                                              'o' = line typed
                                                                    by the
                                                                    server
                                                                    which
                                                                    trigerred a
                                                                    suspicious
                                                                    answer of
                                                                    the host
                                                              '?' = command of
                                                                    a non CTERM
                                                                    connec-
```

```
                                                                    tion
                                                              'i' = IPaddresses
                                                                    contained
                                                                    in an
                                                                    encapsula-
                                                                    ted
                                                                    packet */
  unsigned char  cmd[MAXSIZE];                        /* contains the command */

  time_t         moment;   /* contains the hour at which the command was seen */

}command;


/* contains the rules to apply while printing the information */
/* the rule of index 'i' applies to the word of index 'i' in the structure */
/* 'keyworlist' */


typedef struct
{
  int  count;         /* contains the number of times we must have seen the word
                         to print the line */
  int  moment;        /* contains the period of time during which the occurences
                         of the word must have been seen. 0 means that there is
                         no restriction about the period of time */
  int  next;     /* contains the index of the word concerned by the second part
                    of the rule if the rule contains an "AND". -1 means that
                    there isn't any second part.
                    note that a rule with an 'OR' may be expressed as two
                    separated rules */

}condition;
```

```
/*************************************************************************
* This file contains all the names and the result's types of the functions *
* used in the program monitor.c.                                          *
*************************************************************************/

void init_keywordlist();

void init_router();

void uppercase( unsigned char *, int, int );

void en_t_ast();

void save_data_of_the_day();

void change_file_name();

void en_a_ast();

void treatment_of_n_packets();

void update_connections( int *, int * );

int find_pair_index( unsigned char *, unsigned char * , int *, char * );

void doplace_connection();

void doplace_link( int );

void remove_link( int, int );

void tidy_hash( int, int );

void save_data( int, int );

void process_frame( int, int );

void keyword( int, int, int, int );

int find_word( unsigned char *, int, int, unsigned char *, int );

void decode_cterm( int, int, unsigned int, unsigned int );

void analyse_server( int, int );

void analyse_host( int, int );

void save_command( int, int, char, unsigned char * );

void treatment_ip( int, int, int );
```

```
#undef DEBUG                                    /* only for debugging purpose */

#define ASSERT( p ) if ( !( p ) ) { printf( "assertion p failed\n" ); abort(); }
#define ustr( x )  ( unsigned char * ) ( x )    /* type casting for the string */
/* macro used to read addresses written in hexadecimal */
#define translt( x ) ( x < 'A' ? x++ - '0' :                    \
                       x < 'a' ? x++ - 'A' + 10 : x++ - 'a' + 10 )

#include <file.h>
#include <string.h>
#include <stdio.h>
#include <time.h>
#include <ctype.h>

#include "data_struct.h"
#include "monitor.h"

/* global variables */
/* this variables are used or modified by all the functions */

int             nc = 0;                         /* number of connections in conn */
int             maxnc = 0;                    /* for statistics : maximum number of
                                                 connections during a day */
int             maxnl = 0;              /* for statistics : maximum number of links
                                           during a day */
int             conn_hash[MAXHASH];                          /* hash table */
int             stat_hash[MAXHASH]; /* for the statistics about the hash table :
                                       number of connections with the same hash
                                       code ( the hash code is the index in the
                                       table ) at a moment */
int             stat[MAXHASH];        /* for the statistics about the hash table :
                                         maximum number of connections with the
                                         same hash code ( the hash code is the
                                         index in the table ) since the program
                                         has been launched */
int             offset;
int             num_of_key;                     /* number of keywords to search */
int             num_of_router;                  /* number of routers' addresses */
int             hour = 99;                     /* used to see if it is midnight */
int             fd = -1;                  /* descriptor of the call records' file */
int             fdcmd = -1;  /* descriptor of the file containing the suspicious
                                commands */

unsigned int    n_packets = 0;              /* total number of analysed packets */
unsigned int    link_id2 = 0;     /* second part of the identifier of the links :
                                     it is increased by one for each new link */

time_t          link_id1;         /* first part of the identifier of the links */
                                  /* it will contain the hour of starting of the
                                     program, thus we can restart the program
                                     during a day and concatanate all the files
                                     of the day because the identifiers will be
                                     unique ( even after the restart of the
                                     program ) */
```

```
char            filename[128];/* name of the file containing the call records */
char            filecommand[128]; /* name of the file containing the suspicious
                                     commands */

unsigned char   address[20];       /* contains the Ethernet part of the packet */
unsigned char   buffer[1500];                          /* contains the packet */

bool            all = false;    /* by default we don't consider the links between
                                   routers */
bool            out = false;    /* by default we only consider the links entering
                                   CERN */
bool            sensitive = false;              /* by default the search is case
                                                   unsensitive */
bool            statistics = false;/* says if we have to print the statistics or
                                      not - here we don't have to */
bool            test = false;             /* says if we are in a period of test -
                                             here it is not the case */

add             router[MAXROUTERS] = { {0,0,0,0,0,0 } }; /* list of the routers'
                                                            addresses */

connection      conn[MAX_CONNECTIONS];    /* information about the connections */

key             keywordlist[MAXKEY+1];        /* list of the keywords to search */
                                              /* MAXKEY+1 because we must take
                                                 into account the case where there
                                                 are more than MAXKEY keyword in
                                                 the file */

FILE            *flog;  /* descriptor of the logfile, containing the warnings */

extern int  assign(), init_read(), last_pkt_size(), SYS$HIBER();
extern void en_a_ast(), en_t_ast();


main( argc, argv )

/*****************************************************************************
* This program intercepts Decnet packets on the network. On basis of these    *
* packets it monitors the different links and tries to discover the suspicious*
* ones. This is done by searching for some keywords in the packets and        *
* counting the number of occurrences of these keywords seen on a link.        *
*****************************************************************************/

int  argc;
char *argv[];

{
  int   i;

  char  *usage = "Usage : monitor [-a][-o][-s]\n";

  link_id1 = time( 0 );                          /* init of the first part of the
                                                    identifier of the links */

  if ( statistics == true )                        /* if we made statistics */
```

```
  {  for ( i=0; i<MAXHASH; i++ ) stat_hash[i] = 0;    /* for the statistics about
                                                         the hash table */
     for ( i=0; i<MAXHASH; i++ ) stat[i] = 0;         /* for the statistics about
                                                         the hash table */
  }
  for ( i=0; i<MAXHASH; i++ ) conn_hash[i] = -1;     /* init of the hash table */
  assign();                                        /* assignment of the channels */
  /* treatment of the parameters */
  for ( i=1; ( ( i<argc) && ( *argv[i]=='-') ); i++ )
  {   switch ( argv[i][1] )
      {  case 'a' : all = true;            /* we consider the connections between
                                              routers */
                    break;
         case 'o' : out = true;     /* we consider the links going out of CERN */
                    break;
         case 's' : sensitive = true;              /* the search for keywords is
                                                      case sensitive */
                    break;
         default  : printf( usage );               /* there is a mistake in the
                                                      syntax of the command */
                    exit();
      }
  }

#ifdef DEBUG
  printf( "MAXINT : %u\n", MAXINT );
  printf( "all : %d out : %d sensitive : %d\n", all, out, sensitive );
#endif

  init_keywordlist();        /* init of the structure containing the keywords */
  if ( all == false ) init_router();             /* if we ignore the connections
                                                    between routers */
  change_file_name();           /* init of the name of the call records' file */
  init_read( en_a_ast, buffer, address);      /* reads a packet and treats it */

#ifdef DEBUG
  printf( "end of init_read\n" );
#endif

  SYS$HIBER();                              /* waiting to be woken up by an ast */
}
```

```
void init_keywordlist()

/*****************************************************************
* This function initializes the list of the keywords to search. *
* This list is in the file "keyword.txt".                        *
*****************************************************************/

{
  FILE  *fkey;
  int   idx, len;
  int   i, j;

  for ( i=0; i<MAXKEY; i++ ) /* init of the structure containing the keywords */
  {   for ( j=0; j<MAXSIZE; j++ )
          keywordlist[i].val[j] = ' ';
      keywordlist[i].length = 0;
  }
  fkey = fopen( "keyword.txt", "r" );
  idx = 0;
  while ( fgets( keywordlist[idx].val, MAXSIZE, fkey ) )     /* gets the word */
  {
       len = strlen( keywordlist[idx].val );     /* gets the length ( including
                                                    '\n' if present ) */
       if ( keywordlist[idx].val[0] == '!' )               /* it's a comment */
       {  if ( keywordlist[idx].val[len-1] != '\n' )/* the line has been cut */
             while ( fgetc( fkey ) != '\n' );  /* skips the end of the line */
          continue;                                    /* skips the comments */
       }
       if ( keywordlist[idx].val[0] == 'R' && keywordlist[idx].val[1] == ' ' )
                                                           /* it's a rule */
       {  if ( keywordlist[idx].val[len-1] != '\n' )/* the line has been cut */
             while ( fgetc( fkey ) != '\n' );  /* skips the end of the line */
          continue;                                     /* skips the rules */
       }
       if ( keywordlist[idx].val[len-1] != '\n' )   /* the word has been cut */
       {
          while ( fgetc( fkey ) != '\n' );      /* skips the end of the word */
          len--;                          /* discards the last char from the length
                                             because MAXSIZE includes a place for
                                             '\n' and '\n' isn't present here */
          if ( len > 0 && keywordlist[idx].val[len-1] == ' ' ) /* if there are
                                                                  blanks after
```

```
                                                                    the word,
                                                                    skips them */
        {  for ( i=len-1; keywordlist[idx].val[i]==' ' && i>=0; i-- ) ;
           len = i + 1;
        }
        if ( len == 0 ) continue;                       /* skips empty lines */
        flog = fopen( "logfile.log", "a");
        fprintf(flog,"the word number %d in the keyword file has been cut\n",
                idx+1 );                        /* signals the error in logfile */
        fclose( flog );
      }
      else                                      /* the word isn't too long */
      {  keywordlist[idx].val[len-1] = '\0';       /* replaces '\n' by '\0' */
         len--;                             /* discards the '\0' from the length */
         if ( len > 0 && keywordlist[idx].val[len-1] == ' ' ) /* if there are
                                                                 blanks after
                                                                 the words,
                                                                 skips them */
         {  for ( i=len-1; keywordlist[idx].val[i]==' ' && i>=0; i-- ) ;
            len = i + 1;
         }
         if ( len == 0 ) continue;                    /* skips empty lines */
      }
      if ( idx >= MAXKEY )        /* there are too much keywords in the file */
                                  /* put here because we must first treat the
                                     comments' lines, the rules' lines and the
                                     empty lines */
      {  flog = fopen( "logfile.log", "a" );/* signals the error in logfile */
         fprintf( flog, "there are more than %d keywords\n", MAXKEY );
         fclose( flog );
         break;                                     /* stops reading words */
      }
      keywordlist[idx].length = len;
      for ( i=len; i<MAXSIZE; i++ )       /* discards garbbage from the end of
                                             the line */
          keywordlist[idx].val[i] = ' ';
      if ( sensitive == false )        /* if the search is case unsensitive */
         uppercase( keywordlist[idx].val, 0, len - 1 );
      idx++;
  }
  fclose( fkey );
  num_of_key = idx;
```

```
  if ( num_of_key == 0 )     /* if the file is empty we exit from the program */
  {  printf( "No keyword to search for. Verify the file keyword.txt.\n" );
     exit();
  }

#ifdef DEBUG
  printf( "num_of_key : %d\n", num_of_key );
  for ( i=0; i<num_of_key; i++ )
      printf( "keyword : %s length : %d\n", keywordlist[i].val,
              keywordlist[i].length );
#endif
}


void init_router()

/*************************************************************************
* This function initializes the list of the routers' addresses.         *
* The list is in the file "routers.txt" and the addresses are written   *
* in hexadecimal.                                                       *
*************************************************************************/

{
  FILE                  *frouter;
  int                   idx,i;
  unsigned char         rout_add[MAXSIZE];
  register short        count;
  register short        len;
  register unsigned char  *desti;
  register unsigned char  *point;

  frouter = fopen( "routers.txt", "r" );
  idx = 0;
  while ( fgets( rout_add, MAXSIZE, frouter ) )          /* gets the address */
  {
     len = strlen( rout_add );                            /* gets the length */
     if ( rout_add[len-1] != '\n' )              /* the address has been cut */
        while ( fgetc( frouter ) != '\n' ) ;  /* skips the end of the address */
     if ( rout_add[0] == '!' ) continue;              /* skips the comments */
     point = ( unsigned char * ) strtok( rout_add, " \n");
     if ( point == NULL ) continue;                    /* skips empty lines */
```

```
    len = strlen( point );                    /* gets the length of the token */
    if ( idx >= MAXROUTERS )      /* there are too much addresses in the file */
    {  flog = fopen( "logfile.log", "a" );
       fprintf( flog, "there are more than %d routers' addresses\n",
                MAXROUTERS );
       fclose( flog );
       break;
    }
    desti = router[idx];
    for ( count=0; count<6; count++ )
    {   if ( ( len--2 ) < 0 )     /* there aren't enough bytes in the address */
        {  flog = fopen( "logfile.log", "a" );
           fprintf( flog,"the address %d in the file contains only %d bytes\n",
                    idx+1, count );
           fclose( flog );
           break;
        }
        /* translate the address written in hexadecimal */
        *desti++ = ( unsigned char ) ( ( ( translt ( *point ) ) << 4 )
                                     | ( translt ( *point ) ) );
        if ( len > 0 )           /* skips the separator '-' between the bytes */
           if ( *point == '-' )
           {  point++;
              len--;
           }
    }
    if ( len > 0 )                                  /* the address is too long */
    {  flog = fopen( "logfile.log", "a" );
       fprintf( flog, "the address %d has been cut\n", idx+1 );
       fclose( flog );
    }
    idx++;
  }
  fclose( frouter );
  num_of_router = idx;
  if ( num_of_router == 0 )      /* if there is no address in the file we exit */
  {  printf( "There is no router's address in the file\n" );
     exit();
  }

#ifdef DEBUG
  printf( "num_of_router : %d\n", num_of_router );
  for ( i=0; i<num_of_router; i++ )
       printf( "router's address : %02x%02x%02x%02x%02x%02x\n",
               router[i][0], router[i][1], router[i][2], router[i][3],
               router[i][4], router[i][5] );
#endif
}


void uppercase( word, begin, end )

/*****************************************************************
* This function capitalizes all lowercase letters of 'word'.     *
* 'begin' is the index of the first letter of the word and 'end'  *
* is the index of the last one.                                   *
*****************************************************************/

unsigned char   *word;
int             begin, end;

{
  int  i;

#ifdef DEBUG
  printf( "beginning of uppercase\n" );
#endif

  for ( i=begin; i<=end; i++ )
     if ( 'a' <= word[i] && word[i] <= 'z' )
        word[i] = word[i] + 'A' - 'a';
}


void en_t_ast()

/*********************************************************************
* This function calls save_data_of_the_day and change_file_name if it is *
* midnight.                                                          *
* This function changes the name of the files every hour but saves the   *
* data only at midnight.                                             *
*********************************************************************/
```

```
{
  time_t        tim;
  struct tm     *ttm;

  int           i;
  int           res;

  tim = time( 0 );
  ttm = localtime( &tim );

#ifdef DEBUG
  printf( "old hour : %d new hour : %d fd : %d\n", hour, ttm->tm_hour, fd );
#endif

  /* we create a new file every hour but save the data only at midnight */
  if ( hour != ttm->tm_hour && hour != 99 && ttm->tm_hour != 0 )
  {  if ( fd != -1 )               /* if there is a call records' file opened */
     {  res = close( fd );                              /* closes the old file */
        if ( res == -1 )       /* if an error occurred while closing the file */
        {
           perror( "close" );
           exit();
        }
        fd = -1;             /* because there is no call records' file opened */
     }
     if ( fdcmd != -1 )               /* if there is a commands' file opened */
     {
        res = close( fdcmd );                                   /* closes it */
        if ( res == -1 )       /* if an error occurred while closing the file */
        {
           perror( "close" );
           exit();
        }
        fdcmd = -1;               /* because there is no commands' file open */
     }
     change_file_name();                     /* changes the name of the files */

#ifdef DEBUG
     printf( "the old file has been closed, the new one will be open\n" );
#endif
```

```
     fd = open( filename, O_WRONLY | O_CREAT, 0644 );  /* opens the new files */
     if ( fd < 0 )             /* if an error occurred while opening the file */
     {  perror( "open" );
        exit();
     }
     fdcmd = open( filecommand, O_WRONLY | O_CREAT, 0644 );
     if ( fdcmd < 0 )          /* if an error occurred while opening the file */
     {  perror( "open" );
        exit();
     }
  }
  if ( hour != ttm->tm_hour && ttm->tm_hour == 0 )
                                                    /* if it is midnight */
  {
     save_data_of_the_day();          /* saves the data of the day in the file */
     if ( fd != -1 )               /* if there is a call records' file opened */
     {  res = close( fd );                                  /* closes it */
        if ( res == -1 )       /* if an error occurred while closing the file */
        {  perror( "close" );
           exit();
        }
        fd = -1;                     /* there is no call records' file opened */
     }
     if ( fdcmd != -1 )                /* if there is a commands' file opened */
     {  res = close( fdcmd );                            /* closes the file */
        if ( res == -1 )       /* if an error occurred while closing the file */
        {  perror( "close" );
           exit();
        }
        fdcmd = -1;                       /* there is no commands' file opened */
     }
     if ( statistics == true )         /* if we've made statistics prints them */
     {  printf( "maximum number of connections : %d\n", maxnc );
        printf( "maximum number of links : %u\n", maxnl );
        maxnc = 0;
        maxnl = 0;
```

```
        printf( "stat about the hash table :\n" );
        for ( i=0; i<MAXHASH; i++ )  printf( " %d : %d\n", i, stat[i] );
      }
      change_file_name();                        /* changes the name of the files */
      /* opens the files of the next day */
      fd = open( filename, O_WRONLY | O_CREAT, 0644 );
      if ( fd < 0 )              /* if an error occurred while opening the file */
      {  perror( "open" );
         exit();
      }
      fdcmd = open( filecommand, O_WRONLY | O_CREAT, 0644 );
      if ( fdcmd < 0 )           /* if an error occurred while opening the file */
      {  perror( "open" );
         exit();
      }
   }
   hour = ttm->tm_hour;                                  /* updates the hour */
}


void save_data_of_the_day()

/*****************************************************************
* This function is called at midnight to save the data about the      *
* connections of the day.                                             *
*****************************************************************/

{
  int i, j;

#ifdef DEBUG
  printf( "beginning of save data of the day\n" );
#endif

  for ( i=0; i<nc; i++ )                              /* for each connection */
      for ( j=0; j<conn[i].num_links; j++ )                /* for each link */
      {   conn[i].links[j].disc_reason = 0x98;     /* signals that the data have
                                                      been saved because it is
                                                      midnight */
          save_data( i, j );                 /* saves the data about the link */
      }
}
```

```
void change_file_name()

/*****************************************************************************
* This function is called at midnight to change the name of the files.       *
* This name is made of the word "call_records" and the date ( with the hour ) *
* and the word "command" and the date ( with the hour ).                      *
*****************************************************************************/

{
  time_t       tim;
  struct tm    *ttm;

  int          day, month, year, h;

  tim = time( 0 );                                          /* gets the hour */
  ttm = localtime( &tim );
  day = ttm->tm_mday;
  month = ttm->tm_mon + 1;
  year = ttm->tm_year + 1900;
  h = ttm->tm_hour;

  sprintf( filename, "call_records_%02d-%02d-%02d-%02d.dat",
           day, month, year, h );           /* name of the call_records' file */

  sprintf( filecommand, "command_%02d-%02d-%02d-%02d.dat",
           day, month, year, h );                /* name of the commands' file */

#ifdef DEBUG
  printf( "name of the call_records' file : %s \n", filename);

  printf( "name of the commands' file : %s\n", filecommand );

#endif

}


void en_a_ast()

/*****************************************************************************
* This function picks up the Decnet packets with data packet format 0x26  *
* and 0x2E and with message type 0x60 ( data message : begin-end ), 0x00  *
* ( data message  : middle ), 0x20 ( data message : begin ), 0x40 ( data  *
* message : end ), 0x18 ( connect initiate ), 0x28 ( connect confirm )    *
* and 0x38 ( disconnect initiate ), sets the offset and calls             *
* update_connections and for the message of type 0x60, 0x00, 0x20 and 0x40*
* calls process_frame.                                                    *
*****************************************************************************/

{
  unsigned char cb;

  int          numcon, numlink;
  int          i,j;
```

```
  bool          treatment;

#ifdef DEBUG
  if ((( address[0]==0xaa && address[1]==0x00 && address[2]==0x04 &&
         address[3]==0x00 && address[4]==0x50 && address[5]==0x58 ) ||
       ( address[0]==0xaa && address[1]==0x00 && address[2]==0x04 &&
         address[3]==0x00 && address[4]==0x09 && address[5]==0x5c ) ||
       ( address[0]==0xaa && address[1]==0x00 && address[2]==0x04 &&
         address[3]==0x00 && address[4]==0x66 && address[5]==0x59 ))
      &&
      (( address[6]==0xaa && address[7]==0x00 && address[8]==0x04 &&
         address[9]==0x00 && address[10]==0x50 && address[11]==0x58 ) ||
       ( address[6]==0xaa && address[7]==0x00 && address[8]==0x04 &&
         address[9]==0x00 && address[10]==0x09 && address[11]==0x5c ) ||
       ( address[6]==0xaa && address[7]==0x00 && address[8]==0x04 &&
         address[9]==0x00 && address[10]==0x66 && address[11]==0x59 )))
  {
#endif

#ifdef DEBUG
  printf( "beginning of en_a_ast\n" );

  printf( "%02x%02x%02x%02x%02x%02x\n",
          address[0], address[1], address[2], address[3], address[4],
          address[5] );

  printf( "%02x%02x%02x%02x%02x%02x\n",
          address[6], address[7], address[8], address[9], address[10],
          address[11] );

  printf( "%02x%02x\n", address[12], address[13] );

  for ( i=0; i<32; i++ )

      printf( "%02x", buffer[i] );

  printf( "\n" );
#endif

  if ( (address[12] == 0x60) && (address[13] == 0x03) )             /* Decnet */
  {  offset = 1 ;                  /* By default, the offset is set to 1 because
                                      C array subscripts have 0 as a lower bound,
                                      and we have to skip the 2 bytes containing
                                      the data length */
#ifdef DEBUG
     printf( "determination of the offset :\n" );

     printf( "buffer[1+offset] : %02x\n", buffer[1+offset] );
#endif

     if ( buffer[1+offset] != 0x81 )
        offset = offset - 1;                        /* there isn't any padding */
```

```
#ifdef DEBUG
     printf( "offset : %d\n", offset );

     printf( "src_add : %02x%02x\n", buffer[17+offset], buffer[18+offset] );

     printf( "dest_add : %02x%02x\n", buffer[9+offset], buffer[10+offset] );
#endif

     cb = buffer[2+offset];

#ifdef DEBUG
     printf( "DRP ? %02x\n", cb );
#endif

     if ( (cb == 0x26) || (cb == 0x2E) )              /* format of the packet */
     {   cb = buffer[23+offset];

#ifdef DEBUG
         printf( "NSP or SCP ? %02x\n", cb );
#endif

         if ( (cb == 0x60) || (cb == 0x38) || (cb == 0x28) || (cb == 0x18) ||
              (cb == 0x00) || (cb == 0x20) || (cb == 0x40) )
                                                          /* type of message */
         {
            treatment = true;       /* start of the comparison with the routers'
                                       addresses, if asked.
                                       Put here for performance reasons */

            if ( all == false )   /* if we don't consider all the connections */

               for ( i=0; i<num_of_router; i++ )         /* for all the routers'
                                                           addresses */

                  if ( memcmp( address, &router[i], 6 ) == 0 ) /* compairs this
                                                                  address with
                                                                  the dest
                                                                  address */

                     for ( j=0; j<num_of_router; j++ )  /* for all the routers'
                                                           addresses */

                        if ( memcmp( &address[6], &router[j], 6 ) == 0 )

                        /* compairs this address with the src address */

                           treatment = false; /* if each address is equal to
                                                 a router's address,
                                                 we don't treat the packet */
```

```
#ifdef DEBUG
            printf( "treatment : %d\n", treatment );
#endif

            /* if 'test == true' we only consider the links going out or
               entering VXCRNA */
            if ( ( buffer[17+offset] == 0x2f && buffer[18+offset] == 0x58 ) ||
                 ( buffer[9+offset] == 0x2f && buffer[10+offset] == 0x58 ) ||
                 ( test == false ) )
            {  if ( treatment == true )
               {
                  treatment_of_n_packets();          /* there is one more packet
                                                        analysed */

#ifdef DEBUG
                  printf( "call update_connections\n" );
#endif

                  update_connections( &numcon, &numlink );
                  /* we have to test the type of message before the direction of
                     the link because if 'cb == 0x38' it is possible that
                     'numcon' and 'numlink' didn't receive any value. It is not
                     a problem because in this case, the treatment of the packet
                     is finished */
                  if ( (cb == 0x60) || (cb == 0x00) || (cb == 0x20) ||
                       (cb == 0x40) )                /* for the data messages */

                  {  ASSERT( numcon<MAX_CONNECTIONS );
                     ASSERT( numlink<MAX_LINKS );
                     /* if we also consider the links going out of CERN or if we
                        have a link entering CERN or if we have a link which we
                        don't know the direction */
                     if ( out == true ||
                          conn[numcon].links[numlink].direction == 'i' ||
                          conn[numcon].links[numlink].direction == '?' )
                        process_frame( numcon, numlink );
                  }
               }
            }                              /* closes the 'if' concerning VXCRNA */
         }
      }
   }
#ifdef DEBUG
   printf( "call init_read\n" );
   }                    /* this close the first if, included in a #ifdef DEBUG */
```

```
#endif

   init_read( en_a_ast, buffer, address );             /* reads another packet */
}


void treatment_of_n_packets()

/*************************************************************************
* This function counts the number of analysed packets and updates it and *
* the links' timestamp if this number is to great.                       *
*************************************************************************/

{
   int            i, j;
   unsigned int   min;

   if ( n_packets == MAXINT )                      /* if n_packets is too great */
   { min = MAXINT;        /* searchs for the smaller last_pack among the links */
     for ( i=0; i<nc; i++ )
         for ( j=0; j<conn[i].num_links; j++ )
             if ( conn[i].links[j].last_packet < min )
                min = conn[i].links[j].last_packet;
     if ( min < MAXPACK )                             /* if min is too small */
     { for ( i=nc-1; i>=0; i-- )                     /* begins with the end
                                                        because of remove_link */
       {   for ( j=conn[i].num_links-1; j>=0; j-- )
               if ( conn[i].links[j].last_packet < MAXPACK ) /* for all the links
                                                               with last_packet
                                                               smaller than
                                                               MAXPACK */
               {  conn[i].links[j].disc_reason = 0x97; /* signals that the data
                                                          have been saved because
                                                          n_packets became too
                                                          great */
                  remove_link( i, j );                  /* removes the link */
               }
       }
       min = MAXPACK;
     }
     for ( i=0; i<nc; i++ )                /* updates last_packet in the links */
         for ( j=0; j<conn[i].num_links; j++ )
             conn[i].links[j].last_packet = conn[i].links[j].last_packet - min+1;
     /* we do +1 because we don't want any link with last_packet = 0 */
```

```
    n_packets = n_packets - min + 1;                    /* updates n_packets */
  }

  n_packets++;                              /* there is one more packet analysed */

#ifdef DEBUG
  printf( "n_packets : %u\n", n_packets );
#endif
}


void update_connections( ncon, nlink )

/*********************************************************************
* This function cheks to see if the current packet is from a         *
* conversation which already exits or if it is a new connection.     *
* If it is a new connection, it adds it to connections' list.        *
* It updates the hash table and the information about the connection, *
* and returns the index of the connection and the link in '*ncon' and *
* '*nlink'.                                                          *
*********************************************************************/

int           *ncon;
int           *nlink;

{
  bool          new_conn, new_link;
  bool          no_remove;

  unsigned int  dlink, slink;
  unsigned int  i, k, l;
  unsigned int  temp;
  unsigned int  hash_code;
  unsigned int  src, dest;

  int           prev, ind, j;

  unsigned char saddr[2];
  unsigned char daddr[2];

  char          ord;

#ifdef DEBUG
  printf( "beginning of update_connections\n" );
#endif

  new_conn = false;
  new_link = false;
  no_remove = false;

  /* calculation of the logical link values */
  /* note : the bytes are inversed */
```

```
  dlink = ( buffer[25+offset] << 8 ) | buffer[24+offset];
  slink = ( buffer[27+offset] << 8 ) | buffer[26+offset];

#ifdef DEBUG
  printf( "logical link values :\n" );
  printf( "buffer[25+offset] : %02x\nbuffer[24+offset] : %02x\n",
          buffer[25+offset], buffer[24+offset] );
  printf( "buffer[27+offset] : %02x\nbuffer[26+offset] : %02x\n",
          buffer[27+offset], buffer[26+offset] );
  printf( "dlink : %u\nslink : %u\n",dlink, slink );
  printf( "dlink : %02x\nslink : %02x\n",dlink, slink );
#endif

  saddr[0] = buffer[17+offset];                          /* source address */
  saddr[1] = buffer[18+offset];
  daddr[0] = buffer[9+offset];                      /* destination address */
  daddr[1] = buffer[10+offset];

#ifdef DEBUG
  printf ( "call find_pair_index\n" );
#endif

  ind = find_pair_index( saddr, daddr, &prev, &ord );     /* does the connection
                                                             exist ? */

#ifdef DEBUG
  printf( "entry of connection : %d\nprevious link : %d\n", ind, prev );
  printf( "ord : %c\n", ord );
#endif

  if ( ind != -1 )                                   /* the connection exists */
  {  conn[ind].c_packets++;     /* there is one more packet on the connection */
     j = 0;                                          /* does the link exist ? */
     while ( j < conn[ind].num_links )
        if ( (dlink == conn[ind].links[j].dest_link &&              /* yes */
              slink == conn[ind].links[j].src_link) ||
             (dlink == conn[ind].links[j].src_link &&
              slink == conn[ind].links[j].dest_link) )
        {  conn[ind].links[j].c_packets++;           /* there is one more packet
                                                        on the link */
           break;
        }
        else j++;
```

```
    if ( j == conn[ind].num_links )
    {  j = 0;
       while ( j < conn[ind].num_links )          /* has the link been created
                                                     via a SCP packet ? */
          if ( dlink == conn[ind].links[j].src_link &&
               conn[ind].links[j].dest_link == 0 )/* the link has been created
                                                     via a SCP packet */
          {  conn[ind].links[j].c_packets++;      /* there is one more packet
                                                     on the link */
             conn[ind].links[j].dest_link = slink; /* we have to complete the
                                                      destination link */
             break;
          }
          else if ( slink == conn[ind].links[j].src_link &&
                    conn[ind].links[j].dest_link == 0 )   /* the link has been
                                                             created via a SCP
                                                             packet */
               {  conn[ind].links[j].c_packets++;  /* there is one more packet
                                                      on the link */
                  conn[ind].links[j].dest_link = dlink; /* we have to complete
                                                           the destination
                                                           link */
                  break;
               }
               else j++;
    }
    if ( j == conn[ind].num_links )               /* the link doesn't exist */
    {  if ( buffer[23+offset] != 0x38 )        /* if the first packet is the */
                                               /* disconnect initiate packet,*/
          new_link = true;                     /* we don't create the link   */
                                               /* and we won't have to       */
       else no_remove = true;                  /* remove any link            */
    }
#ifdef DEBUG
    printf( "j : %u\n", j );
#endif
  }
  else                                      /* the connection doesn't exist */
  {  if ( buffer[23+offset] != 0x38 )       /* if the first packet isn't
                                               the disconnect initiate packet,
                                               creates an entry in the table
                                               for the connection */
    {
#ifdef DEBUG
       printf( "number of connections : %d\n", nc );
#endif
```

```
       if ( nc > MAX_CONNECTIONS - 1 )     /* the connections' table is full */
       {  doplace_connection();
          /* calls find_pair_index because the organization of the hash table
             has perhaps changed because of doplace_connection and 'prev' may
             have changed */
          ind = find_pair_index( saddr, daddr, &prev, &ord );
       }
       nc = nc + 1;                          /* there is one connection more */
       ASSERT( nc<=MAX_CONNECTIONS );
       if ( statistics == true )                   /* if we made statistics */
          if ( nc > maxnc ) maxnc = nc;      /* to know the greatest number of
                                                connections in the table during
                                                the day */
#ifdef DEBUG
       printf( "maxnc : %d\n", maxnc );
#endif

       ind = nc - 1;                /* index of the connection being created */

#ifdef DEBUG
       printf( "index of the connection being created : %d\n", ind );
#endif

       /* init of the connection */
       conn[ind].dest_node[0] = buffer[9+offset];            /* dest address */
       conn[ind].dest_node[1] = buffer[10+offset];
       conn[ind].src_node[0] = buffer[17+offset];             /* src address */
       conn[ind].src_node[1] = buffer[18+offset];
       conn[ind].num_links = 0;              /* the link isn't created yet */
       conn[ind].c_packets = 1;  /* number of packets seen on the connection */
       conn[ind].next = -1;               /* it is not followed by a connection
                                             having the same hash code */
       ASSERT( prev != nc-1 );
       /* calculates the hash code by making an integer of the source and the
          destination addresses, multiplying both areas and both nodes,
          adding them and making the logical and with 1023 - according to
          statistics, this hash code is better */
       dest = ( buffer[10+offset] << 8 ) | buffer[9+offset];
       src = ( buffer[18+offset] << 8 ) | buffer[17+offset];
       hash_code = ( ( ( src / 1024 ) * ( dest / 1024 ) ) +
                   ( ( src % 1024 ) * ( dest % 1024 ) ) ) & 1023;
```

```
        if ( statistics == true )                    /* if we made statistics */
        {  stat_hash[hash_code]++;                   /* statistics about the use
                                                        of the hash table */
           if ( stat_hash[hash_code] > stat[hash_code] )
              stat[hash_code] =  stat_hash[hash_code];
        }
        /* entry in the hash table */
        if ( prev == -1 )                        /* if it is the first connection
                                                    with that hash code */
        {  conn_hash[hash_code] = ind;

#ifdef DEBUG
           printf( "conn_hash[%u] : %d\n", hash_code, conn_hash[hash_code] );
#endif
        }
        else         /* if there is other connections with the same hash code */
        {  conn[prev].next = ind;

#ifdef DEBUG
           printf( "conn[%d].next : %d\n", prev, conn[prev].next );
#endif

           ASSERT( prev != ind );
        }
        new_conn = true;                             /* we have a new connection */
        new_link = true;                                  /* we have a new link */
     }
     else no_remove = true;     /* if we only see the disconnect initiate packet,
                                   we haven't create the link and the connection,
                                   and we won't have to remove any link or
                                   connection */
  }

#ifdef DEBUG
  printf( "new_conn : %d new_link : %d no_remove : %d\n",
          new_conn, new_link, no_remove );
#endif

  if ( new_link == true )                    /* the logical link doesn't exist */
  {  en_t_ast();               /* before creating a new link, we look at the hour
                                  and eventually change the name of the files */
     /* there are MAX_LINKS links for the connection */
```

```
     if ( conn[ind].num_links > MAX_LINKS - 1 ) doplace_link( ind );
     conn[ind].num_links ++;                         /* there is one more link */
     ASSERT( conn[ind].num_links<= MAX_LINKS );
     if ( statistics == true )                        /* if we made statistics */
        if ( conn[ind].num_links > maxnl )
           maxnl = conn[ind].num_links;  /* to know the greatest number of links
                                            for a connection during a day */
#ifdef DEBUG
     printf( "maxnl : %u\n", maxnl );
#endif

     j = conn[ind].num_links - 1;           /* index of the link being created */

#ifdef DEBUG
     printf( "index of the link being created : %d\n", j );
#endif

     /* init of the link */
     conn[ind].links[j].dest_link = dlink;                        /* dest link */
     conn[ind].links[j].src_link = slink;                          /* src link */
     conn[ind].links[j].c_packets = 1;  /* number of packets seen on the link */
     conn[ind].links[j].data_seg[0] = 0;/* data segment number of the last
     conn[ind].links[j].data_seg[1] = 0;   packet seen-one for each direction */
     conn[ind].links[j].last_host = 0;
     conn[ind].links[j].last_server = 0;
     for ( i=0; i<MAXSIZE; i++ )
     {   conn[ind].links[j].host[i] = ' ';
         conn[ind].links[j].server[i] = ' ';
         conn[ind].links[j].old_server[i] = ' ';
     }
     link_id2++;         /* this link will receive the next identifying number */
     if ( link_id2 > MAXINT ) link_id2 = 1;
     conn[ind].links[j].id2 = link_id2;
     conn[ind].links[j].id1 = link_id1;
     conn[ind].links[j].begin = time( 0 );   /* records the hour of beginning */
     conn[ind].links[j].disc_reason = 0x99;     /* if we don't see the disconnect
                                                   initiate packet, disc_reason
                                                   will be 0x99 else it will
                                                   contain the reason of the
```

```
                                                disconnection in hexa */
    for ( i=0; i<MAXKEY; i++ )                  /* init the counters of keywords */
        conn[ind].links[j].count[i] = 0;

    conn[ind].links[j].ip = false;  /* says that we haven't seen any IP address
                                       on the link */

    /* determines the sens of the link 'r' means that the sens is OK
       'i' means that the sens has to be inversed */

    if ( buffer[23+offset] == 0x28 )        /* the first packet seen on the link
                                               is a connect confirm packet */
    {  if ( new_conn == true )                  /* it is a new connection */
          conn[ind].links[j].order = 'i';
       else                                     /* the connection already exists */
       {  if ( ord == 'r' )
             conn[ind].links[j].order = 'i';
          else conn[ind].links[j].order = 'r';
       }
    }
    else                                    /* the first packet seen on the link is
                                               not a connect confirm packet */
    {  if ( buffer[23+offset] == 0x18 ) /* the first packet seen on the link is
                                           a connect initiate packet */
       {  if ( new_conn == true )                   /* it is a new connection */
             conn[ind].links[j].order = 'r';
          else conn[ind].links[j].order = ord;      /* the connection already
                                                       exists */
       }
       else conn[ind].links[j].order = '?';
    }
    /* finds if a link is entering CERN or going out from CERN by looking
       were the source address of the connection is located */
    if ( conn[ind].links[j].order == 'r' )      /* if the order is OK we look at
                                                   the source node */
    {
       /* calculates the area of the address */
       temp = ( (conn[ind].src_node[1] << 8) | conn[ind].src_node[0] ) / 1024;
       if ( temp == 22 || temp == 23 )   /* 22 and 23 are the areas for CERN */
          conn[ind].links[j].direction = 'o';       /* the origin of the link
                                                       is in CERN */
       else conn[ind].links[j].direction = 'i';     /* the origin of the link
                                                       is not in CERN */
    }
    else if ( conn[ind].links[j].order == 'i' )  /* if the order is inversed we
                                                    look at the destination
                                                    address */
       {
```

```
          /* calculates the area of the address */
          temp = ((conn[ind].dest_node[1] << 8)|conn[ind].dest_node[0])/1024;
          if ( temp == 22 || temp == 23 )
             conn[ind].links[j].direction = 'o';    /* the origin of the link
                                                       is in CERN */
          else conn[ind].links[j].direction = 'i';  /* the origin of the link
                                                       is not in CERN */
       }
       else conn[ind].links[j].direction = '?';         /* we don't know */

#ifdef DEBUG
    printf( "temp : %u\n", temp );
#endif

    strcpy( conn[ind].links[j].src_uname, "unknown" );
    strcpy( conn[ind].links[j].dest_uname, "unknown" );
    if ( buffer[23+offset] == 0x18 )            /* connect initiate message -
                                                   we can find the source and
                                                   destination name in the
                                                   packet */
    {
#ifdef DEBUG
       printf( "buffer[32+offset] : %02x\n", buffer[32+offset] );
       printf( "buffer[33+offset] : %02x\n", buffer[33+offset] );
#endif

       if ( buffer[32+offset] == 0x00 )             /* no destination name */
       {  if ( buffer[33+offset] == 0x2a )      /* determines the protocol */
             strcpy( conn[ind].links[j].dest_uname, "CTERM" );
          else if ( buffer[33+offset] == 0x11 )
                strcpy( conn[ind].links[j].dest_uname, "FAL File Acc" );
             else if ( buffer[33+offset] == 0x1b )
                   strcpy( conn[ind].links[j].dest_uname, "VMS Mail" );
                else if ( buffer[33+offset] == 0x2c )
                      strcpy( conn[ind].links[j].dest_uname,
                              "DNS Trans Agt" );
                   else if ( buffer[33+offset] == 0x13 )
                         strcpy( conn[ind].links[j].dest_uname,
                                 "NML (NICE)" );
                      else strcpy( conn[ind].links[j].dest_uname,
                                   "unknown" );
```

```
#ifdef DEBUG
        printf( "buffer[34+offset] : %02x\n", buffer[34+offset]);
#endif

        if ( buffer[34+offset] == 0x02 )            /* with a source name */
        {  i = buffer[40+offset];                  /* length of the src name */

#ifdef DEBUG
            printf( "length of the source name :\n" );
            printf( "i : %u i : %02x\n", i, i );
            printf( "buffer[40+offset] : %02x\n", buffer[40+offset] );
#endif

            if ( i > 15 ) i = 15;
            for ( k=0; k<i; k++ )                          /* gets the name */
                conn[ind].links[j].src_uname[k] = buffer[41+offset+k];
            conn[ind].links[j].src_uname[k] = '\0';     /* to make a string */
        }
      }
      else if ( buffer[32+offset] == 0x01 )      /* with a destination name */
          {  i = buffer[34+offset];              /* length of the dest name */

#ifdef DEBUG
            printf( "length of the dest name :\n" );
            printf( "i : %u i : %02x\n", i, i );
            printf( "buffer[34+offset] : %02x\n", buffer[34+offset] );
#endif

            /* we have to use the variable 'l' because we have to keep a
               track of the length of the destination name which is in the
               variable 'i' */
            if ( i > 15 ) l = 15; else l = i;
            for ( k=0; k<l; k++ )                          /* gets the name */
                conn[ind].links[j].dest_uname[k] = buffer[35+offset+k];
            conn[ind].links[j].dest_uname[k] = '\0';  /* to make a string */

#ifdef DEBUG
```

```
            printf( "buffer[35+offset+i] : %02x\n", buffer[35+offset+i] );
#endif
            if ( buffer[35+offset+i] == 0x02 )      /* with a source name */
            {  l = buffer[41+offset+i];          /* length of the src name */

#ifdef DEBUG
                printf( "length of the source name :\n" );
                printf( "l : %u l : %02x\n", l, l );
                printf( "buffer[41+offset+i] : %02x\n\n",
                        buffer[41+offset+i] );
#endif

                if  ( l > 15 ) l = 15;
                for ( k=0; k<l; k++ )                      /* gets the name */
                    conn[ind].links[j].src_uname[k] = buffer[42+offset+i+k];
                conn[ind].links[j].src_uname[k] = '\0';/* to make a string */
            }
        }
    }
  }

  if ( no_remove == false )              /* if no_remove == true, i.e. the link
                                            hasn't been created, we don't have
                                            to update this information */
  {  conn[ind].links[j].last_packet = n_packets;                /* timestamp */
     conn[ind].links[j].end = time( 0 );      /* records the hour of the last */
                                              /* packet seen on the link */
     *ncon = ind;                                  /* index of the connection */
     *nlink = j;                                          /* index of the link */
     ASSERT( ind<MAX_CONNECTIONS );
     ASSERT( j<MAX_LINKS );

#ifdef DEBUG
     printf( "ind : %d\nconn[ind].next : %d\nconn[ind].num_links : %u\n",
             ind, conn[ind].next, conn[ind].num_links );
     printf( "conn[ind].c_packets : %u\n", conn[ind].c_packets );
     printf( "conn[ind].src_node : %02x%02x\n",
             conn[ind].src_node[0], conn[ind].src_node[1] );
     printf( "conn[ind].dest_node : %02x%02x\n",
             conn[ind].dest_node[0], conn[ind].dest_node[1] );
     printf( "ind : %d j : % d\n", ind, j );
```

```
    printf( "conn[ind].links[j].id1 : %u\n", conn[ind].links[j].id1 );
    printf( "conn[ind].links[j].id2 : %u\n", conn[ind].links[j].id2 );
    printf( "conn[ind].num_links : %u\n", conn[ind].num_links );
    printf( "number of packets of the link : %u\n",
            conn[ind].links[j].c_packets );
    printf( "conn[ind].links[j].src_link : %u\n",
            conn[ind].links[j].src_link );
    printf( "conn[ind].links[j].dest_link : %u\n",
            conn[ind].links[j].dest_link );
    printf( "src_uname : %s\ndest_uname : %s\n",
            conn[ind].links[j].src_uname, conn[ind].links[j].dest_uname );
    printf( "time of the first packet : %d\n", conn[ind].links[j].begin );
    printf( "time of the last packet : %d\n", conn[ind].links[j].end );
    printf( "conn[ind].links[j].last_packet : %u\n",
            conn[ind].links[j].last_packet );
    printf( "order : %c\n", conn[ind].links[j].order );
    printf( "direction : %c\n", conn[ind].links[j].direction );
#endif
  }

#ifdef DEBUG
  printf( "buffer[23+offset] : %02x\n", buffer[23+offset] );
#endif

  if ( ( buffer[23+offset] == 0x38 ) && ( no_remove == false ) )   /* we have to
                                                                      remove the
                                                                      link */
  {   conn[ind].links[j].disc_reason = buffer[28+offset];    /* reason of the
                                                               disconnection */
      remove_link( ind, j );                /* if the disconnect initiate packet
                                               is not the first of the link,
                                               the link has been created and
                                               we can remove it when we see
                                               the disconnect initiate packet */
      if ( statistics == true )                    /* if we made statistics */
      {  dest = ( buffer[10+offset] << 8 ) | buffer[9+offset];
         src = ( buffer[18+offset] << 8 ) | buffer[17+offset];
         hash_code = ( ( ( src / 1024 ) * ( dest / 1024 ) ) +
                     ( ( src % 1024 ) * ( dest % 1024 ) ) ) & 1023;
         stat_hash[hash_code]--;             /* statistics about the use of the
                                                hash table */
      }
  }
}
```

```
int find_pair_index ( saddr, daddr, previous, sens )

/*************************************************************************
* This function returns the entry index of the connection having 'saddr' *
* and 'daddr' as nodes if found, else returns -1.                        *
* 'Previous' is the pointer to the index of the connection with          *
* 'conn[*previous].next' pointing to this connection.                    *
* 'sens' gives the sens of the connection.                               *
*************************************************************************/

unsigned char saddr[2];
unsigned char daddr[2];
int           *previous;
char          *sens;

{
   unsigned int idx;
   unsigned int src, dest;
   int          i;

   *previous = -1;
   *sens = 'x';
   /* calculates the hash code by making an integer of the source and the
      destination addresses, multiplying both areas and both nodes,
      adding them and making the logical and with 1023 */
   src = ( saddr[1] << 8 ) | saddr[0];
   dest = ( daddr[1] << 8 ) | daddr[0];
   idx = ( ( ( src / 1024 ) * ( dest / 1024 ) ) +
         ( ( src % 1024 ) * ( dest % 1024 ) ) ) & 1023;

#ifdef DEBUG
   printf( "index found in find_index :\n" );
   printf( "idx : %u idx : %02x\n", idx, idx );
#endif

   if ( conn_hash[idx] == -1 )
      return(-1);                       /* no entry index for this connection */
   else
      { i = conn_hash[idx];
        while ( i != -1 ) /* finds if there is an entry index among conn.next */
        {     if ( ( conn[i].dest_node[0] == daddr[0] &&
```

```
                    conn[i].dest_node[1] == daddr[1] &&
                    conn[i].src_node[0] == saddr[0] &&
                    conn[i].src_node[1] == saddr[1]) ) ||
                  ( conn[i].dest_node[0] == saddr[0] &&
                    conn[i].dest_node[1] == saddr[1] &&
                    conn[i].src_node[0] == daddr[0] &&
                    conn[i].src_node[1] == daddr[1] ) )

              {  if ( conn[i].dest_node[0] == daddr[0] &&    /* determines the */
                      conn[i].dest_node[1] == daddr[1] )     /* sens of the */
                                                             /* connection */
                    *sens = 'r';

                 else *sens = 'i';

                 return(i);
              }

              else

              {  *previous = i;

                 i = conn[i].next;                 /* looks at the next connection
                                                      with this code */

#ifdef DEBUG

                 printf( "*previous : %d\n", *previous );

                 printf( "i : %d\n", i );

#endif

                 ASSERT( i != *previous );       /* only for debugging purpose */

              }
        }

        return ( -1 );                          /* no entry for this connection */
      }
}


void doplace_connection()

/*************************************************************************
* This function looks for the connection that may be removed from the table *
* 'conn' and calls the function remove_link to remove all the links of this *
* connection and the connection itself. The index MAX_CONNECTIONS-1 will be *
* free for a new connection.                                                *
*************************************************************************/

{
  unsigned int  old, recent;

  int           i, j, idx;
  int           x, y;                             /* only for debugging purpose */
```

```
#ifdef DEBUG
  printf( "beginning of doplace_connection\n" );
#endif

  old = MAXINT;
  for ( j=0; j<MAX_CONNECTIONS; j++ )                  /* for each connection */
  { recent = 0;
    for ( i=0; i<conn[j].num_links; i++ )              /* considers each link */
        if ( conn[j].links[i].last_packet > recent )           /* takes the more
                                                                  recent one */
           recent = conn[j].links[i].last_packet;
    if ( recent < old )                          /* among these links chooses the
                                                    older one */
    {  old = recent;
       idx = j;                            /* index of the connection to remove */
    }
  }

#ifdef DEBUG
  printf( "list of connections :\n" );
  for ( x=0; x<nc; x++ )
      for ( y=0; y<conn[x].num_links; y++ )
           printf( "conn[%d].links[%d].last_packet : %u\n", x, y,
                   conn[x].links[y].last_packet );
  printf( "entry of the connection being removed %d\n", idx);
#endif

  for ( j=conn[idx].num_links-1; j>=0; j-- )     /* we must first remove the last
                                                    link because of the work of
                                                    remove_link */
  {   conn[idx].links[j].disc_reason = 0x99;
      remove_link( idx, j );           /* removes the links and the connection */
  }
}


void doplace_link( ind )

/*************************************************************************
* This function removes the older link of the connection with index 'ind'.*
* The index MAX_LINKS-1 will be free for a new link.                      *
*************************************************************************/

int ind;
```

```
{
  unsigned int   max;
  int            j, nlink;

#ifdef DEBUG
  printf( "beginning of doplace_link with ind : %d\n", ind );
#endif

  max = MAXINT;
  for ( j=0; j<conn[ind].num_links; j++ )  /* for each link of the connection */
      if ( conn[ind].links[j].last_packet < max )      /* takes the older one */
      {  max = conn[ind].links[j].last_packet;
         nlink = j;                            /* index of the link to remove */
      }

#ifdef DEBUG
  for ( j=0; j<conn[ind].num_links; j++ )
      printf( "conn[%d].links[%d].last_packet : %u\n",
               ind, j, conn[ind].links[j].last_packet );
  printf( "entry of the link being removed %d\n", nlink );
#endif

  conn[ind].links[nlink].disc_reason = 0x99;
  remove_link ( ind, nlink );                              /* removes the link */
}


void remove_link( ncon, nlink )

/*************************************************************************
* This function saves the data about the link with index 'nlink' of the     *
* connection with index 'ncon' and then removes this link.                   *
* If it is the last link of the connection, it also removes the connection. *
*************************************************************************/

int           ncon, nlink;

{
#ifdef DEBUG
  printf( "beginning of remove link with ncon : %d nlink : %d\n", ncon, nlink );
#endif
```

```
  save_data( ncon, nlink );                  /* saves the data about the link */
  if ( nlink < conn[ncon].num_links - 1 )              /* if the link is not the
                                                          last of the table */
  {
     /* replaces the removed link with the last one in the table */
     memcpy ( &conn[ncon].links[nlink],
              &conn[ncon].links[conn[ncon].num_links-1],
              sizeof( logical_link ) );

#ifdef DEBUG
     printf( "link moved\n" );
     printf( "conn[ncon].links[nlink].id2 : %u\n",
              conn[ncon].links[nlink].id2 );
     printf( "conn[ncon].links[conn[ncon].num_links-1].id : %u\n",
              conn[ncon].links[conn[ncon].num_links-1].id );
#endif

     ASSERT( conn[ncon].links[nlink].id2 ==
             conn[ncon].links[conn[ncon].num_links-1].id2 );
  }
  conn[ncon].num_links--;                            /* there is one link less */
  if ( conn[ncon].num_links == 0 )/* if it is the last link of the connection */
  {  if ( ncon < nc - 1 )   /* if the connection is not the last in the table */
     {
#ifdef DEBUG
        printf( "call of tidy_hash with new!=-1\n" );
#endif

        tidy_hash( ncon, nc-1 );                /* reorganizes the hash table */
        /* replaces the removed connection with the last in the table */
        memcpy( &conn[ncon], &conn[nc-1], sizeof( connection ) );

#ifdef DEBUG
        printf( "conn[ncon].dest_node : %02x%02x ", conn[ncon].dest_node[0],
                 conn[ncon].dest_node[1] );
        printf( "conn[nc-1].dest_node : %02x%02x\n", conn[nc-1].dest_node[0],
                 conn[nc-1].dest_node[1] );
#endif

        ASSERT( conn[ncon].dest_node[0] == conn[nc-1].dest_node[0] );
```

```
        ASSERT( conn[ncon].dest_node[1] == conn[nc-1].dest_node[1] );
      }
      else tidy_hash( -1, nc-1 );                    /* reorganizes the hash table */
      nc--;                                         /* there is one connection less */
    }

#ifdef DEBUG
    printf( "number of links : %u\nnumber of connections : %d\n",
            conn[ncon].num_links, nc );
#endif
}


void tidy_hash( new, old )

/*****************************************************************************
* This function updates the hash table. It removes the new entry and changes *
* what is pointing to 'new' to point to 'conn[new].next' and what is pointing *
* to 'old' to point to 'new'.                                                 *
* If new = -1 the entry old must be removed and what was pointing to 'old' must*
* point to 'conn[old].next'.                                                  *
*****************************************************************************/

int new, old;

{
  unsigned int ih;
  unsigned int src, dest;
  int          i;

#ifdef DEBUG
  printf( "beginning of tidy_hash with new : %d old : %d\n", new, old );
#endif

  if ( new != -1 )
  {
     /* calculates the hash code by making an integer of the source and the
        destination addresses, multiplying both areas and both nodes,
        adding them and making the logical and with 1023 */
     dest = ( conn[new].dest_node[1] << 8 ) | conn[new].dest_node[0];
     src = ( conn[new].src_node[1] << 8 ) | conn[new].src_node[0];
     ih = ( ( ( src / 1024 ) * ( dest / 1024 ) ) +
          ( ( src % 1024 ) * ( dest % 1024 ) ) ) & 1023;
```

```
     if ( conn_hash[ih] == new )                /* the hash table points to new */
        conn_hash[ih] = conn[new].next;                 /* changes the pointer */
     else                                /* the hash table doesn't point to new */
     {  i = conn_hash[ih];                           /* finds the pointer to new */
        while ( i != -1 )
              if ( conn[i].next == new )
              {  conn[i].next = conn[new].next;         /* changes the pointer */
                 break;
              }
              else i = conn[i].next;
         ASSERT( i != -1 );
     }
     /* calculates the hash code by making an integer of the source and the
        destination addresses, multiplying both areas and both nodes,
        adding them and making the logical and with 1023 */
     dest = ( conn[old].dest_node[1] << 8 ) | conn[old].dest_node[0];
     src = ( conn[old].src_node[1] << 8 ) | conn[old].src_node[0];
     ih = ( ( ( src / 1024 ) * ( dest / 1024 ) ) +
          ( ( src % 1024 ) * ( dest % 1024 ) ) ) & 1023;
     if ( conn_hash[ih] == old )                /* the hash table points to old */
        conn_hash[ih] = new;                            /* changes the pointer */
     else                                /* the hash table doesn't point to old */
     {  i = conn_hash[ih];                           /* finds the pointer to old */
        while ( i != -1 )
              if ( conn[i].next == old )
              {  conn[i].next = new;                    /* changes the pointer */
                 break;
              }
              else i = conn[i].next;
        ASSERT( i != -1 );
     }
  }
  else
  {
     /* calculates the hash code by making an integer of the source and the
        destination addresses, multiplying both areas and both nodes,
        adding them and making the logical and with 1023 */
     src = ( conn[old].src_node[1] << 8 ) | conn[old].src_node[0];
     dest = ( conn[old].dest_node[1] << 8 ) | conn[old].dest_node[0];
```

```
    ih = ( ( ( src / 1024 ) * ( dest / 1024 ) ) +
         ( ( src % 1024 ) * ( dest % 1024 ) ) ) & 1023;

    if ( conn_hash[ih] == old )              /* the hash table points to old */
       conn_hash[ih] = conn[old].next;               /* changes the pointer */
    else
    { i = conn_hash[ih];                          /* finds the pointer to old */
      while ( i != -1 )
            if ( conn[i].next == old )
            {  conn[i].next = conn[old].next;        /* changes the pointer */
               break;
            }
            else i = conn[i].next;
       ASSERT( i != -1 );
    }
  }
}


void save_data( ncon, nlink )

/*****************************************************************************
* This function saves the data about the link with index 'nlink' of the      *
* connection with index 'ncon'.                                              *
*****************************************************************************/

int           ncon, nlink;

{
  int           res;
  int           i;

  call_record   c_record;

#ifdef DEBUG
  printf( "beginning of save data\n" );
#endif

  if ( fd == -1 )
  {   fd = open( filename, O_WRONLY | O_CREAT, 0644 );
                                        /* opens the first call records' file */
      if ( fd < 0 )            /* if an error occurred while opening the file */
      {  perror( "open" );
```

```
         exit();
      }
  }
  c_record.src_node[0] = conn[ncon].src_node[0];     /* fills in the stucture */
  c_record.src_node[1] = conn[ncon].src_node[1];

  c_record.dest_node[0] = conn[ncon].dest_node[0];
  c_record.dest_node[1] = conn[ncon].dest_node[1];

  c_record.id1 = conn[ncon].links[nlink].id1;
  c_record.id2 = conn[ncon].links[nlink].id2;

  c_record.disc_reason = conn[ncon].links[nlink].disc_reason;

  c_record.src_link = conn[ncon].links[nlink].src_link;

  c_record.dest_link = conn[ncon].links[nlink].dest_link;

  c_record.c_packets = conn[ncon].links[nlink].c_packets;

  for ( i=0; i<MAXKEY; i++ )
      c_record.count[i] = conn[ncon].links[nlink].count[i];

  strcpy( c_record.src_uname, conn[ncon].links[nlink].src_uname );

  strcpy( c_record.dest_uname, conn[ncon].links[nlink].dest_uname );

  c_record.order = conn[ncon].links[nlink].order;

  c_record.begin = conn[ncon].links[nlink].begin;

  c_record.end = conn[ncon].links[nlink].end;

  res = write( fd, &c_record, sizeof( call_record ) );   /* writes the structure
                                                            in the file */
  if ( res != sizeof( call_record ) )              /* if an error occurred during
                                                      the writing */
  {
     perror( "write" );

     exit();
  }

  ASSERT ( res == sizeof( call_record ) );

#ifdef DEBUG
  printf( "size of call_record : %d\n", sizeof( call_record ) );
  printf( "res for c_record : %d\n", res );
#endif

  res = write( fd, "\n", sizeof( char ) );    /* delimitates the call records */

#ifdef DEBUG
  printf( "res for CR : %d\n", res );
#endif
```

```
}


void process_frame( ncon, nlink )

/*********************************************************************
* This function sets the offset according to the presence of the data ack  *
* number and searchs for keywords if the packet isn't a CTERM message.     *
* For the CTERM messages it calls decode_cterm to reconstruct the lines    *
* sent by the user or the host.                                            *
* The retransmitted data are ignored.                                      *
* 'ncon' and 'nlink' are the index of the connection and the link on which *
* the packet was seen.                                                     *
*********************************************************************/

int             ncon, nlink;

{
  unsigned char   daddr[2];
  unsigned char   cb;

  unsigned int    link_seq_nb;                  /* contains the last data segment
                                                   number seen on the link */
  unsigned int    seq_nb;   /* contains the data segment number in the packet */
  unsigned int    length;
  unsigned int    l;

  int             sens;
  int             pos;
  int             start;
  int             end;
  int             i;


  daddr[0] = buffer[9+offset];                          /* destination address */
  daddr[1] = buffer[10+offset];

  if ( daddr[0] == conn[ncon].dest_node[0] &&           /* determines the sens */
       daddr[1] == conn[ncon].dest_node[1] )            /* of the packet */

     sens = 0;

  else sens = 1;

#ifdef DEBUG
  printf( "sens : %d\n", sens );

  printf( "buffer[29+offset] : %02x\n", buffer[29+offset] );
#endif

  cb = buffer[29+offset];               /* contains the most significant byte of
                                           the ack number if the most
                                           significant bit is set to 1 */

  if ( ! ( cb & 0x80 ) )                 /* if the ack number isn't present   */
     offset = offset - 2;                /* ( the most significant bit is set */
                                         /*   to 0 ) */
```

```
  pos = 30 + offset;            /* position of the least significant byte of the
                                   data segment number in the buffer */

#ifdef DEBUG
  printf( "offset : %d pos : %d\n", offset, pos );
#endif

  /* calculates the data segment number (ignore the most significant 4 bits) */
  /* note : bytes are inversed */

  seq_nb = ( ( buffer[pos+1] & 0x0F ) << 8 ) + buffer[pos];

  link_seq_nb = conn[ncon].links[nlink].data_seq[sens];

#ifdef DEBUG
  printf( "seq_nb : %u link_seq_nb : %u\n", seq_nb, link_seq_nb );
#endif

  if ( link_seq_nb != seq_nb )              /* if it isn't retransmitted data */

  {
     conn[ncon].links[nlink].data_seq[sens] = seq_nb;/* updates the data segment
                                                       number of the link */
     start = 32 + offset;     /* index of the beginning of the data to search */

     end = last_pkt_size() - 1;     /* index of the end of the data to search */

     if ( sensitive == false )           /* if the search is case unsensitive */

        uppercase( buffer, start, end );

#ifdef DEBUG
        printf( "buffer[start] : %02x\n", buffer[start] );
#endif

     if ( buffer[start] == 0x09 )       /* first byte of the foundation services
                                           interface protocol message.
                                           0x09 means common data message */
     {  l = 34 + offset;    /* position of the least significant byte containing
                               the length of the first CTERM message */
        while ( l + 2 <= end )          /* while there are CTERM messages in the
                                           packet - l+2 is the position of the
                                           beginning of the CTERM message */

        {  /* calculates the length of the message */
           /* the bytes are inversed */

           length = ( ( buffer[l+1] << 8 ) | buffer[l] );

           l = l + 2;       /* position of the beginning of the CTERM message */

#ifdef DEBUG
```

```
            printf( "length : %u l : %d\n", length, l );
#endif

            decode_cterm( ncon, nlink, l, length );

            l = l + length;          /* goto the least significant byte containing
                                        the length of the next CTERM message */
          }
      }
      else                                  /* we don't have a CTERM message */
      {
        /* if we have an IP packet encapsulated in a Decnet packet and the one
           of the IP address is inside CERN, i.e. 128.141 or 808d in hexa */

        if ( ( buffer[start] == 0x45 ) &&
             ( ( buffer[start+16] == 0x80 && buffer[start+17] == 0x8d ) ||
               ( buffer[start+12] == 0x80 && buffer[start+13] == 0x8d ) ) )

            treatment_ip( ncon, nlink, start );/* searchs for the IP addresses */

        else                                /* we don't have neither an IP packet
                                               neither a CTERM message */
        {

#ifdef DEBUG
            printf( "start : %d end : %d\n", start, end );
            printf( "data :\n" );
            for ( i=start; i<=end; i++ )
            {   if ( isalnum(buffer[i]) ) printf( "%c ", buffer[i] );
                else printf( "0x%02x ", buffer[i] );
            }
            printf( "\n" );
#endif

            keyword( ncon, nlink, start, end );    /* searchs for the keywords */
          }
      }
  }
}


void keyword( ncon, nlink, start, end )

/************************************************************************
* This function searchs for keywords in the buffer.                     *
* This keywords are found in the structure "keywordlist".               *
* 'ncon' and 'nlink' are the index of the conection and the link on which *
* the packet was seen. 'start' and 'end' are the index of the beginning  *
* and the end of the data to search.                                     *
```

```
************************************************************************/

int           ncon, nlink;
int           start, end;

{
  int         count;
  int         i;

  for ( i=0; i<num_of_key; i++ )                        /* for each keyword */
  {   count = find_word( buffer, start, end, keywordlist[i].val,
                         keywordlist[i].length );      /* is the word present in
                                                          the packet ? */
#ifdef DEBUG
      printf( "counter for %s : %d\n", keywordlist[i].val, count );
#endif

      if ( count == 1 )                  /* the word is present in the packet */
      {  conn[ncon].links[nlink].count[i]++;           /* updates the counter */
         save_command( ncon, nlink, '?', keywordlist[i].val );
         break;                  /* if we've found a word, we stop the search */
      }
  }

#ifdef DEBUG

  for ( i=0; i<num_of_key; i++ )
      printf( "conn[ncon].links[nlink].count[%d] : %d\n", i,
              conn[ncon].links[nlink].count[i] );
#endif
}


int find_word( tab, begin, end, word, w_length )

/************************************************************************
* This function looks to see if the word pointed by 'word' is present in *
* 'tab'. The length of the word is 'w_length' and 'begin' and 'end' are  *
* the index that indicates the beginning and the end of the data to search in *
* 'tab'.                                                                 *
************************************************************************/

int           begin, end;
int           w_length;
unsigned char *word;
```

```
unsigned char  *tab;

{
  register unsigned char  *start;
  register unsigned char  *stop;
  register unsigned char  *place;

  start = &tab[begin];          /* it points to the first char to be analysed */
  stop = &tab[end];              /* it points to the last char to be analysed */

  place = memchr( start, *word, end-begin+1 );          /* searchs for the first
                                                           occurrence of the char
                                                           *word */
#ifdef DEBUG
  printf( "place : %d\n", place );
#endif

  if ( place == NULL )                        /* there isn't such an occurrence */
     return( 0 );
  else
  {  while ( place != NULL )
       if ( memcmp( place, word, w_length ) != 0 )        /* compairs the end of
                                                             the word */
       {  place = memchr( place+1, *word, stop-place ); /* if it isn't the same,
                                                           searchs for the next
                                                           occurrence */
#ifdef DEBUG
          printf( "place : %d\n", place );
#endif
       }
       else return( 1 );                     /* if the word is found, returns 1 */
     return( 0 );                                    /* the word isn't found */
  }
}


void decode_cterm( ncon, nlink, begin, length )

/*************************************************************************
* This function decodes the CTERM protocol. It analyses the CTERM messages *
* of type 0x02, 0x03 and 0x07, i.e. start read, read and write messages    *
* to reconstruct the lines typed by the user or sent by the host to the    *
* user and tries to find some keywords in these reconstructed lines by     *
* calling analyse_host and analyse_server.                                 *
* 'begin' contains the index of the beginning of the CTERM message and     *
* 'length' contains the length of the message.                             *
* 'ncon' and 'nlink' are the index of the connection and the link on which *
* the packet has been seen.                                                *
*************************************************************************/
```

```
int           ncon, nlink;

unsigned int  begin;
unsigned int  length;

{
  unsigned char  ctype;

  int            i;

  unsigned int   first;
  unsigned int   last;

  bool           addcc;

  ctype = buffer[begin];                          /* type of the CTERM message */

#ifdef DEBUG
  printf( "beginning of decode_cterm\n" );
  printf( "ctype : %02x\n", ctype );
#endif

  if ( ctype == 0x02 )                      /* start read message,
                                               sent from the host to the server */
  {  first = begin + 17 + buffer[begin+16];/* index of the beginning of the data
                                              the header contains 17 bytes plus
                                              a variable length field which the
                                              length is contained in the 17st
                                              byte of the header */
     last = begin + length - 1;              /* index of the end of the data */
     if ( last >= 1500 ) last = 1499;        /* to respect the buffer length */
     if ( first <= last )     /* fills in the line - we put it in the same table
                                 as the command typed by the user so that we
                                 can reconstruct the all line, so we can find,
                                 for example a username */
        for ( i=first; i<=last; i++ )
           if ( conn[ncon].links[nlink].last_server < MAXSIZE )
           {  conn[ncon].links[nlink].
              server[conn[ncon].links[nlink].last_server] = buffer[i];
              conn[ncon].links[nlink].last_server++;
           }
           else break;

#ifdef DEBUG
     printf( "first : %u last : %d last_server : %d\n", first, last,
             conn[ncon].links[nlink].last_server );
#endif
```

```
    /* if a line is complete, i.e. if the last char is '0x0d' or '0x0a' or
       if we arrived at the end of the table - generally, it won't arrive,
       but it may occur that we fill the all table with this line,
       and we have to analyse it */

    if ( conn[ncon].links[nlink].
         server[conn[ncon].links[nlink].last_server-1] == 0x0d ||
         conn[ncon].links[nlink].
         server[conn[ncon].links[nlink].last_server-1] == 0x0a ||
         conn[ncon].links[nlink].last_server == MAXSIZE )
    {

#ifdef DEBUG

        for ( i=0; i<conn[ncon].links[nlink].last_server; i++ )

            printf( "%c", conn[ncon].links[nlink].server[i]);

        printf( "test of the CR\n" );

#endif


        analyse_server( ncon, nlink );

        conn[ncon].links[nlink].last_server = 0;          /* resets the counter
                                                            for the new line */

        for ( i=0; i<MAXSIZE; i++ )                /* to avoid having extra char
                                                      the next time */
            conn[ncon].links[nlink].server[i] = ' ';
    }
  }

  else if ( ctype == 0x03 )                /* read data message,
                                              sent from the server to the host */
      {

#ifdef DEBUG

        printf( "buffer[begin+1] : %02x\n", buffer[begin+1] );

#endif


        /* if the completion code == termination char or valid escape sequence
           or timeout
           else we ignore the message */

        if ( buffer[begin+1] == 0x00 || buffer[begin+1] == 0x01 ||
             buffer[begin+1] == 0x05 )

        {  first = begin + 8; /* index of the beginning of the data -
                                 the header of the message contains 8 bytes */
           last = begin + length - 1;       /* index of the end of the data */

           if ( last >= 1500 ) last = 1499; /* to respect the buffer length */

           if ( first <= last )     /* fills in the line containing the command
                                       typed by the user */
              for ( i=first; i<=last; i++ )

                  if ( conn[ncon].links[nlink].last_server < MAXSIZE )

                  {  conn[ncon].links[nlink].
```

```
                     server[conn[ncon].links[nlink].last_server] = buffer[i];

                     conn[ncon].links[nlink].last_server++;
                  }

                  else break;


#ifdef DEBUG

            printf( "first : %d last : %d last_server : %d\n", first, last,
                    conn[ncon].links[nlink].last_server );

#endif


            /* if the line is complete, i.e. if the last char is '0x0d' or
               '0x0a' or if we arrived at the end of the table */

            if ( conn[ncon].links[nlink].
                 server[conn[ncon].links[nlink].last_server-1] == 0x0d ||
                 conn[ncon].links[nlink].
                 server[conn[ncon].links[nlink].last_server-1] == 0x0a ||
                 conn[ncon].links[nlink].last_server == MAXSIZE )

            {

#ifdef DEBUG

               for ( i=0; i<conn[ncon].links[nlink].last_server; i++ )

                   printf( "%c", conn[ncon].links[nlink].server[i] );

               printf( "test of the CR\n" );

#endif


               analyse_server( ncon, nlink );

               conn[ncon].links[nlink].last_server = 0;  /* resets the counter
                                                           for the new line */
               for ( i=0; i<MAXSIZE; i++ )         /* to avoid having extra char
                                                      the next time */
                   conn[ncon].links[nlink].server[i] = ' ';
            }
         }
       }

       else if ( ctype == 0x07 )           /* write message,
                                              sent from the host to the server */
            {  addcc = false;

               if ( buffer[begin+4] == 0x0d )       /* we have a complete line -
                                                       we examine the postfix
                                                       value */

               {  addcc = true;

                  if ( conn[ncon].links[nlink].last_host != 0 )/*if we have the
                                                                 beginning of
                                                                 another line
                                                                 in the table */
                  {  analyse_host( ncon, nlink );       /* we first analyse this
                                                           line */
```

```
                    conn[ncon].links[nlink].last_host = 0;    /* we reset the
                                                                 counter for
                                                                 the new line */
                    for ( i=0; i<MAXSIZE; i++ )    /* to avoid having extra char
                                                      the next time */
                        conn[ncon].links[nlink].host[i] = ' ';
                }
            }

            first = begin + 5;      /* index of the beginning of the message -
                                       the header of the message contains
                                       5 bytes */
            last = begin + length - 1;      /* index of the end of the data */

            if ( last >= 1500 ) last = 1499;          /* to respect the buffer
                                                         length */
            if ( first <= last )      /* fills in the line sent by the host */

               for ( i=first; i<=last; i++ )

                   if ( conn[ncon].links[nlink].last_host < MAXSIZE )

                   {  conn[ncon].links[nlink].
                      host[conn[ncon].links[nlink].last_host] = buffer[i];

                      conn[ncon].links[nlink].last_host++;
                   }

                   else break;

#ifdef DEBUG

            printf( "buffer[begin+1] : %02x\n", buffer[begin+1] );

            printf ( "buffer[begin+4] : %02x\n", buffer[begin+4] );

            printf( "addcc : %d\n", addcc );

            printf( "first : %d last : %d last_host : %d\n", first, last,
                    conn[ncon].links[nlink].last_host );

#endif


            /* if we have a complete line, i.e. if the last char is '0x0d'
               or '0x0a' or addcc==true or we arrived at the end of the
               table */

            if ( conn[ncon].links[nlink].host[conn[ncon].links[nlink].
                 last_host-1] == 0x0d ||
                 conn[ncon].links[nlink].
                 host[conn[ncon].links[nlink].last_host-1] == 0x0a ||
                 addcc == true ||
                 conn[ncon].links[nlink].last_host == MAXSIZE )
            {

#ifdef DEBUG

                for ( i=0; i<conn[ncon].links[nlink].last_host; i++ )

                    printf( "%c", conn[ncon].links[nlink].host[i] );

                printf( "test of the CR" );

#endif
```

```
                    analyse_host( ncon, nlink );

                    conn[ncon].links[nlink].last_host = 0;/* resets the counter
                                                            for the new line */
                    for ( i=0; i<MAXSIZE; i++ )     /* to avoid having extra char
                                                       the next time */
                        conn[ncon].links[nlink].host[i] = ' ';
                }
            }

            else return;          /* not an interesting CTERM function for us */
}


void analyse_server( ncon, nlink )

/*************************************************************************
* This function analyses a line sent from the server to the host on the  *
* link identified by 'ncon, nlink'. If it finds a keyword in this line, the *
* line is saved in a file by calling save_command.                       *
*************************************************************************/

int  ncon, nlink;

{
  int  i;
  int  count;

  for ( i=0; i<num_of_key; i++ )                          /* for each keyword */

  {   count = find_word( conn[ncon].links[nlink].server, 0,
            conn[ncon].links[nlink].last_server-1, keywordlist[i].val,
            keywordlist[i].length );   /* is the word present in the line ? */

      if ( count == 1 )                    /* the word is present in the line */

      {  conn[ncon].links[nlink].count[i]++;           /* updates the counter */

         save_command( ncon, nlink, 's' , conn[ncon].links[nlink].server );

         break;                  /* if we've found a word, we stop the search */
      }
  }

  /* we need to record this command typed by the user in order to
     eventually save it with a suspicious answer of the host which will
     be analysed later */

  for ( i=0; i<MAXSIZE; i++ )

      conn[ncon].links[nlink].old_server[i] = conn[ncon].links[nlink].server[i];
}


void analyse_host( ncon, nlink )
```

```
/*************************************************************************
* This function analyses a line sent from the host to the server on the  *
* link identified by 'ncon, nlink'. If it finds a keyword in this line, this *
* line and the one sent from the server to the host before are saved     *
* in a file by calling save_command.                                     *
*************************************************************************/

int  ncon, nlink;

{
  int  i;
  int  count;

  for ( i=0; i<num_of_key; i++ )                          /* for each keyword */

  {   count = find_word( conn[ncon].links[nlink].host, 0,
              conn[ncon].links[nlink].last_host-1, keywordlist[i].val,
              keywordlist[i].length );   /* is the word present in the line ? */

      if ( count == 1 )                    /* the word is present in the line */

      {  conn[ncon].links[nlink].count[i]++;          /* updates the counter */

         save_command( ncon, nlink, 'o' ,conn[ncon].links[nlink].old_server );

         save_command( ncon, nlink, 'h', conn[ncon].links[nlink].host );

         break;
      }
  }

  for ( i=0; i<MAXSIZE; i++ )

      conn[ncon].links[nlink].old_server[i] = ' ';        /* to avoid having
                                                            extra char the next
                                                            time */
}


void save_command( ncon, nlink, origin, cmd )

/*************************************************************************
* This function saves the command 'cmd' together with the identifier of  *
* the link to which this line owns, i.e. the link now identified by      *
* 'ncon, nlink'.                                                         *
* 'origin' specifies who has sent the command : 'h' means host, 's'      *
* means server, 'o' means server with the line triggering a suspicious   *
* answer of the host, '?' means that it is not a command seen on a CTERM *
* connections and 'i' means that it is the IP addresses seen in an IP    *
* packet encapsulated in a Decnet packet.                                *
*************************************************************************/

int            ncon, nlink;

unsigned char  *cmd;

char           origin;
```

```
{
  int      res;
  int      i;

  command  line;

#ifdef DEBUG
  printf( "beginning of save command\n" );
#endif

  if ( fdcmd == -1 )
  {  fdcmd = open( filecommand, O_WRONLY | O_CREAT, 0664 );
                                          /* opens the first commands' file */
     if ( fdcmd < 0 )          /* if an error occurred while opening the file */

     {  perror( "open" );

        exit();
     }
  }

  line.id_link1 = conn[ncon].links[nlink].id1;      /* fills in the structure */

  line.id_link2 = conn[ncon].links[nlink].id2;

  line.origin = origin;

  for ( i=0; i<MAXSIZE; i++ )

      line.cmd[i] = cmd[i];

  line.moment = time ( 0 );

  res = write( fdcmd, &line, sizeof( command ) );       /* writes the command in
                                                           the file */
  if ( res != sizeof( command ) )  /* if an error occurred during the writing */

  {
     perror( "write" );

     exit();
  }

  ASSERT( res == sizeof( command ) );

#ifdef DEBUG
  printf( "res for the line : %d size of a cmd : %d\n", res, sizeof( command ));
#endif

  res = write( fdcmd, "\n", sizeof( char ) );     /* delimitates the commands */

}
```

```
void treatment_ip( ncon, nlink, start )

/*****************************************************************************
* This function sees if we have already saves the IP addresses contained in *
* the IP packets of the link identified by 'ncon, nlink'. If it is not the  *
* case, saves the addresses in the commands' file by calling save_command.  *
* 'start' is the index of the beginning of the IP packet.                   *
*****************************************************************************/


int     ncon, nlink;
int     start;


{  unsigned char     cmd[MAXSIZE];

   int               i;


   if ( conn[ncon].links[nlink].ip == false )       /* if we don't have saved the
                                                       addresses yet */
   {  conn[ncon].links[nlink].ip = true;

      cmd[0] = 'I';                                 /* fills in the command's line */
      cmd[1] = 'P';
      cmd[2] = ' ';
      cmd[3] = ':';
      cmd[4] = ' ';
      cmd[5] = buffer[start+12];
      cmd[6] = buffer[start+13];
      cmd[7] = buffer[start+14];
      cmd[8] = buffer[start+15];
      cmd[9]=' ';
      cmd[10]=' ';
      cmd[11]= buffer[start+16];
      cmd[12]= buffer[start+17];
      cmd[13]= buffer[start+18];
      cmd[14]= buffer[start+19];

      for ( i=15; i<MAXSIZE; i++ )

          cmd[i] = ' ';

      save_command( ncon, nlink, 'i', cmd );
   }
}
```

Le programme de traitement

```
/*****************************************************************
* This file contains all the name and the result's type of the functions *
* used in the program treatment.c.                                *
*****************************************************************/

void init_keywordlist();

void init_rules();

void uppercase( unsigned char *, int, int );

int find_word( unsigned char *, unsigned char *, unsigned char *, int,
               unsigned char ** );

void treatment_of_rule( unsigned char *, int, int, int );

void treatment_links( int, char **  );

int search_counter( int, call_record );

bool apply_rule( call_record, int, command *, int );

bool treatment_moment( command *, int, call_record, int );

void display( call_record, int, command * );

void make_decaddress( unsigned char *, unsigned int *, unsigned int * );

void treatment_of_counter( int * );
```

```
#undef DEBUG

#include <file.h>
#include <time.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stat.h>
#include <unixio.h>

#include "data_struct.h"
#include "treatment.h"

int             num_of_key;
bool            all = false;        /* by default, we don't print all the links */
bool            defil = false;    /* by default, we present the information screen
                                     by screen */
key             keywordlist[MAXKEY+1];
condition       rule[MAXKEY];


int main( argc, argv )

/*****************************************************************************
* This program prints the information about the links according to the rules *
* contained in the file "keyword.txt".                                       *
*****************************************************************************/

int   argc;
char  *argv[];

{
  init_keywordlist();
  init_rules();
  treatment_links( argc, argv );
  exit( 0 );
}


void init_keywordlist()
```

```
/*****************************************************************
* This function initializes the list of the keywords to search. *
* This list is in the file "keyword.txt".                        *
*****************************************************************/

{
  FILE  *fkey;
  FILE  *flog;
  int   idx, len;
  int   i;

#ifdef DEBUG
  printf( "beginning of init_keywordlist\n" );
#endif

  fkey = fopen( "keyword.txt", "r" );
  idx = 0;
  while ( fgets( keywordlist[idx].val, MAXSIZE, fkey ) )      /* gets the word */
  {
       len = strlen( keywordlist[idx].val );      /* gets the length ( including
                                                     '\n' if present ) */
       if ( keywordlist[idx].val[0] == '!' )                  /* it's a comment */
       {  if ( keywordlist[idx].val[len-1] != '\n' )/* the line has been cut */
            while ( fgetc( fkey ) != '\n' );  /* skips the end of the line */
          continue;                                      /* skips the comments */
       }
       if ( keywordlist[idx].val[0] == 'R' && keywordlist[idx].val[1] == ' ' )
                                                             /* it's a rule */
       {  if ( keywordlist[idx].val[len-1] != '\n' )/* the line has been cut */
            while ( fgetc( fkey ) != '\n' );  /* skips the end of the line */
          continue;                                       /* skips the rules */
       }
       if ( keywordlist[idx].val[len-1] != '\n' )    /* the word has been cut */
       {
          while ( fgetc( fkey ) != '\n' );       /* skips the end of the word */
          len--;                        /* discards the last char from the length
                                           because MAXSIZE includes a place for
                                           '\n' and '\n' isn't present here */

          if ( len > 0 && keywordlist[idx].val[len-1] == ' ' ) /* if there are
                                                                  blanks after
                                                                  the word,
                                                                  skips them */
          {  for ( i=len-1; keywordlist[idx].val[i]==' ' && i>=0; i-- ) ;
```

```
            len = i + 1;
         }
         if ( len == 0 ) continue;                    /* skips empty lines */
         flog = fopen( "logtrt.log", "a");
         fprintf(flog,"the word number %d in the keyword file has been cut\n",
                 idx+1 );                    /* signals the error in logfile */
         fclose( flog );
      }
      else                                        /* the word isn't too long */
      {  keywordlist[idx].val[len-1] = '\0';       /* replaces '\n' by '\0' */
         len--;                          /* discards the '\0' from the length */
         if ( len > 0 && keywordlist[idx].val[len-1] == ' ' ) /* if there are
                                                                 blanks after
                                                                 the words,
                                                                 skips them */
         {  for ( i=len-1; keywordlist[idx].val[i]==' ' && i>=0; i-- ) ;
            len = i + 1;
         }
         if ( len == 0 ) continue;                    /* skips empty lines */
      }
      if ( idx >= MAXKEY )       /* there are too much keywords in the file */
                                 /* put here because we must first treat the
                                    comment's lines, thr rules' lines and the
                                    empty lines */
      {  flog = fopen( "logtrt.log", "a" ); /* signals the error in logfile */
         fprintf( flog, "there are more than %d keywords\n", MAXKEY );
         fclose( flog );
         break;                                       /* stops reading words */
      }
      keywordlist[idx].length = len;
      for ( i=len; i<MAXSIZE; i++ )       /* discards garbbage from the end of
                                             the line */
          keywordlist[idx].val[i] = ' ';
      uppercase( keywordlist[idx].val, 0, len - 1 ); /* capitalizes all the
                                                        letters of the word */
      idx++;
  }
  fclose( fkey );
  num_of_key = idx;
  if ( num_of_key == 0 )     /* if the file is empty we exit from the program */
  {  printf( "No keyword to search for. Verify the file keyword.txt.\n" );
     exit( 0 );
```

```
  }
#ifdef DEBUG
  printf( "num_of_key : %d\n", num_of_key );
  for ( i=0; i<num_of_key; i++ )
      printf( "keyword : %s length : %d\n", keywordlist[i].val,
              keywordlist[i].length );
#endif
}


void uppercase( word, begin, end )

/*****************************************************************
* This function capitalizes all lowercase letters of 'word'.     *
* 'begin' is the index of the first letter of the word and 'end'  *
* is the index of the last one.                                   *
*****************************************************************/

unsigned char   *word;
int             begin, end;

{
  int  i;

#ifdef DEBUG
  printf( "beginning of uppercase\n" );
#endif

  for ( i=begin; i<=end; i++ )
     if ( 'a' <= word[i] && word[i] <= 'z' )
        word[i] = word[i] + 'A' - 'a';

#ifdef DEBUG
  printf( "end of uppercase\n" );
#endif
}


void init_rules()
```

```
/*****************************************************************************
* This function reads the rules in the file "keyword.txt" and initializes    *
* the structure of the rules to apply while reading the call records' file.  *
* The rule of index "j" applies to the word of index "i" in the structure    *
* 'keywordlist'. In each rule, the first member is the minimum number of     *
* times we must have seen the word to print the line, the second member is    *
* the period of time during which the occurrences of the word must have been *
* seen ( 0 means that there is no restriction about the period of time )     *
* and the last member is the index of the word concerned by the second part  *
* of the rule if we have a rule with an "AND" ( -1 means that there isn't    *
* any second part ). Note that a rule with an "OR" may be expressed as two   *
* separated rules.                                                           *
*****************************************************************************/

{
  FILE            *frule;

  int             i, j;
  int             count;
  int             nb_rule;
  int             len;

  unsigned char   line[MAXSIZE];
  unsigned char   *pos;
  unsigned char   *p;
  unsigned char   *rule1;
  unsigned char   *rule2;
  unsigned char   *word;

#ifdef DEBUG
  printf( "beginning of init_rules\n" );
#endif

  nb_rule = 0;
  for( i=0; i<MAXKEY; i++)                              /* init of the structure */
  {  rule[i].count = -1;
     rule[i].moment = -1;
     rule[i].next = -1;
  }
  frule = fopen( "keyword.txt", "r" );
  while ( fgets( line, MAXSIZE, frule ) )                       /* gets a line */
  {  len = strlen( line );                          /* gets the length of the line */
     if ( ( line[0] != 'R' ) || ( line[1] != ' ' ) )
     {
        if ( line[len-1] != '\n' )                      /* the line has been cut */
           while ( fgetc( frule ) != '\n' ) ;     /* skips the end of the line */
        continue;                  /* skips the lines which don't contain rules */
     }
```

```
     else
     {
        nb_rule++;                                      /* there is one more rule */
        if ( line[len-1] != '\n' )                       /* the line has been cut */
        {   printf( "the rule number %d is too long\n", nb_rule );
            exit( 0 );
        }
        uppercase( line, 0, len-1 );                      /* capitalizes all the
                                                             letters of the line */
        count = find_word( line, &line[len-1], " OR ", 4, &pos );
                                          /* does the rule contain an 'OR' ? */

#ifdef DEBUG
        printf( "count for OR : %d\n", count );
        printf( "pos : %d\n", pos );
#endif

        if ( count == 1 )                           /* the rule contains an 'OR' */
        {  rule1 = line;            /* beginning of the first part of the rule */
           rule2 = pos + 3;         /* beginning of the second part of the rule */

#ifdef DEBUG
           printf( "rule1: %d rule2 : %d\n", rule1, rule2 );
#endif

           /* we look for other 'AND' or 'OR' in the first part of the rule.
              in the first part of the rule we only have to search for an 'AND'
              because the first call of find_word found the first occurrence of
              'OR' */

           count = find_word( rule1, pos, " AND ", 5, &p );

#ifdef DEBUG
           printf( "count for an AND : %d\n", count );
#endif

           if ( count == 1 )             /* there is also an 'AND' in the rule */
           {  printf( "there are an 'OR' and an 'AND' in the rule number %d\n",
                      nb_rule );
              exit( 0 );
           }
           for ( i=0; i<num_of_key; i++ )  /* treats the 1st part of the rule */
```

```
        {   count = find_word( rule1, pos, keywordlist[i].val,
                               keywordlist[i].length, &word ); /* finds the
                                                                  keyword of
                                                                  the rule */
            if ( count == 1 )
            {  treatment_of_rule( rule1, rule2-rule1, i, nb_rule );
               rule[i].next = -1;                  /* the 2 parts of the rule
                                                      can be separated */
               break;
            }
        }
        if ( i == num_of_key )    /* we don't find any keyword in the rule */
        {  printf( "the word in the 1st part of rule %d isn't a keyword\n",
                   nb_rule );
           exit( 0 );
        }
        /* we look for other 'OR' or 'AND' in the second part of the rule */
        count = find_word( rule2, &line[len-1], " OR ", 4, &p );

#ifdef DEBUG
        printf( "count for another OR : %d\n", count );
#endif

        if ( count == 1 )             /* there is another 'OR' in the rule */
        {  printf( "there are more than one 'OR' in the rule number %d\n",
                   nb_rule );
           exit( 0 );
        }
        count = find_word( rule2, &line[len-1], " AND ", 5, &p );

#ifdef DEBUG
        printf( "count for an AND : %d\n", count );
#endif

        if ( count == 1 )            /* there is also an 'AND' in the rule */
        {  printf( "there are an 'OR' and an 'AND' in the rule number %d\n",
                   nb_rule );
           exit( 0 );
        }
        for ( i=0; i<num_of_key; i++ )  /* treats the 2nd part of the rule */
        {   count = find_word( rule2, &line[len-1], keywordlist[i].val,
                               keywordlist[i].length, &word ); /* finds the
                                                                  keyword of
                                                                  the rule */
```

```
            if ( count == 1 )
            {  treatment_of_rule( rule2, &line[len-1]-rule2+1, i, nb_rule );
               rule[i].next = -1;              /* the 2 parts of the rule can
                                                  be separated */
               break;
            }
        }
        if ( i == num_of_key )    /* we don't find any keyword in the rule */
        {  printf( "the word in the 2nd part of rule %d isn't a keyword\n",
                   nb_rule );
           exit( 0 );
        }
     }
     else
     {  count = find_word( line, &line[len-1], " AND ", 5, &pos );
                                  /* does the rule contain an 'AND' ? */

#ifdef DEBUG
        printf( "count for AND : %d\n", count );
        printf( "pos : %d\n", pos );
#endif

        if ( count == 1 )                    /* the rule contains an 'AND' */
        {  rule1 = line;        /* beginning of the first part of the rule */
           rule2 = pos + 4;     /* beginning of the second part of the rule */

#ifdef DEBUG
           printf( "rule1 : %d rule2 : %d\n", rule1, rule2 );
#endif

           /* we look for other 'AND' or 'OR' in the rule.
              in the first part of the rule we only have to search for an 'OR'
              because the first call of find_word found the first occurrence
              of 'AND' */
           count = find_word( rule1, pos, " OR ", 4, &p );

#ifdef DEBUG
           printf( "count for an OR : %d\n", count );
#endif

           if ( count == 1 )           /* there is also an 'OR' in the rule */
```

```
            {  printf( "there are 'OR' and 'AND' in the rule number %d\n",
                       nb_rule );
               exit( 0 );
            }
            for ( i=0; i<num_of_key; i++ )              /* treats the 1st part
                                                           of the rule */
            {   count = find_word( rule1, pos, keywordlist[i].val,
                                   keywordlist[i].length, &word );/* finds the
                                                                     keyword
                                                                     of the
                                                                     rule */
               if ( count == 1 )
               {  treatment_of_rule( rule1, rule2-rule1, i, nb_rule );
                  break;
               }
            }
            if ( i == num_of_key ) /* we don't find any keyword in the rule */
            {  printf( "the word in the 1 part of rule %d isn't a keyword\n",
                       nb_rule );
               exit( 0 );
            }
            /* we look for other 'OR' and 'AND' in the second part of the
               rule */
            count = find_word( rule2, &line[len-1], " AND ", 5, &p );

#ifdef DEBUG
            printf( "count for another AND : %d\n", count );
#endif

            if ( count == 1 )          /* there is another 'AND' in the rule */
            {  printf("there are more than one 'AND' in the rule number %d\n",
                       nb_rule );
               exit( 0 );
            }
            count = find_word( rule2, &line[len-1], " OR ", 4, &p );

#ifdef DEBUG
            printf( "count for an OR : %d\n", count );
#endif

            if ( count == 1 )           /* there is also an 'OR' in the rule */
            {  printf( "there are 'OR' and 'AND' in the rule number %d\n",
                       nb_rule );
```

```
               exit( 0 );
            }
            for ( j=0; j<num_of_key; j++ )              /* treats the 2nd part
                                                           of the rule */
            {   count = find_word( rule2, &line[len-1],
                                   keywordlist[j].val, keywordlist[j].length,
                                   &word );                /* finds the keyword
                                                              of the rule */
               if ( count == 1 )
               {  treatment_of_rule( rule2, &line[len-1]-rule2+1, j,
                                     nb_rule );
                  break;
               }
            }
            if ( j == num_of_key ) /* we don't find any keyword in the file */
            {  printf( "the word in the 2 part of rule %d isn't a keyword\n",
                       nb_rule );
               exit( 0 );
            }
            rule[i].next = j;             /* links the two parts of the rule */
            rule[j].next = i;
         }
         else                /* there isn't an 'OR' or an 'AND' in the rule */
         {  for ( i=0; i<num_of_key; i++ )
            {   count = find_word( line, &line[len-1], keywordlist[i].val,
                                   keywordlist[i].length, &word );/* finds the
                                                                     keyword
                                                                     of the
                                                                     rule */
               if ( count == 1 )
               {  treatment_of_rule( line, len, i, nb_rule );
                  rule[i].next = -1;           /* there isn't any second part
                                                  of the rule */
                  break;
               }
            }
            if ( i == num_of_key ) /* we don't find any keyword in the rule */
            {  printf( "the word in the rule %d isn't one of the keywords\n",
                       nb_rule );
               exit( 0 );
            }
         }
      }
   }
}
fclose( frule );
if ( nb_rule == 0 )                        /* if there is no rule in the file,
                                              we print all the links */
```

```
    all = true;

#ifdef DEBUG

   for ( i=0; i<num_of_key; i++ )

      printf( "rule : %d %d %d\n", rule[i].count,
              rule[i].moment, rule[i].next );

#endif

}



int find_word( start, end, word, w_length, pos )

/*****************************************************************
* This function looks to see if the word pointed by 'word' is present   *
* between the character pointed by 'start' and the one pointed by 'end'.*
* If so, the position of the first character of the word is put into    *
* '*pos'. 'w_length' is the length of the word.                         *
*****************************************************************/

unsigned char  *start;
unsigned char  *end;
unsigned char  *word;
unsigned char  **pos;

int            w_length;

{
  unsigned char  *place;

#ifdef DEBUG

  printf( "beginning of find_word\n" );

#endif

  place = memchr( start, *word, end-start+1);/* searchs for the first occurrence
                                                of the char *word */

#ifdef DEBUG

  printf( "place : %d\n", place );

#endif

  if ( place == NULL )                      /* there isn't such an occurrence */

     return( 0 );

  else

  {  while ( place != NULL )

     if ( memcmp( place, word, w_length ) != 0 )          /* compairs the end of
```

```
                                                          the word */
     {  place = memchr( place+1, *word, end-place );    /* if it isn't the same,
                                                          searchs for the next
                                                          occurrence */

#ifdef DEBUG

        printf( "place : %d\n", place );

#endif

     }

     else

     {
        *pos = place;

        return( 1 );                                /* the word has been found */
     }

     return( 0 );                                  /* the word hasn't been found */
  }
}


void treatment_of_rule ( line, length, num, nb_rule )

/*****************************************************************
* This function treats the rule contained in 'line' and puts it in the  *
* struture 'rule'. 'length' is the length of the line.                  *
* 'num' is the index of the word found in the rule.                     *
* 'nb_rule' is the number of the ryule treated.                         *
* This is organized as follow : for the word of index i, we find in     *
* the structure at the place 'i' the minimum number of times we must have *
* seen the word, the interval of time between the occurrences of the word *
* and the index of the other word which appears in the second part of the *
* rule if we have a rule with an "AND". Note that a rule with an "OR" may *
* be expressed as two different rules.                                  *
*****************************************************************/

unsigned char  *line;

int            num;
int            length;

{
  unsigned char  *pos;

#ifdef DEBUG

  printf( "beginning of treatment of rule\n" );

#endif

  if ( rule[num].count != -1 )          /* the word already appears in a rule */

  {  printf( "the word number %d occurs in more than one rule\n", num+1 );
```

```
      exit( 0 );
   }
   pos = memchr( line, '(', length );              /* searchs for the '('.
                                                      After the '(' we eventually
                                                      find the interval of time */
   if ( pos != NULL )
   {  pos++;                                      /* points to the char after '(' */
      rule[num].moment = atoi( pos );            /* converts the char into an int */
      if ( rule[num].moment < 0 )     /* the interval of time is smaller than 0 */
      {  printf( "the time's restriction in the rule %d is smaller than 0\n",
                 nb_rule );
         exit( 0 );
      }
   }
   else rule[num].moment = 0;                      /* there is no '(' in the rule */
   pos = memchr( line, '>', length );          /* searchs for the '>'.
                                                  After '>' we find the '=' and then
                                                  the counter for  the word */
   if ( pos != NULL )
   {  pos++;
      if ( *pos == '=' )                       /* if the '>' is followed by an '=' */
      {  pos++;
         rule[num].count = atoi( pos );         /* converts the char into an int */
         if ( rule[num].count <= 0 )                      /* the counter is <= 0 */
         {  printf( "the counter of the rule %d is smaller or equal to zero\n",
                    nb_rule );
            exit( 0 );
         }
      }
      else                                    /* the '>' is not followed by an '=' */
      {  printf( "missing '=' in the rule number %d\n", nb_rule );
         exit( 0 );
      }
   }
   else                                         /* there is no '>' in the rule */
   {  printf( "missing '>' in the rule number %d\n", nb_rule );
      exit( 0 );
   }
#ifdef DEBUG
   printf( "rule : %d %d %d\n", rule[num].count, rule[num].moment );
#endif
```

```
}


void treatment_links( argc, argv )

/*************************************************************************
* This function reads the call records' file, the commands' file and prints *
* the lines which must be printed according to the rules.                   *
* 'argc' contains the number of arguments passed when calling the program   *
* 'argv' contains the list of these arguments ( one of these arguments      *
* should be the name of the call records' file we want to treat.            *
*************************************************************************/

int   argc;
char  *argv[];

{
  int             fd, fdcmd;
  int             res, ok;
  int             i;
  int             maxindex;

  char            *usage = "Usage : treatment [-a][-d] filename\n";
  char            filename[128];
  char            ret;

  bool            impression;

  call_record     link;

  command         *ptr;
  command         *file;

  struct stat     buf;

  for ( i=1;  ( ( i<argc ) && ( *argv[i] == '-' ) ); ++i )   /* treatment of the
                                                                parameters */
  {   switch( argv[i][1] )
      {  case 'a' : all = true;              /* we have to print all the links */
                    break;
         case 'd' : defil = true;  /* we don't have to stop after each screen */
                    break;
         default  : printf ( usage );      /* there is a mistake in the syntax */
                    exit( 0 );
      }
  }
  if ( i<argc )
  {  fd = open ( argv[i], O_RDONLY, 0644 );                 /* opens the file */
     if ( fd <= 0 )                        /* there is a problem with the file */
```

```
    {  printf( "\nVerify the name of the file\n\n" );
       exit( 0 );
    }
    else
       /* builds the name of the commands' file */
       sprintf( filename, "command%s", strrchr( argv[i], '_' ) );

#ifdef DEBUG
    printf( "name of the commands' file : %s\n", filename );
#endif
  }
  else                    /* there is a mistake in the syntax of the command */
  {  printf( usage );
     exit( 0 );
  }

#ifdef DEBUG
  printf( "fd : %d\n", fd );
#endif

  /* puts the commands' file in memory */
  fdcmd = open( filename, O_RDONLY, 0644 );

#ifdef DEBUG
  printf( "fdcmd : %d\n", fdcmd );
#endif

  if ( fdcmd <= 0 )             /* there is a problem with the commands' file */
  {  printf( "\nWarning : there is no commands file\n\n" );
     maxindex = 0;                                    /* there is no command */
  }
  else                         /* there is no problem with the commands' file */
  {  res = fstat( fdcmd, &buf );          /* obtains the information concerning
                                             the file */

#ifdef DEBUG
     printf( "res for fstat : %d buf.st_size : %d\n", res, buf.st_size );
#endif
```

```
     file = ( command * ) malloc( buf.st_size + 1 );    /* reserves the space in
                                                           memory */
     if ( file == NULL )                          /* there is not enough memory */
     {  printf( "There is a not enough memory, launch the program again\n" );
        close( fd );                        /* closes the files before exiting */
        close( fdcmd );
        exit( 0 );
     }
#ifdef DEBUG
     printf( "file : %d\n", file );
#endif

     /* number of structures in the file */
     maxindex = ( buf.st_size ) / ( sizeof( command ) + sizeof( char ) );

#ifdef DEBUG
     printf( "sizeof( command ) : %d maxindex : %d\n", sizeof( command ),
             maxindex );
#endif

     ptr = file;

#ifdef DEBUG
     printf( "ptr : %d\n", ptr );
#endif

     /* reads the commands and puts them in the structure in memory */
     while ( ( res = read( fdcmd, ptr, sizeof( command ) ) ) != 0 )
     {  ptr++;
        ok = read( fdcmd, &ret, sizeof( char ) ); /* reads the CR separating the
                                                     the commands */
        if ( ret != '\n' )
        {  printf( "\nWarning : " );
           printf( "problem with the reading of the commands file\n\n" );
           break;
        }

#ifdef DEBUG
        printf( "res : %d\n", res );
```

```
#endif
     }
#ifdef DEBUG
     printf( "res : %d\n", res );
#endif

     close( fdcmd );

#ifdef DEBUG
     ptr = file;
     for ( i=0; i<maxindex; i++ )
     {   printf( "id_link2 : %u origin : %c\n", ptr->id_link2, ptr->origin );
         ptr++;
     }
#endif
  }
  while ( ( res = read( fd, &link, sizeof( call_record ) ) ) != 0 )
  {
     ok = read( fd, &ret, sizeof( char ) );       /* reads the CR separating the
                                                     call records */
     if ( ret != '\n' )   /* there is a problem with the reading of the file */
     {  printf( "problem with the reading of the call records file\n" );
        close( fd );
        close( fdcmd );
        exit( 0 );
     }
     impression = false;
     if ( all == true )                    /* we have to print all the links */
        impression = true;
     else
     {  i = search_counter( 0, link ); /* searchs for the first counter != 0 */
        while ( i < MAXKEY )                /* while there is a counter != 0 */
        {
#ifdef DEBUG
           printf( "counter != 0 : %d\n", i );
#endif
```

```
           impression = false;
           if ( rule[i].count != -1 )   /* there is a rule for this counter */
              impression = apply_rule( link, i, file, maxindex );
#ifdef DEBUG
           printf( "impression : %d\n", impression );
#endif

           if ( impression == false )    /* if there is no rule or if the rule
                                            doesn't say to print the link */
              i = search_counter( i+1, link );
                                     /* searchs for the next counter != 0 */
           else break;
        }
     }
     if ( impression == true )
        display( link, maxindex, file );
  }
  close( fd );
}


int search_counter( start, link )

/*********************************************************************
* This function searchs for the first counter different from 0 for the *
* link 'link' starting from the counter with index 'start'.            *
* It returns the index of this counter or MAXKEY if there isn't any    *
* counter different from 0.                                            *
*********************************************************************/

int           start;
call_record   link;

{  int   i;

   for ( i=start; i<MAXKEY; i++ )
      if ( link.count[i] > 0 )
         return( i );              /* returns the index of the counter != 0 */
   return( MAXKEY );                        /* there isn't any counter != 0 */
}
```

```
bool apply_rule( link, nb_rule , file, maxindex )

/*****************************************************************
* This function applies the rule number 'nb_rule' to the link 'link'     *
* and returns true if the line must be printed or false if the line      *
* must not be printed.                                                   *
* 'file' points to the first command of the commands' file and 'maxindex'*
* is the number of commands in the file.                                 *
*****************************************************************/

call_record   link;

command       *file;

int           nb_rule;
int           maxindex;

{  int   next;
   bool  ok;

#ifdef DEBUG
   printf( "beginning of apply_rule\n" );
#endif

   if ( link.count[nb_rule] >= rule[nb_rule].count )     /* according to one part
                                                            of the rule, (count)
                                                            the line must be
                                                            printed */
   {  if ( rule[nb_rule].moment > 0 )      /* if there is a restriction about the
                                              moment at which the words passed */
         ok = treatment_moment( file, maxindex, link, nb_rule );
      else ok = true;    /* there is no restriction about the interval of time */

#ifdef DEBUG
      printf( "ok : %d\n", ok );
#endif

      if ( ok == true )                          /* if the line must be printed */
      {  next = rule[nb_rule].next;              /* index of the word concerned by
                                                    the second part of the rule,
                                                    if any */
         if ( next != -1 )                    /* there is a second part in the rule */
         {  if ( link.count[next] >= rule[next].count )   /* according to the 2d
                                                             part of the rule,
                                                             (count) the line
                                                             must be printed */
            {  if ( rule[next].moment > 0 )   /* if there is a restriction about
                                                 the moment at which the words
                                                 passed */
                  ok = treatment_moment( file, maxindex, link, next );
               else ok = true;                  /* there is no restriction about
                                                   the interval of time */
               return( ok );
            }
            else return( false );   /* according to the second part of the rule,
                                       (count) the line must not be printed */
         }
         else return( true );          /* there is no second part of the rule */
      }
      else return( false );          /* according to the first part of the rule,
                                        (moment) the line must not be printed */
   }
   else return( false );             /* according to the first part of the rule,
                                        (count) the line must not be printed */
}


bool treatment_moment( file, maxindex, link, nb_rule  )

/*****************************************************************
* This function treats the part of the rule 'nb_rule' concerning the moment *
* at which the commands were seen on the link 'link'.                       *
* It returns true if the part of the rule concerning the period of time is  *
* verified.                                                                 *
* It is based on the chronology of the commands' file.                      *
* 'file' points to the first command of the file and 'maxindex' is the      *
* number of commands in the file.                                           *
*****************************************************************/

int         maxindex;
int         nb_rule;

command     *file;

call_record link;

{
  int       i;
  int       count;
  int       num;

  char      *p;

  command   *ptr;

  time_t    min;
  time_t    max;

#ifdef DEBUG
  printf( "beginning of treatment_moment\n" );
#endif
```

```
ptr = file;
num = 0;
min = 0;
max = MAXINT;
for ( i=0; i<maxindex; i++ )
/* we ignore the commands saved  as "old_server" */
{   if ( ( ptr->id_link1 == link.id1 ) && ( ptr->id_link2 == link.id2 ) &&
         ( ptr->origin != 'o' ) )
    {  uppercase( ptr->cmd, 0, MAXSIZE - 1 );   /* capitalizes all the letters
                                                   of the command because we
                                                   want a case unsensitive
                                                   search in the treatment of
                                                   the rules */
       count = find_word( &(ptr->cmd[0]), &(ptr->cmd[MAXSIZE-1]),
                          keywordlist[nb_rule].val,
                          keywordlist[nb_rule].length, &p ); /* sees if the
                                                                command
                                                                contained
                                                                the word
                                                                concerned
                                                                by the rule */
#ifdef DEBUG
       printf( "%d %d ", &(ptr->cmd[0]), &(ptr->cmd[MAXSIZE-1]) );
       printf( "%s %d\n", keywordlist[nb_rule].val,
               keywordlist[nb_rule].length );
       printf( "%s\n", ptr->cmd );
       printf( "count : %d\n", count );
#endif
       if ( count == 1 )                    /* the command contains the word */
       {  num++;           /* there is one more command containing the word */
#ifdef DEBUG
          printf( "num : %d\n", num );
#endif
          if ( num == 1 )             /* records the time of the first command
                                         containing the keyword */
             min = ptr->moment;
          else if ( num == rule[nb_rule].count )    /* records the time of the
                                                       command number
                                                       rule[nb_rule].count
                                                       containing the word */
                  max = ptr->moment;
       }
    }
    ptr++;                                          /* goto the next command */
}
#ifdef DEBUG
  printf( "max : %u min : %u\n", max, min );
  printf( "moment : %d\n", rule[nb_rule].moment );
#endif
  if ( max - min <= rule[nb_rule].moment )           /* if the interval of time
                                                        is repected */
     return( true );
  else return( false );
}

void display( link, maxindex, file )

/*************************************************************
* This function displays the information about the link 'link'.  *
* 'file' points to the first command of the commands' file and   *
* 'maxindex' is the number of commands of the commands'file.     *
*************************************************************/

call_record  link;
command      *file;
int          maxindex;

{  unsigned int         src_decarea, dest_decarea;
   unsigned int         src_decnode, dest_decnode;
   int                  i,j;
   int                  duration;
   static int           counter = 1;
   struct tm            *ttm;
   command              *ptr;

   make_decaddress( link.src_node, &src_decarea, &src_decnode );
                                    /* puts the source address in the
                                       form AA.NNNN */
   make_decaddress( link.dest_node, &dest_decarea, &dest_decnode );
                                    /* puts the destination address in the
                                       form AA.NNNN */
   if ( link.order == 'r' )     /* prints the nodes address in the good order */
   {  printf( "%2u.%-7u ", src_decarea,src_decnode );
      printf( "%2u.%-7u ", dest_decarea, dest_decnode );
```

```
      printf( "   " );
   }
   else
   {  if ( link.order == 'i' )
      {  printf( "%2u.%-7u ", dest_decarea, dest_decnode );
         printf( "%2u.%-7u ", src_decarea,src_decnode );
         printf( "   " );
      }
      else
      {  printf( "%2u.%-7u ", src_decarea,src_decnode );
         printf( "%2u.%-7u ", dest_decarea, dest_decnode );
         printf( "%c ", link.order );
      }
   }
   printf( "%15s %15s ", link.src_uname, link.dest_uname );
   for ( i=0; i<MAXKEY; i++)
       printf( "%d ", link.count[i] );
   ttm = localtime( &link.begin );       /* puts the time in form hour-min-sec */
   printf( " %02d-%02d:%02d:%02d ", ttm->tm_mday, ttm->tm_hour,ttm->tm_min,
           ttm->tm_sec );
   duration = link.end - link.begin;    /* calculates the duration of the link */
   printf( " %5d ", duration );
   printf( " %02x ", link.disc_reason );
   printf( "\n" );
   if ( defil == false )             /* we have to count the lines to display the
                                        information screen by screen */
   {  counter++;                      /* there is one more line on the screen */
      treatment_of_counter( &counter );     /* treats the number of lines on the
                                               screen */
   }
   ptr = file;     /* searchs for the commands associated with the call record */
   for ( i=0; i<maxindex; i++ )                        /* for all the commands */
   {
#ifdef DEBUG
       printf( "link.id1 : %u ptr->id_link1 : %u\n", link.id1, ptr->id_link1 );
       printf( "link.id2 : %u ptr->id_link2 : %u\n", link.id2, ptr->id_link2 );
#endif
```

```
       if ( ( ptr->id_link1 == link.id1 ) && ( ptr->id_link2 == link.id2 ) )
                                   /* if the command belongs to the link */
       {  ttm = localtime( &ptr->moment );          /* puts the time in the form
                                                       hour-min-sec */
          printf( "%02d:%02d:%02d - ", ttm->tm_hour, ttm->tm_min, ttm->tm_sec );
          if ( ptr->origin == 'i' )                /* if it is a line containing
                                                      IP addresses */
          {  /* prints the addresses in hexa */
             printf( "%c%c%c%c%c%d.%d.%d.%d%c%c%d.%c.%d.%d\n",ptr->cmd[0],
                     ptr->cmd[1], ptr->cmd[2], ptr->cmd[3], ptr->cmd[4],
                     ptr->cmd[5], ptr->cmd[6], ptr->cmd[7], ptr->cmd[8],
                     ptr->cmd[9], ptr->cmd[10], ptr->cmd[11], ptr->cmd[12],
                     ptr->cmd[13], ptr->cmd[14] );
          }
          else
          {  printf ( "%c : ", ptr->origin );
             for ( j=0; j<MAXSIZE; j++ )
                 /* control characters are replaced by spaces */
                 if ( ptr->cmd[j] == 0x00 || ptr->cmd[j] == 0x01 ||
                      ptr->cmd[j] == 0x02 || ptr->cmd[j] == 0x03 ||
                      ptr->cmd[j] == 0x04 || ptr->cmd[j] == 0x05 ||
                      ptr->cmd[j] == 0x06 || ptr->cmd[j] == 0x07 ||
                      ptr->cmd[j] == 0x08 || ptr->cmd[j] == 0x09 ||
                      ptr->cmd[j] == 0x0a || ptr->cmd[j] == 0x0b ||
                      ptr->cmd[j] == 0x0c || ptr->cmd[j] == 0x0d ||
                      ptr->cmd[j] == 0x0e || ptr->cmd[j] == 0x0f ||
                      ptr->cmd[j] == 0x10 || ptr->cmd[j] == 0x11 ||
                      ptr->cmd[j] == 0x12 || ptr->cmd[j] == 0x13 ||
                      ptr->cmd[j] == 0x14 || ptr->cmd[j] == 0x15 ||
                      ptr->cmd[j] == 0x16 || ptr->cmd[j] == 0x17 ||
                      ptr->cmd[j] == 0x18 || ptr->cmd[j] == 0x19 ||
                      ptr->cmd[j] == 0x1a || ptr->cmd[j] == 0x1b ||
                      ptr->cmd[j] == 0x1c || ptr->cmd[j] == 0x1d ||
                      ptr->cmd[j] == 0x1e || ptr->cmd[j] == 0x1f ||
                      ptr->cmd[j] == 0x7f || ptr->cmd[j] == 0x80 ||
                      ptr->cmd[j] == 0x81 || ptr->cmd[j] == 0x82 ||
                      ptr->cmd[j] == 0x83 || ptr->cmd[j] == 0x84 ||
                      ptr->cmd[j] == 0x85 || ptr->cmd[j] == 0x86 ||
                      ptr->cmd[j] == 0x87 || ptr->cmd[j] == 0x88 ||
                      ptr->cmd[j] == 0x89 || ptr->cmd[j] == 0x8a ||
                      ptr->cmd[j] == 0x8b || ptr->cmd[j] == 0x8c ||
                      ptr->cmd[j] == 0x8d || ptr->cmd[j] == 0x8e ||
                      ptr->cmd[j] == 0x8f || ptr->cmd[j] == 0x90 ||
                      ptr->cmd[j] == 0x91 || ptr->cmd[j] == 0x92 ||
                      ptr->cmd[j] == 0x93 || ptr->cmd[j] == 0x94 ||
                      ptr->cmd[j] == 0x95 || ptr->cmd[j] == 0x96 ||
                      ptr->cmd[j] == 0x97 || ptr->cmd[j] == 0x98 ||
                      ptr->cmd[j] == 0x99 || ptr->cmd[j] == 0x9a ||
                      ptr->cmd[j] == 0x9b || ptr->cmd[j] == 0x9c ||
                      ptr->cmd[j] == 0x9d || ptr->cmd[j] == 0x9e ||
                      ptr->cmd[j] == 0x9f || ptr->cmd[j] == 0xff    )
                    printf( " " );
                 else
                    printf( "%c", ptr->cmd[j] );
```

```
            printf( "\n" );
         }
         if ( defil == false )       /* we have to count the line to display the
                                        information screen by screen */
         {  counter++;                /* there is one more line on the screen */
            treatment_of_counter( &counter );      /* treats the number of lines
                                                      on the screen */
         }
      }
      ptr++;                                       /* goto the next command */
   }
}


void make_decaddress( addr, area, node )

/*************************************************************
* This function returns the node address 'addr' in the form  *
* AA.NNNN ( Area.Node ).                                     *
* The area number is in '*area' and the node number is in    *
* '*node'.                                                   *
*************************************************************/

unsigned char   addr[2];

unsigned int    *area;
unsigned int    *node;

{
  unsigned int  temp;

  temp = ( ( addr[1] << 8 ) | addr[0] );             /* calculates the address */
  *area = temp / 1024;                                  /* calculates the area */

  *node = temp % 1024;                                  /* calculates the node */
}


void treatment_of_counter( count )

/*************************************************************
* This function controls the number of lines on the screen   *
* 'count' contains the number of lines now on the screen.    *
*************************************************************/

int   *count;

{
```

```
   char  c;

   if ( ! ( *count % 20 ))             /* if there are 19 lines on the screen */
   {  printf( "\npress return to continue\n" );
      scanf( "%c", &c );                        /* waits for the CR to continue */
      *count = 1;
   }
}
```

Annexe 2
L'utilisation des programmes

The user's guide

What is the program for

The program monitors the different Decnet links entering and going out of CERN and tries to discover the suspicious ones.

The objective is to find people who try to connect to machines at CERN and who are not authorized. Indeed, these people can cause lots of damage inside CERN : unauthorized reading of information, unauthorized change or removal of information, ... The program also detects people who try to connect to external machines without authorization from a machine at CERN. With these people CERN may get a bad reputation in the field of security.

The aim of the program is to detect the security violations not to prevent them. The program collects information about the links and analyses this information in order to find some intrusion attempts so we can learn significant information about the intruders and their methods.

What the program does

The program intercepts Decnet packets on the network. On basis of these packets, it monitors the different links and tries to discover the suspicious ones. This is done by searching for some keywords in the packets, like "password", "user authorization failure"... and counting the number of occurrences of these keywords seen on a link. The program also reconstructs the lines typed by a user who made a "set host" and the response sent from the host to this user's terminal. Once a line is reconstructed the program searchs for some keywords in this line and if a keyword is found the line is saved in a file.

By default, this search is case unsensitive but a case sensitive search is offered as option.

The program also detects the IP packets encapsulated in Decnet packets. The IP addresses contained in these packets are saved in a file.

The information concerning a link is saved in the form of a call record.

For each link, the program records in the call records file the address of the source node and the destination node, the sens of the link, the source link and the destination link, the source username and the destination username, the number of packets observed

on the link, counters for the keywords, the reason of the disconnection, the hour the link was created, the hour the link was aborted and an identifier of the link, given by the program.

The commands file contains all the "suspicious" commands and the IP addresses of the IP packets encapsulated in Decnet packets together with the identifier of the link on which they were seen, the hour they were seen and their origin (i.e. the host, the server, it isn't a CTERM connection or IP addresses).

It is possible for the program to only consider the links for which the origin is in CERN in the search of keywords and the reconstruction of the lines.

The results' files

Each hour new files are created.

The name of the call records file is built as follows :

Call_records_dd-mm-yy-hh.dat

while the name of the file containing the suspicious commands is built as follows :

Command_dd-mm-yy-hh.dat

where

- dd represents the day of creation of the file,
- mm represents the month of creation of the file,
- yy represents the year of creation of the file,
- hh represents the hour of creation of the file.

Each day at midnight the information concerning the links created during the day but not yet saved is saved in the call records file of the day.

These files are created in the directory where the monitor runs.

These are binary files and another program will have to read these files and display the information on the screen.

The keyword.txt file

The keywords to search for in the packets and in the reconstructed lines are found in a file written by the user : "keyword.txt". This file also contains the rules to apply when we display the information.

The file "keyword.txt" must not contain more than MAXKEY keywords. If the file contains too many keywords, the illegal keywords will be ignored and a warning will appear in the logfile. MAXKEY is currently 20.

There is only one word per line, and the maximum length of a word is MAXSIZE-2. If the word is longer, it will be truncated and a warning will appear in the logfile. MAXSIZE is currently 82.

The blank characters after a word and empty lines are ignored but not the blank characters before a word.

Remark that the search takes into account each sequence of characters that corresponds to a keyword. So, if you want to search, for example, the word "user" the word "username" will also be taken into account.

A line containing a rule begins with "R ". Note that the space is important because we may have keywords beginning with "R".

Each rule may contain an "AND" or an "OR" but only one. If there are several "AND" or/and "OR" in a rule you will receive a warning and the program will stop.

The rules are expressed as follows :

R keyword >= occurrence (time)

This means that if the counter for the 'keyword' has a value greater or equal to occurrence and the interval of time, expressed in seconds, between the first occurrence and the occurrence number 'occurrence' is greater or equal to 'time' the line will be printed. If 'occurrence' is smaller or equal to zero and/or 'time' is smaller than zero you will receive a warning and the program will stop. If '>=' is not present in one of the rules you will receive a warning and the program will stop.

We can also have :

R keyword1 >= occurrence1 (time1) AND keyword2 >= occurrence2 (time2)

R keyword1 >= occurrence1 (time1) OR keyword2 >= occurrence2 (time2)

Be careful the spaces before and after "AND" and "OR" are necessary.

A word may only appear in one rule. If a word appears in more than one rule, you will receive a warning and the program will stop.

If a word in a rule is miss-spelt or doesn't figure in the keywords list, you will receive a warning and the program will stop. Nevertheless, words may be lowercase or uppercase. The match is case unsensitive.

The maximum size of a rule is MAXSIZE-2.

If a rule concerned a word ignored during the search because there were too many keywords, you will receive a warning and the program will stop.

A line beginning with '!' is a comment line.

Be careful the treatment program must be executed with the same keywords file as monitor. Nevertheless the rules can be modified. The important thing is that the file contains the same keywords in the same order.

Be careful if you change the keywords file, you have to start the monitor program again if you want the new words to be considered.

If the keywords file doesn't contain any keywords, you will receive a warning on the screen and the programs will stop.

The keyword.txt file must be in the directory where the programs run.

The routers.txt file

It is also possible for the program to ignore the connections between some addresses. These addresses are found in a file written by the user, the file "routers.txt".

The purpose of this is to ignore the packets which enter CERN via a router and go out of CERN immediately via another router. These packets are of no interest for us.

The addresses are the Ethernet addresses written in hexadecimal, and there is only one address per line.

The file must not contains more than MAXROUTERS addresses. If so, the illegal addresses will be ignored with a warning in the logfile. MAXROUTERS is currently 10.

An address contains 6 bytes, i.e. 12 characters, between 0-9 and a-f. If it contains more than 6 bytes, it will be cut (only 6 bytes will be considered), with a warning in the logfile. If it contains less than 6 bytes, the address will be filled with '0', with a warning in the logfile.

The bytes may be separated with '-'. These separators must be placed properly.

Blank characters before or after the address and empty lines are ignored.

A line beginning with '!' is a comment line.

Be careful if you change the file, you have to start the program again if you want the new addresses to be considered.

If the file is empty and the program is asked to ignore the connections between some addresses , you will receive a warning on the screen and the program will stop.

The routers.txt file must be in the directory where the programs run.

The logfile.log file

This file contains all the warnings about the reading of the files "keyword.txt" and "routers.txt" As mentionned above, it will contain a warning if there are too many keywords or too many addresses, if a word or an address is too long or if an address is too short.

This file is created in the directory where the monitor runs.

The treatment program

This program has to read the files and display the information on the screen according to the rules contained in the file "keyword.txt". It has to make the link between the two result files which is feasible via the link identifier present in both files.

The treatment program presents the data in the following order :

source address (in the form area.node) - destination address (in the form area.node) - link order - source username - destination username - counters for the other keywords (in the same order as the words in the file) - starting hour - link's duration - reason of the disconnection - eventually, below this, the suspicious commands seen on the link together with the hour at wich these commands were seen. The hour is at the beginning of the line.

The link order takes the value '?' if we don't know which node has initiated the link. Otherwise, the link order doesn't have any value.

The disconnection reason may take several values :

'00' means that there was no error. '04' means that the destination end user does not exist. '09' means that the logical link has been aborted by the end user. '26' means that the connect request has been aborted by the end user. '99' means that we haven't see the disconnect initiate packet. '98' means that we haven't see the disconnect initiate packet, but we had to save the data at the end of the day. '97' means that we had to save the data because the timestamp became too great.

The starting hour contains the day, hour, minutes and seconds of the beginning of the link.

The duration of the link is expressed in seconds.

A command preceeded by 's' is a "suspicious" command sent by the server, i.e. the user, in a CTERM connection. A command preceeded by 'h' is a "suspicious" line sent by the host, i.e. the system, in a CTERM connection. A command preceeded by 'o' is a command sent by the server which triggered a "suspicious" response from the host (this

response figures just after the command) in a CTERM connection. A command preceeded by '?' doesn't belong to a CTERM connection.

(Be careful there can be transmission errors and some commands may by preceeded by '?' even on a CTERM connection because the byte identifying the type of connection on which a command is detected may be corrupted.)

The IP addresses detected are preceeded by 'IP'.

The program may print the information about all the links or print the information according to the rules contained in the file "keyword.txt".

Note that the commands are saved in the file opened when they are detected. However, the call records associated with these commands may also appear in a file at a later date. If this is the case, the corresponding commands will not be printed together with the call record, they are only printed while reading the file of the day during which they were detected. Observe that the call record is in this file because all the call records of the day are saved at midnight. If a link exists for more than one day the commands will be split into several files and therefore, the printing of the commands will also be split. However the printed counters consider all the occurrences of the keywords detected until the display of the information. The treatment of the interval of time only takes into account the commands present in the file of the day treated.

The logtrt.txt file

This file contains all the warnings occurred during the reading of the keywords during the execution of the treatment program.

This file is created in the directory where the treatment runs.

Operation

To start the programs you have to add two lines in the login.com file :

```
monitor :== $disk:[directory]monitor
treatment :== $disk:[directory]treatment
```

where

- disk is the name of the disk where the program is,
- directory is the name of the directory where the program is.

You can then start the programs by typing :

monitor

treatment filename

where filename is the name of the call records file you want read.

This file must be in the directory where the program runs.

The program will build the name of the commands file from the name of the call records file. You should concatenate all the call record files of the day, and all the command files of the day before reading them. This can be done with the 'append' or the 'copy' command of VMS.

By default the program monitor ignores the connections between routers and consider only the links which origin is in CERN for the search for keywords and the reconstruction of the lines. Furthermore, by default, the search for keywords is case unsensitive.

By default the program treatment prints the information according to the rules contained in the file "keyword.txt" and stops the display after each screen of information. You then have to press RETURN to continue the display.

If you want the monitor to also consider the links for which the source is inside CERN, you have to type :

monitor -o.

If you want a case sensitive search for keywords, you have to type :

monitor -s.

If you want the monitor not to ignore the links between the addresses contained in the file "routers.txt", you have to type :

monitor -a.

These arguments can be combined.

If you want the treatment to print the information about all the links, you have to type :

treatment -a filename.

If you want the treatment not to stop the display after each screen, you have to type :

treatment -d filename.

These arguments can be combined.

Furthermore if the keywords file doesn't contain any rules the program will print the information about all the links.

If the command to launch one of the program is badly typed, the program will inform you and give you some help.

Be careful if you interrupt the program the last call record may be corrupted.

Compiling

To compile and link the monitor program, type @makemon.

To compile and link the treatment program, type @maketrt.

Makemon.com and maketrt.com must be in the same directory as the source of the programs.