Marc Girard

SÉCURITÉ SOUS WINDOWS NT 4.0 ® : MONITORAGE D'EXÉCUTION EN VUE DE DÉTECTER DU CODE MALICIEUX

Mémoire présenté
à la Faculté des études supérieures
de l'Université Laval
pour l'obtention du grade
de maître ès sciences (M.Sc.)

Département d'informatique
FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL

Mars 2001

Si Dieu avait voulu que l'on prit la Vie sérieusement,
il ne nous aurait pas donné le sens de l'humour.
Anonyme. (La-Bas, 4/3/96)

RÉSUMÉ

L'informatique fait maintenant partie de la vie courante. De plus en plus de gens utilisent l'ordinateur comme outil de travail. Ils emmagasinent des informations, souvent sensibles, sur leurs machines. Selon les outils utilisés, tels les systèmes d'exploitation, il est possible de protéger ces informations contre des gens qui voudraient accéder à ces informations sensibles. Par contre, même avec des outils spécialisés, il existe différents problèmes qui ne peuvent être résolus pour protéger les informations car il y a trop d'éléments extérieurs qui affectent la protection des informations. Le choix des outils, le niveau de connaissance des utilisateurs, la configuration du réseau ne sont que quelques exemples qui, une fois combinée, peuvent affaiblir la protection des informations.

Le présent document tente de promouvoir une approche qui saura répondre aux exigences de l'analyse dynamique. La conception d'un moniteur contrôlant les accès aux ressources critiques est un complément aux autres techniques de détection de code malicieux. L'avantage de l'utilisation d'un moniteur réside dans sa capacité à réagir sur-le-champ lorsqu'il détecte une action jugée malicieuse (en fonction des politiques de sécurité).

Ce travail présente une approche fondée sur une application capable de réagir et de s'ajuster en fonction des actions passées. En concevant un moniteur d'analyse dynamique contrôlant les accès à des ressources critiques, l'objectif de ce travail a été atteint. Ce contrôle ne se fait pas sans la participation de règles ou de politiques de sécurité. En définissant des politiques de sécurité, il devient possible de combiner les différents éléments intervenant dans le processus de gestion.

Ce projet a permis de montrer qu'il est possible de surveiller et de contrôler les accès à des ressources sensibles. Les moyens mis en place ne couvrent pas tous les aspects du projet. Certains éléments restent à concevoir pour réaliser le plein potentiel du projet.

Avant-propos

Mes études de Maîtrise en informatique ont été faites pour aller chercher des connaissances sur un domaine très précis de l'informatique. Mon désir d'apprendre, de connaître et de maîtriser des connaissances en ce qui concerne les systèmes d'exploitation et la sécurité informatique m'a grandement motivé à poursuivre le projet que j'ai entrepris. Pour réaliser ce projet, j'ai eu le bonheur d'avoir des gens très précieux autour de moi pour m'encourager, me motiver, me former et m'écouter.

Merci à ma conjointe, Shirley et mes enfants Émilie, Cédrick, Maxime et Francis pour leur grande patience et leur compréhension. La réussite de ce travail provient en grande partie d'eux. Ils furent ma source de motivation constante. Merci à mon directeur, Mourad Debbabi, de m'avoir donné la chance de participer activement au projet Mali-COTS en me confiant la tâche de concevoir un moniteur pour l'analyse dynamique de programmes. Merci à Jules Desharnais, Nadia Tawbi, Jean Bergeron, Robert Charpentier, Martin Salois, Béchir Ktari, Lamia Ketari pour vos conseils, connaissances et votre support. À certaines occasions, vous m'avez fourni aide, réconfort et encouragement. Merci à Matthieu Bourdeau, Sébastien Lehr, Zained Ben Fredj, Vincent Labbé et Luc Poulin pour m'avoir aidé à la programmation et à la conception. Votre aide fût grandement utile et apprécié.

Marc Girard
Août 2000

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES EXTRAITS

PARTIE I – ÉTAT DE L'ART

Chapitre 1

INTRODUCTION

" Ce n'est pas parce que les choses sont difficiles
que nous n'osons pas, c'est parce que nous n'osons pas
qu'elles sont difficiles. "
Sénèque

L'informatique fait maintenant partie de la vie courante. De plus en plus de gens utilisent l'ordinateur comme outil de travail. Ils emmagasinent des informations, souvent sensibles, sur leurs machines. Selon les outils utilisés, tels les systèmes d'exploitation, il est possible de protéger ces informations contre des gens qui voudraient accéder à ces informations sensibles. Par contre, même avec des outils spécialisés, il existe différents problèmes qui ne peuvent être résolus pour protéger les informations car il y a trop d'éléments extérieurs qui affectent la protection des informations. Le choix des outils, le niveau de connaissance des utilisateurs, la configuration du réseau ne sont que quelques exemples qui, une fois combinée, peuvent affaiblir la protection des informations.

La sécurité informatique devient donc un élément essentiel dans la gestion courante du travail. Tous les utilisateurs sont conscients, à des degrés différents, de l'importance de protéger les informations. Les virus sont un exemple connu par la majorité des utilisateurs et ils savent l'importance de se protéger pour ne pas perdre d'informations. Pour optimiser la gestion de la sécurité informatique, il est important d'avoir des outils qui prennent en compte plusieurs facteurs et qui, en les combinant, ne diminuent pas la protection de l'environnement.

Des recherches dans le domaine informatique apportent des solutions aux problèmes de la sécurité informatique. L'approche statique, qui consiste à faire une analyse d'un programme, permet de certifier si le programme a un comportement souhaitable en fonction de certaines règles de sécurité. L'approche dynamique propose, quant à elle, une surveillance en temps réel du comportement environnemental de l'utilisateur en fonction de certaines politiques de sécurité. Dans les deux cas, chaque approche tente de détecter des séquences malicieuses qui pourraient s'exécuter pendant que l'utilisateur est au travail.

Le présent document tente de promouvoir une approche qui saura répondre aux exigences de l'analyse dynamique. La conception d'un moniteur contrôlant les accès aux ressources critiques est un complément aux autres techniques de détection de code malicieux. L'avantage de l'utilisation d'un moniteur réside dans sa capacité à réagir sur-le-champ lorsqu'il détecte une action jugée malicieuse (en fonction des politiques de sécurité).

Ce travail présente une approche fondée sur une application capable de réagir et de s'ajuster en fonction des actions passées. Pour réaliser cette approche, il est essentiel de définir les intervenants impliqués. Premièrement, il y a le moniteur qui agit comme superviseur et qui dicte les gestes à poser aux gardiens surveillant les ressources. Deuxièmement, il y a les gardiens ou filtres qui se collent aux ressources critiques afin d'intercepter les requêtes et de les traiter. Ces gardiens correspondent à des pilotes intermédiaires dans l'environnement Windows NT. Troisièmement, il y a les politiques de sécurité qui dictent les règles de conduite de chaque application. Ces politiques de sécurité devront être définies pour le contrôle des ressources. Ensemble, ces intervenants forment une approche permettant de gérer et de contrôler tous les accès aux ressources importantes du système d'exploitation Windows NT.

Le document se présente comme suit : la première partie présente les motivations et l'état de l'art sur lequel s'appuie cette recherche, la deuxième partie se rapporte au moniteur en y présentant les concepts, les filtres ou pilotes jouant un rôle de gardien et le mécanisme de surveillance et de contrôle mis en place, la troisième partie présente l'implantation de DaMon, le moniteur d'analyse dynamique et la quatrième partie correspond aux annexes contenant des informations supplémentaires rattachées à certains sujets.

Chapitre 2

MOTIVATIONS

"Le plus grand arbre est né d'une graine menue;
une tour de neuf étages est partie d'une poignée de terre."
Lao-Tseu

L'utilisation quotidienne d'outils informatiques fait maintenant partie intégrante de notre vie. Malheureusement, certains fabricants de ces outils profitent de l'incapacité des utilisateurs à déterminer si les produits utilisés sont crédibles. La crédibilité d'un produit peut être vue selon différents aspects. Un outil crédible pourrait être l'équivalent d'un produit stable ne comportant aucune erreur de conception assurant à son utilisateur l'intégrité de ces données (aucune perte de données). Il pourrait être l'équivalent d'un produit assurant une protection à son utilisateur contre toute manipulation frauduleuse de ces données (destruction ou modification) et divulgation (envoi sur un réseau sans que l'utilisateur le sache).

La sécurité des données devient donc un aspect important de la vie quotidienne. Avec une utilisation croissante de ressources comme le réseau Internet, il devient impératif de s'assurer que les informations manipulées par les différents outils logiciels seront protégées contre toute action malicieuse; une action malicieuse étant considérée comme une action non désirée et/ou non consentie par l'utilisateur.

Pour les raisons énumérées précédemment, la détection de code malicieux devient un sujet très important. Utiliser des outils permettant de circonscrire les effets néfastes, voire de les éliminer, est un atout important dans un environnement nécessitant un contrôle serré de sécurité. Ces outils de détection permettent aussi de rassurer l'utilisateur. Avec l'utilisation de plus en plus élevée de logiciels conçus par des firmes spécialisées ou « COTS » (Commercial-Off-The-Shelf), il est essentiel, pour certaines organisations, d'obtenir une certification ou une assurance que les produits utilisés sauront répondre à des normes strictes de sécurité. De plus, aucun outil commercial peut garantir qu'un logiciel ne contient pas de codes malicieux et que ce logiciel n'exécute que ce qu'il est supposé faire. Par exemple, une organisation, manipulant des informations sensibles et utilisant un logiciel particulier de traitement de ces données, voudra s'assurer que ce même logiciel ne contiendra aucune séquence de code malicieux pouvant envoyer des informations sensibles à une adresse quelconque sur un réseau. Même si le logiciel

n'a pas été conçu pour envoyer des informations sur un lien de communication, il n'existe aucune preuve que ce logiciel ne le fait pas. À la lumière de cet exemple, il apparaît évident qu'une preuve ou autres moyens devraient être créés pour vérifier et contrôler ce genre de comportement malicieux. Actuellement, certains groupes de recherche [2][10] essaient de mettre en place des moyens certifiant que les produits sont sécuritaires et qu'ils sauront répondre à des normes de sécurité précises. Les prochaines sections présentent les aspects importants qui devront être étudiés pour bien cerner les composantes de la sécurité informatique. Dans un premier temps, les approches statiques et dynamiques sont présentées afin de connaître leurs caractéristiques. Dans un deuxième temps, il est essentiel de définir ce sur quoi porte la sécurité informatique en introduisant la notion de ressources critiques. Dans un troisième temps, le concept de politiques de sécurité est présenté pour démontrer l'importance d'avoir des règles de sécurité. La dernière section apporte des exemples qui montrent la nécessité d'avoir un environnement sécuritaire et de concevoir des moyens de l'atteindre.

2.1 Les approches

Plusieurs approches sont proposées pour développer des outils fournissant un support sécuritaire. L'analyse statique d'un programme permet de découvrir des séquences possibles de codes malicieux, d'optimiser le code pour s'assurer de sa solidité et/ou de fournir une preuve de sa crédibilité à l'aide d'un certificat. Malheureusement, dans certain cas, l'analyse statique ne peut garantir que le code ne contient pas de séquences potentiellement malicieuses dû au fait de certaines situations indécidables. L'analyse du flot de contrôle du programme peut ne pas couvrir tous les chemins possibles à cause d'une conception déficiente, de structures de code mal conçues ou par le simple manque d'information au moment de la prise de décision.

L'analyse dynamique, quant à elle, est une autre approche. Celle-ci permet de suivre l'exécution d'une application en temps réel. En surveillant des ressources précises, l'analyse dynamique peut réagir et empêcher instantanément toutes actions malicieuses. Un des avantages de l'analyse dynamique est sa capacité d'intercepter une action qui n'avait pas été détectée avec l'analyse statique. L'analyse dynamique n'a pas à se préoccuper du chemin que prendra une application avant d'être exécutée. Elle n'a pas à décider si le chemin conduit à une action malicieuse. Un autre avantage de l'analyse dynamique est qu'elle permet de surveiller toute application n'ayant pas été certifiée auparavant. L'analyse dynamique, contrairement à l'analyse statique, ne se préoccupe pas de la validité de toutes les informations manipulées. Tant que les ressources sensibles ne sont pas atteintes, les applications s'exécutant dans l'environnement contrôlé peuvent travailler en toute liberté.

Naturellement, l'analyse dynamique fait référence à des situations différant comparativement à l'analyse statique. Par exemple, même si l'action de copier un fichier est légale, elle peut devenir problématique dans un environnement où l'information est contrôlée. Un fichier classé « top secret » ne pourra pas être copié dans un répertoire classé « secret ». Des gens n'ayant pas le privilège de lire des documents classés « top secret » pourraient y parvenir si un contrôle n'est pas mis en place. L'analyse dynamique permet ce genre de surveillance alors que l'analyse statique n'est pas en mesure de déterminer si l'action de copier un fichier est permise ou non ne sachant pas à l'avance quel type de document l'application devra copier.

À la lecture de ces faits, il devient impératif de se doter d'outils permettant un contrôle étroit de l'environnement dans lequel tournent les applications logicielles. Ce travail tend vers l'approche de dynamique parce qu'il est important d'avoir des moyens, autre que la certification de code, permettant de surveiller et de réagir à des situations qui pourraient être jugées critiques.

Pour surveiller et contrôler l'environnement d'exécution de programmes, il est nécessaire de concevoir un moniteur. Ce moniteur devra protéger des ressources critiques, répondre à des obligations précises selon des politiques de sécurité rigoureuses, réagir et être constamment à l'affût de toutes activités présentes dans son environnement.

2.2 Les ressources

Un des aspects du monitorage est la capacité du moniteur à protéger des ressources critiques. La principale difficulté est la détermination des ressources critiques. Les attaques commises dans un environnement informatique quelconque ne sont pas nécessairement les mêmes dans un autre environnement. Dans chaque cas, les ressources visées ne sont peut-être pas les mêmes. Il apparaît évident que chaque environnement possède ses propres ressources critiques et qu'il appartient au responsable de les identifier.

À la lumière de cette constatation, la définition des ressources critiques devient essentielle pour quiconque veut protéger son environnement. Il suffit d'observer le travail des usagers malfaiteurs pour s'apercevoir que le type de ressources et d'attaque est diversifié. Ces observations suffisent à conclure à la nécessité de définir les ressources critiques et de mettre les moyens nécessaires pour contrer les attaques contre celles-ci.

2.3 Politiques de sécurité

Les logiciels anti-virus permettent de détecter la présence de virus connus. Selon certaines règles précises et possiblement définies par l'utilisateur, il sera permis de supprimer les virus. Ces règles définissent les actions à prendre en cas de détection de codes malicieux reconnus. Ce type de règles est assez simple. Qu'advient-il si les situations se

complexifient? Par exemple, comment réagira le logiciel anti-virus si celui-ci fait face à une forme mutante de virus et pour laquelle la détection devient difficile parce que l'heuristique de détection ne peut confirmer la présence de ce virus? Si une règle n'a pas été définie, il est possible que le logiciel n'informe pas l'utilisateur de ce problème. Il est aussi possible que le logiciel signale ce problème et demande à l'utilisateur s'il peut supprimer le code suspect. Dans les deux cas, il s'agit d'actions extrêmes (laisse passer ou supprime le code suspect). Une approche intéressante serait de permettre à l'utilisateur d'exprimer ses préférences et ainsi obtenir une action intermédiaire. Par exemple, l'utilisateur pourrait signifier au logiciel anti-virus, qu'advenant une situation complexe, il préfère surveiller le comportement du code suspect en fonction de balises précises. Ainsi, si certaines balises sont atteintes, l'usager considérera ces gestes comme étant des actions non permises et le code suspect comme étant malicieux.

La possibilité d'inclure des politiques de sécurité régissant les actions à surveiller et à commettre offrent une plus grande flexibilité à l'utilisateur. Cette constatation ouvre la voie à des recherches intéressantes quant à la possibilité de définir des règles de comportement applicables à des programmes informatiques.

2.4 Exemples

Afin de se convaincre davantage sur la nécessité d'avoir un environnement sécuritaire et de concevoir des moyens pour l'atteindre, il est intéressant de voir et de comprendre que les menaces proviennent de tous les côtés et qu'il est obligatoire de les circonscrire pour éviter tout dommage irréparable.

2.4.1 Les menaces

La Figure 1 montre un graphique exposant l'importance de la sécurité pour protéger les informations. Les types d'attaque présentés par cette figure démontrent qu'une fois à l'intérieur d'un environnement informatique, l'usager peut commettre plusieurs actions qui auront des conséquences désastreuses. Ce que la figure ne mentionne pas est à savoir si les usagers commettant des actions malicieuses proviennent essentiellement de l'extérieur ou de l'intérieur de l'environnement informatique. S'ils proviennent de l'extérieur, on peut supposer qu'ils ont réussi à obtenir les autorisations nécessaires sans la permission du responsable de l'environnement.

Cette figure illustre donc plusieurs problèmes à résoudre. Un premier problème est l'obligation de bien contrôler les accès pour éviter qu'un agent extérieur accède à l'environnement. Un deuxième problème est de s'assurer que, même si l'agent a les autorisations nécessaires, il n'exécute pas d'actions malicieuses afin d'accéder, détruire, modifier ou émettre des informations sensibles. Malheureusement, même si des efforts sont

faits pour contrôler les accès, il existe très peu de moyens pour contrôler dynamiquement l'accès à toutes les ressources critiques d'un environnement informatique.

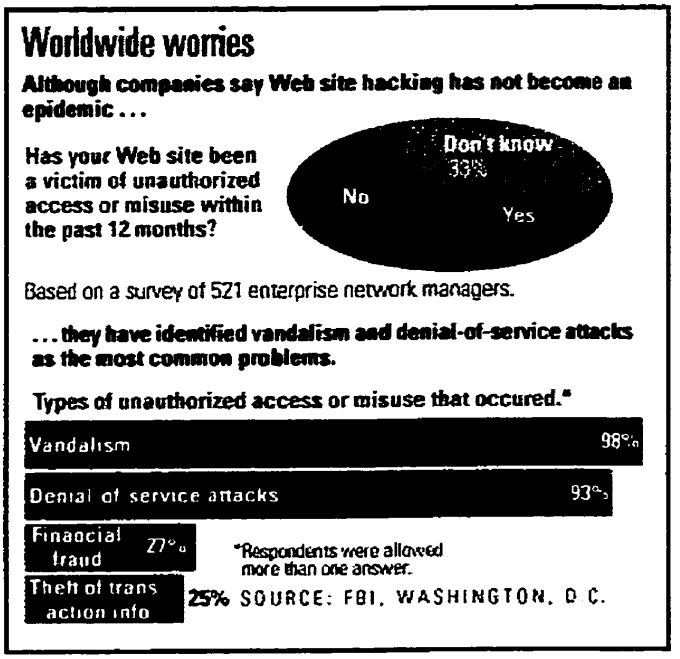


Figure 1: exemple de menaces[1]

2.4.2 Les règles de comportement

Les logiciels anti-virus, les pare-feux « firewalls », « proxies » et compagnie offrent la possibilité à l'utilisateur de définir des règles de conduite. Par exemple, les logiciels anti-virus ont la tâche de surveiller les accès aux fichiers. Avant d'accéder aux fichiers, le logiciel anti-virus doit vérifier si le fichier demandé n'a pas été modifié. S'il détecte une anomalie, le logiciel réagira en fonction de paramètres définis par l'usager en désinfectant le fichier ou laissant passer celui-ci sans le nettoyer.

Les « firewalls » et « proxies » sont un autre type d'exemple. Ceux-ci permettent de filtrer les communications entrantes et sortantes d'un environnement informatique. Ces logiciels ont la responsabilité d'intercepter les communications et de voir si les paquets répondent à des critères de sélection définis par l'usager. Si oui, les paquets continuent leur route ou, dans le cas contraire, seront détruits.

[1] Graphique tiré d'un article **www.nwfusion.com/news/2000/0403intrusion.html**

Dans les deux cas mentionnés précédemment, on constate qu'il y a toujours une action suivie d'une réaction. La combinaison des deux (actions-réactions) constituent les règles de comportement du logiciel. Ce dernier exécute des actions (surveiller) et réagit en fonction de critères ou règles définis par l'usager (laisser passer ou détruire). On note toutefois, à l'aide de ces exemples, que les règles de comportement sont souvent limitées. Dans un environnement informatique, les interactions sont souvent plus complexes. Dans les exemples ci-dessus, le logiciel anti-virus ne surveille que les accès aux fichiers et les logiciels « firewall » et « proxy » ne surveillent que les communications. Qu'advient-il si le logiciel anti-virus ne détecte aucun virus dans le fichier contenant tous les mots de passe de l'environnement informatique et que celui-ci est envoyé par courrier électronique sachant que le « firewall » accepte de laisser passer les communications reliées au courrier électronique?

Chacun des logiciels a fait son travail mais personne ne s'est préoccupé de la sensibilité des informations véhiculées et de la pertinence à transmettre ces informations. Aucune règle n'a été définie pour contrer ce type de problème. Dans ce cas précis, il existe deux problèmes qui n'ont pas été résolus; à savoir la sensibilité du fichier accédé (fichier de mots de passe) et la séquence d'exécution (accéder et envoyer un fichier). Si des règles précises avaient été mises en place, ce fichier aurait été mis sous surveillance et seulement quelques actions auraient été possibles.

À la lecture des tous ces aspects, il apparaît évident que des moyens doivent être mis en place pour protéger un environnement informatique. Un de ces moyens est la conception d'un moniteur servant au monitorage d'exécution en vue de détecter des actions malicieuses.

Chapitre 3

ÉTAT DE L'ART

"Avec le temps et la patience,
la feuille du mûrier devient de la soie."
Proverbe Chinois

En analysant les différents aspects reliés à la sécurité informatique, plusieurs points restent à couvrir pour protéger efficacement les informations sensibles. À partir des constatations faites au chapitre précédent, il en découle qu'une attention particulière devra être portée pour la mise en application d'un moniteur. Pour générer ce moniteur, il est essentiel de connaître et de maîtriser les points forts et faibles de tous les intervenants impliqués.

La présente section couvre les aspects nécessaires pour mettre en place un outil de surveillance et de contrôle. Il est nécessaire de connaître l'environnement Windows NT 4.0 , les ressources devant être surveillées et contrôlées dans cet environnement, d'avoir de nouveaux horizons pour la mise en place de politiques de sécurité et de voir les produits pouvant répondre le mieux possibles à la détection de codes malicieux.

3.1 Système d'exploitation Microsoft Windows NT 4.0 ®

Dans le cadre du projet de détection de codes malicieux, MaliCOTS, il a été convenu que l'environnement de travail répondrait aux attentes du marché. Pour cette raison, le système d'exploitation Microsoft Windows NT 4.0 SP5 (« Service Pack 5 ») a été choisi. Il est nécessaire de connaître, de comprendre et d'analyser l'architecture de ce système d'exploitation. Le fonctionnement du noyau et la mise en place de la sécurité sont les points forts de cette étude.

Puisque le système d'exploitation sélectionné est produit par une entreprise privée, il est difficile d'obtenir des informations pertinentes sur les sujets intéressants. Certains ouvrages existent mais ne contiennent pas toutes les informations pertinentes. Solomon [29]présente l'architecture de Windows NT en expliquant le fonctionnement, l'interaction et la description de toutes les composantes du système d'exploitation. Ce livre est une excellente référence parce que l'auteur possède le code source du système d'exploitation

Windows NT 4.0 . De plus, les informations sont approuvées par les concepteurs du système d'exploitation.

Les concepteurs de Windows NT ont misé sur une architecture en couche permettant ainsi une modularité des composantes. Chacune des composantes a un rôle précis à jouer. Cette modularité donne l'avantage de pouvoir modifier une partie du noyau sans avoir à tout recompiler. De plus, un des aspects importants de la conception du système d'exploitation est la notion de sécurité. La sécurité proposée par Windows NT permet de protéger les informations contre les accès non permis. D'ailleurs, la conception du système d'exploitation suit les normes de sécurité développées par le DoD [6] (« Department of Defense ») de l'armée américaine. Aujourd'hui, le système d'exploitation Windows NT 4.0 possède la cote C2. En révisant ces normes, on comprend mieux la conception du système d'exploitation. L'Annexe I explique brièvement les différentes cotes définies par le DoD.

Entre autres, Windows NT sépare les applications du noyau. Les applications exécutées dans le système d'exploitation possèdent un environnement qui leur est réservé. De cette façon, il est interdit à une application d'accéder à l'environnement d'une autre application. Par contre, lorsque l'exécution se transfère en mode noyau, cette restriction ne tient plus. Les composantes du noyau peuvent accéder à n'importe quel environnement, lire ou écrire des données se rapportant à l'application concernée et traiter ces données sans que ces actions soient considérées illégales. Cette caractéristique est importante puisqu'elle est la cause de certaines faiblesses dans la sécurité de Windows NT. Un programme malicieux peut profiter de ces faiblesses et causer des problèmes. Windows NT a mis en place certains mécanismes qui contrôlent les accès (Gestionnaire de mémoire) mais ne peut fournir une preuve qu'il s'agit bien de la bonne personne ou de la bonne application. Si une application obtient les privilèges d'administrateur, situation causée par une faiblesse du système d'exploitation par exemple, aucune composante ne pourra bloquer l'accès à cette application puisque celle-ci a maintenant tous les droits. On voit donc l'importance de contrôler les accès surtout ceux visant des ressources critiques.

Solomon réussit à couvrir les aspects essentiels de l'architecture de Windows NT 4.0 . Par contre, certaines composantes méritent d'être approfondies pour s'assurer que le système d'exploitation couvre bien les aspects reliés à la sécurité. Russinovich [20][21][22][23][24] a écrit une série d'articles sur les composantes de Windows NT afin d'expliquer précisément le fonctionnement de ces dernières. Il a conçu des outils qui permettent de surveiller des ressources de l'environnement Windows NT. **Filemon v4.28, Regmon v4.24, PMon v1.0** et **TCPView v1.0**[2] sont respectivement des produits surveillant les accès aux disques, la base de registres, la gestion des processus et

[2] Il est possible de récupérer gratuitement tous ces logiciels.
http://www.sysinternals.com/ntinternals.htm

les ports de communication utilisant le protocole TCP/IP. Dans certains cas, le code source est disponible. Ces informations sont d'un grand apport dans la compréhension détaillée de l'architecture de Windows NT. L'Annexe II fournit un échantillon des informations affichées par ces applications.

Suite aux informations recueillies, il est apparu évident que la conception d'un moniteur d'analyse dynamique passerait par la conception de filtres. Les filtres sont des pilotes de périphérique ou drivers[3] ayant un rôle précis à jouer. Baker [1], Bessonov [3], Russinovich [20][21][22][23][24] et Viscarola [31][32][33][34] fournissent des informations très précises sur l'architecture du noyau, les rôles joués par les composantes du noyau et les fonctions utilisées par les pilotes. Le **Chapitre 5 – Les filtres** porte exclusivement sur la composition des filtres. En réalité, la conception du moniteur repose essentiellement sur la conception des filtres nécessaires pour surveiller les ressources critiques. Les recherches effectuées démontrent que les filtres sont la meilleure façon de surveiller et contrôler les accès aux ressources critiques.

D'autres auteurs, [12][25][28], ont contribué à comprendre la sécurité de Windows NT. Leurs contributions se situent à la mise en place des politiques de sécurité du système d'exploitation Windows NT. Ces politiques de sécurité représentent la gestion des comptes d'usager, des permissions accordées et de la configuration de certains services pour assurer une optimisation de la sécurité tout en respectant les normes de sécurité du DoD.

Le travail accompli, afin de connaître et de comprendre l'environnement de travail sélectionné, a permis de découvrir que la conception d'un moniteur d'analyse dynamique d'exécution de programmes devait se faire en utilisant des filtres. Ces filtres auront le rôle de gardien et seront rattachés à des ressources précises.

3.2 Identification des ressources sensibles

Connaissant l'environnement de développement et ses composantes, il est maintenant nécessaire d'identifier les ressources sensibles. Les ressources critiques sont facilement reconnaissables parce qu'elles contiennent des informations importantes (reliées aux données) ou stratégiques (reliées aux actions). Le fichier contenant tous les mots de passe peut être considéré comme « information importante » alors que l'accès à la création des processus pourrait être considéré comme « information stratégique ». Le contrôle de la gestion des processus permettrait, à une application malicieuse, de surcharger l'environnement de travail, d'occuper le processeur et de réussir à ne plus faire fonctionner l'environnement correctement. Toutes les attaques répertoriées depuis plusieurs an-

[3] Pour des raisons de simplicité, les termes filtre et driver auront la même signification et seront utilisés dans le reste du document.

nées permettent de cerner des ensembles communs de ressources visées. Même si les types d'attaque sont différents mais qu'ils visent une ressource identique, il est possible de croire que cette ressource peut être importante ou stratégique. Sur le site web de Microsoft[4], la compagnie avise les utilisateurs de la découverte de nouveaux types d'attaque faite contre ses produits et des solutions apportées. Elle fournit également la liste des applications possédant des bogues et leur solution. CERT[5] diffuse également des informations concernant les incidents de sécurité informatique. À partir d'information de cette nature, il est possible d'identifier les ressources sensibles pouvant être la cible d'attaques malicieuses. Les ressources identifiées sont les fichiers, les ports de communication, la base de registres et les processus incluant la gestion de la mémoire. Essentiellement, les attaques ou les bogues découverts atteignent une ou plusieurs de ces ressources.

Actions	Cibles	Ressources
Sonder ("Probe") Balayer ("Scan") Inonder ("Flood") Authentifier ("Authenticate") Éviter ("Bypass") Déguiser ("Spoof") Lire ("Read") Copier ("Copy") Voler ("Steal") Modifier ("Modify") Effacer ("Delete")	Compte d'usager ("Account") Processus ("process") Données ("Data") Composantes ("Component") Ordinateur ("Computer") Réseau ("Network") Inter-réseau ("Internetwork")	Fichiers Communications Base de registres Processus (mémoire)

Tableau 1: identification des ressources critiques

À noter que le système d'exploitation Windows NT possède des caractéristiques personnelles. Windows NT possède une ressource, la base de registres, où des informations importantes sont stockées (mots de passe et autre). Il est possible, par exemple, de modifier la base de registres de sorte qu'au démarrage de la machine une application s'exécute sans que l'usager le sache[6]. On voit donc l'importance de surveiller cette ressource. Howard et Longstaff [7] définissent un langage commun se rapportant à la sécurité informatique et duquel il est possible de mettre en relation une action et une cible. À partir de ces relations, il en découle des regroupements qui permettent de définir des ressources critiques. Le Tableau 1 montre les relations entre les actions et les cibles visées

[4] Site web dédié à la sécurité des produits Microsoft : **www.microsoft.com/security**
[5] Computer Emergency Response Team : **www.cert.org**
[6] En insérant les bonnes informations dans la clé **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce**, l'application malicieuse démarrera au lancement de Windows NT.

par ces actions pour aboutir à identifier les ressources critiques. L'Annexe III définit les termes utilisés dans le tableau.

3.3 Politiques de sécurité

Un aspect important de l'analyse dynamique est le fait de permettre au moniteur de guider ses actions et réactions selon des comportements précis. Jusqu'à présent, certaines applications permettent ce type de comportement. Par exemple, les « firewalls » surveillent les actions faites sur les ports de communication. Une action peut être définie comme la surveillance des ports par l'analyse des paquets reçus ou envoyés. La réaction, exécutée suite à une action commise et détectée par le « firewall », pourrait être de laisser passer ou bloquer le paquet. Les comportements réactifs sont définis par l'usager et guident le « firewall ».

Ces comportements correspondent à la définition de politiques de sécurité lesquelles guideront les décisions du moniteur. Autrement dit, chaque gardien ayant la responsabilité de surveiller une ressource devra réagir à une situation donnée en se basant sur les politiques de sécurité définies par l'usager. Actuellement, les produits offerts ne définissent que des politiques très sommaires et souvent limitées à une seule ressource.

Le problème soulevé par la conception d'un moniteur d'analyse dynamique est qu'il doit réagir à des situations parfois complexes. Si une application accède à un fichier et envoie le contenu sur un port de communication, et que le gardien surveillant les accès aux fichiers et le gardien surveillant les ports de communication aient respectivement la permission de laisser passer la requête, il est possible que cette séquence d'actions **LIRE – ENVOYER** corresponde à une séquence malicieuse. Une définition et une conception de politiques de sécurité rigoureuses ne devraient pas permettre une telle séquence malicieuse.

Schneider [27] propose une solution très intéressante en utilisant les automates. Le concept des automates implique une participation de tous les intervenants reliés à la surveillance. Ce concept permet de connaître l'état dans lequel se trouve une séquence d'actions. Selon l'exemple précédent, si la séquence se trouve dans l'état **LIRE**, il est possible de définir dans les politiques de sécurité que l'état suivant ne peut être **ENVOYER**. De cette manière, il serait impossible pour le gardien surveillant les ports de communication d'accepter la requête. La Figure 2 montre, à l'aide d'états et d'actions, les séquences permises.

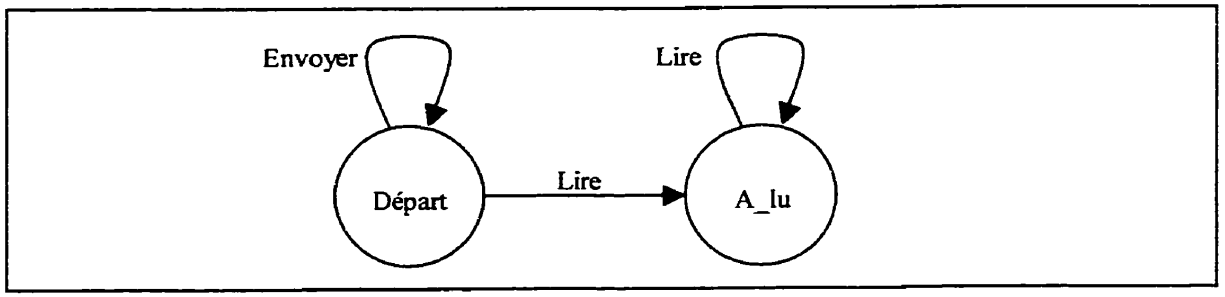


Figure 2: séquences d'exécution exprimées par des automates

Au départ, les actions permises sont **ENVOYER** et **LIRE**. Si l'action **ENVOYER** est exécutée, l'automate reste dans le même état **DÉPART**. Tant que l'action **ENVOYER** sera exécutée, l'automate restera dans le même état. Si l'action **LIRE** est exécutée, l'automate change d'état et se retrouve dans l'état **A_LU**. La seule action permise dans cet état sera **LIRE**. Toute autre action enverra l'automate dans un état **BAD** qui n'est pas représenté dans la figure. Cet état **BAD** est la conséquence de toutes les actions commises et non permises dans un état donné.

En appliquant l'exemple des automates à un moniteur, on pourrait exprimer des comportements similaires. Autrement dit, dès le début de la surveillance du moniteur, celui-ci laisse passer les actions **ENVOYER**. Dès que l'action **LIRE** est commise, il y a changement d'état et le moniteur ne peut laisser passer l'action **ENVOYER**; celle-ci n'étant plus permise.

Schneider [27] définit des automates de sécurité. Ces automates de sécurité sont définis ainsi :

- Un ensemble (fini) Q d'états,
- Un ensemble $Q_0 \subseteq Q$ d'états initiaux,
- Un ensemble (dénombrable) I de symboles d'entrée, et
- Une fonction de transition, $\delta \subseteq (Q \times I) \rightarrow 2^Q$

Les symboles dans I peuvent correspondre à des états du système, des actions atomiques, des actions de haut-niveaux du système ou à une paire état/action. Pour traiter une séquence $s_1 s_2 \ldots$ de symboles d'entrée, l'automate débute en mettant son état initial à Q_0 et lit la séquence, un symbole à la fois. À chaque symbole lu, l'automate change son état courant Q' à un état Q'' si

$$Q'' = \bigcup_{q \in Q'} \delta(q, s_i)$$

Si Q'' est vide, l'entrée est rejetée; sinon l'entrée est acceptée.

Un problème se pose avec la définition précédente. Il peut arriver que des applications légales et reconnues comme les logiciels de courrier électronique ne puissent plus fonctionner à cause de ces automates de sécurité. Schneider propose un raffinement qui permet d'inclure des attributs aux automates. Ainsi, on retrouve le principal (prin), l'opération à exécuter (oper) et l'objet sur lequel le principal et l'opération se rapporte (obj). On note *A(prin,oper,obj)* le prédicat de transition, une valeur Booléenne ayant ***I*** comme domaine. De plus, selon Schneider, on peut ajouter des étiquettes de sécurité ordonnées à chaque attribut. Par exemple, un ordonnancement possible des attributs pourrait être :

top secret > secret > sensible > non-classifié

En utilisant cet ordonnancement et en supposant les deux opérations suivantes – **LIRE** et **ÉCRIRE**, on pourrait exprimer les restrictions suivantes sur l'exécution de ces opérations ainsi :

- Un principal p avec l'étiquette $\lambda(p)$ à la permission de **LIRE(*F*)**, où F a une étiquette $\lambda(F)$, si $\lambda(p) \geq \lambda(F)$.
- Un principal p avec l'étiquette $\lambda(p)$ à la permission d'**ÉCRIRE(*F*)**, où F a une étiquette $\lambda(F)$, si $\lambda(F) \geq \lambda(p)$.

À partir des travaux de Schneider, il est possible de spécifier une politique de sécurité où *A(prin,oper,obj) est instancié par :*

$$
\begin{aligned}
A(prin,oper,obj) : \quad & (oper = \textbf{LIRE} \wedge \lambda(prin) \geq \lambda(obj)) \\
\vee \quad & (oper = \textbf{ÉCRIRE} \wedge \lambda(obj) \geq \lambda(prin)) \\
\vee \quad & (oper \notin \{\textbf{LIRE},\textbf{ÉCRIRE}\}
\end{aligned}
$$

En utilisant ce concept, il est maintenant possible de définir un ensemble de politiques de sécurité portant sur plusieurs ressources sensibles. En créant un langage de programmation permettant de définir des politiques de sécurité, l'utilisateur pourra exprimer ses préférences. Celles-ci peuvent être formulées avec plus de précision et être plus riches de contenu. Actuellement, il n'existe aucun produit pouvant satisfaire ce type de politiques de sécurité.

3.4 Codes malicieux

Un des aspects de ce travail est la connaissance et l'apprentissage de la notion de codes malicieux. La détection de codes malicieux est l'essence même du moniteur. Ce dernier doit identifier toute séquence de codes malicieux s'exécutant ou sur le point de s'exécuter. Le problème est de définir ce qu'est un code malicieux. Cette section met en

relief les sources possibles de codes malicieux, les caractéristiques définissant ce qu'est un code malicieux. La section 3.4.3 propose un tableau représentant l'ensemble des informations recueillies.

3.4.1 Spécification des sources

En général, l'usager sait après coup qu'il a été victime d'un problème ou de l'insertion involontaire de codes malicieux. C'est le cas des virus entre autre. Il est même possible qu'il ne le sache jamais s'il n'a pas d'outil détectant la présence de virus. Cet exemple illustre à quel point les utilisateurs doivent se prémunir contre l'insertion de codes malicieux et d'avoir des moyens de prévention et d'élimination. L'insertion de codes malicieux peut se faire de différentes façons. L'utilisation d'Internet peut être une source d'infection si l'usager n'est pas à l'affût de ce type de problème. En téléchargeant une page Web contenant des applications mobiles (applets java, ActiveX, ...), l'usager s'expose à recevoir du code malicieux, ne sachant pas que ces applications mobiles pouvaient contenir une séquence malicieuse. Dans ce cas-ci, la source du problème est externe à l'environnement de travail

Dans les entreprises possédant des ordinateurs nécessaires au travail des employés, il est possible qu'un de ceux-ci insère volontairement du code malicieux par vengeance personnelle (congédiement, conflits personnels, par pur plaisir, etc.). Cette source d'infection provient de l'intérieur et prétend être volontaire puisqu'une personne ou plusieurs personnes sont impliquées.

Une autre source d'insertion peut provenir de l'utilisation de logiciels commerciaux, COTS (« Commercial-Off-The-Shelf »). De plus en plus de personnes utilisent des logiciels provenant de différents endroits et de fabricants. Dans ce cas-ci, il est possible que les concepteurs aient inséré des séquences de codes malicieux dans leurs produits. L'exécution de ces séquences de codes malicieux peut avoir différents niveaux de conséquence allant de conséquences inoffensives aux conséquences catastrophiques. Alors pourquoi utiliser des COTS s'ils peuvent être dangereux? Il existe plusieurs raisons qui poussent les gens ou les entreprises à les utiliser. Premièrement, la fabrication d'un logiciel est très coûteuse et longue. Pour des raisons d'économie, il apparaît évident d'acquérir un logiciel commercial offrant les caractéristiques recherchées à moindre coût. Deuxièmement, la maturité des produits permet de supposer qu'il y a moins de vices de conception et qu'il est plus fonctionnel qu'un produit maison nouvellement conçu. Troisièmement, la réutilisation de certains programmes (applets Java, ActiveX, ...) permet une économie de temps et d'argent ce qui peut pousser des concepteurs à les utiliser même si ces programmes contiennent des séquences de codes malicieux.

Comme on le constate, l'usager n'est pas à l'abri des séquences de code malicieux. Les sources d'infection proviennent de différents endroits et obligent les usagers à se prémunir de moyens pour les contrer.

3.4.2 Caractéristiques du code malicieux

Sachant qu'il existe des séquences de codes malicieux, il faut que l'usager sache les reconnaître. Pour les identifier, il est essentiel de définir ce qu'est du code malicieux. Selon Bergeron [2][7], un code malicieux est un fragment de programme qui peut affecter, ou potentiellement peut aider à affecter, la confidentialité, l'intégrité, les flots de contrôle et de données et les fonctionnalités du système sans la connaissance et le consentement explicite de l'usager. À la lecture de cette définition, une séquence ou fragment de codes malicieux peut avoir différentes formes et peut s'attaquer à différentes cibles.

Une séquence de codes malicieux pourrait avoir la forme d'un programme qui envoie des informations sur un canal de communication (communication cachée). Une autre séquence pourrait essayer d'intercepter des informations pour découvrir des mots de passe (attaque sur les mots de passe). Un autre programme ou fragment de programme pourrait se déclencher à un moment précis et créer des problèmes (bombe logique ou temporelle). D'autres fragments pourraient essayer d'obtenir les privilèges de l'administrateur de la machine visée (attaque sur les privilèges).

Il existe plusieurs types de codes malicieux. Chacun des programmes malicieux vise une cible précise et a une forme particulière. Les chevaux de Troie[8] en sont un exemple de formes particulières. De même; les virus qui s'attachent au programme ont une forme différente de ceux qui s'attachent à des documents Word. L'usager est donc confronté à lutter contre une espèce particulière d'ennemis ayant plusieurs formes.

3.4.3 Tableau récapitulatif

Chia [4] propose un tableau démontrant la provenance de codes malicieux. L'Annexe IV explique en détail ce tableau. Ce tableau exprime bien le problème du code malicieux et répond aux trois questions se rapportant aux codes malicieux à savoir :

- Comment est-il entré dans le système (« Genesis »)?
- Quand est-il entré dans le système (« Time of introduction »)?
- À quel endroit dans le système s'est-il manifesté (« Location »)?

---

[7] La définition provient d'un article en anglais et a été traduite pour conserver le langage utilisé dans ce document.

[8] Voir [2] pour une description exhaustive des différentes formes de codes malicieux.

<table>
<tr><td rowspan="11">Genesis</td><td rowspan="7">Intentional</td><td rowspan="4">Malicious</td><td rowspan="2">Trojan Horse</td><td>Non-Replicating</td></tr>
<tr><td>Replicating (virus)</td></tr>
<tr><td colspan="2">Trapdoor</td></tr>
<tr><td colspan="2">Logic/Time Bomb</td></tr>
<tr><td rowspan="3">Non-Malicious</td><td rowspan="2">Covert Channel</td><td>Storage</td></tr>
<tr><td>Timing</td></tr>
<tr><td colspan="2">Other</td></tr>
<tr><td rowspan="5">Inadvertent</td><td colspan="3">Validation error (Incomplete/Inconsistent)</td></tr>
<tr><td colspan="3">Domain Error</td></tr>
<tr><td colspan="3">Serialization/aliasing (Including TOCTTOU Errors)</td></tr>
<tr><td colspan="3">Identification/Authentication Inadequate</td></tr>
<tr><td colspan="3">Boundary Condition violation</td></tr>
<tr><td rowspan="5">Time of Introduction</td><td colspan="3">Other ecploitable Logic Error</td></tr>
<tr><td rowspan="3">During Development</td><td colspan="3">Requirements/Specification/Design</td></tr>
<tr><td colspan="3">Source Code</td></tr>
<tr><td colspan="3">Object Code</td></tr>
<tr><td colspan="4">During Maintenance</td></tr>
<tr><td colspan="4">During Operation</td></tr>
<tr><td rowspan="11">Location</td><td rowspan="10">Software</td><td rowspan="7">Operating System</td><td colspan="2">System Initialization</td></tr>
<tr><td colspan="2">Memory Management</td></tr>
<tr><td colspan="2">Process management/Scheduling</td></tr>
<tr><td colspan="2">Device management (Including I/O, networking)</td></tr>
<tr><td colspan="2">File Management</td></tr>
<tr><td colspan="2">Identification/Authentication</td></tr>
<tr><td colspan="2">Other/Unknown</td></tr>
<tr><td rowspan="2">Support</td><td colspan="2">Priviliged Utilities</td></tr>
<tr><td colspan="2">Unprivileged Utilities</td></tr>
<tr><td colspan="3">Application</td></tr>
<tr><td colspan="4">Hardware</td></tr>
</table>

Tableau 2: taxonomie du code malicieux

3.5 Produits de détection dynamique

Le marché informatique regorge de produits se rapportant à la sécurité. Certains domaines sont soumis à de nombreux produits jouant sensiblement le même rôle. Depuis l'utilisation massive d'Internet, plusieurs entreprises ont mis sur le marché des produits

reliés aux communications en réseau. Tous sont d'accord à vouloir sécuriser les échanges afin de protéger les environnements contre des attaques malicieuses provenant de l'extérieur. Qu'en est-il des attaques provenant de l'intérieur? Peut-on garantir qu'un logiciel (COTS) ne divulgue pas d'informations sans le consentement de l'utilisateur? Certains produits offrent la possibilité de limiter les accès vers l'extérieur (« firewall » personnel) en surveillant les requêtes internes. La majorité des produits en réseautique veulent protéger toutes tentatives d'accès vers l'intérieur contre les agents extérieurs mais aucun produit ne se préoccupe des activités internes de l'environnement. Les communications en réseau étant un secteur chaud de l'informatique[9], beaucoup d'efforts sont déployés pour sécuriser ce domaine. Les autres ressources sont mises de côté et très peu d'efforts sont faits pour les surveiller. Plusieurs recherches sont entreprises pour détecter des comportements malicieux provenant de l'extérieur comme la détection d'intrusion par exemple. Si la menace ne provient pas des accès réseaux mais par l'installation de logiciels commerciaux (COTS), aucun produit ne peut garantir la sécurité de l'environnement. Le Tableau 3 montre la liste des produits testés jusqu'à maintenant pour chaque ressource identifiée. Il est à noter que cette liste n'est pas exhaustive puisqu'il s'ajoute de nouveaux produits régulièrement. Ce tableau est présenté pour montrer qu'il existe des produits pour chaque ressource critique.

Ressource	Nom	Compagnie	Description	Commentaires
Base de registres	Grr! (Greyware Registry Rearguard)	Greyware Automation Product **www.greyware.com**	▪ Protège la base de registres et les répertoires et fichiers de démarrage. ▪ Permet de bloquer les accès à ces objets.	▪ Permet aussi de contrôler d'autres fichiers. ▪ Produit commercial.
	Regmon v4.24	Sysinternals **www.sysinternals.com**	▪ Filtre les appels fait à la base de registres.	▪ Aucune capacité à contrôler les accès à la base de registres. ▪ Code source disponible.

[9] Ce site web, **http://www.ntsecurity.net/**, contient beaucoup d'informations reliées à la sécurité de Windows NT

Ressource	Nom	Compagnie	Description	Commentaires
Fichiers	Esafe	Aladdin Knowlegde Systems **www.esafe.com**	▪ Protège l'environnement contre les applications mobiles (applet, ActiveX et autres) en surveillant les accès aux disques.	▪ Inclus un module anti-virus et un « firewall » personnel.
	Filemon v4.28	Sysinternals **www.sysinternals.com**	▪ Filtre les appels aux fichiers.	▪ Aucune capacité à contrôler les accès aux fichiers. ▪ Code source disponible.
Processus	PMon v1.0	Sysinternals **www.sysinternals.com**	▪ Surveillance des processus.	▪ Fonctionnalités de monitorage exclusivement
	Spy++	Microsoft Laboratories & Jeffrey M. Richter	▪ Surveillance des processus.	▪ Fonctionnalités de monitorage exclusivement
	Process viewer	Produit inclus dans Visual C++ 6.0	▪ Surveillance des processus.	▪ Possibilité d'arrêter un processus manuellement.
Communications	SurfinShield	Finjan Software Inc. **www.finjan.com**	▪ Contrôle des applications mobiles ▪ Définition de certaines politiques de sécurité et de réactions.	▪ Nécessite une mise à jour constante de la base de données contenant la liste des applications mobiles
	TCPView v1.0	Sysinternals **www.sysinternals.com**	▪ Filtre les communications.	▪ Aucune capacité à contrôler les accès aux ports de communication. ▪ Code source disponible.
Autre	LogCaster	RippleTech **www.rippletech.com**	▪ Surveille et fournit des informations en temps réels sur l'état de la machine	▪ Signalement des problèmes après coup seulement

Ressource	Nom	Compagnie	Description	Commentaires
	STAT	Harris Corporation /STAT **www.statonline.com/main.html**	▪ Surveille les vulnérabilités connues de l'environnement Windows NT	▪ Nécessite une mise à jour continuelle des vulnérabilités

Tableau 3: liste des produits évalués

PARTIE II – LE MONITEUR

Chapitre 4

ARCHITECTURE DU MONITEUR

"Les passions sont les vents qui enflent les voiles du navire;
elles le submergent quelquefois,
mais sans elles il ne pourrait voguer."
Voltaire

La conception du moniteur passe par l'apprentissage de l'environnement Windows NT et la définition des ressources qui devront être mises sous surveillance. Le choix de l'environnement Windows NT repose essentiellement sur les tendances du marché. De plus en plus d'utilisateurs se tournent vers ce système d'exploitation. En développant des outils permettant de contrôler cet environnement, la surveillance ou le monitorage sera grandement facilité. Pour construire ce type d'outil, il est nécessaire de connaître l'environnement Windows NT. Naturellement, ce produit étant de nature commerciale, toutes informations se rapportant à sa conception, à son fonctionnement et à la définition de ses structures sont difficiles à obtenir. La première partie de ce chapitre présente l'architecture du système d'exploitation Windows NT. Cette couverture permet de cerner et de visualiser l'environnement dans lequel le travail sera effectué.

La deuxième partie identifie les ressources jugées pertinentes au monitorage. Il est important de noter que cette partie présente toutes les ressources identifiées mais que seulement deux ressources (fichiers et ports de communication) sont mises en évidence par des explications détaillées.

À la lumière des informations recueillies jusqu'à présent, la troisième partie propose un modèle de moniteur d'analyse dynamique permettant le contrôle et la surveillance de l'environnement Windows NT. Cette partie fait l'ébauche des caractéristiques que devra posséder l'application principale. Ces caractéristiques sont définies et présentées afin de regrouper les efforts de recherche en un produit concret.

4.1 Environnement de travail

Cette section présente l'architecture, les concepts et la définition des termes nécessaires à la compréhension du système d'exploitation Windows NT. Elle sera utile au moment de la conception des filtres (pilotes). Une discussion approfondie sur les filtres est présentée au chapitre 5.

4.1.1 Principales caractéristiques de Windows NT

La littérature fournit des informations sur les objectifs de conception de Windows NT. Ce qui est important de retenir sont les finalités ou les éléments mis en place pour réaliser ces objectifs. Ces éléments fournissent des informations importantes qui ont une incidence sur la conception d'une application de monitorage.

- L'architecture supporte les concepts de construction par couche et client-serveur. Cette façon de présenter le système permet de le rendre extensible (en ajoutant ou modifiant des couches) et de créer des applications modulaires pouvant être facilement ajoutées ou enlevées.
- Chaque processus possède un espace de mémoire virtuelle protégée. Aucune autre application ne peut accéder à l'espace mémoire qui n'est pas le sien.
- Le système d'exploitation possède deux modes — Usager et Noyau. Les applications tournent en mode Usager ce qui signifie qu'elles n'ont pas accès directement aux fonctionnalités du mode Noyau. À l'opposé, les fonctionnalités du mode Noyau peuvent accéder à tout l'environnement ce qui lui confère des privilèges qui doivent être maîtrisés.
- Implantation d'un nouveau gestionnaire de fichiers, NTFS (NT File System), qui permet d'accroître la sécurité des fichiers.
- Apparition du concept de la base de registres permettant de stocker des informations sur le système et les composantes rattachées au système.

4.1.2 Présentation de l'environnement Windows NT

La Figure 3 présente l'environnement Windows NT. Comme on peut le constater, la section noyau (« kernel mode ») montre un modèle en couche. Chaque module joue un rôle précis qu'il est important de connaître pour identifier les intervenants durant l'exécution du système d'exploitation.

- **Mode usager (user mode) :**
  - **Sous-systèmes d'environnement (Win32, OS/2 et POSIX).** Environnements définis pour répondre aux applications de type Win32, POSIX ou OS/2. L'environnement Win32 est obligatoire pour que Windows NT fonctionne.
  - **Sous-système de sécurité (« Security subsystem »).** Le sous-système de sécurité a été défini pour contrôler les accès aux différents objets et s'assurer que les applications ou les usagers ont les droits correspondants.

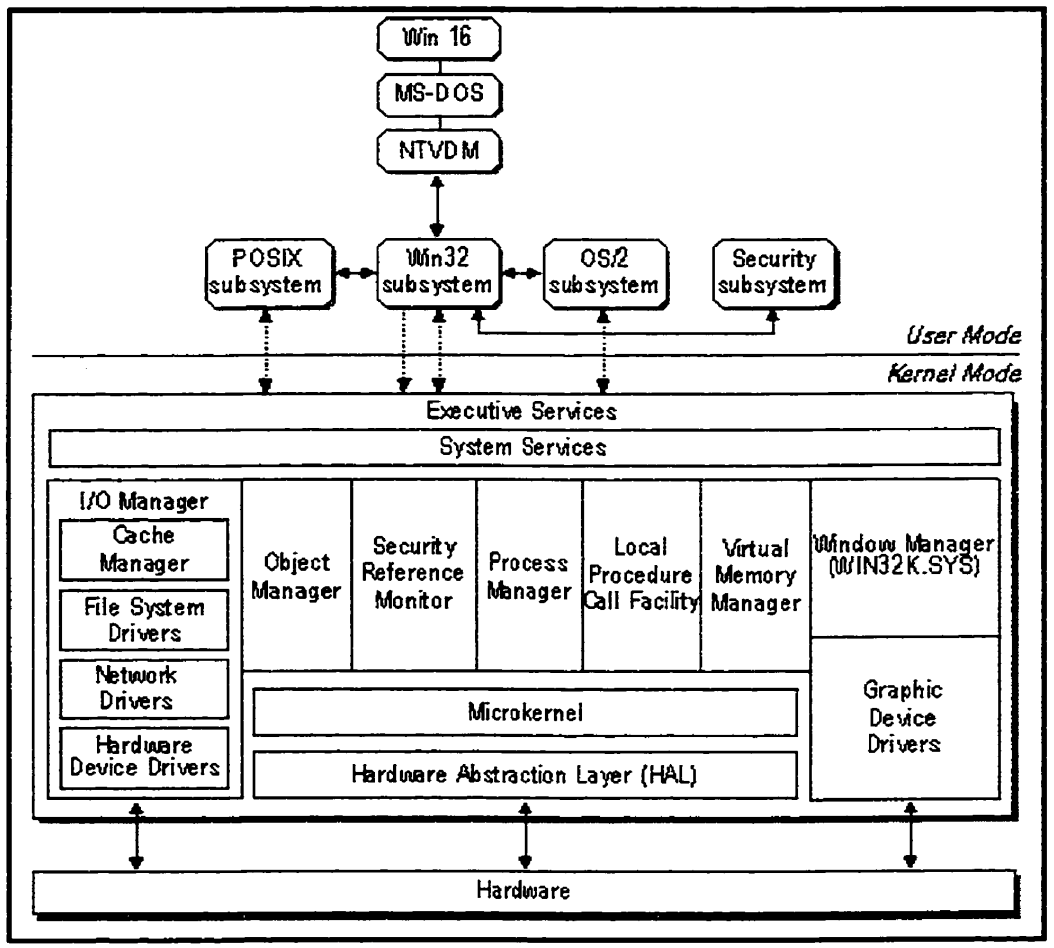


Figure 3: architecture de Windows NT[10]

- **Applications.** Les applications correspondent aux différents programmes qui peuvent fonctionner sous les différents sous-systèmes supportés par Windows NT. Certaines applications portent le nom de « services » et agissent comme serveur.

- **Mode noyau (kernel mode) :**
  - **Services système (« System services »)**: couche supérieure du noyau agissant comme interface entre le mode usager et le mode noyau.
  - **Gestionnaire de mémoire virtuelle (« Virtual Memory Manager »)** : composante gérant l'espace d'adressage de chaque processus.

[10] Cette figure provient du document d'aide de MicrosoftWindows NT Server Resource Kit [15].

- **Utilitaire d'appel local de procédures (« LPC – Local Procedure Call »)** : composante faisant passer les messages entre les processus clients et les processus serveurs sur un même ordinateur.
- **Gestionnaire de processus (« Process Manager »)** : composante qui crée, gère et détruit les processus ou les « threads » nécessaires au fonctionnement du système d'exploitation.
- **Moniteur des références de sécurité (« SRM – Security Reference Monitor »)** : composante surveillant les opérations du système d'exploitation. Elle prévient l'accès direct aux objets par les usagers ou les processus. Elle valide les accès aux objets.
- **Gestionnaire d'objets (Object Manager »)** : composante qui crée, gère et détruit les objets représentant les ressources (fichiers, processus, port, etc.) du système d'exploitation.
- **Gestionnaire des entrées/sorties (« I/O Manager »)** : composante implémentant les divers gestionnaires de périphériques nécessaires (pilotes).
- **Noyau (Microkernel »)** : application qui exécute les opérations fondamentales du système d'exploitation, détermine l'utilisation du ou des processeurs et s'assure qu'ils sont utilisés prudemment. Ordonnancement des « threads », synchronisation des processeurs, gestion des interruptions et gestion des exceptions sont des fonctions de base que doit effectuer le noyau.
- **Couche d'abstraction du matériel (« HAL – Hardware Abstraction Layer »)** : interface de bas niveau entre le matériel et le système d'exploitation. Cette couche permet d'obtenir la portabilité souhaitée telle que mentionnée précédemment.

La Figure 4 montre tous les intervenants ayant un rôle à jouer lorsqu'on active un pilote. Tous les modules du noyau (NTOSKRNL.EXE) sont mis à contribution sauf le LPC. Le LPC n'est pas actif dans l'exécution d'un pilote. Ce module est utilisé pour transférer de l'information entre deux applications en mode usager. En mode noyau, il n'est pas utilisé.

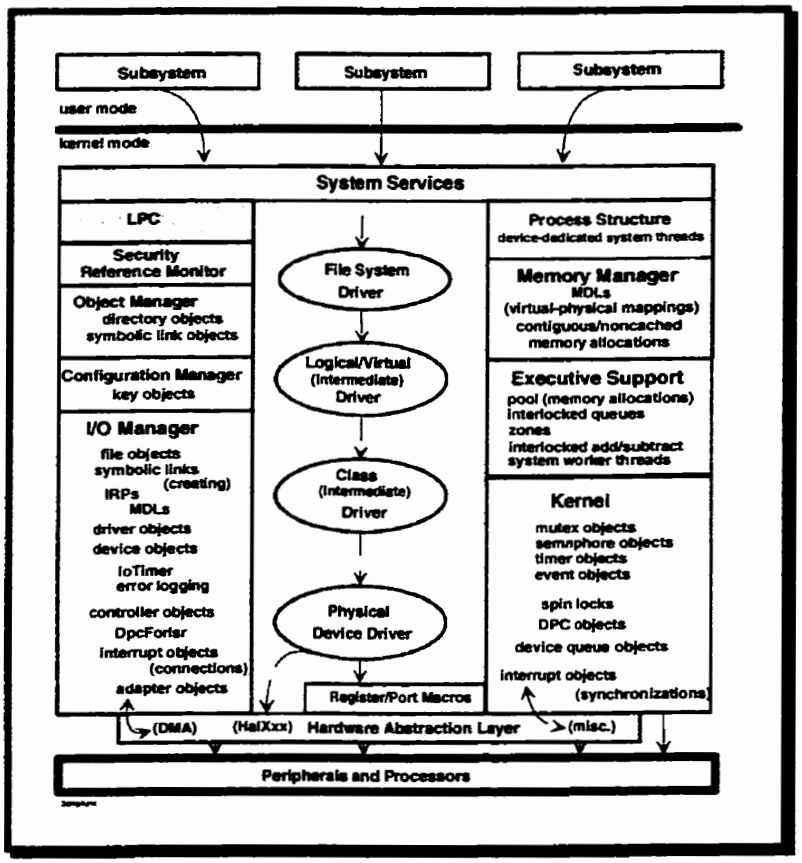


Figure 4 : aperçu des services fournis à un pilote[11]

4.2 Ressources critiques

Le rôle principal d'un système d'exploitation est de gérer les différentes ressources mises à sa disposition. Essentiellement, les ressources sont semblables d'un système d'exploitation à un autre. Certains systèmes peuvent avoir leurs particularités qui devront être définies pour déterminer si celles-ci doivent faire partie des ressources à surveiller. Dans le cas du système d'exploitation Windows NT, les ressources identifiées et susceptibles d'être soumises à des attaques de codes malicieux ont été recensées. Le Tableau 4 énumère les types de ressources sensibles ainsi que des exemples de requêtes possibles. D'ailleurs, le « security manager » de JAVA [30] suggère la surveillance de ces ressources. À noter que la liste des ressources et requêtes pourra s'allonger au fur et à mesure que progressera la conception du moniteur d'analyse dynamique et de la définition des politiques de sécurité.

[11] Idem.

Types de ressources	Activités
Fichiers	**Créer** un fichier.
	Renommer un fichier.
	Copier un fichier.
	Transférer des fichiers entre zones protégées et non-protégées.
	Effacer un fichier.
	Lire un fichier.
	Écrire dans un fichier.
Communication	**Accepter** une communication sur un port de communication.
	Ouvrir une communication sur un port de communication.
	Attendre une communication sur un port de communication.
Registres	**Modifier** les propriétés du système.
	Accéder aux propriétés du système.
Processus	**Modifier** l'état d'un « thread ».
	Créer des « threads ».
	Arrêter une application.
	Allouer de la mémoire.
	Créer un nouveau « process ».

Tableau 4: ressources à surveiller

Tel que mentionné précédemment, le tableau présente les ressources sensibles répertoriées dans l'environnement Windows NT qui devront être surveillées. Les sections suivantes passent en revue chacune des ressources et identifient les moyens à mettre en place pour surveiller et contrôler chacune des ressources.

4.2.1 Ressource Fichiers

L'accès aux fichiers ou à la structure de fichiers doit être surveillé et contrôlé pour éviter toute requête pouvant causer du tort. Dans l'environnement Windows NT, le Gestionnaire E/S doit répondre à toutes les requêtes E/S qui peuvent survenir. Il a aussi le rôle de mettre en place tous les pilotes nécessaires à la gestion des fichiers et de diriger les requêtes entre les pilotes. Les deux sous-sections suivantes présentent le Gestionnaire E/S et le processus d'échange entre les pilotes.

- **Gestionnaire E/S (« I/O Manager »)**

Le Gestionnaire E/S est une composante de l'exécutif (NTOSKRNL.EXE) qui convertit les requêtes E/S (autant les requêtes en mode usager qu'en mode noyau) en une séquence d'appel aux diverses fonctions des pilotes. À travers l'utilisation d'une interface formelle, le Gestionnaire E/S est capable de communiquer à tous les pilotes de la même façon. Cette approche n'oblige pas le Gestionnaire E/S à connaître tout le fonctionnement interne de chaque pilote. En somme, le Gestionnaire E/S gère les pilotes et propose une interface pour échanger de l'information entre les intervenants en mode noyau.

- **Communication inter pilote**

La Figure 5 propose l'exemple du cheminement d'un appel E/S. Une application veut lire un fichier (`ReadFile()`). Cette fonction est interprétée par l'API Win32 et est transformée en fonction `NtReadFile()` au niveau de KERNEL32.DLL. `NtReadFile()` est à son tour transformée en interruption dans NTDLL.DLL. L'Exécutif (NTOSKRNL.EXE) avec l'aide du Gestionnaire E/S réagit à l'interruption en appelant le pilote correspondant. Ce dernier exécute la requête et retourne de l'information à l'appelant. L'information retournée refait le chemin inverse pour se retrouver à l'application initiale.

À partir de ce cheminement, on constate qu'il est possible d'intercepter des informations et de les traiter. Plusieurs solutions existent pour intercepter les informations. Une première façon serait de capturer tous les appels au niveau de NTDLL.DLL en mode Usager. Le problème majeur est qu'il n'est pas possible de pouvoir intercepter tous les appels. Les appels faits en mode Noyau ne passent pas par ce chemin. Une deuxième façon, pour éviter les contournements, serait d'aller se positionner le plus bas possible, près de la ressource à surveiller, afin d'intercepter tous les appels E/S. Les filtres ou pilotes intermédiaires permettent ce positionnement.

Lorsque le Gestionnaire E/S reçoit une requête, il la divise en une séquence d'opérations de base que les pilotes comprennent. Le Tableau 5 montre la liste des opérations de base qui sont comprises par les pilotes. Pour générer la séquence, le Gestionnaire E/S utilise une structure de données contenant toutes les informations nécessaires à son traitement.

L'IRP correspond à l'endroit où le Gestionnaire E/S stocke des informations pour exécuter une requête E/S. Lorsqu'un « thread » fait un appel au Gestionnaire E/S, ce dernier construit un IRP qui représente l'opération à exécuter et qui progresse à travers le système E/S.

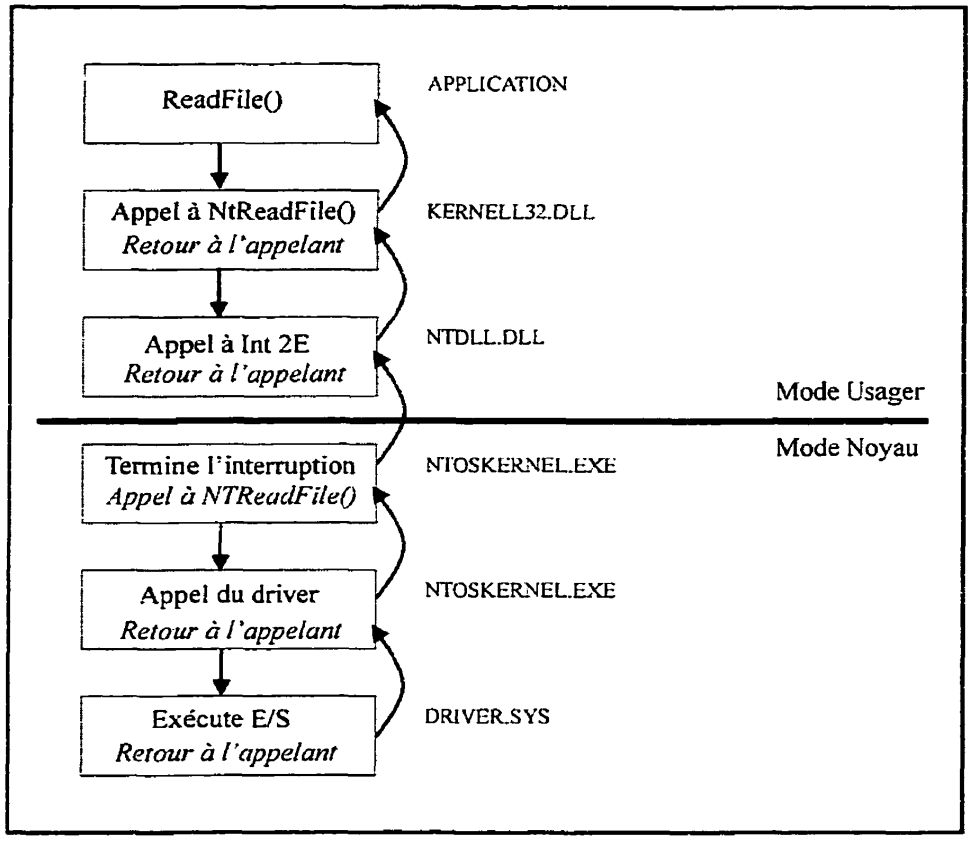


Figure 5: cheminement d'un appel E/S

Cette structure s'appelle un IRP. L'Annexe V présente la structure d'un IRP ainsi que d'autres structures de données couramment employées dans le IRP.

```
#define IRP_MJ_CREATE                   0x00
#define IRP_MJ_CREATE_NAMED_PIPE        0x01
#define IRP_MJ_CLOSE                    0x02
#define IRP_MJ_READ                     0x03
#define IRP_MJ_WRITE                    0x04
#define IRP_MJ_QUERY_INFORMATION        0x05
#define IRP_MJ_SET_INFORMATION          0x06
#define IRP_MJ_QUERY_EA                 0x07
#define IRP_MJ_SET_EA                   0x08
#define IRP_MJ_FLUSH_BUFFERS            0x09
#define IRP_MJ_QUERY_VOLUME_INFORMATION 0x0a
#define IRP_MJ_SET_VOLUME_INFORMATION   0x0b
#define IRP_MJ_DIRECTORY_CONTROL        0x0c
#define IRP_MJ_FILE_SYSTEM_CONTROL      0x0d
#define IRP_MJ_DEVICE_CONTROL           0x0e
#define IRP_MJ_INTERNAL_DEVICE_CONTROL  0x0f
#define IRP_MJ_SHUTDOWN                 0x10
#define IRP_MJ_LOCK_CONTROL             0x11
#define IRP_MJ_CLEANUP                  0x12
#define IRP_MJ_CREATE_MAILSLOT          0x13
#define IRP_MJ_QUERY_SECURITY           0x14
```

#define IRP_MJ_SET_SECURITY	0x15
#define IRP_MJ_QUERY_POWER	0x16
#define IRP_MJ_SET_POWER	0x17
#define IRP_MJ_DEVICE_CHANGE	0x18
#define IRP_MJ_QUERY_QUOTA	0x19
#define IRP_MJ_SET_QUOTA	0x1a
#define IRP_MJ_PNP_POWER	0x1b
#define IRP_MJ_MAXIMUM_FUNCTION	0x1b

Tableau 5: opérations de base[12]

La Figure 6 représente la description des étapes lors d'un appel E/S.

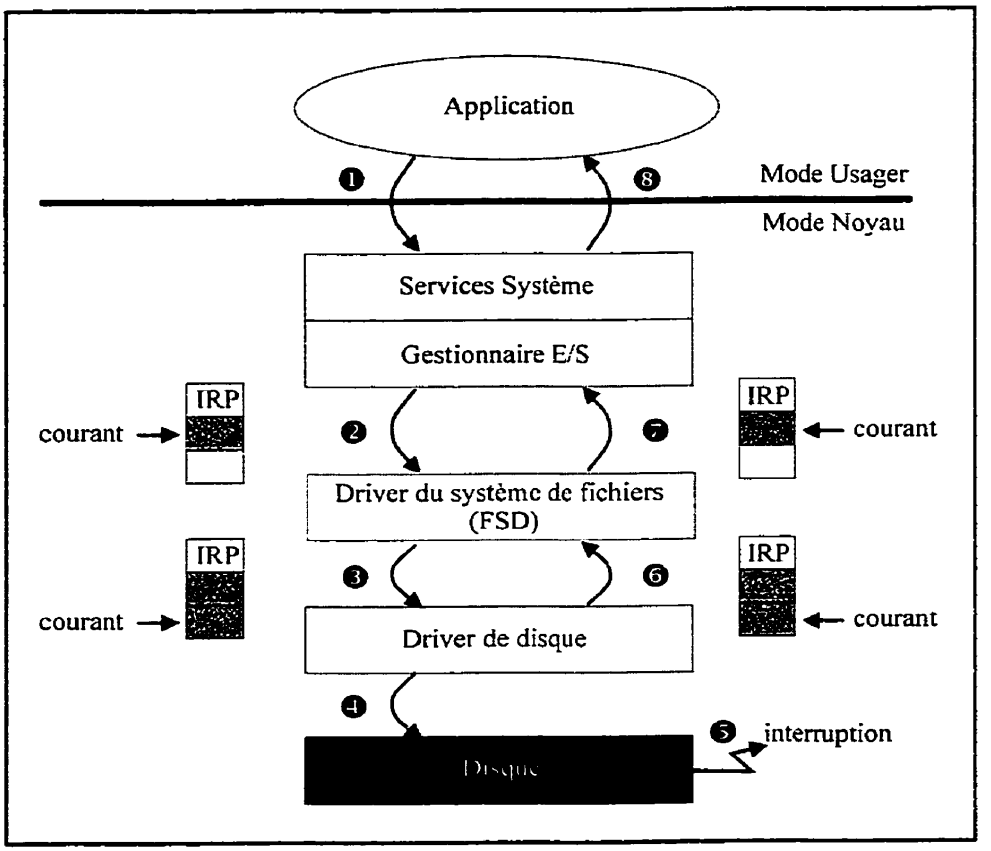


Figure 6: description des étapes lors d'un appel E/S

[1] Appel au service E/S.

[2] Le Gestionnaire E/S crée un IRP, remplit la première location de la pile et appelle le pilote du système de fichiers.

[12]« Define the major function codes for IRPs. The lower 128 codes, from 0x00 to 0x7f are reserved to Microsoft. The upper 128 codes, from 0x80 to 0xff, are reserved to customers of Microsoft. » Tiré de Microsoft DDk [13]

[3] Le pilote du système de fichiers remplit la deuxième location de la pile et appelle le pilote de disque.

[4] Le pilote de disque envoie les données contenues dans le IRP au disque.

[5] Une interruption est levée par le disque.

[6] Le pilote de disques traite l'interruption et exécute certains traitements au retour du IRP.

[7] Le pilote du système de fichiers exécute les opérations de nettoyage nécessaire.

[8] Les résultats sont retournés à l'appelant par l'intermédiaire de son espace d'adressage.

4.2.2 Ressource Communications

L'architecture réseau de Windows NT fonctionne selon le modèle de référence OSI (« Open Systems Interconnexion ») à sept couches. La Figure 7 présente le modèle de référence OSI. Le principe, dans le modèle, est d'établir l'indépendance des couches les unes par rapport aux autres et de facilité la conception et la réalisation des protocoles et des règles qui les constituent. Les relations qui existent entre les couches sont de deux types :

- **Relations d'interface**
- **Relations de protocole**

Dans un système d'information fonctionnant selon le modèle OSI, les relations d'interface vont intervenir d'une façon verticale (hiérarchique) entre les sept couches tandis que les relations de protocole ne sont pas hiérarchiques. Les relations de protocole sont plutôt des relations qui interagissent entre même couche définissant ainsi les formats des messages et les règles d'échanges.

- **Couche Application (« Application layer »)** : les programmes de cette couche offrent les interfaces et les services qui constituent des points d'entrée des programmes utilisateurs dans le modèle OSI.

- **Couche Présentation (« Presentation layer »)** : la couche Présentation sert de traducteur de données au réseau. Cette couche, sur l'ordinateur émetteur, traduit les données envoyées par la couche application dans un format commun à l'ordinateur récepteur ou la couche présentation traduit le format commun reçu à un format connu de la couche application. Autrement dit, elle assure une compréhension syntaxique entre les utilisateurs en gérant les formats de données à échanger

et effectue les transformations nécessaires sur les structures de données pour quelles soient compréhensibles entre matériels hétérogènes.

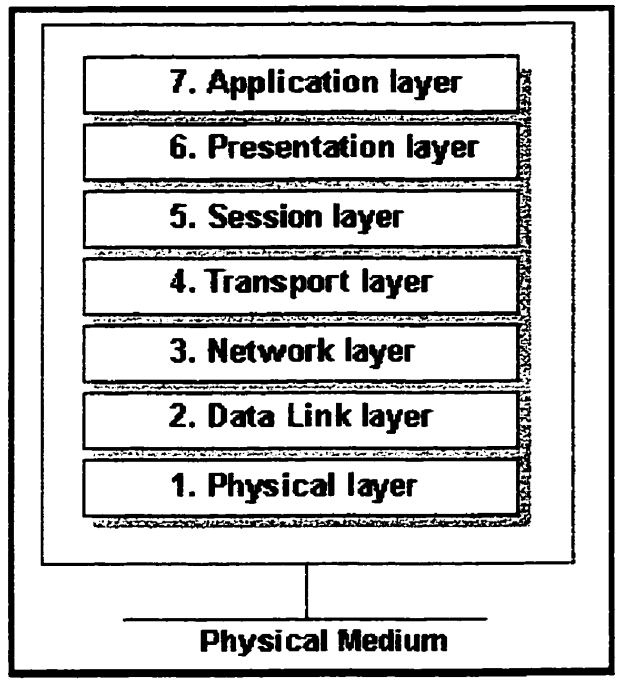


Figure 7: les sept couches du modèle OSI[13]

- **Couche Session (« Session layer »)** : la couche Session permet à des utilisateurs, sur différentes machines, d'établir des sessions entre elles. Une session permet le transport ordinaire de données, tout comme la couche transport, mais fournit également quelques services utiles à quelques applications. Une session pourrait être utilisée pour permettre à un usager d'enregistrer à distance ou pour transférer un fichier entre deux machines.

- **Couche Transport (« Transport layer »)** : les modules de cette couche fournissent les règles de contrôle, de bout en bout, relatives à un transfert d'information entre deux stations. Elle permet à la machine destinataire de communiquer directement avec la machine source. Elle s'assure de la transition entre les couches de traitement 5,6,7 et les couches de transmission 1,2 et 3 et qu'aucune machine intermédiaire n'ait failli.

- **Couche Réseau (« Network layer »)** *:* cette couche contrôle l'exécution du sous-réseau. Elle détermine le chemin que le paquet de données va prendre pour se rendre à son destinataire. Elle inclut le concept d'adressage et de routage.

[13] Cette figure provient du document d'aide de MicrosoftWindows NT Server Resource Kit [15].

- **Couche Liaison de données (« Data link layer »)** : elle définit la façon dont les données sont échangées entre les ordinateurs. On utilise le terme « trame » pour référencer l'unité de donnée élémentaire transférée sur la couche Liaison de données. La couche liaison de données doit définir la structure des trames et indiquer comment les deux machines les délimitent.

- **Couche Physique (« Physical layer »)** : la couche Physique est la plus basse couche du modèle OSI. Cette couche définit une norme pour la connexion physique entre les machines. Cette couche décrit les interfaces électriques ou optiques, mécaniques et fonctionnelles au support physique de réseau. La couche physique porte les signaux pour toutes les couches plus élevées. À ce niveau, l'information est définie en bits.

La sous-section suivante présente en détail les différents éléments qui composent l'architecture réseau de l'environnement Windows NT. Ces connaissances sont essentielles pour situer le rôle de chacune des composantes. À la suite de cette sous-section, une attention particulière est donnée pour expliquer le rôle de la composante NDIS (« Network Driver Interface Specification ») de l'architecture Windows NT.

- **Éléments de l'architecture réseau sous Windows NT**

Toutes les couches du modèle de référence OSI sont représentées sous Windows NT par différentes composantes. La Figure 8 montre chacune de ces composantes. Chaque composante a ses rôles et fonctions.

- **Couche Application**

  - **« Remote Procedure Call » (RPC)** : le RPC est basé sur les concepts utilisés pour créer des programmes structurés et qui peuvent être visualisés comme étant un " circuit principal " avec une série de " nervures ". Le circuit principal est la logique traditionnelle du programme. Les nervures sont des procédures que le circuit principal invite pour effectuer le travail ou pour exécuter des fonctions. Le RPC va plus loin que le concept de base en plaçant le circuit principal et les nervures sur différents ordinateurs.

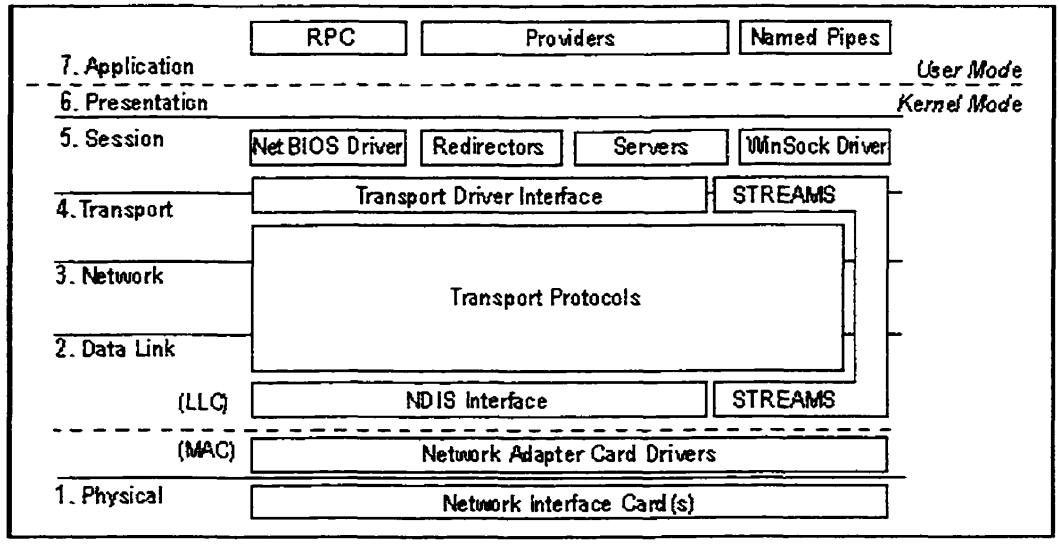


Figure 8: intégration des composantes Windows NT dans le modèle OSI[14]

- « **Provider** »: le « provider » est la composante qui permet à un ordinateur Windows NT de communiquer avec d'autres types de réseau. Par exemple, en installant le service Client NetWare, un ordinateur tournant sous Windows NT pourra se connecter à un réseau NetWare.

- « **Named Pipes** » : un « pipe » est une partie de mémoire qui peut être employée par un processus pour passer de l'information à un autre processus. Un « pipe » relie deux processus de sorte que la sortie d'un « pipe » puisse être utilisée comme entrée de l'autre.

- **Couche Session**

- « **NetBios Driver** »: NetBios est une interface de programmation qui facilite le développement d'application client/serveur. NetBios fait partie d'une des huit composantes des mécanismes IPC (« Interprocess Communication »)[15].

- « **Redirector** » : le « redirector » (RDR) est une composante qui réside au-dessus de l'interface TDI (définie plus loin) et c'est par les « redirectors » qu'un ordinateur accède à un autre ordinateur. Le « redirector » du système d'exploitation de Windows NT permet la connexion aux Windows for Workgroups, au gestionnaire de réseau local, au serveur de réseau local et autres

[14] Idem.

[15] Les autres mécanismes sont : « named pipes », « mailslots », « Windows Sockets », « Remote Procedure Calls (RPC) », « Network Dynamic Data Exchange (NetDDE) », « Server Message Blocks (SMB) », et « Distributed Component Object Model (DCOM) ».

serveurs de type MS-Net-based. Le « redirector » communique aux protocoles au moyen de l'interface TDI (« Transport Driver Interface »).

- « **Server** » : le service de serveur se place au-dessus de TDI. Il agit comme gestionnaire de système de fichiers et communique directement avec les autres gestionnaires de systèmes de fichiers pour satisfaire des demandes d'E/S telles que la lecture ou l'écriture d'un fichier.

- « **WinSock Driver** » : les API Windows Socket fournissent une interface standard aux protocoles avec différents systèmes d'adressage. Windows Socket est supporté par les protocoles TCP/IP et IPX/SPX. L'interface est composée de Wsock32.dll et Windows Sockets Emulator et ce dernier est l'intermédiaire entre les applications « Windows Socket » et le TDI. La Figure 9 montre les composantes nécessaires pour toutes communications utilisant les protocoles TCP/IP et IPX/SPX.

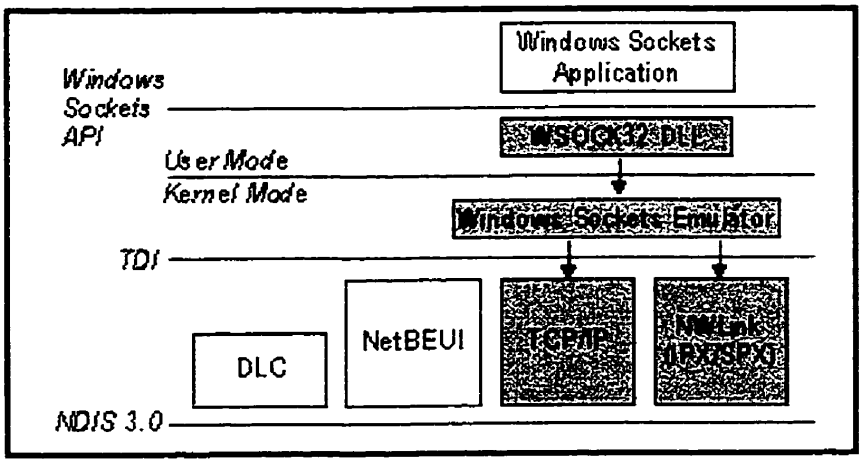


Figure 9: donnée transmise par WinSock[16]

- **Couche Transport**

- « **Transport driver interface** » **(TDI)** : TDI est une interface commune pour un pilote tel que le « redirector » ou le serveur de Windows NT. Elle permet de communiquer avec les divers protocoles de transport réseau. Ceci permet au « redirector » ou au serveur de rester indépendant de la couche transport. À la différence de NDIS , il n'y a aucun gestionnaire pour TDI; c'est simplement une norme pour passer des messages entre deux couches.

[16] Cette figure provient du document d'aide de MicrosoftWindows NT Server Resource Kit [15].

- **Streams** : les « streams » sont des canaux de données qui permettent une bande passante plus large pour le transfert de données. À cause de la surcharge d'instructions nécessaires pour passer les requêtes, les « streams » ne sont plus utilisés depuis la version 3.5 de Windows NT.
- **« Transport protocols »** : ce sont les protocoles de transfert de données de Windows NT. Il y en a quatre :
  - TCP/IP
  - NetBEUI
  - Nwlink (IPX/SPX)
  - Data link control (DLC)

- **Couche Liaison de données**
  - **« NDIS Interface » :** NDIS (« Network Driver Interface Specification ») décrit l'interface standard de communication entre la machine et les différentes cartes réseaux. L'interface permet à des composantes de haut niveau d'être indépendantes de la carte réseau. Elle contient une librairie de fonctions qui aide à la conception de gestionnaire de carte réseau.
  - **« Logical Link Control » (LLC) :** lien commun entre tous les protocoles réseau. Il détermine le mécanisme d'échange de données entre deux utilisateurs et le mécanisme d'adressage des stations sur le réseau.
  - **« Media Access Control » (MAC) :** un gestionnaire MAC contrôle le matériel d'adaptateur de réseau qui fournit la conductivité électronique par un câble ou d'autres médias à d'autres ordinateurs. Le gestionnaire MAC déplace des trames entre la pile des protocoles et le matériel d'adaptateur.
  - **« Network Adapter Card Driver » :** le pilote de la carte réseau.

- **Couche physique**
  - **« Network Adapter Card »** : la carte réseau.

- **Description du NDIS (Network Driver Interface Specification)**

La Figure 10 montre un aspect très important du modèle OSI. Chaque couche ajoute un en-tête permettant d'identifier le paquet envoyé ou reçu. Rendu au niveau de la couche Liaison de données, les trames possèdent tous les en-têtes des couches supérieu-

res. Il devient donc facile, à partir des trames de la couche Liaison de données, de recueillir toutes les informations nécessaires pour l'interception des communications.

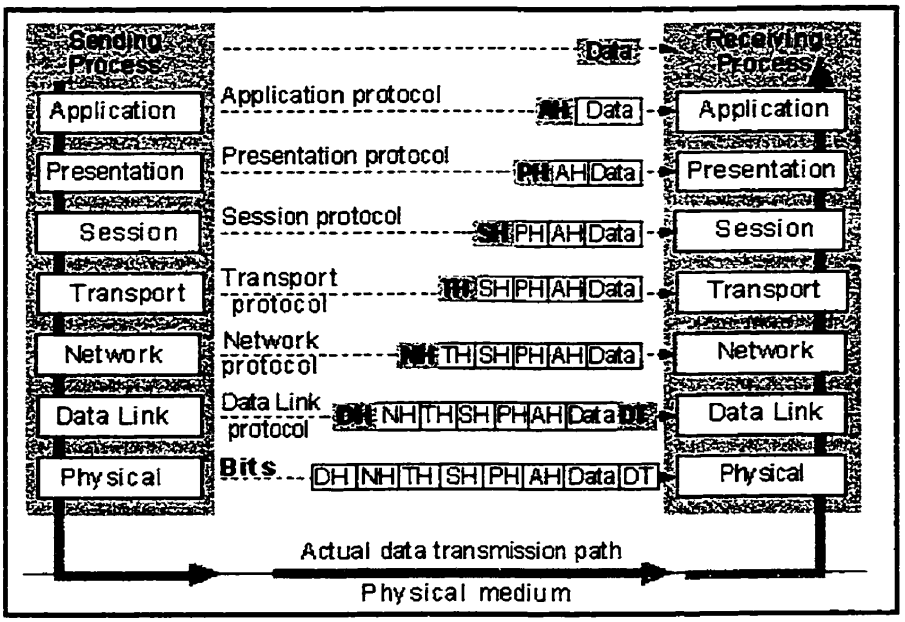


Figure 10: fonctionnement vertical du modèle OSI[17]

Puisque le but est d'intercepter toutes les communications pouvant avoir lieu sur une machine, celui-ci influencera l'emplacement du filtre. De plus, la couche Liaison de données est constituée du NDIS 3.0 (Network Driver Interface Specification). Le NDIS 3.0 est une norme qui permet aux multiples cartes réseau et protocoles de coexister. NDIS permet aux composantes de niveau supérieur d'être indépendantes de la carte réseau en fournissant une interface standard. NDIS 3.0 permet un nombre illimité de cartes réseau dans un ordinateur ainsi qu'un nombre illimité de protocoles pouvant être liés à une seule carte réseau.

Dans Windows NT, NDIS a été mis en application dans un module appelé Ndis.sys et est désigné sous le nom d'emballage NDIS (« NDIS Wrapper »). L'emballage NDIS est un petit morceau de code entourant les gestionnaires de protocole et les gestionnaires de périphérique et fournit une interface uniforme entre ceux-ci.

NDIS fournit un environnement, à l'aide d'une bibliothèque de fonctions, où un gestionnaire de protocole peut indiquer les adresses de destination ou les types d'adresse qu'il est intéressé à recevoir. Le pilote de la carte réseau passe simplement un paquet en-

[17] Idem

trant vers le haut en appelant une fonction NDIS. NDIS transmet le paquet aux couches supérieures intéressées.

L'interface NDIS capture toutes les informations sur la carte réseau. De cette façon, aucune donnée ne peut contourner le filtre. De plus, la librairie de fonctions NDIS devrait faciliter la conception d'un filtre.

À la lecture de ces informations, il est essentiel d'intercepter les communications au niveau de la couche Liaison de données parce que les trames contiennent les informations nécessaires pour déterminer si on les laisse passer ou non. Puisque NDIS.SYS est situé au niveau de la couche Liaison de données et en attachant le filtre au NDIS, ce filtre sera en mesure d'intercepter les trames entrantes et sortantes de la machine. Un grand avantage à s'attacher au NDIS.SYS, au lieu de s'attacher au pilote de la carte réseau, est que NDIS.SYS soit un standard sous Windows NT.

4.2.3 Ressource Base de registres

Pour conserver certaines informations au sujet de la configuration de son environnement, les concepteurs de Windows NT ont mis en place une structure de données qui est appelée la base de registres. Cette structure est unique à l'environnement Windows. Elle contient des informations concernant les composantes à installer au démarrage, des instructions sur les composantes à démarrer et des informations sur les usagers (mots de passe et profils).

Cette base de registres contient des informations sensibles qui doivent être à l'abri des manipulations malicieuses. Une modification erronée peut produire des résultats qui affecteront la sécurité du système. Pour ces raisons, peu de gens doivent avoir accès à ces informations sensibles. Par défaut, l'administrateur n'a pas accès à toutes les informations contenues dans la base de registres. En modifiant certaines permissions, il peut toutefois y accéder mais devra rester prudent car il pourrait endommager la fonctionnalité du système.

- **Qui interagit avec la base de registres?**

Puisque la base de registres contient des informations nécessaires pour faire fonctionner le système d'exploitation, diverses composantes et applications de Windows NT les requièrent pour obtenir ou mettre à jour des clés contenues dans la base de registres. La Figure 11 présente les interactions entre les composantes et les applications de Windows NT et sa base de registres.

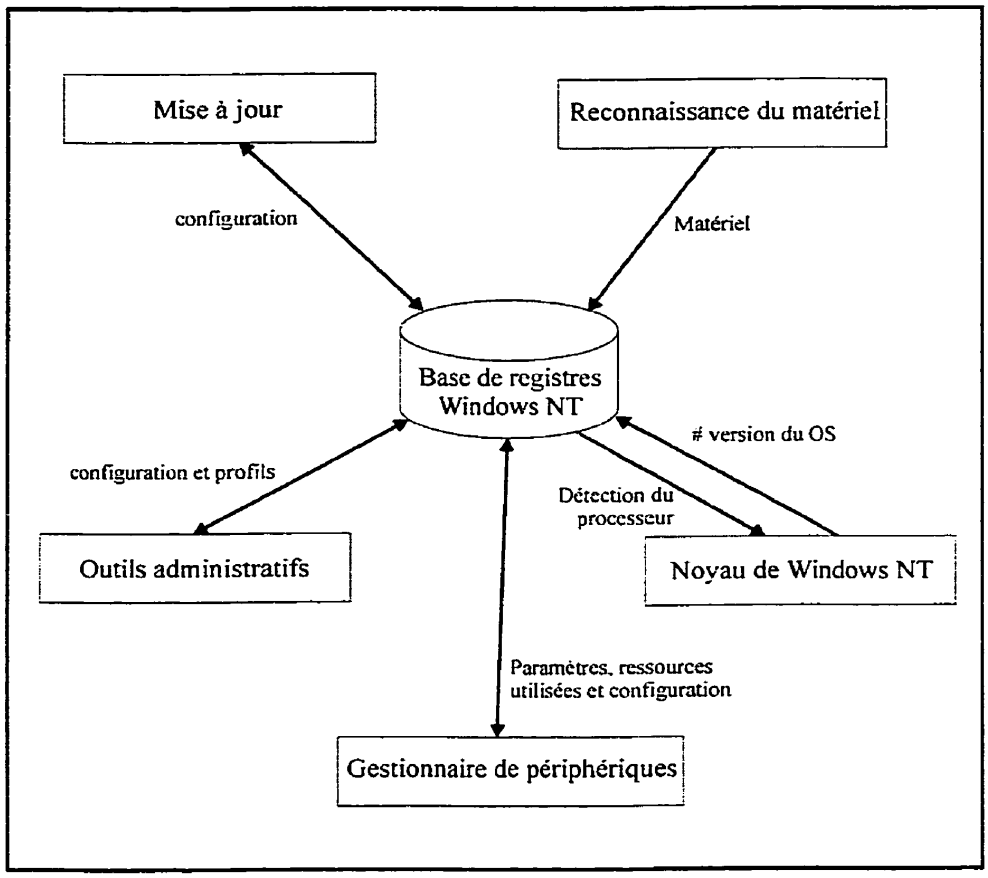


Figure 11 : intervenants avec la base de registres

- *Mise à jour.* Les applications de mise à jour inscrivent des informations se rapportant à la configuration des équipements.

- *Reconnaissance du matériel.* Au démarrage de Windows NT, le programme Ntdetect.com détecte la liste des équipements en place sur la machine et va l'inscrire dans la base de registres.

- *Noyau de Windows NT.* Durant le démarrage de Windows NT, le noyau va chercher des informations dans la base de registres, tels que les gestionnaires de périphérique et l'ordre dans lequel ils doivent être chargés. De plus, le noyau passe son numéro de version.

- *Gestionnaires de périphérique.* Les gestionnaires de périphérique envoient et reçoivent des paramètres de chargement ainsi que des données de configuration. Le ges-

tionnaire de périphérique doit mentionner quelles ressources il utilise (interruption, DMA).

- *Outils administratifs*. Les outils inclus dans Windows NT permettent de modifier certaines informations qui seront conservées dans la base de registres.

- **Clés sensibles**

Du point de vue de la sécurité, certaines sections doivent être surveillées attentivement afin d'éviter qu'un usager détruise ou modifie des informations qui affecteraient l'efficacité du système d'exploitation. Le Tableau 6 contient le nom des clés sensibles.

Clés	Abréviation	Emplacement sur disque
HKEY_LOCAL_MACHINE\SAM	HKLM\SAM	%SYSTEM_ROOT%\SYSTEM32\CONFIG\SAM (SAM.LOG)
HKEY_LOCAL_MACHINE\SECURITY	HKLM\SECURITY	%SYSTEM_ROOT%\SYSTEM32\CONFIG\SECURITY (SECURITY.LOG)
HKEY_LOCAL_MACHINE\SOFTWARE	HKLM\SOFTWARE	%SYSTEM_ROOT%\SYSTEM32\CONFIG\SOFTWARE (SOFTWARE.LOG)
HKEY_LOCAL_MACHINE\SYSTEM	HKLM\SYSTEM	%SYSTEM_ROOT%\SYSTEM32\CONFIG\SYSTEM (SYSTEM.LOG)
HKEY_USERS\DEFAULT	HKU\DEFAULT	%SYSTEM_ROOT%\SYSTEM32\CONFIG\DEFAULT (DEFAULT.LOG)

Tableau 6: énumération des clés sensibles

Ces clés peuvent être décomposées en sous-clés contenant chacune des informations sur la configuration du système. Rutstein[25] mentionne différentes clés servant à protéger le système d'exploitation sur lesquelles une attention particulière devrait être portée. Il présente une liste non exhaustive de clés qui doivent être ajoutées ou modifiées pour rendre le contrôle plus efficace.

Toutes les demandes faites à la base de registres doivent passer par les fonctions du noyau de Windows NT. Ces fonctions se retrouvent dans une table. En interceptant ces appels de fonction, il devient possible de surveiller et contrôler les requêtes faites à la base de registres. Regmon[18] est un exemple de produit qui surveille les accès à la base de registres.

4.2.4 Ressource Processus

La gestion des processus requiert une surveillance étroite parce qu'elle inclut indirectement la gestion de la mémoire. Lorsqu'un usager démarre une application, le Gestionnaire de processus crée un contexte qui est vu par le système d'exploitation comme un espace d'adressage, un ensemble d'objets et un ensemble de « threads » qui

[18] Voir l'Annexe II pour avoir un aperçu des informations qui sont recueillies lors de la surveillance de la base de registres.

s'exécutent dans ce contexte. Il y a donc à la création d'un processus, des ressources du système qui sont utilisées comme l'espace mémoire et du temps de processeur.

Puisque l'environnement Windows NT est un environnement supportant le multi-tâches, cela signifie qu'un quantum de temps est alloué à chaque processus. À la fin du quantum de temps, le système d'exploitation émet une interruption qui permet d'interrompre le processus en cours et de passer au suivant. Ce passage se fait en changeant de contexte d'exécution ce qui demande un certain temps de processeur et une gestion de la mémoire. Plus il y a de processus en attente, plus il y a de demande d'espace mémoire et de temps de processeur.

La surveillance et le contrôle rattachés à la gestion des processus deviennent donc essentielles afin de contrer les applications malicieuses qui pourraient surcharger la gestion. La gestion des processus comprend la création et la destruction des processus et la création et la destruction des « threads ». Dans l'environnement Windows NT, la gestion de la mémoire se fait par l'intermédiaire du Gestionnaire de mémoire virtuelle qui travaille en étroite collaboration avec le Gestionnaire des processus.

Pour surveiller la gestion des processus, Windows NT utilise des fonctions qui permettent de gérer les processus et les « threads ». Chaque fois qu'une demande est faite pour démarrer une application, ces fonctions du noyau sont appelées pour réaliser la requête. Le Tableau 7 montre une partie des fonctions incluses dans la table des services et qui sont utilisées par le noyau de Windows NT. Les zones grises donnent un exemple des fonctions appelées lorsqu'il s'agit de gérer les processus. En interceptant les appels de fonction, il devient possible de surveiller et contrôler la gestion des processus.

Table des services			
0003	0008:8015A7F0	params=02	ntoskrnl!NtAddAtom
0004	0008:80199B38	params=06	ntoskrnl!NtAdjustPrivilegesToken+0422
0005	0008:80199716	params=06	ntoskrnl!NtAdjustPrivilegesToken
0006	0008:80198A6A	params=02	ntoskrnl!PsGetProcessExitTime+1844
0008	0008:8015ACB8	params=01	ntoskrnl!NtAllocateLocallyUniqueId
0009	0008:8015AF86	params=03	ntoskrnl!NtAllocateUuids
000A	0008:8017D42A	params=06	ntoskrnl!NtAllocateVirtualMemory
000B	0008:8011BA74	params=03	ntoskrnl!ZwYieldExecution+0898
000C	0008:8016904A	params=02	ntoskrnl!NtCreateFile+022C
000D	0008:8010B214	params=02	ntoskrnl!_purecall+01B8
000E	0008:80155F70	params=01	ntoskrnl!ExEnumHandleTable+0584
001E	0008:80195720	params=08	ntoskrnl!PsCreateSystemThread+0696

Tableau 7: fonctions utilisées par le noyau de Windows NT

4.3 Caractéristiques de l'application principale

Il existe différentes façons de surveiller le déroulement d'exécution d'un programme pour se prémunir contre des attaques malicieuses. Une première méthode serait de suivre pas à pas l'exécution d'une application [26] et de bloquer toute tentative qui ne respecterait pas les politiques de sécurité établies. Cette approche semble, à première vue, surcharger le temps d'exécution d'une application. Une autre approche consiste à mettre en place des gardiens qui surveillent les ressources importantes. Ils devront consulter le superviseur (moniteur) pour obtenir la permission de laisser passer la requête formulée par une application. Cette approche a l'avantage de surveiller que les ressources importantes et de ne pas suivre à la trace toutes les applications. Seules les applications voulant utiliser une ressource seront surveillées.

Un autre élément à considérer est la définition des politiques de sécurité rattachées à chacune des applications. Schneider [27] propose une approche très intéressante pour concevoir des politiques de sécurité. Il utilise la notion d'automate ce qui permet de définir les actions qui sont permises en tenant compte des actions réalisées auparavant et de l'interaction entre celles-ci. Ainsi, l'exécution d'une tâche sera possible que si elle réponde à des critères de validation précis ne contrevenant pas à la sécurité de l'environnement. On ne regarde plus seulement la conséquence d'une action individuelle mais l'ensemble des actions d'une application.

Pour superviser les filtres rattachés aux ressources mises sous surveillance, une application principale sera implantée. Cette application agira comme l'entité responsable des gardiens (filtres).

Le rôle de l'application principale ou moniteur est de superviser le travail de quatre modules de surveillance (communément appellé filtre). Chaque filtre doit répondre à des exigences précises quant au déroulement de son exécution et des actions à poser lorsqu'il détecte un accroc aux politiques de sécurité établies. Ainsi, le moniteur aura les caractéristiques suivantes:

- S'exécutera en mode **service** dans l'environnement Windows NT.
- Possèdera un interface graphique où chaque fenêtre représentera un module.
- Implantera automatiquement le pilote (drivers) au démarrage du module.
- Recevra, analysera, conservera et confirmera les demandes des filtres.
- Travaillera conjointement avec le module de gestion des politiques de sécurité.

- Affichera tout message indiquant une tentative de non-respect des politiques de sécurité.
- Aura le pouvoir d'interdire les requêtes demandées (en fonction de la définition des politiques de sécurité).

La Figure 12 présente le modèle général du moniteur. Chaque filtre agit comme intermédiaire entre les ressources à surveiller et le moniteur.

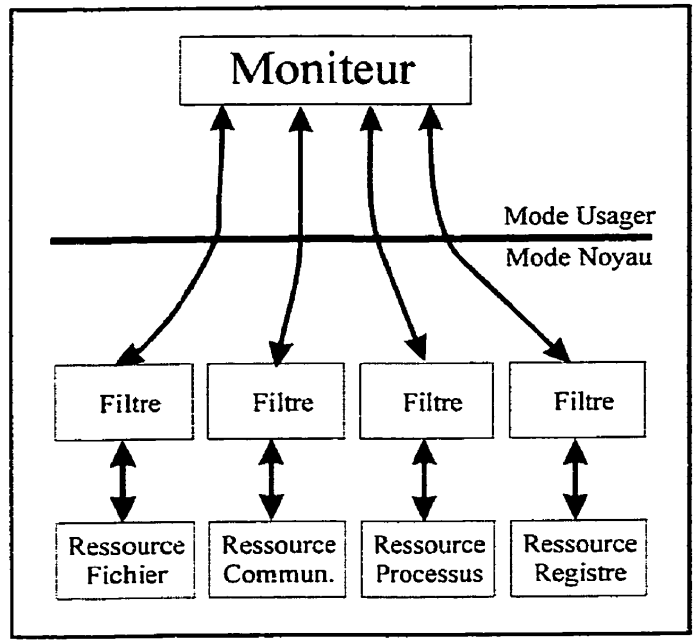


Figure 12 : modèle général du moniteur

En résumé, ce chapitre met en relief tous les aspects nécessaires pour définir un moniteur d'analyse dynamique de programmes. Il est nécessaire de connaître l'environnement dans lequel le moniteur s'exécutera, de décrire les ressources critiques sur lesquelles portera le monitorage et de définir les caractéristiques que devra supporter le moniteur.

Chapitre 5

LES FILTRES

Un des avantages de l'utilisation de l'environnement Windows NT est sa modularité. Ce système d'exploitation a été conçu de façon à permettre l'ajout ou la suppression d'éléments à son noyau. Les pilotes, qui sont essentiels au fonctionnement du noyau, peuvent être ajoutés ou retirés de la liste des éléments à installer. Dans certains cas, il n'est même pas nécessaire de redémarrer l'ordinateur pour que les changements soient pris en compte. L'ajout ou la suppression peut se faire sur demande et le résultat est immédiat. Le présent chapitre présente un des éléments clé du système d'exploitation Windows NT. Puisque les pilotes sont appelés à jouer un rôle actif dans la conception du moniteur, il est essentiel de comprendre son fonctionnement.

La première section de ce chapitre définit les différents types de pilotes qui se retrouvent dans l'environnement Windows NT. La deuxième section montre où les filtres (pilotes) doivent se positionner pour obtenir les résultats désirés. La troisième section couvre la structure interne d'un pilote qui est identique pour tous les types de pilote. La quatrième section fait un survol de la procédure d'installation d'un pilote. La cinquième section explique les différents moyens existants pour permettre une communication entre une application et un pilote. La dernière section présente les rôles des différents filtres créés pour le moniteur.

5.1 Différents types de pilotes

Il existe différents types de pilote. Les pilotes de bas niveaux correspondent aux pilotes rattachés directement à un périphérique tel un disque ou une carte de son et sont responsables de la gestion du matériel. Les pilotes de hauts niveaux correspondent aux pilotes qui ont une fonction plus générale tel un pilote FSD (« File System Driver »). Ce dernier reçoit les demandes d'accès à des fichiers et selon le type de gestionnaire de fichiers en place, des partitions et de l'équipement, il répartit la tâche. Dans tous les cas, il est possible de concevoir de nouveaux pilotes qui remplaceront ces derniers à condition de connaître leurs fonctionnalités intrinsèques.

L'ajout d'un pilote sous Windows NT permet d'intervenir à différents niveaux. Il existe principalement trois types de pilote : les pilotes de périphérique, les pilotes intermédiaires (filtre) et les pilotes de système de fichiers. Dans ce cas-ci, ce sont les pilotes intermédiaires qui sont visés parce que le but de ces pilotes est de filtrer les requêtes. Il ne

s'agit pas d'intervenir directement avec un périphérique comme une carte de son ou un disque physique.

Entre les types de pilotes de haut niveau (« FSD ») et de bas niveau (pilotes de périphérique), il est possible d'en ajouter d'autres. Windows NT permet donc d'avoir des couches de pilotes. Ces pilotes sont considérés comme des pilotes intermédiaires. À ce niveau, il existe différents types de pilotes intermédiaires. Il y a le *pilote générique* qui utilise les appels standards du Gestionnaire E/S pour envoyer des requêtes aux autres pilotes. Il existe un deuxième type, le *filtre*, qui a la particularité d'être transparent et d'intercepter les requêtes faites aux autres pilotes. Il utilise les appels standards du Gestionnaire E/S. Il existe un troisième type de pilotes intermédiaires, les *pilotes joints*, formant une paire avec un autre pilote et qui interagissent intimement entre eux à l'aide d'une interface privée.

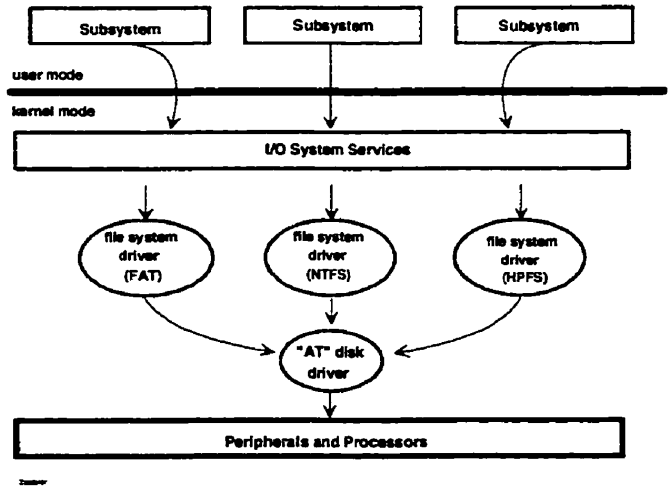


Figure 13: système de pilotes de disque AT[19]

Le choix du type de pilote est important puisque la conception de ce dernier dépendra de ce choix. Dans le cas du pilote surveillant les accès aux fichiers, celui-ci est considéré comme un filtre parce que son rôle est d'intercepter les requêtes adressées directement au pilote en dessous de lui. La conception de ce type de pilote est différente des autres parce qu'il n'a pas à ce préoccuper de certaines fonctions telles l'initialisation du matériel. Il suffit de définir les différentes fonctions nécessaires et d'attacher le pilote au pilote directement en dessous de lui. De cette façon, les requêtes seront envoyées au filtre avant d'être transmises au pilote de plus bas niveau.

[19] Cette figure provient du document d'aide de MicrosoftWindows NT Server Resource Kit [15].

Les figures suivantes montrent des exemples d'architecture de couches de pilotes. Ces exemples se rapportent exclusivement aux systèmes de fichiers. La Figure 13 représente la structure des pilotes en couche pour le système de pilotes de disque AT. En fonction de la définition des partitions faite par l'usager, un ou plusieurs systèmes de fichiers peuvent être installés au-dessus du pilote de disque AT. Dans ce cas-ci, un pilote intermédiaire pourrait s'insérer entre un pilote de systèmes de fichiers et le pilote de disque.

Il existe une autre configuration matériel utilisant le bus SCSI. Ce bus est connecté à un contrôleur DMA (« Direct Memory Access ») ce qui permet le branchement de plusieurs périphériques. La Figure 14 présente un pilote pour le port SCSI implémenté sous forme d'un fichier DLL. Ce dernier exporte des routines pouvant être utilisées par des pilotes de miniports. Au plus haut niveau, on retrouve différents pilotes de systèmes de fichiers pouvant être implémentés sous Windows NT. Les ellipses en pointillées représentent les pilotes intermédiaires pouvant être insérés dans la structure en couche des pilotes.

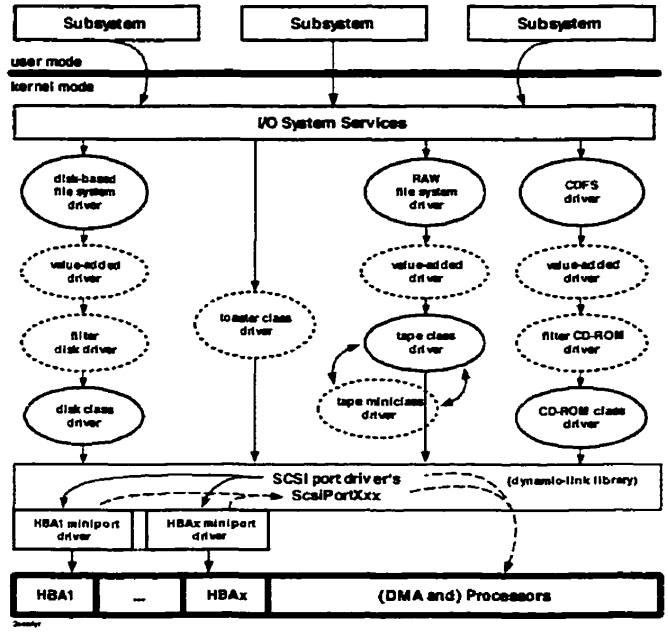


Figure 14: système de pilotes de disques SCSI[20]

5.2 Positionnement du filtre

À partir de ces connaissances, on constate qu'il est possible d'intercepter des informations et de les traiter. Plusieurs solutions existent pour intercepter les informations.

[20] Idem

Une première façon serait de capturer tous les appels au niveau de NTDLL.DLL[21] en mode usager. Le problème majeur est qu'il n'est pas garanti de pouvoir intercepter tous les appels. Une deuxième façon, pour éviter les contournements, serait d'aller se positionner le plus bas possible, près de la ressource à surveiller, afin d'intercepter tous les appels E/S. Les filtres ou pilotes intermédiaires permettent ce positionnement.

En tenant compte de la possibilité qu'offre Windows NT d'introduire des pilotes intermédiaires pour filtrer les requêtes, la solution retenue pour gérer les requêtes est présentée à la Figure 15. On introduit un filtre entre deux pilotes. On remarque, de plus, l'ajout des politiques de sécurité gérées par le moniteur (ces politiques seront définies plus tard). La Figure 15 présente une partie de l'environnement Windows NT en se concentrant exclusivement sur le Gestionnaire E/S et de l'interaction avec les pilotes. Le filtre est inséré entre le pilote du système de fichiers et le pilote de périphérique. La raison principale sur laquelle repose cette insertion tient au fait que c'est au pilote du système de fichiers de gérer les droits d'accès aux fichiers (NTFS). Le filtre évide donc de surveiller toutes les requêtes inutiles se rapportant à la sécurité des accès.

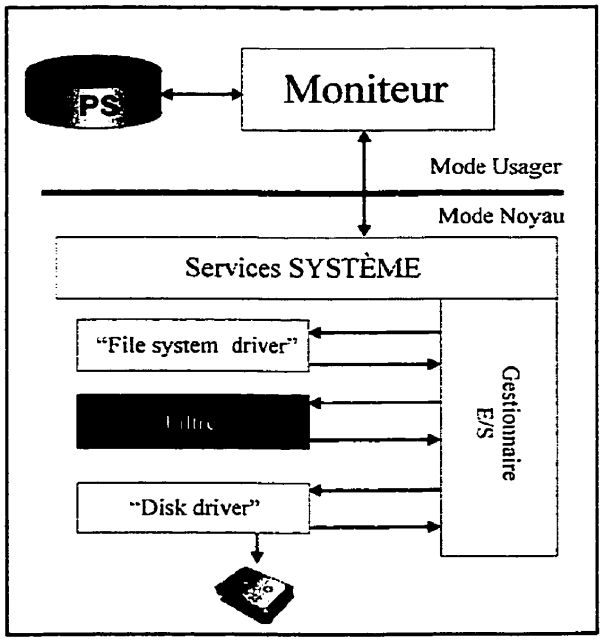


Figure 15 : positionnement du filtre

De plus, lorsqu'une requête est faite, le moniteur doit vérifier cette requête à partir de la base de données contenant les politiques de sécurité rattachées à chacune des applications pour voir s'il accorde ou non la requête.

[21] La Figure 5 de la section 4.2.1, page 28, explique le cheminement d'un appel E/S et le rôle de NTDLL.DLL

5.3 Structure d'un pilote

Contrairement à une application, un pilote ne possède aucune fonction « main » permettant au programme de s'exécuter. Un pilote correspond à une collection de fonctions qui sont appelées, au besoin, par le Gestionnaire E/S. Le Tableau 8 résume la structure d'un pilote en donnant une brève explication du rôle de chaque fonction.

Catégories	Fonctions	Description
Initialisation et nettoyage	DriverEntry	Gestionnaire E/S appelle cette fonction lorsqu'il charge le pilote. Cette fonction exécute une série d'étapes pour initialiser le pilote. Elle n'est exécutée qu'une seule fois.
	Reinitialize	Cette fonction est appelée si le pilote ne s'est pas chargé correctement durant l'exécution de la fonction DriverEntry.
	Unload	Cette fonction permet de décharger le pilote en défaisant ce que la fonction DriverEntry a fait.
	Shutdown	Le Gestionnaire E/S appelle cette fonction pour mettre le matériel dans un état connu.
	Bugcheck callback	Cette fonction est utilisée en cas de panne système afin de mettre le matériel dans un état connu.
« Dispatch »	Open et Close	Tous les pilotes doivent fournir une fonction qui supporte une requête Win32 « CreateFile ».
	Device operations	Peut contenir une ou plusieurs fonctions pour répondre aux appels Win32 « ReadFile() », « WriteFile() » et « DeviceIoControl() ».
Transfert de données	StartI/O	Cette fonction est appelée par le Gestionnaire E/S lorsqu'il arrive le temps de transférer des données.
	Interrupt Service (ISR)	Cette fonction est responsable de la reconnaissance du périphérique et de l'acquisition d'informations volatiles.
	DPC	Un pilote peut avoir plusieurs fonctions DPC qui s'exécutent après les « Devices operations » afin de compléter les requêtes.
Synchronisation	ControllerControl	Cette fonction contrôle les accès au périphérique.
	AdapterControl	Cette fonction contrôle les accès aux opérations DMA (Direct Memory Access).
	SynchCritSection	Cette fonction permet de faire exécuter une partie du code à un niveau d'interruption plus élevé.
Autres	Timer	Cette fonction est utilisée pour les pilotes qui désirent conserver le temps de passage d'une opération.

Catégories	Fonctions	Description
	I/O completion	Cette fonction est utilisée pour les pilotes de plus haut niveau afin de recevoir un signal que la requête est complétée au niveau inférieur.
	Cancel I/O	Cette fonction est utilisée pour le nettoyage quand une requête prend trop de temps à s'exécuter.

Tableau 8: structure d'un pilote

5.4 Installation des pilotes

Avant de pouvoir installer un pilote dans l'environnement Windows NT, il est essentiel de définir tout périphérique et pilote comme un objet. Une application ne peut accéder directement à une ressource (pilote) parce que l'accès à l'objet est réservé au Gestionnaire E/S. Pour y accéder, il est nécessaire de définir un nom ou lien symbolique (« symbolic link »). De cette façon, toute application peut envoyer et recevoir des informations d'une ressource à la condition de connaître les codes de contrôle nécessaires à la communication.

L'utilisateur ou l'application qui veut envoyer un message au pilote doit passer par le lien symbolique. Personne n'accède à l'objet directement. Par exemple, sous Windows NT, il existe un objet périphérique « \device\parallel0 » qui correspond au port parallèle 1. Pour y accéder, on doit envoyer un message aux liens symboliques LPT1 ou PRN correspondant au périphérique « port parallèle 1 ».

Le code suivant montre la fonction implémentée et nécessaire pour installer le filtre. Cette fonction utilise `CreateService()`[22] qui permet d'installer le pilote. Sous Windows NT, un pilote est considéré comme un service. Il est aussi possible d'arrêter ou de démarrer un pilote de périphérique à l'aide du Gestionnaire de services inclus dans le Panneau de configuration.

[22] Code extrait du programme Instdrv.c contenu dans les exemples de Microsoft DDK [13] et utilisé dans Filemon.c

```
/*****************************************************************************
*
*    FUNCTION: InstallDriver( IN SC_HANDLE, IN LPCTSTR, IN LPCTSTR)
*
*    PURPOSE: Creates a driver service.
*
*****************************************************************************/
BOOL InstallDriver( IN SC_HANDLE SchSCManager, IN LPCTSTR DriverName, IN LPCTSTR Service-
Exe )
{
    SC_HANDLE  schService;

    //
    // NOTE: This creates an entry for a standalone driver. If this
    //       is modified for use with a driver that requires a Tag,
    //       Group, and/or Dependencies, it may be necessary to
    //       query the registry for existing driver information
    //       (in order to determine a unique Tag, etc.).
    //

    schService = CreateService( SchSCManager,          // SCManager database
                                DriverName,            // name of service
                                DriverName,            // name to display
                                SERVICE_ALL_ACCESS,    // desired access
                                SERVICE_KERNEL_DRIVER, // service type
                                SERVICE_DEMAND_START,  // start type
                                SERVICE_ERROR_NORMAL,  // error control type
                                ServiceExe,            // service's binary
                                NULL,                 // no load ordering group
                                NULL,                  // no tag identifier
                                NULL,                  // no dependencies
                                NULL,                  // LocalSystem account
                                NULL                   // no password
                                );
    if ( schService == NULL )
        return FALSE;

    CloseServiceHandle( schService );

    return TRUE;
}
```

Extrait 1: fonction InstallDriver()

Si l'opération ne réussit pas, le service est immédiatement fermé. De cette façon, il est impossible de communiquer avec le pilote puisque celui-ci n'est pas installé. Si l'installation est réussie, les informations se rapportant au pilote nouvellement installé sont stockées dans la base de registres.

5.5 Communication avec l'application principale

Le transfert d'informations entre le Gestionnaire E/S et les pilotes se fait toujours de la même façon. Windows NT utilise les IRP[23](« I/O Request Packet ») pour envoyer les commandes et informations nécessaires pour que les pilotes puissent les exécuter. Un

[23] L'Annexe V contient la structure complète d'un IRP.

IRP est un jeton qui contient plusieurs informations. Ces informations sont utilisées durant le voyage du jeton. On retrouve des informations telles le moyen de transférer des données entre l'application principale et le pilote, l'opération à exécuter (exemple : lire ou écrire) et des informations de retour pour connaître l'état (« status ») de la requête. Tous les accès aux disques sont transformés en requêtes gérées par le Gestionnaire E/S. Ce dernier se charge de construire les IRP nécessaires et d'inclure les informations pertinentes à la requête en cours. C'est aussi lui qui reçoit le IRP à son retour et qui le détruit. Lorsque le traitement est terminé, la structure « I/O Status Block » incluse dans l'entête du paquet IRP, contient des informations se rapportant à la réussite de la requête. Le Gestionnaire E/S transmet alors le résultat de la requête à l'application. De cette façon, l'application principale sait si la requête a réussi ou échoué. Cette information est très utile parce qu'elle permet de connaître la nature du problème.

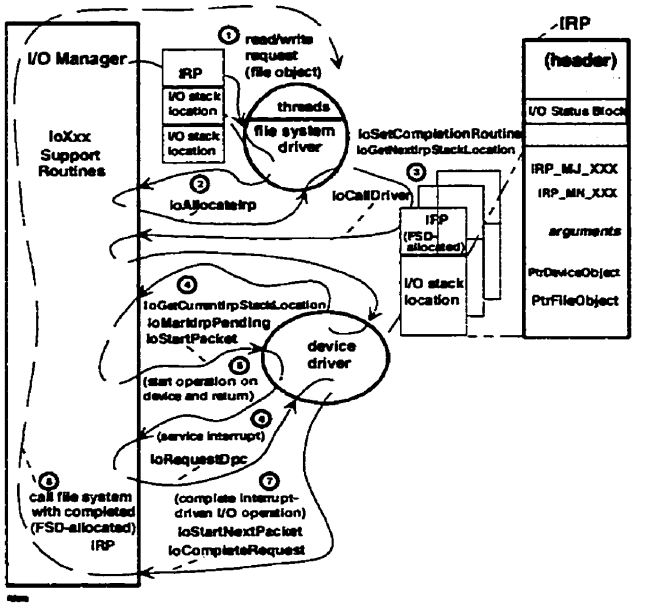


Figure 16 : exemple du cheminement d'un paquet IRP[24]

La Figure 16 représente le chemin que doit parcourir un IRP ainsi que les fonctions utilisées durant son voyage.

[1] Le Gestionnaire appelle le pilote du système de fichiers (**FSD**) avec le IRP alloué pour la requête de lecture/écriture. Le FSD accède à son espace personnel dans la pile du IRP (« I/O Stack location ») pour déterminer l'opération qu'il doit exécuter.

[24] Cette figure provient du document d'aide de Microsoft DDK [13].

[2] Le FSD peut définir la requête principale en plus petite requête en allouant un ou plusieurs IRP. Cette possibilité est réalisée à l'aide de l'appel à la fonction `IoAllocateIrp()`.

[3] Pour chaque IRP alloué, le FSD appelle une fonction E/S pour déterminer si le pilote de plus bas niveau a terminé l'exécution de son travail et libéré chaque IRP alloué à ce pilote. Le Gestionnaire E/S appelle la fonction d'achèvement (« completion routine ») du FSD pour signifier que le travail est complété et retourner un code de statut. Les pilotes de haut niveau sont responsables de la gestion et de la destruction des IRP qu'ils ont alloués. Une fois le travail complété, le Gestionnaire E/S libère les IRP qu'il avait alloués.

[4] Au moment de l'appel avec le IRP, le périphérique accède à son espace réservé dans la pile du IRP (« I/O Stack Location ») pour déterminer l'opération à exécuter (IRP_MJ_XXX[25], où XXX peut avoir plusieurs valeurs).

[5] Le Gestionnaire E/S détermine si le pilote du périphérique est occupé à traiter un autre IRP; si oui, il met le IRP dans la queue d'attente et retourne. Sinon, le Gestionnaire E/S dirige le IRP vers la fonction correspondante à l'opération E/S désirée.

[6] Quand le périphérique lance une interruption (niveau de priorité très élevé), la fonction d'interruption de service (« ISR – Interrupt Service Routine ») fait le strict nécessaire pour traiter l'interruption et faire une sauvegarde du contexte de l'opération. La fonction place le IRP dans la file associée à une fonction différée de traitement (« DPC – Deferred Procedure Call ») qui continuera le travail plus tard mais à un niveau moins élevé de priorité.

[7] Quand la fonction différée reprend le contrôle, elle utilise le contexte sauvegardé (appel à `IoRequestDpc()`) pour compléter l'opération. La fonction différée (DPC) appelle une fonction pour retirer de la file le prochain IRP à traiter. Le DPC règle le statut de l'opération complétée dans l'espace réservée à cet effet dans le IRP et le retourne au Gestionnaire E/S en appelant la fonction `IoCompleteRequest()`.

[8] Le Gestionnaire E/S appelle la fonction d'achèvement du FSD et lui passe le IRP correspondant. Cette fonction vérifie le « I/O Status Block » du IRP pour déterminer si elle doit recommencer la requête ou mettre à jour certains états internes maintenus à cause de la requête initiale et libère les IRP qu'elle avait alloué. Quand il a terminé son traitement, le Gestionnaire E/S retourne le résultat du traitement de la requête à l'application.

[25] Voir le Tableau 5, page 31, pour la liste des opérations de base.

L'autre aspect important dans la communication entre l'application principale et les pilotes est la possibilité d'envoyer et de recevoir de l'information du pilote. Il existe deux méthodes présentées dans les sections suivantes.

5.5.1 Transfert par tampon (« Buffered I/O »)

Cette technique de transfert des données est utilisée pour les périphériques lents et qui transfèrent une petite quantité d'information à la fois. Dans cette catégorie, on retrouve le clavier, la souris et les ports séries et parallèles. L'idée principale de cette technique repose sur le fait que le Gestionnaire E/S réserve un espace mémoire non paginée en mode noyau pour y déposer des données allant ou provenant d'une application qui tourne en mode usager. Dès que le « thread » reprend le contrôle du processeur et que la requête est complétée par un appel `IoCompleteRequest()`, le Gestionnaire E/S transfère les informations dans le tampon de l'application qui est situé dans la mémoire en mode usager.

La Figure 17 représente le transfert par tampon entre une application tournant en mode usager et le tampon en mode noyau.

[1] Le Gestionnaire E/S répond à la requête de lecture du « thread » courant. Ce dernier transmet un espace d'adressage représentant son tampon en mode usager.

[2] Le Gestionnaire E/S vérifie l'accessibilité du tampon de l'usager et appelle `ExAllocatePool()` pour créer un tampon de même grosseur en mode noyau.

[3] Pour une requête de lecture, le pilote lit les données provenant du périphérique dans le tampon système.

[4] Quand le pilote a reçu un `IoCompleteRequest()`, et que le « thread » original est actif, le Gestionnaire E/S copie les données lues du tampon système vers le tampon usager. De plus, il libère l'espace réservé en mode noyau en faisant un appel `ExFreePool()` et dispose du IRP.

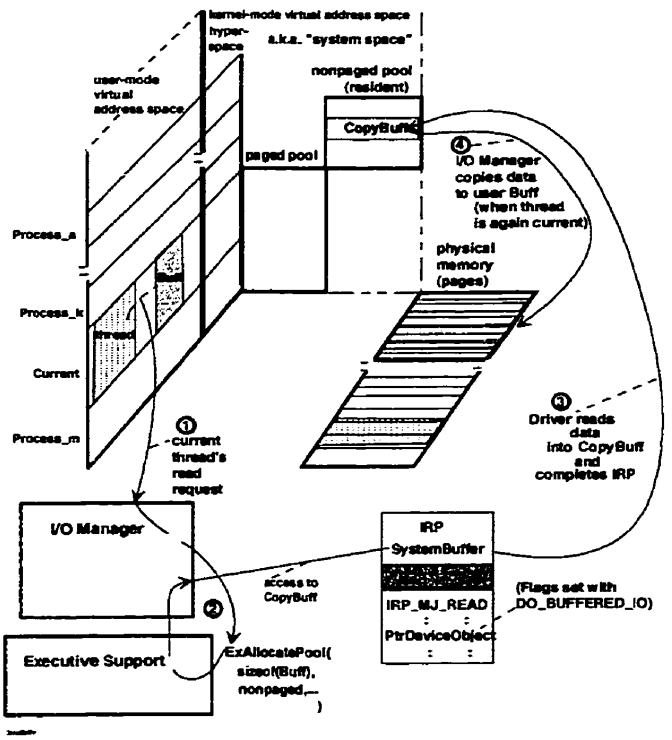


Figure 17 : transfert par tampon (Buffered I/O)[26]

5.5.2 Transfert direct (« Direct E/S »)

Un périphérique qui doit transférer une quantité importante d'information devrait spécifier un type de transfert direct. Cette technique augmente la performance du pilote en réduisant la surcharge de travail imposée par les interruptions (changement de contexte) et en éliminant les opérations d'allocation et de copiage de la mémoire relatives à la technique de transfert par tampon.

La Figure 18 représente les opérations exécutées lorsque la technique du transfert direct est utilisée.

[1] Le Gestionnaire E/S répond à la requête de lecture du « thread » courant. Ce dernier transmet un espace d'adressage représentant son tampon en mode usager.

[2] Le Gestionnaire E/S ou le FSD (« File System Driver ») vérifie l'accessibilité du tampon usager et appelle `MmProbeAndLockPages()` avec un

[26] Cette figure provient du document d'aide de Microsoft DDK [13].

MDL[27](« Memory Descriptor List ») qui spécifie un espace d'adressage virtuel pour le tampon usager et référence les pages physiques de la mémoire.

[3] Comme le démontre la Figure 18, le Gestionnaire E/S fournit un pointeur vers cette structure MDL (`MdlAddress`) dans le IRP correspondant à la requête. Tant que le Gestionnaire E/S ou le FSD n'a pas fait l'appel `MmUnlockPages()`, après que le pilote a complété le IRP, les pages de mémoire physique restent bloquées et assignées au tampon.

[4] Si le pilote utilise le DMA (« Direct Memory Access »), il appelle `MmGetMdlVirtualAddress()` pour obtenir le champ `MdlAddress` du IRP pointant vers la structure MDL.

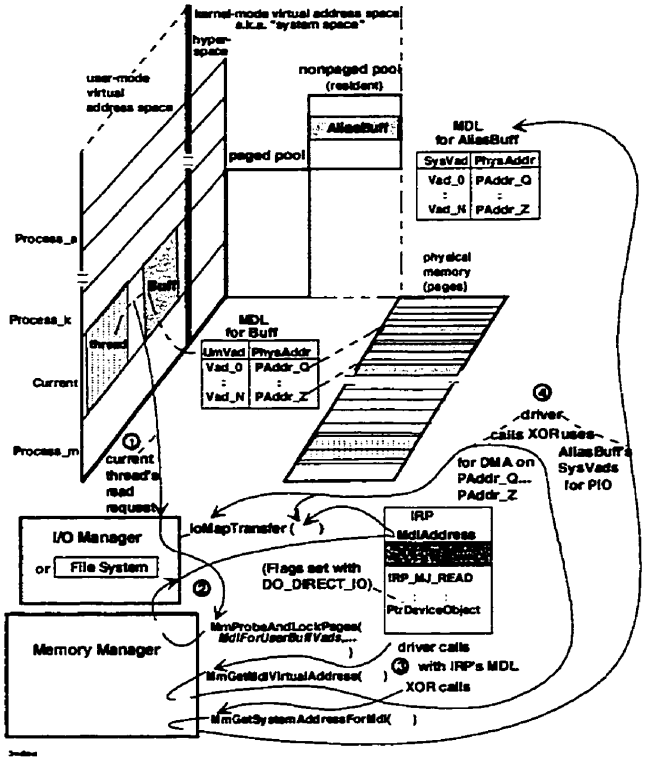


Figure 18 : transfert direct (Direct I/O)[28]

[27] « Memory Descriptor List » : structure de données opaque définie par le Gestionnaire de Mémoire qui contient des références de l'espace d'adressage virtuel vers des pages de mémoire physique.

[28] Cette figure provient du document d'aide de Microsoft DDK [13].

5.6 Description des filtres existants

Cette section fait un survol des différents pilotes conçus pour surveiller les ressources critiques. Chaque pilote est en mesure de surveiller les requêtes faites à ces ressources mais aucun n'est complété pour assurer un contrôle total. L'ajout des politiques de sécurité ainsi que la définition des règles de sécurité précises contribuent à l'amélioration des filtres.

5.6.1 FileGuard – Surveillance des fichiers

FileGuard est le pilote qui surveille les accès aux fichiers et à certains IPC (« Interprocess Communication ») tels les « named pipes » et « mailslots ». Ce pilote est l'extension d'un pilote existant, *Filemon.sys*[29]. FileGuard possède les caractéristiques de Filemon. De plus, FileGuard surveille tous les types de fichiers et non pas seulement les applications. Il reçoit ses instructions du moniteur et peut réagir s'il détecte une tentative de violation des règles de sécurité.

5.6.2 PortGuard – Surveillance des ports de communication

PortGuard est le nom du pilote surveillant les communications sous Windows NT. Le travail accompli jusqu'à maintenant permet de surveiller toutes les communications supportées par le protocole TCP/IP. Ce pilote fournit des informations relatives aux adresses de l'émetteur et du récepteur, les ports de communication utilisés, le type de protocole en cause et les données transmises. PortGuard devra surveiller les communications provenant et sortant de la machine en fonction des politiques de sécurité. Toute action violant les politiques de sécurité sera interrompue par le pilote et signalée au moniteur.

5.6.3 RegGuard – Surveillance de la base de registres

RegGuard est le filtre qui surveille les requêtes faites à la base de registres. Ce filtre est l'extension du pilote *Regsys.sys*[30]. RegGuard possède toutes les caractéristiques de Regsys mais a été enrichi de fonctionnalités lui permettant d'intercepter les requêtes tentant de violer les politiques de sécurité.

5.6.4 ProcGuard – Surveillance des processus

ProcGuard est le filtre qui surveille la liste des applications s'exécutant dans l'environnement Windows NT. Ce filtre ne fait que retourner des informations au moniteur. Ce filtre nécessite une attention particulière parce que le noyau fait appel à des fonc-

[29] Code source provenant du site : **www.sysinternals.com** .
[30] Idem.

tionnalités non documentées de l'environnement Windows NT. Il reste beaucoup de travail à faire pour le rendre plus fonctionnel.

À la lecture des informations incluses dans ce chapitre, on constate que les pilotes jouent un rôle important dans l'environnement Windows NT. En voulant rendre modulaire ce système d'exploitation, les concepteurs permettent aux programmeurs d'insérer des éléments ayant un lien direct avec le noyau. En exploitant cet avantage, quatre filtres ont été conçus qui permettent de surveiller les ressources sensibles de l'environnement Windows NT.

La faiblesse de ce type de conception réside dans le fait qu'il serait possible d'insérer un pilote malicieux. Par contre, pour insérer un pilote, il faut obtenir les privilèges d'administrateur.

Chapitre 6

LE MÉCANISME DE SURVEILLANCE ET DE CONTRÔLE

La surveillance et le contrôle des ressources se font à deux niveaux. En mode Usager, le moniteur joue un rôle actif puisque c'est lui qui dicte les directives aux pilotes. En mode Noyau, les pilotes signalent au moniteur, au moyen d'événements, qu'il y a violation de règles de sécurité. Le mécanisme mis en place permet d'échanger des informations en tout temps entre les deux niveaux. Cet échange d'information permet au moniteur de s'adapter aux changements. À l'aide d'un automate et des informations reçues, le moniteur sait exactement dans quel état se trouvent les ressources mises sous surveillance ce qui lui permet de réagir aux nouvelles requêtes. Cet aspect dynamique du monitorage joue un rôle crucial dans l'exécution des requêtes faites aux ressources critiques.

Ce chapitre présente le mécanisme de surveillance et de contrôle élaboré pour permettre au moniteur et aux pilotes de réagir lorsqu'il y a détection de violation de règles de sécurité. Le chapitre est divisé principalement en trois parties. La première partie illustre le modèle conceptuel sur lequel s'appuie le mécanisme. La deuxième partie couvre toute la mécanique nécessaire pour que le moniteur puisse travailler et réagir correctement. La troisième partie présente les fonctionnalités implantées en mode Noyau ou, plus précisément, dans les pilotes de surveillance.

6.1 Modèle conceptuel

Pour illustrer le mécanisme de surveillance et de contrôle[31], la Figure 19 présente le modèle sur lequel s'appuie ce mécanisme. Lorsqu'une application fait une demande E/S, cette dernière doit passer par la librairie NTDLL.DLL. Cette librairie transforme la requête pour que le Gestionnaire E/S la comprenne. À partir ce moment, c'est le Gestionnaire E/S qui prend les commandes de la requête.

[31] L'illustration du mécanisme est faite à partir du filtre FileGuard. Le mécanisme devrait être sensiblement le même pour tous les autres filtres à mettre en place.

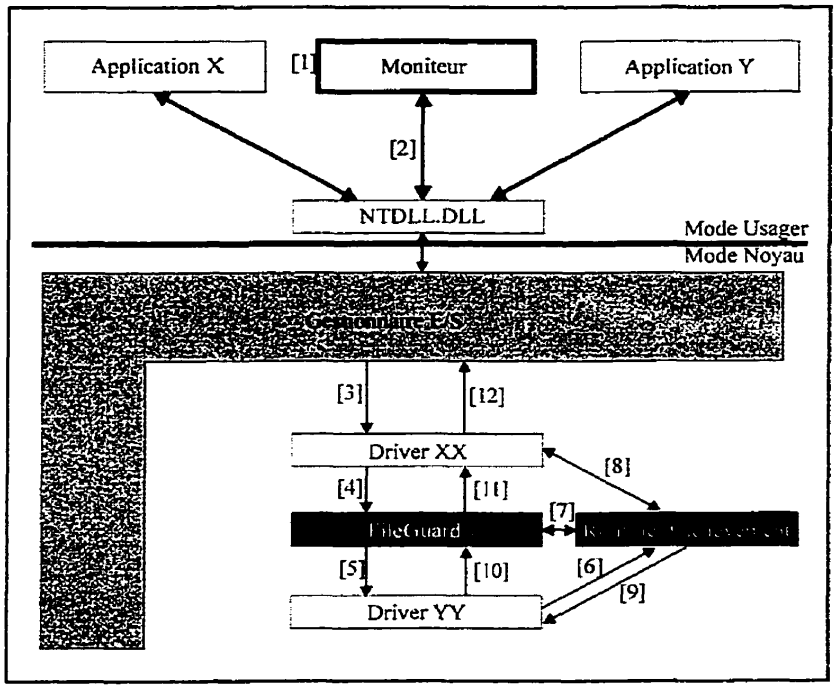


Figure 19 : modèle conceptuel

[1] Au démarrage du moniteur, il y a la création d'un événement nommé DAMON_ALERT et la création d'un « thread » qui aura la responsabilité d'envoyer un message au moniteur lui demandant d'afficher une boîte de dialogue. Ce « thread » est initialement mis en état d'attente par l'utilisation de la fonction `WaitForSingleObject()` et sera réactivé par le signalement de l'événement DAMON_ALERT.

[2] Une application émet une requête E/S au Gestionnaire E/S pour accéder à un fichier.

[3] Le Gestionnaire E/S construit un IRP qu'il transmet au premier pilote de la chaîne.

[4] Le Driver XX continue le traitement en envoyant le IRP au pilote suivant, FileGuard.

[5] FileGuard continue le traitement en envoyant le IRP au Driver YY.

[6] Lorsque le IRP est complété par le dernier pilote de la chaîne d'exécution, ce dernier utilise la fonction `IoCompleteRequest()` pour le signifier au Gestionnaire E/S et initie le stage d'achèvement.

[7] Durant le stage d'achèvement, la routine d'achèvement de FileGuard est appelée. Voir Figure 20 pour des explications détaillées sur le rôle de la routine d'achèvement.

[8] Durant le stage d'achèvement, la routine d'achèvement du Driver XX est appelée.

[9] Lorsque toutes les routines d'achèvement ont été traitées, le contrôle est retourné au Driver YY.

[10] Le Driver YY retourne une valeur représentant le statut de l'opération. Ex. : SUCCESS, ACCESS_DENIED, CANCELLED, ...

[11] FileGuard retourne le statut au Driver XX.

[12] Le Driver XX retourne le statut au Gestionnaire E/S et ce dernier le retournera à l'application.

Avec un traitement approprié, l'application qui a initié la requête réagira à la valeur du statut retournée par le Gestionnaire E/S.

Tel que mentionné précédemment à l'étape [7], un stage d'achèvement est initié lorsqu'un IRP a franchi tous les maillons de la chaîne d'exécution. La Figure 20 explique le fonctionnement des routines d'achèvement incluses dans le stage d'achèvement.

[1] Lorsque le Driver YY a terminé son traitement, il appelle la fonction `IoCompleteRequest(IRP)` pour le signifier au Gestionnaire E/S.

[2] Le Gestionnaire E/S appelle la fonction d'achèvement du Driver YY, si elle existe, pour un traitement final au niveau de ce pilote.

[3] Le Gestionnaire E/S appelle la fonction d'achèvement de FileGuard, `FilemonHookDone(IRP)` pour un traitement final au niveau de ce pilote.

[4] Le IRP entre dans la fonction `GetPermissions()` pour vérifier si l'application qui demande l'accès à un des fichiers mis sous surveillance a les permissions requises, c'est-à-dire qu'il ne viole pas une des règles associées au fichier. S'il n'a pas les permissions requises, l'événement `DAMON_ALERT` passe à l'état « signalé ». Du même coup, la fonction tombe en attente et attend que l'événement `DAMON_ALERT_FOR_DRIVER` soit à l'état « signalé ». Le contrôle est retourné au moniteur puisque le « thread » créé et endormi au démarrage du moniteur vient de se réactiver à cause de l'événement signalé. Lorsque le moniteur répond, le « thread » se rendort et attend de nouveau après

l'événement DAMON_ALERT. Lorsque le pilote reçoit la commande de continuer le traitement, il met l'événement DAMON_ALERT_FOR_DRIVER à l'état « signalé » ce qui permet à la fonction `GetPermissions()` de continuer son travail et de retourner le statut du IRP défini par le moniteur.

[5] Le statut associé au traitement de la routine d'achèvement est inscrit dans le champ correspondant dans le IRP et le IRP est retourné au Gestionnaire E/S.

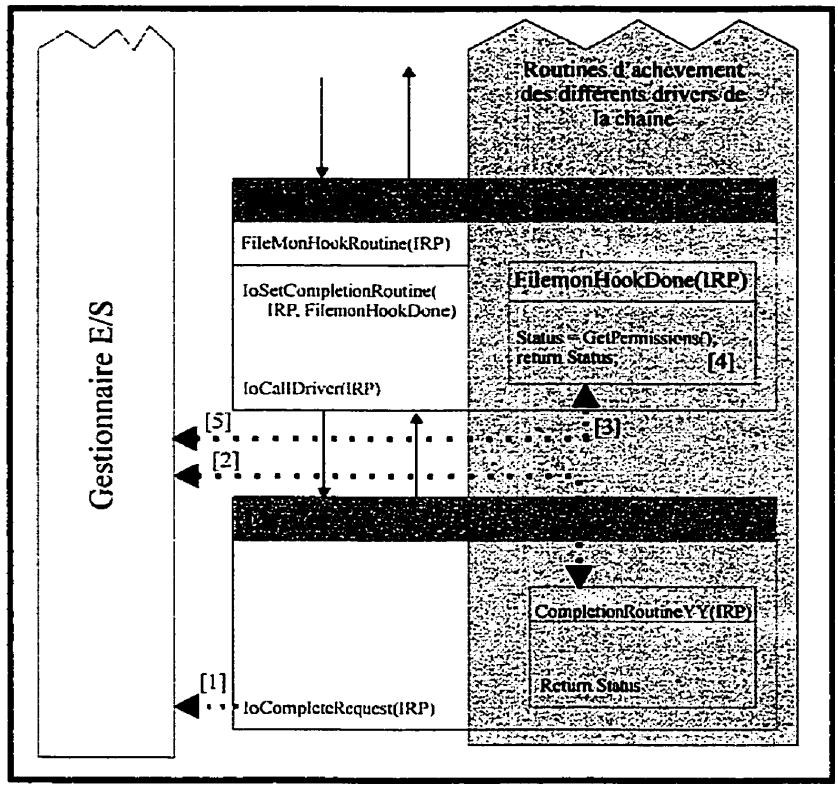


Figure 20: routines d'achèvement des différents pilotes

Par la suite, le IRP continue son chemin pour remonter jusqu'à l'application qui a initié la requête E/S. En fonction du statut du IRP, l'application réagira en traitant ce statut.

6.2 Implantation au niveau Usager – le moniteur

Pour parvenir à gérer les requêtes faites aux ressources, il est essentiel de mettre en place des éléments nécessaires à la gestion des requêtes. Cette partie couvre la définition des restrictions qui doivent être envoyées aux filtres, la définition des réactions qui seront

émises par le moniteur et retournées aux filtres pour être exécutées, la création du mécanisme dédié aux traitements des violations et la création d'un automate pour valider les requêtes faites aux ressources. Tous ces éléments, reliés à la gestion des requêtes, se situent en mode Usager.

6.2.1 Définition des restrictions

Dans le modèle proposé, une restriction signifie que la requête sera refusée si elle est activée. La Figure 21 présente l'interface se rapportant à la définition des restrictions pour chacun des fichiers faisant partis de la liste. Ces restrictions seront utilisées dans la construction du filtre des restrictions qui sera envoyé aux pilotes (filtres).

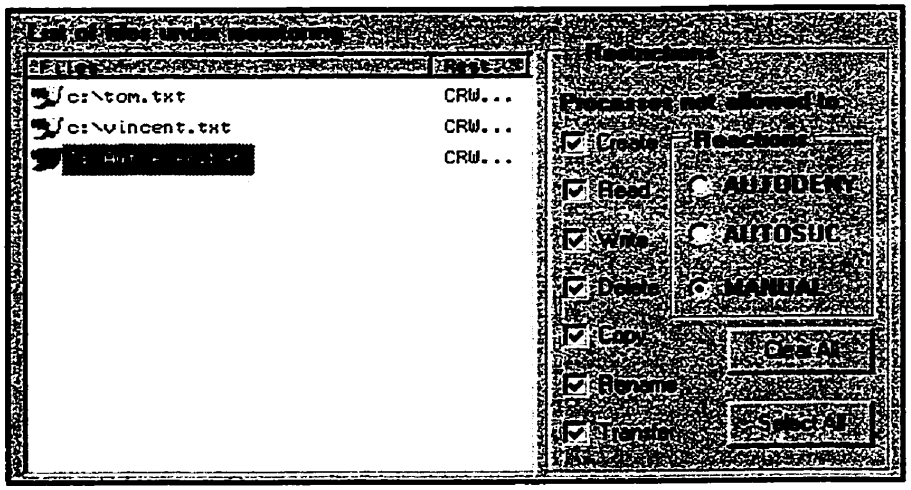


Figure 21: définition des restrictions

Il est important de noter qu'à chaque restriction est associée une valeur en hexadécimale. Cette valeur représente la position précise d'un bit dans une variable en contenant trente-deux (quatre octets). Le tableau suivant montre les valeurs qui agiront comme masque lors de la validation des actions demandées sur certains fichiers mis sous surveillance.

```
#define NULLFILE           0x00000000
#define READFILE           0x00000001
#define WRITEFILE          0x00000002
#define DELETEFILE         0x00000004
#define COPYFILE           0x00000008
#define RENAMEFILE         0x00000010
#define TRANSFERFILE       0x00000020
#define CREATEFILE         0x00000040
#define FULLRESTRICTIONS   0x0000007f  // somme des valeurs précédentes
```

Extrait 2: masques de validation

- **Construction du filtre**

Le filtre correspond aux informations qui sont passées aux pilotes. Ce filtre contient les informations qu'il peut traiter. Dans le cas de FileGuard, le filtre est représenté par une structure qui permet d'envoyer différentes données au pilote. L'Extrait 3 propose la structure de données qui sera transmise au gardien FileGuard. Le champ `processfilter` est un pointeur sur une chaîne de caractères. Cette chaîne correspond à la liste des fichiers mis sous surveillance ainsi que les restrictions assignées à chacun. La chaîne a cette forme :

« c:\\autoexec.bat*|17f;d:\\tom.txt*|43e; »

Les caractères « | » et « ; » sont nécessaires pour délimiter chaque fichier et leurs restrictions.

Les champs `excludeprocess[]`, `pathfilter[]` et `excludefilter[]` ne sont pas utilisés dans la présente version de FileGuard. Ces champs servent à raffiner le filtrage des fichiers. Les champs `logreads` et `logwrites` permettent un contrôle sur l'affichage des informations. Par défaut, ces valeurs sont initialisées à `VRAI` pour s'assurer d'obtenir toutes les informations pertinentes.

```
typedef struct {
    char    *processfilter;
    char     excludefilter[MAXFILTERLEN];
    BOOLEAN  logreads;
    BOOLEAN  logwrites;
} FILTER, *PFILTER;

FILTER FilterDefinition;
```

Extrait 3: structure du filtre du moniteur

Lorsqu'un usager ajoute ou modifie la liste des fichiers mis sous surveillance, les informations relatives à chaque fichier sont emmagasinées dans la base de registres. La base de registres sert d'entrepôt de stockage temporaire afin de minimiser le temps de développement. De plus, l'environnement Windows NT a, en place, des moyens de gestion efficace de la base de registres (sauvegarde, récupération et sécurité).

Puisque les informations sont stockées dans la base de registres, le moniteur doit récupérer ces informations. L'Extrait 4 montre le traitement appliqué à la base de registres pour créer le filtre et initialiser le pointeur, `processfilter`, sur ce filtre.

```
// Make the filter string containing the whole items of the listview

    try{
    FilterDefinition.processfilter = new char[length*ListView4->Items->Count];
    }
    catch (std::bad_alloc) {  // ENTER THIS BLOCK ONLY IF bad_alloc IS THROWN.
      // YOU COULD REQUEST OTHER ACTIONS BEFORE TERMINATING
      exit(-1);
      }


    if(FilterDefinition.processfilter != NULL)
    {
        *(FilterDefinition.processfilter) = 0;
        Registre->RootKey = HKEY_LOCAL_MACHINE;
        if(Registre->OpenKey(Key,false))
        {
            Registre->GetKeyInfo(Value);
            Registre->GetKeyNames(Buffer);
            Registre->CloseKey();
            for (i=0; i < Value.NumSubKeys;i++)
            {
                Registre->RootKey = HKEY_LOCAL_MACHINE;
                TempKey = Key + "\\" + Buffer->Strings[i];
                if(Registre->OpenKey(TempKey ,false))
                {
               // Read the registry keys for obtaining the name
               // and the associated restrictions
                    int length = strlen(Registre->ReadString("Name").c_str());
                    char *temp = new char[ length + 6];
                    *temp = 0;
                    char *number = new char[sizeof(int)];
                    *number = 0;

               // Concatenate name and restrictions separated by « | »
               // and terminated by « ; »
                    strcat(temp,Registre->ReadString("Name").c_str());
                    strcat(temp,"*|");

                    itoa(Registre->ReadInteger("Flags"),number,10);

                    strcat(temp,number);
                    strcat(temp,";");
                    temp[strlen(temp)] = 0;

               // We have the string containing all the names in the ListView
                    strcat(FilterDefinition.processfilter, StrUpper(temp));
                    delete [] temp,number;
                }
                Registre->CloseKey();
            }
        }


        sprintf( FilterDefinition.excludeprocess, "" );
        sprintf( FilterDefinition.pathfilter, "*" );
        sprintf( FilterDefinition.excludefilter, "");
        FilterDefinition.logreads = true;
        FilterDefinition.logwrites = true;
```

Extrait 4: création du filtre à partir de la base de registres

- **Transmission du filtre**

L'échange d'information entre le moniteur et les gardiens se fait en envoyant un code de contrôle unique au gardien concerné. En utilisant le code de contrôle[32] `IOCTL_FILEMON_SETFILTER`, le gardien FileGuard est en mesure d'interpréter la demande du moniteur. Le gardien récupère les informations en entrée qu sont contenues dans le tampon ayant le type associé à la variable `FilterDefinition`. L'Extrait 5 utilise la fonction `DeviceIoControl()` avec les paramètres `IOCTL_FILEMON_SETFILTER` et `FilterDefinition` pour transmettre les fichiers mis sous surveillance.

```
// Send data to the driver
if ( ! DeviceIoControl(handle, IOCTL_FILEMON_SETFILTER, (PVOID) &FilterDefinition,
                        sizeof(FilterDefinition),NULL, 0, &nb, NULL ))
     MessageBox(NULL,"SetFilter Off","",MB_OK);
```

Extrait 5: transmission du filtre

6.2.2 Définition des réactions

Le rôle du moniteur étant de surveiller les accès à des ressources sensibles, il est essentiel que celui-ci puisse réagir lorsqu'il y aura tentative de violation des restrictions. Pour des raisons de prototypage, trois choix de réaction ont été définis. L'Extrait 6 affiche la définition des réactions. Lorsque l'usager définit les restrictions associées à un fichier mis sous surveillance, il a la possibilité de définir la réaction qui devra être exécutée s'il y a violation des restrictions. Le premier choix, `AUTODENY`, signifie au gardien qu'il devra automatiquement refuser la requête s'il détecte une tentative de violation. Le deuxième choix, `AUTOSUC`, est un moyen d'éviter les restrictions en acceptant toutes les requêtes même celles qui restreignaient l'accès au fichier mis sous surveillance. Ce choix permet de faire des tests de validation et de voir le comportement du moniteur.

Le troisième choix, `MANUAL`, permet à l'usager d'avoir le contrôle sur la réaction et oblige, par le fait même, le gardien à signaler toutes tentatives de violation des restrictions. Lorsque ce choix est envoyé au gardien, celui-ci a la responsabilité de signaler l'événement au moniteur, d'attendre la réponse de l'utilisateur et d'exécuter l'ordre reçu.

[32] L'Annexe VI présente la liste des codes de contrôles utilisés par le moniteur.

```
#define AUTODENY        0x00000100
#define AUTOSUC         0x00000200
#define MANUAL          0x00000400
```

Extrait 6: définition des réactions

- **Ajout des réactions au filtre**

Le filtre contient les éléments nécessaires pour connaître les ressources mises sous surveillance ainsi que la façon dont le gardien doit réagir s'il détecte une requête sur une des ressources sensibles.

Dans le cas du filtre envoyé au pilote FileGuard, l'information se rapportant à la réaction se retrouve dans le champ dédié aux restrictions sur le deuxième octet. Par exemple, la chaîne de caractères suivante est envoyée au pilote FileGuard :

« c:\\autoexec.bat*|17f;d:\\tom.txt*|43e; »

Cette chaîne contient un premier élément qui est le fichier **« c:\\autoexec.bat* »** à surveiller. Le champ dédié aux restrictions contient la valeur **17f**. Le premier octet, **7f**, correspond au total des restrictions appliquées sur le fichier. Le deuxième octet, **1** ou **01**, représente la réaction. Dans ce cas-ci, le gardien reçoit l'ordre `AUTODENY` (voir Extrait 6).

6.2.3 Création du mécanisme dédié aux traitements des violations – mode Usager

La gestion du monitorage se fait à deux niveaux, en mode Usager et en mode Noyau. La présente section couvre le mode Usager et explique les moyens mis en place pour permettre aux deux niveaux de s'échanger des informations. Essentiellement, le mécanisme de gestion se divise en trois parties. Les parties comprennent la création d'un événement, la création d'un « thread » dédié à l'écoute d'un gardien et le traitement de la violation.

- **Création d'un événement**

Le concept d'événement mis en place dans l'environnement Windows NT permet à des processus ou des « threads » d'être avertis d'un fait venant de se produire. L'Extrait 7 montre la fonction `CreateEvent()` qui permet d'obtenir un identifiant « handle » sur un événement portant un nom précis, `DAMON_ALERT`. Naturellement, cet événement aura été créé auparavant. Par ce nom, un processus ou un « thread » pourra signifier au système d'exploitation, au moyen de la fonction `WaitForSingleObject()`, qu'il attend

que cet événement soit signalé pour continuer son travail. Un événement peut être dans un état « signalé » ou « non-signalé ». Si un « thread » attend un événement, c'est que l'événement est dans l'état « non-signalé ». Dès que l'état changera, c'est-à-dire qu'il passera de « non-signalé » à « signalé », le « thread » pourra continuer son travail.

```
HANDLE hEvent;

hEvent = CreateEvent(NULL,FALSE,FALSE,"DAMON_ALERT");
```

Extrait 7: création de l'événement signalé lors d'une violation

- **Création d'un « thread »**

Au démarrage du moniteur, un « thread » est généré et dédié à l'écoute d'un gardien. L'Extrait 8 montre le code mis en place dans le moniteur et affecté au pilote FileGuard. Au démarrage du moniteur, la fonction `CreateThread()` est appelée pour initialiser les éléments nécessaires. Un des éléments est la déclaration de la fonction `MessageAlert()` qui correspond au code qui sera exécuté dès que le « thread » sera créé. La fonction `MessageAlert()` a pour rôle d'envoyer un message au moniteur pour qu'il affiche une boîte de dialogue. Le message ne sera envoyé que si l'événement `DAMON_ALERT` est signalé. Dans la fonction `MessageAlert()`, l'appel à la fonction `WaitForSingleObject()` permet au « thread » d'attendre que l'événement `DAMON_ALERT` soit signalé.

```
#define WM_ALERTFROMDRIVER  (WM_USER)+2000

 HANDLE hThread,hEvent;
DWORD dwThreadId,dwsize = 0,valeur;
bool loop = true;


hThread = CreateThread(NULL,0,MessageAlert,NULL,0,&dwThreadId);
.
.
.
//---------------------------------------------------------------------------
DWORD WINAPI MessageAlert (LPVOID nothing)
{
        while(loop)
        {
                WaitForSingleObject(hEvent,INFINITE);
                PostMessage(Application->Handle,WM_ALERTFROMDRIVER,0,0);
                ResetEvent(hEvent);
        }

        return 0;
}
//---------------------------------------------------------------------------
```

Extrait 8: déclaration de la fonction MessageAlert() associée au « thread »

- **Traitement de la violation**

Au moment où un gardien signale un événement parce que la réaction a été fixée à MANUAL, un message est envoyé au moniteur pour qu'il affiche une boîte de dialogue indiquant les informations se rapportant à la tentative de violation. La Figure 22 montre la fenêtre permettant de traiter l'événement. La partie du haut affiche le nom de l'usager demandant l'accès à la ressource, le nom de l'application, le nom de la ressource et l'action à être exécutée.

À l'aide de ses informations, l'usager peut réagir en sélectionnant l'action appropriée. La partie du bas permet à l'usager de faire un seul choix et, en appuyant sur le bouton « Apply », d'envoyer son choix au pilote qui a déclenché l'événement. L'usage de cette fenêtre est réservé à des situations particulières où l'utilisateur veut être informé et avoir la possibilité de choisir la réaction à appliquer.

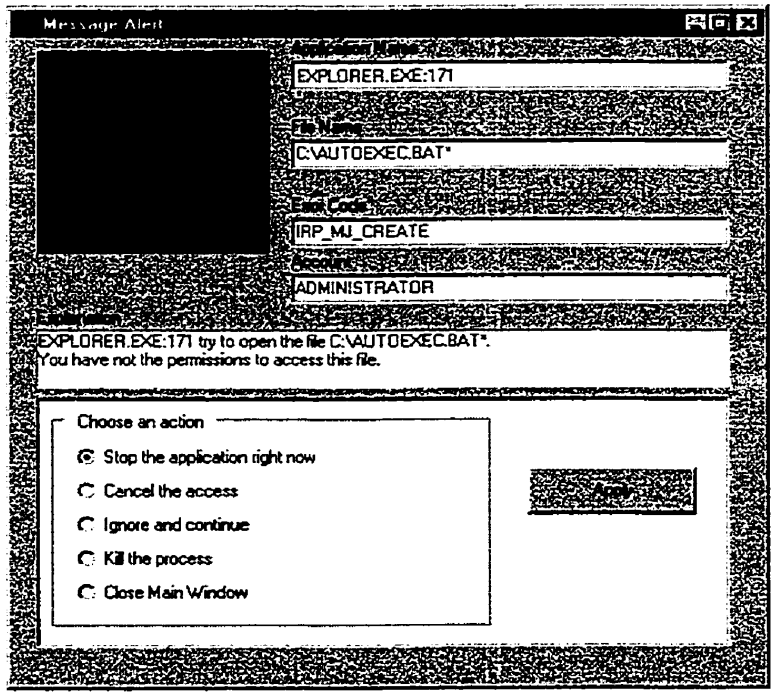


Figure 22: boîte de dialogue pour le traitement des violations

La section suivante présente le concept d'automate mis en place pour gérer les événements. Normalement, selon les règles qui ont été définies, c'est l'automate qui réagira aux événements déclenchés par les gardiens à la place de l'usager. L'usager sera remplacé par un automate et une série de règles de sécurité qui permettront de le construire.

6.2.4 Création d'un automate pour valider les requêtes faites aux ressources

Le rôle du moniteur étant de gérer les accès aux ressources sensibles, celui-ci doit avoir en tout temps une vue d'ensemble du système. Pour l'aider dans sa gestion, le moniteur a des gardiens postés à des endroits stratégiques. Pour que le moniteur puisse réagir correctement, un automate est généré pour aider le moniteur dans sa gestion des requêtes faites aux ressources. Lorsqu'un gardien détecte une action faite à une ressource sensible, il signale le fait au moniteur. Ce dernier récupère les données, demande à l'automate et réagit en fonction de l'état dans lequel il se trouve. La réaction du moniteur sera transmise au gardien concerné.

L'Extrait 9 présente la classe Automate. Dans cette version, l'automate est créé à partir d'un fichier contenant les règles de sécurité. Chaque fois qu'un gardien signale un événement au moniteur, ce dernier exécute la fonction `MakeTransition()` afin de déterminer si l'action est permise et si l'automate doit changer d'état à cause de l'événement.

```
typedef struct
{
        unsigned int StartState;
        unsigned int Action;
        AnsiString Ressource;
        unsigned int FinishState;
}Transition;


////////////////////////////////////////////////////////////////////////////
//Class SecurityAutomata
//
//
//
////////////////////////////////////////////////////////////////////////////


class SecurityAutomata
{
        //data members
private:
        unsigned int InitialState;              //Initial state of the automata
        unsigned int CurrentState;              //Current state of the automata
        Transition   TransitionFunction[2];     //Transition function of the automata
                                                //on considere seulement deux transitions
pour l'instant
        //member functions
public:
        unsigned int GetCurrentState(){return CurrentState;}
        SecurityAutomata()
        {
                //read the data form a file
                int NumberTransitions;
                char InputString[150];
                FILE*stream;
                stream = fopen("Automate.txt", "r");

                //read the initial state
                fseek(stream, 14, SEEK_CUR);fscanf(stream, "%ui", &InitialState);
                //read the number of transitions
```

```
            fseek(stream, 42, SEEK_CUR);
            //for each transition
            while(NumberTransitions > 0)
            {
                    NumberTransitions--;
                    fscanf(stream, "%ui",
                           &TransitionFunction[NumberTransitions].StartState);
                    fscanf(stream, "%x",
                           &TransitionFunction[NumberTransitions].Action);
                    fscanf(stream, "%s", InputString);
                       TransitionFunction[NumberTransitions].Ressource = InputString;
                    fscanf(stream, "%ui",
                           &TransitionFunction[NumberTransitions].FinishState);
            }

            //end of read
            fclose(stream);
            //CurrentState Initialisation
            CurrentState = InitialState;
    };
    ~SecurityAutomata()
    {
    };
    //Member function MakeTransition
    unsigned int MakeTransition(unsigned int Action, AnsiString Ressource)
    {
            int i;
            for(i=0; i<2; i++) //on considere seulement deux transitions
            {
                    if(CurrentState == TransitionFunction[i].StartState)
                    {
                            if(Action == TransitionFunction[i].Action)
                            {
                                    if(Ressource.AnsiCompareIC(TransitionFunction[i].Ressource))
                                    {
                                            CurrentState = TransitionFunction[i].FinishState;
                                    }
                            }
                    }
            }
            return CurrentState;
    }
};
```

Extrait 9: implantation de l'automate

En résumé, cette partie du chapitre explique les différents éléments nécessaires à la gestion des requêtes. Il est important de noter que les réactions retournées aux filtres peuvent provenir de deux intervenants. Le premier, l'usager, a la possibilité de dicter son propre choix dès que la boîte de dialogue apparaît. L'avantage de ce mode de fonctionnement est qu'il permet à l'usager d'éliminer la rigidité des règles de sécurité. L'inconvénient est qu'il oblige l'usager à être présent à chaque tentative de violation. Par contre, en utilisant l'autre intervenant, l'automate, on élimine la présence de l'usager pour définir les réactions mais on doit préciser les règles de sécurité qui devront être exécutées par l'automate.

En somme, en ajoutant un automate au moniteur, on bonifie la qualité de la surveillance et du contrôle à la condition que les règles de sécurité qui définissent l'automate soient bien construites.

6.3 Implantation au niveau Noyau – le pilote

La surveillance et le contrôle des politiques de sécurité se font également au niveau du noyau ou, plus spécifiquement, à l'aide de pilotes positionnés en mode noyau. Puisqu'il y a un échange constant d'informations entre le moniteur et les pilotes, il est important de connaître les traitements associés à ces informations.

Un pilote, par sa conception, est un programme autonome qui traite les données qui lui sont transmises. De plus, un pilote peut recevoir des directives d'une entité supérieure lui permettant de travailler différemment. Normalement, seule cette entité connaît les directives à transmettre au pilote.

Dans ce projet, les pilotes jouent un rôle de gardien et exécutent les ordres transmis par le moniteur. Seul le moniteur connaît les directives à transmettre. Ces directives correspondent à des codes de contrôle. Les sections suivantes couvrent les aspects reliés au transfert des informations ainsi qu'à certains traitements effectués par le gardien FileGuard affecté à la surveillance des fichiers. Dans un premier temps, un filtre est transmis entre le moniteur et le FileGuard. Ce filtre permet de mettre en place la liste des fichiers sensibles.

Dans un deuxième temps, le mécanisme de surveillance est expliqué pour comprendre la synchronisation nécessaire entre le moniteur et le gardien lorsque celui-ci détecte une tentative d'accès à un fichier mis sous surveillance.

6.3.1 Traitement du filtre

Un filtre correspond à des informations concernant les fichiers ou autre à surveiller. Le filtre est généré par le moniteur et correspond à une structure de données précise. L'Extrait 10 montre les éléments de la structure.

```
//
// Filter definition
//
typedef struct {
    PCHAR    includefilter;
    char     excludefilter[MAXFILTERLEN];
    BOOLEAN  logreads;
    BOOLEAN  logwrites;
} FILTER, *PFILTER;
```

Extrait 10: structure du filtre

Le champ `includefilter` est un pointeur sur une chaîne de caractères qui contient les noms de fichiers à surveiller et leurs restrictions. La chaîne a cette forme :

« c:\\autoexec.bat*|17f;d:\\tom.txt*|43^{c}; »

Les caractères « | » et « ; » sont nécessaires pour délimiter chaque fichier et leurs restrictions.

Dans le présente version de FileGuard, les autres champs ne sont pas utilisés de façon efficace. Il serait toujours possible de surveiller tous les fichiers sauf ceux qui ne semblent d'aucun intérêt. Le champ `excludefilter[]` permet ce type de surveillance. Les champs `logreads` et `logwrites` permettent de signifier au gardien si le moniteur désire être avisé seulement lorsqu'il s'agit d'une lecture (logreads) ou d'une écriture (logwrites).

```
//
// Structure for specific data exchange between the main application
// and the driver for the security policies
//
typedef struct {
        int restrictions;
        PCHAR name;
} POLICIES, *pPOLICIES;
```

Extrait 11: structure utilisée pour convertir le filtre en élément du tableau

Une fois que le filtre est créé, le moniteur l'envoie au gardien. Ce dernier reçoit l'ordre de prendre ce filtre et de générer un tableau. Ce tableau contient, en mémoire, les noms et restrictions des fichiers à surveiller et les informations sont stockées en utilisant la structure de données présentée à l'Extrait 11. Ce tableau deviendra le point névralgique de surveillance du gardien. Sans ce tableau, le gardien n'est plus en mesure de contrôler les accès. L'Extrait 12 présente la fonction `MakeFilterArray()` qui permet de décomposer le filtre pour le transformer en *tableau de filtres*.

```
//--------------------------------------------------------------------
//
// MakeFilterArray
//
// Takes a filter string and splits into components (a component
// is seperated with a ';'). A component is in this form:
// "name*|double;" where "name*" is the name of the file to be under
// monitoring and "double" is a number corresponding to the restrictions
// associated to "name*"
//
//--------------------------------------------------------------------
```

```
                    pPOLICIES FilterArray[],
                    PULONG NumFilters )
{
        PCHAR number;
        PCHAR filterStart;
ULONG filterLength;
        ULONG numberLength;

    //
    // Scan through the process filters
    //
    filterStart = FilterString;
    while( *filterStart )
    {
        filterLength = 0;
        while( filterStart[filterLength] &&
               filterStart[filterLength] != '|' &&
               filterStart[filterLength] != ';' ) {

            filterLength++;
        }
        //
        // Ignore zero-length components
        //
        if( filterLength )
        {
            FilterArray[ *NumFilters ] =
                ExAllocatePool( NonPagedPool, sizeof(POLICIES));
            FilterArray[ *NumFilters ]->name =
                ExAllocatePool( NonPagedPool, filterLength);

            RtlMoveMemory( FilterArray[ *NumFilters ]->name,filterStart, filterLength );


           // Copy the number associated to restrictions
            numberLength = (filterLength + 1);
            while( filterStart[numberLength] &&
                   filterStart[numberLength] != ';' ) {
                   numberLength++;
            }

            number = ExAllocatePool(NonPagedPool,numberLength - filterLength);
            RtlZeroMemory(number,numberLength - filterLength);
            RtlMoveMemory(number, filterStart + filterLength + 1,numberLength -
                          filterLength - 1);
           // Be sure to have a null terminated string
           number[strlen(number)] = 0;
           FilterArray[ *NumFilters ]->restrictions = atoi(number);
           ExFreePool(number);


           // Create a null terminated string
           FilterArray[ *NumFilters ]->name[filterLength] = 0;
           (*NumFilters)++;
        }



        //
        // Are we done? Check filterStart from its new position to see
              // if we have read the whole string.
        //
        if( !filterStart[numberLength+1] ) break;

        //
        // Move to the next component (skip over ';')
```

```
        filterStart += numberLength+1;
    }
}
```

Extrait 12: fonction MakeFilterArray()

6.3.2 Création du mécanisme dédié aux traitements des restrictions – mode Noyau

Le rôle d'un gardien est d'intercepter les requêtes faites à des ressources mises sous surveillance. Lorsqu'un gardien détecte ces requêtes, il a la responsabilité de réagir immédiatement en le signalant au moniteur. À ce moment, le gardien doit attendre la réponse du moniteur. Dès que le moniteur retourne un ordre au gardien, celui-ci a la tâche d'exécuter l'ordre du moniteur. Ce mécanisme, mis en place, permet de synchroniser le travail des gardiens, d'aviser le moniteur des accès aux ressources mises sous surveillance, de centraliser les informations et d'ajuster l'état du moniteur.

Pour mettre en place ce mécanisme, il est nécessaire de synchroniser le travail de tous et chacun. De plus, il faut établir un procédé qui permet, à tout moment, de signaler au moniteur toutes détections par le gardien. Le concept d'événement (« event »), sous Windows NT, permet d'obtenir cet effet.

- **Création d'un événement**

Un événement (« event »), dans l'environnement Windows NT, correspond à un objet partagé entre toutes les applications et est utilisé pour synchroniser le travail des applications. Cet objet possède un état, « signalé » ou « non-signalé ». L'application qui désire utiliser un événement doit le signifier en utilisant les appels de fonctions nécessaires.

L'Extrait 13 est retiré de la fonction `DriverEntry()` de FileGuard. Dans ce cas-ci, il y a deux événements qui sont créés. Le premier, `DAMON_ALERT`, servira à signaler au moniteur la détection d'un accès à un fichier mis sous surveillance. Le deuxième, `DAMON_ALERT_FOR_DRIVER`, sera utilisé localement pour synchroniser le travail du driver FileGuard. Cet événement permettra au driver d'enclencher une période d'attente, période durant laquelle le driver attendra une réponse provenant du moniteur. Les zones grises montrent la création des événements.

À noter que chaque événement est initialisé à l'état « non-signalé » à l'aide de la fonction `KeClearEvent()`.

```
    //
```

```
        //
        RtlInitUnicodeString(&eventName,L"\\BaseNamedObjects\\DAMON_ALERT");
        eventForApplication = IoCreateNotificationEvent(&eventName, &handleEvent);
        if(!eventForApplication)
        {
                IoDeleteDevice(guiDevice);
                return ntStatus;
        }

        KeClearEvent(eventForApplication);

        RtlInitUnicodeString(&eventNameDriver,
                L"\\BaseNamedObjects\\DAMON_ALERT_FOR_DRIVER");
        eventForDriver = IoCreateNotificationEvent(&eventNameDriver, &handleEventDriver);
        if(!eventForDriver)
        {
                IoDeleteDevice(guiDevice);
                return ntStatus;
        }

        KeClearEvent(eventForDriver);
```

Extrait 13: initialisation des événements

- **Fonctions GetPermissions() et GetReaction()**

À la Figure 20, page 62, on remarque la présence de la fonction `GetPermissions()` à l'intérieur de la routine d'achèvement. La fonction `GetPermissions()` a pour rôle de recueillir certaines informations portant sur la requête demandée. Puisque cette fonction fait partie de la routine d'achèvement, cela signifie que l'appel de cette fonction a été programmé pour faire suite à l'interception de la requête par le gardien. Cette requête vise une ressource mise sous surveillance.

Dans un premier temps, la fonction reçoit en paramètre le nom du fichier demandé par la requête et l'action désirée exprimée sous forme de code de contrôle. Par la suite, la fonction récupère le nom du processus qui demande l'accès au fichier à l'aide de la fonction `FilemonGetProcess()`. Dans un deuxième temps, une comparaison est effectuée entre le tableau des filtres (voir la section 6.3.1) et leurs restrictions et le nom du fichier reçu en paramètre.

Dès que la comparaison retourne une valeur positive, un branchement est effectué en fonction de l'action reçue. Ce branchement permet de faire l'appel de la fonction `GetReaction()` avec tous les paramètres nécessaires pour traiter la requête. L'Extrait 14 affiche le code de la fonction `GetPermissions()`.

```
NTSTATUS GetPermissions(PCHAR FileName, UCHAR MajorFunction)
```

```
        ULONG i;
        CHAR AppName[256];

        FilemonGetProcess(AppName);

        strcat(FileName,"*");

        for( i=0;i< NumIncludeFilters; i++)
        {
                if(!strcmp(IncludeFilters[i]->name, _strupr(FileName)))
                {
                        switch(MajorFunction)
                        {
                        case IRP_MJ_CREATE:
                         return GetReaction(AppName,FileName,MajorFunction,i,CREATEFILE);

                        case IRP_MJ_READ:
                         return GetReaction(AppName,FileName,MajorFunction,i,READFILE);

                        case IRP_MJ_WRITE:
                         return GetReaction(AppName,FileName,MajorFunction,i,WRITEFILE);

                        default:
                         return STATUS_SUCCESS;
                        }

                }

        }
        return STATUS_SUCCESS;
}
```

Extrait 14: fonction GetPermission()

Une fois l'appel fait à la fonction `GetReaction()`, celle-ci entre en action. Son rôle est de valider l'action demandée et, si nécessaire, de demander, au moniteur s'il accepte la requête. Dans un premier temps, la fonction reçoit en entrée le nom du processus qui désire accéder au fichier mis sous surveillance, le nom du fichier, l'action à être commise, la position du fichier dans le tableau des filtres et leurs restrictions associées et un masque pour valider l'action.

À partir du tableau des filtres, la fonction récupère les restrictions associées au fichier ainsi que la réaction qui devra être exécutée en cas de détection d'un accès à un fichier mis sous surveillance. Ce sont des opérations binaires qui permettent de récupérer ces informations. Les données du tableau des filtres sont stockées dans une variable globale, `IncludeFilters[]`, contenant une structure de données dont le champ des restrictions. À l'aide d'un index, la fonction peut obtenir la valeur hexadécimale des restrictions et procéder à une transformation pour obtenir séparément les restrictions (`tempRestrictions`) et la réaction (`tempReaction`).

Par la suite, la fonction vérifie à l'aide des restrictions et du masque si la restriction est valide. Par exemple, si le masque permet de valider la restriction READFILE et que la réponse est positive à la comparaison binaire suivante

```
tempRestrictions & READFILE,
```

cela signifie qu'il y a une restriction de lecture sur le fichier. Ensuite, une comparaison binaire est effectuée pour connaître le type de réaction à laquelle le gardien doit répondre. Dans la présente version, il existe trois types de réaction :

AUTODENY	:	refuser automatiquement la requête sans le demander au moniteur et retourne le statut STATUS_ACCESS_DENIED.
AUTOSUC	:	accepter automatiquement la requête sans le demander au moniteur. Ne se préoccupe pas de la restriction et retourne le statut STATUS_SUCCESS.
MANUAL	:	demander au moniteur (plus précisément à l'usager) et retourne le statut défini par l'usager.

Le type de réaction MANUAL demande plus d'attention car il nécessite une synchronisation entre les différentes fonctions du filtre et le moniteur. L'utilisation des fonctions `KeEnterCriticalRegion()` et `KeLeaveCriticalRegion()` permettent de bloquer l'accès à une zone mémoire et de la débloquer par la suite. Il est important de se rappeler que le travail effectué par la fonction `GetReaction()` peut être exécuté en parallèle et en même temps qu'une autre fonction désirant utiliser la même zone mémoire. Pour des raisons évidentes de protection des informations, il est essentiel que chaque fonction du programme puisse accéder à la zone mémoire commune de façon ordonnée. Cette zone mémoire contient les informations qui seront transmises au moniteur si le gardien doit demander à l'usager la réaction désirée. Ces informations contiennent le nom de l'application désirant accéder à un fichier mis sous surveillance, le nom du fichier et l'action.

Lorsque les informations sont inscrites dans la structure qui sera retournée à l'utilisateur, la fonction appelle `KeSetEvent(eventForApplication)`[33] pour signaler au moniteur qu'il vient de détecter une tentative d'accès et qu'il est prêt à transmettre les informations. Ensuite, c'est à son tour de se mettre en attente, `KeWaitForSingleObject(eventForDriver)`[34]. La fonction `GetReaction()` sera en attente tant que l'usager n'aura pas répondu. Au moment où l'usager aura répondu, la fonc-

[33] Cette fonction met l'événement DAMON_ALERT à l'état « signalé ».

[34] Cette fonction attend que l'événement DAMON_ALERT_FOR_DRIVER soit à l'état « signalé » avant de poursuivre.

tion `GetReaction()` se réactivera et retournera le statut défini par l'usager à l'aide de la variable `ActionToDo`.

L'Extrait 15 montre la fonction incluse dans la routine d'achèvement et qui gère les restrictions et réactions émises par le moniteur. La zone grise montre les appels de fonctions nécessaires à la synchronisation lorsque la réaction est définie à MANUAL.

```
NTSTATUS GetReaction(PCHAR AppName, PCHAR FileName, UCHAR MajorFunction, ULONG index,
ULONG xxxFILE)
{
        ULONG tempRestrictions = 0;
        ULONG tempReaction = 0;

// Skip the reaction
        tempRestrictions = (IncludeFilters[index]->restrictions & FULLRESTRICTIONS);

// Skip the restrictions
        tempReaction = (IncludeFilters[index]->restrictions & ~FULLRESTRICTIONS);

        if(tempRestrictions & xxxFILE)
        {
                if(tempReaction & MANUAL)
                {
                        KeEnterCriticalRegion();
                        strcpy(OutBuffer.AppName,AppName);
                        strcpy(OutBuffer.FileName,FileName);
                        OutBuffer.Code = MajorFunction;
                        KeLeaveCriticalRegion();
                        KeSetEvent(eventForApplication,0,FALSE);
                        KeWaitForSingleObject(eventForDriver,
                                              Executive,KernelMode,FALSE,NULL);
                        KeClearEvent(eventForDriver);
                        return ActionToDo;
                }
                else if(tempReaction & AUTODENY)
                                return STATUS_CANCELLED;//STATUS_ACCESS_DENIED;
                        else
                                return STATUS_SUCCESS;
        }
        else return STATUS_SUCCESS;
}
```

Extrait 15: fonction GetReaction()

- **Traitement de la réaction**

Si le gardien a reçu la consigne de signaler l'événement associé à une tentative de violation de règles de sécurité, c'est-à-dire que le bit MANUAL contient la valeur 1 dans la variable `IncludeFilter[]→restrictions`, cela signifie que le contrôle des opérations devra être transféré au moniteur au moment de la détection.

Lorsque l'événement DAMON_ALERT est signalé, le moniteur demande au gardien correspondant de lui envoyer les informations. L'Extrait 16 présente le traitement lié aux codes de contrôles IOCTL_FILEMON_GETMESSAGE et IOCTL_FILEMON_SETMESSAGE. IOCTL_FILEMON_GETMESSAGE permet de demander au gardien d'envoyer les informations relatives à la tentative de violation des règles de sécurité. À l'aide de ces informations, l'usager ou l'automate pourra réagir.

Dès que la réponse est connue, un message est envoyé au gardien à l'aide du code de contrôle IOCTL_FILEMON_SETMESSAGE. Ce message contient la réponse de l'usager ou de l'automate qui sera affectée à la variable ActionToDo. Par la suite, l'événement DAMON_ALERT_FOR_DRIVER est signalé pour permettre à la fonction GetReaction() de continuer son travail.

```
//--------------------------------------------------------------------
//
// FilemonFastIoDeviceControl
//
//--------------------------------------------------------------------
BOOLEAN  FilemonFastIoDeviceControl( IN PFILE_OBJECT FileObject, IN BOOLEAN Wait,
                                     IN PVOID InputBuffer, IN ULONG InputBufferLength,
                                     OUT PVOID OutputBuffer, IN ULONG OutputBufferLength,
                                     IN ULONG IoControlCode,
                                     OUT PIO_STATUS_BLOCK IoStatus,
                                     IN PDEVICE_OBJECT DeviceObject )
{
    BOOLEAN             retval = FALSE;
    BOOLEAN             logMutexReleased;
    PHOOK_EXTENSION     hookExt;
    PLOG_BUF            oldLog, savedCurrentLog;
    CHAR                fullPathName[MAXPATHLEN], name[PROCNAMELEN], errorBuf[ERRORLEN];
    KIRQL               oldirql;
    LARGE_INTEGER       timeStampStart, timeStampComplete, timeResult;
    ANSI_STRING         FileName;

    hookExt = DeviceObject->DeviceExtension;


    if ( hookExt->Type == GUIINTERFACE ) {

        //
        // Its a message from our GUI!
        //
        IoStatus->Status      = STATUS_SUCCESS; // Assume success
        IoStatus->Information = 0;      // Assume nothing returned

        switch ( IoControlCode ) {

      case IOCTL_FILEMON_GETMESSAGE:

        //
        // Driver is sending the data concerning the violation
        //
             KeEnterCriticalRegion();

            if ( OutputBufferLength >= sizeof(OUTPUTBUFFER)) {
                memcpy( OutputBuffer, &OutBuffer, sizeof(OUTPUTBUFFER) );
                IoStatus->Information = sizeof(ULONG);
```

```
            IoStatus->Status = STATUS_BUFFER_TOO_SMALL;
        }
        KeLeaveCriticalRegion();

        break;

    case IOCTL_FILEMON_SETMESSAGE:

    //
    // Gui is sending the user option
    //
    DbgPrint(("Filemon: set message\n"));

        if( InputBufferLength >= sizeof(OutputBuffer) ) {
            ActionToDo = *(NTSTATUS*)InputBuffer;
            KeSetEvent(eventForDriver,0,FALSE);
            IoStatus->Status = STATUS_SUCCESS;
        } else {
            IoStatus->Status = STATUS_BUFFER_TOO_SMALL;
        }
        break;
```

Extrait 16: traitement des codes de contrôle associés aux réactions

Comme on le constate, la connaissance de l'environnement Windows NT est essentielle. L'échange d'informations entre les deux niveaux, Usager et Noyau, requiert des procédures particulières. Il n'est pas possible pour une application d'accéder directement aux filtres sans passer par les appels de fonctions du noyau. Il faut donc synchroniser tous ces transferts d'informations et s'assurer que les appels sont faits dans le bon ordre.

Le mécanisme de surveillance et de contrôle joue donc un rôle essentiel permettant ainsi au moniteur de bien faire son travail et d'être informé à temps par les filtres lorsque survient une violation des règles de sécurité ou plus généralement des politiques de sécurité.

PARTIE III – IMPLANTATION DE DAMON

Chapitre 7

IMPLANTATION DE DAMON

DaMon (« Dynamic analysis Monitoring ») est le résultat de tout le travail amorcé précédemment. La présente version est un prototype évolutif dans lequel des fonctionnalités plus avancées seront implantées. DaMon est composé d'un moniteur, c'est-à-dire une interface graphique permettant de surveiller et contrôler les requêtes faites au système d'exploitation. Cette interface se situe au niveau Usager. DaMon est aussi composé de drivers ou filtres qui, eux, sont situés au niveau Noyau. Ces filtres, FileGuard, PortGuard. RegGuard et ProcGuard, permettent respectivement de surveiller les ressources critiques fichiers, ports de communication, base de registres et les processus.

Les sections suivantes présentent le fonctionnement général de l'application en montrant les fonctionnalités actives.

7.1 Fonctionnement général

L'architecture présentée à la Figure 23 propose un modèle correspondant aux caractéristiques mentionnées dans les chapitres précédents. Le noyau du logiciel (DaMon.exe) communique avec toutes les parties (modules) du logiciel. Certains modules sont implantés sous forme de DLL (« Dynamic Link Library ») qui seront chargés sur demande par le noyau. INSTDRV.DLL a pour rôle de charger et d'initier la communication entre le noyau et les différents drivers. Ce rôle ne sera joué qu'une seule fois, soit au démarrage de l'application. L'appel à l'installation des drivers se fera par l'intermédiaire de la fonction `LoadDeviceDriver()`.

La base de données PS[35] (Politiques de Sécurité) est en relation directe avec DaMon. Cette base de données contient les informations qui seront transmises aux différents filtres pour qu'ils puissent exercer une surveillance sur les ressources. De plus, la base de données PS transmet à DaMon les règles de sécurité nécessaires pour construire l'automate l'aidant dans sa gestion de surveillance et de contrôle.

[35] Dans cette présente version du prototype et pour des raisons de gestion, les politiques de sécurité sont insérées dans la base de registres. La base de registres est une zone de stockage mise en place par les concepteurs de Windows NT avec des fonctionnalités de sécurité tant au niveau de l'accès à ces données qu'au niveau du mécanisme de stockage.

En somme, DaMon est au cœur de l'activité et c'est lui qui dirige les échanges d'information entre les différentes composantes du modèle.

Les échanges d'informations entre DaMon et les drivers se font par l'envoi d'un code de contrôle au driver, par l'appel de fonctions dans une librairie spécialisée[36] (« Packet.dll ») ou par la passation de données déposées dans un tampon.

INTERFACE.DLL regroupe toutes les fonctions d'affichage nécessaires pour la gestion du DaMon. Puisque DaMon s'exécutera en mode ***service***[37] sous Windows NT, il est préférable de réduire le code exécutable et d'appeler, sur demande, les fonctions d'affichage. Cette fonctionnalité n'est pas encore implantée.

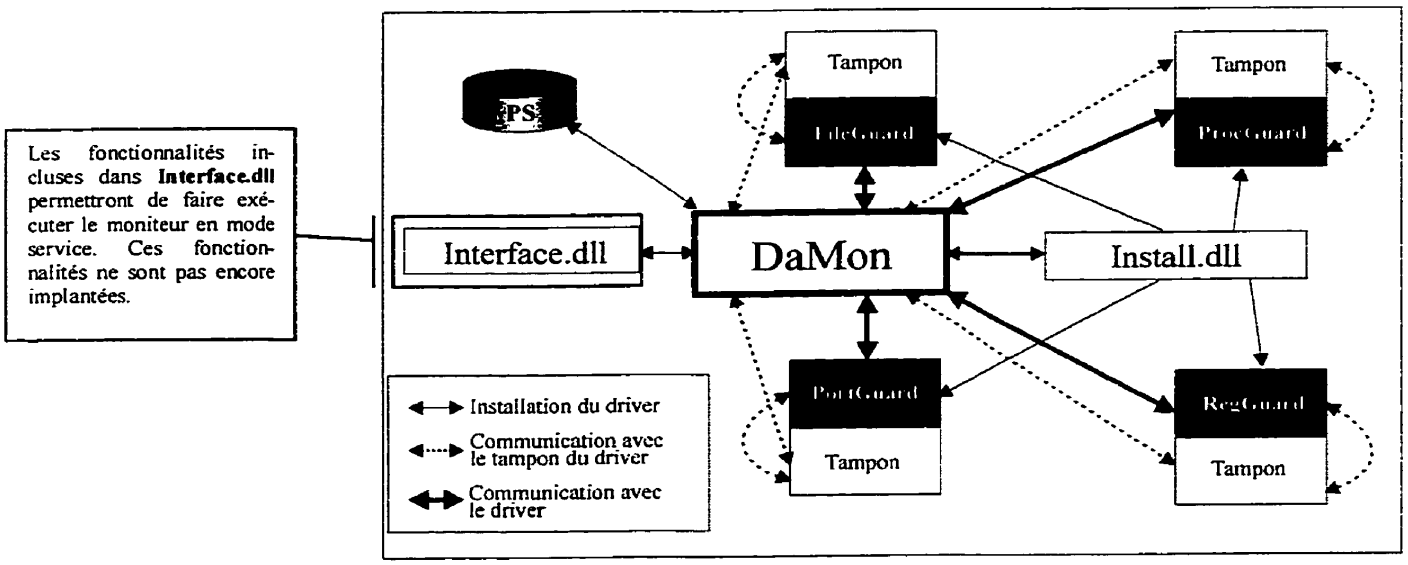


Figure 23: architecture de DaMon

La Figure 24 présente la fenêtre principale de DaMon. Cette fenêtre est divisée en deux parties : la partie gauche représente les boutons de menus et est fixe. La partie droite affiche les différents onglets nécessaires à la gestion de DaMon.

Les sous-sections suivantes présentent les écrans de DaMon pour les fonctionnalités déjà implantées. À noter que certains onglets sont définis mais leurs fonctionnalités ne sont pas encore introduites.

---

[36] L'Annexe VI présente les codes de contrôle et les appels de fonctions utilisés pour les communications entre DaMon et les filtres.

[37] Une application qui s'exécute en mode service agit comme un serveur et peut être démarrée au lancement du système d'exploitation. Lorsque DaMon sera implanté comme su service, il aura la possibilité d'intercepter les premières requêtes qui seront demandées au système d'exploitation.

7.2 Définition des menus

- **Bouton « Policies »** : ce bouton permettra l'accès à la gestion des politiques de sécurité.
- **Bouton « Resources »** : ce bouton permet de visualiser les restrictions appliquées sur les ressources. De plus, il est possible de visualiser les requêtes faites à ces ressources.
- **Bouton « Actions »** : ce bouton ne permet que l'installation des différents filtres.
- **Bouton « Status »** : ce bouton permet d'accéder au tableau de bord de la machine et de voir les niveaux d'activité des différentes ressources.
- **Bouton « Log Files »** : ce bouton permet d'ouvrir une fenêtre qui affiche le contenu de différents fichiers d'activité sur les ressources.

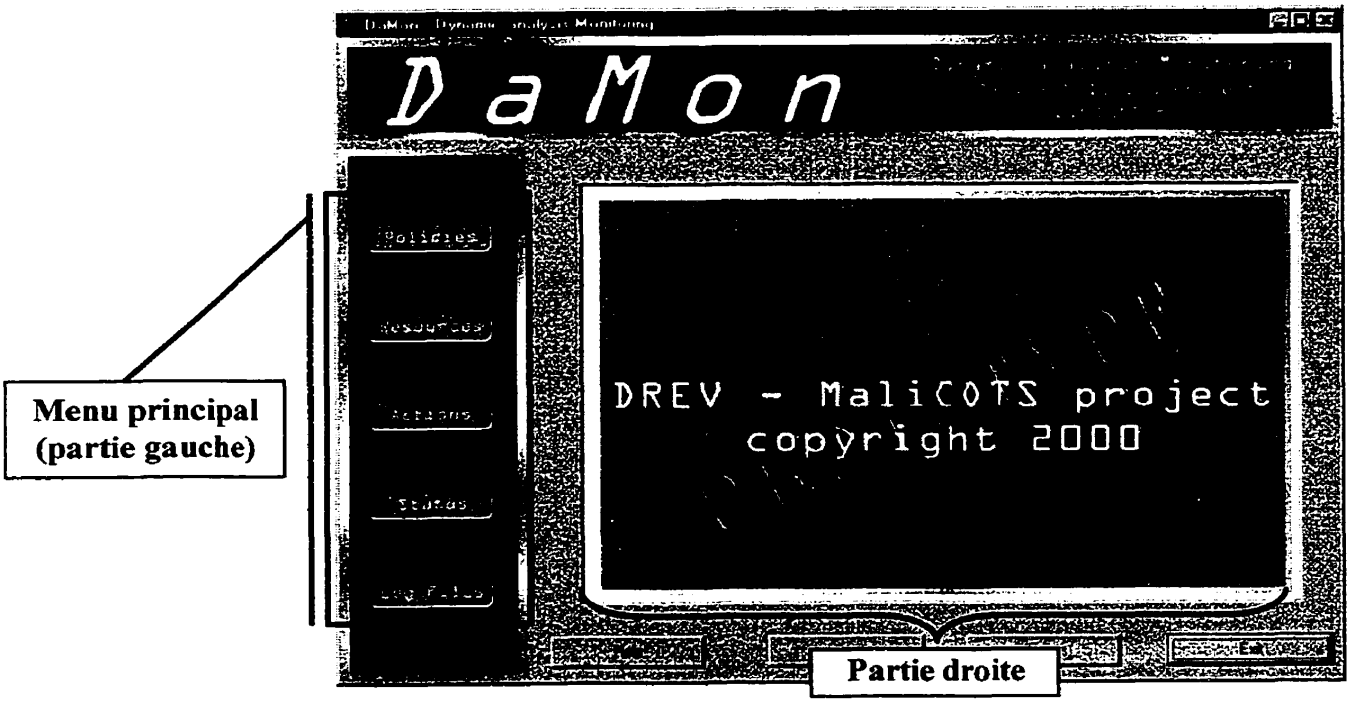


Figure 24: fenêtre principale de moniteur DaMon

7.2.1 Surveillance des fichiers

Les deux sous sections suivantes montrent les écrans pour définir les fichiers qui devront être surveillés ainsi que les restrictions associées et la fenêtre pour visualiser en temps réel les opérations de base faites sur les fichiers surveillés.

- **Mise en service de la surveillance des accès aux fichiers**

En choisissant le menu « RESOURCES » et en sélectionnant l'onglet « Files », on obtient l'écran affiché à la Figure 25. Cet onglet permet de choisir les fichiers sensibles, les restrictions et la réaction associées à chacun de ces fichiers. Lorsque la définition du fichier choisi est complétée, il suffit de peser sur le bouton « Add ». Ce bouton permet d'inclure le fichier dans la liste et de l'inscrire dans la base de données. Cette opération est répétée autant de fois qu'il y a de fichiers à inscrire.

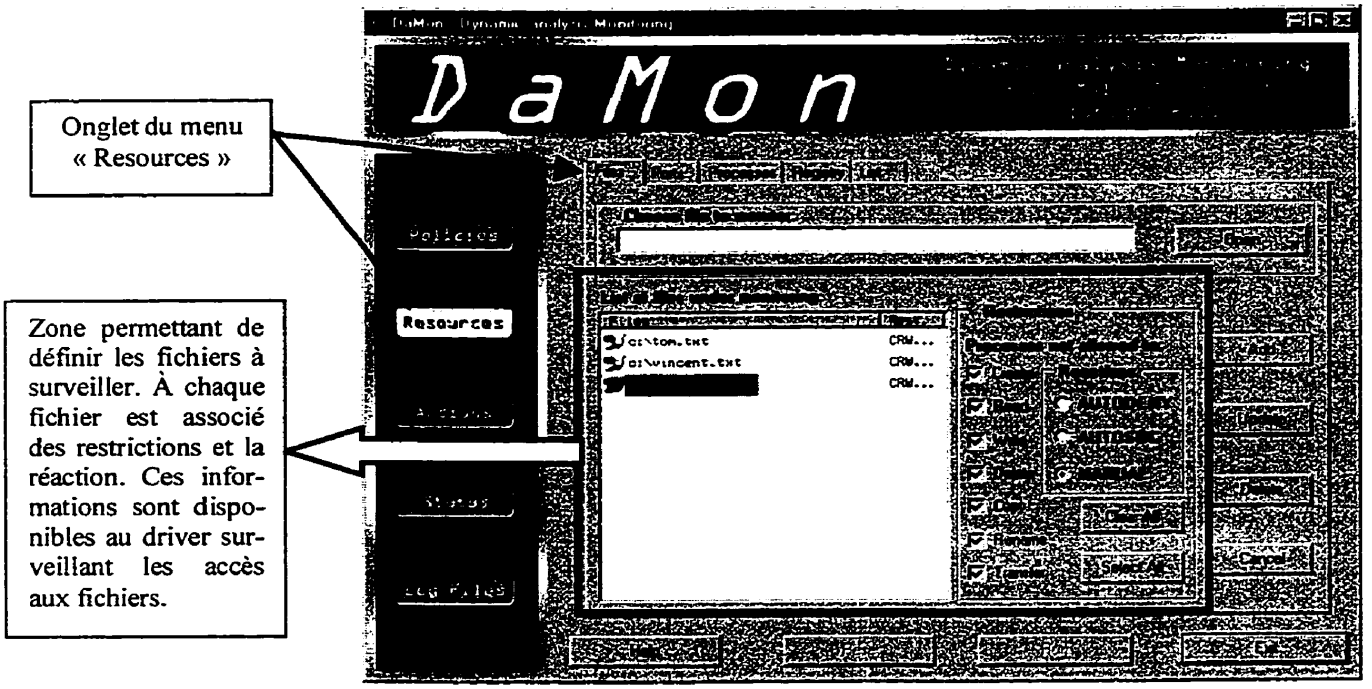


Figure 25: mise en place de la surveillance des fichiers

Si on veut modifier des restrictions sur un ou des fichiers inclus dans la liste, il suffit de cliquer sur le nom et les restrictions apparaîtront. Une fois les corrections faites, il suffit de peser sur le bouton « Update » pour signifier au filtre qu'il y a eu des changements. Ce dernier obtiendra des informations à jour. À noter que le chemin du fichier est très important. Il peut exister deux ou plusieurs fichiers avec le même nom mais ne contenant pas nécessairement le même type d'informations. Pour s'assurer que la surveil-

lance s'exerce sur le bon fichier, il est obligatoire de fournir le chemin précis du fichier. S'il manque le chemin, aucune surveillance ne sera faite.

- **Fenêtre de surveillance**

En choisissant l'onglet « List », vous avez la possibilité de voir les informations recueillies durant l'exécution du moniteur. Pour activer l'affichage, il suffit de sélectionner le boutons « Files ». La Figure 26 montre un exemple d'affichage lorsqu'on désire surveiller les accès sur le fichier « C:\Autoexec.bat ».

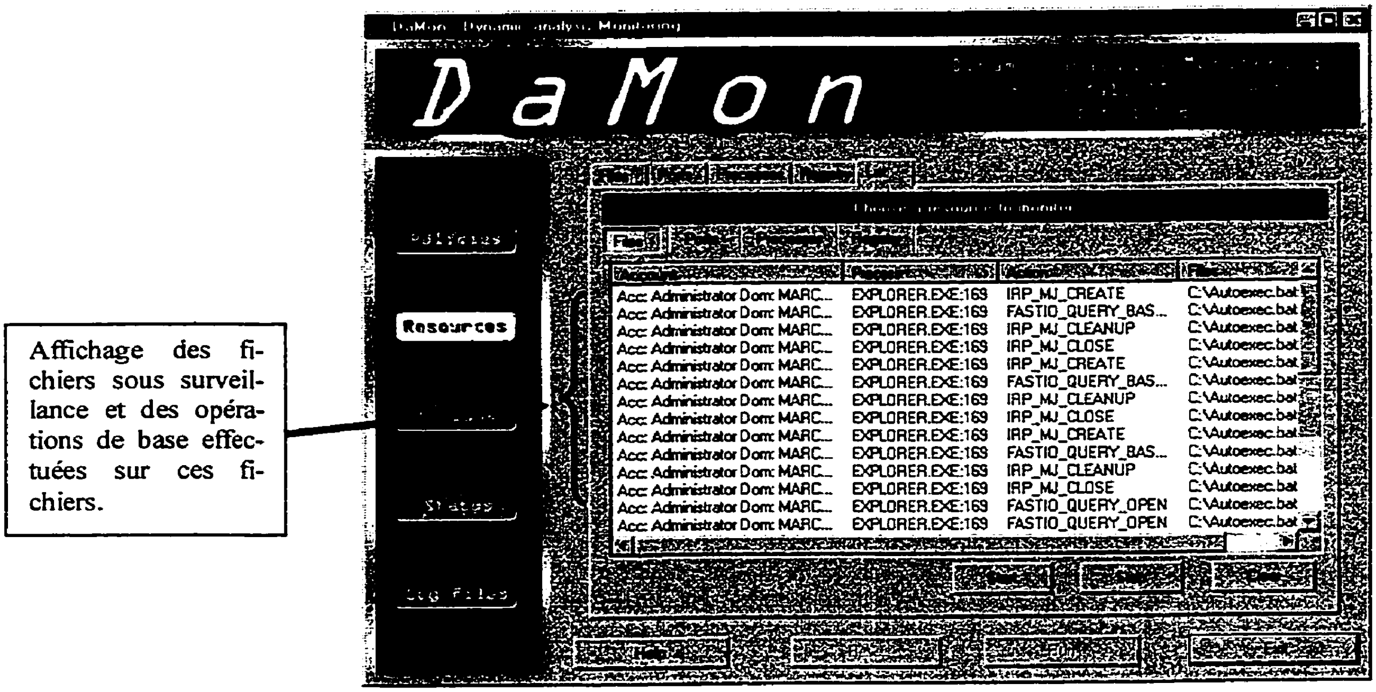


Figure 26: fenêtre de surveillance

Durant l'affichage, un fichier est créé pour enregistrer toutes les transactions se rapportant aux fichiers mis sous surveillance. En choisissant le menu « Log Files », vous sélectionnez le bouton « Files » pour faire afficher le contenu du fichier de transactions. La Figure 27 affiche la liste des transactions se rapportant à la surveillance du fichier « C:\Autoexec.bat ». L'entête du fichier contient des informations qui sont utilisées en ce moment pour la mise au point du filtre.

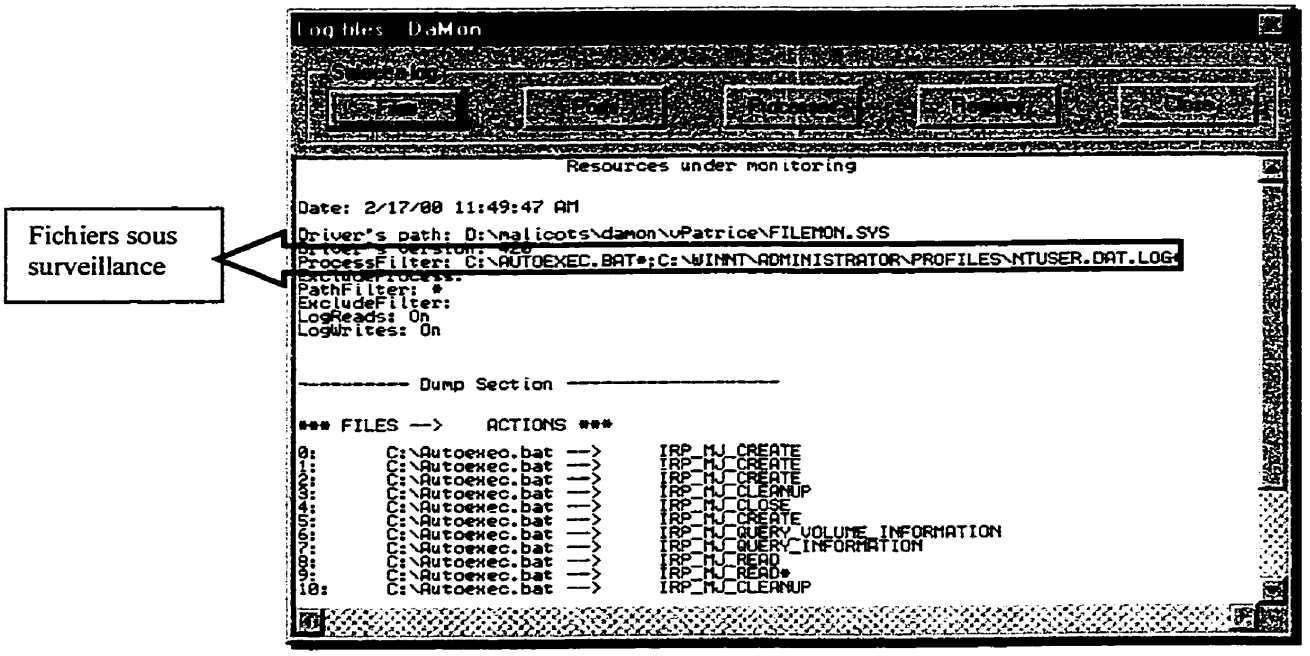


Figure 27: exemple d'archivage du monitorage des fichiers

7.2.2 Surveillance des ports de communication

Le filtre de surveillance des ports affiche tous les paquets arrivants ou sortants. Les deux sous-sections suivantes montrent les écrans nécessaires quant à la mise en service de la surveillance de ports de communication.

- **Mise en service de la surveillance des ports de communication**

En choisissant le menu « RESOURCES » et en sélectionnant l'onglet « Ports », on aura la possibilité de définir les ports à surveiller ainsi que les restrictions associées. La Figure 28 affiche l'écran correspondant.

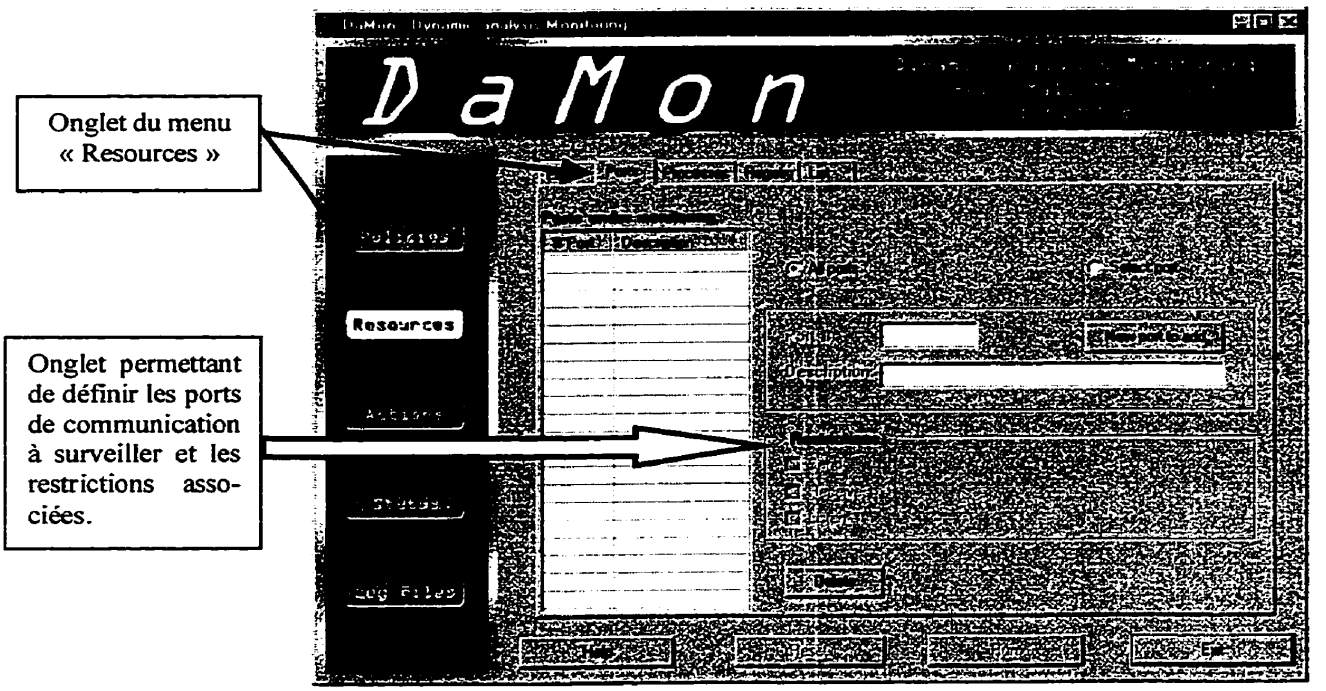


Figure 28: définition des ports de communication sous surveillance

- **Fenêtre de surveillance**

À partir des informations qu'il a été possible de recueillir dans les paquets jusqu'à présent, il est possible d'afficher le type de protocole en cours, les adresses de l'expéditeur et du destinataire, le port de communication utilisé et le contenu du paquet. À noter que le contenu du paquet n'est pas toujours en texte clair. La Figure 29 présente les informations ci-haut mentionnées. Tel que mentionné précédemment, on voit très bien les informations provenant des paquets. En utilisant, par exemple, le protocole « telnet », on voit clairement le contenu (« Data ») des paquets y compris le nom de l'usager et son mot de passe.

Il est important de noter que le rôle de ce filtre n'est pas de surveiller, pour l'instant, le contenu des informations transportées. Son rôle est de vérifier s'il y a des données qui sont envoyées sur un port de communication mis sous surveillance et si oui, d'accepter la communication si les politiques de sécurité le permettent.

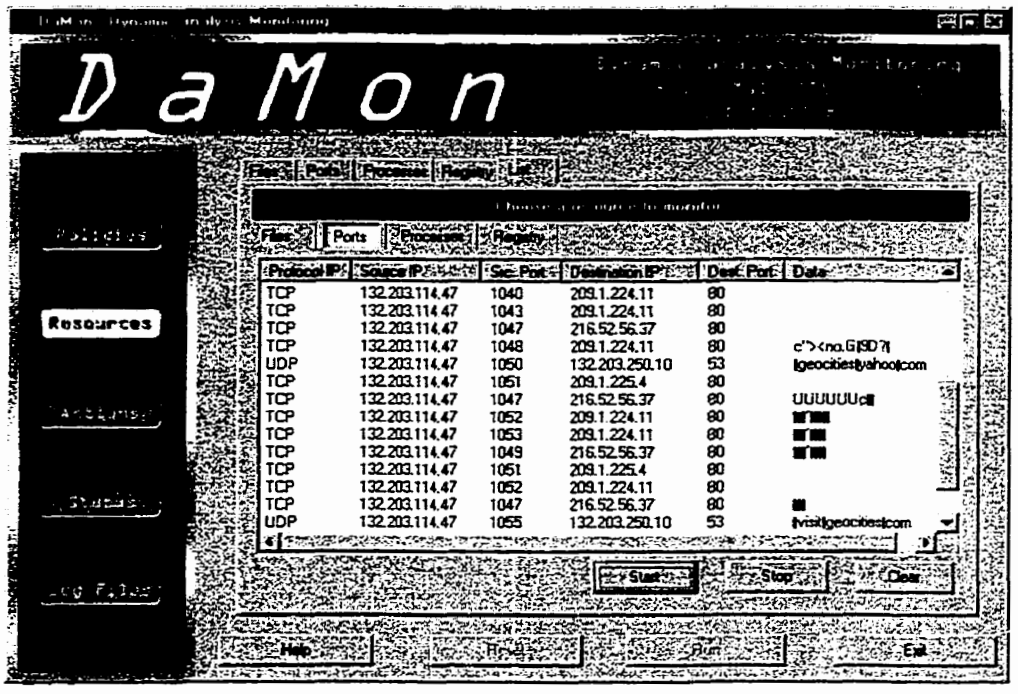


Figure 29: première partie de la fenêtre de surveillance des ports de communication

7.2.3 Surveillance de la base de registres

Le filtre de surveillance de la base de registres, RegGuard, permet de suivre les requêtes faites à celle-ci. Les deux sous-sections suivantes montrent les écrans nécessaires pour surveiller les accès à la base de registres ainsi qu'un exemple d'écran de surveillance.

- **Mise en service de la surveillance de la base de registres**

En choisissant le menu « RESOURCES » et en sélectionnant l'onglet « Registry », on aura la possibilité de définir les clés à surveiller ainsi que les restrictions associées. La Figure 30 affiche l'écran correspondant.

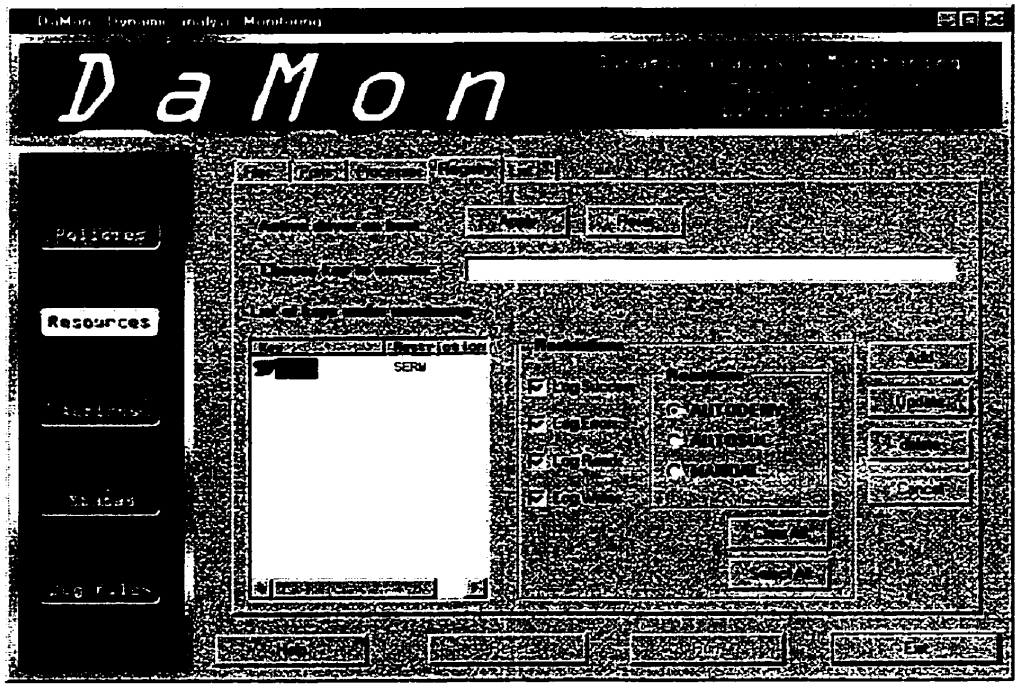


Figure 30: définition des restrictions et réaction associées à des clés de la base de registres

- **Fenêtre de surveillance**

Le rôle du filtre de surveillance de la base de registres est d'intercepter toutes les requêtes faites. La Figure 31 montre un écran de surveillance dans lequel on retrouve toutes les informations nécessaires à la gestion des accès.

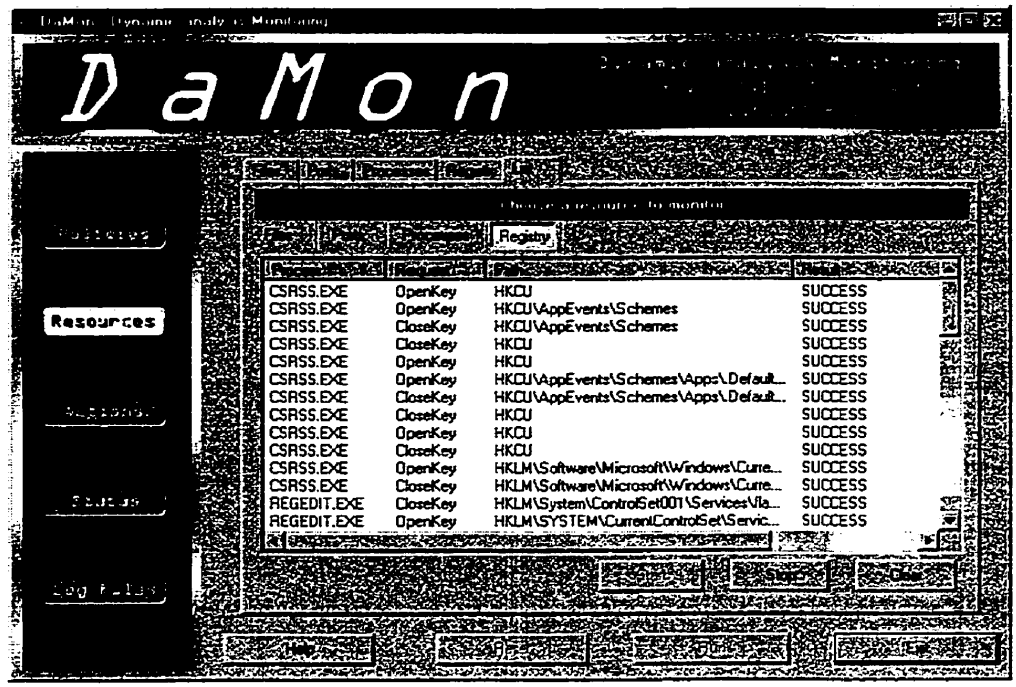


Figure 31: fenêtre de surveillance de la base de registres

7.2.4 Surveillance des processus

La surveillance des processus pose des difficultés supplémentaires surtout lorsque vient le temps de surveiller les processus systèmes. Le filtre de surveillance des processus, ProcGuard, permet de voir les processus qui ont été créés ainsi que le processus parent. La Figure 32 montre un exemple contenant les informations ci-haut mentionnées. Compte tenu des difficultés rencontrées, ce filtre est encore en phase de construction et ne fournit que très peu d'information.

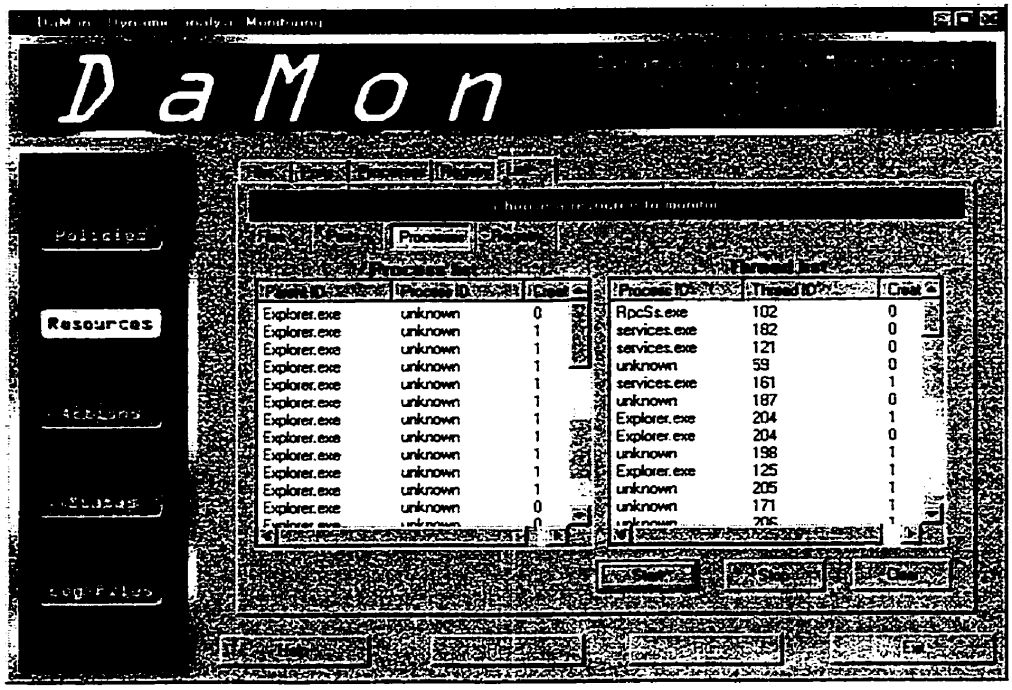


Figure 32: fenêtre de surveillance des processus

Chapitre 8

CONCLUSION

Expérience: nom dont les hommes baptisent leurs erreurs.
Oscar Wilde

La sécurité informatique est un domaine en constante évolution. Son rôle est de fournir des bases théoriques et pratiques pour protéger l'information qui est manipulée par les ordinateurs et les humains. La difficulté principale reliée à la protection de l'information dans un environnement informatique est la gestion des différents éléments intervenant dans le processus de gestion. Les composantes matérielles et logicielles ainsi que le comportement humain sont tous des éléments de ce processus de gestion.

Ce travail s'insère dans ce processus et propose un moyen de gérer les différents intervenants. Ce travail a permis de répondre de façon positive à la question posée en introduction à savoir s'il est possible de contrôler le comportement d'un environnement de façon dynamique. En concevant un moniteur d'analyse dynamique contrôlant les accès à des ressources critiques, l'objectif de ce travail a été atteint. Ce contrôle ne se fait pas sans la participation de règles ou de politiques de sécurité. En définissant des politiques de sécurité, il devient possible de combiner les différents éléments intervenant dans le processus de gestion.

Des efforts ont été faits pour trouver une façon d'implanter un moniteur qui surveillera les ressources critiques et qui s'adaptera facilement aux changements. Les motivations poussant à réaliser ce projet et une revue de l'état de l'art ont contribué fortement à poursuivre dans ce sens. En analysant scrupuleusement l'environnement Windows NT, il a été possible de bien cerner le problème, d'identifier les ressources sensibles et de définir un modèle de moniteur pouvant répondre aux exigences fixées. En insérant des filtres aux endroits stratégiques, le contrôle des ressources devient plus facile. De plus, Windows NT permet de le faire et supporte ce principe d'insertion. C'est une approche modulaire et adaptable.

Pour supporter le moniteur, il est essentiel d'avoir des politiques de sécurité. Celles-ci ont été conçues de manière à répondre le plus efficacement possible aux différentes contraintes associées à chaque environnement de travail.

En utilisant le langage XML pour définir les politiques de sécurité, cette approche permet de les définir et qui pourront être appliquées à d'autres environnements différents. Ce langage augmente la portabilité du modèle. À partir d'un fichier XML, il est possible de définir des règles de sécurité, de séparer les éléments inclus dans ces règles et d'envoyer à chaque filtre les informations pertinentes. Les filtres ont l'obligation d'exécuter les ordres reçus et de signaler chaque requête se rapportant aux informations à surveiller. Par la suite, le moniteur doit valider, à l'aide de l'usager ou de l'automate, la séquence retournée par le filtre; une séquence étant composée d'un acteur, d'une ressource et d'une action. En fonction du résultat de la validation, une réaction est transmise au gardien correspondant pour qu'il l'exécute. Ce mécanisme de surveillance et de contrôle permet de connaître l'état de l'environnement à tout moment et de réagir en fonction des informations transmises par les filtres. L'interaction entre les filtres et les séquences passées sont prises en compte par le moniteur. Ce mécanisme permet de recueillir des données, de combiner différents éléments intervenant dans le processus de gestion et de réagir en fonction de toutes ces informations. Avec le moniteur d'analyse dynamique, il devient plus facile de valider les requêtes faites sur les ressources afin de les protéger contre des actions malicieuses.

Ce projet a permis de montrer qu'il est possible de surveiller et de contrôler les accès à des ressources sensibles. Les moyens mis en place ne couvrent pas tous les aspects du projet. Certains éléments restent à concevoir pour réaliser le plein potentiel du projet. Seulement un seul filtre à été mis en place, FileGuard, permettant de démontrer que le monitorage d'un environnement est possible. La terminaison du projet passe par la conception de filtres surveillant et contrôlant les accès aux autres ressources sensibles telles la base de registres, les ports de communication et la gestion des processus. Dans tous les cas, la structure principale est implantée mais aucun mécanisme de surveillance et de contrôle est en place.

Le filtre de surveillance des ports de communication permet d'obtenir beaucoup d'information pour le contrôle des communications. L'étape principale suivante sera d'identifier l'application voulant communiquer. Cette information est nécessaire pour déterminer le processus en cours d'exécution et voir si ce dernier a les droits requis pour communiquer.

La surveillance et le contrôle de la création et la destruction des processus sont des aspects importants puisque des applications malicieuses peuvent de servir de ces actions pour annihiler un environnement. Il apparaît donc évident qu'un monitorage sera effectué sur cette ressource pour vérifier si les demandes de création et de destruction sont permises.

La base de registres est une composante propre à l'environnement Windows NT, c'est-à-dire que les autres systèmes d'exploitation de même envergure ne possèdent pas ce type de ressources. Cette ressource renferme des informations sensibles sur différents aspects de la gestion des opérations du système d'exploitation. Cette ressource est une cible très intéressante pour les « hackers » ou « crackers » qui sont à la recherche d'informations critiques telles les mots de passe. Le mécanisme de surveillance et de contrôle devra permettre au moniteur d'avoir les informations nécessaires lorsque viendra le temps de réagir.

Le développement des politiques de sécurité est en branle. Sans politique de sécurité, il devient difficile de contrôler efficacement les ressources sensibles. Partout où une décision doit être prise, celle-ci doit être faite à partir des politiques de sécurité. De plus, les politiques de sécurité permettent de diriger vers un même objectif le travail des filtres. La définition précise des politiques de sécurité est la pierre angulaire du projet. C'est l'unificateur qui permettra d'obtenir des résultats intéressants en fonction des besoins de l'utilisateur.

Chapitre 9

BIBLIOGRAPHIE

[1] Baker Art. *The Windows NT Device Driver Book – A Guide For Programmers.* Prentice Hall, 1997, 522 pages.

[2] Bergeron Jean, Debbabi Mourad, Desharnais Jules, Ktari Béchir, Salois Martin, Tawbi Nadia, Charpentier Robert & Patry Michel. *Detection of Malicious Code in COTS.* Rapport de recherche. Groupe L.S.F.M. (Université Laval), novembre 1998, 46 pages.

[3] Bessonov, Alex V.. *A Serial Port Spy for NT.* Windows developper's journal, October 1999. Vol. 10 Number 10.

[4] Chia Abeleen, Gao Yan and Strobele Jing-Ping. *Security in Operating System.* CECS 526 Advanced Operating System Term Project, June 1999. **http://www.engr.csulb.edu/~ygao**

[5] Common Criteria Project Sponsoring Organisations. *Common Criteria for Information Technology Security Evaluation, version 2.0.* May 1998. **http://www.radium.ncsc.mil**

[6] Department of Defense. *Trusted Computer System Evaluation Criteria.* DoD 5200.28-STD, CSC-STD-001-83, Library no.S225,711. Décembre 1985. **http://www.radium.ncsc.mil**

[7] Howard John D., Longstaff Thomas A. *A Common Language for Computer Security Incidents.* SANDIA REPORT SAND98-8667,October 1998.

[8] Ktari Béchir. *Détection De Code Malicieux.* Rapport d'examen de synthèse. Département d'informatique (Université Laval), janvier 1998, 22 pages.

[9] Landry Linda. *Taking Virus Protection into the 21st Century.* novembre 1998. **http://www.summitonline.com/sysmanage/papers/trend2.html**

[10] Lee Peter & Necula George. *Research on Proof-Carrying Code of Mobile-Code Security.* DARPA Workshop on Foundations for Secure Mobile Code, mars 1997, 6 pages.

[11] Lo Raymond W., Levitt Karl N. & Olsson Ronald A.. *MCF : A Malicious Code Filter.* Department of Computer Science (University of California, Davis), may 1994, 27 pages.

[12] Microsoft. *Meeting Enterprise Security Needs : Microsoft® Windows NT® and UNIX.* novembre 1998, 17 pages.
http://www.microsoft.com/security.

[13] Microsoft. *Microsoft Windows NT 4.0 Device Driver Kit – DDK Documentions.*

[14] Microsoft. *Microsoft Windows NT 4.0 Software Developperr Kit – SDK Documentions.*

[15] Microsoft. *Microsoft Windows NT Server Resource Kit.* Microsoft Press, 1996.

[16] National Computer Security Center. *FINAL EVALUATION REPORT Microsoft Inc.* Windows NT Workstation and Server Version 3.5 with U.S. Service Pack 3. Report No. CSC-FER-95/003, Library No. S243,073, 29 avril 1996. **http://www.radium.ncsc.mil**

[17] Necula George C.. *Proof-Carrying Code.* Proceedings of the 24th ACM Symposium on Principles of Programming Languages, (Paris, France), january 1997, 14 pages

[18] Potvin Williams. *NTFS « Multiple Data Streams ».* Mersoft Software, 1999.
http://www.mersoft.com/mersoftFree/information/ntfsmds.htm

[19] Qun Zhong et Nigel Edwards. *Security Control for COTS Components.* Computer, IEEE Computer Society, vol. 31, No.6, June 1998, p. 67 – 73.

[20] Russinovich Mark. Inside NTFS. *NT's native file system — past, present and future.* Windows NT Magazine, January 1998, p. 61.
http://www.winntmag.com/magazines/article.cfm

[21] Russinovich Mark. *Windows NT Architecture, Part 1.* Windows NT Magazine, March 1998, p. 59.
http://www.winntmag.com/magazines/article.cfm

[22] Russinovich Mark. *Windows NT Architecture, Part 2.* Windows NT Magazine, April 1998, p. 59.
http://www.winntmag.com/magazines/article.cfm

[23] Russinovich Mark. *Windows NT Security, Part 1.* Windows NT Magazine, May 1998, p. 59. **http://www.winntmag.com/magazines/article.cfm**

[24] Russinovich Mark. *Windows NT Security, Part 2.* Windows NT Magazine, June 1998, p. 59. **http://www.winntmag.com/magazines/article.cfm**

[25] Rutstein Charles B.. *Windows NT Security A Practical Guide to Securing Windows NT Servers & Workstations.* McGraw-Hill NCSA guides, 1997,332 pages.

[26] Salois Martin. *Dynamic Detection of Malicious Code in COTS Software.* Mémoire, Département d'informatique, Université Laval. Avril 1999.

[27] Schneider Fred B.. *Enforceable Security Policies.* Departement of Computer Science, Cornell University, New York. January 15, 1998.

[28] Sheldon Tom. *WINDOWS NT SECURITY HANDBOOK.* Osborne McGraw-Hill, 1997, 679 pages.

[29] Solomon David A.. *Inside Windows NT – Second Edition.* Microsoft Press, 1998, 528 pages.

[30] Venners Bill. *Java security : How to install the security manager and customize your security policy.* **http://www.javaworld.com/javaworld/jw-11-1997/jw-11-hood.html.**

[31] Viscarola Peter G. *Cracking Rename Operations.* OSR Open Systems Resources , 1997. **http://www.osr.com/ntinsider/1997/rename.htm**

[32] Viscarola Peter G. *How NT Handles I/O Completion.* OSR Open Systems Resources , 1997.
http://www.osr.com/ntinsider/1997/iocomp/iocomp.htm

[33] Viscarola Peter G. *IRP Cancel Operations.* OSR Open Systems Resources , 1997.
http://www.osr.com/ntinsider/1997/cancel_1.htm

[34] Viscarola Peter G. *IRP Cancel Operations (Part II).* OSR Open Systems Resources , 1998.
http://www.osr.com/ntinsider/1998/cancel2/cancel_2.htm

PARTIE IV – LES ANNEXES

ANNEXE I – DÉFINITION DES COTES TCSEC

Le TCSEC (« Trusted Computer System Evaluation Criteria ») correspond à « l'Orange Book » définit par le DoD (« Department of Defense ») des Etats-Unis. Ce standard est reconnu pour évaluer le niveau de sécurité des logiciels utilisé par l'armée américaine. Ce standard permet de coter une application selon le niveau de sécurité atteint.

Les critères d'évaluation sont regroupés par division. Chaque division signifie une amélioration dans la confiance qu'une personne peut porter à un système face à sa capacité à protéger les données sensibles. Les divisions vont de « D » à « A », où « A » possède la plus grande cote de confiance. Une division peut être séparée en classe pour définir des niveaux de confiance différents, à l'intérieur même d'une division. La définition des divisions suppose qu'une division supérieure possède les caractéristiques des divisions inférieures. Les divisions supérieures sont des améliorations aux divisions inférieures.

- **Division (D)** : protection minimale (« minimal protection ») Cette division ne contient qu'une classe (D1). Cette division est réservée aux systèmes qui ont été évalués mais qui n'ont pas réussi à atteindre un niveau supérieur.

- **Division (C)** : les classes dans cette division signifient qu'il y a une protection discrétionnaire et des moyens de comptabilisation des actions des usagers à l'aide de capacités d'audit.

  Classe (C1) : sécurité discrétionnaire (« discretionary security protection ») Cette classe doit satisfaire les exigences de sécurité discrétionnaire en différenciant les usagers des données. L'usager doit être capable de protéger certaines données et empêcher certains usagers de lire ou de détruire ses données.

  Classe (C2) : accès contrôlé (« controlled access protection ») Cette classe possède un niveau plus élevé de raffinement dans le contrôle des accès. Elle permet de comptabiliser les actions des usagers à travers des procédures de démarrage, l'enregistrement des transactions se rapportant à la sécurité et l'isolation des ressources.

- **Division (B)** : le contrôle obligatoire des accès est l'exigence principale de cette division. Des étiquettes de sensibilité pour protéger l'intégrité des données doivent être rattachées à chaque structure de données dans le système. Le concept de moniteur de références doit être implanté et démontré.

  Classe (B1) : sécurité étiquetée (« labeled security protection ») Cette classe nécessite toutes les caractéristiques de la classe C2. En plus, le contrôle obliga-

toire des accès doit être présent. Une politique de sécurité doit être présente. Toute faiblesse doit être identifiée et enlevée.

Classe (B2) : protection structurée (« structured protection ») Dans cette classe, la politique de sécurité doit être bien définie et documentée. Cette politique exige un contrôle discrétionnaire et obligatoire des accès sur toutes les ressources du système. De plus, la notion de « covert channel » est présente. Une gestion rigoureuse de configuration est imposée. Les mécanismes d'authentification sont renforcés. Le système est relativement résistant aux pénétrations.

Classe (B3): domaines de sécurité (« security domains ») Le moniteur de références doit être en place pour contrôler les accès, entre les usagers et les ressources, et être inviolable. Un administrateur doit être présent, des mécanismes d'audit doivent signaler des problèmes en ce qui concerne la sécurité et des procédures de recouvrement doivent être en place. Le système est très résistant aux pénétrations.

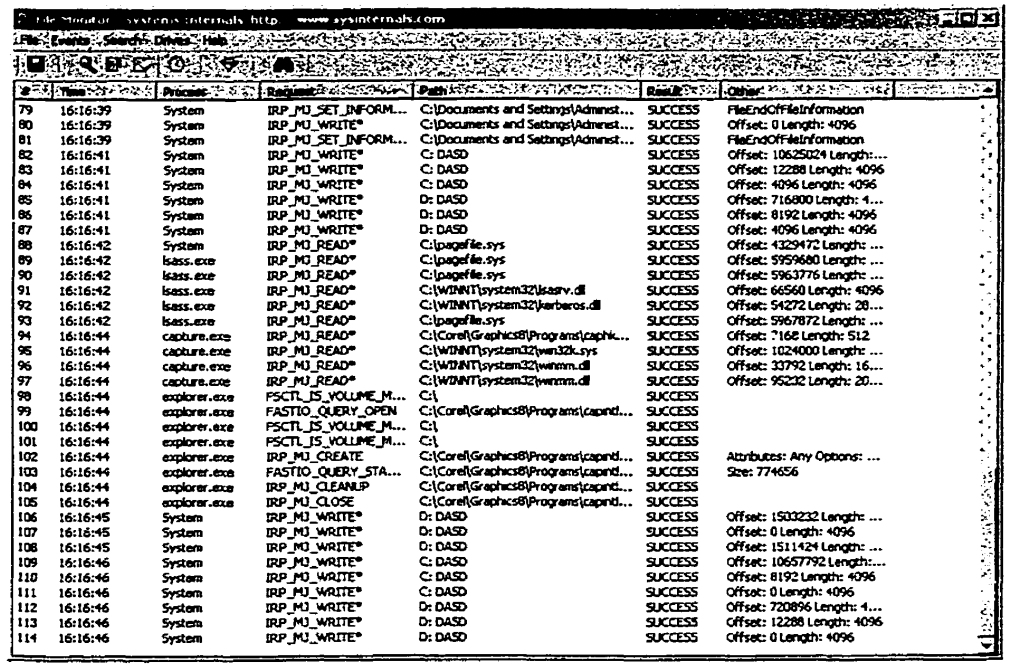
- **Division (A)**: design vérifié (« verified design ») Cette division est caractérisée par l'utilisation formelle de méthodes de vérification de la sécurité pour assurer que le système utilise la protection discrétionnaire et obligatoire afin de protéger les données sensibles du système. Une documentation élaborée doit être fournie pour démontrer que le produit rencontre bien les exigences de sécurité dans tous les aspects de design, de développement et d'implantation. Cette division ne possède que la classe (A1).

ANNEXE II - OUTILS DE SURVEILLANCE

La compagnie SYSINTERNALS a conçu différents logiciels fonctionnant dans l'environnement Windows NT. Ces logiciels sont fournis gratuitement via Internet. Dans certains cas, le code source est présent. Pour la conception des différents drivers travaillant conjointement avec le moniteur, ceux-ci ont été grandement inspirés des travaux réalisés par SYSINTERNALS.

▸ FILEMON

Ce logiciel permet de surveiller l'activité se rapportant à l'accès aux fichiers. Il utilise un filtre pour surveiller toutes les requêtes faites aux fichiers.

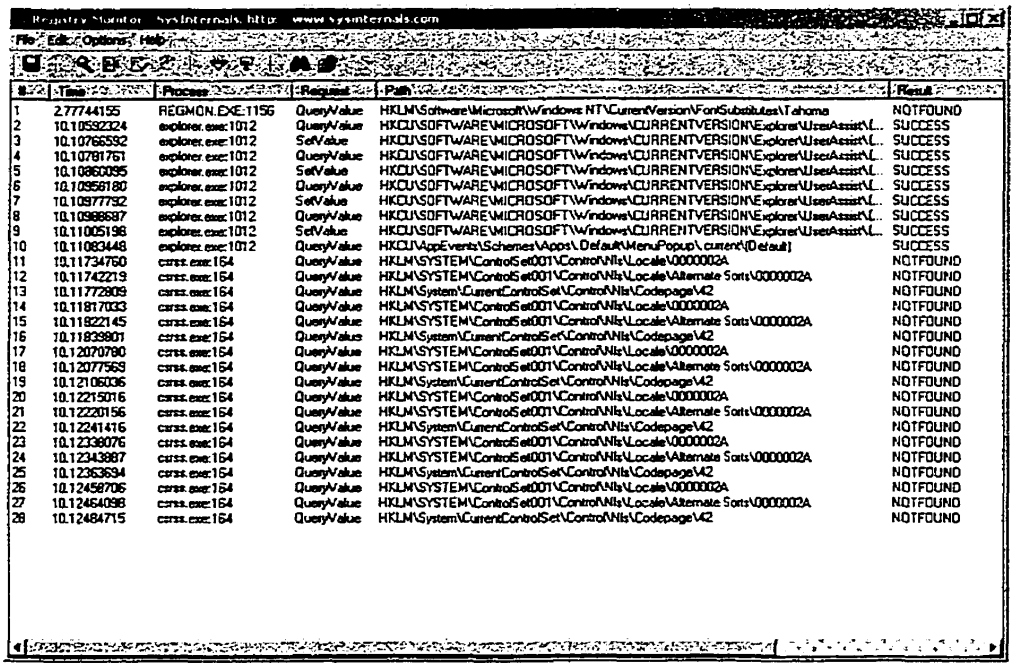


#	Time	Process	Request	Path	Result	Other
79	16:16:39	System	IRP_MJ_SET_INFORM...	C:\Documents and Settings\Administ...	SUCCESS	FileEndOfFileInformation
80	16:16:39	System	IRP_MJ_WRITE*	C:\Documents and Settings\Administ...	SUCCESS	Offset: 0 Length: 4096
81	16:16:39	System	IRP_MJ_SET_INFORM...	C:\Documents and Settings\Administ...	SUCCESS	FileEndOfFileInformation
82	16:16:41	System	IRP_MJ_WRITE*	C: DASD	SUCCESS	Offset: 10625024 Length:...
83	16:16:41	System	IRP_MJ_WRITE*	C: DASD	SUCCESS	Offset: 12288 Length: 4096
84	16:16:41	System	IRP_MJ_WRITE*	C: DASD	SUCCESS	Offset: 4096 Length: 4096
85	16:16:41	System	IRP_MJ_WRITE*	D: DASD	SUCCESS	Offset: 716800 Length: 4...
86	16:16:41	System	IRP_MJ_WRITE*	D: DASD	SUCCESS	Offset: 8192 Length: 4096
87	16:16:41	System	IRP_MJ_WRITE*	D: DASD	SUCCESS	Offset: 4096 Length: 4096
88	16:16:42	System	IRP_MJ_READ*	C:\pagefile.sys	SUCCESS	Offset: 4329472 Length: ...
89	16:16:42	lsass.exe	IRP_MJ_READ*	C:\pagefile.sys	SUCCESS	Offset: 5959680 Length: ...
90	16:16:42	lsass.exe	IRP_MJ_READ*	C:\pagefile.sys	SUCCESS	Offset: 5963776 Length: ...
91	16:16:42	lsass.exe	IRP_MJ_READ*	C:\WINNT\system32\lsasrv.dll	SUCCESS	Offset: 66560 Length: 4096
92	16:16:42	lsass.exe	IRP_MJ_READ*	C:\WINNT\system32\kerberos.dll	SUCCESS	Offset: 54272 Length: 28...
93	16:16:42	lsass.exe	IRP_MJ_READ*	C:\pagefile.sys	SUCCESS	Offset: 5967872 Length: ...
94	16:16:44	capture.exe	IRP_MJ_READ*	C:\Corel\Graphics8\Programs\caphk...	SUCCESS	Offset: 7168 Length: 512
95	16:16:44	capture.exe	IRP_MJ_READ*	C:\WINNT\system32\wsn32k.sys	SUCCESS	Offset: 1024000 Length: ...
96	16:16:44	capture.exe	IRP_MJ_READ*	C:\WINNT\system32\wnmm.dll	SUCCESS	Offset: 33792 Length: 16...
97	16:16:44	capture.exe	IRP_MJ_READ*	C:\WINNT\system32\wnmm.dll	SUCCESS	Offset: 95232 Length: 20...
98	16:16:44	explorer.exe	FSCTL_IS_VOLUME_M...	C:\	SUCCESS	
99	16:16:44	explorer.exe	FASTIO_QUERY_OPEN	C:\Corel\Graphics8\Programs\capntl...	SUCCESS	
100	16:16:44	explorer.exe	FSCTL_IS_VOLUME_M...	C:\	SUCCESS	
101	16:16:44	explorer.exe	FSCTL_IS_VOLUME_M...	C:\	SUCCESS	
102	16:16:44	explorer.exe	IRP_MJ_CREATE	C:\Corel\Graphics8\Programs\capntl...	SUCCESS	Attributes: Any Options: ...
103	16:16:44	explorer.exe	FASTIO_QUERY_STA...	C:\Corel\Graphics8\Programs\capntl...	SUCCESS	Size: 774656
104	16:16:44	explorer.exe	IRP_MJ_CLEANUP	C:\Corel\Graphics8\Programs\capntl...	SUCCESS	
105	16:16:44	explorer.exe	IRP_MJ_CLOSE	C:\Corel\Graphics8\Programs\capntl...	SUCCESS	
106	16:16:45	System	IRP_MJ_WRITE*	D: DASD	SUCCESS	Offset: 1503232 Length: ...
107	16:16:45	System	IRP_MJ_WRITE*	D: DASD	SUCCESS	Offset: 0 Length: 4096
108	16:16:45	System	IRP_MJ_WRITE*	D: DASD	SUCCESS	Offset: 1511424 Length: ...
109	16:16:46	System	IRP_MJ_WRITE*	C: DASD	SUCCESS	Offset: 10657792 Length:...
110	16:16:46	System	IRP_MJ_WRITE*	C: DASD	SUCCESS	Offset: 8192 Length: 4096
111	16:16:46	System	IRP_MJ_WRITE*	C: DASD	SUCCESS	Offset: 0 Length: 4096
112	16:16:46	System	IRP_MJ_WRITE*	D: DASD	SUCCESS	Offset: 720896 Length: 4...
113	16:16:46	System	IRP_MJ_WRITE*	D: DASD	SUCCESS	Offset: 12288 Length: 4096
114	16:16:46	System	IRP_MJ_WRITE*	D: DASD	SUCCESS	Offset: 0 Length: 4096

Figure 33: Filemon

▸ REGMON

REGMON est un logiciel qui surveille tous les accès à la base de registres. Pour fonctionner, il requiert l'utilisation d'un driver qui intercepte tous les appels de fonction se rapportant à la gestion de la base de registres.



#	Time	Process	Request	Path	Result
1	2.777441155	REGMON.EXE:1156	QueryValue	HKLM\Software\Microsoft\Windows NT\CurrentVersion\FontSubstitutes\Tahoma	NOTFOUND
2	10.10592324	explorer.exe:1012	QueryValue	HKCU\SOFTWARE\MICROSOFT\Windows\CURRENTVERSION\Explorer\UserAssist\{...	SUCCESS
3	10.10766592	explorer.exe:1012	SetValue	HKCU\SOFTWARE\MICROSOFT\Windows\CURRENTVERSION\Explorer\UserAssist\{...	SUCCESS
4	10.10781761	explorer.exe:1012	QueryValue	HKCU\SOFTWARE\MICROSOFT\Windows\CURRENTVERSION\Explorer\UserAssist\{...	SUCCESS
5	10.10860095	explorer.exe:1012	SetValue	HKCU\SOFTWARE\MICROSOFT\Windows\CURRENTVERSION\Explorer\UserAssist\{...	SUCCESS
6	10.10956180	explorer.exe:1012	QueryValue	HKCU\SOFTWARE\MICROSOFT\Windows\CURRENTVERSION\Explorer\UserAssist\{...	SUCCESS
7	10.10977732	explorer.exe:1012	SetValue	HKCU\SOFTWARE\MICROSOFT\Windows\CURRENTVERSION\Explorer\UserAssist\{...	SUCCESS
8	10.10988687	explorer.exe:1012	QueryValue	HKCU\SOFTWARE\MICROSOFT\Windows\CURRENTVERSION\Explorer\UserAssist\{...	SUCCESS
9	10.11005198	explorer.exe:1012	SetValue	HKCU\SOFTWARE\MICROSOFT\Windows\CURRENTVERSION\Explorer\UserAssist\{...	SUCCESS
10	10.11083448	explorer.exe:1012	QueryValue	HKCU\AppEvents\Schemes\Apps\.Default\MenuPopup\.current\(Default)	SUCCESS
11	10.11734750	csrss.exe:164	QueryValue	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\00000002A	NOTFOUND
12	10.11742219	csrss.exe:164	QueryValue	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\Alternate Sorts\00000002A	NOTFOUND
13	10.11772809	csrss.exe:164	QueryValue	HKLM\System\CurrentControlSet\Control\Nls\Codepage\42	NOTFOUND
14	10.11817033	csrss.exe:164	QueryValue	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\00000002A	NOTFOUND
15	10.11822145	csrss.exe:164	QueryValue	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\Alternate Sorts\00000002A	NOTFOUND
16	10.11839801	csrss.exe:164	QueryValue	HKLM\System\CurrentControlSet\Control\Nls\Codepage\42	NOTFOUND
17	10.12070780	csrss.exe:164	QueryValue	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\00000002A	NOTFOUND
18	10.12077563	csrss.exe:164	QueryValue	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\Alternate Sorts\00000002A	NOTFOUND
19	10.12106036	csrss.exe:164	QueryValue	HKLM\System\CurrentControlSet\Control\Nls\Codepage\42	NOTFOUND
20	10.12215016	csrss.exe:164	QueryValue	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\00000002A	NOTFOUND
21	10.12220156	csrss.exe:164	QueryValue	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\Alternate Sorts\00000002A	NOTFOUND
22	10.12241416	csrss.exe:164	QueryValue	HKLM\System\CurrentControlSet\Control\Nls\Codepage\42	NOTFOUND
23	10.12336076	csrss.exe:164	QueryValue	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\00000002A	NOTFOUND
24	10.12343887	csrss.exe:164	QueryValue	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\Alternate Sorts\00000002A	NOTFOUND
25	10.12363634	csrss.exe:164	QueryValue	HKLM\System\CurrentControlSet\Control\Nls\Codepage\42	NOTFOUND
26	10.12458706	csrss.exe:164	QueryValue	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\00000002A	NOTFOUND
27	10.12464098	csrss.exe:164	QueryValue	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\Alternate Sorts\00000002A	NOTFOUND
28	10.12484715	csrss.exe:164	QueryValue	HKLM\System\CurrentControlSet\Control\Nls\Codepage\42	NOTFOUND

Figure 34: Regmon

- **TCPView**

TCPView est un outil qui affiche des informations sur les protocoles de communication TCP/UDP. Malheureusement, ce produit ne fournit pas le nom des applications qui utilisent les ports de communication.

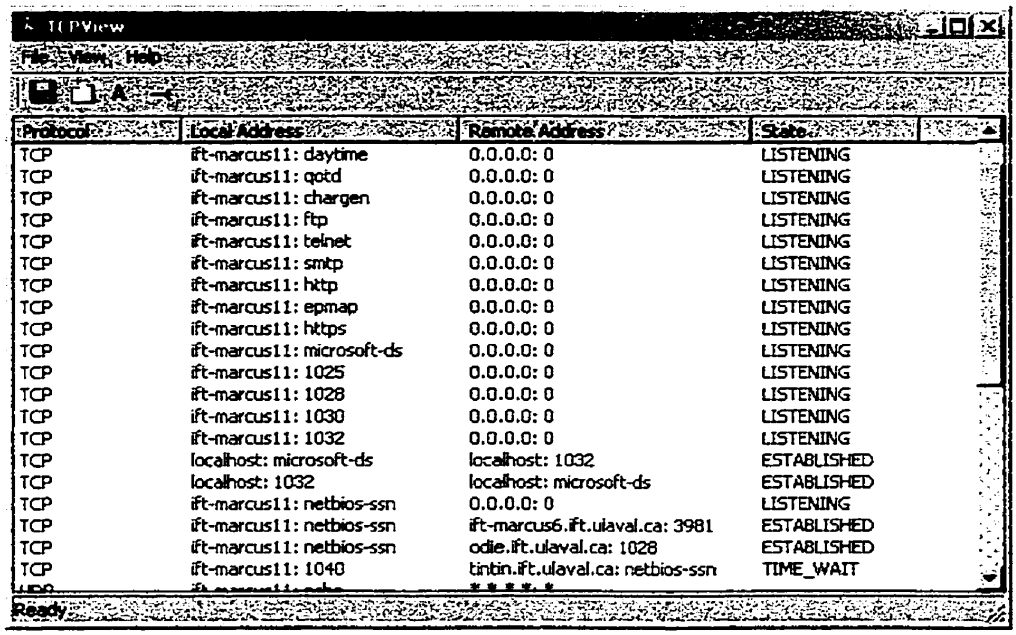


Figure 35: TCPView

- **Pmon**

Pmon est un petit logiciel qui affiche les informations se rapportant à la gestion des processus. De plus, il fournit les actions commises par les processus et le nom du processus parent créant un processus fils.

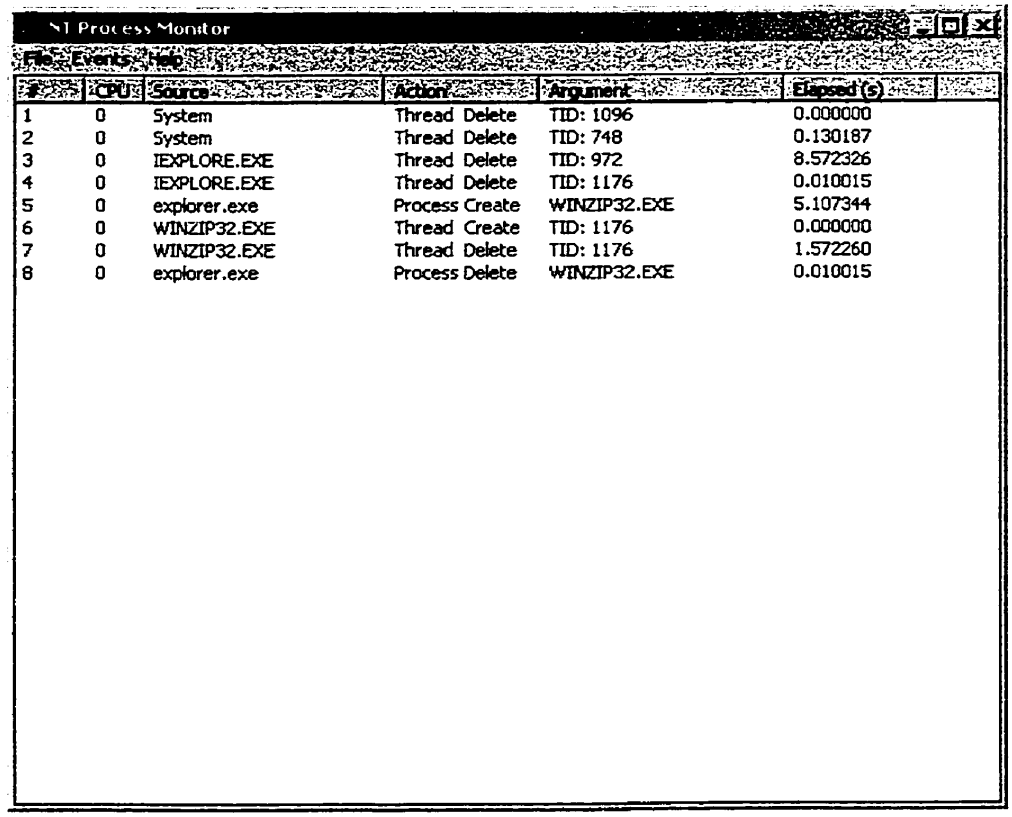


Figure 36: PMon

ANNEXE III – DÉFINITION DES TERMES POUR L'UTILISATION D'UN LANGAGE COMMUN EN SÉCURITÉ INFORMATIQUE

À partir des résultats obtenus dans le « Common Language Project » de Howard et Longstaff [7], les termes utilisés dans la Tableau 1, page 12, y sont définis. Voici la définition des termes inclus dans le tableau :

- **Actions** – opération faite par un usager ou un processus pour atteindre un résultat.
  - **Sonder ("probe")** – accéder à une cible pour obtenir ses caractéristiques.
  - **Balayer ("scan")** – accéder à un ensemble de cibles pour identifier quelles sont les cibles qui ont une caractéristique particulière.
  - **Surcharger ("flood")** – accéder en répétition à une cible pour la surcharger.
  - **Authentifier ("authenticate")** – présenter l'identité de quelqu'un à un processus et, si nécessaire, vérifier cette identité afin d'accéder à une cible.
  - **Éviter ("bypass")** – éviter un processus en utilisant une méthode alternative pour accéder à une cible.
  - **Déguiser ("spoof")** – changer d'apparence en assumant l'apparence d'une entité différente dans un réseau de communications.
  - **Lire ("read")** – obtenir les données continues sur un unité de stockage ou autres médiums de données.
  - **Copier ("copy")** – reproduire une cible en laissant la cible originale inchangée.
  - **Voler ("steal")** – prendre possession d'une cible sans laisser de copie à l'endroit original.
  - **Modifier ("modify")** – changer le contenu ou les caractéristiques de la cible.
  - **Effacer ("delete")** – effacer une cible ou la rendre irrécupérable.

- **Cibles** – entité logique d'un ordinateur ou d'un réseau (comptes d'usager, processus ou données) ou une entité physique (composante, ordinateur, réseau ou interréseau).
  - **Compte d'usager ("account")** – ensemble d'informations sur un usager contenant son nom, son mot de passe et ses restrictions.

- **Processus ("process")** – un programme en exécution incluant la partie exécutable, les données et la pile du programme et différents pointeurs permettant d'exécuter le programme.
- **Données ("data")** – représentation des faits, concepts ou instructions de manière à les rendre convenables pour la communication, l'interprétation ou le traitement par les humains ou autres moyens automatiques.
- **Composantes ("component")** – une des parties constituant un ordinateur ou un réseau.
- **Ordinateur ("computer")** – une machine qui est composée d'une ou plusieurs composantes, incluant les unités de traitement et périphériques, qui est contrôlée par des programmes internes et qui ne nécessite pas l'intervention de l'humain durant son exécution.
- **Réseau ("network")**– un groupe interconnecté ou inter relié d'ordinateurs.
- **Inter-réseau ("internetwork")** – un réseau de réseaux.

- **Ressources** – regroupement d'une ou plusieurs cibles ayant des caractéristiques communes.
  - **Fichiers** – toute cible ayant accès à une unité de stockage.
  - **Communications** – toute cible nécessitant l'utilisation des moyens de communication.
  - **Base de registres** – toute cible ayant accès à la base de registres pour lire ou écrire des informations.
  - **Processus (mémoire)** – toute cible voulant utiliser le gestionnaire des processus.

ANNEXE IV – TAXONOMIE DU CODE MALICIEUX

Chia [4] présente une taxonomie du code malicieux. Le tableau ci-dessous répond aux trois questions se rapportant aux codes malicieux et caractérisant celui-ci. Les questions sont :

- Comment est-il entré dans le système (« Genesis »)?
- Quand est-il entré dans le système (« Time of introduction »)?
- À quel endroit dans le système s'est-il manifesté (« Location »)?

Le tableau est divisé en trois parties; chaque partie répondant à une des questions. La première partie permet de savoir comment est entré le code malicieux.

- ***Genèse*** *(« Genesis »)* : la Genèse explique comment une brèche de sécurité peut se retrouver dans un programme ou un système. Elle peut y être insérer intentionnellement ou par inadvertance. Dans le cas d'une insertion intentionnelle, on retrouve des sous-catégories : malicieuse et non-malicieuse. Selon l'auteur, la sous-catégorie non-malicieuse risque de provoquer des brèches de sécurité que très rarement. Elle inclut les « Covert channel[38] » qui sont définis comme étant un chemin utilisé pour transférer des informations et qui n'était pas prévu par le concepteur du système. Les brèches de ce type se produisent lors du partage des ressources. Cette définition explique pourquoi l'auteur considère une sous-catégorie non-malicieuse. En réalité, du moment qu'une application commet une action pouvant affecter la sécurité des informations ou le comportement du système, cette action est considérée malicieuse.

La deuxième partie du tableau permet de connaître à quel moment du processus de développement du logiciel le code malicieux a été introduit. Ce tableau ne répond pas complètement à la deuxième question mais permet de poser des points de repère aux vérificateurs de logiciels. Dans le contexte de la détection de code malicieux, cette partie du tableau ne donne pas d'information pertinente mais à l'avantage de cibler les sources possibles d'insertion de code malicieux durant le développement d'un produit.

- **Moment de l'introduction** (« time of introduction ») : le moment de l'introduction est basé sur le cycle de développement de la conception du logiciel. Si, durant la phase des tests, aucune brèche n'a été découverte et que nous savons qu'il y en a, il est possible de croire que le code malicieux a été introduit durant la phase de programmation.

[38] Voir Salois [26] pour une description complète des codes malicieux.

La troisième partie du tableau permet de savoir à quel endroit un code malicieux peut se trouver. Cette partie du tableau répond à la troisième question à savoir à quel endroit le code malicieux s'est-il manifesté?

Genesis	Intentional	Malicious	Trojan Horse	Non-Replicating
				Replicating (virus)
			Trapdoor	
			Logic/Time Bomb	
		Non-Malicious	Covert Channel	Storage
				Timing
			Other	
	Inadvertent	Validation error (Incomplete/Inconsistent)		
		Domain Error		
		Serialization/aliasing (Including TOCTTOU Errors)		
		Identification/Authentication Inadequate		
		Boundary Condition violation		
		Other ecploitable Logic Error		
Time of Introduction	During Development	Requirements/Specification/Design		
		Source Code		
		Object Code		
	During Maintenance			
	During Operation			
Location	Software	Operating System	System Initialization	
			Memory Management	
			Process management/Scheduling	
			Device management (Including I/O, networking)	
			File Management	
			Identification/Authentication	
			Other/Unknown	
		Support	Priviliged Utilities	
			Unprivileged Utilities	
		Application		
	Hardware			

- **Localisation** (« location ») : l'auteur divise les localisations en partie : le logiciel et le matériel. Dans le cas du matériel, l'auteur parle de problèmes reliés à l'exécution (résultats incorrects après une instruction) ou de transfert de données. Pour les logiciels, l'auteur inclut le système d'exploitation et tous les autres logiciels installés dans un système en les catégorisant (système d'exploitation, support, applications). Les programmes de la catégorie support inclut les compilateurs, éditeurs, SGBD et autres programmes du même genre.

Il est très important de noter que ce tableau n'est qu'un point de départ à la détection de code malicieux. L'idée principale, derrière ce tableau, est de permettre à un utilisateur de connaître l'existence d'une taxonomie des codes malicieux pour les identifier.

ANNEXE V –STRUCTURE DE DONNÉES

Pour aider à la compréhension de la structure du IRP, cette annexe fournit les autres structures de données couramment utilisées pour obtenir les informations pertinentes. Ces structures sont tirées du fichier NTDDK.H contenu dans le Microsoft DDK [13] (« Device Driver Kit ») pour Microsoft Windows NT.

▸ Structure IRP

Le IRP (« I/O Request Packet ») est un élément essentiel dans les requêtes E/S. Cet élément est construit par le Gestionnaire E/S pour chaque opération de base nécessaire à la gestion des accès aux fichiers et aux ports de communication. Un IRP est une structure très complexe contenant beaucoup d'information. Pour aider à la compréhension des échanges entre les applications et le Gestionnaire E/S, il est avantageux de connaître cette structure pour avoir une idée des informations véhiculées.

```
typedef struct _IRP {
    CSHORT Type;
    USHORT Size;

    //
    // Define the common fields used to control the IRP.
    //

    //
    // Define a pointer to the Memory Descriptor List (MDL) for this I/O
    // request.  This field is only used if the I/O is "direct I/O".
    //

    PMDL MdlAddress;

    //
    // Flags word - used to remember various flags.
    //

    ULONG Flags;

    //
    // The following union is used for one of three purposes:
    //
    //    1. This IRP is an associated IRP.  The field is a pointer to a master
    //       IRP.
    //
    //    2. This is the master IRP.  The field is the count of the number of
    //       IRPs which must complete (associated IRPs) before the master can
    //       complete.
    //
    //    3. This operation is being buffered and the field is the address of
    //       the system space buffer.
    //

    union {
        struct _IRP *MasterIrp;
        LONG IrpCount;
```

```
} AssociatedIrp;

//
// Thread list entry - allows queueing the IRP to the thread pending I/O
// request packet list.
//

LIST_ENTRY ThreadListEntry;

//
// I/O status - final status of operation.
//

IO_STATUS_BLOCK IoStatus;

//
// Requestor mode - mode of the original requestor of this operation.
//

KPROCESSOR_MODE RequestorMode;

//
// Pending returned - TRUE if pending was initially returned as the
// status for this packet.
//

BOOLEAN PendingReturned;

//
// Stack state information.
//

CHAR StackCount;
CHAR CurrentLocation;

//
// Cancel - packet has been canceled.
//

BOOLEAN Cancel;

//
// Cancel Irql - Irql at which the cancel spinlock was acquired.
//

KIRQL CancelIrql;

//
// ApcEnvironment - Used to save the APC environment at the time that the
// packet was initialized.
//

CCHAR ApcEnvironment;

//
// Allocation control flags.
//

UCHAR AllocationFlags;

//
// User parameters.
//

PIO_STATUS_BLOCK UserIosb;
PKEVENT UserEvent;
```

```
        struct {
            PIO_APC_ROUTINE UserApcRoutine;
            PVOID UserApcContext;
        } AsynchronousParameters;
        LARGE_INTEGER AllocationSize;
    } Overlay;

    //
    // CancelRoutine - Used to contain the address of a cancel routine supplied
    // by a device driver when the IRP is in a cancelable state.
    //

    PDRIVER_CANCEL CancelRoutine;

    //
    // Note that the UserBuffer parameter is outside of the stack so that I/O
    // completion can copy data back into the user's address space without
    // having to know exactly which service was being invoked.  The length
    // of the copy is stored in the second half of the I/O status block. If
    // the UserBuffer field is NULL, then no copy is performed.
    //

    PVOID UserBuffer;

    //
    // Kernel structures
    //
    // The following section contains kernel structures which the IRP needs
    // in order to place various work information in kernel controller system
    // queues.  Because the size and alignment cannot be controlled, they are
    // placed here at the end so they just hang off and do not affect the
    // alignment of other fields in the IRP.
    //

    union {

        struct {

            union {

                //
                // DeviceQueueEntry - The device queue entry field is used to
                // queue the IRP to the device driver device queue.
                //

                KDEVICE_QUEUE_ENTRY DeviceQueueEntry;

                struct {

                    //
                    // The following are available to the driver to use in
                    // whatever manner is desired, while the driver owns the
                    // packet.
                    //

                    PVOID DriverContext[4];

                } ;

            } ;

            //
            // Thread - pointer to caller's Thread Control Block.
            //

            PETHREAD Thread;
```

```
            //
            // Auxiliary buffer - pointer to any auxiliary buffer that is
            // required to pass information to a driver that is not contained
            // in a normal buffer.
            //

            PCHAR AuxiliaryBuffer;

            //
            // List entry - used to queue the packet to completion queue, among
            // others.
            //

            LIST_ENTRY ListEntry;

            //
            // Current stack location - contains a pointer to the current
            // IO_STACK_LOCATION structure in the IRP stack.  This field
            // should never be directly accessed by drivers.  They should
            // use the standard functions.
            //

            struct _IO_STACK_LOCATION *CurrentStackLocation;

            //
            // Original file object - pointer to the original file object
            // that was used to open the file.  This field is owned by the
            // I/O system and should not be used by any other drivers.
            //

            PFILE_OBJECT OriginalFileObject;

        } Overlay;

        //
        // APC - This APC control block is used for the special kernel APC as
        // well as for the caller's APC, if one was specified in the original
        // argument list.  If so, then the APC is reused for the normal APC for
        // whatever mode the caller was in and the "special" routine that is
        // invoked before the APC gets control simply deallocates the IRP.
        //

        KAPC Apc;

        //
        // CompletionKey - This is the key that is used to distinguish
        // individual I/O operations initiated on a single file handle.
        //

        ULONG CompletionKey;

    } Tail;

} IRP, *PIRP;
```

Extrait 17: structure de données - IRP

Structure DEVICE_OBJECT

Cette structure est utilisée pour définir chaque « device » attaché au système d'exploitation. À l'intérieur de cette structure, il y a un champ, `DeviceExtension`, contenant un pointeur qui permet aux programmeurs de définir une structure particulière pouvant stocker des informations précises sur le traitement en cours. Cette structure particulière est définie selon les besoins du programmeur et est couramment utilisée.

```
typedef struct _DEVICE_OBJECT {
    CSHORT Type;
    USHORT Size;
    LONG ReferenceCount;
    struct _DRIVER_OBJECT *DriverObject;
    struct _DEVICE_OBJECT *NextDevice;
    struct _DEVICE_OBJECT *AttachedDevice;
    struct _IRP *CurrentIrp;
    PIO_TIMER Timer;
    ULONG Flags;                                // See above:  DO_...
    ULONG Characteristics;                      // See ntioapi:  FILE_...
    PVPB Vpb;
    PVOID DeviceExtension;
    DEVICE_TYPE DeviceType;
    CCHAR StackSize;
    union {
        LIST_ENTRY ListEntry;
        WAIT_CONTEXT_BLOCK Wcb;
    } Queue;
    ULONG AlignmentRequirement;
    KDEVICE_QUEUE DeviceQueue;
    KDPC Dpc;

    //
    //  The following field is for exclusive use by the filesystem to keep
    //  track of the number of Fsp threads currently using the device
    //

    ULONG ActiveThreadCount;
    PSECURITY_DESCRIPTOR SecurityDescriptor;
    KEVENT DeviceLock;

    USHORT SectorSize;
    USHORT Spare1;

    struct _DEVOBJ_EXTENSION  *DeviceObjectExtension;
    PVOID  Reserved;
} DEVICE_OBJECT;
```

Extrait 18: structure de données - DEVICE_OBJECT

▸ Structure DRIVER_OBJECT

Chaque pilote installé dans le noyau de Windows NT possède une structure l'identifiant comme un objet. Cette structure contient toutes les informations que le Gestionnaire E/S utilise pour le traitement des requêtes.

```
typedef struct _DRIVER_OBJECT {
    CSHORT Type;
    CSHORT Size;

    //
    // The following links all of the devices created by a single driver
    // together on a list, and the Flags word provides an extensible flag
    // location for driver objects.
    //

    PDEVICE_OBJECT DeviceObject;
    ULONG Flags;

    //
    // The following section describes where the driver is loaded.  The count
    // field is used to count the number of times the driver has had its
    // registered reinitialization routine invoked.
    //

    PVOID DriverStart;
    ULONG DriverSize;
    PVOID DriverSection;
    PDRIVER_EXTENSION DriverExtension;

    //
    // The driver name field is used by the error log thread
    // determine the name of the driver that an I/O request is/was bound.
    //

    UNICODE_STRING DriverName;

    //
    // The following section is for registry support.  Thise is a pointer
    // to the path to the hardware information in the registry
    //

    PUNICODE_STRING HardwareDatabase;

    //
    // The following section contains the optional pointer to an array of
    // alternate entry points to a driver for "fast I/O" support.  Fast I/O
    // is performed by invoking the driver routine directly with separate
    // parameters, rather than using the standard IRP call mechanism.  Note
    // that these functions may only be used for synchronous I/O, and when
    // the file is cached.
    //

    PFAST_IO_DISPATCH FastIoDispatch;

    //
    // The following section describes the entry points to this particular
    // driver.  Note that the major function dispatch table must be the last
    // field in the object so that it remains extensible.
    //

    PDRIVER_INITIALIZE DriverInit;
    PDRIVER_STARTIO DriverStartIo;
```

```
    PDRIVER_DISPATCH MajorFunction[IRP_MJ_MAXIMUM_FUNCTION + 1];
} DRIVER_OBJECT;
```

Extrait 19: structure de données - DRIVER_OBJECT

▸ Structure FILE_OBJECT

La structure FILE_OBJECT contient les informations se rapportant aux fichiers qui sont ouverts. Chaque fois qu'une requête est entreprise, le Gestionnaire E/S doit demander au Gestionnaire des Objets de créer un objet « FILE ». C'est à partir de ces objets que le travail s'exécute.

```
typedef struct _FILE_OBJECT {
    CSHORT Type;
    CSHORT Size;
    PDEVICE_OBJECT DeviceObject;
    PVPB Vpb;
    PVOID FsContext;
    PVOID FsContext2;
    PSECTION_OBJECT_POINTERS SectionObjectPointer;
    PVOID PrivateCacheMap;
    NTSTATUS FinalStatus;
    struct _FILE_OBJECT *RelatedFileObject;
    BOOLEAN LockOperation;
    BOOLEAN DeletePending;
    BOOLEAN ReadAccess;
    BOOLEAN WriteAccess;
    BOOLEAN DeleteAccess;
    BOOLEAN SharedRead;
    BOOLEAN SharedWrite;
    BOOLEAN SharedDelete;
    ULONG Flags;
    UNICODE_STRING FileName;
    LARGE_INTEGER CurrentByteOffset;
    ULONG Waiters;
    ULONG Busy;
    PVOID LastLock;
    KEVENT Lock;
    KEVENT Event;
    PIO_COMPLETION_CONTEXT CompletionContext;
} FILE_OBJECT;
```

Extrait 20: structure de données - FILE_OBJECT

Structure IO_STATUS_BLOCK

Cette structure est utilisée pour retourner le statut de l'opération à l'application qui a initié la requête.

```
typedef struct _IO_STATUS_BLOCK {
    NTSTATUS Status;
    ULONG Information;
} IO_STATUS_BLOCK, *PIO_STATUS_BLOCK;
```

Extrait 21: structure de données - IO_STATUS_BLOCK

Structure _IO_STACK_LOCATION

Dans la structure d'un IRP, le Gestionnaire E/S réserve un espace correspondant à chacun des pilotes que le IRP devra traverser. En utilisant la fonction `IoGetCurrentIrpStackLocation()`, un pilote peut obtenir les informations se rapportant à lui. Les informations sont contenues dans la structure IO_STACK_LOCATION.

```
typedef struct _IO_STACK_LOCATION {
    UCHAR MajorFunction;
    UCHAR MinorFunction;
    UCHAR Flags;
    UCHAR Control;

    //
    // The following user parameters are based on the service that is being
    // invoked.  Drivers and file systems can determine which set to use based
    // on the above major and minor function codes.
    //

    union {

        //
        // System service parameters for:  NtCreateFile
        //

        struct {
            PIO_SECURITY_CONTEXT SecurityContext;
            ULONG Options;
            USHORT FileAttributes;
            USHORT ShareAccess;
            ULONG EaLength;
        } Create;


        //
        // System service parameters for:  NtReadFile
        //

        struct {
            ULONG Length;
            ULONG Key;
            LARGE_INTEGER ByteOffset;
        } Read;
```

```
//
// System service parameters for:  NtWriteFile
//

struct {
    ULONG Length;
    ULONG Key;
    LARGE_INTEGER ByteOffset;
} Write;


//
// System service parameters for:  NtQueryInformationFile
//

struct {
    ULONG Length;
    FILE_INFORMATION_CLASS FileInformationClass;
} QueryFile;

//
// System service parameters for:  NtSetInformationFile
//

struct {
    ULONG Length;
    FILE_INFORMATION_CLASS FileInformationClass;
    PFILE_OBJECT FileObject;
    union {
        struct {
            BOOLEAN ReplaceIfExists;
            BOOLEAN AdvanceOnly;
        };
        ULONG ClusterCount;
        HANDLE DeleteHandle;
    };
} SetFile;


//
// System service parameters for:  NtQueryVolumeInformationFile
//

struct {
    ULONG Length;
    FS_INFORMATION_CLASS FsInformationClass;
} QueryVolume;


//
// System service parameters for:  NtFlushBuffersFile
//
// No extra user-supplied parameters.
//


//
// System service parameters for:  NtDeviceIoControlFile
//
// Note that the user's output buffer is stored in the UserBuffer field
// and the user's input buffer is stored in the SystemBuffer field.
//

struct {
    ULONG OutputBufferLength;
    ULONG InputBufferLength;
```

```
    PVOID Type3InputBuffer;
} DeviceIoControl;

//
// System service parameters for:  NtQuerySecurityObject
//

struct {
    SECURITY_INFORMATION SecurityInformation;
    ULONG Length;
} QuerySecurity;

//
// System service parameters for:  NtSetSecurityObject
//

struct {
    SECURITY_INFORMATION SecurityInformation;
    PSECURITY_DESCRIPTOR SecurityDescriptor;
} SetSecurity;

//
// Non-system service parameters.
//
// Parameters for MountVolume
//

struct {
    PVPB Vpb;
    PDEVICE_OBJECT DeviceObject;
} MountVolume;

//
// Parameters for VerifyVolume
//

struct {
    PVPB Vpb;
    PDEVICE_OBJECT DeviceObject;
} VerifyVolume;

//
// Parameters for Scsi with internal device contorl.
//

struct {
    struct _SCSI_REQUEST_BLOCK *Srb;
} Scsi;

//
// Parameters for StartDevice
//

struct {
    PCM_RESOURCE_LIST AllocatedResources;
} StartDevice;

//
// Parameters for Cleanup
//
// No extra parameters supplied
//

//
// Others - driver-specific
//
```

```
        struct {
            PVOID Argument1;
            PVOID Argument2;
            PVOID Argument3;
            PVOID Argument4;
        } Others;

    } Parameters;

    //
    // Save a pointer to this device driver's device object for this request
    // so it can be passed to the completion routine if needed.
    //

    PDEVICE_OBJECT DeviceObject;

    //
    // The following location contains a pointer to the file object for this
    //

    PFILE_OBJECT FileObject;

    //
    // The following routine is invoked depending on the flags in the above
    // flags field.
    //

    PIO_COMPLETION_ROUTINE CompletionRoutine;

    //
    // The following is used to store the address of the context parameter
    // that should be passed to the CompletionRoutine.
    //

    PVOID Context;

} IO_STACK_LOCATION, *PIO_STACK_LOCATION;
```

Extrait 22: structure de données - IO_STACK_LOCATION

ANNEXE VI – CODES DE CONTRÔLE ENVOYÉS AUX DIFFÉRENTS PILOTES

Durant l'exécution du moniteur, celui-ci doit communiquer avec le filtre de surveillance des fichiers. Le Tableau 9 présente les codes de contrôle que le moniteur doit envoyer au filtre pour échanger des informations.

Code de contrôle	Description
IOCTL_FILEMON_SETDRIVES	Fournir la liste des disques à surveiller.
IOCTL_FILEMON_ZEROSTATS	Initialiser ou réinitialiser le tampon d'échange.
IOCTL_FILEMON_GETSTATS	Obtenir les informations incluses dans le tampon d'échange.
IOCTL_FILEMON_UNLOADQUERY	Décharger le filtre de l'environnement.
IOCTL_FILEMON_STOPFILTER	Arrêter la surveillance.
IOCTL_FILEMON_STARTFILTER	Commencer la surveillance.
IOCTL_FILEMON_SETFILTER	Signifier aux filtres de récupérer les informations de surveillance dans la base de registres.
IOCTL_FILEMON_TIMETYPE	Obtenir l'heure des transactions.
IOCTL_FILEMON_VERSION	Obtenir le numéro de la version du filtre.
IOCTL_FILEMON_HOOKSPECIAL	Inclure les « named-pipes » et les « mailslots » dans les objets à surveiller.
IOCTL_FILEMON_UNHOOKSPECIAL	Retirer les « named-pipes » et les « mailslots » dans les objets à surveiller.

Tableau 9: liste des codes de contrôle émis par DaMon vers le filtre des fichiers

Le Tableau 10 fournit la liste des fonctions qui sont utilisées pour communiquer avec le filtre de surveillance des communications.

Nom de la fonction	Description
PacketGetAdapterNames	Retourne le nom de la carte réseau.
PacketOpenAdapter	Permet d'obtenir un pointeur sur une structure ADAPTER
PacketCloseAdapter	Libère le pointeur sur la structure ADAPTER.
PacketAllocatePacket	Permet d'obtenir un pointeur sur une structure PACKET.
PacketInitPacket	Initialise la structure PACKET.
PacketFreePacket	Libère le pointeur sur la structure PACKET.
PacketReceivePacket	Permet de recevoir des paquets.
PacketWaitPacket	Vérifie si la réception des paquets est terminée.
PacketSendPacket	Permet d'envoyer des paquets.
PacketResetAdapter	Mise à zéro de la structure ADAPTER.
PacketSetHwFilter	Permet de fixer le filtre pour les paquets entrants.
PacketRequest	Interroge ou règle la structure ADAPTER.
PacketSetBuff	Permet de redimensionner le tampon.
PacketSetBpf	Permet de définir un filtre au niveau de ADAPTER.
PacketGetStats	Permet de connaître la valeur de deux variables internes du pilote.
PacketGetNetType	Retourne le type de carte réseau.

Tableau 10: liste des fonctions utilisées pour communiquer avec le filtre de surveillance des communications

Le Tableau 11 affiche les codes de contrôle qui sont utilisés entre DaMon et le pilote RegGuard pour échanger des informations.

Code de contrôle	Description
REGMON_hook	Accrocher le pilote après la table des services.
REGMON_unhook	Décrocher le pilote de la table des services.
REGMON_zerostats	Initialiser le tampon stockant la liste des requêtes.
REGMON_getstats	Obtenir le contenu du tampon.
REGMON_setfilter	Initialiser la liste des clés à surveiller.
REGMON_version	Obtenir le numéro de version du pilote

Tableau 11: liste des codes de contrôle émis par DaMon vers le filtre de la base de registres