REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Khider – BISKRA

Faculté des Sciences Exactes, et des Sciences de la Nature et de la Vie
Département d'Informatique

Mémoire en vue de l'obtention du diplôme de Magister en informatique

***Option*: Data Mining et Multimédia**

Intitulé :

Détection des virus informatiques par les techniques de datamining.

Présenté par :

BRAHIMI Mohamed

Devant le jury composé de :

PR. DJEDI NOUREDDINE	**PROFESSEUR**	**PRESIDENT**	**UNIVERSITE DE BISKRA**
PR. MOUSSAOUI ABDELOUAHAB	**PROFESSEUR**	**RAPPORTEUR**	**UNIVERSITE DE SETIF**
DR. BABAHENINI MOHAMED CHAOUKI	**MAITRE DE CONFERENCES A**	**EXAMINATEUR**	**UNIVERSITE DE BISKRA**
DR. FOUDIL CHERIF	**MAITRE DE CONFERENCES A**	**EXAMINATEUR**	**UNIVERSITE DE BISKRA**

Année Universitaire 2013/2014

إهداء

انقضاء مرحلة من مسيرتي يجعلني أقف متأملا.أقف و عالم من الأفكار يضطرب بداخلي حتى أعجز عن مجارات نفسي.أذكر حديث أمي بحرقة عن علم لم يتح لها وعن أمل تضعه في شخصي غير تاركة مجالا للتقاعس.أذكر أب يكد ليل نهار ويحترق مع السنون لعلنا نبلغ المرام.

أي أبوايا ، لم تحتضنكما أي مدرسة و لكن كنتما لي أكثر من مدرسة.

أي أبوايا، يخالط دمعي حبري و أنا أذكر تضحياتكما.

أي أبوايا، لا أملك إلى أن أهديكما ثمرة جهدكما و يا ليتني قدرت على أكثر.

وبالمثل أهدي عملي هاذا إلى كل أخواتي **أسيا و سندس ودعاء** متمنيا لهن الإبداع و التميز. و إلى أخي **بلال** راجيا أن يوفقه الله إلى التخرج بتفوق. كما إهديه إلى أخ لم تلده أمي إسمه **بن موسى سمير** متمنيا له خير الدنيا والأخرة.

أهدي جهدي أيضا إلى **كل أصدقائي** و **زملائي في دفعة الماجستير**.

و ختاما، غير قمين بي أن أنسى **أساتذتي في كل الأطوار**، فما أنا إلى نتيجة إجتماع جهودهم.

REMERCIEMENT

Ma reconnaissance va à tous ceux qui m'ont aidé à conduire ce travail à son terme : tout particulièrement, j'exprime toute ma gratitude à mon encadreur le professeur **Moussaoui Abdelouahab** pour l'effort fourni, les conseils prodigués, sa patience et sa persévérance dans le suivi.

J'adresse également mes remerciements, à tous mes enseignants, qui m'ont donné les bases de la science, je remercie très sincèrement, les membres de jury d'avoir bien voulu accepter de faire partie de la commission d'examinateur.

Je remercie aussi **Dr Kraim Khairedine**, **Tazir Lotfi**, **Youcef Reda, Laraba Sohaib, Khelifa Alla** et **Guessab Hanane** qui ont contribué à améliorer la qualité du travail en lisant mon mémoire.

Merci à ma famille et mes amis pour leurs soutiens et leurs Encouragements et à tous ceux qui ont, de près ou de loin, attribué et aidé à l'aboutisse ment de ce travail.

ملخص

منذ ظهور أول نظام كشف عن الفيروسات يعتمد على مبدأ التنقيب في البيانات و الذي إقترحه شولتز في عام 2001، أثبتت العديد من الدراسات فعالية هاته التقنيات في مكافحة فيروسات الكمبيوتر. إنها أنظمة تعمل على إستغلال البيانات الناتجة عن هجمات سابقة قصد تحقيق طرق كشف أكثر ذكاءا مستندة في معظمها على التعلم تحت الإشراف. وهذا يحد من تكيفها في البيئات الديناميكية (كبيئة فيروسات الكمبيوتر) لأنها تصبح عاجزة عن التكيف بعد إنقضاء مرحلة التدريب. وعلاوة على ذلك، فتدريب هاته الأنظمة يتطلب عددا كبيرا من البرامج المعلمة كمجموعة التدريب.

في هذه الرسالة، نقترح نظاما للكشف عن الفيروسات يعتمد أساسا على عملية التنقيب في البيانات بطريقة تطورية، حيث نكون قادرين على التقليل من عدد الأمثلة خلال التدريب مخفضين بذلك كلفة وضع العلامات دون الإخلال بدقة الكشف. فعلا لقد تم إجراء تحسينات كبيرة في عملية الكشف عن فيروسات الكمبيوتر من خلال إستعمال طرق تعلم جديدة كالتعلم التدريجي و التعلم النشط.

الكلمات المفتاحية:

الكشف عن فيروسات الكمبيوتر؛ التنقيب في البيانات؛ التعلم تحت الإشراف؛ التعلم النشط؛ التعلم التدريجي؛ شعاع الدعم الآلي.

RESUME

Depuis l'apparition du premier système de détection de virus basé sur le datamining proposé par Schultz en 2001, plusieurs études ont montré l'efficacité des techniques de datamining dans la lutte contre les virus informatiques. Ces systèmes, basés sur le datamining, exploitent les données disponibles sur les attaques précédentes pour aboutir à une méthode de détection plus intelligente. La majorité de ces systèmes est basée sur l'apprentissage supervisé. Ceci limite leurs adaptations dans les environnements dynamiques (environnement de virus informatiques) car ils ne sont pas évolutifs, après la phase d'entrainement. De plus, l'entrainement d'un modèle nécessite un nombre élevé de programmes étiquetés comme base d'apprentissage.

Dans ce mémoire, nous proposons un système de détection de virus basé sur un processus évolutif de datamining où on sera capable d'optimiser le nombre d'exemples pour l'entrainement tout en diminuant le coût de l'étiquetage. Une amélioration considérable est faite dans le processus de détection de virus informatiques grâce à notre système basé sur une nouvelle architecture d'apprentissage actif et incrémental.

Mots clés :

Détection de virus informatiques; Datamining ; Apprentissage supervisé ; Apprentissage actif ; Apprentissage incrémental ; Machine à vecteurs de support (SVM).

ABSTRACT

Since the emergence of the first virus detection system based on the data mining model proposed by Schultz in 2001, several studies have demonstrated the efficiency of the data mining tools in the fight against computer viruses. These systems, based on data mining, exploit the available data on previous attacks in order to achieve a more intelligent detection method. The majority of these systems is based on supervised learning. This limits their adaptations in dynamic environments (computer viruses environment) because they are not flexible after the training phase. Moreover, the training of a model requires a large number of labeled programs as training set.

In this paper, we propose a virus detection system based on an incremental process of data mining which will be able to optimize the number of training examples while reducing the cost of labeling. A considerable improvement is made in the process of detecting computer viruses through our system based on a new active and incremental learning architecture.

Keywords:

Computer virus detection; Data mining; Supervised learning; Active learning; Incremental learning; Support vector machine (SVM).

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

Chapter 1 : INTRODUCTION GENERALE

1. Contexte de l'étude

Depuis l'apparition du premier virus informatique en 1986, un très grand nombre de nouveaux virus informatiques apparaissent chaque année. Avec le développement des réseaux et plus particulièrement de l'internet, la vitesse de propagation des virus informatiques devient de plus en plus rapide et les dégâts occasionnés par ces derniers sont de plus en plus importants. Par conséquent le virus informatique devient l'un des problèmes les plus importants auxquels l'informatique s'est penchée ces dernières années. Ceci s'est senti avec les millions d'attaques que subsistent les systèmes chaque jour. De ce fait, L'utilisation de logiciels appropriés (antivirus) est devenue indispensable pour lutter contre ces virus et plus généralement contre tout programme malveillant.

Principalement, il y a deux techniques utilisées pour la détection des virus informatiques : la détection par signature et l'analyse comportementale. Le problème avec la première technique est qu'elle se limite aux virus connus et analysés, elle ne permet de détecter ni les nouveaux virus ni les variantes des virus connus. Tandis que la deuxième technique ne résiste pas aux techniques de leurres utilisées par certains virus dans le but d'échapper à l'analyse comportementale.

Les méthodes de détection de virus informatiques se basent essentiellement sur la détection par signature. Cette méthode, malgré son efficacité dans la détection des virus connus, nécessite une mise à jour périodique pour lutter contre les nouveaux virus. Ceci a permis aux programmeurs de virus d'avoir toujours un pas d'avance sur les experts de sécurité.

Parallèlement, le datamining s'est imposé comme une autre alternative puissante pour l'extraction d'informations pertinentes, qui est de plus en plus utilisé quand une grande masse d'informations est disponible. Cependant, l'abondance d'informations sur les attaques virales incite les chercheurs à s'orienter vers l'usage de techniques issues de l'apprentissage automatique pour lutter contre les virus. De même, la majorité des virus qui apparaissent sont, à la base, des variantes de virus déjà identifiés afin d'induire en erreur les techniques basées sur la signature. De ce fait, les techniques de datamining peuvent extraire des modèles très utiles pour l'antivirus en exploitant la base de données des attaques précédentes.

2. Problématique

La majorité des systèmes développés pour la détection de virus, basés sur l'apprentissage supervisé (VDS-DM : *Viruse Detection System based on Dataminng*) ***[Schultz, et al., 2001; Wang, et al., 2003; Kolter, et al., 2004; Shabtai, et al., 2009; Sami, et al., 2010; Moonsamy, et al., 2012]***, ont opté pour une architecture générique linéaire (**Figure 1-1**).

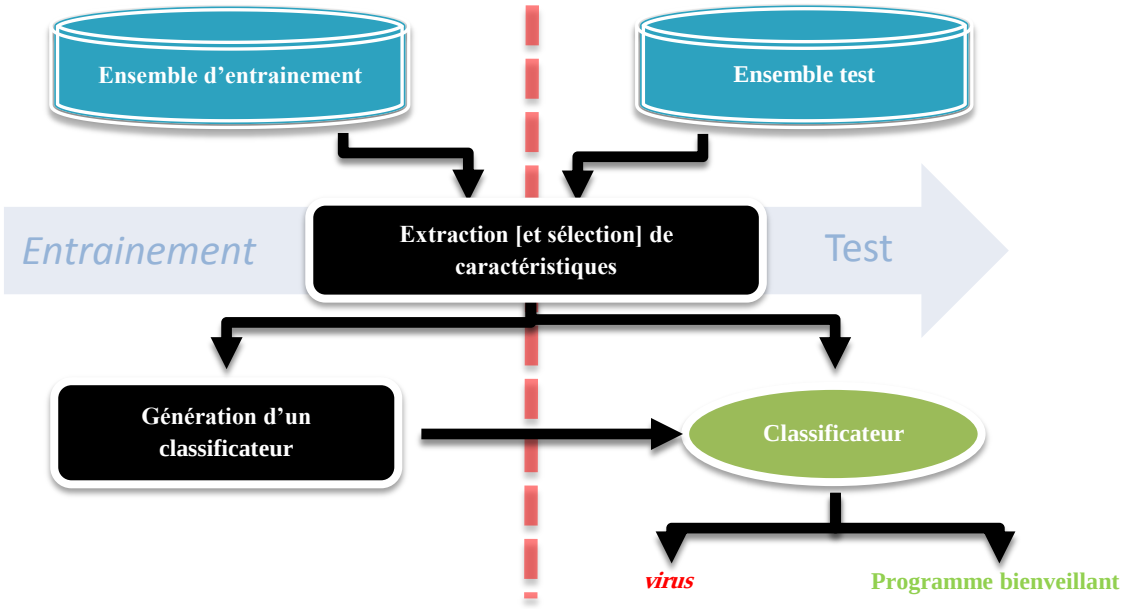


Figure 1-1 Construction d'un VDS-DM

D'après cette architecture, après une seule étape d'entrainement et de test, le VDS-DM obtenu sera de nature fixe limitant en conséquence son efficacité dans un environnement dynamique (environnement de virus informatiques). Des limitations majeures pour cette approche sont résumées par les points suivants :

- Non-évolutivité du VDS-DM : l'élaboration d'un VDS-DM, selon cette approche, le rend incapable d'acquérir de nouveaux exemples étiquetés (programmes analysés) et par conséquent l'interaction VDS-DM avec un expert de sécurité sera impossible. De ce fait, le VDS-DM est dépourvu de la capacité de s'évoluer contrairement au virus qui évoluent, quant à eux, rapidement ***[Moskovitch, et al., 2009]***.
- La nécessité d'avoir un nombre élevé de programmes étiquetés : l'approche du VDS-DM exige, et sans optimisation en nombre, de programmes étiquetés utilisés comme

exemples d'entrainement. Sachant que l'étiquetage des programmes est une tâche laborieuse, car elle exige l'intervention d'expert, la disponibilité de ressources d'analyse et un temps d'analyse considérable ***[Santos, et al., 2011; Moskovitch, et al., 2009]***

3. Objectifs

Les limitations citées au-dessus nous ont incitées d'améliorer l'approche VDS-DM. Notre but est de rendre évolutif le VDS-DM, d'une part, et d'optimiser le nombre d'exemples d'étiquetages, d'autre part. Ce qui nous permettra de traiter la deuxième limite liée au coup d'étiquetages. Autrement dit, l'approche proposée à l'amélioration de l'interaction Expert de sécurité / VDS-DM. En effet, l'évolutivité offre à l'expert la possibilité de mettre à jour son VDS-DM avec de nouveaux exemples acquis. Nous pouvons résumer les objectifs de notre travail par les points suivants :

- Utiliser d'autres types d'apprentissage tels que l'apprentissage incrémental et/ou actif afin de répondre à la problématique posée, au début du chapitre.
- Formaliser la notion d'incrémentation du VDS-DM.
- Utiliser le classificateur Machine à Vecteur de Support (SVM) comme support pour notre système évolutif VDS-DM.

4. Organisation du mémoire

L'organisation de ce mémoire reflète la démarche que nous avons adoptée lors de la réalisation de ce travail. Ce mémoire est divisé en quatre chapitres :

Le premier chapitre est consacré à la présentation des infections informatiques, en se focalisant sur les infections les plus importantes et les plus dangereuses, à savoir : les virus et les vers informatiques.

Dans le second chapitre, nous présentons les techniques de datamining et leurs applications à la détection de virus informatique.

Dans le troisième chapitre, nous présentons notre approche adoptée pour la construction d'un système de détection de virus informatiques basé sur les techniques du datamining.

Le chapitre quatre, quant à lui, est consacré à la description de la réalisation de notre système ainsi qu'aux différents résultats des tests qui nous ont permis d'évaluer notre système de détection de virus informatiques. Une conclusion générale et des perspectives clôturent ce mémoire.

Chapter 2 : INFECTIONS INFORMATIQUES

1. Introduction

Depuis l'apparition du premier virus informatique lors des années 80, le nombre de nouveau virus n'a jamais cessé de croitre d'une façon fulgurante. Cette explosion est bien soutenue par l'essor des réseaux informatiques et particulièrement par l'émergence de l'internet. Sans doute, l'internet est devenu un vecteur de propagation important des infections informatiques. En parallèle, de nouvelles infections peuvent provoquer, quasi instantanément, des dégâts dans les systèmes d'information et engager la responsabilité pénale des utilisateurs. Ceux-ci ont pourtant du mal à saisir les risques de ces nouvelles attaques, et ne se protègent pas assez. Dans ce chapitre, on se limite à répondre aux questions suivantes :

- Qu'est-ce qu'une infection informatique ?
- Quels sont les différents types de l'infection informatique ?
- Quel est l'impact économique des infections informatiques ?

2. Infection informatique

Il existe plusieurs définitions de l'infection informatique. Cependant, Éric Filiol dans son célèbre ouvrage ***[Filiol, 2009]*** a proposé une définition plus générale, en ajoutant le fait qu'une infection peut inculper l'utilisateur touché dans un crime qu'il n'a jamais commis.

Définition 2.1 (infection informatique) :

Programme simple ou autoreproducteur, à caractère offensif, s'installant dans un système d'information, à l'insu du ou des utilisateurs, en vue de porter atteinte à la confidentialité, l'intégrité ou la disponibilité de ce système, ou susceptible d'incriminer à tort son possesseur ou l'utilisateur dans la réalisation d'un crime ou d'un délit ***[Filiol, 2009].***

Cette définition récapitule les dommages qu'une infection peut causer :

- Violer la confidentialité des informations du système touché.
- Utiliser les ressources du système en faveur de l'attaquant, ce qui diminue la performance et la disponibilité du système.
- Détruire et endommager les données ou les composantes matérielles.
- Incriminer l'utilisateur dans des crimes ou des délits, en utilisant son système dans l'attaque d'autres systèmes.

3. Infections informatiques selon la classification d'Adleman

Le terme « virus » est généralement utilisé pour désigner tous les programmes malveillants ***[Filiol, 2009]***. Néanmoins, Adleman a regroupé les infections informatiques (programmes malveillants) en deux classes principales (infections simples et codes autoreproducteurs) qui seront présentées dans la section suivante (**Figure 2-1**).

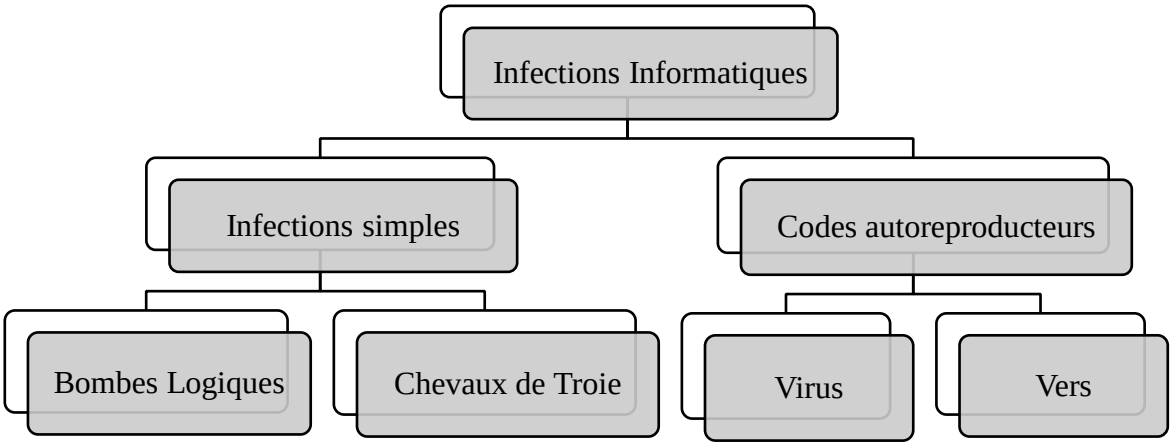


Figure 2-1 Classification des infections informatiques d'Adleman *[Filiol, 2009]*

3.1. Infections Simples

Les infections simples se divisent essentiellement en deux sous classes : bombes logiques et chevaux de Troie.

3.1.1. Bombe logique

Définition 2.2 :

Une bombe logique est un programme injectant simple, s'installant dans le système qui attend un événement (date, action, données particulières...) appelé en général « gâchette », pour exécuter sa fonction offensive ***[Filiol, 2009]****.*

La bombe logique se constitue de deux parties ***[Aycock, 2006]*** :

- Charge finale : une action exécutable qui présente un effet malveillant.
- Gâchette : une expression booléenne qui s'évalue pour contrôler le déclenchement de la charge finale. Elle dépend de l'imagination du programmeur, et pourrait être une date ou une action de la part de l'utilisateur.

Exemple d'une bombe logique : un administrateur système ayant implanté un programme vérifiant la présence de son nom dans les registres de feuilles de paie de son entreprise. En cas d'absence de ce nom (ce qui signifie par exemple que l'administrateur a été renvoyé), le programme chiffrerait tous les disques durs. Ce qui empêchera l'entreprise d'utiliser ses donné à moins d'avoir la clé de chiffrement ***[Filiol, 2009]***.

3.1.2. Cheval de Troie

Définition 2.3 :

Un cheval de Troie est un programme qui prétend faire des tâches bienveillantes, mais accomplit secrètement une certaine tâche malveillante additionnelle ***[Aycock, 2006]***

Définition 2.4 :

Un cheval de Troie est un programme simple, composé de deux parties : le module serveur et le module client. Le module serveur, installé dans l'ordinateur de la victime, donne discrètement à l'attaquant accès à tout ou une partie de ses ressources dont il dispose via le réseau (en général), grâce à un module client (il est le « client » des « services » délivrés inconsciemment par la victime) ***[Filiol, 2009]****.*

- Le module serveur : un programme dissimulé dans un autre programme à l'insu de la victime.
- Le module client : un programme installé chez l'attaquant afin de prendre le contrôle aux machines où le module serveur a été installé. Il utilise la commande « **Ping »** pour trouver les machines infectées. Le mécanisme d'action de cheval de Troie est illustré par la figure suivante (**Figure 2-2**).

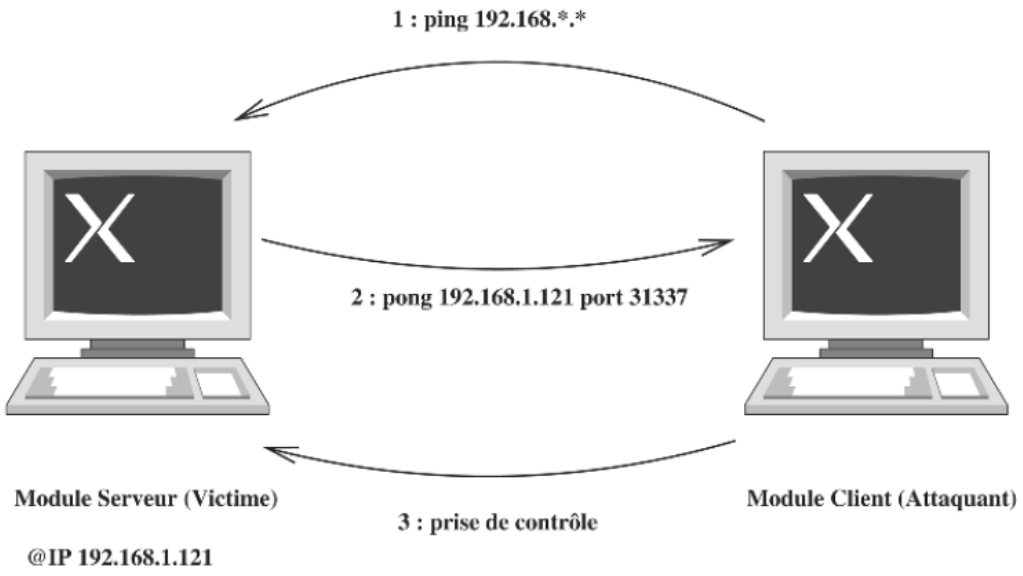


Figure 2-2 Mécanisme d'action d'un cheval de Troie ***[Filiol, 2009]***

3.2. Infections autoreproductrices

Les infections autoreproductrices sont caractérisées par une duplication de code. Selon cette duplication, on peut classifier ces infections en deux types :

- Virus : dans ce type d'infection, la duplication est limitée à la machine où on aura plusieurs copies de virus au sein du système cible.
- Ver : la duplication se fait de machine en machine à travers le réseau.

3.2.1. Virus

D'après Fred Cohen qui a donné une définition assez précise de la notion de virus informatique, un virus informatique est :

Définition 2.5 :

Un virus est une séquence de symboles qui, interprétée dans un environnement donné (adéquat), modifie d'autres séquences de symboles dans cet environnement, de manière à y inclure une copie de lui-même, cette copie ayant éventuellement évolué. ***[Filiol, 2009]***.

De cette définition on peut dire qu'un virus est un programme qui, une fois exécuté, modifie d'autres programmes (hôte), c'est-à-dire, se greffe sur un programme utilisé sur le système cible afin d'en modifier le comportement.

Une fois implanté sur son hôte, le greffon possède aussi en général la capacité de se recopier sur d'autres programmes, ce qui accroît la virulence de l'infection et peut contaminer tout le système, en conséquence la désinfection sera plus laborieuse.

3.2.2. Ver

Un ver informatique représente une autre catégorie de programmes appartenant à la famille des programmes autoreproducteurs. Il est possible de le considérer comme étant un virus particulier, capable de propager l'infection à travers un réseau ***[Filiol, 2009]***.

Définition 2.6

Un ver est un code malveillant qui se propage souvent via les réseaux informatiques en exploitant les failles de sécurité des ordinateurs connectés. En général, les vers n'ont pas besoin d'intervention humaine pour se propager. Cependant, une catégorie de vers appelés vers passifs, lors de sa propagation, exige un comportement d'hôte ou bien une intervention humaine ***[LI, et al., 2008]****.*

Les vers partagent plusieurs caractéristiques avec les virus informatiques, dont la plus importante est l'autoreproduction. Cependant, cette autoreproduction est différente de celle des virus informatiques en deux points ***[Aycock, 2006]***:

- Les vers sont autonomes, c'est-à-dire qu'ils n'ont pas besoin d'un fichier hôte à infecter pour se reproduire.
- Les vers se propagent de machine en machine à travers un réseau contrairement aux virus informatiques qui se propagent localement dans le système cible.

4. Fonctionnement des virus informatiques

4.1. Diagramme fonctionnel d'un virus informatique

Contrairement aux virus biologiques, il est difficile de parler de structure d'un virus informatique, nous parlerons plutôt de diagramme fonctionnel (**Figure 2-3**). Ce dernier résume de manière schématique l'articulation des différentes routines composant un programme autoreproducteur.

Un diagramme fonctionnel standard comprend ***[Ludwig, 1996]***:

- **Routine de recherche de la cible** : Avant d'infecter un nouveau fichier, le virus vérifie si la cible est accessible en écriture, de bon format et n'est pas infectée. En effet, la phase de contrôle de surinfection est importante, car elle améliore l'efficience du virus et limite les possibilités de détection.

- **Routine de copie** : Grâce à cette routine, le virus écrit une copie de son propre code dans sa cible. C'est la présence d'une routine de copie qui fait qu'un simple programme informatique devient un virus informatique (programme autoreproducteur).

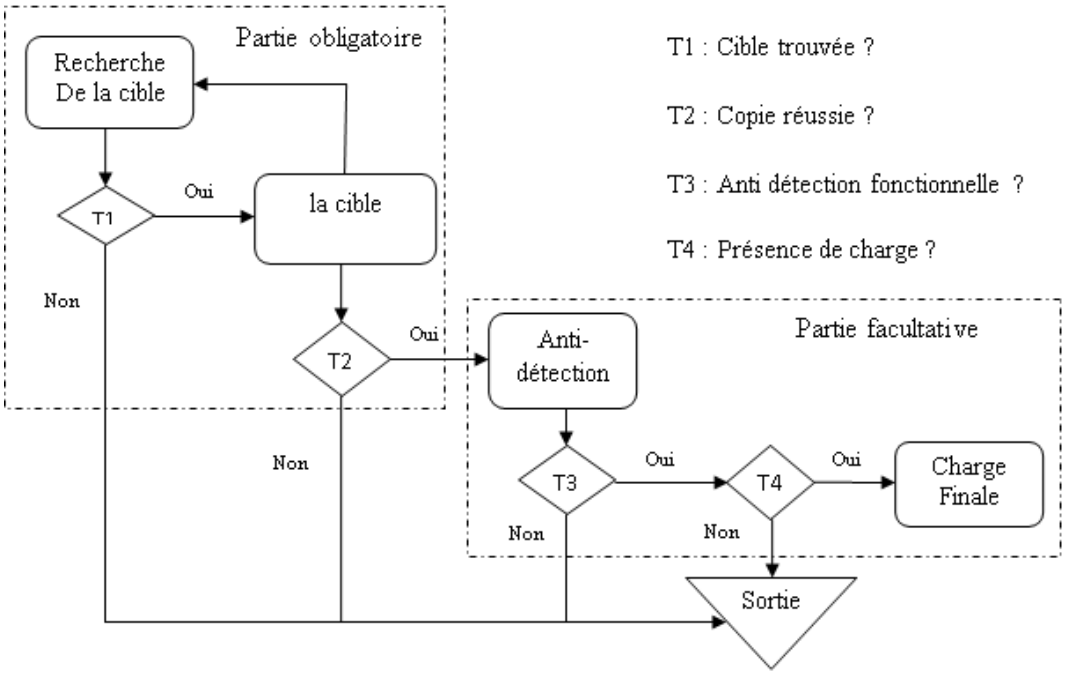


Figure 2-3 Diagramme fonctionnel d'un virus informatique *[Dubois]*.

- **Routine d'anti-détection** : Afin d'améliorer ses chances de survie, un virus doit être indétectable sur le système qu'il infecte.
- **Routine de charge finale** : Le virus peut contenir une charge finale. La volonté de nuire de la plupart des programmeurs de virus fait que la présence d'une charge finale destructive est quasiment systématique. Cependant, cette dernière routine n'est pas obligatoire et n'entre pas dans la définition d'un virus informatique (**Définition 2.5**).

4.2. Techniques d'infection de fichiers

Les virus se reproduisent en infectant les fichiers du système cible. Le processus d'infection consiste donc à identifier le fichier cible et à intégrer dans son code binaire une copie du virus. Le fichier résultant est donc un mélange du fichier original et du virus. Il existe quatre techniques d'infection d'un fichier définissant quatre familles de virus :

4.2.1. Écrasement

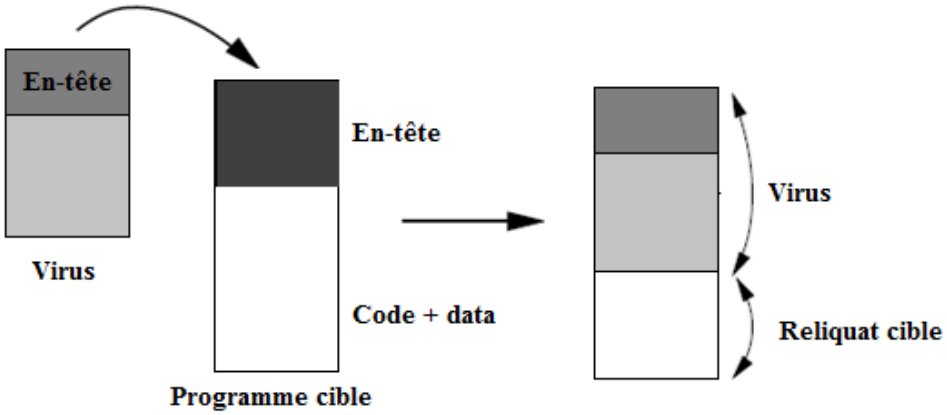


Figure 2-4 Infection par écrasement *[Filiol, 2009]*

Ce virus infect les fichiers cibles en écrasant leur code avec son propre code (**Figure 2-4**), cela permet d'éviter une grande altération de la taille du fichier infecté. Le code du virus peut être placé dans un endroit où il prendra le contrôle ***[Aycock, 2006]***. De toute évidence, le remplacement du code aveuglément provoque des erreurs d'exécution, en conséquence, le virus sera détecté rapidement ***[Ludwig, 1995]***. Ainsi si la désinfection est impossible, les fichiers infectés doivent être supprimés définitivement ***[Al Daoud, et al., 2008]***.

4.2.2. Ajout

Dans ce mode d'infection, le virus ajoute son code à celui de la cible, il en résulte une augmentation de la taille du fichier infecté, si aucune technique de furtivité n'est appliquée le virus sera détecté facilement ***[Filiol, 2009]***.

Ces virus accolent leur code selon deux méthodes :

- **À la fin du code infecté** : Dans cette technique, une instruction du saut (JMP) est insérée au début du code infecté (**Figure 2-5**). Lors de l'exécution de la cible, le virus est exécuté en premier, grâce à l'instruction du saut. Ensuite il rend le contrôle au fichier légitime ***[Filiol, 2009]***. Par cette méthode, le virus peut infecter n'importe quel type de fichier exécutable (EXE, NE, PE, ELF…) ***[Al Daoud, et al., 2008].***

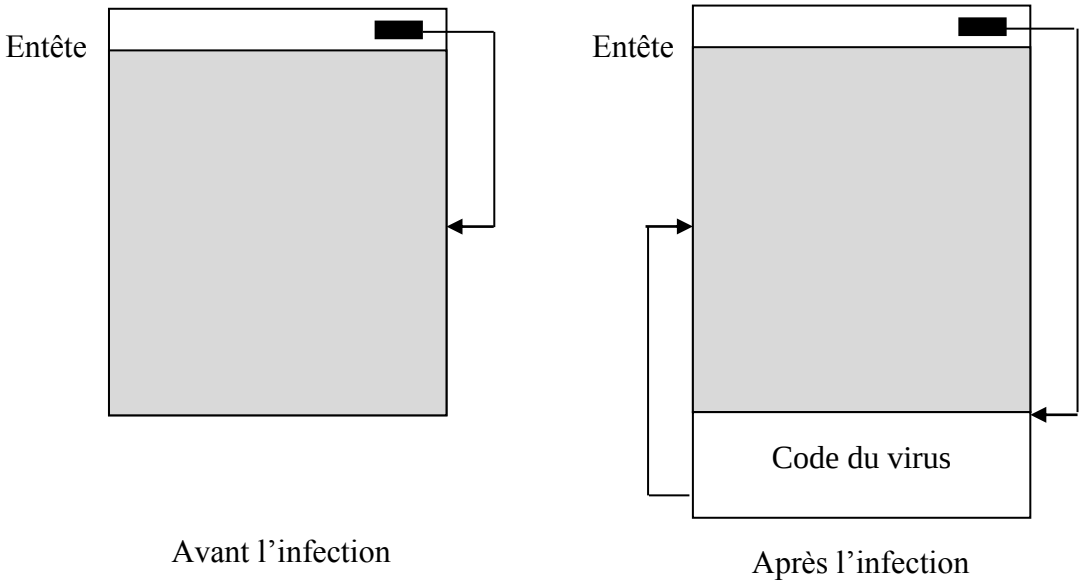


Figure 2-5 Virus d'ajout en fin de fichier.

- **En tête du code infecté** : Ce virus insère son code au début de fichier hôte (**Figure 2-6**) ***[Aycock, 2006]***. C'est le cas le moins fréquent, car il est plus délicat à mettre en œuvre. Un ajout en tête du code nécessite un recalcul d'adresses des données et instructions du programme légitime ***[Filiol, 2009]*.**

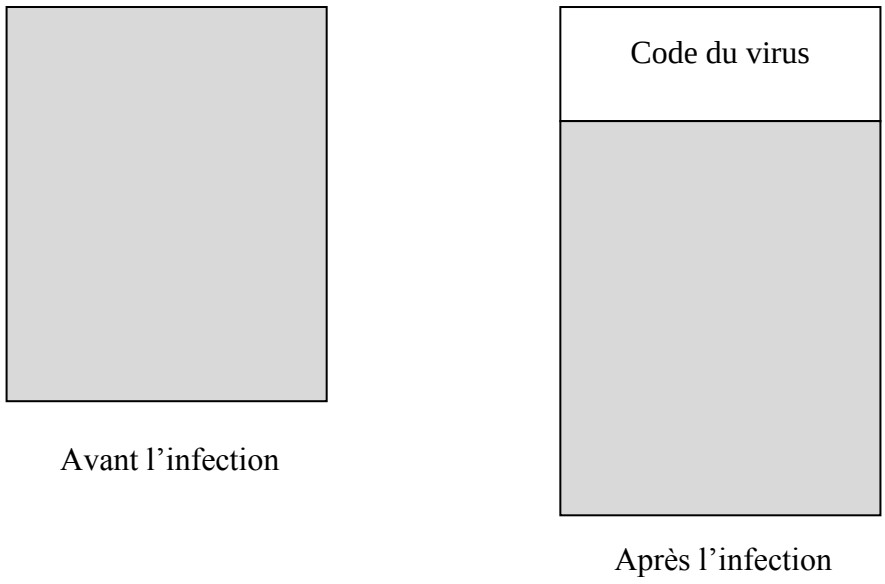


Figure 2-6 Virus d'ajout en tête de fichier.

4.2.3. Entrelacement

Le virus qui utilise cette technique se place dans l'espace non utilisé du fichier cible (**Figure 2-7**). Un avantage recueilli de cette méthode est que le virus n'augmente pas la taille du fichier infecté, par conséquent, il peut éviter le besoin de quelques techniques de furtivité ***[Al Daoud, et al., 2008]***.

À titre d'exemple, les tailles de sections du fichier exécutable de format PE (Portable Exécutable) sont arrondies au multiple de 512 octets supérieurs. C'est dans cet arrondi que réside la vulnérabilité du format PE permettant au virus de s'y intégrer sans changer la taille du fichier exécutable. En effet, si par exemple, le code contenu dans une section occupe 1600 octets, la place qu'il occupe réellement sur le disque est de 2048 octets, il reste donc 448 octets vides ***[Filiol, 2009]***.

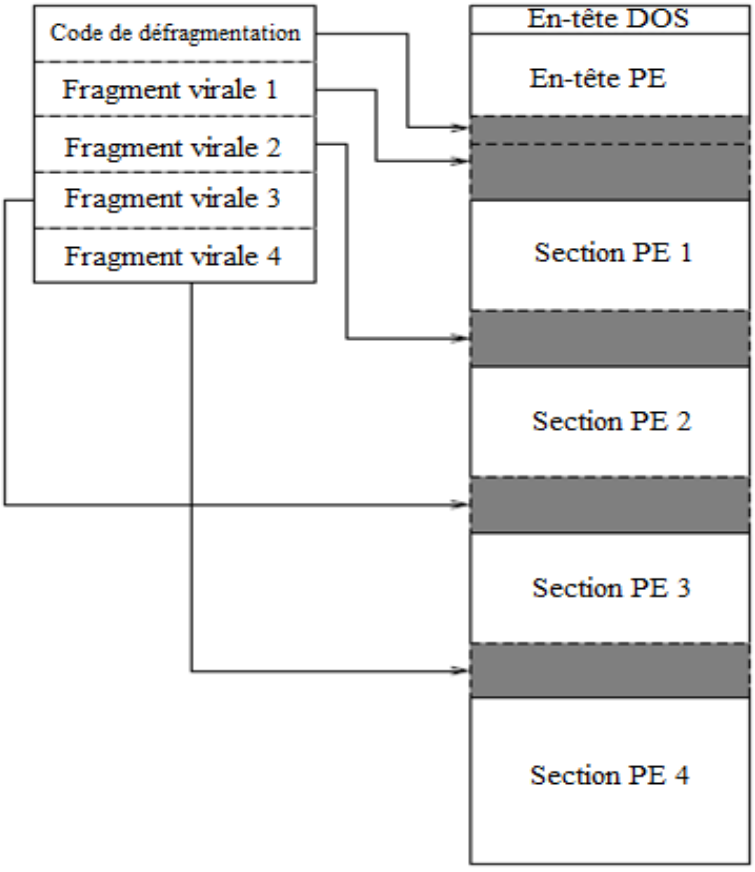


Figure 2-7 Infection par entrelacement de code (fichier PE) ***[Filiol, 2009]***

4.2.4. Accompagnement de code

Dans toutes les techniques précédentes, le fichier hôte est altéré. Par contre cette technique respecte l'intégrité du fichier. En conséquence, la détection des virus utilisant cette approche est un réel défi. Le principe de cette approche est de créer un fichier supplémentaire qui va « accompagner » la cible, ce fichier représente le virus. Lors de l'utilisation du fichier cible, le virus contenu dans le fichier supplémentaire est exécuté en premier. Ensuite, ce dernier appel le fichier légitime (cible) (**Figure 2-8**) ***[Filiol, 2009]***.

Il y a plusieurs manières de mettre en œuvre la technique d'accompagnement, parmi lesquelles on cite :

- Tromper l'utilisateur en renommant le fichier cible par un autre nom, dans ce cas le virus prend le nom du fichier cible. Lors de l'exécution du fichier cible, le virus qu'il accompagne s'exécute en premier, ensuite il rend la main au fichier cible ***[Ludwig, 1995].***
- Donner au virus le même nom que celui d'un fichier cible avec une autre extension plus prioritaire dans la hiérarchie d'exécution du système d'exploitation. Pour le système d'exploitation MS-DOS, les fichiers *.COM sont plus prioritaires que les fichiers *.EXE qui sont, à leurs tours, plus prioritaires que les fichiers *.dat. Lors de l'appel (d'exécution) du fichier sans spécifier son extension, le système va appeler le fichier qui a l'extension la plus prioritaire (le virus) ***[Al Daoud, et al., 2008]***.
- Cacher l'icône du virus sous l'icône d'un programme légitime. Quand l'utilisateur exécute le programme cible en double cliquant sur son icône, le virus s'exécute ***[Harley, et al., 2001]***.
- Modifie la variable PATH du système en insérant le PATH de virus à la place de celui du fichier cible ***[Filiol, 2009]***.

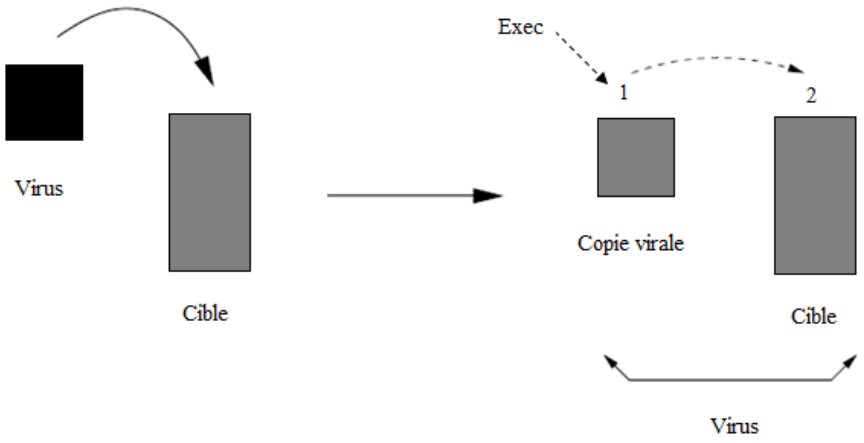


Figure 2-8 Infection par accompagnement de code *[Filiol, 2009]*

4.2.5. Virus de code source

Dans cette technique le virus infecte le code source haut niveau (*.c, *.pas, etc.) plutôt que le fichier exécutable. Ce virus est conçu pour infecter les machines de développeurs software. Cependant, il n'infecte pas les machines passives au développement de software, car ces machines ne contiennent pas de codes sources ***[Ludwig, 1995]*** (**Figure 2-9**). La puissance de ce virus vient du fait que l'exécutable produit présente une homogénéité parfaite, contrairement aux autres modes d'infection, où le code binaire est modifié extérieurement. Cette homogénéité complique la tâche de détection. Un autre avantage de ce mode est son indépendance du système d'exploitation. En effet, l'attaquant pourra juste supposer que sa victime utilise un compilateur conforme à une norme donnée et largement répandu (norme ANSI par exemple pour le langage C) ***[Filiol, 2009]***.

4.3. Techniques anti-antivirales

Tous les virus se reproduisent, mais ils n'agissent pas tous hostilement et sans dissimulation à l'égard de l'antivirus. Les techniques anti-antivirales sont utilisées par les virus afin de ***[Aycock, 2006]*** :

- Attaquer agressivement l'antivirus.
- Compliquer la tâche de détection.
- Compliquer la tâche de son analyse par les experts de sécurité informatique.

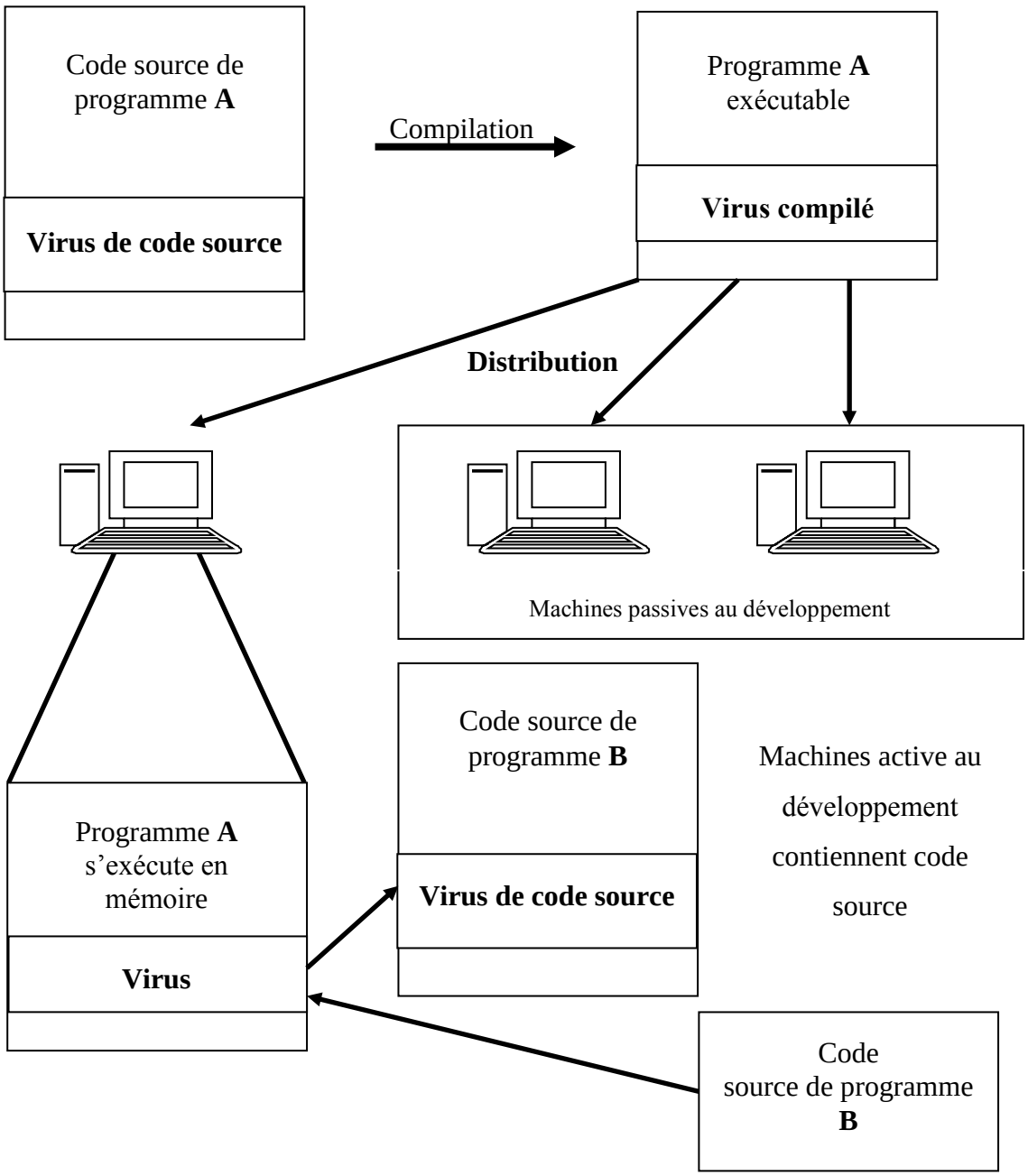


Figure 2-9 Virus de code source *[Ludwig, 1995]*

Les principales techniques utilisées sont présentées dans ce qui suit :

4.3.1. Furtivité

Il s'agit d'un ensemble de techniques visant à leurrer l'utilisateur, le système et les logiciels de protection afin de faire croire à l'absence d'un code malveillant, en le rendant hors de portée de la surveillance ***[Filiol, 2009]***. La Furtivité est tout simplement l'art de convaincre un antivirus que le virus n'est pas là ***[Ludwig, 1995]***.

Parmi les techniques utilisées ***[Filiol, 2009]***:

- La dissimulation dans des secteurs clefs (secteurs déclarés faussement défectueux, zones non utilisées par le système d'exploitation, etc.).
- Leurrer des structures particulières (table d'allocation des fichiers FAT).
- Dans certains cas, le virus peut se désinfecter totalement ou partiellement après l'action de la charge finale.

4.3.2. Polymorphisme

Le but du polymorphisme est de faire varier, de copie en copie virale, tout élément fixe pouvant être exploité par l'antivirus pour identifier le virus ***[Filiol, 2009]***.

Les principales techniques sont les suivantes ***[Aycock, 2006]***:

- Réordonnancement des instructions du virus.
- Utilisation d'instructions équivalentes.
- Utilisation d'une autre séquence de code équivalente.
- Renommage des registres utilisés.
- Changement des emplacements des données dans la mémoire.
- Insertion d'un code inerte en respectant le code original du virus.

4.3.3. Blindage

Définition 2.7:

Capacité plus ou moins grande du code à résister à l'analyse par le désassemblage et le débogage ***[Filiol, 2009]***.

Cette technique a pour objectif de rendre la tâche d'analyse difficile pour les chercheurs d'antivirus. On peut classifier les méthodes de blindage en deux catégories ***[Aycock, 2006]***:

- Anti-débogage : Elle consiste à leurrer le débogueur quand il est détecté, en évitant tout comportement viral.
- Anti-désassemblage : le but de l'anti désassemblage est :
  - Le désassemblage ne doit pas être facilement automatisé, dont le but est de compliquer l'analyse du code.
  - Le code complet ne devrait pas être disponible jusqu'à ce que le code s'exécute effectivement.

4.3.4. Rétrovirus

Ce virus attaque l'antivirus lui-même. Il tente activement de le désactiver ou le désinstaller sur la machine infectée. Pour accomplir cet objectif, le rétrovirus utilise les techniques suivantes ***[Aycock, 2006]***:

- Arrêter les processus utilisés par l'antivirus.
- Un rétrovirus plus agressif peut cibler l'antivirus sur le disque ainsi que dans la mémoire, afin que l'antivirus soit désactivé, même après que le système infecté ait redémarré.
- Diminuer la priorité de l'antivirus.

5. Classification des virus et des vers

La classification des virus et des vers n'est pas une chose simple. Il est relativement difficile de classer un virus informatique. En effet, les virus modernes, afin d'être plus efficaces, combinent plusieurs techniques d'infection ou visent plusieurs systèmes cibles ou encore utilisent plusieurs techniques de camouflage.

5.1. Classification de virus

Les virus informatiques peuvent être classés en fonction de différents critères, comme ***[Filiol, 2009]***:

- Le format de fichier visé : exécutable ou document.
- L'organe cible : secteur de boot, pilote de périphérique, BIOS, etc.
- Le comportement : résident ou non.
- Etc.

5.1.1. Virus d'exécutable

Les virus d'exécutables sont les premiers types de virus apparus historiquement. Leur principe de réplication consiste à se loger, à partir d'un fichier binaire infecté, au sein des fichiers binaires en employant l'un des modes d'infection étudiées précédemment. Ces virus sont donc fortement dépendants du format des fichiers binaires cibles. En effet, chaque type de fichier binaire a sa propre architecture interne, le virus doit donc s'adapter à cette architecture pour infecter sa cible. En raison de ces contraintes, le programmeur de virus d'exécutables doit connaître l'architecture interne des fichiers qu'il veut infecter. De plus, ces virus sont souvent développés en langage assembleur. Les principales cibles binaires sont : *.COM, *.EXE, exécutable au format PE, les fichiers de drivers, fichier VxD de Windows ***[Filiol, 2009]***.

5.1.2. Virus de document

Définition 2.8

Un virus de document est un code viral contenu dans un fichier de données non exécutable, activé par interpréteur contenu de façon native dans l'application associée au format de ce fichier (déterminé par son extension). L'activation du code malveillant est réalisée, soit par une fonctionnalité prévue dans l'application (cas le plus fréquent), soit en vertu d'une faille interne de l'application considérée (de type buffer overflow par exemple) ***[Filiol, 2009]***.

L'avantage de cette définition est qu'elle n'est pas restreinte à la catégorie des virus de documents la plus connue (macrovirus). En fait, la liste des cibles potentielles des virus de documents est relativement importante. Nous pouvons citer notamment ***[Filiol, 2009]***:

- Les documents utilisant des langages de scripts comme HTML, JavaScript, Visual Basic Script, Perl, PHP, Python, Ruby, AppleScript.
- Les documents au format Adobe Acrobat, CorelDraw, AutoCAD, Lotus, Microsoft Office, Microsoft Project, Visio, OpenOffice, MapInfo.
- Les documents permettant de mettre en œuvre des routines spécifiques comme Postscript, LATEX ou TEX, Macromedia et Shockwave.

5.1.3. Virus de démarrage

La définition de Fred Cohen, qui exige qu'un virus est toujours attaché à un autre programme hôte, a provoqué des idées fausses au sujet de l'infection du secteur boot MBR *(*Master Boot Record*)* et du BIOS (Basic Input Output System) ***[Harley, et al., 2001]***, donc le virus informatique a la possibilité d'infecté le MBR ou le BIOS, car ces derniers ne sont que des programmes exécutés au démarrage de l'ordinateur, on appelle ce type de virus par, virus de démarrage. *Alors, un virus de démarrage est un code viral contenu dans un fichier objet exécutable et inséré, soit dans le BIOS d'un ordinateur, soit dans le MBR d'un système de stockage de données. L'activation du code malveillant est réalisée par l'ordinateur lors de sa séquence de démarrage.* ***[Filiol, 2009]***.

5.1.4. Virus non résident

Les programmes viraux non-résidents obtiennent seulement une chance pour se propager à chaque lancement du programme infecté. Le code viral va chercher et infecter un programme cible. Le virus passe ensuite le contrôle au programme original. Ce sont, bien sûr, les plus simples des programmes viraux ***[Harley, et al., 2001].***

5.1.5. Virus résident

Ce type de virus, une fois exécuté, reste actif dans la mémoire du système sous forme d'un processus. Seul l'arrêt de l'ordinateur met fin au processus viral. Une fois en mémoire, le virus a la possibilité d'agir sur le système infecté : désactivation ou inhibition de l'antivirus ou de certaines fonctionnalités du système d'exploitation, infection d'autres exécutables …etc. La différence avec celui vu précédemment est qu'il n'a pas besoin d'appeler la procédure de recherche pour trouver une cible, puisque c'est l'utilisateur qui la désigne en exécutant le programme cible ***[Filiol, 2009]***.

5.2. Classification de vers

Dans la classification des risques informatiques, le ver appartient à la catégorie des programmes autoreproducteurs au même niveau que le virus. Pourtant, la seule particularité du ver, par rapport au virus, réside dans le fait que son vecteur de propagation n'est pas un autre fichier, mais un réseau informatique. Ainsi, la routine de copie du ver consiste à

chercher des systèmes accessibles par le réseau, à s'y connecter et s'y installer. Un ver est donc bien un programme informatique qui a la capacité de se reproduire. À ce titre il répond en tous points à la définition du virus informatique. C'est la raison pour laquelle, un ver peut être considérer comme une sous-classe particulière de virus ***[Filiol, 2009].***

Trois sous-classes de vers sont considérées :

5.2.1. Ver simple

Aussi connus sous le nom de Worm, ils se propagent sur les réseaux. Généralement ce type de vers exploite les failles logicielles permettant l'exécution de programme sur une machine distante. Il exploite également des faiblesses dans les protocoles réseau.

5.2.2. Macro-ver

Sont des programmes hybrides, mélange de macrovirus et de ver. Le mode de dissémination se fait par des pièces jointes contenant des documents bureautiques infectés par un macrovirus. Une fois ouvert, le ver infecte le système et parcourt le carnet d'adresses local afin de s'envoyer par email sous forme de pièce jointe.

5.2.3. Ver d'email

Aussi connus sous le nom de mass-mailing Worm, ces vers se propagent de façon fulgurante sous forme de pièce jointe d'email. Ils utilisent les failles du lecteur d'email cible pour s'exécuter automatiquement lors de leur arrivée sur un ordinateur.

6. Similitude entre les virus biologiques et informatiques

Les termes utilisés en virologie informatique (infection, virus, …) incitent d'emblée à établir un parallèle avec la biologie. En effet, le choix du terme « virus » n'est pas fortuit, il est indiqué par le fait que la nature d'un virus informatique correspond bien à celle du virus biologique. En conséquence, tout mécanisme viral biologique trouve son équivalent dans le monde des virus informatiques ***[Filiol, 2009]***. Toutefois, plusieurs chercheurs ont proposé des méthodes antivirales inspirées du système immunitaire biologique. Les pionniers de cette approche sont Kephart dans ***[Kephart, et al., 1995]*** et Forrest en 1997 ***[Forrest, et al., 1994]***.Le tableau (**Tableau 2-1**) ci-dessous résume les similitudes entre les virus biologiques et informatiques.

Virus biologiques	Virus informatiques
Attaques spécifiques de cellules	Attaques spécifiques de format
Les cellules touchées produisent de nouveaux virus	Le programme infecté génère d'autres programmes viraux
Modification de l'information génétique de la cellule	Modification des actions du programme
Le virus utilise les structures de la cellule hôte pour se multiplier	Multiplication uniquement via un programme infecté
Interactions virales	Virus binaire ou virus – antivirus
Multiplication uniquement dans des cellules vivantes	Nécessité d'une exécution pour la dissémination
Une cellule infectée n'est pas surinfectée par le même virus	Lutte contre la surinfection
Rétrovirus	Virus luttant spécifiquement contre un antivirus – virus de code source
Mutation virale	Polymorphisme viral
Antigènes	Signatures

Tableau 2-1 Virus biologiques – Virus informatiques : Comparaison (Filiol, 2009)

7. Impact économique des virus informatiques

Un virus peut perturber plus ou moins gravement le fonctionnement de l'ordinateur infecté, dont certains virus informatiques augmentent le voltage de certains composants, provoquant ainsi une chauffe inhabituelle. Or, lors d'une chauffe du microprocesseur par exemple, l'ordinateur ralentit ou même s'arrête pour éviter une détérioration du matériel. Le virus peut se répandre à travers tout moyen d'échange de données numériques comme les réseaux informatiques, CD-ROM, clefs USB,… etc. Afin d'illustrer l'importance du risque viral, résumons-le par quelques Chiffres particulièrement pertinents ***[Filiol, 2009]*** :

- Le ver « *I LoveYou* » a infecté en 1999 plus de 45 millions d'ordinateurs dans le monde.
- Le ver « *Sapphire/Slammer* » a infecté 200 000 serveurs dans le monde, dont plus de 75000 l'ont été dans les dix premières minutes suivant le début de l'infection.
- Le ver « *W32/Sobig.F »* a infecté, en Août 2003, plus de 100 millions d'utilisateurs.
- Le ver « *W32/Lovsan »* a également frappé en Août 2003, infectant TOUS les abonnés d'un grand fournisseur d'accès Internet.
- Le virus « CIH » dit « Chernobyl » a obligé des milliers d'utilisateurs, en 1998, à changer la carte mère de leur ordinateur après en avoir détruit le programme BIOS. Les dégâts provoqués par ce virus sont estimés à près de 250 millions d'euros pour la seule Corée du Sud, par exemple.
- Enfin, fin janvier 2004, le ver d'emails « *MyDoom »* a infecté plus de cent millions de mails dans les trente-six premières heures après le début de l'infection.

Ces chiffres montrent avec force les pertes économiques causées par les infections informatiques ce qu'il les rend un problème crucial, non seulement pour les individus, mais pour l'industrie et les gouvernements.

8. Nouvelle tendance des infections informatiques

Le développement des virus, au début, a été une sorte d'amateurisme pratiqué par les programmeurs pour des buts personnels. Néanmoins et durant les dernières années, cette industrie des infections devenait une cybercriminalité ayant des buts économiques et politiques bien déterminés. Cette nouvelle ère des infections est caractérisée par une convergence vers des crimes financiers tout en diminuant leur visibilité (**Figure 2-10)**. Effectivement, les nouvelles infections visent un gain financier bien étudié qui s'additionne aux dégâts touchant les utilisateurs et les entreprises ***[OECD, 2008; Gostev, 2012; RSA, 2012]***.

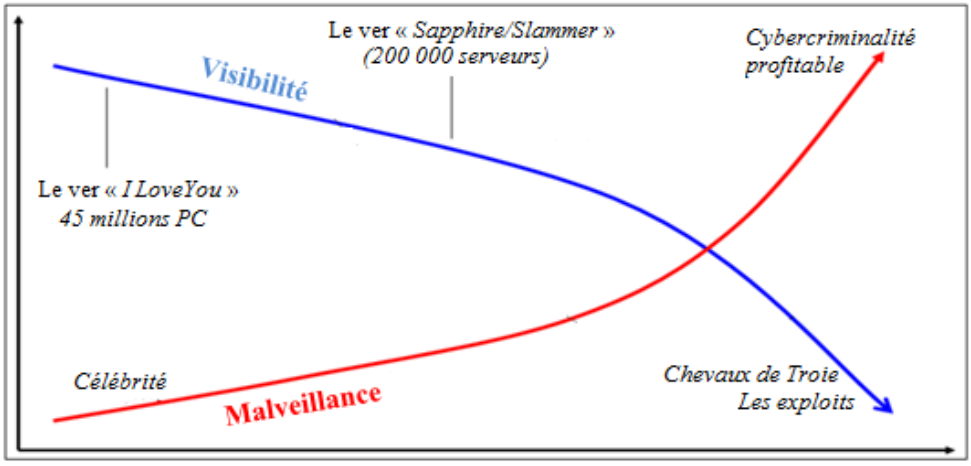


Figure 2-10 Visibilité vs. Malveillance des infections informatiques *[OCED, 2008]*

Depuis la découverte du ver ***Stuxnet*** en 2010, il a été considéré comme un «changement de paradigme» de cybercriminalité. Il présent une sophistication inhabituelle et inaugure une nouvelle dimension dans les attaques informatiques. Les experts de sécurité informatique estiment que cette infection très complexe a été développée par un organisme bien financé, privée ou gouvernemental, pour attaquer l'infrastructure industriel de l'Iran (le centrale nucléaire de Bushehr) ***[Greengard, 2010; Langner, 2011]***.

En effet, Stuxnet a changé la thèse des chercheurs en sécurité sur le fait que les infections informatiques se limitent aux ordinateurs (**Tableau 2-2**), dont, ils peuvent affecter les infrastructures physiques contrôlés par des logiciels, ce qu'implique que les menaces pourraient s'étendre à la vie réelle ***[Chen, et al., 2011; Karnouskos, 2011]***. En d'autre terme, l'exemple de Stuxnet renforce l'hypothèse d'occurrence d'une guerre électronique en perspective.

	Stuxnet	**Les autres infections**
La recherche de la cible	Extrêmement sélectif	Sans sélections
Type de cible	Systèmes de contrôle industriel	PCs
La Taille	500 Ko	Inferieur 1Mb
La méthode d'infection	Flash disque	Réseaux informatiques(Internet)
Exploits[1]	Quatre zero-day	Eventuellement un zero-day

Tableau 2-2 Les nouvelles caractéristiques de Stuxnet *[Chen, et al., 2011]*

[1] Un exploit est un élément de programme permettant à un individu ou un logiciel malveillant d'exploiter une faille de sécurité informatique dans un système d'exploitation ou dans un logiciel.

9. Conclusion

Dans ce chapitre, nous avons défini la notion d'infection informatique, en donnant les différents types d'infection. Nous avons aussi présenté le mode de fonctionnement des différents types d'infection comme les virus, vers, chevaux de Troie, bombe logique.

Pour lutter contre ces infections, beaucoup de chercheurs ont proposé de diverses techniques de détection heuristique, y compris les techniques de datamining et l'apprentissage automatique, afin d'améliorer l'efficacité de la détection et de compléter les insuffisances des autres techniques déjà utilisé dans la sécurité informatique. Dans le chapitre suivant, nous présentons un état de l'art des travaux qui ont étudié le datamining pour la détection des programmes malveillants.

Chapter 3 : DETECTION DE VIRUS INFORMATIQUES BASEE SUR LE DATAMINING

1. Introduction

Depuis l'apparition du premier système de détection de virus basé sur le datamining (VDS-DM) ***[Schultz, et al., 2001]***, plusieurs autres approches ont vu le jour. Ces systèmes exploitent des données disponibles sur les attaques précédentes pour aboutir à une méthode de détection plus intelligente. En outre, l'approche datamining ajoute un degré d'automatisation aux techniques antivirales.

Avant de proposer notre VDS-DM, il s'avère indispensable de faire une synthèse de ces systèmes en catégorisant les méthodes existantes. L'objet de ce chapitre est de présenter un état de l'art sur les VDS-DM. Pour cela, nous présentons les différents outils de datamining évoqués dans ces systèmes. Ces outils de datamining sont classés en trois axes principaux :

- La représentation de programmes par le biais d'un ensemble de caractéristiques.
- La sélection de caractéristiques pertinentes de point de vue détection de virus.
- Les algorithmes de classification de programmes afin de détecter ceux qui sont malveillants.

2. Datamining

Le datamining représente la résolution de problème en analysant les données dans les bases de données disponibles. C'est un processus de découverte automatique ou semi-automatique de modèles à partir de données ***[Witten, et al., 2011]***. Afin de s'appuyer sur ces modèles comme moyens d'aide à la décision, ces derniers doivent nécessairement obéir à deux conditions :

- Être nouveaux et non connues auparavant. Ceci qui reflète la notion de découverte.
- Être utile lorsqu'ils sont utilisés dans l'aide à la décision en fournissant des valeurs ajoutées (gain économique, gain de temps, etc.).

L'orientation vers les techniques de datamining, pour lutter contre les virus informatiques, est incitée par l'abondance de données disponibles sur les attaques informatiques. La thèse de datamining dans la détection de virus informatique est la suivante :

L'exploitation de techniques du datamining pour fournir des modèles utiles tout en exploitant les bases de données disponibles sur les attaques précédentes.

3. Apprentissage automatique supervisé

Lors du processus datamining, plusieurs techniques sont appelées. Ces techniques sont dérivées de différentes disciplines (statistiques, algorithmiques, base de données, etc.). Parmi les domaines très impliqués, *l'apprentissage automatique* qui est défini par la science qui étudie des méthodes et les algorithmes pour doter la machine par la faculté d'apprendre. Dans l'apprentissage automatique, l'étude est centrée sur l'*exemple,* ce dernier est caractérisé par ses valeurs sur un ensemble de *caractéristiques* prédéfinies. Dans notre contexte, l'*exemple est un programme* représenté par le biais d'un ensemble de *caractéristiques de programme.*

La détection est un problème de classification binaire (la classe de virus et les programmes bienveillants). Par conséquent, la *classification*, appelée également *apprentissage supervisé,* est un choix adéquat. Elle consiste à utiliser un ensemble d'exemples nommé *ensemble d'entrainement* pour générer *un classificateur* capable de classifier un exemple en regardant seulement ses caractéristiques. L'ensemble d'entrainement contient des exemples avec des classes ou *exemples étiquetés* sous la forme (exemple, classe). Fréquemment, un autre ensemble est utilisé pour mesurer la performance du classificateur appelé *ensemble test*. Les

deux ensembles précédents (entrainement et test) sont indépendants afin de bien évaluer le classificateur et sa capacité de généralisation. Le tableau (**Tableau 3-1**) montre le lien entre l'apprentissage supervisé et la détection de virus.

Apprentissage supervisé	*Détection de virus informatiques*
Exemple	Programme informatique.
Caractéristique	Caractéristiques de programme.
Exemple étiqueté	Programme avec classe connue (virus/bienveillant).
Ensemble d'entrainement	Ensemble de programmes avec des classes connues (Programmes étiquetés).
Ensemble test	Ensemble de programmes pour évaluer la performance de détection.
Les classes	$C = \{virus, programme\ bienveillant\}$

Tableau 3-1 L'apprentissage supervisé et la détection de virus

Finalement, l'objective de l'application de l'apprentissage supervisé est le suivant :

L'utilisation des algorithmes d'apprentissage supervisé avec deux classes (virus/bienveillant) pour produire un système de détection de virus informatiques.

4. Système de détection de virus informatique basé sur l'apprentissage supervisé (VDS-DM)

Pour parvenir à un classificateur utilisable dans la détection, on passe par deux phases principales, comme le montre la **(Figure 3-1)**

1. Phase d'entrainement :

Dans cette phase, *une représentation de programmes* composée d'un ensemble de caractéristiques est élaborée.

Ensuite, cette représentation est raffinée par une technique appelée *sélection de caractéristiques* afin de ne garder que les caractéristiques pertinentes.

Puis l'ensemble d'entrainement est formé par des programmes étiquetés et représentés selon la représentation déjà produite. Finalement, des *algorithmes de génération de classificateurs* sont utilisés, avec l'ensemble d'entrainement comme entrée, pour produire le classificateur.

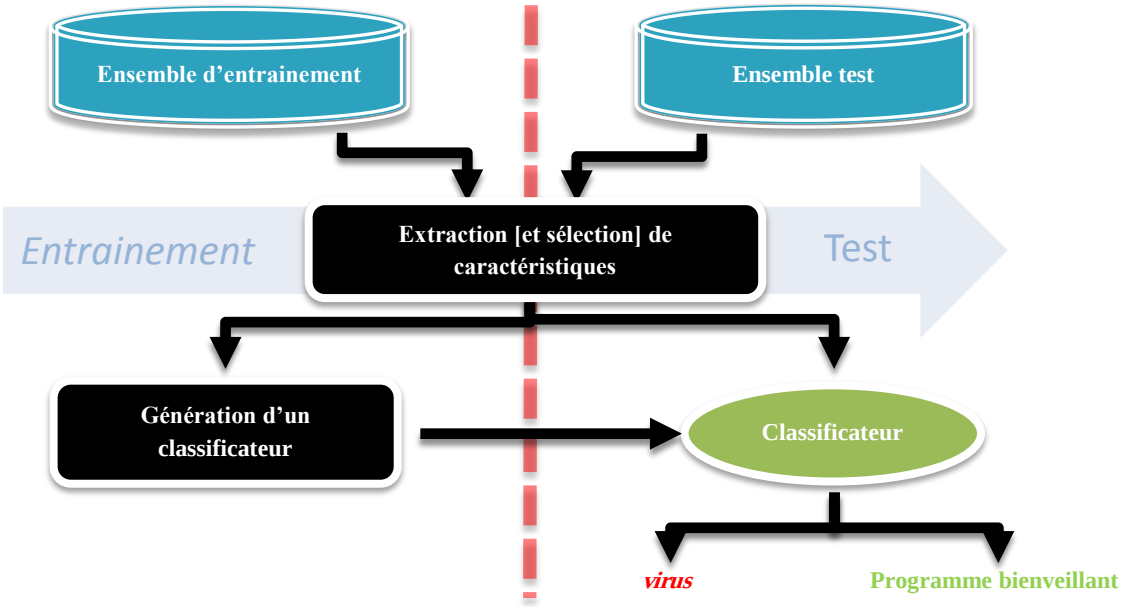


Figure 3-1 L'apprentissage supervisé pour la détection de virus

1. Phase de test

Après avoir formé un classificateur qui peut prédire la classe de programme, la question qui se pose est : est-ce que ce classificateur est performant en matière de prédiction de la classe réelle du programme ? Pour répondre à cette question, on compare la classe prédite et réelle (étiquette) pour chaque exemple dans l'ensemble test. Autrement dit, l'évaluation se fait en cachant les étiquettes des exemples de test et en laissant le classificateur les deviner.

Dans la suite du chapitre, nous passons en revue (1) les représentations de programmes, (2) les méthodes de sélection de caractéristiques et (3) les algorithmes de génération d'un classificateur. Notre but est d'exposer les différentes approches existantes dans la détection de virus afin de former une vue d'ensemble qui nous permet par la suite de proposer notre système de détection.

5. Représentation de programmes

Dans le domaine de détection de virus informatiques par le datamining, la représentation des programmes est indispensable pour les présenter au classificateur.

Les caractéristiques utilisées pour représenter les programmes peuvent être catégorisées selon le type d'analyse en deux classes. Dans l'analyse statique, les caractéristiques sont extraites directement à partir du programme hors contexte d'exécution. Tandis que celles qui sont dynamiques, sont extraites à partir du comportement durant l'exécution.

5.1. Caractéristiques statiques

Ces caractéristiques sont extraites directement à partir du code du programme sans la nécessité de son exécution ***[Dube, et al., 2012]***. Dans les sections suivantes, nous citons quelques types des caractéristiques statiques utilisés dans l'état de l'art.

5.1.1. Représentation n-gramme

Afin d'entamer ce type de représentation, il convient de définir la notion de n-gramme comme suit :

Un n-gramme est une sous-séquence $(G_i G_{i+1} \dots G_{n+i-1})$ *de n grammes extrait à partir d'une autre séquence* $(G_0 G_1 \dots G_l)$.

Cette notion est utilisée dans la représentation d'un document D composé de mots. L'unité gramme dans ce cas est un mot et chaque n-gramme est un n mots consécutifs extraits à partir de D. Dans cette représentation, une fenêtre de (n) mot est glissée sur D pour fournir tous les n-gramme possibles ***[Sammut, et al., 2011]***.

Afin de fournir une représentation de programmes, les chercheurs ont adapté les n-grammes. Cette adaptation consiste à voir le programme comme étant une séquence d'octet (n-gramme d'octet) ou de mnémoniques d'instruction (n-gramme de code d'opération).

a. N-gramme d'octets

N-gramme d'octets est un fragment de ***n*** octet extrait directement à partir du code binaire du programme (représenté en hexadécimale) sans prétraitement préalable. L'exemple suivant montre le processus d'extraction de n-gramme à partir d'un programme *p* donné, où ***n=3*** :

[12 F5 12] F5 12 → n-grammes produits à partir de la fenêtre F_1 = {12 F5 12}

12 [F5 12 F5] 12 → n-grammes produits à partir de la fenêtre F_2 = {F5 12 F5}

12 F5 [12 F5 12] → n-grammes produits à partir de la fenêtre F_3 = {12 F5 12}

Donc l'ensemble de n-grammes extraits à partir de *p* est ***E* = {12 F5 12, F5 12 F5}.**

Cette représentation génère un nombre élevé de n-grammes, ce qui exige l'utilisation d'une stratégie de sélection de caractéristiques. En plus, les n-gramme d'octets ne contiennent pas des informations interprétables directement. Malgré toutes ces limitations, plusieurs études montrent l'exactitude des systèmes de détection qui utilisent ce type de caractéristiques ***[Abou-Assaleh, et al., 2004; Kolter, et al., 2004a; Kolter, et al., 2006b; Moskovitch, et al., 2008d; Shabtai, et al., 2009; Dube, et al., 2012] .***

b. N-gramme de code d'opération

Ce type de N-gramme est une série de ***n*** code d'opération extraite à partir du désassemblage du programme. Le code d'opération représente la partie de l'instruction (instruction du langage machine) qui spécifie l'opération à effectuer en omettant les opérandes ***[Shabtai, et al., 2009]***. L'extraction de n-gramme de code d'opération passe par deux phases (**Figure 3-2**) ***[Shabtai, et al., 2012]***:

1. Désassemblage de la partie code de programme.
2. Extraction de n-grammes après l'élimination des opérandes en considérant un gramme comme un code d'opération.

L'efficacité de cette représentation a été mesurée empiriquement par ***[Karim, et al., 2005; Bilar, 2007; Siddiqui, et al., 2008; Moskovitch, et al., 2008a; Shabtai, et al., 2012;***

Santos, et al., 2011]. Les résultats expérimentaux ont montré l'exactitude des n-gramme de code d'opération dans la détection de virus informatiques. Néanmoins, l'extraction des n-gramme de code d'opération n'est pas toujours possible à cause de la difficulté due au désemballage ***[Shabtai, et al., 2009]***.

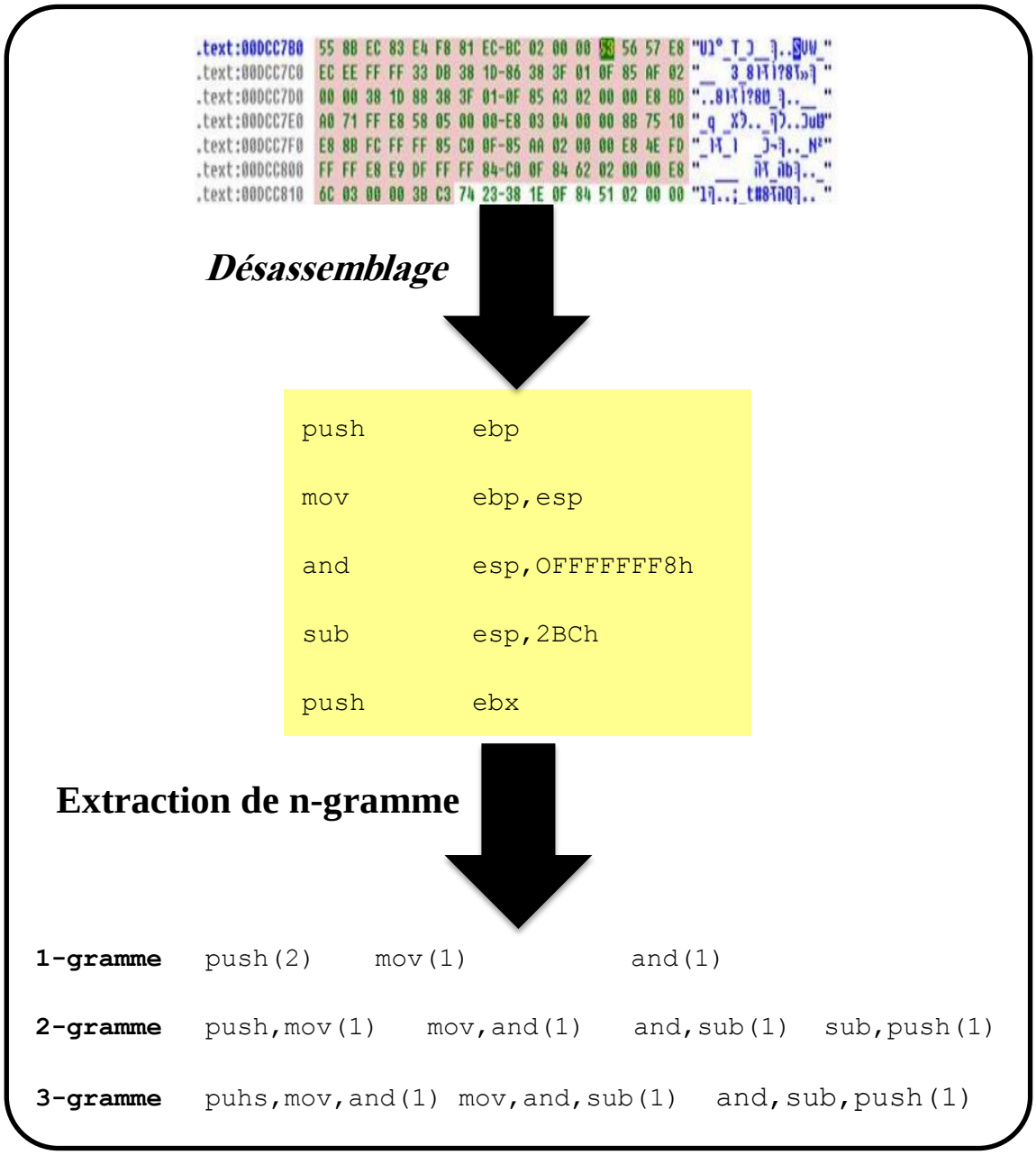


Figure 3-2 N-gramme de code d'opération (Shabtai, et al., 2012)

5.1.2. Chaines de caractères

Les chaines de caractères sont des textes inclus dans le code source, elles sont de différentes formes (chemins de fichiers, textes affichables, …etc.) ***[Islam, et al., 2013]***. Elles sont utilisées premièrement par ***[Schultz, et al., 2001]*** tout en montrant une exactitude de détection acceptable (>90%).

5.1.3. Caractéristiques extraites du format PE

Avant de citer ces caractéristiques, il convient de donner une vue globale sur ce format de fichier. Le format PE (Portable Executable) est une spécification dérivée du format Unix COFF (Common Object File Format) pour les exécutables Windows.

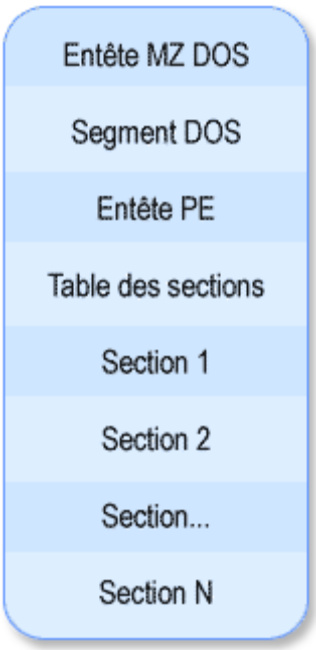


Figure 3-3 Organisation d'un fichier avec format PE
[Lance, 2005]

Un fichier de ce format (**Figure 3-3**) est composé de :

- L'entête MS-DOS : il permet au système d'exploitation de reconnaitre le fichier au moment du lancement à partir d'MS-DOS et ensuite il exécute le segment DOS.
- L'entête PE : il décrit le fichier PE en fournissant des informations utiles pour son chargement (nombre de sections, temps de création/modification, type de machine, la taille… etc.).

- Le tableau de sections : composé d'un ensemble de structures, chaque structure contient des informations sur une section du PE (attribut lecture/écriture, déplacement dans le fichier, déplacement dans la mémoire …etc.)
- Les sections : le contenu d'un fichier PE est divisé en sections. Une section est un bloc de données et/ou code qui ont les mêmes attributs. A titre d'exemple, on peut regrouper dans une section le code et les données qui sont en lecture seule.
- Le chargeur PE (PE loader) : est un élément du système d'exploitation (Windows) qui permet de connaitre et de charger en mémoire les fichiers PE.

La richesse de ce format en matière d'information a incité les chercheurs d'en tirer profit. En effet, plusieurs études sont basées sur des caractéristiques extraites à partir de ce format. Parmi ces caractéristiques nous citons ***[Tahan, et al., 2012; Schultz, et al., 2001]***:

- Les données extraites à partir d'entête PE qui décrit la structure du fichier (temps de création / modification, type de la machine, la taille du fichier…Etc.)
- Les caractéristiques d'interaction du fichier exécutable avec le système via les bibliothèques de liens dynamiques (DLL) (liste des DLL utilisées, liste de fonctions DLL appelées… etc.)
- Les ressources utilisées par l'exécutable.

5.1.4. Caractéristiques extraites du Graphe de flot de contrôle (GFC)

C'est une représentation sous forme de graphe de tous les chemins qui peuvent être suivis par un programme durant son exécution. Plus précisément, c'est un graphe où les blocs de base sont des nœuds et les chemins d'exécution reliant ces blocs représentent des arcs. Un bloc de base est une séquence d'instructions qui ne contient aucun branchement (conditionnel ou inconditionnels) ***[Cesare, et al., 2012]***.

Le processus d'extraction de caractéristiques à partir de GFC passe par trois phases illustrées dans la **Figure 3-4**:

1. Le désassemblage du code afin de repérer les branchements.

2. La création du GFC en se basant sur les branchements[2].
3. L'extraction des caractéristiques à partir du CFG tout en profitant de la théorie de graphe.

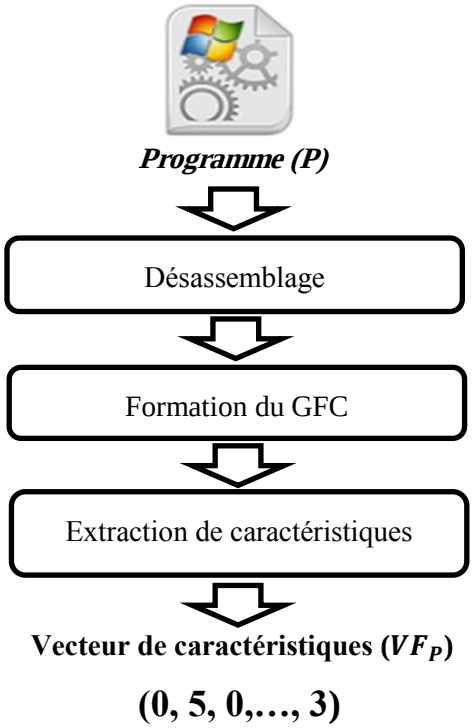


Figure 3-4 Extraction de caractéristiques à partir du Graphe de flot de contrôle (GFC)

Les caractéristiques extraites sont des statistiques sur la structure du graphe (Nombre de nœuds, Nombre d'arcs, Degrés des nœuds, Matrice d'adjacence,…etc.) ***[Eskandar, et al., 2011; Eskandari, et al., 2012; Chao, et al., 2009; Sirageldin, et al., 2012]***. Ces caractéristiques produites sont rangées dans un vecteur de caractéristiques ***[Eskandar, et al., 2011].***

L'avantage de cette représentation réside dans la capture de la sémantique d'exécution en traçant les chemins d'exécutions ***[Eskandar, et al., 2011]***. De plus, elle est moins sensible au changement de l'ordre d'instructions. En effet, le réordonnancement d'instructions au sein d'un bloc de base n'influe pas sur la structure du GFC.

[2] Les branchements déterminent les blocs de base et les arcs entre eux.

5.2. Caractéristiques dynamiques

L'analyse statique soufre de plusieurs limitations qui rendent l'extraction des caractéristiques statique difficile. En effet, les virus informatiques utilisent les techniques de blindage et de cryptage afin d'induire en erreur l'analyse de code ***[Tilborg, et al., 2011]***. De ce fait, l'analyse dynamique fait l'extraction de caractéristiques lors de l'exécution de programmes.

L'analyse dynamique est le processus d'analyse de comportement d'un programme dans un environnement virtuel appelé ***Sandbox [Willems, et al., 2007; Egele, et al., 2012; Rieck, et al., 2011b]***. L'exécution d'un programme au sein d'un Sandbox permet d'extraire un ensemble de caractéristiques comportementales appelées dynamiques. Ces caractéristiques sont présentées et formatées dans un rapport d'exécution (**Figure 3-5**).

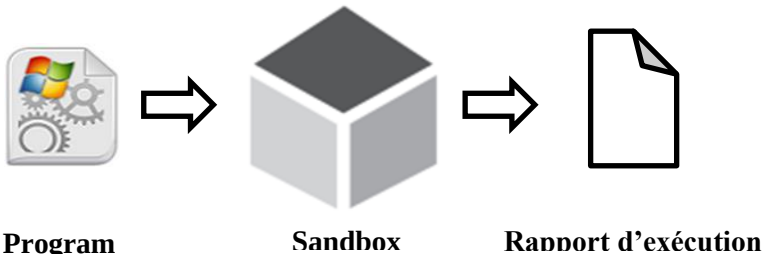


Figure 3-5 Analyse dynamique dans le Sandbox

Dans les sections suivantes, nous passons en revue les types de ces caractéristiques.

5.2.1. Caractéristiques extraites à partir des appels API

Les programmeurs utilisent l'interface de programmation (*API : Application programming interface*) fournie par le système d'exploitation pour accéder aux ressources du système (fichiers, processus, réseau, registre…etc.). Cette interaction indirecte entre les programmes et le système d'exploitation à travers les appelles API offre la possibilité de surveiller le comportement ***[Willems, et al., 2007; Egele, et al., 2012]*** . En fait, L'interception et l'analyse de ces appelles API fournie un outil puissant d'analyse de comportement du virus.

a. N-grammes d'appels de fonctions API

L'extraction de ces n-grammes se fait par le glissement d'une fenêtre de taille **N** sur la séquence d'appels. L'exemple de la **Figure 3-6** illustre le processus avec les 5-grammes extraits à partir un fragment d'une séquence. Ce type de caractéristique est utilisé intensivement dans les travaux à cause de sa simplicité. Ainsi, les résultats expérimentaux des n-gramme d'API sont très encouragent ***[Zhao, et al., 2009; Ahmed, et al., 2009; Eskandari, et al., 2012; Ahmadi, et al., 2013]***.

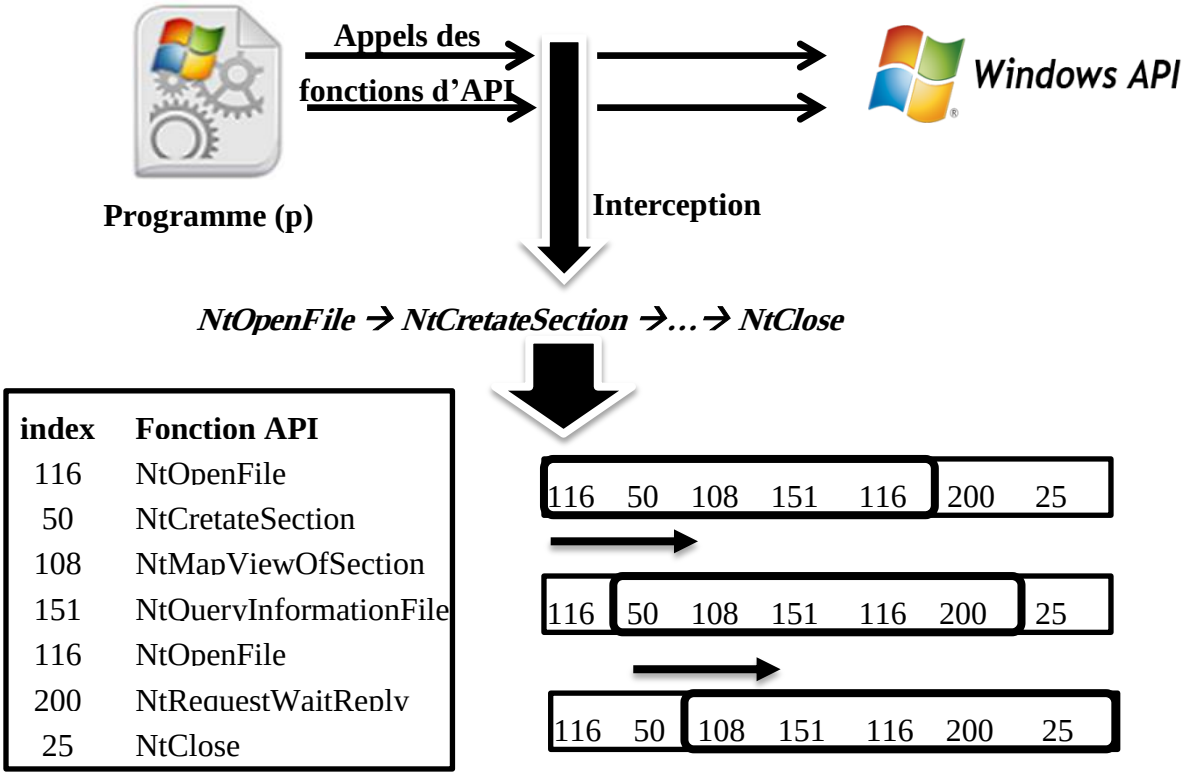


Figure 3-6 Extraction des 5-gramme d'un programme P

D'autres chercheurs ont utilisé des cas spéciaux de n-grammes. A titre d'exemple, dans ***[Tian, et al., 2010; Islam, et al., 2013]*** la fréquence de chaque appel est calculée lors de l'exécution, ce qui est équivalent à l'utilisation des 1-grammes. Également, dans ***[Wagener, et al., 2008]*** la totalité de la séquence d'appel est utilisée comme une caractéristique, ce qui est similaire à des n-grammes longs.

b. Paramètres d'appels de fonctions d'API

Les n-grammes d'appels de fonction d'API capturent l'aspect temporel du comportement d'un programme ***[Ahmed, et al., 2009]***. Cependant, ce type de caractéristiques ne contient pas des informations sur l'impact de chaque appel sur le système d'exploitation. Autrement dit, la même fonction d'API peut être appelée avec différents paramètres. A titre d'exemple, ***LocalAlloc*** est une fonction d'allocation mémoire ayant un paramètre appelé ***uBytes qui*** détermine la taille de la mémoire allouée. La moyenne de ce paramètre dans les virus est plus élevée par rapport à celle de programmes bienveillants ***[Ahmed, et al., 2009]***.

Figure 3-7 Paramètres de l'appel de la fonction API (NtOpenFile)

Pour détecter les taches malveillantes, les chercheurs ont proposé d'exploiter les paramètres de chaque appel d'API (**Figure 3-7**). Tandon et Chan dans ***[Tandon, et al., 2003]*** proposent une liste blanche de paramètres pour chaque fonction API. Durant l'exécution, si un programme fait un appel avec des paramètres qui n'appartient pas à cette liste alors il sera détecté comme un virus. Cependant, Darren Mutz et al dans ***[Mutz, et al., 2006]*** proposent de former un profile pour chaque fonction d'API de manière automatique. Chaque déviation de ce profile, lors d'exécution, est considérée comme une tâche malveillante. Faraz Ahmed et Al ***[Ahmed, et al., 2009]*** suggèrent d'extraire des caractéristiques statistiques sur l'usage de paramètres (moyenne, minimum, maximum, variance). De plus, ils utilisent l'entropie pour quantifier l'information présente au niveau de paramètres. Ils existent d'autres travaux qui font l'agrégation des informations bas niveau, disponible au niveau des paramètres, afin de former des caractéristiques plus abstraites. Par exemple, si les deux fonctions ***CreateFile*** et ***WriteFile*** sont appelées en appelant le même fichier alors nous pouvons regrouper ces deux appels en une seule action ***[Egele, et al., 2012]***. En faisant cette abstraction des appels, plusieurs informations sont délivrées sur le trafic réseau, les clés modifiées de registre, les fichiers système modifiés …etc. ***[Bailey, et al., 2007; Willems, et al., 2007; Bayer, et al., 2009]***. Finalement, dans ***[Canali, et al., 2012]*** qui est une étude comparative qui montre l'efficacité de ce type de caractéristiques par rapport aux autres caractéristiques dynamiques.

Néanmoins, l'exploitation de ce type de caractéristiques peut ajouter une charge supplémentaire sur les systèmes de détection ce qui augmente le temps de réponse.

c. Graphe d'appels API

Afin d'échapper à la détection, les programmeurs de virus réordonnent les appels indépendants ou insèrent des appels inutiles. Ces mesures induisent facilement en erreur les représentations basées sur la séquence comme les n-grammes d'appels ***[Kolbitsch, et al., 2009]***. Pour remédier ce problème, il est nécessaire de combiner les informations temporelles et les paramètres d'appels pour inférer une représentation plus élaborée.

Les chercheurs ont proposé de convertir la séquence d'appels en un graphe tout en capturant les relations entre ces appels. Christodorescu et al. ***[Christodorescu, et al., 2007]*** proposent un graphe acyclique[3] appelé ***Malspec.*** Les nœuds de ce dernier sont des fonctions API et les arcs sont des relations de dépendance de paramètres. Autrement dit, si un appel retourne un paramètre utilisé ensuite par un autre appel alors il existe un arc entre ces deux appels. L'avantage de ce type de représentation réside dans le résumé du comportement dans une spécification abstraite.

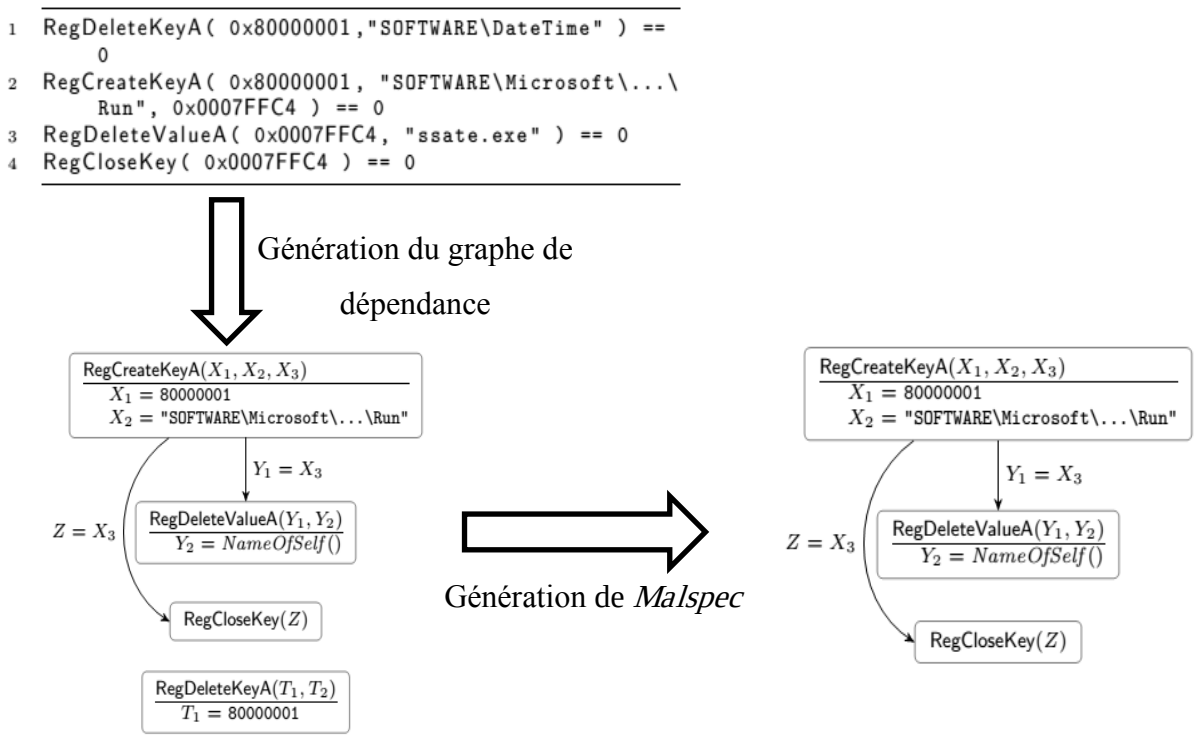


Figure 3-8 Génération de Malspec (Christodorescu, Jha et Kruegel 2007)

[3] Graphe sans cycle

Ainsi, cette représentation est plus robuste contre l'insertion et le réordonnancement d'appels.

Dans l'exemple de la **Figure 3-8**, *RegCreateKey* situé dans la deuxième ligne retourne un paramètre de sortie X_3=0x0007FFC4 utilisé par *RegDeleteValueA* comme un paramètre d'entré. Autrement dit, ce programme crée une clé de registre et la supprime par la suite afin de cacher ces tâches malveillantes ***[Christodorescu, et al., 2007]***. L'appel *RegDeleteValueA* dans la première ligne est éliminé dans le Malspec car elle est indépendante aux autres appels.

Younghee Park et al dans ***[Park, et al., 2010]*** utilisent aussi une représentation similaire à Malspec pour mesurer la similarité entre les virus de la même famille. Alors que, Clemens Kolbitsch et al dans ***[Kolbitsch, et al., 2009]*** ont construit un graphe similaire à Malspec en modifiant l'étiquette de l'arc par le paramètre qui cause l'indépendance (**Figure 3-9**).

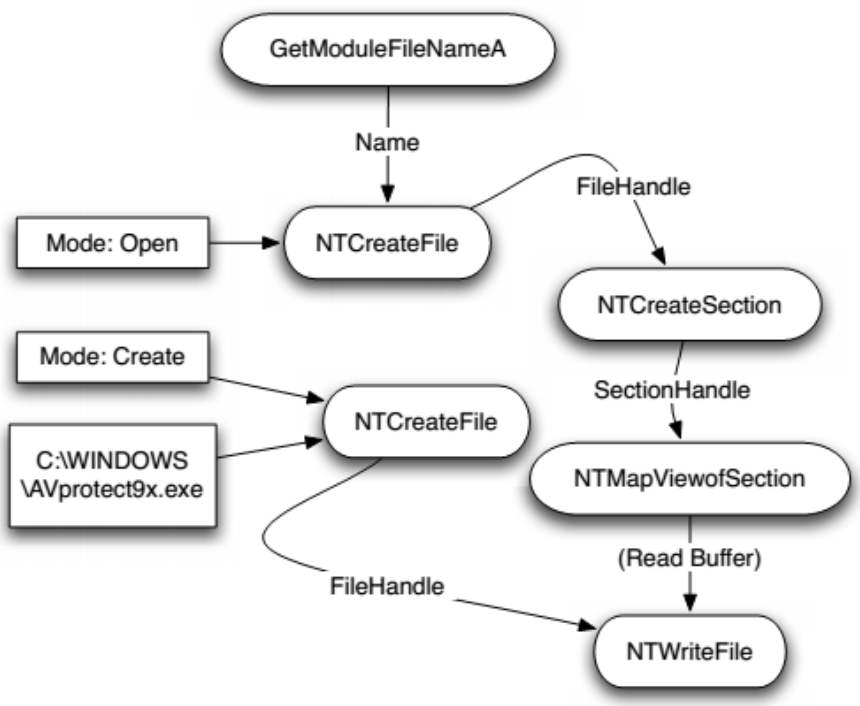
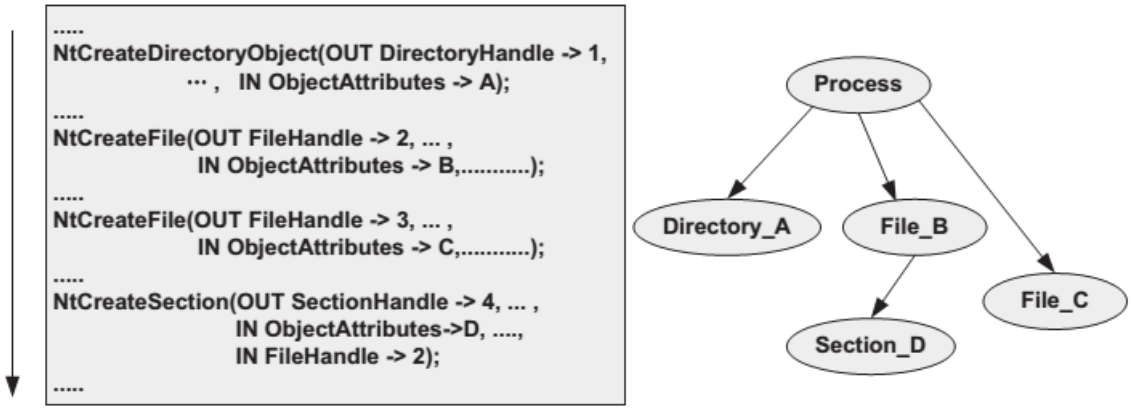


Figure 3-9 Partie du graphe de comportement de Clement Kolbitsch *[Kolbitsch, et al., 2009]*

Une approche récente ***[Park(b), et al., 2013]*** propose la formation du graphe entre les objets de noyau du système au lieu de le former entre les appels. Les objets du noyau (kernel objects) dans Windows peuvent être des processus, des threads, des fichiers, des sockets, etc. Ces objets ne sont pas accessibles directement par les applications. En revanche, l'accès se



Trace d'appels d'API KOBG

Figure 3-10 Partie du graphe de comportement avec les objets de noyau (KOBG) (Park(b), Reeves et Stamp 2013)

fait à travers les appels de fonction d'API. Dans ***[Park(b), et al., 2013]***, ils ont proposé de construire le graphe de comportement avec les objets du noyau (*KOBG :kernel object behavior graph*). Les nœuds de ce graphe sont les objets de noyau et les arcs sont les relations de dépendance induites par les appels (**Figure 3-10)**.

5.2.2. Caractéristiques de processus en cours d'exécution

Ces caractéristiques sont des informations sur le processus du programme en cours d'exécution. Pour les extraire, le système d'exploitation Windows possède le Windows Management Instrumentation (WMI). WMI prend en charge la surveillance et le contrôle des ressources système via un ensemble d'interfaces. YU-FENG LIU et al. ***[Liu, et al., 2010]*** utilisent les informations fournies par WMI sur l'exécution afin de classifier les programmes. Ces informations résument les ressources exploitées par le processus en cours d'exécution (l'utilisation du processeur, l'utilisation de réseau …etc.). Dans le système d'exploitation Lunix, Farrukh Shahzad et al ***[Shahzad, et al., 2013]*** ont extrait 16 caractéristiques du processus à partir son PCB (*PCB : le bloc de contrôle de processus)*. Le PCB est une structure

de données du noyau d'un système d'exploitation représentant l'état d'un processus en cours d'exécution.

5.2.3. Analyse textuelle d'un rapport d'exécution

L'exécution d'un programme dans Sandbox résulte un rapport d'exécution en format texte. En principe, ce rapport est analysé par un expert de sécurité afin de détecter les virus. Cependant, le nombre de rapports étant très grand ce qui rend cette tâche très laborieuse pour ces experts. Par conséquent, l'automatisation de l'analyse de ces rapports est devenue indispensable ***[Rieck, et al., 2011b].***

Konrad Rieck et ses collaborateurs proposent l'analyse des rapports d'exécution de point de vue textuel. En d'autres mots, un rapport est vu comme un vecteur de fréquences de chaines de caractères prédéfini dans un vocabulaire. Cette représentation est caractérisée par le nombre de caractéristiques élevé ce qui produit des vecteurs de caractéristiques ayant une dimension élevée ***[Rieck, et al., 2008a]***.

Les mêmes auteurs proposent une amélioration par la conversion du rapport à un format intermédiaire appelé (MIST : Malware Instruction SeT). Ensuite, les caractéristiques sont extraites en utilisant les n-grammes de chaines de caractères (chaque gramme est une chaine de caractère) ***[Rieck, et al., 2011b]***.

6. Sélection de caractéristiques de programmes

Le domaine de détection de virus informatiques est reconnu par le nombre de caractéristiques élevé. Cette abondance de caractéristiques de programme peut augmenter le temps de réponse du système de détection face aux nouveaux virus ***[Jiang, et al., 2011]***. En outre, la majorité de caractéristiques sont redondantes et pourvues de la capacité de discrimination entre les virus et les programmes bienveillants. A titre d'exemple, le nombre de caractéristiques n-gram d'octet extraites est très élevé, cependant le nombre de n-gram discriminants est très petits ***[Shabtai, et al., 2009; Jiang, et al., 2011]***.

En conséquence, une stratégie de filtrage de caractéristiques s'avère indispensable. Afin de surmonter ce problème, les chercheurs ont proposé plusieurs méthodes de sélection de

caractéristiques pertinentes. Parmi ces derniers on cite : fréquence de document, gain d'information, score de Fisher et la sélection hiérarchique de caractéristiques.

6.1. Fréquence de document

Cette méthode exploite le nombre d'apparitions d'une caractéristique dans les programmes de chaque classe (bienveillant/malveillant). Autrement dit, une caractéristique est sélectionnée si sa fréquence dans une classe dépasse un seuil prédéfini ***[Shabtai, et al., 2009]***.

6.2. Gain d'information

Le gain d'information se base essentiellement sur la mesure de la pureté d'un ensemble quantifié par l'entropie de ce dernier ***[Mitchell, 1997]***. Considérons un ensemble E qui contient des éléments qui ont des classes dans l'ensemble C . L'entropie de E est calculée selon la formule suivante :

$$Entropie(E) = \sum_{c \in C} -\frac{|E_c|}{|E|} log_2 \frac{|E_c|}{|E|}$$

Tel que :

- E_c est un sous-ensemble contient les éléments de E ayant la classe $c \in C$.

Après avoir formulé l'entropie, le gain d'information d'une caractéristique f représente la réduction d'entropie après la division de l'ensemble E selon les valeurs de f.

Le gain d'information est calculé selon la formule suivante :

$$GI(f, E) = Entropie(E) - \sum_{v \in V(f)} \frac{|E_v|}{|E|} Entropie(E_v)$$

Tel que :

- E_v est un sous-ensemble contient les éléments de E ayant une valeur de f égale $v \in V(f)$.

Dans le contexte de détection de virus ***[Kolter, et al., 2006b; Zhang, et al., 2007]***, les deux classes utilisées sont les virus informatiques et les programmes malveillants ($C = \{bienveillant, malveillant\}$).

6.3. Score de Fisher

Cette méthode calcule la différence, décrite en termes de la moyenne et de l'écart type, entre les exemples positifs et négatifs par rapport à une certaine caractéristique ***[Moskovitch, et al., 2008a; Moskovitch, et al., 2008d]***. La moyenne d'une caractéristique de programme est calculée dans la classe des programmes malveillants ($C+ : classe\ postive$) et bienveillants ($C- : classe\ négative$) comme suit:

- $\mu_{f,C-} = \frac{\sum_{p \in C-} f(p)}{|C-|}$
- $\mu_{f,C+} = \frac{\sum_{p \in C+} f(p)}{|C+|}$

Ensuite, les écarts types dans les deux classes $C+$ et $C-$ sont calculés comme suit :

- $\sigma_{f,C-} = \sqrt{\frac{\sum_{p \in C-} \left(f(p) - \mu_{f,C-}\right)^2}{|C-|}}$
- $\sigma_{f,C+} = \sqrt{\frac{\sum_{p \in C+} \left(f(p) - \mu_{f,C+}\right)^2}{|C+|}}$

En fin, le score de Fisher est calculé selon la formule suivante :

$$R_f = \frac{\left|\mu_{f,C-} - \mu_{f,C+}\right|}{\sigma_{f,C-} + \sigma_{f,C+}}$$

Les caractéristiques qui ont un score de Fisher élevé sont sélectionnées pour représenter les programmes. Effectivement, un R_f élevé est synonyme d'une différence entre les valeurs de f dans les exemples positifs et négatifs. Par conséquent, cette caractéristique est pertinente pour discriminer les exemples.

6.4. Sélection hiérarchique de caractéristiques

Cette méthode proposée par Henchiri et Japkowicz dans ***[Henchiri, et al., 2006]*** est divisée en deux phases (**Figure 3-11**). La première phase consiste à sélectionner les caractéristiques qui ont un support dans chaque famille. Tandis que dans la deuxième phase, les caractéristiques ayant un support inter-familles sont éliminées afin de produire l'ensemble final de caractéristiques.

Cette approche tente de générer un ensemble de caractéristiques génériques[4] tout en évitant la domination des familles majoritaires. A titre d'exemple, si une famille avec cent exemples et une autre avec 10 exemples. Dans ce cas, si le nombre d'apparitions global est utilisé, une caractéristique apparait dix fois dans la première famille est éliminée au détriment d'une autre apparait neuf fois dans l'autre famille. Cependant, l'utilisation de la sélection hiérarchique évite ce problème en faisant la phase de sélection dans chaque famille indépendamment.

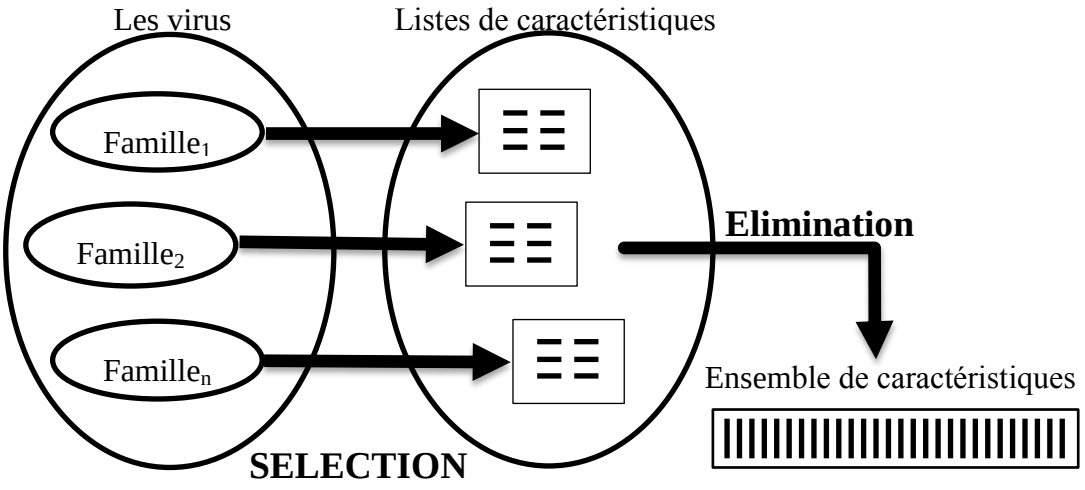


Figure 3-11 Sélection hiérarchique de caractéristiques (Henchiri, et al., 2006)

[4] Caractéristiques qui ne sont pas spécifiques pour une famille

7. Algorithmes de génération du classificateur

Dans la phase d'entrainement, les vecteurs de caractéristiques de programmes sont les entrées des algorithmes de génération d'un classificateur. Il existe plusieurs algorithmes de génération dans la littérature. Ici, nous présentons seulement les méthodes les plus populaires dans le domaine de la détection de virus informatique.

7.1. Classification naïve bayésienne

C'est un classificateur probabiliste qui produit la probabilité conditionnelle d'appartenance de programme p, ayant le vecteur de caractéristiques $v(p)$, à une classe C_i ***[H. George, et al., 1995]***. Afin de calculer cette probabilité $P(C_i|v(p))$, le théorème de Bayes est défini par la formule suivante :

$$P(C_i|v(p)) = \frac{P(C_i) * P(v(p)|C_i)}{P(v(p))}$$

La probabilité $P(v(p))$ représente la probabilité qu'un programme choisi au hasard ait le vecteur $v(p)$ comme représentation, et $P(C_i)$ est la probabilité qu'un programme choisi au hasard appartienne à C_i. Ces deux probabilités sont estimées en utilisant les exemples de l'ensemble d'entrainement ***[Dziczkowski, 2008]***.

Pour faciliter le calcul de $P(v(p)|C_i)$ exige une hypothèse naïve. En effet, les caractéristiques sont considérées statistiquement indépendantes. Par conséquent, la valeur de $P(v(p)|C_i)$ est estimée selon la formule ***[Dziczkowski, 2008]*** :

$$P(v(p)|C_i) = \prod_{j=1}^{m} P(f_j(p)|C_i)$$

- $f_j(p)$ est une caractéristique de programme de rang j dans le vecteur de caractéristiques $v(p)$ définie comme suit : $v(p) = (f_1(p), f_2(p), \dots, f_j(p), \dots, f_m(p))$.

7.2. Réseau bayésien

Les réseaux bayésiens représentent une généralisation du classificateur naïf Bayes en omettant l'hypothèse d'indépendance. Cet algorithme de génération de classificateur essaye de modéliser les relations de causalités entre les caractéristiques en graphe orienté acyclique.

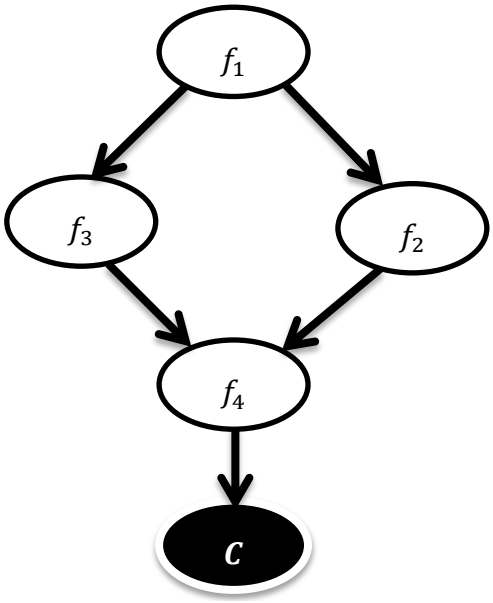


Figure 3-12 Exemple d'un réseau bayésien

Dans ce graphe, les nœuds sont des caractéristiques et les arcs décrits les relations de causalité. Ces relations de cause entre les variables ne sont pas déterministes, mais probabilistes.

La **Figure 3-12** montre un exemple d'un réseau bayésien. La caractéristique f_1 influe sur f_2 et f_3, également, ces deux caractéristiques ont une relation de causalité sur f_4. En fin, $f_4$influe directement sur la réponse du classificateur C.

7.3. Arbre de décision

Un classificateur de texte basé sur la méthode d'arbre de décision est un arbre de nœuds internes qui sont marqués par les caractéristiques, les branches qui sortent des nœuds sont des conditions, et les feuilles sont marquées par les classes possibles. Un exemple est classifié en testant récursivement les valeurs de caractéristiques jusqu'à ce qu'une feuille soit atteinte. Alors, le résultat de classification est l'étiquette de la feuille atteinte lors du parcours de

l'arbre. Un arbre de décision est interprétable facilement grâce à sa structure intuitive ce qui facilite la tâche pour un analyste humain ***[Dziczkowski, 2008]***.

Il existe plusieurs algorithmes de génération de l'arbre à partir de l'ensemble d'entrainement. Les plus populaires sont ID3 ***[Quinlan, 1986]***,C4.5 ***[Quinlan, 1993]*** et **C5** ***[Kohavi, et al., 2002]***.

L'exemple de la **(Figure 3-13)** illustre la structure d'un arbre binaire graphiquement et montre aussi la technique de classification d'un exemple en parcourant l'arbre jusqu'à la feuille $\boldsymbol{C_2}$**.**

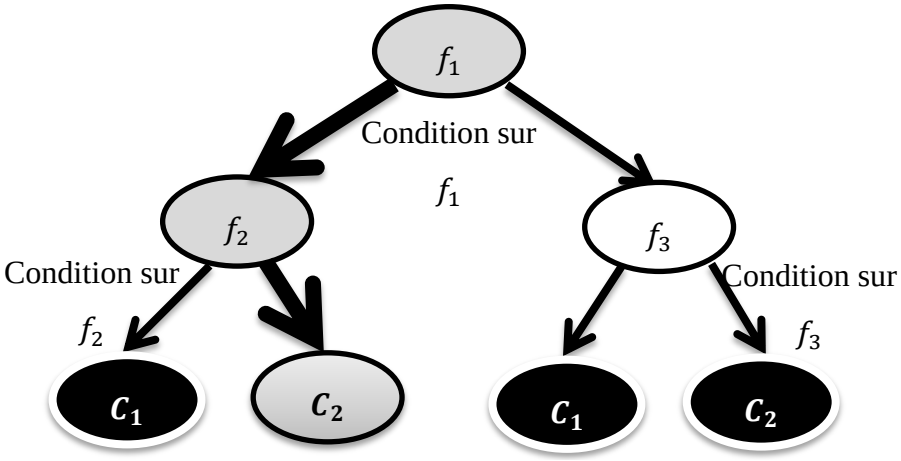


Figure 3-13 Exemple d'un arbre de décision binaire

7.4. Machines à vecteur support (SVM : Support Vector Machine)

Dans cette section, nous présentons les machines à vecteur support de manière plus détaillée que les autres algorithmes de classification, car ils représentent l'outil utilisé par la suite dans notre approche.

La discrimination entre les programmes bienveillants et les virus est un problème de classification binaire. De ce fait, nous évoquons dans cette section la machine à vecteur support binaire avec deux classes. Le principe de base du SVM consiste à chercher à un hyperplan qui sépare le mieux ces deux classes. Si un tel hyperplan existe, c'est-à-dire que les données sont linéairement séparables, on parle d'une machine à vecteur support à marge dure

7.4.1. SVM à marge dure

Concéderons l'ensemble d'entrainement suivant $\{(\boldsymbol{x_1}, \boldsymbol{y_1}), (\boldsymbol{x_2}, \boldsymbol{y_2}), \dots, (\boldsymbol{x_n}, \boldsymbol{y_n})\}$ où $\boldsymbol{x_i} \in \boldsymbol{R^m}$ est un exemple (vecteur de m caractéristiques d'un programme) et $\boldsymbol{y_i} \in \{\mathbf{1}, -\mathbf{1}\}$ est sa classe $(\mathbf{1} : \boldsymbol{bienveillant}, -\mathbf{1} : \boldsymbol{virus})$. L'objectif du SVM consiste à chercher l'hyperplan ayant l'équation $\boldsymbol{w^T \cdot x + b = 0}$ $\boldsymbol{(w, b \in R^m)}$ qui sépare les exemples de deux classes, tout en maximisant la marge entre l'hyperplan et ces exemples (**Figure 3-14**). C'est-à-dire, établir une fonction appelée $\boldsymbol{H(x)}$ vérifiant : $\begin{cases} \boldsymbol{H(x) = w^T \cdot x + b} \\ \boldsymbol{H(x) > 0\ si\ \ classe = 1} \\ \boldsymbol{H(x) < 0\ si\ \ classe = -1} \end{cases}$

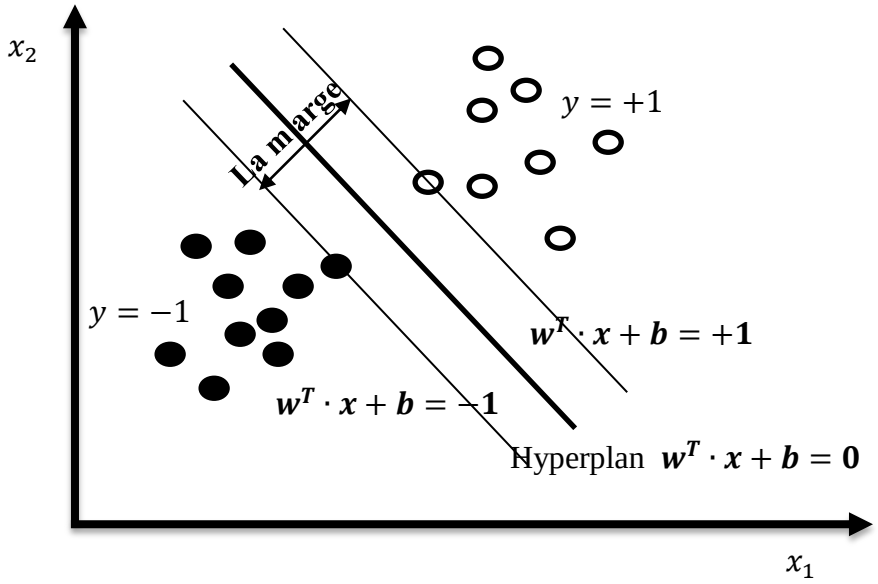


Figure 3-14 SVM à marge dure (dimension deux)

Durant l'entrainement, cette condition doit être vérifiée avec tous les exemples d'entrainement ce qui donne l'équation suivante :

$$\boldsymbol{y_i(w^T \cdot x_i + b) > 1\ , i = 1..n} \quad \text{.....(1)}$$

De plus, nous voulons trouver un hyperplan spécifique qui maximise la distance aux exemples proches appelée la marge. Mathématiquement, la maximisation de la marge est équivalente à la minimisation de la norme euclidienne de w notée $\|w\|$.

Cette minimisation est écrite comme suit :

$$\boldsymbol{Minimiser}\ \frac{1}{2}\|\boldsymbol{w}\|^2 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

En groupant **(1)** et **(2)**, le problème d'entrainement de SVM devient un problème d'optimisation quadratique sous contraintes, formalisé comme suit :

$$\begin{cases} \boldsymbol{Minimiser}\ \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \boldsymbol{sous\ contraintes} \\ \boldsymbol{y_i}(\boldsymbol{w^T}\cdot\boldsymbol{x_i}+\boldsymbol{b})>\boldsymbol{1}\ ,\forall\ \boldsymbol{i}=\boldsymbol{1}..\boldsymbol{n} \end{cases} \ldots\ldots\ldots (3)$$

Le problème de l'équation **(3)** peut être résolu en introduisant les multiplicateurs de Lagrange dans le problème dual suivant :

$$\begin{cases} \boldsymbol{Maximiser}\ \sum_{i=1}^{n}\boldsymbol{\alpha_i}-\frac{\boldsymbol{1}}{\boldsymbol{2}}\sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{\alpha_i\alpha_j y_i y_j {x_i}^T\cdot x_j} \\ \boldsymbol{sous\ contraintes} \\ \sum_{i=1}^{n}\boldsymbol{\alpha_i y_i}=\boldsymbol{0} \\ \boldsymbol{\alpha_i}\geq\boldsymbol{0} \end{cases}$$

La résolution de ce problème d'optimisation fournit la fonction de décision suivante :

$$\boldsymbol{H(x)}=\sum_{i=1}^{n}\boldsymbol{\alpha_i y_i\, {x_i}^T\cdot x+b}$$

Les exemples ayant des $\boldsymbol{\alpha_i \neq 0}$ représentent les vecteurs supports appartenant aux deux Classes.

7.4.2. SVM à marge souple

Le cas de données linéairement séparables est pratiquement rare à cause de la non-linéarité intrinsèque ainsi que la présence du bruit dans les données d'entrainement. Pour résoudre ce problème, le SVM à marge souple est introduit (**Figure 3-15**).

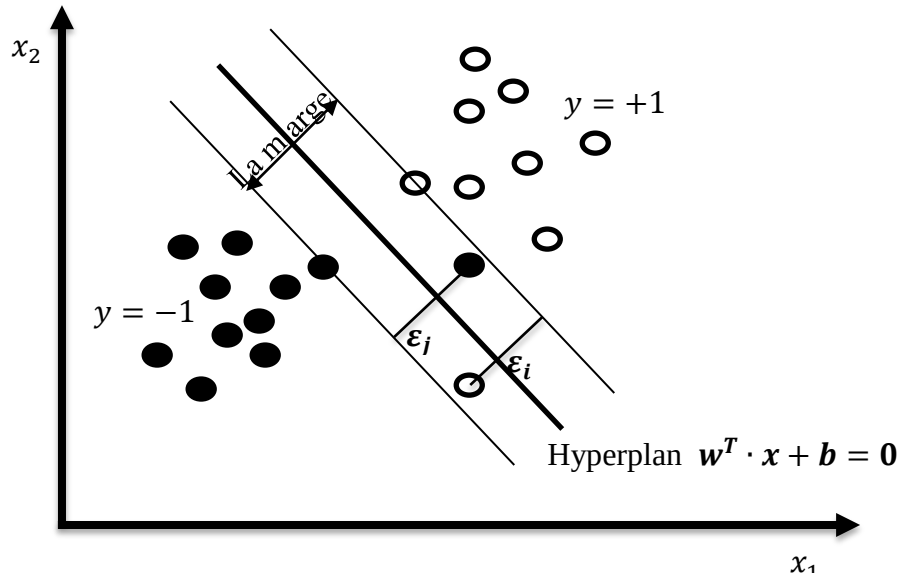


Figure 3-15 SVM à marge souple (dimension deux)

La résolution de SVM à marge souple nécessite l'introduction de variables relaxation $\boldsymbol{\varepsilon_i}$ pour but de relaxer les contraintes exprimées dans **(1)** et les modifiés par **(4)** :

$$\boldsymbol{y_i(w^T \cdot x_i + b) > 1 - \varepsilon_i, i = 1..n} \quad \text{.....(4)}$$

En modifiant le problème d'optimisation (3) en changeant les contraintes par (4), on obtient l'équation (5) suivante :

$$\begin{cases} \boldsymbol{Minimiser\ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\varepsilon_i} \\ \boldsymbol{sous\ contraintes} \\ \quad \boldsymbol{y_i(w^T \cdot x_i + b) > 1 - \varepsilon_i, \forall\, i = 1..n} \\ \quad \boldsymbol{\varepsilon_i > 0, \forall\, i = 1..n} \end{cases} \quad \text{.....(5)}$$

Où C est un paramètre de régularisation qui permet de contrôler le compromis entre le nombre d'erreurs de classement, et la largeur de la marge.

De même, le problème de l'équation **(5)** peut être résolu en introduisant les multiplicateurs de Lagrange α_i dans le problème dual suivant :

$$\begin{cases} \boldsymbol{Maximiser}\ \sum_{i=1}^{n} \boldsymbol{\alpha_i} - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{\alpha_i \alpha_j y_i y_j {x_i}^T \cdot x_j} \\ \boldsymbol{sous\ contraintes} \\ \qquad \sum_{i=1}^{n} \boldsymbol{\alpha_i y_i} = \boldsymbol{0} \\ \qquad \boldsymbol{0 \le \alpha_i \le C} \end{cases} \quad \ldots(6)$$

Après le calcule de multiplicateurs de Lagrange la fonction de décision est similaire à celle présentée dans le cas de la marge dure.

7.4.3. SVM avec le noyau

L'utilisation du SVM avec la marge souple ne peut pas surpasser le problème de non-linéarité dans tous les cas. En effet, la situation illustrée dans (**Figure 3-16**) est un exemple où

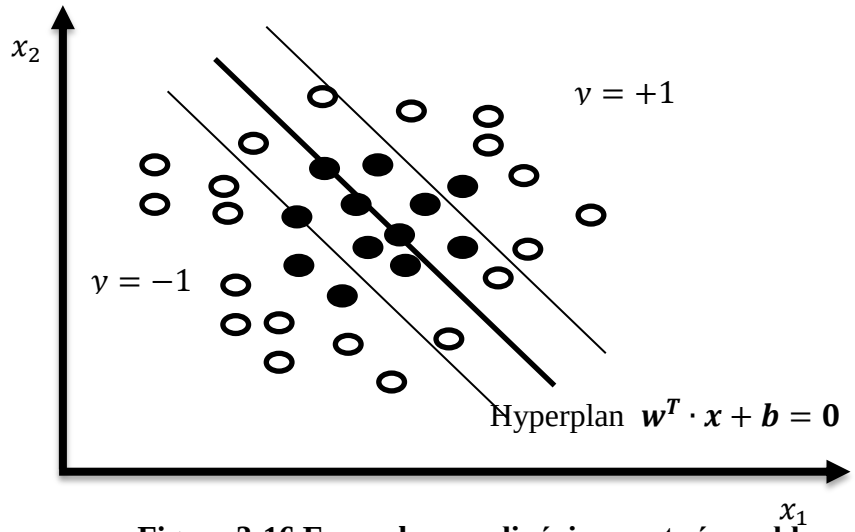


Figure 3-16 Exemples non linéairement séparable

l'utilisation de la marge souple ne fournit pas un classificateur performant, juste en relaxant les contraintes.

Afin de résoudre ce problème de non-linéarité intrinsèque, les exemples doivent être amenés à une espace où ils deviennent linéament séparables. Autrement dit, une transformation d'espace est nécessaire pour appliquer le principe de SVM dans cette espace. L'exemple de (**Figure 3-17**) montre le principe de transformation d'un espace de caractéristiques à une autre qui assure la séparation linéaire.

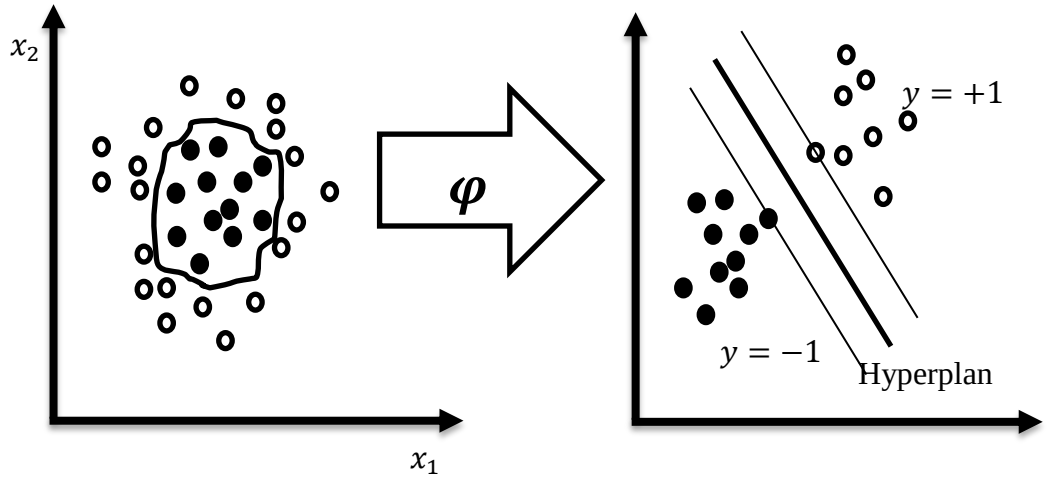


Figure 3-17 Le principe de noyau

Le produit scalaire $\boldsymbol{x_i}^{\boldsymbol{T}} \cdot \boldsymbol{x_j}$ qui représente le produit scalaire $\langle \boldsymbol{x_i}, \boldsymbol{x_j} \rangle$ devient dans la nouvelle espace$\langle \boldsymbol{\varphi}(\boldsymbol{x_i}), \boldsymbol{\varphi}(\boldsymbol{x_j}) \rangle$ ($\boldsymbol{\varphi}(\boldsymbol{x_i})$ est une fonction de transformation à la nouvelle espace). Ce produit scalaire forme une fonction avec deux variables, appelée le noyau et notée :

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \langle \boldsymbol{\varphi}(\boldsymbol{x_i}), \boldsymbol{\varphi}(\boldsymbol{x_j}) \rangle$$

En substituant $\boldsymbol{x_i}^{\boldsymbol{T}} \cdot \boldsymbol{x_j}$ dans (6) par $K(\boldsymbol{x_i}, \boldsymbol{x_j})$, on obtient l'équation suivante :

$$\begin{cases} \boldsymbol{Maximiser} \ \sum_{i=1}^{n} \boldsymbol{\alpha_i} - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \boldsymbol{\alpha_i \alpha_j y_i y_j K}(\boldsymbol{x_i}, \boldsymbol{x_j}) \\ \boldsymbol{sous\ contraintes} \\ \qquad \sum_{i=1}^{n} \boldsymbol{\alpha_i y_i} = \boldsymbol{0} \\ \qquad \boldsymbol{0} \leq \boldsymbol{\alpha_i} \leq \boldsymbol{C} \end{cases} \quad \ldots(7)$$

Et la fonction de décision devient :

$$\boldsymbol{H(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x_j) + b}$$

Dans la littérature, il existe plusieurs modèles du noyau, par exemple :

- Noyau linéaire

$$K(x_i, x_j) = x_i^T \cdot x_j$$

- Noyau polynomial

$$K(x_i, x_j) = (x_i^T \cdot x_j)^d$$

- Noyau gaussien

$$K(x_i, x_j) = e^{\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)}$$

Il faut noter que cette transformation peut nous amener à des espaces de dimension infinie à condition que le produit scalaire dans cette espace soit fini. Autrement dit, la résolution du SVM ne nécessite pas le calcule de $\boldsymbol{\varphi(x_i)}$, néanmoins, il exige seulement le calcul de produit scalaire $\langle \boldsymbol{\varphi(x_i)}, \boldsymbol{\varphi(x_j)} \rangle$. A titre d'exemple, le noyau gaussien est un produit scalaire dans une espace ou le nombre de caractéristiques est infini.

Finalement, après avoir exposé le principe de SVM succinctement, nous pouvons remarquer la base mathématique solide de ce type de classificateur. En plus, avec la technique de relaxation et la transformation non linéaire dit noyau, nous obtenons un algorithme de générations des modèles de décision robuste et capable d'agir avec des données complexes.

8. Conclusion

Ce chapitre expose une taxonomie pour classer les différentes méthodes de détection de virus informatiques basée sur l'apprentissage automatique supervisé (ML) en utilisant des caractéristiques de programmes statiques et dynamique. Pour réaliser ces systèmes, il est nécessaire de représenter les programmes à un classificateur binaire entrainé avec des programmes bien classés préalablement. La représentation de programme consiste à choisir

les caractéristiques puis sélectionner parmi eux celles qui sont pertinentes. Ces programmes représentés généralement sous forme d'un vecteur de caractéristiques seront classifiés par un classificateur généré par des algorithmes d'entrainement.

Après avoir formé une vue globale qui nous permet à proposer notre approche. Dans les chapitres suivants, nous évoquons notre approche de détection de virus informatiques. Ensuite, nous présentons la concrétisation de cette approche ainsi que les résultats expérimentaux.

Chapter 4 : CONCEPTION D'UN SYSTEME DE DETECTION DE VIRUS INFORMATIQUE BASE SUR LE DATAMINING

1. Introduction

Dans ***[Maloof , 2006]***, Maloof a exposé deux approches différentes utilisées pour construire un système de détection des virus informatiques. Dans la première approche, les experts de sécurités programment les systèmes de détection de virus informatiques, nécessitant ainsi l'expression de l'expertise suivant des règles précises. Tandis que la seconde, généralisation automatique de systèmes de détection de virus (VDS) à partir des données, représente l'approche basée sur le datamining et qui ne demande que des données et des objectifs voulus.

La complexité des réseaux informatiques et des systèmes d'exploitation ainsi que l'explosion du nombre des virus informatiques ont incité l'utilisation de datamining, en exploitant l'abondance des données existantes, pour faciliter la tâche de détection de virus informatiques. Dans notre travail, on s'est focalisé sur la conception d'un système de détection de virus informatiques basé sur les techniques de datamining en ajoutant l'acquisition des nouveaux exemples étiquetés et l'incrémentation du modèle de détection.

Dans le reste du chapitre, nous allons développer les trois points suivants :

- Formalisation mathématique pour les systèmes VDS-DM.
- Construction du VDS-DM.
- Formalisation de l'incrémentation d'un VDS-DM.

2. Architecture globale de notre approche

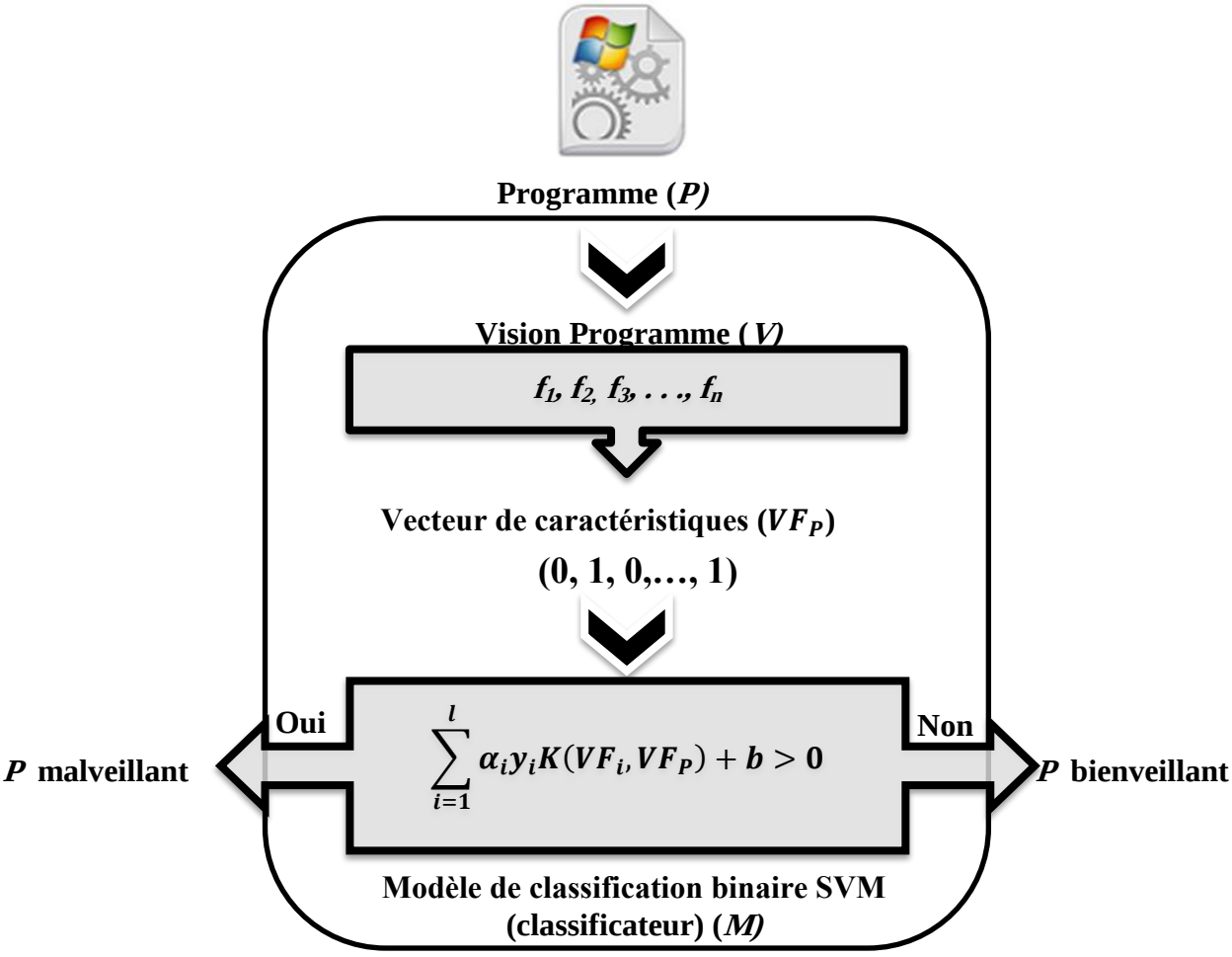


Figure 4-1 Architecture globale VDS-DM

La figure **(Figure 4-1)** rassemble tous les éléments fonctionnels pour un VDS-DM ***[Shabtai, et al., 2009]***.

D'après cette architecture, le VDS-DM commence par la présentation du programme (***P***) comme un vecteur de caractéristiques $\boldsymbol{VF_P}$, à l'aide d'une fonction « **Vision Programme (V)** ». Ensuite, le vecteur $\boldsymbol{VF_P}$ est classifié par un modèle de classification M en deux classes (malveillant, bienveillant). Les composants de cette architecture seront détaillés dans les sections suivantes.

3. Formalisation mathématique de VDS-DM

En cette partie, nous avons proposé une formalisation mathématique constituée de cinq éléments représentant une plateforme pour les différents VDS-DM.

3.1. Caractéristique de programme

Une caractéristique de programme f est une fonction associe chaque programme p de l'ensemble de programmes P l'une des deux valeurs (0 ou 1), indiquant respectivement la présence ou l'absence d'une caractéristique.

On note la fonction :

$$f: P \rightarrow \{0,1\}$$
$$p \rightarrow f(p)$$

Exemples :

1. $f_1(p) = \begin{cases} 1 : si\ le\ fragment\ AF\ 25\ 88\ est\ présent\ dans\ le\ code\ du\ programme\ p. \\ 0 : sinon \end{cases}$

2. $f_2(p) = \begin{cases} 1 : si\ dans\ l'execution\ de\ p\ la\ fonction\ API\ \boldsymbol{CreateFile}^{5}\ est\ appelée. \\ 0 : sinon \end{cases}$

3.2. Vecteur de caractéristiques

Le vecteur de caractéristiques est obtenu lors d'une analyse d'un fichier $p \in P$ par l'utilisation d'un ensemble de caractéristiques de programme. Ce vecteur représente la projection de p dans l'espace $\{0,1\}^n$.

$$VF_p; vecteur\ de\ caractéristiques.$$

Exemple :

Considérant le programme *p* représenté par la chaine hexadécimale suivante AB 0F 12 56 32.

[5] Fonction API Windows permet la Création d'un fichier.

On prend les deux caractéristiques f_1, f_2 telle que :

1. $f_1(p) = \begin{cases} 1 : si\ le\ fragment\ 0F\ 12\ 56\ est\ présent\ dans\ le\ code\ du\ programme\ p. \\ 0 : sinon \end{cases}$

2. $f_2(p) = \begin{cases} 1 : si\ le\ fragment\ 46\ 35\ 11\ est\ présent\ dans\ le\ code\ du\ programme\ p \\ 0 : sinon \end{cases}$

On obtient le vecteur de caractéristiques $VF_p = (\ f_1(p)\ ,\ f_2(p)\ ) = (1\ ,0)$

3.3. Vision Programme (Représentation programme)

La fonction Vision Programme présente un programme sous la forme d'un vecteur de caractéristiques. En d'autres termes, cette fonction, constituée d'un ensemble de caractéristiques de programme, est appelée pour la projection du programme (fichier) dans l'espace de caractéristiques.

On note une vision V:

$$V: P \rightarrow \{0,1\}^n$$
$$p \rightarrow V(p) = (f_1(p), f_2(p), \dots, f_n(p))$$

Telle que : $f_1, f_2, \dots, f_n$ sont les caractéristiques de programme qui composent la vision.

Exemple :

Dans l'exemple précédent, f_1, f_2 forment une vision V telle que :

V(p)=(0,1) : *p* est représenté par la chaine hexadécimale suivante AB 0F 12 56 32.

3.4. Modèle de classification binaire (classificateur)

Le modèle de classification est une fonction qui utilise le vecteur de caractéristiques comme entrée (input) et donne comme sortie (output) un résultat de type booléen. On note un modèle de classification binaire M comme suit :

$$M: \{0,1\}^n \rightarrow \{vrai, faux\}$$
$$VF \rightarrow M(VF) \quad : VF = (f_1(p), f_2(p), \dots, f_n(p))$$

Telle que :

$$M(VF) = \begin{cases} vrai & : si\ VF\ est\ vecteur\ de\ caractéristique\ d'un\ programme\ malvaillent. \\ faux & : sinon. \end{cases}$$

Dans notre travail, nous avons fait appel à la machine à vecteurs de support (SVM) comme un modèle de classification binaire.

$$M(VF) = \begin{cases} vrai & : \sum_{i=1}^{l} \alpha_i y_i K(VF_i, VF) + b > 0 \\ faux & : sinon \end{cases}$$

Telle que :

- $K(*,*): Noyau\ utilisé$
- $\{(VF_1, y_1), (VF_2, y_2), \dots, (VF_l, y_l)\}$: Des exemples étiquetés préalablement (les vecteurs supports).

$$y_i = \begin{cases} 1 & : si\ VF_i\ \ etiqueté\ d'un\ programme\ malvaillent. \\ -1 & : sinon \end{cases}$$

3.5. Système de détection de virus informatique basé sur datamining

Le VDS-DM est une fonction qui reçoit un programme et produit un booléen représentant la malveillance ou la bienveillance du fichier (programme). Sachant que le problème de détection de virus constitue un problème indécidable, cette fonction est limitée par un taux d'exactitude.

Un système de détection de virus informatiques basé sur le datamining VDS-DM est formulé comme suit :

$$\text{VDS} - \text{DM} : P \rightarrow \{vrai, faux\}$$

$$p \rightarrow \text{VDS} - \text{DM(p)}$$

Telle que :

$$\mathrm{VDS-DM(p)} = \begin{cases} vrai \ : si\ p\ est\ un\ programme\ est\ malveillant \\ faux : sinon \end{cases}$$

Les VDS-DM basés sur l'apprentissage supervisé contiennent deux composants principaux :

$$VDS - DM = \{v: Vision, m: modèle\ telle\ que: VDS - DM(p) = m(v(p)) = m \circ v(p)\}$$

VDS-DM peut être formulé aussi comme suit :

$$VDS - DM(p) = m(v(p)) = m(f_1(p), f_2(p), \dots, f_n(p))$$

4. Etapes de construction de VDS-DM

Après la description des différents composants de VDS-DM, les étapes de construction de notre VDS-DM seront présentées en détail par la suite de cette section.

4.1. Elaboration de la vision programme V

Dans notre travail, on s'est basé essentiellement sur des caractéristiques de programme générées automatiquement à partir d'un ensemble de programmes étiquetés « E ». On a opté pour les caractéristiques de programme qui capturent les sous séquences dans une chaine. Cette chaine est une succession des appels API système du programme lors de l'exécution (analyse dynamique). Ce type de caractéristiques est évoqué précédemment dans (**Chapter 3 :5.2.1.a)**.

Afin d'élaborer la vision V, deux modules sont utilisés : *Extraction de caractéristiques des programmes, Sélection de caractéristiques des programmes*. Le premier (*Extraction de caractéristiques de programmes*) a pour rôle l'extraction de toutes les caractéristiques de programme d'un ensemble de programmes étiquetés (E). La figure (**Figure 4-2**) illustre le processus de cette extraction. Le second module (sélection de caractéristiques de programme) se charge de faire la sélection des caractéristiques de programme (générés dans la phase d'extraction de caractéristiques) discriminantes qui seront les composants d'une vision V. La figure (**Figure 4-2**) schématise le processus de sélection de caractéristiques de programme discriminantes.

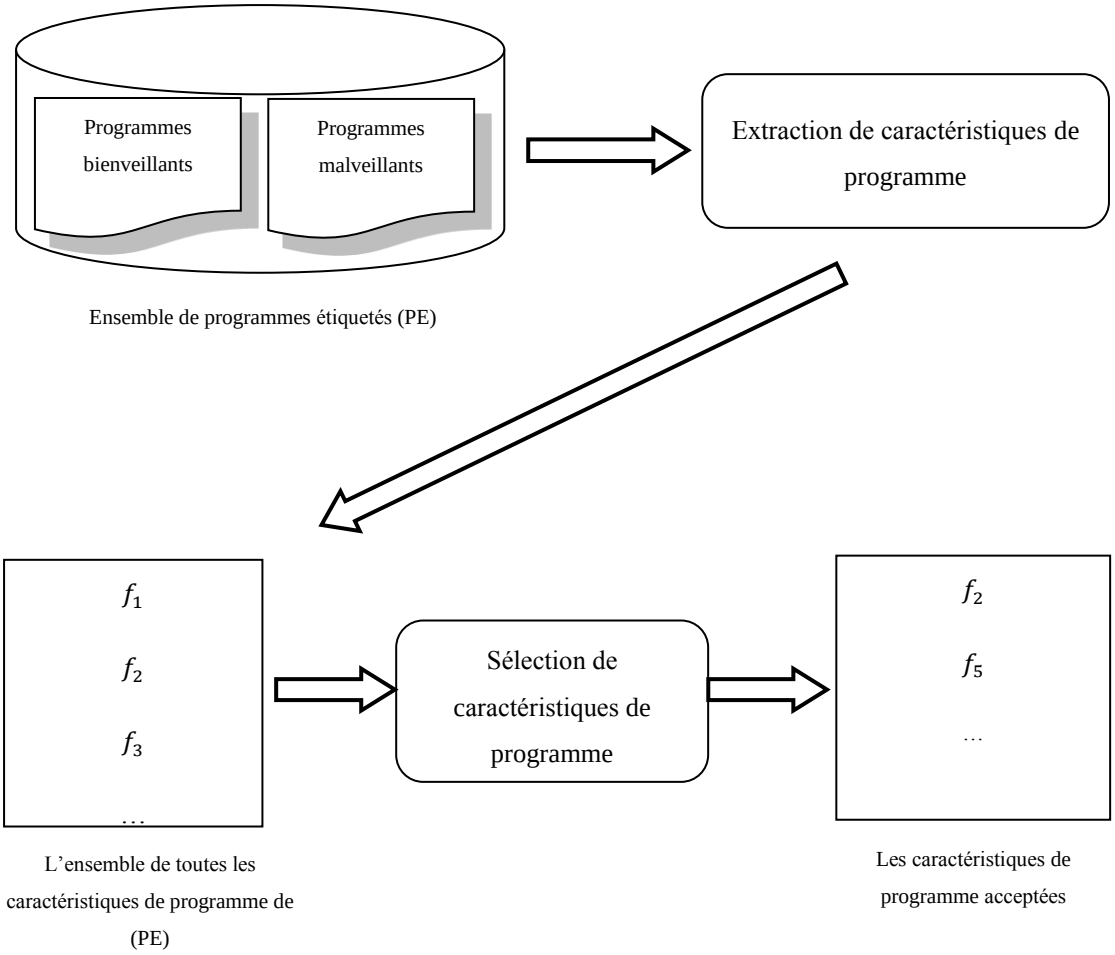


Figure 4-2 Elaboration d'une Vision Programme

4.1.1. Génération des caractéristiques de programme

Le module **Extraction de caractéristiques de programme** est utilisé afin d'extraire toutes les caractéristiques de programme d'un ensemble de programmes étiquetés E pour former une vision V. De ce fait, une fenêtre de taille N gramme est glissée par un gramme sur la séquence des appels API ***[Salehi, et al., 2012]*** de chaque programme de l'ensemble E. Le contenu de cette fenêtre sera utilisé à chaque fois pour produire les différentes caractéristiques de programme dont la taille maximale de la fenêtre est (N).

Exemple 1 : Soit un programme p génère la séquence d'appel API suivante :"NtFreeVirtualMemory" → "LdrGetDllHandle"→ "LdrGetProcedureAddress"→ "NtFreeVirtualMemory" → "NtFreeVirtualMemory".

Si N=2 on obtient l'ensemble de caractéristiques de programme suivant :

FP={"NtFreeVirtualMemory", "NtFreeVirtualMemory" → "LdrGetDllHandle",

"LdrGetDllHandle","LdrGetDllHandle"→ "LdrGetProcedureAddress",

"LdrGetProcedureAddress", "LdrGetProcedureAddress"→ "NtFreeVirtualMemory",

"NtFreeVirtualMemory" , "NtFreeVirtualMemory" → "NtFreeVirtualMemory"}

On associe à chaque caractéristique de programme extraite deux entiers AM et AB. Le premier entier (AM) représente le nombre d'apparitions de caractéristiques de programme dans les programmes malveillants, tandis que le deuxième (AB) représente le nombre d'apparitions de ce dernier dans les programmes bienveillants.

Le principe de l'extraction de toutes les caractéristiques de programme de taille inférieure à N est présenté par les deux algorithmes (**Figure 4-3, Figure 4-4**). Les structures de donnée utilisées dans les deux algorithmes sont :

- La caractéristique de programme *f* représente une structure qui encapsule:
  - Sous séquence s : tableau d'une dimension.
  - AB : Entier pour compter les fichiers bienveillants contenant la sous-séquence s.
  - AM: Entier pour compter les fichiers malveillants contenant la sous-séquence s.
  - IG : Réel représente le gain d'information de la sous séquences s.
- La Vision V : Une liste contient toutes les caractéristiques de programme.

```
        Extraction_Caractéristiques_Programme_Séquence
            (IdProgramme,séquence,N,Etiquette,V)
{
 Pour chaque sous-séquence (s) en séquence avec |s|<=N
      {
      /* Ajouter la caractéristique de programme f
         à la vision V  Si f n'existe pas déjà.
         Sinon mettre à jour AM et AB */
      f.s = s;
      f.IdProgramme = IdProgramme;
      Si(f.s existe déjà en V dans la position index)
           {
           /* L'apparition multiple d'une sous-séquence
            dans la même séquence n'est pas considérée.
           */
            Si (V[index].IdProgramme != IdProgramme)
                {
                 Si (Etiquette=="malveillant")
                    V[index].AM++;
                 Sinon
                    V[index].AB++;
                }
           }
      Sinon
           {
           Si (Etiquette == "malveillant")
               f.AM = 1;f.AB = 0;
           Sinon
               f.AB = 1;f.AB = 0;
           V = V∪{f};
           }
      }
}
```

Figure 4-3 Extraction de toutes les caractéristiques de programme d'une seule séquence programme.

```
        Extraction de caractéristiques de programme
            (N : taille maximale de la fenêtre)
{
   /* IdProgramme : identificateur programme */
   IdProgramme=0;
   /* V : Vison programme.*/
   V = ∅;

   /*
    * Une Séquence est :
    * Une succession d'appels API système du programme
    *                 (Analyse dynamique).
    */

    /* Extraction des séquences des programmes étiquetés
       malveillants */
   Pour toute (séquence) d'un programme p ∈ E étiqueté
   malveillant
      {
       Extraction_Caractéristiques_Programme_Séquence
         (IdProgramme,séquence,N,"malveillant",V)
       IdProgramme ++;
      }
   /* Extraction des séquences de programmes étiquetés
      malveillants */
   Pour toute (séquence) d'un programme p ∈ E étiquetés
   bienveillant
      {
       Extraction_Caractéristiques_Programme_Séquence
         (IdProgramme,séquence,N,"bienveillants",V)
       IdProgramme ++;
      }
}
```

Figure 4-4 Extraction de toutes les caractéristiques de programme de E étiquetés (Malveillants/ Bienveillants).

4.1.2. Sélection de caractéristiques de programmes

Si toutes les caractéristiques de programme extraites à partir de l'ensemble E (ensemble de programmes étiquetés bienveillants/malveillants) sont utilisées, le nombre (par conséquent la taille) de ces caractéristiques de programme risque d'être très grand, ce qui augmente la durée

du calcul et dégrade l'exactitude des résultats. De plus, les caractéristiques de programme qui apparaissent dans la plupart de programmes ne sont pas capables de faire la discrimination parce que la plupart des programmes contiennent ce type de caractéristiques de programme ***[Jiang, et al., 2011; Moonsamy, et al., 2012]***.

Afin de ne garder que les caractéristiques de programme les plus représentatifs, une stratégie de sélection de caractéristiques de programme s'avère être indispensable. Cette stratégie est appliquée par le module (*sélection de caractéristiques de programme*) (**Figure 4-5**). Dans ce module, une caractéristique de programme est sélectionnée si son gain d'information (IG) dépasse un seuil donné. Le gain d'information (IG) d'une caractéristique de programme représentant son degré de discrimination (entre malveillant et bienveillant).

Ce processus est décrit par (**Figure 4-5**), dans lequel **`NbrCaracterstiques`** est un paramètre qui représente le nombre des caractéristiques de programme acceptées. Le gain d'information d'une caractéristique de programme X dans une base de caractéristiques de programme C est défini par la formule suivante ***[Wang, et al., 2010; Jiang, et al., 2011]*** :

$$IG(X) = \sum_{x\in\{0,1\}\ C\in\{Cv,Cb\}} P(X = x \cap C = c).log_2 \frac{P(X = x \cap C = c)}{P(X = x).P(C = c)}$$

Tel que :

- $P(X = 1 \cap C = Cb)$: la probabilité qu'un fichier bienveillant contienne la caractéristique de programme X.
- $P(X = 1 \cap C = Cv)$: la probabilité qu'un fichier malveillant contienne la caractéristique de programme X.
- $P(X = 0 \cap C = Cb)$: la probabilité qu'un fichier bienveillant ne contienne pas la caractéristique de programme X.
- $P(X = 0 \cap C = Cv)$: la probabilité qu'un fichier malveillant ne contienne pas la caractéristique de programme X.
- $P(X = 1), P(X = 0)$: La probabilité d'existence (respectivement d'inexistence) de la caractéristique de programme X dans tous les fichiers.

- $P(C = C_b), P(C = C_v)$: La probabilité qu'un fichier soit bienveillant (respectivement malveillant).

Dans notre échantillon de fichiers de l'ensemble E étiquetés, les valeurs de probabilités précédentes sont estimées, en utilisant les deux nombres *f*.AB et *f*.AM. Sachant que les fichiers de l'ensemble E se divisent en deux sous-ensembles :

- E_M : ensemble de fichiers étiquetés malveillants.
- E_B : ensemble de fichiers étiquetés bienveillants.

Dans le but de calculer le gain d'information IG, les probabilités sont calculées comme suit :

$P(f = 1 \cap C = Cb) \approx \frac{f.ab}{|E|}$, $P(f = 1 \cap C = Cv) \approx \frac{f.am}{|E|}$.

$P(f = 0 \cap C = Cb) \approx \frac{|E_b| - f.ab}{|E|}$, $P(f = 0 \cap C = Cv) \approx \frac{|E_m| - f.am}{|E|}$

$P(f = 1) \approx \frac{f.am + f.ab}{|E|}$, $P(f = 0) = 1 - P(f = 1)$

$P(C = c_b) \approx \frac{|E_b|}{|E|}$, $P(C = c_v) \approx \frac{|E_m|}{|E|}$

La méthode de calcul est montrée par les exemples numériques suivants :

Exemples :

1. |E|= 1000, | E_m | = 500, | E_b | = 500, *f*.AB=100,*f*.AM=100

IG(f) = $\left(\frac{100}{1000} * log_2\left(\frac{\frac{100}{1000}}{\frac{200}{1000}*\frac{500}{1000}}\right)\right) * 2 + (\frac{400}{1000} * log_2\left(\frac{\frac{400}{1000}}{\frac{800}{1000}*\frac{500}{1000}}\right)) * 2 = 0$

IG(f)= 0

2. |E|= 1000, | E_m | = 500, | E_b | = 500, *f*.AB=0,*f*.AM=500

IG(f)= 0,311

3. |E|= 1000, | E_m | = 500, | E_b | = 500, *f*.AB=0,*f*.AM=900

IG(f)=0.758

4. |E|= 1000, | E_m | = 500, | E_b | = 500, *f*.AB=1000,*f*.AM=0

IG(f)=1

D'après les exemples ci-dessus, les caractéristiques *discriminantes* ont un gain d'information plus élevé.

```
sélection de caractéristiques de
programme(NbrCaracterstiques,V:Vision)
{
    Pour toute (f) en V
        {
        /* Calcul du gain d'information de f */
        f.IG = IG(f.AB,f.AM,NombreFichiers)
        }
    Trier V par IG ;
    Sélectionner NbrCaracterstiques caractéristique de
    programme  qui ont IG grand et supprimer les autres
    caractéristiques;
}
```

Figure 4-5 Sélection de caractéristiques de programme.

4.1.3. Présentation des programmes par une vision V

Une vision V est définie comme suit:

$$V: P \rightarrow \{0,1\}^n$$
$$p \rightarrow V(p) = (f_1(p), f_2(p), \dots, f_n(p))$$

Telle que :

$$f_i: P \rightarrow \{0,1\}$$
$$p \rightarrow f(p) = f_i(p)$$

$$f_i(p) = \begin{cases} 1 : si\ le\ sous - séquence\ f_i.s\ est\ présente\ la\ séquence\ du\ programme. \\ 0 : sinon \end{cases}$$

4.2. Elaboration d'un modèle de classification M

Dans l'étape précédente, on a formé une vision V capable de représenter les programmes comme étant des vecteurs de caractéristiques. A ce stade, on peut se baser sur le processus d'apprentissage supervisé afin de produire le modèle en question (M). Bien entendu, l'apprentissage supervisé nécessite un ensemble des exemples étiquetés ***[Shabtai, et al., 2009]***.

La génération du modèle (M) commence par la division de l'ensemble de ces exemples étiquetés en deux sous-ensembles (d'entrainement $E_{training}$ et de teste E_{test}). Le sous-ensemble d'entrainement $E_{training}$ est utilisé pour la construction du modèle. Le sous-ensemble E_{test} est appelé pour mesurer, avec des métriques standards, le pouvoir de généralisation du modèle (M). Finalement, on obtient un modèle M capable à classifier les vecteurs de caractéristiques en deux classes (malveillant/bienveillant) ***[Shabtai, et al., 2009]***.

Dans cette étude, nous avons opté pour le classifieur Machine à vecteurs de support (SVM) en utilisant l'algorithme d'entrainement LIBSVM. Ce choix est motivé par son background mathématique solide. En plus, l'implémentation de l'acquisition de nouveaux exemples étiquetés et l'incrémentation du modèle sont faisables.

4.3. Elaboration d'un VDS-DM

Le VDS-DM est élaboré en utilisant les deux composants (vision V ; modèle M) pour classifier les programmes (malveillant et bienveillant) suivant la formule :

$$VDS - DM(p) = m\big(V(p)\big) = M(f_1(p), f_2(p), \dots, f_n(p))$$

Telle que :

$$\mathrm{VDS} - \mathrm{DM(p)} = \begin{cases} vrai \ : p \ est \ un \ programme \ est \ malveillant \\ faux : p \ est \ un \ programme \ est \ bienveillant \end{cases}$$

5. Formalisation de l'incrémentation d'un système de détection de virus

Notre contribution principale est de rendre le VDS-DM incrémental, basé sur l'apprentissage incrémental et l'apprentissage actif, ce qui offert :

- La possibilité de la mise à jour du VDS-DM.
- L'acquisition des nouveaux virus est possible.
- L'interaction avec un expert distant est possible (Expert humain ou un autre VDS).
- Un nombre d'exemples d'entrainement réduit.

5.1. Apprentissage incrémental/actif

Dans l'apprentissage supervisé, la totalité de l'ensemble d'entrainement doit être disponible dès le début. Néanmoins, l'apprentissage incrémental peut produire un modèle de décision à partir d'un nombre limité d'exemples d'entrainement tout en laissant la possibilité d'acquérir de nouveaux exemples étiquetés ***[Ruping, 2001; Tong, et al., 2002]***.

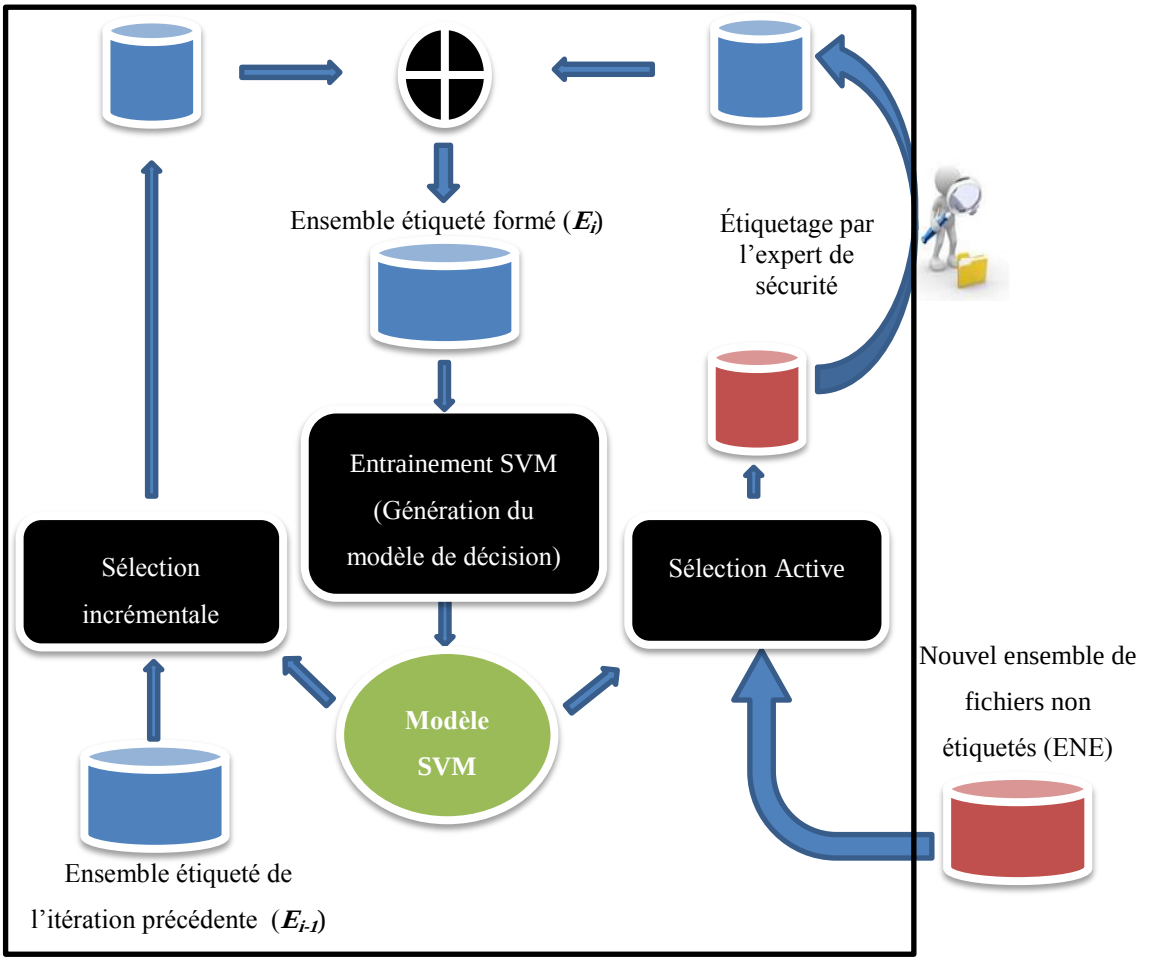


Figure 4-6 Système de détection de virus basé sur l'apprentissage incrémental/actif

Dans l'apprentissage supervisé, l'apprenant (modèle) est une entité passive ne participe pas à l'entrainement. Par contre, en apprentissage actif l'apprenant peut influer sur le choix des exemples d'entrainement par la sélection des exemples qui améliore la précision du modèle pour l'étiquetage ***[Peng, et al., 2002]***.

Pour incrémenter notre VDS-DM, le modèle de classification M est incrémenté à chaque acquisition des fichiers étiquetés et présentés par une vision V. On essaye de tirer profit de deux techniques d'apprentissage citées (incrémental et actif), dont notre algorithme proposé est noté **apprentissage incrémental/actif (Figure 4-7).**

On s'est basé sur la machine à vecteur support (SVM) afin de produire M. Par conséquent, l'apprentissage incrémental/actif doit être implémenté sur SVM.

5.2. Algorithme d'apprentissage incrémental/actif proposé

L'algorithme présenté en (**Figure 4-7**) récapitule notre approche :

```
//initialisation de l'entrainement
M0  = EntrainementSVM(E0)
//Boucle d'apprentissage incrémental/actif
i=1;
Tant que (vrai)
     {
          Si un nouvel ensemble des exemples non étiquetés ENE
               {
                    E1  = SelectionIncrementale(E(i-1),M(i-1))
                    E2* = SelectionActive(ENE,M(i-1))
                    E2  = EtiquetageParEntitéExterne(E2*)
                          Ei = E1 U E2
                    Mi  = EntrainementSVM(Ei)
                    i++;
               }
     }
```

Figure 4-7 Algorithme d'apprentissage incrémental/actif

L'exécution de l'algorithme en boucle infinie permet d'acquérir les exemples incessamment. De ce fait, l'algorithme se déclenche à chaque réception d'exemples en incrémentant le modèle M (**Figure 4-6**).

Les fonctions utilisées dans cet algorithme sont :

5.2.1. Entrainement SVM

C'est une fonction d'entrainement standard de SVM qui a comme entré un ensemble des exemples étiquetés et produits un modèle de classification M.

5.2.2. Sélection Incrémentale

C'est une fonction de sélection pour les exemples déjà étiquetés dans les itérations précédentes. On ne prend pas la totalité des exemples anciens étiquetés, néanmoins, on sélectionne des représentants qui ont une influence sur la formation du modèle. Cependant, il existe plusieurs heuristiques conçues pour sélectionner ces représentants. Parmi ces heuristiques on a utilisé les vecteurs supports à inclure dans les itérations suivantes ***[Ruping, 2001]***. On utilise aussi des exemples de révision, c'est-à-dire les exemples classés incorrectement.

En résumé, les exemples anciens déjà étiquetés sélectionnés pour l'incrémentation sont :

1. Les vecteurs supports.
2. Les exemples de révision,

C'est-à-dire :

$$y * (\sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b) \leq 1$$

Tel que:

- x : vecteur de caractéristiques d'un exemple étiqueté.
- y : l'étiquette réelle de l'exemple.
- $K(*,*)$: $Noyau\ utilisé$
- $\{(x_1, y_1), (x_2, y_2), \dots, (x_l, y_l)\}$: les vecteurs supports.

5.2.3. Sélection Active

C'est une fonction qui sélectionne les exemples à étiqueter à partir d'un ensemble d'exemples non étiquetés. Son objective est de demander à une entité externe (expert de sécurité par exemple) l'étiquetage d'un minimum d'exemples sans perdre la précision du modèle résultant. La participation du modèle, en sélectionnant les exemples pertinents, dans le processus d'apprentissage représente l'avantage majeur de l'utilisation de l'apprentissage actif ***[Tong, et al., 2002]***.

Pour l'implémentation de l'apprentissage actif, on fait appel aux deux heuristiques suivantes :

- L'entropie de voisinage d'un exemple non étiqueté ***[Wang, et al., 2010]***.
- La distance entre la frontière de décision du SVM et l'exemple non étiqueté ***[Tong, et al., 2002]***.

Remarque :

On se limite seulement aux techniques d'apprentissage incrémental et actif applicables avec le classifier SVM.

a. Sélection active basée sur l'entropie de voisinage

Cette heuristique, proposée par Wang et ces collaborateurs ***[Wang, et al., 2010]***, vise à sélectionner les exemples qui ont une incertitude afin de demander leur étiquette. Pour mesurer cette incertitude, l'entropie des étiquettes des exemples situant dans le voisinage d'un exemple non étiqueté est utilisée.

Cette méthode se base essentiellement sur deux concepts:

- **L'entropie d'un ensemble des exemples :**

L'entropie représente une mesure de la pureté d'un ensemble quelconque. Elle est déjà utilisée dans la formation d'un arbre de décision en mesurant la pureté des nœuds produits après une division par un attribut ***[Bramer, 2007]***.

Considérant un ensemble S avec K étiquettes possibles, l'entropie est donnée par la formule suivante :

$$\mathbf{Entropie(S)} = -\sum_{\mathbf{i=1}}^{\mathbf{K}} \mathbf{p_i log_2(p_i)}$$

Tel que : $\mathbf{p_i}$ est la probabilité qu'un exemple de l'ensemble S est étiqueté par l'étiquette i. Dans la pratique, cette probabilité est estimée en calculant le taux des exemples de chaque étiquette dans des enchantions.

Une valeur élevée de l'entropie reflète l'impureté de l'ensemble S et aussi l'incertitude de l'étiquette d'un exemple de *S*.

- **Le voisinage d'un exemple non étiqueté :**

On peut définie le voisinage d'un exemple par les exemples qui situent proche de lui dans un espace R^n. Formellement :

$$\boldsymbol{voisnage(X) = \{x\ exemble\ etiqueté \in R^n : \|X - x\| < r\}}$$

Tel que :

- $\|\ .\ \|$: une distance dans R^n.
- r : le rayon du voisinage.

L'utilisation des deux concepts ensembles : entropie et voisinage, nous a permet d'obtenir une méthode de la sélection active.

Exemple :

L'exemple suivant (**Figure 4-8**) montre la technique dans le cas de détection des virus. Il y a deux étiquettes possibles $K = \{vrai, faux\}$.

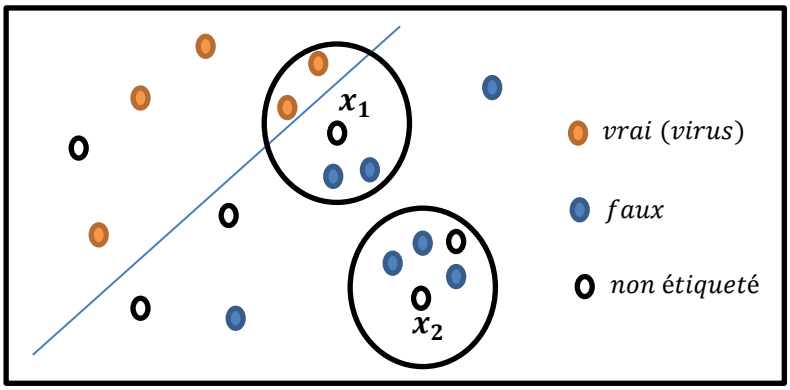


Figure 4-8 Sélection active basée sur l'entropie

$$Entropie(x,r) = -\left(p_{vrai} * log_2(p_{vrai}) + p_{faux} * log_2(p_{faux})\right)$$

Tel que :

- p_{vrai} : taux des exemples étiquetés vrai (virus) dans le voisinage du rayon r.
- p_{vrai} : taux des exemples étiquetés faux dans le voisinage du rayon r.

$$Entropie(x_1,r) = -\left(\frac{2}{2+2} * log_2\left(\frac{2}{2+2}\right) + \frac{2}{2+2} * log_2\left(\frac{2}{2+2}\right)\right) = 1$$

$$Entropie(x_2,r) = -\left(\frac{3}{1+2} * log_2\left(\frac{3}{1+2}\right)\right) = 0$$

L'entropie de x_1 est supérieur à $Entropie(x_2,r)$ car x_1 se situe dans la zone de l'incertitude (frontière entre les deux classes). De ce fait, l'exemple x_1 est sélectionné pour l'étiquetage.

Finalement, l'algorithme de sélection active basée sur l'entropie a trois paramètres :

- ENE : ensemble des exemples non étiquetés
- r : Rayon de voisinage
- NbrRequettes : nombre des exemples demandés pour l'étiquetage par une entité externe.

L'algorithme (**Figure 4-9**) suivant récapitule les étapes :

```
SélectionActive (ENE,r,NbrRequettes < |ENE|)
{
   1. Pour chaque  x ∈ ENE calculer Entropie(x,r).
   2. Trier l'ensemble ENE selon l'entropie.
   3. Sélectionné NbrRequettes qui ont une entropie
      grande pour l'étiquetage.
}
```

Figure 4-9 Sélection Active basée sur l'entropie

b. Sélection active basée sur la distance à un modèle SVM

Dans le travail de Tong et al. ***[Tong, et al., 2002]***, il a été démontré que les exemples qui sont proches au modèle de décision (modèle de l'itération précédente) contribuent étroitement dans la formation d'un modèle SVM pour l'itération suivante. Sami et al. ***[Sami, et al., 2010]*** ont proposé une gamme des heuristiques basées sur la distance de l'exemple (marge) au modèle de décision SVM.

Nous avons utilisé dans cette étude la méthode « marge simple » basée sur la distance de l'exemple au modèle SVM. Pratiquement, cette distance est calculée à travers le modèle comme suit ***[Peng, et al., 2002]*** :

$$Distance(x, M) = \left| \sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b \right|$$

Tel que :

- *M* représente le modèle de classification de l'itération précédente.

Exemple :

L'exemple suivant (**Figure 4-10**) montre la technique dans le cas de détection des virus.

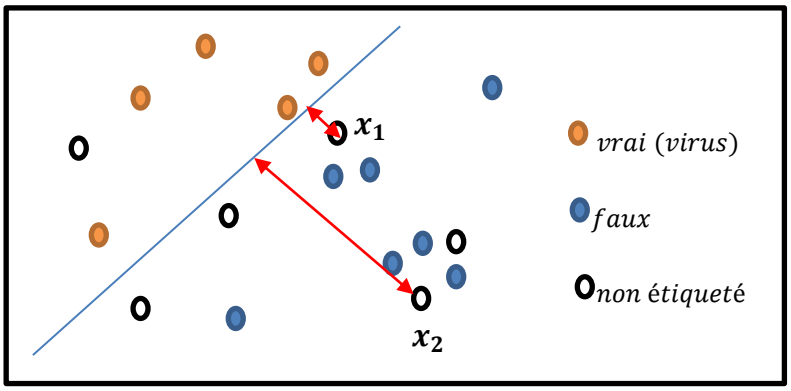


Figure 4-10 Technique de la marge simple.

Dans (**Figure 4-10**) il apparait clairement que x_1 est plus proche à la droite de décision résultant du modèle M que x_2 .

Remarque :

L'exemple illustre l'application de la technique pour le cas d'un noyau linéaire, mais elle reste valide pour les autres noyaux en raisonnant en plan transformé.

L'algorithme de cette heuristique est donné dans (**Figure 4-11**)

```
SélectionActive (ENE,NbrRequettes<|ENE|,M)
{
   1. Pour chaque  x ∈ ENE calculerDistance(x,M).
   2. Trier l'ensemble ENE selon les distances.
   3. Sélectionné NbrRequettes qui ont une distance
      petite pour l'étiquetage.
}
```

.

Figure 4-11 Sélection active

5.2.4. EtiquetageParEntitéExterne

Cette fonction est responsable de l'étiquetage d'un exemple ou programme. Elle peut faire une requête à un expert de sécurité ou un autre système de détection de virus (ou antivirus).

5.3. Formation du modèle incrémenté

Après avoir formé les deux ensembles des exemples étiquetés E_1 et E_2, on les fusionne dans un seul ensemble d'entrainement E_i de l'itération i. On fait appel à la fonction **EntrainementSVM** afin de produire le modèle incrémenté, M (**Figure 4-6**).

Pour la détection de virus, l'ancien modèle est utilisé jusqu'à la formation du nouveau modèle. Autrement, l'incrémentation et la détection (utilisation du modèle) peuvent être déroulées parallèlement.

6. Conclusion

Nous nous somme focalisé dans le chapitre présent sur le point de vue conceptuel. En effet, nous avons proposé une approche afin d'améliorer les systèmes de détection qui utilisent le datamining. Cette approche est une plateforme de détection caractérisée par l'évolutivité ainsi que l'optimisation du nombre de programmes étiquetés.

Après avoir présenté notre approche, nous allons présenter sa concrétisation ainsi que les résultats expérimentaux pour mesurer son apport.

Chapter 5 : IMPLEMENTATION ET RESULTATS EXPERIMENTAUX

1. Introduction

Dans une démarche scientifique rigoureuse, chaque proposition doit être évaluée empiriquement et comparée avec l'état de l'art. A l'avenant, pour mener notre étude rigoureusement, il s'avère indispensable d'implémenter notre approche tout en effectuant une série de tests expérimentaux.

Dans le chapitre précèdent, nous avons présenté notre approche de détection de virus informatiques en citant nos contributions par rapport à l'état de l'art. Afin de mesurer l'apport de notre proposition, dans ce chapitre, nous nous sommes focalisés sur la description de l'implémentation de notre système en justifiant nos choix techniques. Ainsi, nous allons présenter une série de tests visant les objectifs suivants:

- Comparaison de notre algorithme avec l'échantillonnage aléatoire pour montrer son utilité.
- Comparaison de notre algorithme avec l'apprentissage supervisé standard.
- Mesure de l'influence du nombre d'exemples sur la performance du VD-SDM.

2. Environnement d'implémentation et de test

Toutes les expérimentations ont été exécutées et réalisées à l'aide d'un ordinateur muni d'un processeur de type Intel (R) Core (TM) i7-2670QM CPU @ 2.20GHz 2.20 GHz avec une mémoire RAM de taille 8 Go. Concernant le système d'exploitation, nous avons utilisé deux systèmes d'exploitation :

- **Linux Mint :** Nous avons utilisé sa version 14.1 (64 bits) GNU/Linux basée sur Ubuntu avec environnement du bureau **MATE.** Nous avons choisi ce système lors de l'exécution des virus pour des raisons de sécurité. En effet, l'implémentation d'un labo d'analyse de comportement au sein de Mint permet d'éviter l'exécution non intentionnelle des virus de Windows[6].
- **Windows 7 :** Dans la phase d'analyse de données nous avons utilisé Windows 7 édition familiale premium.

3. Architecture globale de l'implémentation du VDS-DM

Dans l'architecture globale de notre implémentation (**Figure 5-1**), nous avons distingué trois phases d'implémentations. Durant la première phase, les fichiers d'extension (*.EXE) sont exécutés en parallèle afin de produire un rapport détaillé de comportement pour chaque fichier. Ensuite dans la seconde phase, ces rapports sont analysés par un Parseur afin d'extraire un vecteur de caractéristiques pour chaque rapport en rangeant tous les vecteurs dans un tableau sauvegardé dans fichier (*.CSV). Dans la dernière phase, nous mettons en œuvre un système de datamining capable de classifier les fichiers en deux classes à savoir bienveillant ou malveillant. Dans les sections suivantes, nous passons en revue chaque phase.

4. Analyse de comportement des fichiers exécutables

C'est la phase la plus laborieuse dans notre implémentation. En effet, dans cette phase, un large nombre de fichiers sont exécutés afin de produire les rapports de ses comportements (rapport en format JSON[7]) dans les machines virtuelles.

[6] Nous restreignons notre étude sur les fichiers de format Windows EXE.
[7] Le rapport de comportement est présenté dans l'annexe

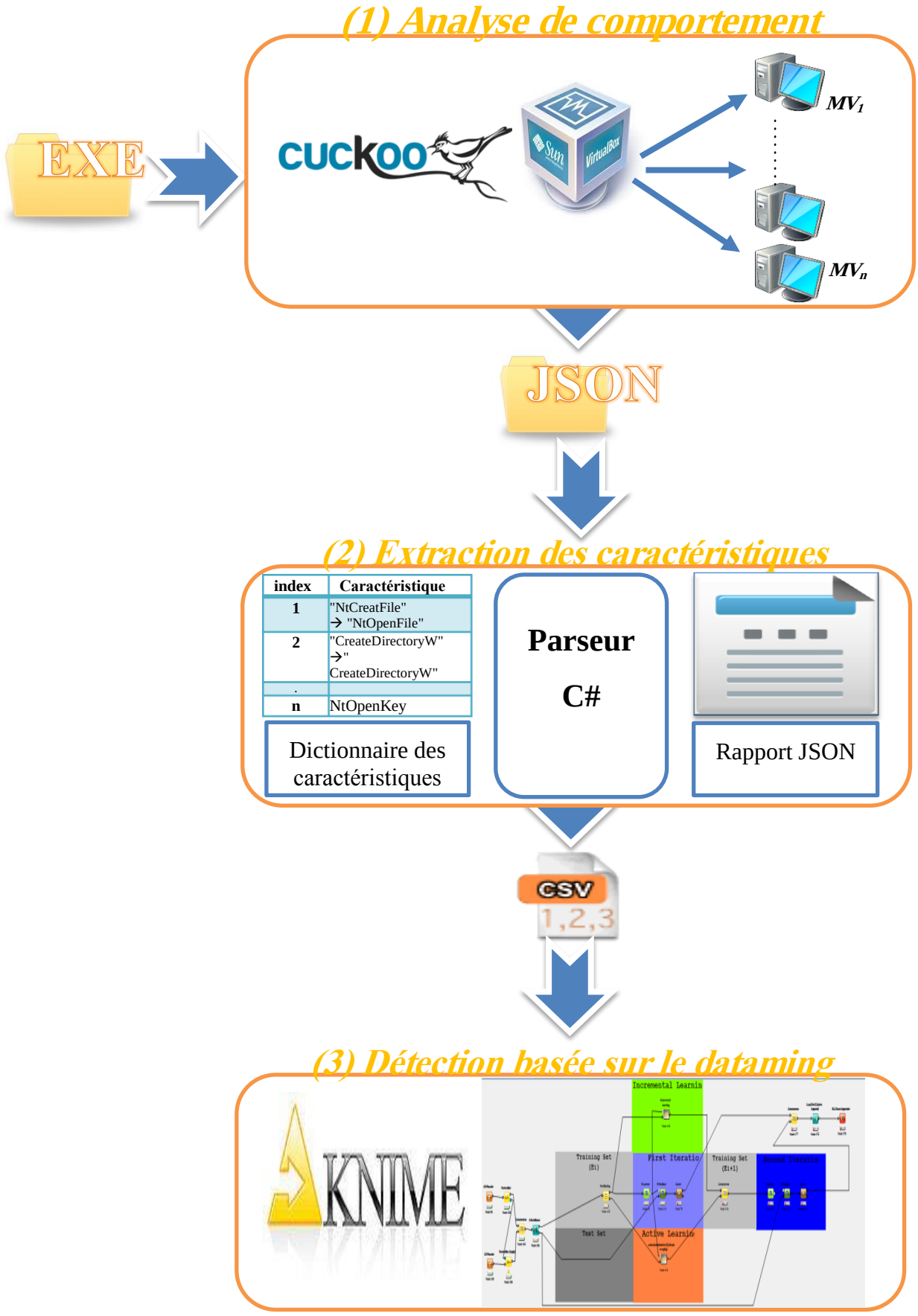


Figure 5-1 Architecture globale de l'implémentation du VDS-DM

Pour faciliter l'analyse comportementale, nous avons recouru à une plateforme publiée récemment (lancé en 2010) appelée Cuckoo Sandbox, un logiciel Open Source pour l'automatisation de l'analyse des fichiers malveillants. Pour le faire, il utilise des composants configurables pour la surveillance des fichiers exécutables lors de l'exécution dans des machines virtuelles (Virtual Box, VMware, QEMU) tout en fournissant des rapports détaillés du comportement. Ainsi, il offre la possibilité d'opérer en parallèle dans plusieurs machines virtuelles.

L'avantage de Cuckoo Sandbox est sa conception modulaire qui permet de personnaliser les phases d'analyse ainsi que les rapports générés. Il est facilement intégrable dans n'importe quel système de détection de virus informatiques ce qui le rend un investissement potentiel pour les experts de sécurité.

Dans notre implémentation, nous avons combiné Cuckoo Sandbox et VirtualBox pour former un laboratoire d'analyse de comportement sous l'Unix, qui peut analyser un nombre élevé de fichiers exécutables. Dans les sections suivantes, nous présentons les configurations et les caractéristiques de notre environnement d'analyse de fichiers exécutables.

4.1. Préparation de l'environnement d'analyse de comportement

4.1.1. Préparation de la machine virtuelle

Nous avons préparé les machines virtuelles orientées pour effectuer une analyse de comportement. Bien entendu, nous avons choisi Windows XP (la version professionnelle 2002 Service Pack2) comme un système d'exploitation des machines virtuelles. De même, nous avons installé des versions anciennes des logiciels utilisés fréquemment (PDF, Microsoft Office, Internet Explorer… etc.). Nous avons aussi désactivé le par-feu du Windows et la mise-à-jour automatique.

Nous avons pris toutes ces mesures pour bénéficier des failles de sécurité lors de l'exécution des virus afin d'extraire le maximum d'information sur les comportements malveillants.

Pour accélérer le processus d'analyse d'un large nombre de fichiers, nous avons cloné sept fois une machine virtuelle préparée selon les mesures citées précédemment.

4.1.2. Configuration de Cuckoo

Nous avons configuré Cuckoo Sandbox comme suit :

```
machine_manager = virtualbox
```

Pour définir le logiciel de virtualisation, dans notre cas VirtualBox.

```
[timeout]

default = 120
```

Pour définir le temps d'analyse par défaut, exprimé en secondes. Cette valeur sera utilisée pour définir la durée d'analyse. Dans notre cas, chaque fichier s'exécute pendant deux minutes.

```
label = machineName
platform = windows
ip = @ip
```

Les trois lignes précédentes représentent une machine en spécifiant son nom, le système d'exploitation et son adresse IP. Ces trois lignes sont répétées autant de machines utilisées.

La liste précédente de configuration n'est pas exhaustive, dans l'annexe nous fournissons tous les fichiers de configuration.

4.2. Extraction des caractéristiques de programme

Le rôle de cette phase est l'extraction des vecteurs de caractéristiques pour chaque programme. Pour faire cela, les rapports de comportement générés dans la phase précédente sont analysés par un parseur (programme C#) afin de produire un fichier CSV. Ce fichier CSV contient un tableau (**Tableau 5-1**), chaque ligne de ce dernier représente un vecteur de caractéristiques (vecteur de présence des caractéristiques) d'un programme P_i. La dernière

colonne contient les étiquettes[8] de programmes. Néanmoins, si l'étiquette n'est pas connue, cette colonne reste vide.

Concernant la nature des caractéristiques de programme $(f_1, f_2, f_3, \dots, f_M)$ utilisées dans notre implémentation, nous avons utilisé les n-gramme des API de longueurs <=5. Autrement dit, chaque caractéristique est une séquence d'appels API (Api_1→ Api_2 → Api_3 → Api_4→ Api_5). Le nombre de caractéristiques de programme choisi dans tous les tests est M= 994. La liste de toutes ces caractéristiques est délivrée dans l'annexe.

Ce type de caractéristiques est utilisé dans plusieurs travaux ***[Ahmed, et al., 2009; Ravi, et al., 2012; Salehi, et al., 2012]***. De même, les virus effectuent leurs tâches malveillantes en passant par les API Windows ce qui justifie ce choix.

	f_1	f_2	f_3	.	.	.	F_M	*Étiquette*
P_1	0	1	0	.	.	.	0	*Virus*
P_2	0	0	1	.	.	.	1	*Non*
.	.	.	.	.	.	.	.	.
P_n	1	0	1	.	.	.	0	*Virus*

Tableau 5-1 Structure du tableau sauvegardé dans un fichier CSV.

5. Détection des virus informatiques basée sur le datamining

Après avoir terminé l'abstraction des programmes en vecteurs des caractéristiques dans les phases précédentes, on peut faire recours à n'importe quelle plateforme de datamining pour analyser les données. Néanmoins, nous avons opté pour le benchmark ***Knime*** à cause de sa richesse en matière de fonctionnalités disponibles sous forme de nœuds (>900 nœuds). En plus, une étude comparative faite par Xiaojun Chen et ses collaborateurs a affirmé que Knime est la meilleure parmi les différentes plateformes célèbres de datamining ***[Chen, et al., 2007]***.

Dans notre implémentation, nous avons profité de l'extensibilité de Knime pour implémenter de nouveaux nœuds, en utilisant les deux langages R et C#.

[8] Une étiquette est une valeur qui distingue les virus de programmes bienveillants lors de l'entrainement du classificateur.

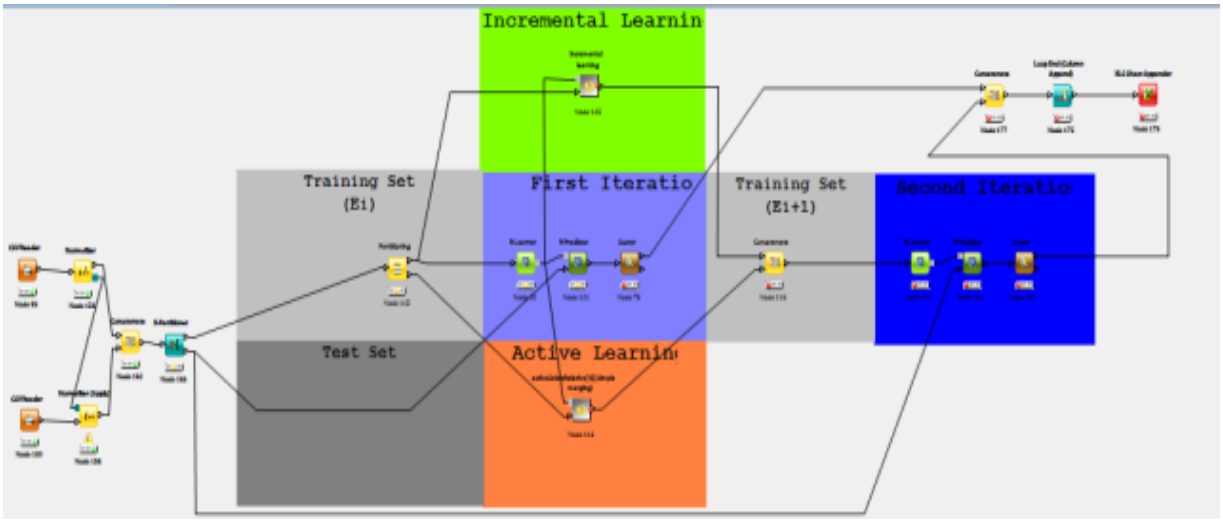


Figure 5-2 Partie implémentation du VDS-DM dans Knime

Le schéma précédent (**Figure 5-2**) représente l'implémentation en Knime de notre architecture du système de détection de virus basé sur le datamining proposée dans le chapitre conception. Les deux nœuds développés sont :

- Apprentissage actif (Active Learning) : le nœud responsable de la sélection des exemples pour l'étiquetage. Dans le chapitre précèdent, nous avons présenté les approches d'apprentissage actif utilisées. Pour programmer ce nœud, nous avons utilisé les langages R et C#.
- Apprentissage incrémental (Incremntal Learning): le nœud qui filtre les exemples étiquetés des itérations précédentes. L'algorithme de filtrage incrémental est évoqué dans le chapitre de conception. Pour programmer ce nœud, nous avons utilisé le langage R.

6. Résultats expérimentaux

6.1. Les données de test

Les tests ont été effectués en utilisant l'ensemble de données de site (http://vxheaven.org/) utilisé dans des travaux antérieurs ***[Wang, et al., 2010; Wang, et al., 2009; Chao, et al., 2009]***. Sachant que la taille de la base est très grande (supérieur à 40 Gigas), nous avons sélectionné un ensemble de fichiers exécutables (extension .EXE) pour former notre base de test.

Dans notre sélection de fichiers, nous avons écarté deux types de fichiers

- Les fichiers qui ne sont pas exécutés à cause d'erreurs.
- Les fichiers qui ont arrêté leur exécutions après la détection de l'environnement de l'analyse de comportement afin d'échapper à la détection.

La base utilisée dans nos tests est présentée dans le tableau (**Tableau 5-2 Base de test**).

Nombre de programmes bienveillant	915
Nombre de programmes malveillants (Virus)	1268
Nombre total de programmes	2183

Tableau 5-2 Base de test

6.2. Le scénario de test

Dans le but d'estimer la performance de notre algorithme d'apprentissage incrémental/actif, nous proposons un scénario de test adéquat à la base de validation croisée. Cette technique statistique peut évaluer la généralisation du modèle. Elle est utilisée lorsqu'on veut savoir comment un modèle prédictif va agir en pratique.

La validation croisée a été utilisée avec l'apprentissage supervisé avec succès. Cependant, dans l'apprentissage actif ou incrémental il existe une stratégie d'échantillonnage d'exemples. Par conséquent, l'adaptation de la validation croisée est nécessaire. Dans notre version modifiée, nous avons ajouté un paramètre **T** qui représente le taux d'exemples sélectionnés. Dans ce cas, le modèle est entrainé en utilisant **L** échantillons sélectionnés à partir les (k-1) groupes. Tel que :

$$L = T * \text{ nombre d'exemples des (k-1) groupes}$$

Une fois le modèle est prêt, le K^{iem} groupe restant est utilisé pour évaluer le modèle (**Figure 5-3**). Selon ce scénario de test, il est possible de comparer plusieurs configurations en variant les taux des exemples sélectionnés et la stratégie d'échantillonnage exercée.

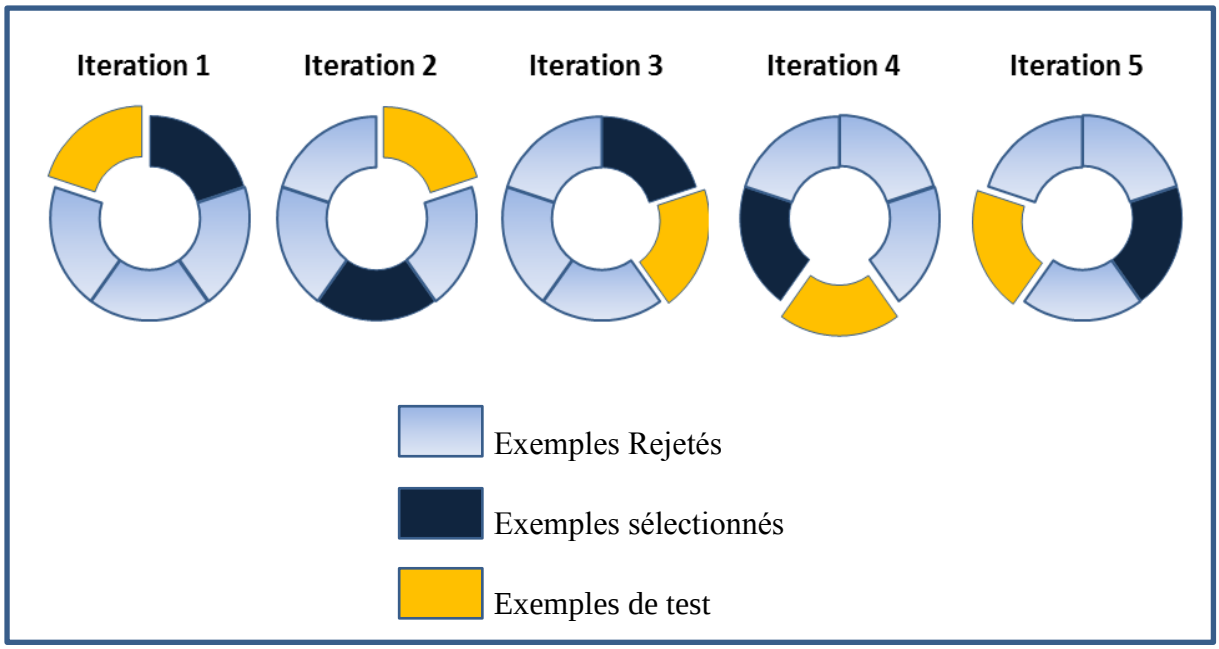


Figure 5-3 Validation croisée modifiée

Dans tous nos tests, nous considérons les stratégies d'échantillonnages suivants :

1. Échantillonnage aléatoire : les T exemples sont sélectionnés aléatoirement sans algorithme de sélection des exemples (**0<T<1**).
2. Échantillonnage basé sur l'apprentissage actif/incrémental : les exemples sont sélectionnés selon un algorithme (**0<T<1**).
3. Apprentissage supervisé standard : tous les exemples sont sélectionnés pour l'entrainement du modèle (**T=1**).

Dans nos tests, nous nous sommes basé sur les métriques suivantes :

- *Exactitude*

Le taux des fichiers classés par VDS-DM dans la classe réelle :

$$Exactitude = \frac{nombre\ de\ fichiers\ classés\ dans\ la\ classe\ réele}{nombre\ de\ tous\ les\ fichier}$$

- *Vrai Positif*

Le taux de virus détectés parmi tous les virus d'ensemble de test donné :

$$Vrai\ Positif = \frac{nombre\ de\ virus\ détectés}{nombre\ de\ tous\ les\ virus}$$

- *Faux Positif*

Le taux de fichiers bienveillants classé à tort comme des virus :

$$Faux\ Positif = \frac{nombre\ de\ \text{fichiers}\ bienveillants\ classé\ comme\ des\ virus}{nombre\ de\ \text{fichiers bienveillants}}$$

6.3. Comparaison avec l'échantillonnage aléatoire

Nous comparons notre algorithme avec l'échantillonnage aléatoire pour montrer son utilité. En effet, si un algorithme n'améliore pas les résultats par rapport à la sélection aléatoire des exemples, on ne peut pas l'utiliser comme une stratégie d'échantillonnage ***[Wang, et al., 2010]***.

6.3.1. Apprentissage actif basé sur la marge simple

Discussion

En observant les résultats illustrés dans (**Figure 5-4**) , il est clair que l'échantillonnage incrémental/actif donne les meilleurs résultats. En effet, le vrai positif (nombre de virus détectés) dépasse constamment celui de l'apprentissage aléatoire. Néanmoins, lorsque le nombre d'exemples n'est pas suffisamment grand (t<20%) le faux positif (le nombre de programmes bienveillants détectés virus ou les fausses alertes) est relativement élevé. De même, l'exactitude est influée par le faux positif lorsque le nombre d'exemples est petit. Les trois courbes convergent avant (T=100) ce qui confirme que la réduction des exemples

n'influe pas gravement sur la performance. En plus, une stratégie d'échantillonnage incrémental/actif converge plus rapidement qu'une stratégie aléatoire.

Finalement, nous pouvons conclure que l'échantillonnage incrémental/actif est le meilleur en matière de détection de virus. Cependant, pour éviter les fausses alertes, un nombre minimum d'exemples s'avèr nécessaire. Autrement dit, la sélection d'exemples basée sur la marge SVM peut réduire le nombre d'exemples étiquetés sans dégrader la performance du VDS-DM.

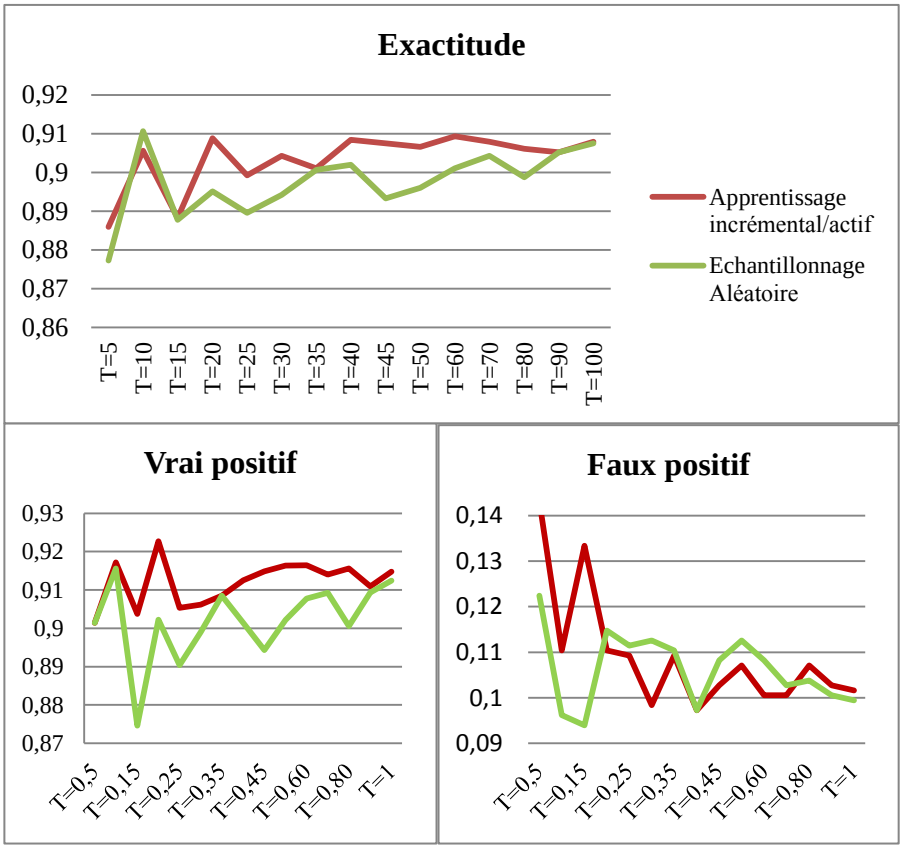


Figure 5-4 Comparaison entre l'échantillonnage incrémental/actif (la marge simple) et l'échantillonnage aléatoire

6.3.2. Apprentissage actif basé sur l'entropie de voisinage

Disscussion

Les résultats illustrés dans (**Figure 5-5**) montre que l'apprentissage basé sur l'échantillonnage aléatoire est plus performant que l'utilisation d'entropie comme méthode de sélection. Plus

précisément, la courbe d'exactitude de l'échantillonnage aléatoire est au-dessus de celle de l'échantillonnage basé sur l'entropie.

Concernant les courbes de vrai positif, l'échantillonnage aléatoire dépasse constamment l'échantillonnage basé sur l'entropie.

Les courbes de faux positif sont en alternance, il existe des intervalles où l'échantillonnage est le meilleur et vice-versa. Cependant, le faux positif de l'échantillonnage basé sur l'entropie est inférieur a celui de l'échantillonnage aléatoire lorsque T>50. Autrement dit, cette technique de sélection d'exemples ne fournit pas des résultats compétitifs avec un minimum d'exemples (T<50).

Dans la suite du chapitre, nous restreignons nos tests sur la sélection basée sur la marge simple. Ce choix est justifié par les résultats de comparaison entre l'échantillonnage incrémental/actif basé sur l'entropie et l'échantillonnage aléatoire.

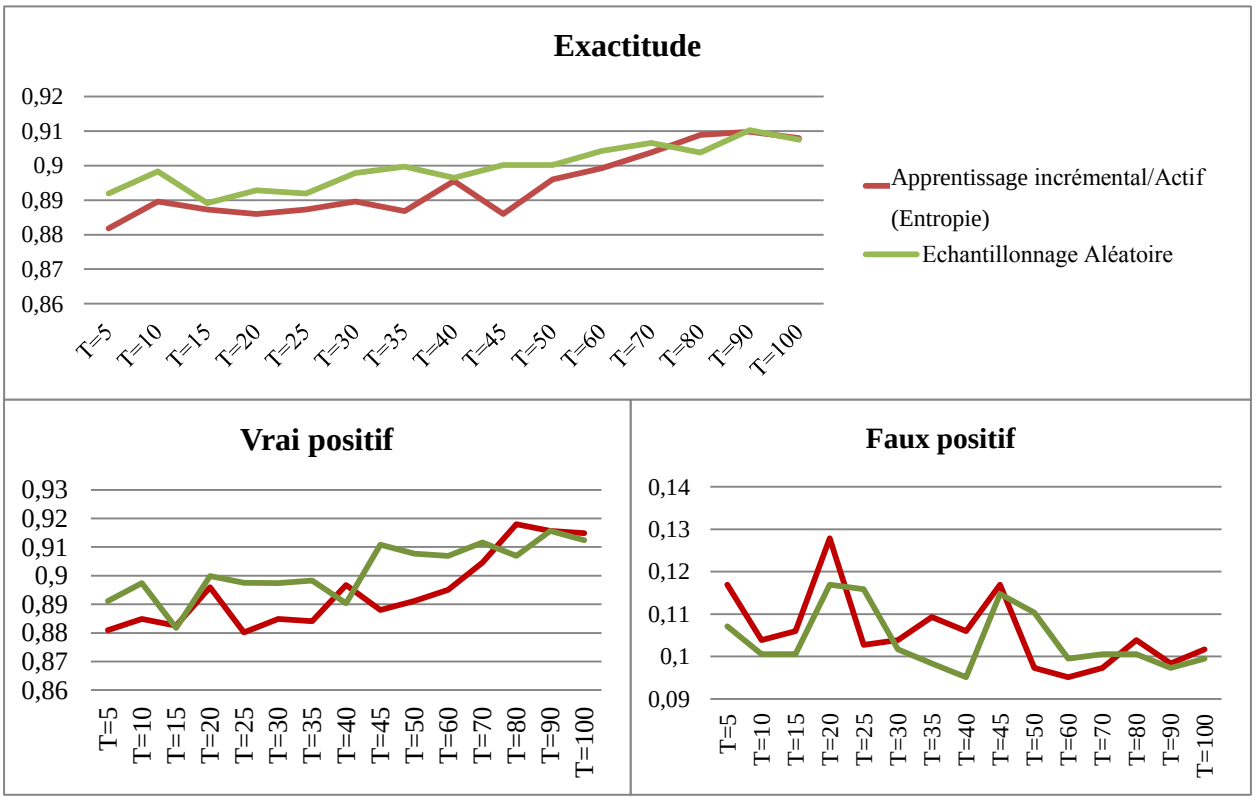


Figure 5-5 Comparaison entre l'échantillonnage incrémental/actif (Entropie de voisinage) et l'échantillonnage aléatoire

6.4. Influence du nombre d'exemples d'entrainement (N) sur la performance

Dans cette section, notre objectif est de mesurer empiriquement l'influence du nombre d'exemples sur la performance de VDS-DM. Pour le faire, nous prenons l'apprentissage supervisé standard (en utilisant tous les exemples disponibles dans l'entrainement) comme un repère.

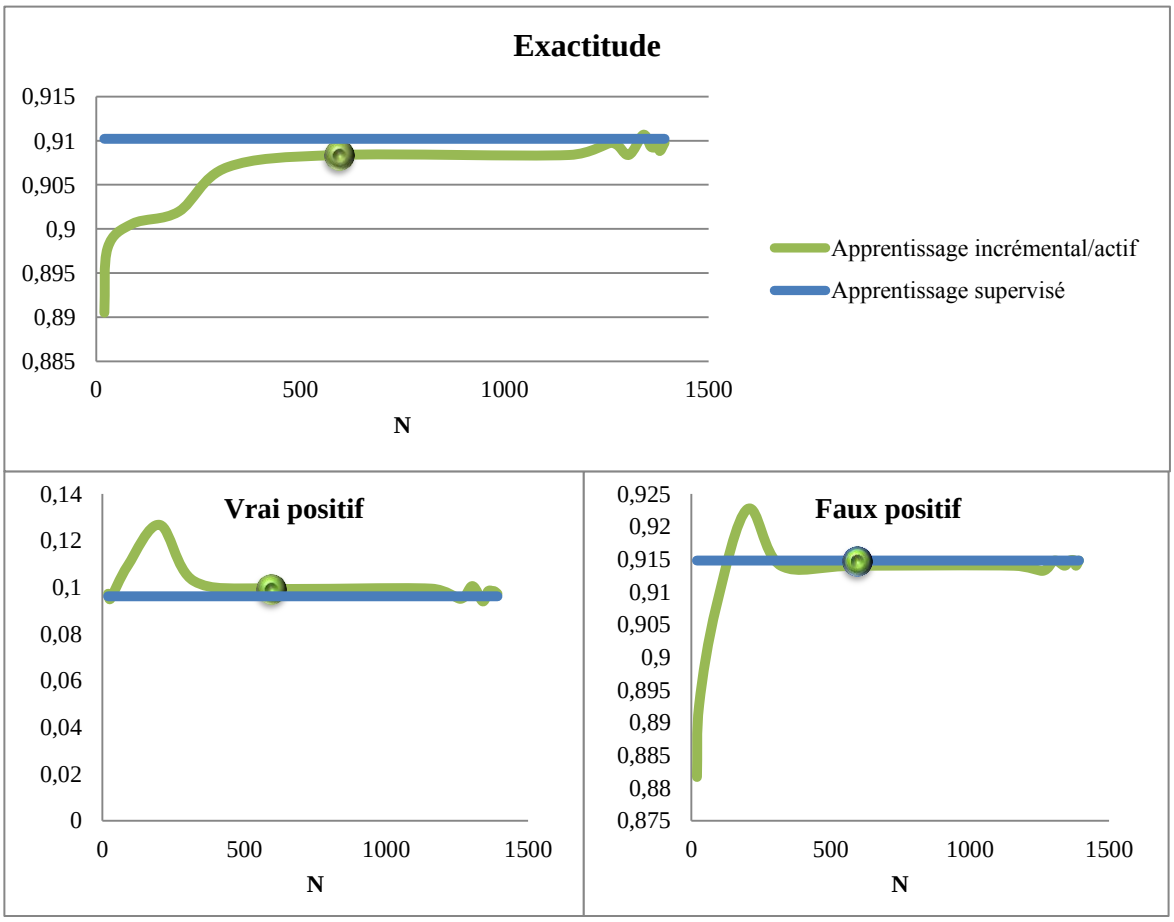


Figure 5-6 Influence de N sur la performance de VDS-DM

Discussion

D'après (**Figure 5-6**), les trois courbes convergent avec un petit nombre d'exemples (environ 500). Néanmoins, un nombre très petit d'exemples (N<500) peut dégrader les résultats.

In fine, nous avons atteint les résultats de l'apprentissage supervisé standard (N=1500 exemples étiquetés) avec un nombre réduit des exemples (N=500). Cette réduction de 1000 exemples étiquetés économise beaucoup les ressources utilisées dans l'étiquetage.

6.5. Influence du nombre initial d'exemples étiquetés $S_0\%$[9]

Dans cette section, l'objectif est de mesurer l'influence de la taille de l'ensemble initial d'entrainement ($S_0 = |E_0|$) sur la performance des itérations suivantes. Autrement dit, nous ciblons la vérification de l'hypothèse suivante :

> « *Est-ce qu'un démarrage aléatoire avec peu d'exemples peut influer sur l'évolution de la performance du VDS-DM dans le futur.* »

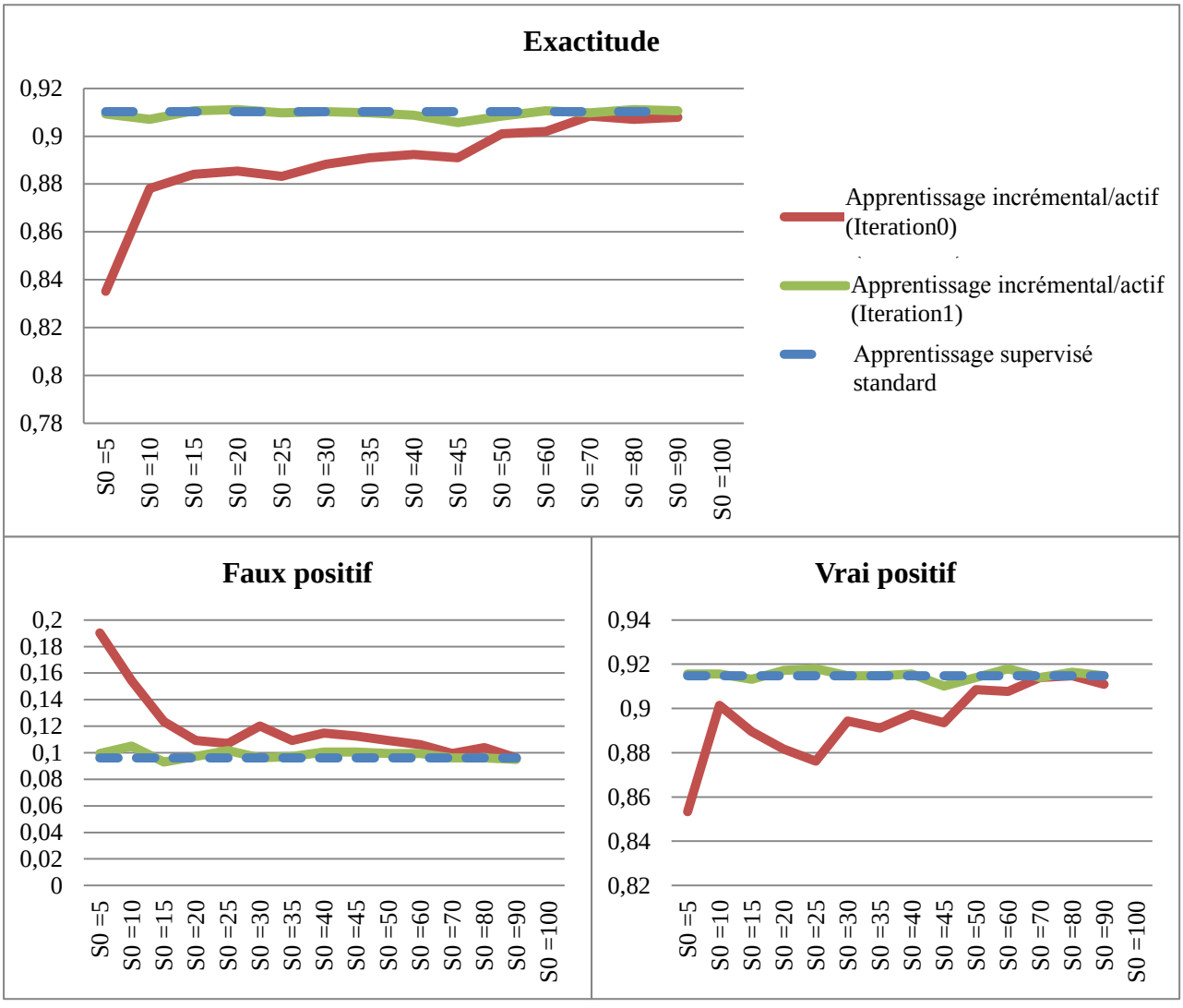


Figure 5-7 Influence du nombre initial d'exemples étiquetés $S_0\%$

[9] S_0 représente le pourcentage d'exemples de E_0 par rapport à la totalité des exemples d'entrainement disponible.

Inévitablement, dans l'itération initiale de l'apprentissage incrémentale/active, nous utilisons un nombre petit d'exemples (sélectionnés aléatoirement) afin de former le modèle SVM initial qui sera utilisé pour sélectionner les exemples dans les itérations suivantes.

Dans (**Figure 5-7**), il existe trois courbes:

- La courbe rouge : représente la performance initiale du VDS-DM (itération 0).
- La courbe verte : représente la performance de la première itération.
- La courbe bleue : représente la performance de l'apprentissage supervisé utilisé comme référence.

Discussion

D'après (**Figure 5-7**), il est clair que la performance de l'itération initiale dépend de la taille de S0 (exprimé dans les courbes en pourcentage par rapport à tout l'ensemble des exemples d'entrainement de l'apprentissage supervisé standard). En effet, la performance du VDS-DM avec S0 petit peut être très dégradée (Exactitude = 84%, Vrai Positif < 85%, Faux Positif=20%). Tandis que, la performance de la seconde itération ne dépend pas de la taille de S0. En effet, la courbe de VDS-DM est presque identique avec celle de l'apprentissage supervisé standard.

Finalement, nous pouvons dire que notre VDS-DM s'ajuste automatiquement quel que soit sa performance initiale. Cette propriété fait du VDS-DM un système stable dans son évolution indépendamment de son démarrage à froid.

6.6. Influence de la distance au plan séparateur du SVM sur la performance (R)

Nous avons présenté dans le chapitre précèdent une méthode de sélection active basée sur SVM. Dans cette méthode, la distance d'un exemple (x) à un modèle SVM (M) est utilisée. Pour la calculer, on utilise la formule suivante :

$$Distance(x, M) = \left| \sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b \right|$$

Pour faire une sélection d'exemples, il existe deux possibilités :

- Sélectionner (***NbrRequettes)*** *d'*exemples qui ont des valeurs de distance élevées.
- Fixer un seuil d'acceptation R en sélectionnant les exemples qui ont des valeurs de distances inférieures à ce seuil.

Dans cette section, nous voulons étudier empiriquement l'influence du seuil R sur la performance de VDS-DM tout en choisissant la valeur optimale de ce paramètre.

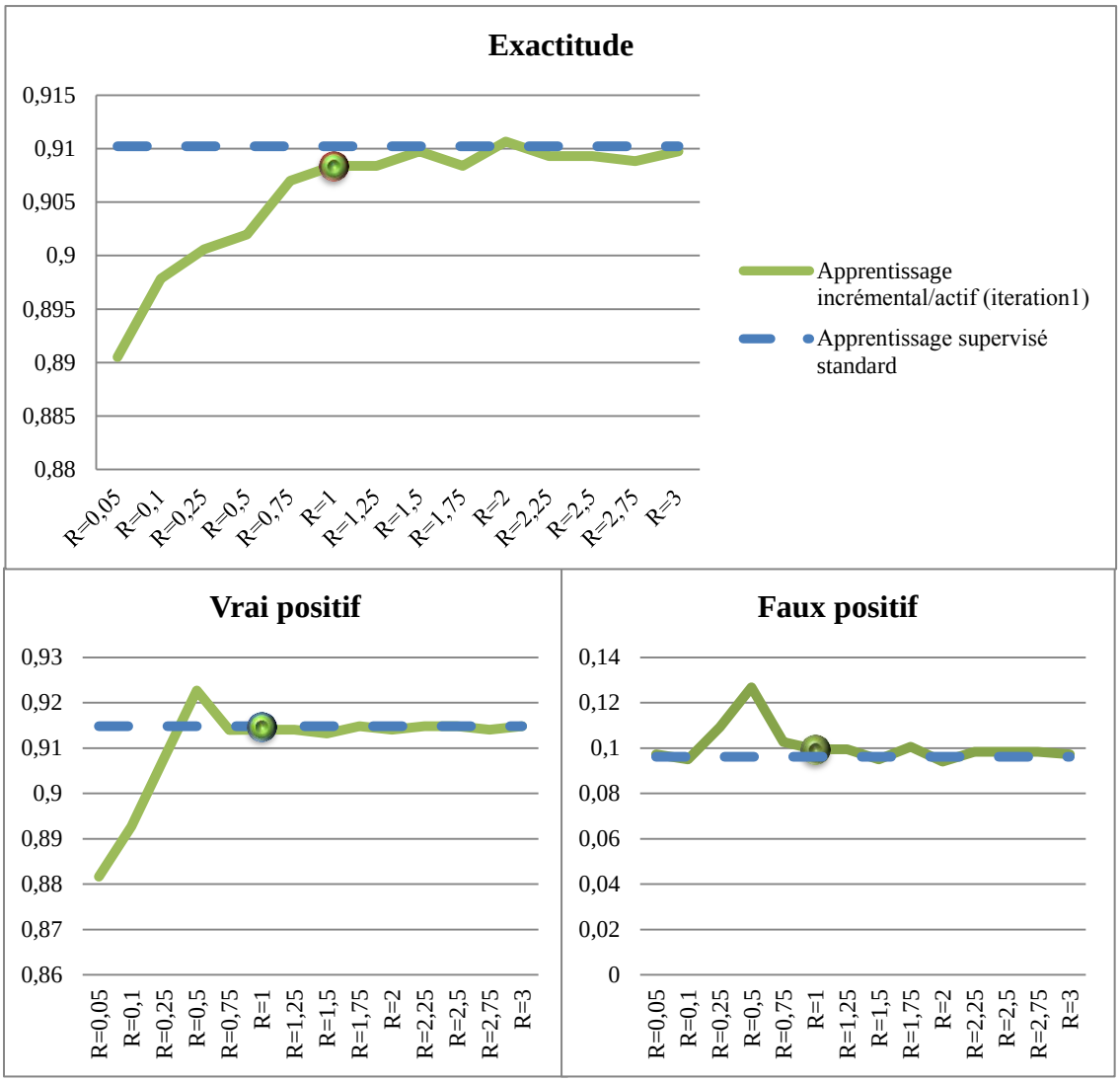


Figure 5-8 Influence de R sur la performance de VDS-DM

Discussion

Dans **(Figure 5-8)** les trois courbes convergent vers la performance référence en augmentant le seuil R. En effet, lorsque (R<1) la performance du VDS-DM est dégradée par rapport à la référence. Cependant, lorsqu'on dépasse un seuil (R=1) le VDS-DM manifeste une performance presque identique à celle de l'apprentissage supervisé standard.

Sachant que les exemples qui vérifient $Distance(x, M) \leq 1$ sont les exemples qui se situent dans la marge du modèle SVM y compris les vecteurs supports. Par conséquent, les exemples de la marge, lorsqu'on demande à un expert de sécurité de les étiqueter, améliorent la performance mieux que les autres exemples qui se situent hors marge.

7. Conclusion

Dans ce chapitre, nous nous sommes focalisés sur les outils utilisés pour implémenter notre système de détection de virus informatiques. Ainsi, nous avons présenté les résultats selon des métriques standards.

Pour conclure, notre travail se résume en deux principales contributions. La première réside dans l'analyse de comportement, en proposant un laboratoire capable d'analyser efficacement un nombre élevé de fichiers. Tandis que, la deuxième réside dans l'amélioration du processus de datamining utilisé dans les systèmes de détection de virus informatiques.

Notre système de détection de virus est évolutif et permet d'optimiser le nombre d'exemples analysés par un expert humain, ce qui fournit un outil datamining puissant aux firmes des anti-virus dans la guerre cybernétique.

Chapter 6 : CONCLUSION GENERALE

Le duel virus/antivirus dure depuis environ une quinzaine d'années. Il est sans fin. Les résultats de Fred Cohen ont montré qu'il n'existe aucune protection absolue. Mais ce que l'histoire nous apprend, c'est qu'il n'existe, non plus, aucune arme absolue. Tout le jeu reste une affaire de connaissances et de compétences. L'équilibre est maintenu par le transfert des connaissances du virus à l'antivirus. Diffuser un virus revient à diffuser également le savoir permettant de lutter contre ***[Filiol, 2009]***.

Beaucoup de chercheurs ont proposé diverses techniques heuristiques de détection de virus informatiques. Parmi ces techniques, on trouve le datamining qui fait l'objet de notre travail.

Au terme de ce mémoire, nous procédons dans les lignes qui suivent à un récapitulatif du travail effectué. Rappelons que notre travail consiste à développer un système de détection de virus basé sur les techniques de datamining.

Nous avons présenté dans un premier temps, les types d'infections informatiques existantes en définissant chacune de ces infections et en les classifiant.

Dans un second temps, nous avons procédé à un état de l'art des outils datamining utilisés dans la détection de virus en présentant les différents travaux existants

Nos investigations dans le domaine de détection des virus informatiques, et particulièrement celui de la détection basée sur le datamining, nous ont permis d'apporter des contributions que nous résumons en deux points.

Dans le premier point, nous nous sommes intéressés à adresser l'évolutivité rapide des virus informatiques. En effet, cette explosion en nombre des virus information représente un réel défi pour les techniques de détection. Pour répondre à cette problématique, nous avons investi dans d'autres types d'apprentissage tels que l'apprentissage incrémental et actif.

Dans le second point, nous avons traité l'optimisation de l'interaction entre l'expert de sécurité et le système de détection de virus informatiques. Cette optimisation est manifestée dans le nombre réduit de programmes analysés par l'expert qui supervise le système de datamining.

Finalement, notre mémoire s'inscrit dans l'adaptation des techniques datamining standards pour le domaine de sécurité informatique et plus particulièrement la lutte contre les virus informatiques.

Comme perspectives par rapport à ce travail, il serait intéressant d'utiliser d'autres techniques d'apprentissage actif et d'apprentissage incrémental tout en profitant de l'état de l'art de ces types d'apprentissage.

Une autre perspective consiste à inclure d'autres types de caractéristiques pour mieux représenter les programmes. Autrement dit, la combinaison entre plusieurs visions peut améliorer l'exactitude de détection de virus informatiques.

Il serait aussi intéressant d'implémenter un système qui peut acquérir des nouvelles caractéristiques de programme d'une manière incrémentale. Plus précisément, nous avons implémenté l'évolutivité de point de vue acquisition de nouveaux virus. Néanmoins, l'ajout de caractéristiques n'est pas évoqué dans notre étude.

Comme dernière perspective, l'utilisation de notre approche dans des scénarios réels ou avec une base de test plus grande que la nôtre peut fournir des résultats plus concrets.

BIBLIOGRAPHIE

Abou-Assaleh Tony [et al.] N-gram-based Detection of New Malicious Code [Conférence] // Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International. - Washington, DC, USA : IEEE Computer Society, 2004. - Vol. 2. - pp. 41-42. - ISBN:0-7695-2209-2.

Ahmadi Mansour [et al.] Malware detection by behavioural sequential patterns [Revue] // Computer Fraud & Security. - [s.l.] : Elsevier, Aout 2013. - 8 : Vol. 2013. - pp. 11 - 19. - ISSN:1361-3723.

Ahmed Faraz [et al.] Using Spatio-temporal Information in API Calls with Machine Learning Algorithms for Malware Detection [Article] // Proceedings of the 2Nd ACM Workshop on Security and Artificial Intelligence. - Chicago, Illinois, USA : ACM, 2009. - pp. 55-62. - ISBN:978-1-60558-781-3.

Ahmed Faraz [et al.] Using Spatio-temporal Information in API Calls with Machine Learning Algorithms for Malware Detection [Conférence] // Proceedings of the 2Nd ACM Workshop on Security and Artificial Intelligence. - New York, NY, USA : ACM, 2009. - pp. 55-62. - ISBN:978-1-60558-781-3.

Al Daoud Essam, Jebril Iqbal H. et Zaqai Belal Computer Virus Strategies and Detection Methods. [Revue] // Open Problems Compt. Math.. - 2008. - Vol. 1. - pp. 29-36.

Aycock John Computer Viruses and Malware [Ouvrage]. - [s.l.] : Springer Science+Business Media, LLC, 2006.

Bailey Michael [et al.] Automated Classification and Analysis of Internet Malware. [Section] // Recent Advances in Intrusion Detection / éd. Kruegel Christopher, Lippmann Richard et Clark Andrew. - [s.l.] : Springer Berlin Heidelberg, 2007. - Vol. 4637. - ISBN:978-3-540-74319-4.

Bayer Ulrich [et al.] Scalable, Behavior-Based Malware Clustering. [Article] // NDSS. - [s.l.] : The Internet Society, 18 Juin 2009.

Bilar Daniel Opcodes As Predictor for Malware [Revue] // Int. J. Electron. Secur. Digit. Forensic. - Inderscience Publishers, Geneva, SWITZERLAND : Inderscience Publishers, May 2007. - 2 : Vol. 1. - pp. 156-168. - ISSN:1751-911X.

Bramer Max Decision Tree Induction: Using Entropy for Attribute Selection [Section] // Principles of Data Mining. - [s.l.] : Springer London, 2007. - ISBN:978-1-84628-765-7.

Canali Davide [et al.] A Quantitative Study of Accuracy in System Call-based Malware Detection [Article] // Proceedings of the 2012 International Symposium on Software Testing and Analysis. - Minneapolis, MN, USA : ACM, 2012. - pp. 122-132. - ISSN:978-1-4503-1454-1.

Cesare Silvio et Xiang Yang Software Similarity and Classification [Ouvrage]. - [s.l.] : Springer, 2012. - ISBN:978-1-4471-2909-7.

Chao Rui et Tan Ying A Virus Detection System Based on Artificial Immune System [Conférence] // International Conference on Computational Intelligence and Security. - 2009. - pp. 6-10.

Chen Thomas M. et Abu-Nimeh Saeed Lessons from Stuxnet [Revue] // Computer. - [s.l.] : IEEE Computer Society, April 2011. - 4 : Vol. 44. - pp. 91-93. - 0018-9162/11.

Chen Xiaojun [et al.] A Survey of Open Source Data Mining Systems [Section] // Emerging Technologies in Knowledge Discovery and Data Mining / auteur du livre Washio Takashi [et al.]. - [s.l.] : Springer Berlin Heidelberg, 2007. - Vol. 4819. - ISBN 978-3-540-77016-9.

Christodorescu Mihai, Jha Somesh et Kruegel Christopher Mining specifications of malicious behavior [Article] // Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering. - Dubrovnik, Croatia : ACM, 2007. - pp. 5-14. - ISBN: 978-1-59593-811-4 .

Dube T. [et al.] Malware target recognition via static heuristics [Revue] // Computers & Security. - [s.l.] : Elsevier, Février 2012. - 1 : Vol. 31. - pp. 137-147. - ISSN:0167-4048.

Dubois Michel Définition des virus [En ligne] // http://vaccin.sourceforge.net. - 11 Mars 2014. - http://vaccin.sourceforge.net/docs/definition2.html.

Dziczkowski Grzegorz Analyse des sentiments : système autonome d'exploration des opinions exprimées dans les critiques cinématographiques. [Rapport] : Thèse de doctorat / École nationale supérieure des mines de Paris (Mines ParisTech). - 2008.

Egele Manuel [et al.] A Survey on Automated Dynamic Malware-analysis Techniques and Tools [Revue] // ACM Computing Surveys (CSUR). - New York, NY, USA : ACM, Février 2012. - 2 : Vol. 44. - pp. 1-42. - ISSN:0360-0300.

Eskandar Mojtabai et Hashemi Sattar Metamorphic Malware Detection using Control Flow Graph Mining [Revue] // International Journal of Computer Science and Network Securityr (IJCSNS). - Décember 2011. - 12 : Vol. 11. - pp. 1-6.

Eskandari M., Khorshidpur Z. et Hashemi S. To Incorporate Sequential Dynamic Features in Malware Detection Engines [Article] // Intelligence and Security Informatics Conference (EISIC), 2012 European. - [s.l.] : IEEE, 22-24 Aout 2012. - pp. 46-52. - ISBN:978-1-4673-2358-1.

Eskandari Mojtaba et Hashem Sattar A graph mining approach for detecting unknown malwares [Revue] // Journal of Visual Languages & Computing. - [s.l.] : Elsevier, 2012. - 3 : Vol. 23. - pp. 154-162. - ISSN:1045-926X.

Filiol Eric Les virus informatiques : théorie, pratique et applications. [Ouvrage]. - [s.l.] : Verlag France, 2009.

Forrest S. [et al.] Self-Nonself Discrimination in a Computer [Revue] // Research in Security and Privacy. - 1994. - pp. 202 - 212.

Gostev Aleks Cyber-threat evolution: the year ahead [Revue] // Computer Fraud & Security. - mars 2012. - 3 : Vol. 2012. - pp. 9-12.

Greengard Samuel The new face of War [Revue] // Commun. ACM. - New York, NY, USA : [s.n.], December 2010. - 12 : Vol. 53. - pp. 20--22. - 10.1145/1859204.1859212.

H. George John et Langley Pat Estimating Continuous Distributions in Bayesian Classifiers. [Article] // Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence.. - Montréal, Qué, Canada : Morgan Kaufmann Publishers Inc, 1995. - pp. 338-345. - ISBN:1-55860-385-9.

Harley David, Slade Robert et Gattiker Urs Viruses Revealed [Ouvrage]. - [s.l.] : The McGraw-Hill Companies, Inc, 2001.

Henchiri Olivier et Japkowicz Nathalie A Feature Selection and Evaluation Scheme for Computer Virus Detection [Conférence] // Proceedings of the Sixth International Conference on Data Mining. - [s.l.] : IEEE, 2006. - pp. 891-895. - ISSN:1550-4786.

Islam Rafiqul [et al.] Classification of malware based on integrated static and dynamic features [Revue] // Journal of Network and Computer Applications. - [s.l.] : Elsevier, 2013. - 2 : Vol. 36. - pp. 646-656. - ISSN:1084-8045.

Jiang Qingshan , Zhao Xinxing et Huang Kai A feature selection method for malware detection [Conférence] // International Conference on Information and Automation. - Shenzhen, China : [s.n.], 2011. - pp. 890-895. - DOI:10.1109/ICINFA.2011.5949122.

Jiang Qingshan, Zhao Xinxing et Huang Kai A feature selection method for malware detection [Conférence] // Information and Automation (ICIA), 2011 IEEE International Conference on. - Shenzhen : IEEE, 2011. - pp. 890-895.

Karim Md.Enamul. [et al.] Malware phylogeny generation using permutations of code [Revue] // Journal in Computer Virology. - [s.l.] : Springer-Verlag, 2005. - 1-2 : Vol. 1. - pp. 13-23. - ISSN:1772-9890.

Karnouskos Stamatis Stuxnet worm impact on industrial cyber-physical system security [Conférence] // IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society. - Karlsruhe, Germany : [s.n.], 2011. - pp. 4490-4494. - ISSN 1553-572X.

Kephart Jeffrey O. [et al.] Biologically Inspired Defenses Against Computer Viruses [Conférence] // International joint conference on Artificial intelligence. - 1995. - Vol. 1.

Kohavi Ronny et Quinlan J. Ross Data Mining Tasks and Methods: Classification: Decision-tree Discovery [Section] // Handbook of Data Mining and Knowledge Discovery. - [s.l.] : Oxford University Press, Inc., 2002. - ISBN:0-19-511831-6.

Kolbitsch Clemens [et al.] Effective and Efficient Malware Detection at the End Host. [Article] // Proceedings of the 18th Conference on USENIX Security Symposium. - Montreal, Canada : USENIX Association, 2009. - pp. 351-366.

Kolter J. Zico et Maloof Marcus A. Learning to Detect and Classify Malicious Executables in the Wild [Revue] // J. Mach. Learn. Res.. - [s.l.] : JMLR.org, 1 12 2006b. - Vol. 7. - pp. 2721-2744. - ISSN:1532-4435.

Kolter Jeremy Z. et Maloof Marcus A. Learning to detect malicious executables in the wild [Conférence] // Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. - Seattle, WA, USA : ACM, 2004. - pp. 470--478. - ISBN:1-58113-888-1.

Kolter Jeremy Z. et Maloof Marcus A. Learning to detect malicious executables in the wild [Conférence] // Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. - Seattle, WA, USA : ACM, 2004a. - pp. 470--478. - ISBN:1-58113-888-1.

Lance Olivier PARTIE 1 : Aperçu du format PE [En ligne] // Developpez.com. - Developpez.com, 03 Septembre 2005. - 11 Mars 2014. - http://olance.developpez.com/articles/windows/pe-iczelion/format-pe/.

Langner Ralph Stuxnet: Dissecting a Cyberwarfare Weapon [Revue] // Security Privacy, IEEE. - mai-juin 2011. - 3 : Vol. 9. - pp. 49-51. - ISSN 1540-7993.

LI Pele, SALOUR MEHDI et SU XIAO A Survey Of Internet Worm Detection And Containment [Revue] // Communications Surveys & Tutorials. - 2008. - Vol. 10. - pp. 20-35.

Liu Yu-Feng [et al.] Detecting Trojan horses based on system behavior using machine learning method. [Article] // Machine Learning and Cybernetics (ICMLC), 2010 International Conference on. - Qingdao : IEEE, 11-14 Juillet 2010. - Vol. 2. - pp. 855-860. - ISBN:978-1-4244-6526-2.

Ludwig Mark A. The Giant Black Book of Computer Viruses [Ouvrage]. - Arizona : American Eagle Publications, Inc., 1995.

Ludwig Mark A. The Little Black Book of Computer Viruses [Ouvrage]. - Arisona : American Eagle Publications, Inc., 1996.

Maloof Marcus A. Machine Learning and Data Mining for Computer Security: Methods and Applications (Advanced Information and Knowledge Processing) [Ouvrage] / éd. Ltd Springer London. - 2006. - p. 228 pages.

Mitchell Thomas M. Machine Learning [Ouvrage]. - [s.l.] : McGraw-Hill, Inc., 1997. - 1. - ISBN:0070428077, 9780070428072.

Moonsamy Veelasha , Tian Ronghua et Batten Lynn Feature reduction to speed up malware classification [Conférence] // Proceedings of the 16th Nordic conference on Information Security Technology for Applications / éd. Laud Peeter. - Tallinn, Estonia : Springer-Verlag, 2012. - Vol. 7161. - pp. 176-188. - ISBN:978-3-642-29614-7.

Moskovitch R. [et al.] Unknown malcode detection via text categorization and the imbalance problem [Conférence] // Intelligence and Security Informatics, 2008. ISI 2008. IEEE

International Conference on. - [s.l.] : IEEE, 2008d. - pp. 156-161. - DOI: 10.1109/ISI.2008.4565046.

Moskovitch Robert [et al.] Unknown Malcode Detection Using OPCODE Representation [Conférence] // Proceedings of the 1st European Conference on Intelligence and Security Informatics. - Esbjerg, Denmark : Springer-Verlag, 2008a. - pp. 204-215. - ISBN:978-3-540-89899-3.

Moskovitch Robert, Nissim Nir et Elovici Yuval Malicious Code Detection Using Active Learning [Revue] // Privacy, Security, and Trust in KDD / éd. Francesco Bonchi [et al.]. - Berlin, Heidelberg : [s.n.], 2009. - Vol. 5456. - pp. 74-91. - 978-3-642-01717-9.

Mutz Darren [et al.] Anomalous System Call Detection [Revue] // ACM Transactions on Information and System Security (TISSEC). - New York, NY, USA : ACM, Février 2006. - 1 : Vol. 9. - pp. 61-93. - ISSN:1094-9224.

OCED Malcious Software (Malware): A Security Threat to the Internet Economy [Rapport] : Rapport d'information ministérielle / Organisation de coopération et de développement économiques (OCDE). - Seoul, Korea : [s.n.], 2008.

OECD Malcious Software (Malware): A Security Threat to the Internet Economy [Rapport] : Rapport d'information ministérielle / Organisation de coopération et de développement économiques (OCDE). - Seoul, Korea : [s.n.], 2008.

Park Younghee [et al.] Fast Malware Classification by Automated Behavioral Graph Matching. [Article] // Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research. - Oak Ridge, Tennessee : ACM, 2010. - pp. 1-4. - ISBN:978-1-4503-0017-9.

Park(b) Younghee, Reeves Douglas S. et Stamp Mark Deriving common malware behavior through graph clustering. [Revue] // Computers & Security. - [s.l.] : Elsevier, Novembre 2013. - 0 : Vol. 39, Part B. - pp. 419-430. - ISSN:0167-4048.

Peng Bin-Bin, Sun Zheng-Xing et Xu Xiao-Gan system, SVM-based incremental active learning for user adaptation for online graphics recognition [Conférence] // Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on. - 2002. - Vol. 3. - pp. 1379-1386. - 0-7803-7508-4.

Quinlan J. R. Induction of Decision Trees. [Revue] // Machine Learning. - [s.l.] : Kluwer Academic Publishers, Mars 1986. - 1 : Vol. 1. - pp. 81-106. - ISSN:0885-6125.

Quinlan J. Ross C4.5: Programs for Machine Learning [Ouvrage]. - [s.l.] : Morgan Kaufmann Publishers Inc., 1993. - ISBN:1-55860-238-0.

Ravi Chandrasekar et Manoharan R Malware Detection using Windows Api Sequence and Machine Learning [Revue] // International Journal of Computer Applications. - New York, USA. : Foundation of Computer Science, April 2012. - 17 : Vol. 43. - pp. 12-16.

Rieck Konrad [et al.] Automatic Analysis of Malware Behavior Using Machine Learning. [Revue] // Journal of Computer Security. - Amsterdam, The Netherlands, The Netherlands : IOS Press, Décember 2011b. - 4 : Vol. 19. - pp. 639-668.

Rieck Konrad [et al.] Learning and Classification of Malware Behavior. [Article] // Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. - Paris, France : Springer-Verlag, 2008a. - pp. 108-125. - ISBN:978-3-540-70541-3.

RSA RSA 2012 CYBERCRIME TRENDS REPORT The Current State of Cybercrime and What to Expect in 2012 [Rapport]. - [s.l.] : RSA, The Security Division of EMC, 2012.

Ruping Stefan Incremental Learning with Support Vector Machines [Conférence] // Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on. - 2001. - pp. 641-642.

Salehi Zahra , Ghiasi Mahboobeh et Sami Ashkan A miner for malware detection based on API function calls and their arguments [Conférence] // Artificial Intelligence and Signal Processing (AISP), 2012 16th CSI International Symposium on. - Shiraz, Fars : [s.n.], 2012. - pp. 563-568. - ISBN:978-1-4673-1478-7.

Sami Ashkan [et al.] Malware detection based on mining API calls [Conférence] // Proceedings of the 2010 ACM Symposium on Applied Computing. - New York, NY, USA : ACM, 2010. - pp. 1020-1025.

Sammut Claude et Webb Geoffrey I. Encyclopedia of Machine [Ouvrage]. - [s.l.] : © Springer Science+Business Media, LLC 2011, 2011. - E-ISBN:978-0-387-30164-8 .

Santos I. [et al.] Using opcode sequences in single-class learning to detect unknown malware [Revue] // Information Security, IET. - [s.l.] : IEEE, 1 Décembre 2011. - 4 : Vol. 5. - pp. 220-227. - ISSN:1751-8709.

Santos Igor , Nieves Javier et Bringas Pablo G. Semi-supervised Learning for Unknown Malware Detection [Conférence] // International Symposium on Distributed Computing and

Artificial Intelligence / éd. Abraham Ajith [et al.]. - [s.l.] : Springer Berlin Heidelberg, 2011. - Vol. 91. - pp. 415-422. - ISSN:1867-5662.

Schultz Matthew G., Eskin Eleazar et Zadok Erez Data Mining Methods for Detection of New Malicious Executables [Conférence] // Proceedings of the 2001 IEEE Symposium on Security and Privacy. - Oakland, CA : IEEE Computer Society, 2001. - pp. 38-49. - ISBN:0-7695-1046-9.

Shabtai Asaf [et al.] Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey [Revue] // Information Security Technical Report. - [s.l.] : Elsevier Advanced Technology Publications Oxford, UK, UK, February 2009. - 1 : Vol. 14. - pp. 16-29. - 1363-4127.

Shabtai Asaf [et al.] Detecting unknown malicious code by applying classification techniques on OpCode patterns [Revue] // Security Informatics. - [s.l.] : Springer-Verlag, 2012. - 1 : Vol. 1. - pp. 1-22. - DOI:10.1186/2190-8532-1-1.

Shahzad Farrukh, Shahzad M. et Farooq Muddassar In-execution dynamic malware analysis and detection by mining information in process control blocks of Linux OS [Revue] // Information Sciences (Data Mining for Information Security). - [s.l.] : Elsevier, 10 May 2013. - 0 : Vol. 231. - pp. 45-63. - ISSN:0020-0255.

Siddiqui Muazzam , Wang Morgan C. et Lee Joohan Data mining methods for malware Detection Using Instruction Sequences [Article] // Proceedings of the 26th IASTED International Conference on Artificial Intelligence and Applications. - Innsbruck, Austria : ACTA Press, 2008. - pp. 358-363. - ISBN:978-0-88986-710-9.

Sirageldin A., Baharudin B. et Jung Low Tang Detecting malicious executable file via graph comparison using support vector machine [Article] // Computer Information Science (ICCIS), 2012 International Conference on. - Kuala Lumpeu : IEEE, 12-14 Juin 2012. - Vol. 1. - pp. 469-473. - ISBN:978-1-4673-1937-9.

Tahan Gil, Rokach Lior et Shahar Yuval Mal-ID: Automatic Malware Detection Using Common Segment Analysis and Meta-features [Revue] // Journal of Machine Learning Research. - [s.l.] : JMLR.org, Janvier 2012. - Vol. 13. - pp. 949-979. - ISSN:1532-4435.

Tandon Gaurav et Chan Philip Learning Rules from System Call Arguments and Sequences for Anomaly Detection. [Article] // ICDM Workshop on Data Mining for Computer Security (DMSEC). - [s.l.] : IEEE Press, 2003. - pp. 20-29.

Tian Ronghua [et al.] Differentiating malware from cleanware using behavioural analysis [Article] // Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on. - 19-20 October 2010. - pp. 23-30. - ISBN:978-1-4244-9353-1.

Tilborg Henk C.A. et Jajodia Sushil Encyclopedia of Cryptography and Security [Ouvrage]. - [s.l.] : Springer, 2011. - ISBN:978-1-4419-5906-5.

Tong Simon et Koller Daphne Support vector machine active learning with applications to text classification [Revue] // J. Mach. Learn. Res.. - [s.l.] : JMLR.org, 1 March 2002. - Vol. 2. - pp. 45-66.

Wagener Gérard, State Radu et Dulaunoy Alexandre Malware behaviour analysis [Revue] // Journal in Computer Virology. - [s.l.] : Springer-Verlag, 2008. - 4 : Vol. 4. - pp. 279-287. - ISSN:1772-9890.

Wang Jau-Hwang et Deng Peter S. Virus detection using data mining techinques [Conférence] // Security Technology, 2003. Proceedings. IEEE 37th Annual 2003 International Carnahan Conference on. - 2003. - pp. 71-76. - ISBN:0-7803-7882-2.

Wang Ran et Kwong Sam Sample selection based on maximum entropy for support vector machines [Conférence] // Machine Learning and Cybernetics (ICMLC). - Qingdao : IEEE, 2010. - Vol. 3. - pp. 1390-1395. - ISBN:978-1-4244-6526-2.

Wang Wei [et al.] A hierarchical artificial immune model for virus detection [Conférence] // International Conference on Computational Intelligence and Security. - 2009. - pp. 1-5.

Wang Wei, Zhang Pengtao et Tan Ying An Immune Concentration Based Virus Detection Approach Using Particle Swarm Optimization [Revue] // LNCS. - [s.l.] : Springer-Verlag Berlin Heidelberg, 2010. - 6145 : Vol. 1. - pp. 347-354.

Willems C., Holz T. et Freiling F. Toward Automated Dynamic Malware Analysis Using CWSandbox [Revue] // Security Privacy, IEEE. - [s.l.] : IEEE, Mars 2007. - 2 : Vol. 5. - pp. 32-39. - ISSN:1540-7993.

Witten Ian H., Frank Eibe et Hall Mark A. Data Mining: Practical Machine Learning Tools and Techniques (Third Edition). [Ouvrage]. - [s.l.] : Morgan Kaufmann, 2011. - ISBN:978-0-12-374856-0.

Zhang Boyun [et al.] Malicious Codes Detection Based on Ensemble Learning [Section] // Autonomic and Trusted Computing. - [s.l.] : Springer Berlin Heidelberg, 2007. - Vol. 4610. - ISBN:978-3-540-73546-5.

Zhao HengLi [et al.] Unknown Malware Detection Based on the Full Virtualization and SVM [Article] // Management of e-Commerce and e-Government, 2009. ICMECG '09. International Conference on. - Nanchang : IEEE, 16-19 Septembre 2009. - pp. 473-476. - ISBN:978-0-7695-3778-8.

ANNEXES.

Annexe I . LES FICHIERS DE CONFIGURATION DU *CUCKOO SANDBOX*

❖ **Cuckoo.conf**

```
[cuckoo]
# Enable or disable startup version check. When enabled, Cuckoo will connect
# to a remote location to verify whether the running version is the latest
# one available.
version_check = off
# If turned on, Cuckoo will delete the original file and will just store a
# copy in the local binaries repository.
delete_original = on
# Specify the name of the machine manager module to use, this module will
# define the interaction between Cuckoo and your virtualization software
# of choice.
machine_manager = virtualbox
# Enable creation of memory dump of the analysis machine before shutting
# down. Even if turned off, this functionality can also be enabled at
# submission. Currently available for: VirtualBox and libvirt modules (KVM).
memory_dump = off
[resultserver]
# The Result Server is used to receive in real time the behavioral logs
# produced by the analyzer.
# Specify the IP address of the host. The analysis machines should be able
# to contact the host through such address, so make sure it's valid.
ip = 192.168.1.141
# Specify a port number to bind the result server on.
port = 2045
# Should the server write the legacy CSV format?
```

```
# (if you have any custom processing on those, switch this on)
store_csvs = off
# Maximum size of uploaded files from VM (screenshots, dropped files, log)
# The value is expressed in bytes, by default 10Mb.
upload_max_size = 10485760
[processing]
# Set the maximum size of analysis's generated files to process.
# This is used to avoid the processing of big files which can bring memory
leak.
# The value is expressed in bytes, by default 100Mb.
analysis_size_limit = 104857600
# Enable or disable DNS lookups.
resolve_dns = on
[database]
# Specify the database connection string.
# Examples, see documentation for more:
# sqlite:///foo.db
# postgresql://foo:bar@localhost:5432/mydatabase
# mysql://foo:bar@localhost/mydatabase
# If empty, default is a SQLite in  db/cuckoo.db.
connection =
# Database connection timeout in seconds.
# If empty, default is set to 60 seconds.
timeout =
[timeouts]
# Set the default analysis timeout expressed in seconds. This value will be
# used to define after how many seconds the analysis will terminate unless
# otherwise specified at submission.
default = 120
# Set the critical timeout expressed in seconds. After this timeout is hit
# Cuckoo will consider the analysis failed and it will shutdown the machine
# no matter what. When this happens the analysis results will most likely
# be lost. Make sure to have a critical timeout greater than the
# default timeout.
critical = 600
# Maximum time to wait for virtual machine status change. For example when
# shutting down a vm. Default is 300 seconds.
vm_state = 300
```

```
[sniffer]
# Enable or disable the use of an external sniffer (tcpdump) [yes/no].
enabled = yes
# Specify the path to your local installation of tcpdump. Make sure this
# path is correct.
tcpdump = /usr/sbin/tcpdump
# Specify the network interface name on which tcpdump should monitor the
# traffic. Make sure the interface is active.
interface = vboxnet0
[graylog]
# Enable or disable remote logging to a Graylog2 server.
enabled = no
# Graylog2 server host.
host = localhost
# Graylog2 server port.
port = 12201
# Default logging level for Graylog2. [debug/info/error/critical].
level = error
```

- **Virtualbox.conf**

```
[virtualbox]
# Specify which VirtualBox mode you want to run your machines on.
# Can be "gui", "sdl" or "headless". Refer to VirtualBox's official
# documentation to understand the differences.
mode = gui
# Path to the local installation of the VBoxManage utility.
path = /usr/bin/VBoxManage
# Specify a comma-separated list of available machines to be used. For each
# specified ID you have to define a dedicated section containing the
details
# on the respective machine. (E.g. cuckoo1,cuckoo2,cuckoo3)
machines = cuckoo1,cuckoo2,cuckoo3,cuckoo4,cuckoo5,cuckoo6,cuckoo7
[cuckoo1]
# Specify the label name of the current machine as specified in your
# VirtualBox configuration.
label = cuckoo1
# Specify the operating system platform used by current machine
# [windows/darwin/linux].
platform = windows
```

```
# Specify the IP address of the current machine. Make sure that the IP address
# is valid and that the host machine is able to reach it. If not, the analysis
# will fail.
ip = 192.168.56.101

[cuckoo2]
# Specify the label name of the current machine as specified in your
# VirtualBox configuration.
label = cuckoo2
# Specify the operating system platform used by current machine
# [windows/darwin/linux].
platform = windows
# Specify the IP address of the current machine. Make sure that the IP address
# is valid and that the host machine is able to reach it. If not, the analysis
# will fail.
ip = 192.168.56.102

[cuckoo3]
# Specify the label name of the current machine as specified in your
# VirtualBox configuration.
label = cuckoo3
# Specify the operating system platform used by current machine
# [windows/darwin/linux].
platform = windows
# Specify the IP address of the current machine. Make sure that the IP address
# is valid and that the host machine is able to reach it. If not, the analysis
# will fail.
ip = 192.168.56.103

[cuckoo4]
# Specify the label name of the current machine as specified in your
# VirtualBox configuration.
label = cuckoo4
# Specify the operating system platform used by current machine
# [windows/darwin/linux].
platform = windows
```

```
# Specify the IP address of the current machine. Make sure that the IP address
# is valid and that the host machine is able to reach it. If not, the analysis
# will fail.
ip = 192.168.56.104

[cuckoo5]
# Specify the label name of the current machine as specified in your
# VirtualBox configuration.
label = cuckoo5
# Specify the operating system platform used by current machine
# [windows/darwin/linux].
platform = windows
# Specify the IP address of the current machine. Make sure that the IP address
# is valid and that the host machine is able to reach it. If not, the analysis
# will fail.
ip = 192.168.56.105

[cuckoo6]
# Specify the label name of the current machine as specified in your
# VirtualBox configuration.
label = cuckoo6
# Specify the operating system platform used by current machine
# [windows/darwin/linux].
platform = windows
# Specify the IP address of the current machine. Make sure that the IP address
# is valid and that the host machine is able to reach it. If not, the analysis
# will fail.
ip = 192.168.56.106

[cuckoo7]
# Specify the label name of the current machine as specified in your
# VirtualBox configuration.
label = cuckoo7
# Specify the operating system platform used by current machine
# [windows/darwin/linux].
platform = windows
```

```
# Specify the IP address of the current machine. Make sure that the IP address

# is valid and that the host machine is able to reach it. If not, the analysis

# will fail.

ip = 192.168.56.107
```

- **processing.conf#**

```
Enable or disable the available processing modules [on/off].

# If you add a custom processing module to your Cuckoo setup, you have to add

# a dedicated entry in this file, or it won't be executed.

# You can also add additional options under the section of your module and

# they will be available in your Python class.

[analysisinfo]

enabled = yes

[behavior]

enabled = yes

[debug]

enabled = yes

[dropped]

enabled = yes

[network]

enabled = yes

[static]

enabled = yes

[strings]

enabled = yes

[targetinfo]

enabled = yes

[virustotal]

enabled = yes

# Add your VirusTotal API key here. The default API key, kindly provided

# by the VirusTotal team, should enable you with a sufficient throughput

# and while being shared with all our users, it shouldn't affect your use.

key = b453ef099f5b8bb9ab25506cd19b187e23b6dcf677f5532918d5eaa0ae82ddc6
```

- **reporting.conf**

```
# Enable or disable the available reporting modules [on/off].

# If you add a custom reporting module to your Cuckoo setup, you have to add

# a dedicated entry in this file, or it won't be executed.
```

```
# You can also add additional options under the section of your module and
# they will be available in your Python class.
[jsondump]
enabled = on
[reporthtml]
enabled = on
[pickled]
enabled = off
[metadata]
enabled = off
[maec11]
enabled = off
[mongodb]
enabled = off
[hpfclient]
enabled = off
port = 10000
```

Annexe II . QUELQUES CARACTERISTIQUES DE PROGRAMME (N-GRAMME D'APPELS API)

- **n-gramme d'appels API fréquents dans les virus**
  - `RegOpenKeyExA`
  - `CreateThread-->NtReadFile-->NtQueryInformationFile-->NtCreateFile`
  - `NtQueryInformationFile-->NtCreateSection-->ZwMapViewOfSection`
  - `DeviceIoControl-->RegOpenKeyExA`
  - `NtReadFile-->ZwMapViewOfSection-->NtQueryInformationFile-->NtCreateFile-->NtOpenFile`
  - `NtOpenFile-->NtSetContextThread-->CreateThread-->NtResumeThread-->NtDelayExecution`
  - `ExitProcess-->FindWindowA-->NtOpenKey-->NtQueryValueKey`
  - `WriteProcessMemory-->NtQueryValueKey-->NtReadFile`
  - `RegDeleteKeyA-->NtCreateFile-->NtSetInformationFile-->ZwMapViewOfSection-->NtCreateFile`
  - `NtWriteFile-->NtQueryDirectoryFile-->NtOpenFile`
  - `RegDeleteKeyA-->LdrLoadDll-->NtCreateFile-->NtQueryInformationFile-->NtCreateFile`
  - `CreateProcessInternalW-->NtFreeVirtualMemory-->NtCreateMutant-->CreateThread`
  - `NtSetContextThread-->CreateThread-->NtResumeThread-->NtDelayExecution-->RegOpenKeyExA`
  - `CreateProcessInternalW-->NtCreateSection-->ZwMapViewOfSection`
  - `LdrLoadDll-->NtCreateFile-->NtQueryInformationFile-->DeleteFileA-->DeleteFileA`
  - `WriteProcessMemory-->NtSetInformationFile`
  - `CopyFileA-->NtDeviceIoControlFile-->NtQueryInformationFile-->NtCreateFile`
  - `WriteProcessMemory-->NtWriteFile-->NtCreateFile`
  - `WriteConsoleW-->NtOpenSection-->NtQueryInformationFile-->NtReadFile`
  - `NtCreateSection-->NtSetInformationFile-->NtReadFile`
  - `ZwMapViewOfSection`
  - `WriteConsoleW-->NtOpenSection-->NtQueryInformationFile-->NtReadFile-->NtCreateFile`
  - `WriteProcessMemory-->NtWriteFile-->NtCreateFile-->NtQueryInformationFile`

- **NtSetInformationFile-->NtReadFile-->NtQueryInformationFile-->NtOpenFile**
- **NtReadFile-->NtSetInformationFile-->ZwMapViewOfSection**
- **WriteConsoleW-->FindFirstFileExW-->FindFirstFileExW-->LdrLoadDll-->LdrGetProcedureAddress**
- **WriteConsoleA-->RegOpenKeyExA-->RegQueryValueExA-->RegCloseKey**
- **FindWindowW-->RegOpenKeyExA-->RegQueryValueExA-->RegCloseKey**
- **RegEnumKeyW-->NtCreateMutant-->CreateThread-->RegOpenKeyExA**
- **RegCloseKey-->NtDeviceIoControlFile-->NtCreateFile-->NtSetInformationFile**
- **NtCreateFile-->NtCreateMutant-->CreateThread-->FindFirstFileExW**
- **RegDeleteKeyA-->NtSetInformationFile-->NtCreateFile-->NtReadFile**
- **RegQueryValueExA-->RegQueryValueExA**
- **GetSystemMetrics-->NtCreateFile**
- **NtCreateFile-->CreateThread-->RegOpenKeyExA**
- **WriteConsoleW-->NtOpenSection-->NtQueryInformationFile**
- **NtWriteFile-->NtCreateFile-->NtFreeVirtualMemory**
- **RegOpenKeyExA-->NtQueryValueKey**
- **WriteProcessMemory-->NtWriteFile-->NtCreateFile-->NtQueryInformationFile-->NtSetInformationFile**
- **WriteProcessMemory-->NtWriteFile-->NtCreateFile-->NtQueryInformationFile-->ZwMapViewOfSection**
- **NtWriteFile-->NtCreateFile-->NtFreeVirtualMemory-->NtCreateSection-->NtFreeVirtualMemory**
- **NtSetInformationFile-->NtReadFile-->NtOpenFile**
- **WriteProcessMemory-->NtWriteFile-->NtCreateFile-->NtQueryInformationFile-->FindFirstFileExW**
- **NtOpenFile**
- **NtSetInformationFile**
- **NtSetInformationFile-->NtReadFile**
- **WriteProcessMemory-->NtSetInformationFile-->ZwMapViewOfSection**
- **NtCreateFile-->NtCreateMutant-->CreateThread-->FindFirstFileExW-->FindFirstFileExW**
- **NtCreateFile-->CreateThread-->RegOpenKeyExA-->RegQueryValueExA**
- **NtDeviceIoControlFile-->NtOpenFile-->RegOpenKeyExA**
- **WriteProcessMemory-->NtSetInformationFile-->ZwMapViewOfSection-->NtCreateFile**
- **NtCreateSection-->NtReadFile-->NtResumeThread-->NtCreateFile-->NtQueryInformationFile**
- **NtCreateSection-->NtReadFile-->FindFirstFileExW-->FindFirstFileExW**
- **NtSuspendThread-->NtCreateFile-->NtQueryInformationFile**
- **RegEnumKeyW-->NtCreateSection-->NtFreeVirtualMemory-->NtFreeVirtualMemory**

- n-gramme d'appels API fréquents dans les programmes bienveillants

- **ecvfrom-->LdrLoadDll-->LdrGetProcedureAddress-->RegOpenKeyExW-->RegOpenKeyExW**
- **NonSetWindowsHookExW-->GetSystemMetrics-->GetSystemMetrics-->LdrGetDllHandle-->RegOpenKeyExW**
- **NonCreateDirectoryW-->NtOpenSection-->NtOpenSection**
- **egDeleteKeyA-->LdrGetProcedureAddress-->LdrGetProcedureAddress-->GetSystemMetrics**
- **egDeleteKeyA-->FindFirstFileExW-->RegCreateKeyExW**
- **NonCreateDirectoryW-->NtOpenSection-->NtOpenSection-->RegOpenKeyExW**
- **NonNtCreateFile-->NtCreateMutant-->RegOpenKeyExW-->GetSystemMetrics**

- egDeleteKeyA-->NtOpenFile-->GetSystemMetrics-->RegOpenKeyExW
- NonCreateDirectoryW-->NtOpenSection-->NtOpenSection-->RegOpenKeyExW-->RegQueryValueExW
- NonCreateDirectoryW-->NtOpenSection-->LdrGetProcedureAddress-->LdrGetProcedureAddress
- NonCreateDirectoryW-->NtOpenSection-->RegEnumKeyExW-->LdrGetProcedureAddress-->LdrGetProcedureAddress
- egDeleteKeyA-->FindFirstFileExW-->RegCreateKeyExW-->RegQueryValueExW
- NonNtCreateFile-->NtCreateMutant-->GetSystemMetrics-->CreateThread-->LdrLoadDll
- egDeleteKeyA-->FindFirstFileExW-->RegCreateKeyExW-->RegQueryValueExW-->RegCloseKey
- NonGetSystemMetrics-->GetSystemMetrics-->GetSystemMetrics-->LdrGetProcedureAddress
- egDeleteKeyA-->FindFirstFileExW-->RegOpenKeyExW-->RegQueryValueExW
- egDeleteKeyA-->LdrGetProcedureAddress-->NtOpenKey-->NtQueryValueKey-->NtCreateFile
- NonLookupPrivilegeValueW-->CreateProcessInternalW-->IsDebuggerPresent-->LdrLoadDll-->GetSystemMetrics
- NonUnhookWindowsHookEx-->RegCreateKeyExW-->FindFirstFileExW-->FindFirstFileExW
- NonUnhookWindowsHookEx-->RegCreateKeyExW-->FindFirstFileExW-->FindFirstFileExW-->FindFirstFileExW
- NonNtOpenSection-->ZwMapViewOfSection-->SetWindowsHookExA
- Nonselect-->LdrGetProcedureAddress-->GetSystemMetrics-->GetSystemMetrics-->GetSystemMetrics
- NonGetSystemMetrics-->GetSystemMetrics-->GetSystemMetrics-->LdrGetProcedureAddress-->GetSystemMetrics
- Nonselect-->GetSystemMetrics-->GetSystemMetrics-->LdrGetProcedureAddress
- egDeleteKeyA-->LdrGetProcedureAddress-->NtOpenKey-->NtQueryValueKey-->CreateThread
- egDeleteKeyA-->RegQueryValueExW-->RegCloseKey-->RegCreateKeyExW-->NtOpenFile
- egDeleteKeyA-->RegQueryValueExW-->RegCloseKey-->RegCreateKeyExW-->LdrGetDllHandle
- egOpenKeyExW-->NtCreateMutant-->NtOpenSection-->RegOpenKeyExW
- egDeleteKeyA-->RegQueryValueExW-->RegCreateKeyExW-->RegCloseKey-->LdrLoadDll
- egDeleteKeyA-->RegCreateKeyExW-->RegCreateKeyExW-->RegQueryValueExW-->RegCloseKey
- NonCreateDirectoryW-->RegCloseKey-->RegQueryValueExW-->RegCloseKey-->NtCreateFile
- ecvfrom-->LdrGetProcedureAddress-->LdrGetProcedureAddress-->RegOpenKeyExW-->RegQueryValueExW
- Nonselect-->FindWindowA-->LdrGetProcedureAddress
- Nonselect-->FindWindowA-->LdrGetProcedureAddress-->LdrGetProcedureAddress
- egOpenKeyExW-->NtCreateMutant-->NtOpenSection-->RegOpenKeyExW-->RegQueryValueExW
- NonCreateProcessInternalW-->NtEnumerateValueKey
- egOpenKeyExW-->RegQueryValueExW-->RegCloseKey-->GetSystemMetrics
- egDeleteKeyA-->RegCreateKeyExW-->RegCreateKeyExW-->RegCloseKey
- NonNtFreeVirtualMemory-->RegEnumKeyW-->NtFreeVirtualMemory-->RegCreateKeyExW

- **egQueryInfoKeyA-->IsDebuggerPresent-->RegQueryValueExW--**
  >RegCreateKeyExW
- **Nonselect-->FindWindowA-->LdrGetProcedureAddress--**
  >LdrGetProcedureAddress-->LdrGetProcedureAddress
- **NonNtCreateMutant-->RegOpenKeyExW-->RegQueryValueExW-->RegCloseKey--**
  >LdrGetDllHandle
- **Nonselect-->FindWindowA-->LdrGetProcedureAddress-->GetSystemMetrics**
- **Nonselect-->FindWindowA-->LdrGetProcedureAddress-->GetSystemMetrics--**
  >GetSystemMetrics
- **NonCreateProcessInternalW-->NtOpenKey-->NtQueryValueKey--**
  >CreateThread
- **NonFindWindowW-->LdrGetDllHandle-->LdrLoadDll-->FindWindowA**
- **NonFindWindowW-->LdrGetDllHandle-->LdrLoadDll-->FindWindowA--**
  >GetSystemMetrics