Les Agents logiciels mobiles et la sécurité

AOUADI Hamed
Laboratoire RIADI, ENSI, Campus universitaire la Manouba, 2010 Tunis, Tunisie
Hamed_aouadi@yahoo.fr

PR BEN AHMED Mohamed
Laboratoire RIADI, ENSI, Campus universitaire la Manouba, 2010 Tunis, Tunisie
Mohamed.benahmed@riadi.rnu.tn

1. Introduction

Les dernières décennies ont été marquées par une révolution dans le domaine de l'informatique distribuée et des télétraitements. L'une des technologies prometteuses est celle des agents logiciels mobiles. Cette dernière est identifiée comme la future plate-forme de base pour une architecture de services électroniques [Hoh 97] et qui pourra s'échapper des contraintes d'hétérogénéité des plates-formes cibles en raison de l'adaptabilité qui caractérise les agents mobiles.

Un agent mobile est un logiciel particulier, doté des capacités d'adaptabilité, d'autonomie et de mobilité [Wil 98]. Cet agent se déplace à travers le réseau (en particulier l'Internet) d'un site vers un autre dans le but de fournir un service précis à son propriétaire. Toutefois le déplacement à travers le réseau n'est pas toujours sans risques, en effet l'agent peut être attaqué par un autre agent, par une personne connectée au réseau ou par un site visité sur lequel l'agent devra s'exécuter [Hoh 97]. Notre travail se propose d'étudier les problèmes de sécurisation des agents logiciels mobiles et de proposer une solution hybride robuste tout en étant à coût abordable.

2. Concept d'agent mobile :

Par sa définition, un agent mobile migre à sa propre initiative d'une plate-forme vers une autre pour l'accomplissement d'un but particulier, défini par son propriétaire. Cette définition met en évidence (comme le montre la figure fig1) l'existence d'une plate-forme source, d'une plate-forme cible et d'une infrastructure de transport d'agents [Wil98]. Une plate-forme d'exécution est composée du matériel, du système d'exploitation et de la plate-forme agent qui peut être matérialisée par un interpréteur, une machine virtuelle ou un run-time. La plate-forme agent devra permettre l'exécution des agents ainsi que leur migration vers une autre plate-forme [Wil98].

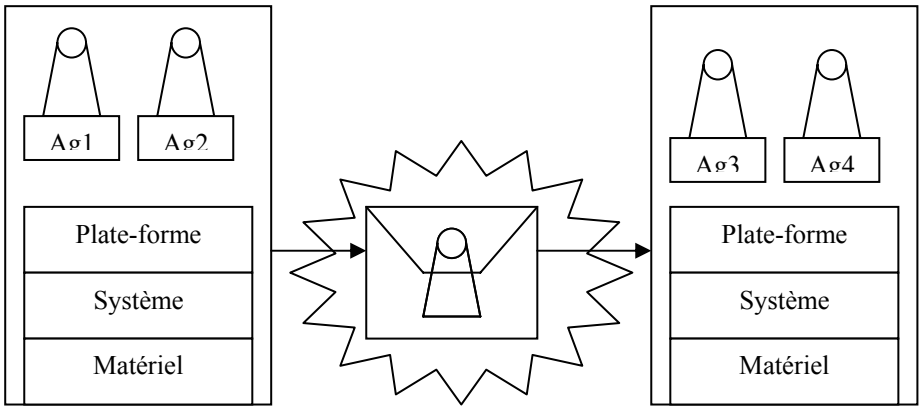

Fig1 : plate-forme d'agents mobiles

Afin de migrer vers un autre site, un agent doit traverser un réseau de communications sur lequel l'agent pourrait subir différents types d'attaques telles que l'écoute, l'altération de son contenu ou même sa destruction. Par ailleurs, au niveau de la plate-forme cible l'agent peut être attaqué par l'utilisateur de la plate-forme ou par d'autres agents. Dans le paragraphe suivant nous présenterons la problématique de la sécurité et les paragraphes qui suivent présenteront les différentes approches de sécurisation proposées.

3. Les problématiques de sécurité liés aux agents mobiles

Comme nous l'avons mentionné dans le paragraphe précédent, un agent mobile peut être cible de plusieurs types d'attaques du fait de son déplacement à travers des milliers de sites, qui peuvent ne pas être de confiance, et à travers un réseau qui n'est pas toujours sécurisé. En effet, au cours de son déplacement, l'agent interagit avec d'autres agents qui peuvent analyser ou altérer son contenu. Par ailleurs, le site visité (sur lequel l'agent s'exécute) a la possibilité de manipuler l'agent ou même de le détruire. En effet, ce dernier a un accès total à l'agent et pourra l'analyser bit par bit et par suite mener tout type d'attaques. Hohl [Hoh97] classe les aspects de sécurité concernant un agent mobile comme suit :

- La sécurité entre deux agents,
- la sécurité entre l'agent et sa plate-forme d'exécution,
- la sécurité entre deux machines et
- la sécurité entre agent et un intervenant externe.

4. Etat de l'art de la sécurisation des agents mobiles

Les approches de sécurisation des agents mobiles sont multiples. Ces dernières peuvent être classées, suivant le moyen de sécurisation, en sécurité basée sur le matériel et sécurité basée sur le logiciel.

4.1 Sécurité basée sur le matériel :

Ce type d'approche est largement utilisé dans le domaine de la sécurisation des logiciels contre le piratage. Dans ce cas, l'exécution du logiciel est reliée à l'existence d'un

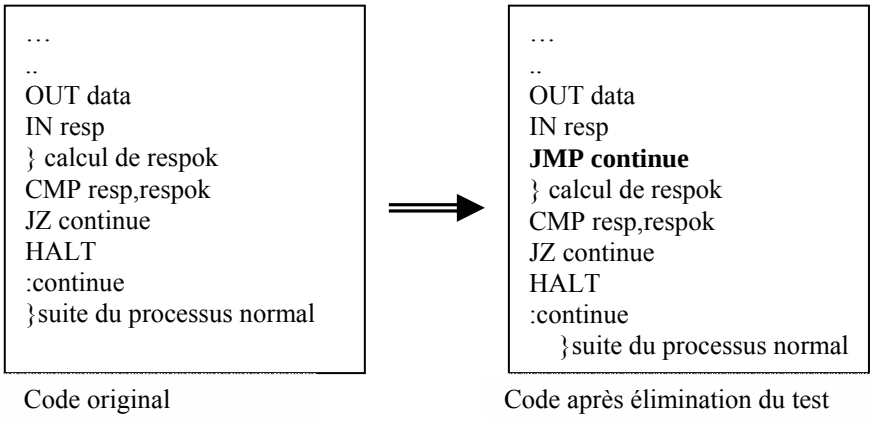

Fig2. Attaque d'un logiciel protégé par un périphérique

périphérique spécial (dongle) que le logiciel teste au début ou en cours d'exécution [Hoh 97]. La protection de ce logiciel peut être contournée par l'élimination (ou saut) de la partie test du code de l'agent[Man02].

4.2.1 Sécurisation à base d'un environnement d'exécution de confiance :

Une autre méthode de sécurisation consiste à utiliser un coprocesseur dédié à l'exécution de l'agent. L'agent s'exécute exclusivement à l'intérieur de ce périphérique et ne dialogue avec le site visité qu'à travers une interface sécurisée [Wil98]. Une approche de ce type a été proposée par Wilhelm [Wil98] et est basée sur ce qu'il appelle « environnement d'exécution de confiance TPE » (Trusted Processing Envirenment). Le périphérique TPE est fabriqué sous le contrôle d'une autorité de confiance et dispose d'un certificat et d'une politique de sécurité. Dans cette approche les agents sont cryptés par la machine de leur propriétaire et ne seront plus exécutés par le site visité mais par le périphérique TPE. Ainsi les agents resteront cryptés tout au long de leurs chemins de migration et sur chaque site visité. Quand ils arrivent à leurs sites de destinations ils sont encore cryptés et accèderont au périphérique TPE où ils seront décryptés puis exécutés [Wil98].

Une telle approche présente une solution très puissante pour sécuriser les agents sur site d'exécution ainsi qu'à travers le réseau. Toutefois, le périphérique TPE n'est pas facile à fabriquer ce qui explique son coût élevé. De plus l'environnement d'exécution de confiance TPE ne présente pas les performances d'un ordinateur, ce qui réduit l'efficacité d'exécution de l'agent. De plus le problème d'exécution de plusieurs agents en parallèle reste posé.

4.2.2 Sécurisation à base de cartes à puces :

Afin de rendre public le concept de périphérique spécial pour l'exécution de l'agent, Mana [Man02] propose l'utilisation de cartes à puces. Son idée consiste à subdiviser le code de l'agent en sections dont certaines seront cryptées par la clé publique d'une carte à puces. Ces dernières seront remplacées par des appels procéduraux vers la carte. Sur site d'exécution, on transmet à la carte comme arguments les sections cryptées qui seront décryptées puis exécutées à l'intérieur de cette dernière (Fig. 3). La paire de clés est générée à l'intérieur de la carte à puce, la clé publique sera publiée tandis que la clé secrète résidera exclusivement à l'intérieur de la carte et ne sera jamais révélée [Man02].

De cette façon on assure la confidentialité du code et on évite l'exécution de la totalité de l'agent sur le périphérique de sécurisation. En cas d'exécution de plusieurs agents sur site, la carte peut jouer le rôle d'une ressource critique à laquelle l'accès est géré par les moyens

connus. Malgré la robustesse apparente de l'approche, le code est caché en partie et ce qui reste est lisible ce qui permet l'analyse et la déduction des fonctionnalités du code clair ainsi que du code caché. La déduction des fonctionnalités des sections cachées permet une attaque à boite noire [BL96]. Les sections cachées peuvent être appelées par un code écrit par celui qui attaque en vue de réaliser des tâches au nom de l'agent et sur le compte de son propriétaire (par exemple la fonction signature peut être exploitée pour signer un document autre que l'agent a choisi).

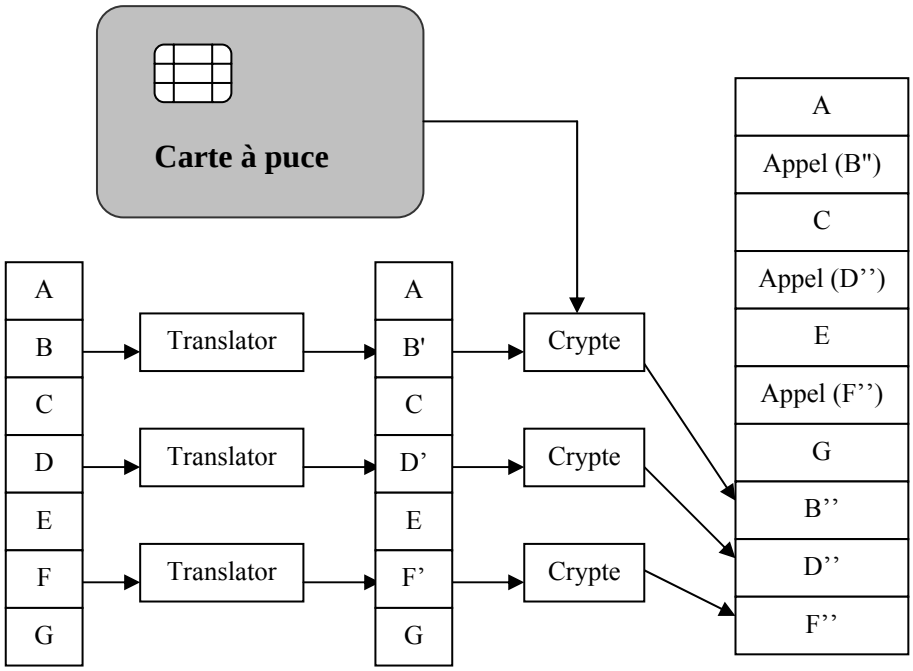

Fig. 3 : Sécurisation à base de cartes à puces

4.2 Sécurisation basée sur le logiciel

Dans ce type d'approches, on cherche à sécuriser les agents de façon purement logicielle ce qui réduit le coût de la sécurité et facilite sa maintenance.

4.2.1 Obscurcissement

Comme travaux nous présentons l'approche proposée par Fritz Hohl [Hoh97] dans laquelle il présente une solution de sécurisation de l'agent par son propre code. L'approche consiste à produire d'un agent A un agent B analogue au premier sur le plan fonctionnalités mais ayant un code difficile à analyser [Wil98]. Ceci peut être assuré par le fait d'introduire un désordre au niveau du code de l'agent. Hohl propose la violation totale des règles du génie logiciel afin de produire un code non lisible. Parmi ces règles nous citons :

- Attribuer des noms significatifs aux variables.
- Ecrire un code modulaire.
- Utiliser des structures de contrôle qui simplifient le programme.

Ces règles seront respectées au niveau de l'agent source A et pour passer à l'agent sécurisé B, tout d'abord on crée à partir des variables originelles de nouvelles variables ayant des noms non significatifs et un nombre différent. Comme indiqué dans la figure Fig4, chaque nouvelle variable est composée de fragments de quelques variables originelles.

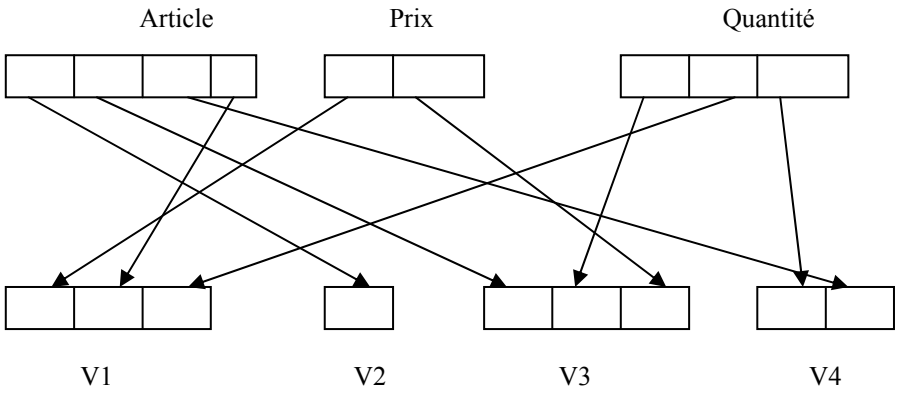

Fig. 4 : génération de variables à noms non significatifs

Après l'étape de génération des variables, on procède à la déstructuration du code. Pour cela on élimine les variables locales et on les remplace par des variables globales, on remplace

l’appel procédural par le corps des procédures et on utilise la structure de contrôle "GOTO" pour remplacer les autres structures [Hoh97]. En fin, on pourra insérer des fragments de code mort pour améliorer la protection. Seulement il faut se méfier des mécanismes de détection de codes morts.

Ces mécanismes assurent la protection de la confidentialité des agents. Pour assurer l'intégrité de ces derniers, Hohl propose l'utilisation de la signature électronique de l'agent en totalité et en association avec sa date de validité après laquelle l'agent ne sera plus accepté par aucun site et ses actions ne seront plus valides [Hoh97].

Dans cette approche, on a pu retrouver une solution logicielle efficace pour sécuriser les agents. Toutefois, on peut lui reprocher le manque de fondement théorique. De plus la sécurité de l'agent est temporaire et n'est valide que pour les agents qui transportent des données à courte durée de validité. Autrement, l'agent sera enregistré et analysé lentement afin de déduire les données qu'il transporte. Et enfin, comme mentionne Chenghui [Che2002] l'élimination de l'appel procédural entraîne la perte de l'utilisation des bibliothèques sur site d'exécution.

4.2.2 Calcul par fonction cryptographique

L'objectif de cette approche est analogue à celui de l'approche précédente. Mais le principe est plus développé et dispose d'un fondement mathématique robuste et la sécurité n'est pas liée à une durée de vie. L’idée proposée par Sander et Tschudin [ST 98] est la suivante : soit une fonction f matérialisée par un agent A, cette fonction sera cryptée en E(f) qui cache les fonctionnalités et les détails de f. Un programme P(E(f)) sera écrit pour implémenter E(f), ce qui produit un nouvel agent B. L’agent B migre sur un site distant où il sera exécuté sur une donnée x et retourne à son site d’origine qui exécute l’algorithme de décryptage E^{-1} (P(E(f)))(x)=f(x). Au niveau du site distant, on exécute P(E(f))(x) qui cache les détails ce qui assure la sécurité de l’agent. Cette approche définit un homomorphisme entre l’espace des données claire et celui des données cryptées tout en se basant sur la fonction PLUS et la

fonction MIXED-MULT. Ces deux fonctions sont valides uniquement sur les polynômes, ce qui représente la défaillance de l'approche de Sander sur les données ordinaires. En plus le travail de l'agent se limite au calcul d'un résultat sur un site distant puis l'agent retourne vers son site d'origine, ce qui ne permet pas la négociation sur site ni la possibilité de signature ou de prise de décision.

4.2.3 Traces cryptographiques

L'objectif de cette approche est de vérifier l'intégrité d'exécution des agents après leurs retour. Pour cela, chaque site visité, génère une trace d'exécution de l'agent. Cette trace contient toute ligne de code exécutée ainsi que toutes les valeurs externes lues par l'agent [VIG98]. Avant la migration de l'agent vers une nouvelle destination, le site calcule par fonction de hachage (exemple MD5 ou SHA [FIPS180]) une représentation condensée de la trace, la signe et l'accompagne à l'agent mobile lors de sa transmission vers le site suivant. Afin de réduire la taille de la trace, l'auteur divise le code de l'agent en segments blancs et segments noirs. Les segments noirs sont ceux qui résultent des interactions avec la plate-forme visitée. Et au lieu de la trace d'exécution du code entier, on génère uniquement la trace d'exécution des segments noirs [Lau 01].

Après retour de l'agent, son propriétaire aura une trace de l'exécution de l'agent sur le dernier site et l'adresse du premier site visité (vers lequel il a envoyé l'agent). De cette façon si le propriétaire a un doute envers un site donné, il pourra suivre le chemin de l'agent et avoir la trace d'exécution sur ce site. Le propriétaire de l'agent pourra simuler son exécution sur le site duquel il doute et comparer la simulation avec la trace reçue [Vig98].

Cette approche permet d'assurer la non répudiation et la détection de toute manipulation de l'exécution de l'agent après son retour. Toutefois l'approche reste détective et n'évite en aucun cas l'utilisation frauduleuse de l'agent. De plus il faut avoir un doute envers un site particulier pour lancer le mécanisme de trace, sinon l'utilisation systématique de ce mécanisme devient assez coûteuse.

4.2.4 Agents coopérants

La sécurisation d'agents à base de clones a été proposée par Roth, Schneider et al. [Sch & al 97]. La protection est assurée en faisant partager l'information par plusieurs agents dit *clones*. La tâche demandée par l'utilisateur est réalisé par plusieurs agents qui coopèrent plutôt que par un seul agent. L'objectif principal est de protéger l'information partagée même si plusieurs sites d'exécution collaborent. Dans un exemple explicatif Roth [Rot 99] définit deux

sous-groupes indépendants de sites que l'agent visitera. Basés le clonage, deux agents seront créés et envoyés chacun à l'un de ces sous-groupes. Si un agent retrouve une information pertinente il l'envoie vers son clone qui la vérifie. Pour avoir plus d'efficacité, l'un des deux agents s'exécute obligatoirement sur un site de confiance. L'inconvénient majeur de cette approche est l'abus d'utilisation des ressources réseau pour assurer la communication entre clones. De plus l'approche se base sur l'hypothèse de non-collaboration entre les différents sites marchands ce qui n'est pas évident. Enfin l'approche protège uniquement les données transportées et non l'agent en totalité.

4.2.5 Appréciation d'état

Dans cette approche Farmer et al. [FGS96a] définissent un mécanisme qui permet à un agent d'évaluer les privilèges dont il dispose sur un site particulier. Ce qui permet au propriétaire de l'agent de limiter les actions que l'agent peut effectuer. L'approche repose sur une fonction protégée qui permet l'évaluation de l'état de l'agent sur chaque site visité. La fonction sera exécutée une fois l'agent arrive sur un site donné et permettra de vérifier la stabilité de son état. L'existence de telle fonction protège l'agent en détectant les manipulations de son état. La fonction repose sur un calcul complexe à partir d'un ensemble de variables d'état. Une amélioration a été proposée par Jansen [Jan01][Jan2 01] [Jan3 00] et qui consiste a séparer la structure de données définissant le comportement de l'agent de son propre code. Cette structure sera définie dans un certificat conforme à la norme X509 [ISO9594-8]. Dans le certificat, on définit les droits et les responsabilités d'un agent sur un site donné. On distingue les certificats d'attributs dans lesquels on définit le comportement de l'agent et les certificats de politique servant à définir le comportement du site envers tous les agents qu'il accueille. L'approche protège l'agent contre une utilisation illicite en fournissant une preuve des intentions réelles de son propriétaire mais n'offre aucune protection des données que l'agent transporte. Une autre approche proposée par Magdy Sayeb [Mag02] consiste à utiliser un arbre de diagnostic permettant la détection d'une attaque possible. Dans cette approche les auteurs associent un symptôme à chaque attaque et la combinaison de ces symptômes définit un arbre. Comme symptômes nous citons : la longue durée d'exécution qui informe sur une analyse du code ou des données au cours de l'exécution, aussi l'enregistrement temporaire de

l'agent informe sur une analyse possible tandis qu'un comportement anormal de l'agent informe sur la manipulation du code de l'agent. L'approche se limite aux attaques dont l'origine ne prend pas en compte un éventuel mécanisme de détection, sinon il cherchera à masquer les symptômes et simuler un comportement normal.

4.2.6 Fonction de validation

Les approches à base de fonctions de validation ont été proposées par Blum [Blu88] afin de vérifier la fiabilité d'un code. Ces dernières peuvent être appliquées dans le but de vérifier l'intégrité d'exécution d'un agent mobile sur un site distant. L'approche se base sur la propriété de réductibilité de la fonction qu'implémente l'agent. Une approche plus récente proposée par Laurero [Lau 01] définit une fonction de vérification V comme suit : Soient un programme P qui implémente une fonction f , Y l'ensemble des résultats possibles de $f(x_i)$, avec x_i appartient à $\{0,1\}^n$ et D(y') le résultat reçu après exécution à distance. La fonction V satisfait la condition suivante si (y=D(y')) n'appartient pas à Y alors P(V(y')= accept) < δ, où P représente la probabilité et δ la probabilité d'erreur.

5. Comparaison entre les différentes approches :

La comparaison entre les approches précédemment présentées sera faite en fonction des critères relatifs à la robustesse de la sécurité des agents et de ceux relatifs au coût de cette sécurité. Comme critères relatifs à la sécurité, nous examinerons tout d'abord les apports de chaque approche en terme de ***confidentialité d'exécution*** du code mobile sur chaque site visité, ensuite nous examinerons si l'approche en question assure ***l'intégrité d'exécution***. Une telle intégrité sera-t-elle assurée tout au long du cheminement de l'agent ou sera-t-elle juste vérifiée après son retour. Concernant les coûts de la solution de sécurité, nous nous intéresserons à l'augmentation de la ***taille du code***, à l'influence de l'approche de sécurité sur le ***temps d'exécution*** de l'agent et essentiellement aux ***communications additionnelles*** à travers le réseau, en raison des coûts de ces dernières et particulièrement des vulnérabilités qu'elles impliquent.

Tableau de comparaison entre les approches :

Critères / Approches	confidentialité d'exécution	intégrité d'exécution	taille du code	temps d'exécution	Communication additionnelle
Environnement d'exécution de confiance	forte	forte	Non modifiée	Dépend du TPE	Aucune
Cartes à puces	forte	forte	Ajout d'instructions d'appel de la carte	Faible ralentis-sement à cause de l'accès à la carte	Aucune
Obscurcissement	Forte pendant une courte durée de vie de l'agent	Forte pendant une courte durée de vie de l'agent	Relativement acceptable en cas d'insertion de code mort	Non modifié	Aucune
Calcul par fonction cryptographique	Forte	forte	Relativement acceptable	Relativement acceptable	L'agent doit retourner à son site après chaque offre
Traces d'exécution	Non assurée	Vérifiable	Augmentée de la taille de la trace associée	acceptable	Aucune
Agents coopérants	Non assurée	forte	Relativement acceptable	Fortement ralenti	Trop de communications additionnelles
Appréciation d'état	Non assurée	Partiellement assurée	Relativement acceptable	Relativement acceptable	Aucune
Fonctions de validation	Non assurée	Vérifiable	Relativement acceptable	Relativement acceptable	Aucune

6. Notre Approche

Dans l'approche proposée, nous cherchons à remplacer l'environnement d'exécution de confiance TPE proposé par Wilhelm par une solution logicielle et à protéger ce logiciel par le matériel afin d'obtenir une solution hybride (matérielle - logicielle) robuste, évolutive et à coût abordable. En effet, nous bénéficions d'une solution de sécurisation solide (analogue à celle que présente un périphérique), à des coûts abordables et avec des performances meilleures (en raison de l'exploitation du parallélisme des transactions). Pour cela nous proposons un environnement logiciel qui se charge d'exécuter l'agent. Cet environnement sera matérialisé par un interpréteur assimilé à une machine virtuelle. Un agent sera tout d'abord écrit d'une façon ordinaire, avec un langage ordinaire puis un transformateur (sorte d'obscurcisseur) génère dans le langage de la machine virtuelle, un nouvel agent analogue au premier de point de vue fonctionnalités, incompréhensible (crypté) et interprétable en fonction d'une clé par la machine virtuelle qu'on a appelé SVM (Secure Virtual Machine). Avant son exécution, des clones seront créés à partir de l'agent et envoyés vers les sites à visiter. De cette façon, nous augmentons la fiabilité par répartition de la tâche sur plusieurs agents, nous augmentons aussi l'efficacité du système par l'exécution en parallèle de plusieurs agents et nous protégeons la réponse qui ne sera plus rapportée par un seul agent visitant tous les sites, mais chaque clone transporte un fragment de la liste et par suite un site ne peut pas déterminer la meilleure offre reçue par le système globalement. Dans la suite nous présenterons les détails de notre approche.

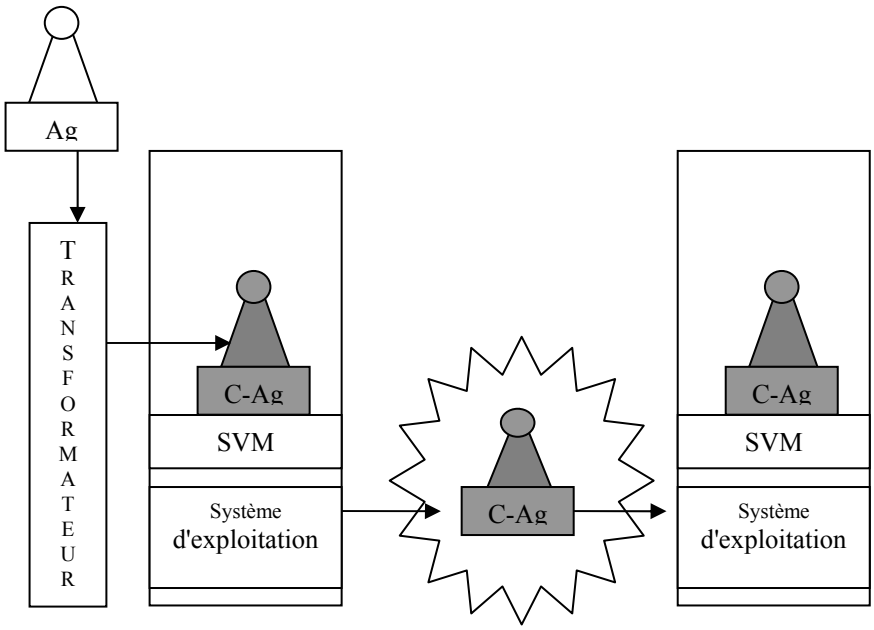

Fig 5 Notre approche

6.1 La machine virtuelle SVM

La machine virtuelle SVM est un logiciel installé (il peut être transporté avec l'agent pour être installé sur le site visité si ce dernier ne contient pas cette machine virtuelle) sur les sites qui seront visités par l'agent comme couche entre le système d'exploitation et les agents visiteurs. Ce logiciel aura pour rôle de recevoir l'agent totalement crypté et de créer une instance SVM propre à cet agent. L'instance se charge d'exécuter l'agent qui reste crypté dans un espace mémoire alloué proprement à cette instance SVM qu'on appellera environnement clos. Dans son environnement clos l'agent sera exécuté suivant un chemin d'exécution défini par la clé publique de l'agent. Au cours de son exécution l'agent contient une (plusieurs) instruction qui permet sa migration et le vidage de l'environnement clos (autodestruction de l'agent). Le vidage de l'environnement clos se fait suite à la demande de migration de l'agent ou à la détection d'une tentative de manipulation ou à l'expiration de la durée de vie de l'agent. La détection d'une éventuelle manipulation se base sur des détecteurs logiciels de manipulation. La notion d'environnement clos permettra ainsi la confidentialité de l'agent et son indépendance des autres agents de façon à simuler une exécution de chaque agent par machine (physique) autonome. Dans le cas d'un agent de recherche d'information sans prise de décision, chaque offre sera cryptée par la clé publique du propriétaire de l'agent avant d'être insérée dans la liste.

6.2 Sécurisation de la machine virtuelle SVM

La machine virtuelle SVM qui a pour objectif de fournir un environnement d'exécution de confiance est un logiciel installé sur un site potentiellement de non confiance et aura à son tour besoin d'être sécurisée. En revanche, la sécurisation de la machine virtuelle SVM consiste à la sécurisation d'un seul logiciel statique par contre sécuriser des agents multiples, mobiles et dynamiques (qui transportent des listes d'offres variables) est une tâche délicate et a priori sans solution acceptable. Puisque SVM est un logiciel installé sur un site marchand, on adoptera la solution proposée par Mana [Man02] qui consiste à subdiviser le code de la machine SVM (au lieu de celui de l'agent) en sections dont certaines jugées pertinentes seront

cryptées par la clé publique Kp d'une carte à puce. Ces dernières seront remplacées par des appels procéduraux à la carte. Sur site d'exécution, on transmet à la carte comme arguments les sections cryptées qui seront décryptées (Ks) puis exécutées à l'intérieur de cette dernière (Fig3). La paire de clés est générée à l'intérieur de la carte à puce, la clé publique sera publiée tandis que la clé secrète résidera exclusivement à l'intérieur de la carte et ne sera jamais révélée [Man02]. Pour cela le code de la machine SVM sera divisé en sections blanches à ne pas crypter et sections noires où le cryptage est fondamental (par exemple les fonctions qui interagissent avec l'agent pour calculer l'emplacement du prochain segment S_{i+1} à partir de S_i et particulièrement le premier segment S_0 à partir de la clé publique de l'agent) . En plus de la sécurisation par cartes à puce, la machine virtuelle SVM masque les interruptions au cours de son exécution et le code de la machine SVM associé à un identifiant (par exemple le numéro de série du processeur) de chaque site sera totalement signé et certifié. Le certificat choisi sera conforme au format X.509 et sera fourni par une autorité de certification qui se charge de plus de la vérification de la machine SVM ainsi que de la carte associée. Avant sa migration, l'agent vérifie obligatoirement le certificat du prochain site.

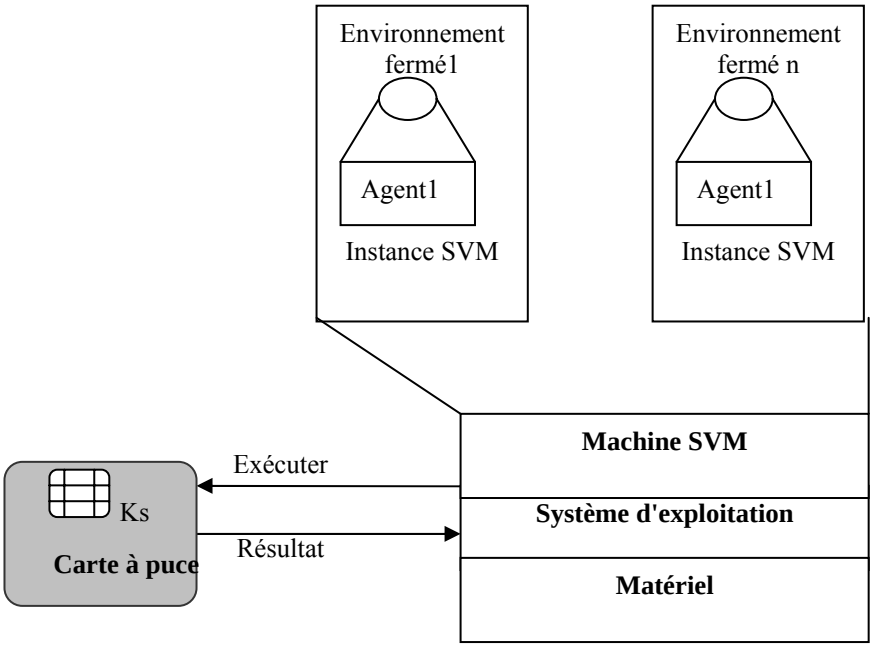

Fig 6: protection de la machine virtuelle SVM

6.3 Le transformateur de programmes

Le transformateur a pour objectif de générer un agent crypté C-A depuis un agent ordinaire A. Le principe consiste à segmenter l'agent A et à crypter différemment les segments. On utilisera un cryptage symétrique (vue l'économie de ressources et de temps que permet un cryptage symétrique par rapport à un cryptage asymétrique) au niveau des différents segments dont la clé change d'un segment au segment suivant. Un segment S_i assure en plus de ses fonctions le décryptage du segment suivant et le saut vers le segment S_{i+1}, sachant que S_{i+1} est le successeur de S_i sur le plan d'exécution et non par rapport à l'emplacement en mémoire. Avant de procéder au cryptage symétrique, on appliquera l'approche proposée par Hohl [Hoh97] en raison des performances qu'elle offre sans augmenter considérablement la taille des agents ou leur temps de réponse. Ainsi, on procédera tout d'abord à la création de nouvelles variables à partir des variables originelles comme présenté dans le paragraphe « Obscurcissement », on remplacera ensuite chaque appel procédural par le corps de la procédure (ou fonction, ou méthode…) et enfin on insérera des fragments de codes morts de façon aléatoire au niveau du code de l'agent. En plus de cette technique, on insérera à l'intérieur du code de l'agent des instructions qui engendrent sa destruction. Ces dernières seront exécutées suite à l'expiration de la durée de vie de l'agent, de la manipulation de sa date de création ou d'une éventuelle détection de la manipulation de son code. Après ces opérations, nous décomposerons le corps de l'agent en segments S_i de tailles aléatoires, nous procédons ensuite au cryptage symétrique des différents segments et nous réécrivons ces segments dans un nouvel ordre. Seul le premier segment S_1 sera crypté par la clé secrète de l'agent. Enfin nous "compilons" notre agent dans le langage interprétable par la machine SVM.

7. Conclusion et perspectives

Dans notre étude, nous avons défini les agents logiciels et nous avons expliqué la notion de mobilité tout en mettant l'accent sur les agents logiciels mobiles. Nous avons présentés ensuite les différentes approches proposées pour la sécurisation des agents mobiles en les classant en approches basées sur le matériel et approches basées sur le logiciel.

Notre étude a montré que les approches basées sur un périphérique apportent une solution solide mais coûteuse et difficile à maintenir. Tandis que les approches logicielles sont faciles à maintenir, évolutives et leurs coûts sont réduits, toutefois elles restent moins robustes. Dans notre travail, nous cherchons une solution hybride qui offre un degré de sécurité aussi solide que les approches basées sur le matériel à des coûts proches de ceux des solutions logicielles. Les agents sont protégés par la machine virtuelle SVM qui à son tour est protégée par une carte à puce. Dans la suite de notre travail, nous chercherons à renforcer encore notre approche par un cryptage solide en nous basant, par exemple, sur un calcul par fonction cryptographique. De plus nous chercherons à utiliser des agents qui sécurisent l'agent principal en exploitant le principe de gardes de corps ou de leurres. Aussi nous chercherons une méthode de sécurisation de SVM par son propre code tout en évitant l'utilisation d'un matériel additionnel pour faire de notre solution, une solution totalement logicielle.

8. Bibliographies

[BL96]: Dan Boneh et Richard J.Lipton, "Algorithms for black-box fields and their application to cryptography (extended abstract) ", Advances in cryptology, CRYPTO'96, volume 1109 of Lecture notes in computer sciences, pages 283-297, August 1996.

[Blu 88]: Manuel Blum et Sampath Kanan, "Designing programs to check their work", Technical report TR-88-009, International Computer Science Institute, December 1988.

[Che002]: Chenghui Luo, "Multi-Layred protection of mobile code", Fraunhofer Centre for Research in Computer Graphics, Inc, 2002.

[FGS96a]: Farmer, William; Guttmann, Joshua; Swarup, Vipin: "Security for Mobile Agents: Issues and Requirements", in: Proceedings of the National Information Systems Security Conference (NISSC 96), 1996.

[FIPS180] : Federal Information Processing Standard, 'Secure Hash Standard' FIPS PUB 180, Mai 1993.

[Hoh97]: Hohl, Fritz: "An approach to solve the problem of malicious hosts". Universität Stuttgart, Fakultät Informatik, Fakultätsbericht Nr. 1997/03, 1997. http://www.informatik.uni-stuttgart.de/cgi-bin/ncstrl_rep_view.pl?/inf/ftp/pub/ library/ncstrl.ustuttgart_fi/TR-1997-03/TR-1997-03.bib

[ISO9594-8] : ITU-T Recommendation X.509 | ISO/IEC 9594-8: Information Technology Open Systems Interconnection -The Directory: Public Key and Attribute Certificate Frameworks, March 2000.

[Jan00]: Wayne Jansen, “Countermeasures for mobile Agent security”, Computer Communications, Special issue on Advance Security Techniques for Network Protection, Elsevier Science, 2000.

[Jan01a]: Wayne Jansen, "Determining Privileges of Mobile Agents", in Proceedings of the Computer Security Applications Conference, December 2001..

[Jan01b]: Wayne Jansen, "A Privilege Management Scheme for Mobile Agent Systems", First International Workshop on Security of Mobile Multiagent Systems, Autonomous Agents Conference, May 2001.

[Jan301c]: Wayne Jansen, "Countermeasures for Mobile Agent Security", National Institute of Standards and Technology, Gaithersburg, MD 20899, USA, October 2001.

[Lau01]: Sergio Laureiro, "Mobile code protection", thèse de doctorat, Institut Eurocom, Janvier 2001.

[Mag02]: Magdy Saeb, Meer Hamza, Ashraf Soliman, "Protecting Mobile Agents against Malicious Host Attacks Using Treat Diagnostic AND/OR Tree", Arab Academy for Science, Technology & Maritime Transport Computer Engineering Department, Alexandria, Egypt, 2002.

[Man02]: Antonio Maña, Ernesto Pimentel, "An efficient software protection scheme", University of Malaga, Spain, 2002.

[Rot99] : V. Roth. "Mutual protection of co-operating agents". In Vitek and Jensen.

[Sch97]: F. B. Schneider and al., "Towards fault-tolerant and secure agentry". In Proceedings of 11th International Workshop on Distributed Algorithms, Saarbrücken, Germany, September 1997.

[ST98]: Sander Tomas and Tshudin Christian F. "Protecting Mobile Agents Against Malicious Hosts", Mobile Agent Security, LNCS Vol.1419, Springer-Verlag, 44-60, (1988).

[VIG98]: G. Vigna. "Cryptographic traces for mobile agents". In "Mobile Agents and Security, pages 137-153.

[Wil 98]: Uwe G. Wilhelm. "Cryptographically Protected Objects". http://lsewww.ep.ch/wilhelm/Papers/CryPO.ps.gz.