UNIVERSITE DE TOULOUSE

Habilitation à diriger les recherches de

l'INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE

présentée par

Vincent NICOMETTE

La protection des systèmes informatiques vis-à-vis des malveillances

Architectures et évaluation expérimentale

Le 19 novembre 2009 devant un jury composé de :

M. Yves Deswarte	Correspondant
M. Eric Filiol	Rapporteur
M. Christian Fraboul	Rapporteur
M. Jean-Louis Lanet	Rapporteur
M. Ludovic Mé	Examinateur
M. Jean-Jacques Quisquater	Examinateur
M. Frédéric Raynal	Examinateur
M. Michel Raynal	Examinateur

Avant-Propos

Les travaux présentés dans ce mémoire ont été effectués au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) du Centre National de la Recherche Scientifique (CNRS). Je remercie leurs directeurs successifs, Jean-Claude Laprie, Malik Ghallab et Raja Chatila de m'avoir permis de mener mes travaux dans de parfaites conditions au sein de ce laboratoire.

Je remercie David Powell, Jean Arlat et Karama Kanoun, chefs successifs du groupe Tolérance aux Fautes et Sûreté de Fonctionnement Informatique pour m'avoir accueilli au sein de leur groupe.

Je tiens à remercier particulièrement Yves Deswarte, Directeur de Recherche CNRS, qui a accompagné mon difficile parcours de chercheur tout au long de ces années. Mes multiples intrusions dans son bureau ont toujours été conclues par des réponses à mes nouvelles questions, ses remarques pertinentes et ses critiques constructives m'ont toujours éclairé et m'ont permis de mener à bien mes travaux.

J'exprime ma gratitude à :

- Christian Fraboul, Professeur à l'ENSEEIHT ;
- Jean Louis Lanet, Professeur à l'Université de Limoges ;
- Eric Filiol, Directeur du laboratoire de cryptologie et de virologie de Laval ;
- Michel Raynal, Professeur à l'Université de Rennes ;
- Jean-Jacques Quisquater, Professeur à l'université catholique de Louvain ;
- Frédéric Raynal, Ingénieur de recherche SOGETI ;
- Ludovic Mé, Profeseur à Supélec ;
- Yves Deswarte, Directeur de Recherche CNRS.

pour l'honneur qu'ils me font en participant à mon jury. Je remercie particulièrement Jean Louis Lanet, Eric Filiol et Christian Fraboul, qui ont accepté la charge d'être rapporteurs.

Table des matières

Chapitre 1

Activités de recherche et d'enseignement

1.1 Renseignements personnels

1.1.1 Etat civil

Vincent Nicomette, 40 ans

- Né le 5 juillet 1968 à Vitry-le-François, nationalité française
- Adresse : 3 Impasse du Levant, 31450 Pompertuzat
- Maître de Conférences à l'INSA de Toulouse, Département Génie Electrique et Informatique (DGEI)
- Recherches effectuées dans le groupe Tolérance aux Fautes et Sûreté de Fonctionnement Informatique (TSF) du Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) du CNRS.
- Emails et téléphones professionels :
  - INSA : vincent.nicomette@insa-toulouse.fr / 05 61 55 98 20
  - LAAS : vincent.nicomette@laas.fr / 05 61 33 64 23

1.1.2 Diplômes

- Diplôme d'ingénieur en Informatique et Mathématiques Appliquées de l'Ecole Nationale Supérieure d'Electronique, d'Electrotechnique, d'Informatique et d'Hydraulique de Toulouse (ENSEEIHT), juin 1992.
- DEA "Informatique Fondamentale et Parallélisme" de l'Institut National Polytechnique de Toulouse (INPT), septembre 1992.
- Doctorat de l'INPT, Spécialité Informatique, décembre 1996.

1.1.3 Cursus

- 1986/1989 : Classes préparatoires au lycée Louis le Grand à Paris
- 1989/1992 : Elève ingénieur à l'ENSEEIHT spécialité Informatique et Mathématiques Appliquées
- 1992/1993 : Service militaire effectué à l'Ecole Nationale Supérieure d'Ingénieurs de Construction Aéronautique (ENSICA)
- 1993/1996 : Formation doctorale au LAAS-CNRS (Groupe TSF), Directeur des recherches : Yves Deswarte. Thèse soutenue le 16 décembre 1996
- 1997/1999 : Ingénieur dans la société Matra Marconi Space à Toulouse
- Depuis 1999 : Maître de Conférences à l'INSA de Toulouse, GEI

1.2 Activités de recherche

Cette section résume nos activités de recherche en termes de publications, d'encadrements et d'animation scientifique. Nos thématiques de recherche et nos contributions sont détaillées dans la deuxième partie de ce mémoire.

1.2.1 Publications

1.2.1.1 Résumé

- Thèse : 1
- Participation à ouvrage : 3 (3 après la thèse)
- Revues internationales : 4 (4)
- Revues nationales : 1 (1)
- Conférences nationales avec acte et comité de lecture : 8 (6)
- Conférences internationales avec acte et comité de lecture : 14 (8)
- Rapport de contrat : 16 (12)
- Soumissions en cours : 3

1.2.1.2 Liste détaillée (hors rapports de contrat)

Thèse

V.NICOMETTE, "La protection dans les systèmes à objets répartis", Rapport LAAS N 96496, *Thèse de Doctorat*, Institut National Polytechnique, Toulouse, 17 décembre 1996, N 1252, 167p.

Revues Internationales : 4

E.LACOMBE, F.RAYNAL, V.NICOMETTE, "Rootkit modeling experiments under Linux", *Journal in Computer Virology*, Vol.4, N 2, pp.135-157, mai 2008.

E.ALATA, I.ALBERDI, V.NICOMETTE, P.OWEZARSKI, M.KAANICHE, "Internet attacks monitoring with dynamic connection redirection mechanisms", *Journal in Computer Virology*, Vol.4, N 2, pp.127-136, mai 2008.

A.SAIDANE, V.NICOMETTE, Y.DESWARTE, "The design of a generic intrusion tolerant architecture for Web servers", *IEEE Journal of Transaction on Dependable and Secure Computing*, Vol. 6, N 1, pp. 45-58, janvier 2009.

E. LACOMBE, V. NICOMETTE, Y. DESWARTE, "Enforcing Kernel Constraints by Hardware-Assisted Virtualization", Juin 2009, rapport LAAS N 09641, accepté pour publication dans *Journal in Computer Virology*.

Revues Nationales : 1

Y.DESWARTE, C.AGUILAR-MELCHOR, V.NICOMETTE, M.ROY, "Protection de la vie privée sur internet", *Revue de l'Electricité et de l'Electronique*, N 9, pp.65-74, octobre 2006.

Participations à ouvrages : 3

E.TOTEL, L.BEUS-DUKIC, J.P.BLANQUART, Y.DESWARTE, V.NICOMETTE, D.POWELL, A .WELLINGS, "GUARDS : multilevel integrity mechanisms", dans *A Generic Fault-Tolerant Architecture for Real-Time Dependable Systems*, Kluwer Academic Publishers, Ed. D.Powell, N ISBN 0-7923-7295-6, 2001, Chapitre 6, pp.99-120.

B.BERARD, P.COUPOUX, Y.CROUZET, P.DAVID, Y.GARNIER, S.GOIFFON, G.MARIANO, V.NICOMETTE, L.PLANCHE, I.PUAUT, J.M.TANNEAU, H.WAESELYNCK "Logiciel libre et sûreté de fonctionnement. Cas des systèmes critiques", *Hermes Science*, N ISBN 2-7462-0727-3, septembre 2003, 234p.

Y.DESWARTE, V.NICOMETTE, A.SAIDANE, "Tolérance aux intrusions sur internet", *Sécurité des systèmes d'information*, Hermes Science, Traité Information-Commande-Communication, Réseaux et Télécoms, N IISBN 2-7462-125 9-5, 2006, Chapitre 11, pp.339-367.

Conférences internationales avec actes et comité de lecture : 14

V.NICOMETTE, Y.DESWARTE, "An authorization scheme for distributed object systems", *OOPSLA'94 Conference. Workshop 8 : Standards for Security in Object-Oriented Systems*, Portland (USA), 23 octobre 1994 , 9p.

J.C.FABRE, V.NICOMETTE, T.PERENNOU, R.J.STROUD, Z.WU, "Implementing fault-tolerant applications using reflective object-oriented programming", *25th International Symposium on Fault-Tolerant Computing (FTCS-25)*, Pasadena (USA), 27-30 juin 1995 , pp.489-498.

V.NICOMETTE, Y.DESWARTE, "A multilevel security model for distributed object systems",

4th European Symposium on Research in Computer Security (ESORICS'96), Rome (Italie), 25-27 septembre 1996, pp.80-98.

V.NICOMETTE, Y.DESWARTE, "Symbolic rights and vouchers for access control in distributed object systems", *2nd Asian Computing Science Conference (ASIAN'96)*, Singapour (Singapour), 2-5 décembre 1996.

V.NICOMETTE, Y.DESWARTE, "An authorization scheme for distributed object systems", *1997 IEEE Symposium on Security and Privacy, Oakland(USA)*, 4-7 mai 1997, pp.21-30.

Y.DESWARTE, N.ABGHOUR, V.NICOMETTE, D.POWELL, "An Internet authorization scheme using smart card-based security kernels", *International Conference on Research in Smart Cards (E-smart 2001)*, Cannes (France), 19-21 septembre 2001.

Y.DESWARTE, N.ABGHOUR, V.NICOMETTE, D.POWELL, "An intrusion-tolerant authorization scheme for internet applications", *2002 International Conference on Dependable Systems & Networks (DSN'2002)*, Workshop on Intrusion Tolerant Systems, Washington (USA), 23-26 juin 2002, pp.C1.1-C1.6.

A.SAIDANE, Y.DESWARTE, V.NICOMETTE, "An intrusion tolerant architecture for dynamic content internet servers", *ACM Workshop on Survivable and Self-Regenerative Systems*, Fairfax (USA), octobre 2003, pp.110-114.

E.ALATA, M.DACIER, Y.DESWARTE, M.KAANICHE, K.KORTCHINSKY, V.NICOMETTE, V.H .PHAM, F.POUGET, "CADHo : Collection and Analysis of Data from Honeypots", *5th European Dependable Computing Conference (EDCC'5)*, Project track, Budapest (Hongrie), 20-22 avril 2005, pp.3-6.

E.ALATA, M.DACIER, Y.DESWARTE, M.KAANICHE, K.KORTCHINSKY, V.NICOMETTE, V.H .PHAM, F.POUGET, "Collection and analysis of attack data based on honeypots deployed on the internet", *1st International Workshop on Quality of Protection (QoP'2005)*, Milan (Italie), 15 septembre 2005.

M.KAANICHE, E.ALATA, V.NICOMETTE, Y.DESWARTE, M.DACIER, "Empirical analysis and statistical modeling of attack processes based on honeypots", *Dependable Systems and Networks (DSN'2006)*, Workshop on Empirical Evaluation of Dependability and Security (WEEDS), Philadelphie (USA), 25-28 juin 2006, pp.119-124.

E.ALATA, V.NICOMETTE, M.KAANICHE, M.DACIER, M.HERRB, "Lessons learned from the deployment of a high-interaction honeypot", *6th European Dependable Computing Conference (EDCC-6)*, Coimbra (Portugal), 18-20 octobre 2006, pp.39-44.

I.ALBERDI, E.ALATA, P.OWEZARSKI, V.NICOMETTE, M.KAANICHE, "Shark : spy honeypot with advanced redirection kit", *Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2007 Workshop)*, Toulouse (France), 5-6 novembre 2007, pp.47-52.

E. LACOMBE, V. NICOMETTE, Y. DESWARTE, "A Hardware-Assisted Virtualization Based Approach on How to Protect the Kernel Space from Malicious Actions", *EICAR 2009*, Berlin (Allemagne), 9-12 mai 2009, pp. 87-104.

Conférences nationales avec actes et comité de lecture : 8

V.NICOMETTE, "Un schéma d'autorisation pour la protection d'objets répartis", *2èmes Journées des Jeunes Chercheurs en Systèmes Répartis*, Rennes (France), 9-10 octobre 1995 , 9p.

E.ALATA, M.DACIER, Y.DESWARTE, M.KAANICHE, K.KORTCHINSKY, V.NICOMETTE, V.H .PHAM , F.POUGET, "Leurré.com : retour d'expérience sur plusieurs mois d'utilisation d'un pot de miel distribué mondialement", *Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC'05)*, Rennes (France), 1-3 juin 2005, pp.281-292.

E.LACOMBE, F.RAYNAL, V.NICOMETTE, "De l'invisibilité des rootkits : application sous Linux", *Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC)*, Rennes (France), 30 Mai- 1 juin 2007, 27p.

E.ALATA, I.ALBERDI, V.NICOMETTE, P.OWEZARSKI, M.KAANICHE, "Mécanisme d'observation d'attaques sur internet avec rebonds", *Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC)*, Rennes (France), 31 mai-1 juin 2007, pp.53-67.

I.ALBERDI, P.OWEZARSKI, V.NICOMETTE, "Conception d'une plateforme expérimentale pour l'étude comportementale de maliciels de type bots", *2nd Conference on Security in Network Architectures and Information Systems (SAR-SSI 2007)*, Annecy (France), 12-15 juin 2007, pp.143-152.

E. ALATA, M. KAANICHE, V. NICOMETTE, "Etude expérimentale d'attaques par dictionnaire", *3ème Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR/SSI'2008)*, Loctudy (France), 13-17 octobre 2008, pp.301-315

E. LACOMBE, V. NICOMETTE, Y. DESWARTE, "Une approche de virtualisation assistée par le matériel pour protéger l'espace noyau d'actions malveillantes", janvier 2009, *SSTIC 2009 (Symposium sur la Sécurité des Technologies de l'Information et des Communications)*, Rennes, 4-6 juin 2009, pp 321-346.

I. ALBERDI, P. OWERZARSKI, V. NICOMETTE, "Plateforme pour l'exécution contrôlée de logiciels malveillants", Rapport LAAS N 09318, avril 2009, 14p., Accepté pour publication dans la conférence CFIP 2009.

Soumission en cours : 3

V.NICOMETTE , D.POWELL , Y.DESWARTE , N.ABGHOUR , C.ZANON, "Fine-grained authorization for internet applications", Rapport LAAS N 08286, juin 2008, 32p. Soumis à la revue *Elsevier Journal of System Architecture (JSA)*.

V. NICOMETTE, M. KAANICHE, E. ALATA, *An Empirical Analysis of Attackers Behavior :*

Experiment and Results, Rapport LAAS N 09062, janvier 2009, 30p., Soumission à la revue *Journal In Computer Virology.*

V. NICOMETTE, M. KAANICHE, E. ALATA, "Une analyse empirique du comportement des attaquants : expérimentations et résultats", Rapport LAAS N 09176, janvier 2009, 30p., Soumis à la revue de *Technique et Science Informatiques (TSI).*

1.2.2 Encadrements

1.2.2.1 Résumé

- Thèses soutenues : 3 (encadrées à 50%)
- Thèses en cours : 2 (encadrées à 50%)
- Stages de DEA/Master Recherche/Master Professionnel : 8
- Nouvelles thèses débutant à la rentrée 2009 : 2 (encadrées à 50%)

1.2.2.2 Encadrements de thèses

Noreddine Abghour (2001 à 2004)

- Thèse de doctorat en Informatique de l'Institut National Polytechnique de Toulouse (INPT) soutenue le 24 juin 2004.
- Encadrement à 50% avec Yves Deswarte (DR CNRS)
- Sujet : Conception des schémas d'autorisation pour des applications réparties sur Internet.
- Mots clés : sécurité informatique , contrôle d'accès, serveur d'autorisation , délégation des droits d'accès, Internet, tolérance aux intrusions, moniteur de référence, preuve d'autorisation.
- Actuellement maître de conférences à l'Université de Casablanca (Maroc).

Ayda Saidane (2002 à 2005)

- Thèse de doctorat en Informatique de l'INSA de Toulouse soutenue le 28 janvier 2005.
- Encadrement à 50% avec Yves Deswarte (DR CNRS)
- Sujet : Conception et réalisation d'une architecture tolérant les intrusions pour des serveurs internet
- Mots clés : tolérance aux fautes, tolérance aux intrusions, sécurité informatique, Internet.
- Actuellement en PostDoc à l'université de Trente (Italie).

Eric Alata (2003 à 2007)

- Thèse de doctorat en Informatique de l'INSA de Toulouse soutenue le 7 décembre 2007.
- Encadrement à 50% avec Mohamed Kâaniche (DR CNRS)
- Sujet : Observation, caractérisation et modélisation de processus d'attaques sur Internet
- Mots clés : sécurité informatique, pot de miel, évaluation quantitative.

– Actuellement maître de conférences à l'INSA de Toulouse.

Ion Alberdi (depuis octobre 2006)

– Thèse de doctorat en Informatique de l'INSA de Toulouse
– Encadrement à 50% avec Philippe Owerzarski (CR CNRS)
– Sujet : Environnement d'exécution contrôlée de maliciels
– Mots clés : Réseau Internet, vulnérabilités, botnets, capture de maliciels, exécution de maliciels, pare-feu.

Eric Lacombe (depuis octobre 2006)

– Thèse de doctorat en Informatique de l'INSA de Toulouse
– Encadrement à 50% avec Yves Deswarte (DR CNRS)
– Sujet : Mécanismes matériels pour la protection de noyaux de système d'exploitation
– Mots clés : noyau de système d'exploitation, vulnérabilités, rootkits, protections matérielles, technologie hyperviseur

1.2.2.3 Encadrements de stages

Christophe Zanon (2001), Maîtrise Informatique

– Ecole Doctorale EDSYS, DEA Systèmes Informatiques
– Sujet : conception et implémentation d'un prototype de schéma d'autorisation pour des applications réparties sur Internet

Eric Alata (2004), élève ingénieur INSA

– Ecole Doctorale EDSYS, DEA Système Informatique
– Sujet : Evaluation de la Sécurité Informatique vis-à-vis des malveillances

Bastien Continsouzas (2004), élève ingénieur INSA

– Ecole Doctorale EDSYS, DEA Système Informatique
– Sujet : Réalisation d'une architecture de serveurs Web tolérant les intrusions

Géraldine Vache (2005), élève ingénieur INSA

– Ecole Doctorale EDSYS, Master de Recherche SAID (Systèmes Automatiques et Informatique Décisionnel)
– Sujet : Outils pour l'analyse et l'évaluation de la sécurité informatique

Aleksandar Kalev (2008), Université Paul Sabatier

– Master Profesionnel "Ingénierie en Systèmes de Télécommunications et Réseaux Informatiques", IUP M1.
– Sujet : Plateforme de collecte de logiciels malveillants.

Fernand Lone Sang (2009), élève ingénieur INSA

– Sujet : Mécanismes matériels incontournables pour la sécurité des systèms d'exploitation.

Rim Akrout (2009), élève ingénieur ENIS (Ecole Nationale d'Ingenieurs de Sfax), Tunisie.

– Sujet : Evaluation expérimentale de mécanismes de protection (en particulier de systèmes de détection d'intrusion) vis-à-vis des malveillances.

Anthony Dessiatnikoff (2009), Université de Limoges

– Ecole Doctorale "Sciences et Ingénierie pour l'Information", Master Recherche "Sécurité de l'Information et Cryptologie".
– Sujet : Etude des mécanismes de confiance dans les systèmes actuels.

1.2.3 Animation scientifique

1.2.3.1 Collaborations industrielles

Projet GUARDS (Generic Upgradable Architectures for Real-Time Dependable Systems) - 1996/1999
Activité réalisée lors de mon emploi chez Matra Marconi Space

– **Nature** : Projet du programme européen ESPRIT
– **Objectifs** : Ce projet avait pour objet la définition et la validation de mécanismes de tolérance aux fautes et d'une architecture générique pour des applications critiques.
– **Partenaires** : Technicatome (France), Ansaldo Segnalamento Ferraviario (Italie), Matra Marconi Space (France), Intecs Sistemi (Italie), Siemens Austria (Autriche), Pisa Dependable Computing Centre (Italie), Universität Ulm (Allemagne) et University of York (Royaume-Uni), LAAS.
– **Contribution** : Développement de composants logiciels particuliers compatible POSIX permettant l'interaction de logiciels de niveaux de criticité différents.
– **Encadrement** : Un sous-traitant

Projet Européen MAFTIA (Malicious and Accidental Fault Tolerance for Internet Applications) - 2000/2003

– **Nature** : Projet IST du 5iéme PCRD européen.
– **Objectifs** : Le projet visait directement à faciliter les développements d'applications sur Internet tolérant les intrusions. Pour cela, des protocoles et intergiciels (*middleware* en anglais) ont été développés pour gérer plus facilement les communications de groupe tolérant les fautes, avec ou non des contraites de temps réel et avec ou non des garanties de confidentialité et d'intégrité. Ces protocoles et intergiciels ont en particulier permis le développement de tierces

parties de confiance (par exemple, une autorité de certification) qui tolérent les intrusions. Des méthodes de détection d'intrusions réparties sur Internet ont été également étudiées avec une attention particulière. Enfin, des schémas d'autorisation ont été développés pour des applications mettant en jeu des organisations mutuellement méfiantes.
- **Partenaires** : Defense Evaluation and Research Agency, Malvern (Royaume-Uni), Universidade de Lisboa (Portugal), University of Newcastle (Royaume-Uni), Universität Saarland (Allemagne) et IBM Research, Zurich (Suisse).
- **Contribution** : Nous avons conçu et réalisé un prototype d'un serveur d'autorisation, tierce partie de confiance tolérant les intrusions.
- **Encadrement** : Ces travaux ont fait l'objet de la thèse de Noreddine Abghour et du stage de DEA de Christophe Zanon.

Projet DIT (Dependable Intrusion Tolerance) - 2002/2003

- **Nature** : Projet du programme OASIS (Organically Assured and Survivable Information Systems) de la DARPA
- **Objectifs** : L'objectif était de développer des serveurs (en particulier des serveurs Web) tolérant les intrusions (en plus des fautes accidentelles) sur Internet. L'architecture est basée sur des plateformes diversifiées (OS + logiciels d'applications) fournissant des contenus identiques, sous le contrôle de mandataires (proxies), eux-mêmes diversifiés. Les mécanismes de détection d'erreurs (comparaisons des contenus, tests d'intégrité, surveillance mutuelle des proxies) est complétée par des outils de détection d'intrusion EMERALD. La redondance s'adapte automatiquement en fonction du niveau d'alerte, au prix d'une dégradation progressive des performances
- **Partenaires** : SRI International (le LAAS était un sous-traitant)
- **Contribution** : Nous avons participé à la conception de l'architecture des serveurs Web et implémenté un prototype comprenant plusieurs mandataires pour des contenus dynamiques.
- **Encadrement** : Ces travaux ont fait l'objet de la thése d'Ayda Saidane et du stage de Bastien Continsouzas

ACI Cadho (Collecte et Analyse de Données issues de Honeypots - 2004/2007

- **Nature** : Projet de l'ACI Sécurité Informatique
- **Objectifs** : Ce projet avant pour buts principaux de : i) fournir à la communauté scientifique une plateforme basée sur des pots de miel dits à "basse interaction" pour analyser les attaques mises en œuvre sur Internet ; ii) développer des méthodes d'analyse et des modèles stochastiques permettant d'extraire des informations pertinentes à partir des données issues de la plateforme et de caractériser les processus d'attaque observés ; iii) développer des pots de miel plus sophistiqués, dits à "haute interaction", permettant d'analyser des scénarios d'attaques réalisés à partir du moment où une machine cible est compromise et d'évaluer leur impact sur la sécurité.
- **Partenaires** : l'Institut Eurecom et le CERT RENATER

- **Contribution** : Notre contribution a porté sur les points ii) et iii). Nous avons contribué à développer des modéles pour caractériser les processus d'attaque observés et nous avons eu la reponsabilité de concevoir et implémenter un pot de miel haute interaction.
- **Encadrement** : Ces travaux ont fait l'objet du stage de DEA et de la thèse d'Eric Alata.

Projet Européen CRUTIAL (CRitical Utility InfrastructurAL resilience) - 2006/2009

- **Nature** : Projet STREP du programme européen IST.
- **Objectifs** : Il vise à analyser et modéliser les interdépendances d'infrastructures critiques et développer des solutions architecturales permettant d'assurer la résilience de ces infrastructures, en considérant comme exemple le cas des infrastructures électriques.
- **Partenaires** : CESI (Italie) (coordinateur), CNR-ISTI (Italie), CNIT (Italie), Katholieke Universiteit Leuven (Belgique), Universidade de Lisboa (Portugal).
- **Contribution** : Utilisation de la technologie des pots de miel pour évaluer les attaques ciblant les environnements SCADA.
- **Encadrement** : Aucun

Projet OSCAR (Overlay Network Security, Characterization And Recovery) - 2005/2008

- **Nature** : Projet ANR-RNRT
- **Objectifs** : L'objectif du projet est d'étudier les faiblesses des réseaux superposés thématiques comme les réseaux de P2P, de voix sur IP ou de jeux, et leurs réactions face aux attaques venant de l'Internet. A partir de l'analyse des attaques et de leur impact - qui sera fait avec des outils de métrologie ainsi que des pots de miel - le projet vise à proposer des solutions pour défendre ces réseaux superposés et les rendre plus robustes, et ce à la fois au niveau du réseau overlay que du réseau d'infrastructure
- **Partenaires** : France Telecom, Université Pierre et Marie Curie (LIP6), INRIA (Unités de Rocquencourt et Sophia Antipolis), Groupement des Ecoles des Télécommunications, Mitsubishi Electric, Ecole Normale Supérieure de Lyon, Miriad Technologies
- **Contribution** : Utiliser la technologie des pots de miel afin de collecter des traces sur les attaques des réseaux d'infrastructures.
- **Encadrement** : Ces travaux font l'objet de la thèse de Ion Alberdi.

Projet ANR DALI (Design and Assessment of application Level Intrusion detection Systems) - 2009/2012

- **Nature** : Projet ANR
- **Objectifs** : Proposer des nouvelles méthodes de conception de systèmes de détection d'intrusion (IDS) au niveau applicatif et en même temps, proposer des méthodes permettant d'évaluer ces IDS.
- **Partenaires** : Kereval, SupElec Rennes, Télécom Bretagne.
- **Contribution** : Proposer des méthodes et des outils permettant d'évaluer les systèmes de détection d'intrusion développés par les partenaires du projet.

– **Encadrement** : Ces travaux font l'objet du stage de Rim Akrout et vont se poursuivre par sa thèse.

1.2.3.2 Laboratoire coopératif et réseaux d'excellence

LIS (Laboratoire d'Ingénierie de la Sûreté de fonctionnement) - 1992/2000

– **Nature** : Laboraroire Coopératif
– **Objectifs** : Le LIS a constitué une forme exemplaire de Laboratoire Coopératif, se caractérisant par la mise en commun de compétences - personnels détachés par les partenaires industriels, chercheurs du groupe Tolérance aux Fautes et Sûreté de fonctionnement informatique du LAAS et doctorants - et de ressources, pour résoudre des problèmes de recherche considérés par l'ensemble des partenaires comme des problèmes clés de la sûreté de fonctionnement. Cette structure et les modalités de fonctionnement, par lesquelles des ingénieurs ont été immergés quotidiennement et sur une longue période au sein d'un laboratoire de recherche académique restent novatrices et originales en matière de coopération recherche-entreprises.
– **Partenaires** : Matra Marconi Space France, Technicatome, Aerospatiale Aéronautique, Électricité de France et Thomson-CSF
– **Contribution** : Mes travaux de thèse sur la protection des systèmes répartis
– **Encadrement** : Aucun

RIS (Réseau d'Ingénierie de la Sûreté de fonctionnement) - 2000/2004

– **Nature** : Réseau coopératif
– **Partenaires** : Astrium, EADS Airbus, Technicatome et THALES
– **Objectifs** : Ce réseau avait pour but de poursuivre et d'étendre la coopération déjà établie dans le cadre du LIS, à une participation industrielle et académique élargie. L'objectif du réseau était de promouvoir les activités d'interaction entre ses membres de manière à favoriser le partage des expériences et à stimuler des réflexions communes sur des thèmes appartenant au domaine de l'ingénierie de la sûreté de fonctionnement des systèmes à logiciel prépondérant.
– **Contribution** : Notre contribution a été de coordonner et de participer en tant qu'auteur à la rédaction de l'ouvrage *Logiciel libre et sûreté de fonctionnement. Cas des systèmes critiques* publié en Septembre 2003.
– **Encadrement** : Aucun

Réseau d'excellence RESIST (Resilience for survavibility in IST) - 2006/2009

– **Nature** : Réseau d'excellence (NoE) du programme européen IST
– **Objectifs** : Son objectif était l'extension d'échelle de la sûreté de fonctionnement, afin de pouvoir relever les défis posés par les grands systèmes émergents, évolutifs, en réseaux fixes ou mobiles, mettant en oeuvre des applications dont les exigences sont de plus en plus fortes.
– **Partenaires** : Budapest University of Technology and Economics (Hongrie), City University (Londres, Royaume-Uni), Technische Universität Darmstadt (Allemagne), Deep Blue Srl (Ita-

lie), France Telecom R&D (France), IBM Research GmbH (Suisse), Institut Eurécom (France), IRISA (France), IRIT(France), Fundação da Faculdade de Ciencas da Universidade de Lisboa (Portugal), University of Newcastle upon Tyne (Royaume-Uni), Università di Pisa (Italie), QinetiQ Limited (Royaume-Uni), Università di Roma "La Sapienza" (Italie), University of Southampton (Royaume-Uni), Universität Ulm (Allemagne), Vytautas Magnus Universitetas (Lituanie). Le LAAS est cooordinateur du projet.
- **Contribution** : Le LAAS était coordinateur du projet mais ma contribution a concerné plus particulièrement la contribution au deliverable D13 du Work Package Evolvability.
- **Encadrement** : Aucun

1.2.3.3 Comités de programmes et comités de lecture

- Participation au comité de programme de la conférence francophone SAR/SSI 2009.
- Participation aux comités de lecture de diverses conférences (IFIPSEC, ESORICS, DSN-PDS, DSN-DCCS, PRDC, SAR, NCA, CMS, SRDS, CARDIS) et de la revue IEEE TDSC.
- Publicity chair de la conférence EDCC 2002.

1.2.3.4 Table ronde invitée

Invitation à participer à la table ronde intitulée "Threats for the Internet : what tools for evaluating the actual risk" du workshop de l'IEEE Monitoring, Attack Detection and Mitigation (MONAM) 2007.

1.2.3.5 Diffusion des résultats de recherche

Suite à la thèse d'Eric Alata, la presse écrite, audiovisuelle et télévisuelle nous a contacté pour diverses entrevues :

- Jean-Marie Decorse. "Internet : il piège les pirates avec son pot de miel", La Dépêche, 23 avril 2008, Toulouse.
  http ://www.ladepeche.fr/article/2008/04/23/450111-Internet-il-piege-les-pirates -avec-son-pot-de-miel.html
- Frank Niedercorn, "L'activité des pirates du Web sous l'oeil des scientiques", Les Echos, 22 avril 2008.
  France. http ://www.lesechos.fr/info/innovation/4717796.htm ?xtor=RSS-2099
- Lysiane Beaumel, "Il espionne les pirates du Net", La Gazette du Midi, 14 avril 2008, Toulouse.
- Philippe Font, "Il traque les pirates informatiques de la Toile", Metro, 20 mai 2008, France.
  http ://www.metrofrance.com/x/metro/2008/05/20/w5EXRuc8Pw62E/index.xml
- Stéphane Iglésis, Le PLUS France Info, France Info, 16 avril 2008.
- Pierre Nicolas, L'invité du jour / Midi-Pyrénées, France 3, 16 avril 2008.

1.2.3.6 Autres types d'animations

- Membre du groupe de travail en Sécurité Informatique de Feria (Fédération de Recherche en Informatique et Automatique)
- Membre du Clusir (Club des Utilisateurs de la Sécurité Informatique). Le Clusir étant officiellement hebergé sur le site de l'INSA de Toulouse, les différentes commissions et les manifestations ont donc lieu à l'INSA. En tant que correspondant local, je participe à l'organisation de ces événements.

1.3 Activités d'enseignement

J'ai été recruté en tant que Maître de Conférences au DGEI de l'INSA de Toulouse en septembre 1999, sur un profil en Informatique et Réseaux. Les responsabilités pédagogiques qui m'ont été confiées comportent trois volets majeurs :
- prise en charge et évolution des cours relatifs au langage C et au système Unix ;
- pour les années 2000/01, 2001/02 et 2002/03 : participation a l'élaboration de la partie réseaux du programme d'enseignement d'une nouvelle Spécialité de 3 ans à l'INSA, dédiée aux Réseaux et Télécommunications ; première promotion sortie en juin 2003. Mon travail a plus précisément consisté à définir des manipulations qui constituent le bureau d'études Réseaux. Ce BE constitue toujours actuellement le plus gros BE de manipulation des réseaux fait par les étudiants.
- Mettre en place et coordonner un certain nombre de cours liés à la sécurité informatique, mon domaine de recherche.

1.3.1 Responsabilités pédagogiques

1.3.1.1 Enseignements

J'enseigne :

- dans la 4ème année, spécialités RT (Réseaux et Télécom), I (Informatique) et AE (Automatique et Electronique) du GEI
- dans le 5ème année, spécialité RT du GEI
- dans les années 2 et 3, pré-orientations MIC (Modélisation, Informatique et Communication) et IMACS (Ingénierie des Matériaux, Composants et Systèmes).

Mes enseignements actuels sont :

- **Le système Unix**
  - Type : cours (6h30 étudiant) et TD (11h15 étudiant)
  - Niveau : 2ème année INSA (pré-orientation MIC)

- **Le langage C**
  - Type : cours (6h30 étudiant) et TD (5h30 étudiant)
  - Niveau : 3ème année INSA (pré-orientation MIC)
- **Codes Secrets et protection de l'information**
  - Type : cours (6h30 étudiant)
  - Niveau : 2ème et 3ème année INSA
- **Projet Réseau**
  - Type : TP (33h étudiant)
  - Niveau : 4ème année RT
- **Interconnexion de réseaux et Sécurité**
  - Type : cours (6h30 étudiant) et TP (5h50 étudiant)
  - Niveau : 4ème année RT
- **TP Réseaux**
  - Type : TP (5h50/8h15 étudiant)
  - Niveau : 4ème année AE/GI

Auparavant, je dispensais également

- **Java**
  - Type : TP (19h15 étudiant)
  - Niveau : 4ème année RT
- **Algorithmique et programmation**
  - Type : TP (48h étudiant)
  - Niveau : 3ème année (ex 3ème année GII)

1.3.1.2 Certification Cisco

Nous avons décidé en 2007 de former nos étudiants à la certification Cisco (CCNA) en devenant une *local academy Cisco*. Cette certification nous a demandé un investissement en matériel mais aussi le montage de nouvelles manipulations qui n'étaient pas nécessairement couvertes par nos enseignements réseaux. J'ai eu en charge le montage de ces manipulations. L'ensemble de ce travail a donné lieu à 7 nouvelles manipulations de TP.

Le travail a également consisté à avoir régulièrement des points de rencontre avec les étudiants pour évaluer l'avancée de leur travail et également pour surveiller les examens intermédiaires, préparatoires à la certification.

1.3.1.3 Orientation sécurité

Le département Génie Electrique et Informatique de l'INSA affirme aujourd'hui la volonté de développer les activités concernant la sécurité informatique et en particulier la sécurité des

réseaux. En tant qu'enseignant et chercheur dans ce domaine, il m'a donc été confié la charge de préparer un projet d'enseignement correspondant à ces activités.

1.3.2 Responsabilités administratives

- Responsable de 6 Unités de Valeur dans les départements STPI (Sciences et Techniques Pour l'Ingénieur) et DGEI.
- Responsable de la 5ème année Temps Réel et Systèmes de 2000 à 2003.
- Membre élu du conseil de département de 2002 à 2004.
- Membre du conseil d'orientation Réseau et Télécom du DGEI.
- Représentation du département GEI au CARI (Conseil d'Administration du Centre des Ressources Informatiques)
- Membre du comité de sélection pour le recrutement d'un maître de conférence pour le DGEI en juin 2009.

Chapitre 2

Travaux de recherche : introduction générale et vue globale des travaux

Les travaux de recherche présentés dans ce manuscrit ont été réaliés dans le groupe "Tolérance aux fautes et Sûreté de Fonctionnement informatique (TSF)" depuis 1993 (avec une interruption de 2 ans 1/2 toutefois, correspondant à un passage dans le milieu industriel au sein d'une équipe Recherche et Développement dans la société Astrium). La sûreté de fonctionnement est un concept générique, qui englobe les propriétés de disponibilité, fiabilité, intégrité, confidentialité, maintenabilité et sécurité par rapport aux défaillances catastrophiques (en raccourci "sécurité-innocuité"). L'association de la confidentialité, l'intégrité et la disponibilité conduit à la "sécurité-immunité". Les activités du groupe couvrent la prévention des fautes, la tolérance aux fautes, l'élimination des fautes et la prévision des fautes. Elles sont structurées selon ces quatre thèmes, et couvrent un large spectre de classes de fautes. Leur finalité est de développer des méthodes et de les concrétiser par des outils. Nos méthodes et outils sont utilisés dans le cadre d'applications issues des domaines des transports, des services, de la production d'énergie, de la défense, de l'espace, etc. L'ensemble des travaux est sous-tendu par la formulation des concepts de base de la sûreté de fonctionnement.

La thématique dans laquelle s'inscrit l'ensemble de nos travaux est la "sécurité-immunité". Tous ces travaux visent à protéger les systèmes informatiques et plus particulièrement les systèmes répartis vis-à-vis des malveillances. Tout au long de ces années, l'évolution de nos travaux a accompagné l'évolution des menaces visant les systèmes informatiques. Ces menaces ont elles-mêmes évolué en fonction des améliorations croissantes des moyens de communications des systèmes informatiques répartis, et en particulier en fonction du développement du réseau Internet, qui est désormais utilisé par tout un chacun dans sa vie quotidienne, mais qui est aussi devenu la principale cible d'attaques informatiques.

Au début des années 90, Internet n'avait pas encore connu l'essor actuel. Il n'était pas question

d'ADSL ou de connexion à la maison pour chacun d'entre nous. En revanche, les systèmes répartis existaient bel et bien et bon nombre de problématiques de sécurité liées à cette répartition étaient déjà à l'étude. C'est dans ce contexte que nos travaux ont débuté. Nous nous sommes penchés sur les problèmes d'autorisation dans ces systèmes répartis. L'autorisation est un problème classique de sécurité qui a fait l'objet de nombreux travaux dans le cadre de systèmes informatiques non répartis. Nous avons donc étudié l'adaptation de ces solutions dans le contexte de systèmes répartis et proposé un modèle d'autorisation. Ce modèle a fait l'objet de ma thèse de doctorat. Ces travaux ont ensuite été naturellement prolongés dans le contexte de systèmes répartis sur le réseau Internet. Alors que les précédents travaux n'avaient envisagé de solution que pour des réseaux locaux où à petite échelle, l'émergence d'Internet nous a conduit à adapter notre schéma d'autorisation à des systèmes plus largement répartis, sur lesquels il n'est pas possible de faire les mêmes hypothèses que sur un réseau local. Ces travaux ont donné lieu à une collaboration avec divers partenaires dans le cadre du projet Européen MAFTIA (Malicious and Accidental Fault Tolerance for Internet Applications). Un prototype d'implémentation d'un serveur d'autorisation implémentant notre schéma a notamment été développé dans le cadre de ce projet. Ces travaux sont résumés dans le chapitre 3.

Dans le cadre de l'utilisation massive d'Internet et des vulnérabilités croissantes concernant les applications développées spécifiquement pour ce réseau, il nous est apparu primordiale de nous pencher sur la sécurité des serveurs de l'Internet, c'est-à-dire des machines offrant des services tels que les serveurs Web par exemple. Ce type de machines est en effet devenu au fil des années une des cibles privilégiées des attaquants. Nos travaux n'ont pas visé à concevoir et implémenter des serveurs Internet inattaquables (ce qui nous semble utopique) mais des serveurs capables de résister aux attaques. Il s'agit ici d'utiliser les techniques classiques de tolérance aux fautes que nous avions déjà éprouvées dans notre groupe de recherche et de les adapter pour la tolérance aux fautes intentionnelles. Ainsi, les techniques de redondance, de diversification et de vote majoritaire par exemple ont été utilisées dans le but d'assurer une protection efficace de serveurs Internet. Ces travaux ont notamment donné lieu à une collaboration avec SRI International dans le cadre du projet DIT (Dependable Intrusion Tolerance) du programme OASIS (Organically Assured and Surviable Information Systems) de la DARPA. Un prototype de serveur Web tolérant aux intrusions a notamment été développé et testé dans le cadre de cette collaboration. Ces travaux sont résumés dans le chapitre 4.

Une autre facette des problèmes de sécurité concernant le réseau Internet est la compréhension et la connaissance des techniques d'attaques et de l'organisation des attaquants. En effet, la facilité d'accès au réseau Internet fait que désormais les attaques peuvent être lancées depuis n'importe quel endroit de la planète, que les attaquants peuvent utiliser de multiples ressources réparties pour organiser leurs attaques, ce qui rend l'identification de l'organisation des attaquants et de leurs outils très difficile. Il devient également très difficile de pouvoir remonter à la source des concepteurs des maliciels (*malware* en anglais) les plus dangereux car ces maliciels se répandent très rapidement sur Internet et bon nombre d'utilisateurs peu scrupuleux, même totalement incultes en matière de systèmes informatiques, peuvent s'"amuser" à les utiliser (utilisateurs ap-

pelés parfois *script kiddies*). Aussi, la connaissance des attaquants et de leur organisation nous apparaît dès lors un défi important à surmonter pour améliorer la protection de nos systèmes informatiques. Cette connaissance permet en effet de mieux anticiper les problèmes et de pouvoir être proactif. Cette connaissance nous permet également de mieux évaluer la sécurité de nos systèmes informatiques car elle nous permet de justifier des hypothèses faites sur le comportement des attaquants lors des tests de la sécurité de nos systèmes. Nous avons, donc, durant trois années environ, observé des attaquants à l'aide d'une architecture de collecte et d'analyse de données basée sur la technologie des *pots de miel*. Ces travaux ont donné lieu à une collaboration dans le cadre d'un projet ANR *CADHO* avec l'Institut Eurécom (Sophia-Antipolis) et le CERT Renater. Nous résumons ces travaux dans le chapitre 5.

La problématique générale de la sécurité du réseau Internet devient de plus en plus complexe et relève souvent de techniques liées aux réseaux assez sophistiquées. Nous avons donc décidé de collaborer étroitement avec des chercheurs de l'équipe OLC (Outils et Logiciels pour la Communication) du LAAS afin de travailler ensemble sur l'identification de trafics malveillants sur Internet. Nous avons donc associé nos compétences en sécurité (et notamment la technologie des pots de miel mise au point dans le cadre des travaux précédents) et les connaissances en métrologie du groupe OLC. Nous avons ainsi proposé de développer une plateforme de collecte et d'exécution contrôlée de maliciels responsables du trafic malveillant sur Internet. Cette problématique est importante du point de vue de la sécurité car elle permet d'identifier au plus tôt les maliciels qui sévissent sur le réseau Internet, ce qui contribue à l'amélioration de notre connaissance des attaquants et de leurs outils, prolongeant par la-même les travaux présentés dans la paragraphe précédent. Pour l'équipe OLC, cette étude est importante au sens où la connaissance au plus tôt du trafic malveillant généré par les différents maliciels en cours de déploiement sur le réseau Internet permet d'améliorer la qualité de service du réseau en bloquant ce trafic au plus près de la source. La plateforme d'exécution et d'analyse de maliciels est présentée dans le chapitre 6. En particulier, cette plateforme fait intervenir un nouveau type de filtre de paquets que nous avons spécifiquement imaginé, de façon à contrôler le plus efficacement possible l'exécution des maliciels analysés.

Assurer la protection des systèmes informatiques ne peut se faire efficacement sans envisager des mécanismes de protection dans les couches les plus basses de ces systèmes et en particulier des mécanismes de protection matériel. Cette problématique, qui sous-tend toutes les autres activités de recherche menées, correspond à une tâche de fond qui nous semble essentielle, même si nous n'avons eu l'opportunité de démarrer ces travaux que récemment. Nous nous intéressons donc, dans ce cadre, à la protection de systèmes opératoires, et en particulier à l'aide de mécanismes matériels. Nous nous sommes notamment penchés sur la technologie des hyperviseurs supportée par le matériel implémenté dans les processeurs récents et nous avons étudié comment cette technologie peut être utilisée pour assurer efficacement la protection d'un noyau de système d'exploitation. Ces travaux sont résumés dans le chapitre 7.

La figure 2.1 donne une présentation de la chronologie de ces thèmes de recherche.

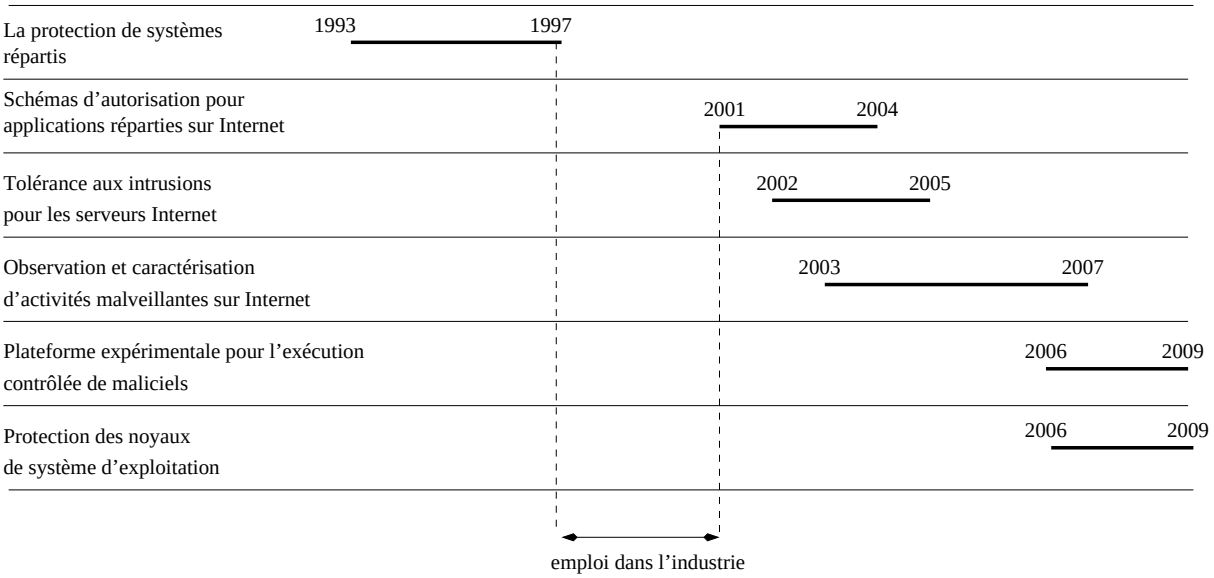


FIG. 2.1 – Chronologie des activités de recherche

Enfin, nous proposons dans le chapitre 8, des prospectives à l'ensemble de ces travaux.

Chapitre 3

Travaux de recherche : schémas d'autorisation pour applications réparties sur Internet

3.1 Problématique

La sécurité informatique repose principalement sur l'authentification des utilisateurs et sur l'autorisation, c'est-à-dire le contrôle des droits d'accès. L'authentification est indispensable pour identifier (avec un degré de confiance suffisant) chaque utilisateur, de façon à lui attribuer les droits qui lui conviennent et aussi de façon à le rendre responsable de ses actions. L'autorisation consiste à ne permettre à l'utilisateur que de réaliser des actions légitimes. L'autorisation devrait, dans la mesure du possible, obéir au principe du moindre privilège : à un moment donné, un utilisateur ne doit pouvoir exécuter que les seules opérations nécessaires à l'exécution de sa tâche légitime. L'autorisation est mise en œuvre par des mécanismes de protection qui permettent de détecter et de bloquer une tentative, volontaire ou involontaire, par un utilisateur d'outrepasser ses privilèges. Cette tentative peut être alors analysée par les responsables de la sécurité pour permettre éventuellement de punir le coupable, ce qui contribuera à la dissuasion d'autres tentatives. Authentification, autorisation, détection, répression et dissuasion constituent donc l'arsenal des défenseurs de la sécurité.

L'autorisation est un problème difficile à résoudre dans le cas des systèmes isolés, qui devient réellement compliqué lorsqu'on envisage le cas des systèmes distribués à l'échelle d'Internet. La plupart des modèles de protection sont basés sur la notion de *moniteur de référence* [TCSEC 85] qui contrôle toutes les interactions au sein du système et qui vérifie si chaque accès est autorisé ou refusé, en fonction d'une matrice qui conserve tous les droits d'accès du système. Si l'on considère des applications distribuées, on peut imaginer une application directe de ce modèle en proposant

un moniteur de référence centralisé qui vérifie toutes les interactions dans le système complet. Une implémentation de ce modèle a d'ailleurs été proposée dans [Rushby 83]. L'inconvénient majeur de cette approche est évidemment le fait que la sécurité du système entier repose sur une seule machine, qui devient donc critique. Même si l'on envisage de rendre tolérant aux fautes et aux intrusions ce moniteur de référence central (au moyen de redondance et de séparation des privilèges d'administration par exemple), ce moniteur reste un goulot d'étranglement du point de vue des performances.

Le Livre Rouge [TNI 87] propose une autre solution, qui consiste à utiliser un moniteur de référence local sur chaque machine du système distribué, contrôlant tous les accès aux ressources locales. Le moniteur de référence fait partie de la base de confiance (*Trusted Computing Base* ou TCB) et chaque TCB fait confiance aux autres TCBs du système : lorsqu'une entité réalise un accès sur un objet distant, la TCB de l'objet fait confiance à l'autre TCB concernant l'identité de l'entité, identité qui est utilisée pour les contrôles d'accès. Dans cette approche, le principal problème provient de cette confiance : si n'importe laquelle des TCBs est corrompue, la sécurité du système complet est compromise.

Notre premier objectif est de concevoir un schéma d'autorisation qui soit un compromis entre l'approche centralisée et l'approche totalement distribuée et qui ne présente pas les inconvénients de ces deux approches.

Aujourd'hui, la majorité des applications Internet est basée sur le modèle client-serveur. Ce modèle n'est pas suffisamment riche pour gérer correctement des opérations complexes impliquant plus de deux participants. Par exemple, une transaction de commerce électronique nécessite typiquement la coopération d'un client, d'un marchand, d'une banque, d'un émetteur de carte de crédit, d'une entreprise de livraison, etc. Chaque participant a des intérêts différents et peut ne pas forcément faire confiance aux autres participants. De plus, dans ce modèle, le serveur ne fait pas confiance au client et décide de lui accorder des droits d'accès en fonction de son identité. Cela permet au serveur d'enregistrer un certain nombre d'informations concernant le client : l'identité mais aussi son adresse IP, son adresse postale, son numéro de carte de crédit, ses habitudes d'achats, etc. Un tel modèle est donc intrusif du point de vue de la vie privée.

Notre second objectif est donc d'offrir plus de souplesse que le modèle client-serveur. Il faut pour cela concevoir un schéma d'autorisation qui est capable de distribuer des preuves d'autorisation aux différents participants d'un traitement réparti tout en respectant le "principe du moindre privilège" : chaque participant doit recevoir seulement les preuves d'autorisation strictement nécessaires à l'exécution du traitement réparti.

Nous proposons dans la suite de présenter dans un premier temps (section 3.2) la conception de notre schéma d'autorisation ainsi que les propriétés de sécurité des principales entités de ce schéma. Nous donnons ensuite dans la section 3.3, les définitions des différentes preuves d'autorisation qui sont utilisées dans notre schéma. Les sections 3.4 et 3.5 présentent en détail les deux entités principales de notre schéma : le serveur d'autorisation et le moniteur de référence local. La section 3.6 présente l'originalité de notre méthode de délégation de droits. La section

3.7 présente très brièvement un prototype d'implémentation de ce service d'autorisation. Enfin, la section 3.8 présente des conclusions sur ce travail.

3.2 Conception du service d'autorisation

Cette section est dédiée à la présentation globale du schéma d'autorisation ainsi que des propriétés de sécurité des différents entités qui composent le service d'autorisation.

3.2.1 Architecture

Une des caractéristiques de notre schéma d'autorisation est qu'il est capable de gérer des droits d'accès pour des opérations *composites*. Une opération composite correspond à l'exécution coordonnée de plusieurs opérations élémentaires réalisées par les participants de ce traitement réparti dans un but commun. En revanche, une opération élémentaire correspond à l'exécution d'une tâche (typiquement l'exécution d'une méthode sur un objet) sur un seul site du système réparti. Par exemple, imprimer le fichier F3 sur l'imprimante P4 est une opération composite impliquant l'exécution d'une opération particulière du gestionnaire d'impression attaché à l'imprimante P4, qui lui-même doit doit demander l'exécution d'une opération particulière du gestionnaire de fichiers qui gère F3, etc.

Notre schéma d'autorisation a pour objectif de distribuer des droits d'accès à chaque participant d'une opération composite de telle façon qu'il puisse réaliser sa partie du traitement sous forme d'une opération élémentaire. Seule la preuve que cette opération élémentaire est autorisée ainsi que les paramètres nécessaires à cette exécution sont fournis, sans aucune autre information (comme par exemple les identités des autres participants). Ce schéma est basé sur deux niveaux de protection :

- Un *serveur d'autorisation* est en charge d'accepter ou de refuser l'autorisation d'effectuer chaque opération composite ; si cette opération est autorisée, le serveur génère et distribue les preuves d'autorisation (cf. section 3.3) pour chaque opération élémentaire faisant partie de l'opération composite. Ce serveur doit être conçu de telle sorte qu'il ne soit pas un point dur du système.
- Sur chaque site participant à l'opération composite, un moniteur de référence est responsable de vérifier tous les accès aux ressources locales, en fonction des preuves d'autorisation qui accompagnent chaque requête. L'autorisation d'invoquer la méthode d'un objet[1] est représentée par une *capacité* générée par le serveur d'autorisation et vérifiée par le moniteur de référence. De façon à empêcher les comportements malicieux des moniteurs de référence situés sur des

[1]Par objet et méthode, nous nous référons aux notions bien connues provenant des langages orientés objets. Nous considérons les applications distribuées comme composées d'objets qui peuvent interagir au moyen d'appels de méthode.

machines “ordinaires” sur Internet, les parties critiques du moniteur de référence doivent être implémentées sur du matériel inviolable (cf. section 3.7 pour un exemple à l'aide de Java-Cards).

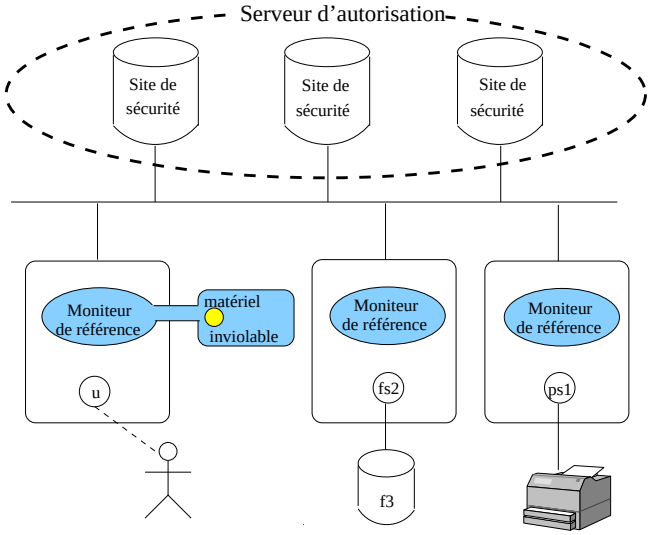


Fig. 3.1 – Architecture d'autorisation

Cette architecture à deux niveaux (voir figure 3.1) permet :

- une simplicité d'administration, puisque seuls les droits d'accès pour les opérations composites doivent être gérés par les administrateurs de sécurité ;
- une protection très fine sur chaque site du système, puisque chaque invocation de méthode est vérifiée.

Il est également intéressant de noter que cette approche est davantage capable de protéger la vie privée que le schéma classique client-serveur : dans le modèle client-serveur, le serveur donne ou refuse l'accès à un client en fonction de son identité (ce qui est enregistré avec d'autres informations qui peuvent être utilisées en cas de litige). Beaucoup plus d'informations que nécessaire sont donc enregistrées. Avec le schéma flexible que nous proposons, une opération est autorisée si la permission correspondante est présentée, ce qui ne dévoile aucune information personnelle a priori.

3.2.2 Propriétés de sécurité

La sécurité du système global est assurée par la collaboration du serveur d'autorisation et des moniteurs de référence. Nous présentons dans cette section les propriétés de sécurité requises pour chacune de ces entités.

3.2.2.1 Propriétés de sécurité du serveur d'autorisation

Le serveur d'autorisation est en charge de créer toutes les preuves d'autorisation qui permettent aux objets dans le système d'effectuer des opérations. En conséquence, la défaillance du serveur d'autorisation compromettrait la sécurité du système tout entier. Par conséquent, les propriétés de sécurité suivantes doivent être assurées :

AS1 : Le serveur d'autorisation génère seulement des preuves d'autorisation valides.

AS2 : Il n'est pas possible d'empêcher le serveur d'autorisation de générer des preuves d'autorisation valides.

3.2.2.2 Propriétés de sécurité du moniteur de référence

Notre schéma d'autorisation concerne les applications réparties sur Internet, c'est-à-dire des applications qui nécessitent la coopération entre objets situés sur des sites distribués géographiquement. Ces machines peuvent, accidentellement ou malicieusement, être défaillantes ou compromises. En particulier, nous considérons que chaque machine est administrée localement par une personne qui a un contrôle total sur cette machine et qui n'est pas nécessairement une personne de confiance. Cet administrateur ne fait pas non plus confiance nécessairement aux administrateurs des autres machines. Ainsi, il n'y a pas de confiance mutuelle entre les machines qui exécutent les différents objets qui participent à une même application répartie. Nous nommons "machine fautive" une machine compromise ou défaillante.

Cependant, même si certaines machines peuvent être fautives, les propriétés de sécurités suivantes doivent être assurées :

RM1 : Seules les opérations autorisées peuvent être exécutées sur une machine non fautive.

RM2 : Il n'est pas possible d'empêcher l'exécution d'une opération autorisée sur une machine non fautive.

Nous montrons dans les sections 3.4 et 3.5 comment nous avons conçu le serveur d'autorisation et les moniteurs de référence de telle sorte que ces propriétés de sécurités soient assurées.

3.3 Preuves d'autorisation

3.3.1 Concepts et définitions

Dans notre schéma, une application peut être vue à deux niveaux d'abstraction : exécutions de méthodes et opérations composites. Une exécution de méthode est simplement l'exécution

déclenchée par l'invocation d'une méthode d'un objet particulier. Une opération composite correspond à l'exécution coordonnée de plusieurs méthodes d'objets dans un but commun.

De plus amples informations sur la façon dont le serveur d'autorisation vérifie si une operation composite est autorisée ou refusée sont données dans [Nicomette et Deswarte 96] et [Deswarte 2002]. Nous en donnons ici une brève description.

Une requête d'un objet O pour exécuter une opération composite est autorisée ou refusée par le serveur d'autorisation en fonction de *règles de droits symboliques* appliquées à des *droits symboliques* stockés dans une matrice de droits. Un droit symbolique est un droit de haut niveau qui est utilisé pour autoriser l'exécution d'une opération composite dans le système. Par exemple, le droit d'imprimer le fichier f sur n'importe quelle imprimante du système est un droit symbolique. Pour imprimer le fichier f sur l'imprimante p, il suffit d'avoir à la fois le droit symbolique pour imprimer f sur n'importe quelle imprimante et le droit symbolique d'imprimer n'importe quel fichier sur p. En général, plusieurs droits symboliques sont nécessaires pour autoriser la réalisation d'une seule opération composite. Pour autoriser l'opération composite, le serveur évalue les règles de droits symboliques. Une règle de droits symboliques décrit l'ensemble de tous les droits symboliques qui doivent être présents dans la matrice des droits d'accès de façon à autoriser la réalisation de l'opération composite.

Si la requête est autorisée, les preuves d'autorisation sont créées par le serveur d'autorisation pour toutes opérations composant l'opération composite globale. Ces preuves d'autorisation sont délivrées sous la forme d'une liste de *permissions* pour O lui permettant de réaliser les opérations op_1, ... , op_n. Cette liste est signée par le serveur d'autorisation (AS) pour éviter la création de fausses permissions :

$(\{\mathbf{Perm}(O; op_1); ...; \mathbf{Perm}(O; op_n)\})_{SK_{as}}$

La notion d'opération composite est récursive : l'exécution d'une opération composite correspond à l'exécution d'un ensemble d'opérations, chacune d'entre elles pouvant être une simple exécution de méthode mais aussi une autre opération composite.

Des permissions différentes sont associées aux invocations de méthodes simples et aux opérations composites.

3.3.1.1 Invocations de méthodes

La permission associée à l'invocation par l'objet O de la méthode *O'.m* (c'est-à-dire de la méthode m' sur l'objet O') contient l'identification de l'objet O autorisé à invoquer la méthode, l'identification de la méthode autorisée *O'.m*, des éventuelles contraintes sur les paramètres d'appel *parC*, une *capacité* pour O lui permettant d'invoquer *O'.m* avec ces contraintes et optionnellement un *coupon* (que nous définissons plus bas) :

$$\mathbf{Perm}(O; O'.m) = \{O; O'.m(parC); \mathbf{Cap}(O; O'.m(parC))[; \mathbf{Vouch}(O'.m)]\}$$

La capacité $\mathbf{Cap}(O; O'.m(parC))$ est la preuve que l'objet O est autorisé à invoquer la méthode m de l'objet O' avec les contraintes $parC$ sur les paramètres d'invocation. Ces contraintes peuvent être simplement des valeurs autorisées pour les paramètres ou des relations sur les valeurs des paramètres, tel qu'un maximum autorisé. Quand O invoque $O'.m$, l'invocation doit être accompagnée de la capacité, qui sera vérifiée par le moniteur de référence situé sur la machine de O'.

Si, durant cette exécution, $O'.m$ a besoin d'invoquer d'autres opérations (méthodes simples ou opérations composites), elle aura besoin de preuves d'autorisation pour ces invocations. Dans ce cas, $\mathbf{Perm}(O; O'.m)$ contient aussi un *coupon* (noté **Vouch**), qui est une liste signée de permissions pour O' lui permettant d'exécuter $op_1, ..., op_n$:

$$\mathbf{Vouch}(O'.m) = (\{\mathbf{Perm}(O', op_1); ...; \mathbf{Perm}(O', op_n)\})_{SK_{as}}$$

$\mathbf{Vouch}(O'.m)$ sera transmis par O à O' quand O invoquera $O'.m$, de la même façon que la liste de permissions est transmise par le serveur d'autorisation à O en réponse à la requête pour demander l'autorisation d'exécution d'une opération composite. Le coupon est créé par le serveur d'autorisation lors de la génération des permissions, et il est signé par le serveur d'autorisation pour éviter la fabrication des faux coupons. Notons que les preuves d'autorisation représentées par le coupon ne peuvent pas être directement utilisées par O. Elles peuvent seulement être déléguées par O à O' de façon à invoquer la méthode m pour le compte de O (l'identité de O' est insérée dans ces permissions). Cette notion de délégation de droits est fondamentale dans notre schéma et son originalité sera discutée plus en détail dans la section 3.6.

3.3.1.2 Opérations composites

La permission associée à une opération composite contient l'identification de l'objet O autorisé à l'invoquer, l'identification de l'opération cop, des contraintes $parC$ sur les paramètres d'invocation et un *jeton* (noté **Token**) pour O lui permettant d'invoquer cop avec ces contraintes :

$$\mathbf{Perm}(O; cop) = \{O; cop(parC); \mathbf{Token}(O; cop(parC))\}$$

$\mathbf{Token}(O; cop(parC))$ constitue la preuve que l'objet O est autorisé à lancer l'exécution de l'opération composite cop avec les contraintes $parC$ sur les paramètres d'invocation. Avec ce jeton, l'objet O peut demander au serveur d'autorisation toutes les preuves d'autorisation

nécessaires à l'exécution de *cop* avec les contraintes *parC*.

3.4 Le serveur d'autorisation

Puisque seules les opérations composites sont gérées par le serveur d'autorisation, la sécurité du système est relativement facile à gérer : les administrateurs de sécurité doivent simplement affecter les droits pour exécuter des opérations composites. De plus, comme une seule requête doit être vérifiée par le serveur pour l'autorisation d'une opération composite, le coût des communications est moindre que s'il fallait contacter ce serveur à chaque opération élémentaire.

Le serveur d'autorisation est une tierce partie de confiance qui pourrait être un point dur du système, à la fois en cas de défaillance accidentelle ou en cas d'intrusion réussie (intrusion pouvant être perpétrée par un administrateur corrompu). Pour éviter cela, le serveur d'autorisation peut être rendu tolérant aux fautes accidentelles et aux intrusions [Deswarte 2002]. Dans ce cas, le serveur est composé de plusieurs *sites de sécurité*, administrés par des personnes indépendantes, de telle façon que les fautes accidentelles et les intrusions puissent être tolérées sans dégradation du service, tant qu'une minorité seulement de sites sont affectés.

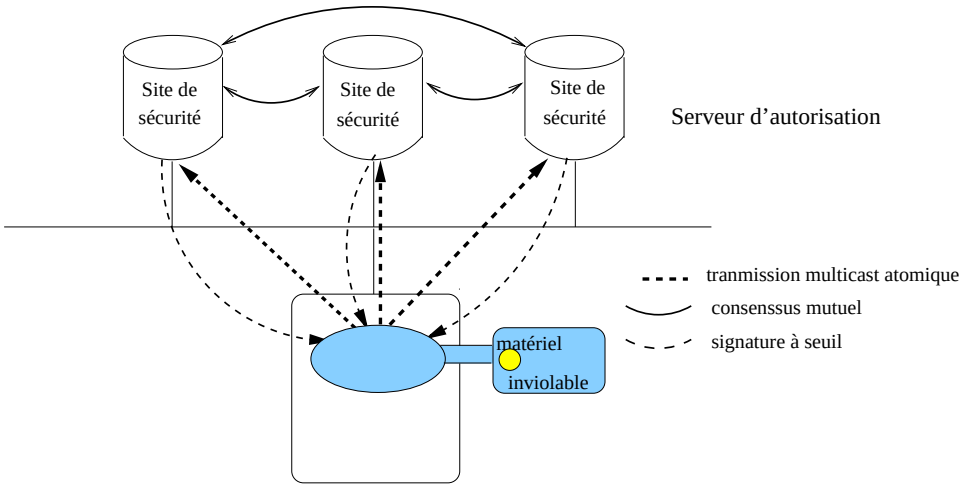


FIG. 3.2 – Protocoles du serveur d'autorisation

De façon à tolérer la défaillance d'un ou d'un petit nombre de sites qui composent le serveur d'autorisation, plusieurs protocoles spécifiés dans [Cachin 2000], [Cachin 2001] et [Algesheimer et al. 2001] sont utilisés (cf. figure 3.2) :

– *Transmission multicast atomique* : une requête envoyée par un objet au serveur d'autorisation est distribuée sur tous les sites qui composent le serveur d'autorisation. La communication multicast est fiable dans le sens que toutes les requêtes sont correctement reçues par tous les

sites non défaillants, et atomique dans le sens que les sites non défaillants reçoivent l'ensemble des messages dans le même ordre. Cette condition est nécessaire pour implémenter le consensus mutuel, et elle est fournie par les protocoles développés dans le projet MAFTIA (cf. chapitre 3.7).

- *Consensus mutuel* : tous les sites non défaillants se mettent d'accord sur la décision d'autorisation correspondant à une requête donnée. Ceci garantit une décision correcte tant qu'il y a au plus f sites fautifs, où le nombre f dépend de certains conditions. Si nous supposons que le mode de défaillance des sites fautifs est le silence sur défaillance (grâce à du matériel spécifique comme présenté dans [Veríssimo et al. 2006]), alors f doit vérifier $(n > 2f)$, où n est le nombre total de sites. Si nous ne faisons pas cette hypothèse sur le mode de défaillance de sites, les fautes Byzantines doivent être prises en compte et, sous ces hypothèses, f doit vérifier $(n > 3f)$ [Cachin 2000].
- *Signature à seuil* : les capacités, jetons et coupons sont globalement signés par le serveur d'autorisation, à l'aide d'un schéma de signatures à seuil. Pour cela, chacun des sites composant le serveur d'autorisation génère une partie de la signature (à partir de sa propre partie de la clé privée) de telle façon que si au moins t parties de signature sont disponibles, alors il est possible de combiner ces parties pour générer une signature unique qui peut être vérifiée à l'aide d'une clé publique globale. Ceci permet de garantir que si une capacité (ou un jeton ou un coupon) est correctement signée, l'opération correspondante est autorisée (la capacité signée ne peut être forgée, même par la coopération de f sites fautifs, tant que f est strictement inférieur au seuil t).

Cet ensemble de protocoles (réalisés dans le projet MAFTIA) permet d'assurer que la propriété de sécurité AS1 (cf. section 3.2.2.1) est respectée. En effet, nous pouvons tolérer un nombre f de sites fautifs : f dépend de l'hypothèse concernant le mode de défaillance des sites fautifs, comme nous l'avons expliqué précédemment ($n > 2f$ ou $n > 3f$), et, de plus, f doit être plus petit que t. Sous l'hypothèse qu'il n'y a pas plus de f sites fautifs, le serveur d'autorisation ne peut pas générer des fausses preuves d'autorisation (propriété AS1), parce que le protocole de consensus mutuel garantit une décision correcte et parce que le schéma de signatures à seuil garantit une signature correcte des preuves d'autorisation.

En ce qui concerne la propriété AS2, des mécanismes de prévention et de tolérance doivent être utilisés de façon à ce que les attaques de déni de service ne puissent réussir et qu'aucune défaillance du réseau ne puisse survenir. L'étude de ces mécanismes ne fait pas l'objet de nos travaux mais on peut néanmoins donner quelques exemples : les pare-feux peuvent filtrer les paquets suspects, les systèmes de détection d'intrusions tels que Snort [Snort] peuvent identifier les tentatives d'intrusion et lever des alarmes, des techniques de redondance et de reconfiguration peuvent permettre aux réseaux de rester opérationnels même en présence d'attaques réussies sur une partie de ces réseaux.

Le dialogue entre un objet initiant une opération composite et le serveur d'autorisation est typiquement le suivant. L'objet O lui envoie une requête d'autorisation de l'opération composite. Si l'autorisation est accordée par la majorité des sites composant le serveur d'autorisation, chaque

site non fautif renvoie à O la même liste de permissions pour les invocations de méthodes, accompagnée de sa partie de signature. Si O reçoit au moins t copies de cette liste, O peut reconstruire la signature globale pour cette liste :

$$(\{\mathbf{Perm}(O; O'_1.m_1); ...; \mathbf{Perm}(O; O'_n.m_n)\})_{SK_{as}}$$

Chacune de ces permissions contient une capacité $\mathbf{Cap}(O; O'_i.m_i(parC))$ ou un jeton.

Pour éviter la contre-façon ou l'utilisation malveillante de capacités, chaque capacité est chiffrée[2] avec la clé publique $PK_{h(O'_i)}$ du site hôte de O'_i et signée avec la clé privée globale SK_{as} du serveur d'autorisation :

$$\mathbf{Cap}(O; O'_i.m_i(parC)) = ((O; O'_i.m_i; parC; nonce)_{PK_{h(O'_i)}})_{SK_{as}}$$

Le *nonce* est utilisé pour éviter les rejeux.

Si la capacité est accompagnée par un coupon, ce coupon contient un ensemble de permissions dont a besoin O'_i pour exécuter m_i pour le compte de O. Ce coupon est lui-même signé par la clé privée globale SK_{as} du serveur d'autorisation :

$$\mathbf{Vouch}(O'_i.m_i) = (\{\mathbf{Perm}(O'; op_1); ...; \mathbf{Perm}(O'; p_n)\})_{SK_{as}}$$

3.5 Le moniteur de référence

Un moniteur de référence est présent sur chaque site participant à une opération composite dans notre architecture. Ce moniteur de référence a la responsabilité d'accorder ou refuser les invocations de méthodes sur les objets locaux, en fonction des capacités générées par le serveur d'autorisation. Dans le contexte de grands réseaux tels qu'Internet, l'implémentation du moniteur de référence est complexe car, à cause de l'hétérogénéité des machines connectées, il serait nécessaire de développer une version particulière du moniteur de référence pour chaque type de machine. De plus, puisque les machines ne sont pas sous le contrôle d'une autorité globale, il n'y a aucun moyen d'assurer que chaque machine exécute une version authentique du moniteur de référence.

[2]La capacité est la seule preuve d'autorisation qui nécessite d'être chiffrée et signée ; les jetons sont seulement signés car la connaissance de leur contenu ne peut donner aucun privilège. Cependant, il pourrait être utile aussi de chiffrer (avec la clé publique du site de O) la liste des permissions et des coupons, si la connaissance de leurs contenus est considérée comme une atteinte à la vie privée.

Le moniteur de référence est donc composé de deux parties : une partie matérielle inviolable et une partie logicielle, appelée *dispatcher*. Le dispatcher est un objet local, exécuté par chaque machine, qui est en charge de contrôler toutes les invocations des objets locaux. Le dispatcher stocke et gère également toutes les permissions qui ont été accordées aux objets locaux de la machine. Par exemple, quand un objet local invoque un objet distant, le dispatcher local vérifie les permissions appartenant à l'objet source afin de vérifier qu'il a bien une capacité lui permettant d'effectuer cette invocation. Il insère alors cette capacité et (optionnellement) le coupon dans le message d'invocation et envoie le tout au dispatcher distant, qui à son tour vérifiera la capacité et stockera les permissions incluses dans le coupon pour l'objet invoqué.

Le dispatcher décide d'autoriser ou non l'invocation de méthodes particulières sur des objets locaux particuliers en demandant à la partie matérielle inviolable de vérifier que les capacités de l'objet appelant sont bien valides (au travers de routines de vérification de signatures). Ces vérifications représentent la partie centrale du schéma d'autorisation et doivent donc être protégées aussi efficacement que possible, d'où le choix de leur implémentation matérielle. Notons que tout logiciel, même faisant partie du système d'exploitation lui-même, peut être copié et modifié par un utilisateur malveillant qui possède les privilèges nécessaires sur la machine. En particulier, sur Internet, n'importe quel pirate peut facilement avoir les privilèges d'administrateur sur sa propre machine ! Aussi, grâce à la vérification matérielle des capacités, nous pouvons avoir confiance que chaque requête externe est saine (si la capacité est correcte) et qu'une requête saine peut seulement être exécutée sur la machine pour laquelle la capacité est valide. Les privilèges qu'un pirate peut acquérir en corrompant localement sa machine ne lui donne donc aucun privilège en dehors de cette machine.

Un couple de clés privée/publique est associé à chaque machine et il est stocké sur la partie matérielle inviolable. La clé publique du serveur d'autorisation est également stockée dans cette partie inviolable. Les clés publiques sont stockées dans cette partie matérielle pour des raisons d'intégrité, afin d'empêcher des attaques qui viseraient à les modifier. La clé privée de la machine est générée et stockée sur la partie inviolable pour des raisons de confidentialité : la clé privée ne doit jamais quitter cette partie inviolable, elle ne peut pas être lue ni modifiée. Chaque capacité générée par le serveur d'autorisation est chiffrée par la clé publique de la machine sur laquelle l'objet invoqué est situé et ensuite signé par la clé privée du serveur d'autorisation.

Ces techniques permettent d'implémenter la propriété RM1 (cf. section 3.2.2.2) : sur une machine non fautive, si la capacité n'est pas correctement chiffrée et signée, l'opération correspondante ne sera pas exécutée. Nous faisons confiance au moniteur de référence pour vérifier correctement les capacités. Nous pensons qu'il est raisonnable de lui faire confiance parce que nous avons confiance dans le code et les données qui sont stockés dans la partie matérielle inviolable. De plus, une machine fautive ne peut pas générer de fausses permissions pour les autres machines puisque ces permissions doivent être signées par la clé privée du serveur d'autorisation.

En ce qui concerne RM2, puisque nous voulons empêcher une machine fautive de se faire passer pour une machine non fautive sans être détectée, un mécanisme d'acquittement peut être utilisé :

chaque fois qu'une preuve d'autorisation est reçue par une machine, cette dernière retourne un message d'acquittement signé par sa propre clé privée. Cet aquittement est accompagné par le certificat généré par le serveur d'autorisation. Même si un site fautif essaie de se faire passer pour une site non fautif, il ne peut pas envoyer cet aquittement car il ne peut pas le chiffrer avec la clé privée du site non fautif (cette clé n'est jamais transmise hors de la partie matérielle inviolable). En conséquence, même si la machine fautive intercepte les messages à destination d'une machine non fautive (provoquant un déni de service), l'attaque sera détectée parce que l'objet émetteur du message ne recevra pas un aquittement correct. Ce mécanisme de protection est également utilisé contre les attaques de déni de service sur le réseau ou contre les défaillances du réseau. Nous pouvons faire confiance à ce mécanisme d'aquittement, parce que, encore une fois, nous faisons confiance à la partie matérielle inviolable de ne pas révéler les clés cryptographiques. Cette confiance est justifiée par le fait qu'accéder aux données stockées dans cette partie matérielle est suffisamment difficile.

Insistons sur le point suivant : les mécanismes présentés ne sont pas suffisants pour garantir le comportement correct d'une application complète. Si une machine est corrompue, les processus locaux peuvent faire n'importe quoi localement et donc peuvent changer complètement leur comportement par rapport à l'application globale. Ils peuvent décider d'invoquer les opérations sur les objets locaux qui ne font pas partie de l'opération composite, ou invoquer constamment la même méthode du même objet local, ou ne pas invoquer les opérations nécessaires sur les objets situés sur d'autres machines. Les mécanismes que nous avons implémentés permettent de respecter la politique de sécurité du système et ne peuvent garantir que le comportement de chaque processus est correct vis-à-vis de l'application dont il fait partie. D'autres mécanismes tels que la réplication d'objets sont nécessaires pour gérer ces problèmes. En résumé, les mécanismes d'autorisation garantissent que seules les opérations autorisées sont exécutées sur une machine non fautive et qu'une machine fautive ne peut empêcher les autres machines non fautives de fonctionner correctement.

3.6 Discussion sur la notion de délégation

Notre schéma de délégation basé sur la notion de coupon est plus flexible que le schéma usuel basé sur la notion de procuration [Neuman 1993, Tardo et Alagappan 1991, Gasser et MacDermott 1990, Parker 1991], schéma selon lequel un objet transmet à un autre objet une partie de ses droits d'accès de façon à ce que l'objet délégué puisse exécuter des opérations pour le compte de l'objet délégateur. Notre schéma est également plus fidèle au principe du moindre privilège, puisqu'il permet de réduire les privilèges nécessaires à la réalisation d'opérations déléguées. Par exemple, si un objet O est autorisé à imprimer un fichier, il doit déléguer le droit de lecture au gestionnaire d'impressions pour qu'il puisse lire le fichier à imprimer. Pour déléguer ce droit, avec le schéma classique de procuration, O doit posséder ce droit de lecture ; par conséquent O pourrait outrepasser ses droits en faisant des copies du fichier et en les distribuant. Le droit de lecture est donc un droit bien supérieur au droit d'imprimer. Dans notre schéma, si O est autorisé à imprimer un

fichier, il reçoit la capacité lui permettant d'invoquer le gestionnaire d'impressions et un coupon permettant au gestionnaire d'impressions de lire le fichier. Le coupon ne peut être utilisé par O. La capacité qu'il a reçue lui permet seulement d'invoquer le gestionnaire d'impressions et lui transmettre le coupon. Le gestionnaire d'impressions peut alors utiliser la capacité contenue dans le coupon et lire le fichier. Ainsi, O peut imprimer un fichier sans avoir le droit de le lire.

L'idée de gérer et déléguer des droits pour des opérations élémentaires, c'est-à-dire une autorisation et une délégation à grains fins a été explorée dans le modèle RBAC (*Role-based access control*), par exemple [Zhang et al. 2003, Wainer et Kumar 2005]. Dans ces articles, les auteurs présentent la notion de délégation dans le modèle RBAC comme une délégation de rôle ; c'est donc une délégation du type utilisateur à autre utilisateur. Ils proposent donc de rendre leur modèle plus flexible afin de contrôler plus finement les droits qu'un utilisateur peut déléguer. Notre schéma d'autorisation implémente ce type d'autorisation à grains fins et ce principe de délégation mais nous considérons qu'il va au-delà. En effet, dans les modèles RBAC, le droit de délégation reste malgré tout une notion classique dans laquelle une entité peut seulement déléguer un droit que si elle le possède déjà par ailleurs. Notre modèle est plus flexible dans le sens qu'il est possible de déléguer une action qu'on n'a pas le droit de réaliser soi-même.

Crispo, dans [Crispo et Christianson 1999, Crispo 1998, Crispo 2001], étudie également la notion de délégation. Il distingue la délégation de droits et la délégation de responsabilités et précise qu'il est important pour la traçabilité des systèmes opératoires de distinguer ces deux aspects. Crispo donne plusieurs exemples de situations réelles de délégation complexe. Il explique que la délégation de responsabilités n'est pas triviale et qu'elle est complémentaire de la notion de délégation de droits. Notre schéma propose une façon très simple de résoudre ce problème de la responsabilité. Chaque objet qui exécute une opération élémentaire dans notre système agit pour son propre compte et est donc entièrement responsable de cette opération (la traçabilité est de plus parfaitement assurée). Dans les systèmes de délégation habituels, quand B exécute une opération pour la compte de A (grâce aux droits délégués de A), A reste responsable de cette opération. Ceci ne permet pas d'identifier B comme fautif ou malveillant. Dans notre modèle, les droits délégués de A à B sont des droits qui peuvent être utilisés par B et par personne d'autre[3]. Aussi, B est identifié comme l'entité responsable de l'exécution des opérations autorisées par ces droits. Dans un cas d'action malveillante, B sera donc suspecté d'abord.

3.7 Le prototype du projet MAFTIA

Dans le projet MAFTIA[4], un prototype d'un schéma d'autorisation complet a été implémenté. Dans ce prototype, nous avons choisi d'implémenter tous les objets qui composent une application distribuée en Java. Les invocations entre objets sont réalisées au moyen des RMIs (*Remote Method Invocations*) Java. La partie matérielle sécurisée (*tamper-resistant*) du moniteur

[3] L'identité de B est incluse dans la capacité ou le jeton correspondant.

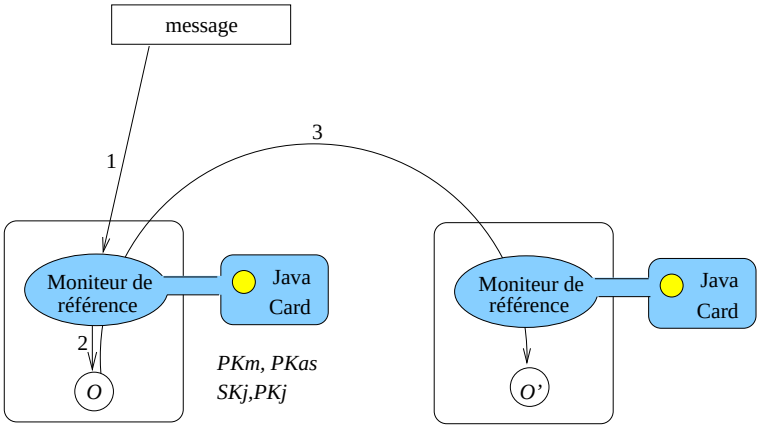
[4] European project IST 1999-11583 : Malicious-and Accidental-Fault Tolerance for Internet Applications.

de référence est constituée d'une JavaCard. Même si ce type de matériel n'est pas totalement inviolable, ainsi que l'ont montré un certain nombre d'attaques [Quisquater et Samyde 2001, Iguchi-Cartigny et Lanet 2009], en particulier par canal de fuite (side-channel attacks), nous avons considéré que nous pouvions lui faire suffisamment confiance dans le cadre de l'utilisation que nous en avons faite dans ce projet. En effet, même si une JavaCard était compromise, cela ne permettrait pas au site corrompu de se faire passer pour un autre site, ce qui limite donc la portée de ces attaques, par ailleurs de plus en plus difficiles à mettre en œuvre en raison des nouvelles contre-mesures que développent les fabricants de carte à puce et autres modules matériels sécurisés.

Plusieurs clés cryptographiques sont stockées sur la JavaCard.

- La clé publique MAFTIA, notée PK_m. Cette clé est associée à la clé privée MAFTIA SK_m, qui n'est pas stockée dans la Java Card. SK_m est utilisée pour signer des codes compatibles MAFTIA.
- Un couple de clés privée/publique spécifique à la JavaCard j, que nous notons SK_j, PK_j.
- La clé publique du serveur d'autorisation, PK_{as}. Cette clé est associé aux parties de clé privée des sites d'autorisation, grâce à un algorithme de signatures à seuil (cf. Section 3.4).

Une application distribuée dans le contexte de MAFTIA est exécutée au moyen d'invocations de méthodes entre les objets MAFTIA situés sur des machines compatibles MAFTIA. Les codes des

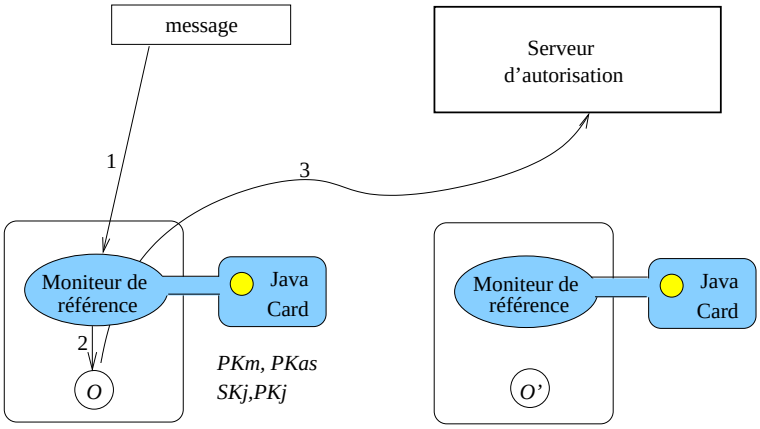


1. Message d'invocation comportant une capacité et un coupon reçu par le moniteur de référence local
2. La méthode correspondante de 0 est invoquée lorsque la capacité autorisant cet accès a été vérifiée par la JavaCard
3. Le coupon contient la capacité pour O d'invoquer une méthode de O'. Cette capacité est vérifiée par le moniteur de référence de la machine de O' quand O invoque la méthode correspondante de O'

FIG. 3.3 – Exemple d'un coupon contenant une capacité pour un objet distant

objets MAFTIA sont signés par la clé privée globale MAFTIA SK_m et cette signature est vérifiée par la JVM locale de chaque machine MAFTIA en utilisant la clé publique correspondante PK_m, qui peut être lue sur la JavaCard par la JVM. Les paramètres d'invocation et les valeurs de retour sont donc échangés entre les objets par des messages du type RMIs Java.

Chaque capacité est chiffrée par la clé publique PK_j de la JavaCard du site sur lequel l'objet invoqué est situé et signée par la clé privée SK_{as} du serveur d'autorisation. La vérification des capacités est faite en trois étapes : 1) la JavaCard vérifie la signature de la capacité en utilisant la clé publique du serveur d'autorisation PK_{as} ; 2) la JavaCard déchiffre la capacité avec sa propre clé privée SK_j ; 3) la JavaCard vérifie que l'invocation est compatible avec le contenu de la capacité (identité de l'objet qui invoque, identité de la méthode invoquée incluant l'identité de l'objet invoqué, compatibilité des paramètres d'invocation avec les contraintes incluses dans la capacité).



1. Un message d'invocation comprenant une capacité et un coupon est reçu par le moniteur de référence local
2. La méthode correspondante de O est invoquée lorsque la capacité autorisant cet accès a été vérifiée par la JavaCard
3. Le coupon contient un jeton pour que O puisse exécuter une opération composite. Le jeton est présenté au serveur d'autorisation afin d'obtenir les permissions correspondantes

FIG. 3.4 – Exemple d'un coupon contenant un jeton

Comme nous l'avons expliqué dans la section 3.3, une capacité est généralement accompagnée par un coupon qui contient les permissions permettant à la méthode appelée d'invoquer à son tour d'autres opérations.

Une permission peut être une capacité d'invoquer une méthode d'objet ou un jeton permettant d'initialiser une opération composite. Si la permission est une capacité, celle-ci est chiffrée avec la clé publique de la JavaCard associée à la machine sur laquelle l'objet invoqué est situé et est vérifiée par la JavaCard correspondante (étape 3 de la figure 3.3). Si la permission est une

autorisation de réaliser une opération composite, le jeton est présenté au serveur d'autorisation en tant que preuve que l'objet est autorisé à réaliser cette opération (étape 3 de la figure 3.4). Le serveur d'autorisation vérifie simplement la validité de la signature de ce jeton et délivre à l'objet les permissions correspondantes.

Par ailleurs, des mesures de performance ont été réalisées de façon à estimer le coût des mécanismes de sécurité inclus dans cette architecture. Ces expérimentations sont décrites en détail dans [Abghour 2004]. Les tests réalisés montrent que les temps d'exécution sont satisfaisants mais ils restent importants et ceci principalement à cause de la technologie JavaCard que nous avons utilisée en 2002. En effet, les communications de type série avec cette JavaCard sont très lentes (communications à 9600bps) et la longueur maximale des messages qui peuvent être envoyés à la JavaCard est trop petite (256 octets), d'où la nécessité de découper les messages avant de les envoyer à la JavaCard et nécessité de les reconstituer ensuite. Les résultats seraient bien meilleurs avec les JavaCards plus puissantes aujourd'hui disponibles.

3.8 Conclusion

Ce chapitre présente l'architecture d'un service d'autorisation pour des opérations complexes impliquant plusieurs entités. L'architecture proposée avait deux objectifs : 1) trouver un compromis entre un schéma totalement centralisé et totalement distribué (qui ont tous les deux des inconvénients significatifs) et 2) concevoir un schéma qui est capable de distribuer des preuves d'autorisation aux participants d'une opération complexe tout en respectant le principe du moindre privilège.

L'architecture repose sur deux concepts principaux : le serveur d'autorisation et les moniteurs de référence. Le serveur d'autorisation est chargé d'accorder ou refuser des droits pour des opérations composites impliquant plusieurs participants. Il est construit de façon à ne pas être un point dur du système. Sur chaque site participant, un moniteur de référence est responsable de l'autorisation à grains fins, c'est-à-dire de contrôler les accès à tous les objets locaux, selon les preuves d'autorisation qui accompagnent chaque requête.

La conception et les propriétés de sécurité de ces composants ont été présentées, ainsi qu'un cas concret d'implémentation dans le cadre du projet européen MAFTIA. Des mesures de performance ont été réalisées et elles montrent que les temps d'exécution sont satisfaisants.

Bibliographie

[TCSEC 85] *US Departement of Defense Trusted Computer Security Evaluation Criteria (TCSEC)*, 5200.28-STD, december 1985.

[Rushby 83] J.M. Rushby et B. Randell, "A Distributed Secure System", *IEEE Computer*, vol. 16, n. 7, July 1983, pp. 55-67.

[TNI 87] *Trusted Network Interpretation of the Trusted Computer Security Evaluation Criteria*, National Computer Security, Center, december 1987.

[Nicomette et Deswarte 96] V. Nicomette et Y. Deswarte, "Symbolic Rights and Vouchers for Access Control in Distributed Object Systems", *Proc. 2nd Asian Computing Science Conference (ASIAN'96)*, Singapour, 1996, "Concurrency and Parallelism, Programming, Networking and Security", J. Jaffar and R.H.C. Yap, Eds., Springer-Verlag n 1179, pp. 193-203.

[Deswarte 2002] Y. Deswarte, N. Abghour, V. Nicomette et D. Powell, "An Intrusion-Tolerant Authorization Scheme for Internet Applications", *Workshop on Intrusion Tolerant Systems*, Supp. of the Proc. 2002 International Conference on Dependable Systems and Networks (DSN 2002), Washington D.C., 23-26 June 2002, pp. C.1.1-C.1.6

[Cachin 2000] C. Cachin, K. Kursawe et V. Shoup, "Random Oracles in Constantinople : Practical asynchronous Byzantine agreement using cryptography", in *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 123-132, 2000.

[Cachin 2001] C. Cachin, M. Correia, T. McCutcheon. N.F. Neves, B. Pfitzmann, B. Randell, M. Schunter, W. Simmonds, R. Stroud, P. Veríssimo, M. Waidner et I. Welch, "Service and Protocol Architecture for the MAFTIA Middleware", *MAFTIA Project IST 1999-11583 Deliverable D23*, 92 pp., 25 January 2001, available at : <http ://www.maftia.org/deliverables/D23final.pdf>.

[Algesheimer et al. 2001] J. Algesheimer, C. Cachin, K. Kursawe, F. Petzold, J.A. Poritz, V. Shoup et M. Waidner, "MAFTIA : Specification of Dependable Trusted Third Parties", IBM Research, Zurich Research Laboratory, Zurich (CH), *MAFTIA Project IST 1999-11583 Deliverable D26*, 98 pp., 22 January 2001, available at : <http ://www.maftia.org/maftia/deliverables/D26-final2.pdf>.

[Veríssimo et al. 2006] P.E Veríssimo, N. F. Neves, C. Cachin, J. Poritz, D. Powell, Y. Deswarte, R. Stroud et I. Welch, "Intrusion-Tolerant Middleware", *IEEE Security and Privacy*, July/August 2006, pp. 54-62.

[Snort] http ://www.snort.org

[Neuman 1993] B. Neuman "Proxy-based Authorization and Accounting for Distributed System" in *Proceedings of the 13th International Conference on Distributed Systems*, Pittsburgh, PA, USA, 25-28 May 1993, pp. 283-291.

[Tardo et Alagappan 1991] J. Tardo et K. Alagappan "Spx : Global Authentication Using Public Key Certificates", *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 20-22 May 1991, pp. 232-244.

[Gasser et MacDermott 1990] M. Gasser et E. MacDermott "An Architecture for Practical Delegation in a Distributed System", *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 7-9 May 1990, pp. 20-30.

[Parker 1991] T.A. Parker,"A Secure European System for Applications in a Multi-vendor Environment (The SESAME Project)", *Proc. of the 14th National Computer Security Conference*, NCSC and NIST, Washington, pp. 505-513, October 1991.

[Zhang et al. 2003] X. Zhang , S. Oh et R. Sandhu, "PBDM : a Flexible Delegation Model in RBAC", *Proceedings of the eighth ACM symposium on Access control models and technologies*, June 02-03, 2003, Como, Italy, pp. 149-157.

[Wainer et Kumar 2005] J. Wainer et A. Kumar, "A Fine-grained, Controllable, User-to-user Delegation Method in RBAC", *Proceedings of the tenth ACM symposium on Access control models and technologies*, pp. 59-66, Stockholm, Sweden, ISBN :1-59593-045-0, 2005.

[Crispo et Christianson 1999] B. Crispo et B. Christianson, "A Note About the Semantics of Delegation", *Proceedings of the 3rd Conference on Autonomous Agents (Agent '99)*, Seattle, May 1-5 1999, pp. 55-64.

[Crispo 1998] B. Crispo, "Delegation of Responsibilities", *Proceedings of the 1998 Security Protocols International Workshop*, April 15 - 17, 1998, Cambridge, UK, Springer-Verlag LNCS vol. 1550, ISBN 3-540-65663-4, pp. 118-13

[Crispo 2001] B. Crispo, "Delegation Protocols for Electronic Commerce", *Proceedings of the 6th IEEE Symposium on computers and communications (ISCC01)*, Hammamet, Tunisia, IEEE press, 3-5 May 2001, pp. 674-679.

[Quisquater et Samyde 2001] Jean-Jacques Quisquater et David Samyde, "ElectroMagnetic Analysis (EMA) : Measures and Counter-Measures for Smart Cards", *Proceedings of the E-smart 2001 conference*, pp. 200-210, 2001.

[Iguchi-Cartigny et Lanet 2009] J. Iguchi-Cartigny et J.L. Lanet, "Evaluation de l'injection de code malicieux dans une JavaCard", *Proceedings du Symposium sur la Sécurité des Technologies de l'Information et des Communications SSTIC*, 2009, pp. 3-18.

[Abghour 2004] N. Abghour, "Schéma d'autorisation pour applications réparties sur Internet", *Thèse de Doctorat de l'Institut National Polytechnique de Toulouse*, juin 2004, Rapport LAAS n^{o}04327.

Chapitre 4

Travaux de recherche : conception et réalisation de serveurs Web tolérant les intrusions

4.1 Problématique

Comme nous l'avons précisé au chapitre 3, la sécurité informatique repose principalement sur l'authentification des utilisateurs et sur l'autorisation. Malheureusement, cet arsenal est peu efficace dans la plupart des systèmes connectés à l'Internet. En effet, tout internaute, même anonyme, a des droits sur toute machine connectée : par exemple, lire les pages des serveurs Web publics, mais aussi connaître l'existence de cette machine et l'identifier par son nom ou son adresse. L'authentification des utilisateurs, lorsqu'elle existe, repose généralement sur des mécanismes faibles, comme un mot de passe qu'il est facile pour un utilisateur de transmettre à un autre, lui permettant ainsi de se faire passer pour l'utilisateur légitime ; d'ailleurs, même sans la coopération de l'utilisateur, il est souvent facile de deviner un mot de passe. Un autre problème lié à l'authentification sur l'Internet tient à sa distribution géographique qui fait que les administrateurs de systèmes n'ont généralement pas la possibilité de rencontrer physiquement leurs utilisateurs. Les utilisateurs ne sont donc identifiés que de façon indirecte, à travers une connexion virtuelle.

L'arsenal classique de la sécurité est également peu efficace pour une autre raison : comme on l'a vu, la plupart des attaques sur l'Internet exploitent les vulnérabilités des réseaux ou des logiciels, et contournent ainsi les mécanismes d'authentification et d'autorisation. Il est donc nécessaire d'implémenter les parades spécifiques à chaque type d'attaque, ces parades étant développées à la suite d'attaques réussies de systèmes similaires, ou simplement à la suite de la découverte de nouvelles vulnérabilités. En particulier, les fournisseurs de logiciels sont maintenant fortement

sensibilisés aux problèmes de sécurité, et aux risques que ceux-ci leur font encourir pour leur image de marque. Ils essaient donc de fournir dès que possible des "rustines" (*patches* en anglais) corrigeant les vulnérabilités identifiées. Néanmoins, il n'est pas si facile d'appliquer des rustines : cela requiert du temps et de la compétence de la part des administrateurs, qui sont une ressource rare et chère. D'abord, les annonces de nouvelles vulnérabilités sont fréquentes, et il est parfois délicat d'identifier si un système donné présente ces vulnérabilités et nécessite qu'on applique les rustines. D'autre part, l'application de rustines modifie parfois les fonctionnalités des systèmes, empêchant ainsi le fonctionnement de certaines applications qui peuvent être vitales pour les entreprises. Enfin, comme cela a déjà été mentionné, de nombreux systèmes sont mal administrés, et ne sont que rarement mis à jour. Le risque est alors d'autant plus grand que, dès leur parution, les pirates analysent les rustines pour découvrir les vulnérabilités qu'elles sont censées corriger, et développer des exploits visant ces vulnérabilités sur des systèmes qui n'auraient pas été encore mis à jour. Les délais entre la parution des rustines et de tels exploits se réduisent de plus en plus : en août 2005, un exploit a été diffusé deux jours après le bulletin de sécurité de Microsoft MS05-039 corrigeant la vulnérabilité de Plug-and-Play ! Certains fournisseurs prévoient une mise à jour automatique, grâce à une connexion Internet, mais cette automatisation elle-même n'est pas exempte de risque.

Il vaut mieux, dit-on, prévenir que guérir, et il faudrait donc éviter d'introduire des failles ou des vulnérabilités dans la conception, la réalisation ou l'exploitation des systèmes. Malheureusement, l'objectif "zéro défaut" est encore loin d'être atteint en matière d'informatique. Les systèmes actuels sont trop complexes pour être totalement maîtrisables, et leur sécurité n'est pas toujours un objectif primordial. Ainsi un constructeur pourra considérer plus rentable de mettre rapidement sur le marché un système imparfait plutôt que de lui faire subir les méthodes de développement et de validation nécessaires pour obtenir un bon niveau de sécurité, comme ceux qui sont définis dans les "critères communs" [CC 1999]. D'ailleurs, même les systèmes les plus sûrs ne résistent pas longtemps à des attaques menées par des professionnels compétents. Il faut donc explorer d'autres voies.

4.2 La tolérance aux intrusions

4.2.1 La tolérance aux fautes

La tolérance aux fautes [Laprie et al. 1996] est une technique qui a fait ses preuves pour réaliser des systèmes informatiques capables de fournir un service correct, même en présence de phénomènes accidentels, comme des perturbations de l'environnement (fautes externes), des défaillances de composants matériels (fautes physiques internes), voire des fautes de conception, en particulier logicielles (bogues).
Selon la terminologie de la sûreté de fonctionnement [Laprie et al. 1996], les fautes sont les causes des erreurs, les erreurs sont des parties anormales de l'état du système informatique, et lorsque

les erreurs se propagent jusqu'à l'interface du système, c'est-à-dire lorsque le service fourni par le système est incorrect, il y a défaillance. Lorsque les fautes sont accidentelles, et suffisamment rares, il est possible de les tolérer. Il faut pour cela détecter les erreurs avant qu'elles ne produisent de défaillance et les corriger : c'est le traitement des erreurs. Il faut aussi effectuer un diagnostic, c'est-à-dire identifier la ou les fautes, isoler les composants fautifs, les remplacer ou les réparer, et enfin remettre le système dans sa configuration nominale : tout cela constitue le traitement des fautes.

Pour détecter les erreurs, on peut utiliser deux classes de techniques. La première classe est constituée par des vérifications de propriétés (contrôles de vraisemblance) : il faut observer l'état du système, en particulier certaines valeurs ou certains événements, et vérifier s'ils sont plausibles. Cela nécessite en général peu de matériel ou logiciel supplémentaire (redondance). Parmi les vérifications par matériel, notons que la plupart des microprocesseurs détectent des codes d'instruction inexistants ou interdits, des commandes inexistantes ou interdites, et que des "chiens de garde" permettent de détecter des dépassements de durée de certaines exécutions. Les contrôles de vraisemblance par logiciel consistent à insérer dans des programmes des tests sur les valeurs de certaines variables ou sur les instants de certains événements ou leur séquence. On parle alors de "programmation défensive" [Rabejac 1995]. Un cas particulier des contrôles sur les valeurs est constitué par les codes détecteurs d'erreurs.

La seconde classe des techniques de détection d'erreur consiste à comparer plusieurs exécutions, soit successives sur le même matériel, soit sur des matériels différents. Cette deuxième classe nécessite donc plus de redondance (on parle de "redondance massive"). Mais elle suppose aussi qu'une faute unique n'ait pas les mêmes effets, c'est-à-dire ne produise pas les mêmes erreurs, sur les différentes exécutions. S'il s'agit de fautes matérielles internes, on peut exécuter les mêmes traitements sur des matériels identiques, puisqu'il est peu probable que les deux exemplaires de matériels subissent chacun une faute interne identique en même temps. En revanche, s'il s'agit de fautes de conception, les mêmes traitements sur des matériels identiques produiront les mêmes erreurs, et la comparaison ne détectera rien. Dans ce cas, il faut "diversifier" la conception des matériels (si on suppose qu'il peut y avoir des fautes de conception dans le matériel), mais surtout des logiciels, où les fautes de conception sont fréquentes [Deswarte et al. 1999]. Dans ce cas, les différents exemplaires doivent être conçus et développés indépendamment, de façon à réduire le risque qu'une faute identique ne produise les mêmes erreurs dans les divers exemplaires.

Pour corriger les erreurs, on peut tenter de remettre le système dans un état qu'il a eu avant qu'il y ait des erreurs, c'est-à-dire effectuer une reprise. Pour cela, il faut avoir construit des points de reprise, c'est-à-dire des sauvegardes de l'état du système. Une autre technique, la poursuite, consiste à remplacer l'état erroné du système par un état sain artificiel, et à poursuivre le traitement. Ceci est possible, par exemple, dans certains systèmes de commande-contrôle, où on peut réinitialiser le système et réacquérir les paramètres de tous les capteurs avant de poursuivre le traitement. Enfin, il est possible aussi de "masquer" les erreurs, lorsqu'on a suffisamment de redondance pour reconstituer un état correct à partir de l'état erroné, par exemple par vote majoritaire sur trois exécutions (ou plus).

Dans la plupart des cas, l'efficacité des techniques de tolérance aux fautes repose sur le fait que les fautes sont des phénomènes rares et distribués aléatoirement dans le temps. Ainsi, par exemple, lorsqu'on utilise une redondance massive, on suppose qu'il est peu probable qu'un exemplaire défaille pendant qu'on en répare un autre. Cette hypothèse n'est malheureusement pas valable lorsqu'on considère les intrusions : si un attaquant réussit à s'introduire dans un système, il essaiera de poursuivre son attaque sur le même système, mais aussi sur d'autres systèmes similaires.

4.2.2 Vulnérabilités, attaques, intrusions

Une intrusion résulte de l'exploitation d'une vulnérabilité (faute de conception ou d'exploitation, dans la terminologie de la sûreté de fonctionnement) par une attaque (faute externe) [Powell et Stroud 2003]. L'intrusion peut être considérée comme une faute interne, qui peut produire des erreurs pouvant provoquer une défaillance vis-à-vis de la sécurité, c'est-à-dire une violation de la politique de sécurité du système. On a vu précédemment qu'il était illusoire d'éviter les attaques sur l'Internet. De même il est impossible d'éliminer toutes les vulnérabilités. Par exemple, on peut considérer que de connecter un système sur l'Internet est en soit une vulnérabilité, mais à quoi servirait un serveur Web s'il n'était pas connecté sur Internet ?

Il serait donc intéressant de tolérer les intrusions, c'est-à-dire de faire en sorte que l'intrusion dans une partie du système n'ait pas de conséquence sur sa sécurité. Pour cela, on peut utiliser les techniques développées dans le cadre plus général de la tolérance aux fautes. Mais cela pose deux problèmes principaux :

- Si une attaque a réussi (au moins partiellement) sur une partie du système, il ne doit pas être trop facile de réussir la même attaque sur une autre partie. Pour cela, il faut que chaque partie soit suffisamment sécurisée et, de préférence, que les parties soient diversifiées.
- Il ne faut pas qu'une seule intrusion dans une partie du système fournisse à l'attaquant des informations sensibles (confidentialité). Ceci est d'autant plus important que la redondance, nécessaire à la tolérance aux fautes, peut fournir plus d'occasions d'attaques aux pirates éventuels.

Si on est capable de résoudre ces problèmes, on peut appliquer les techniques de la tolérance aux fautes : traitement des erreurs (détection et correction) et traitement des fautes (diagnostic, isolation, réparation, reconfiguration). Notons cependant que pour ce qui concerne la détection des erreurs dues à des intrusions, des techniques de détection spécifiques ont été développées, improprement appelées "de détection d'intrusions", alors qu'elles ne détectent pas directement les intrusions, mais seulement leurs effets, c'est-à-dire les erreurs dues à des intrusions (ou mêmes seulement à des attaques, sans qu'il n'y ait intrusion).

Pour corriger les dégâts causés par l'intrusion, on peut, comme pour la tolérance aux fautes classique, effectuer une reprise (si on dispose de sauvegardes à jour), ou une poursuite (si on peut construire un état sain artificiel), mais il est souvent plus facile et plus efficace de masquer

les erreurs, en utilisant une redondance massive. C'est donc cette approche qui a été choisie pour la plupart des architectures de systèmes tolérant les intrusions.

4.3 Un exemple d'architecture tolérant les intrusions : le projet DIT

L'architecture DIT (Dependable Intrusion Tolerance) [Valdes et al. 2002] a été développée par une coopération entre SRI International et le LAAS-CNRS, dans le cadre du programme OASIS (*Organically Assured and Survivable Information Systems*) de la DARPA (*US Defense Advanced Research Projects Agency*). L'objectif est de réaliser des serveurs Web capables de continuer à fournir un service correct en présence d'attaques. Pour ce type d'application, la confidentialité n'est pas essentielle ; en revanche l'intégrité et la disponibilité doivent être assurées, même en cas d'attaque par des services compétents. Il est donc primordial qu'une attaque réussie sur un des éléments du système ne facilite pas l'attaque d'autres éléments. C'est donc un souci de diversification qui a présidé à la conception de l'architecture. D'autres travaux sur la tolérance aux intrusions ont d'ailleurs également adopté ce type d'architecture [Majorczyk et al. 2005]. Nous présentons dans cette section les principaux composants de cette architecture sans entrer dans les détails. Des informations plus détaillées sont fournies dans [Saidane 2005].

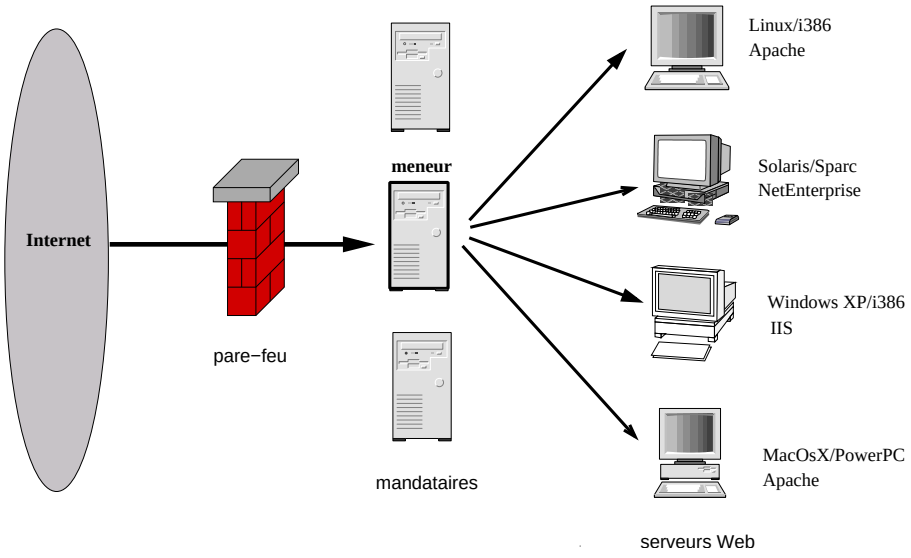


FIG. 4.1 – Architecture DIT

L'architecture se compose d'un banc de serveurs Web ordinaires, mais aussi diversifiés que possible en ce qui concerne la plateforme matérielle (processeurs SPARC, PENTIUM, PowerPC, etc.), les systèmes d'exploitation (Solaris, Microsoft Windows, Linux, MacOS, etc.) et les logiciels d'application Web (Apache, IIS, Enterprise Server, Openview Server, etc.) (voir la figure

4.1). Seul le contenu des pages Web est identique sur chaque serveur. Les serveurs d'application sont en nombre suffisant pour assurer un temps de réponse satisfaisant sur un taux de requêtes nominal dans un régime de redondance donné (cf. 4.3.4). Les serveurs ne sont pas directement connectés sur Internet, mais en sont isolés par un pare-feu et par un ensemble de *mandataires* qui servent de relais pour les requêtes émises par des clients sur Internet et les réponses générées par les serveurs Web. Pour tolérer les intrusions et les défaillances éventuelles, les mandataires sont eux-mêmes diversifiés pour ce qui concerne le matériel, mais animés par un logiciel développé spécifiquement. C'est le même programme qui s'exécute sur chaque mandataire, mais dans des conditions différentes, chaque mandataire jouant un rôle distinct. Les exécutions sont donc elles aussi diversifiées. La section suivante décrit le fonctionnement des mandataires.

4.3.1 Les mandataires

À un instant donné, un seul mandataire reçoit les requêtes des clients. Ce mandataire est appelé *meneur* et il joue le rôle serveur du point de vue du client. Il est ainsi chargé de transmettre chaque requête à un certain nombre de serveurs d'application, ce nombre dépendant du régime de redondance courant (cf. 4.3.4), de récupérer les réponses des serveurs et, après avoir vérifié leur cohérence, d'envoyer une réponse au client. Les autres mandataires sont appelés *auxiliaires*.

Tous les mandataires (meneur et auxiliaires) jouent un rôle essentiel dans la politique de tolérance aux intrusions. Ils sont chargés de surveiller l'état des serveurs d'application ainsi que des autres mandataires et de prendre les décisions en commun dans le cas où une intrusion est détectée. Pour détecter une intrusion, ils se basent sur un ensemble de mécanismes de détection présentés dans le paragraphe suivant (4.3.2). Ils analysent les rapports de ces différents mécanismes et adaptent en conséquence la politique de tolérance du système (cf. 4.3.3). Comme nous le verrons dans les sections suivantes, la modification de la politique de tolérance peut aller de l'augmentation de la sévérité des différents contrôles effectués par les mécanismes de détection jusqu'au redémarrage de certains éléments qui sont considérés comme corrompus.

4.3.2 Les mécanismes de détection

Cette approche de la tolérance aux intrusions ne peut être efficace que si les mécanismes de détection inclus dans l'architecture sont eux-mêmes suffisamment efficaces. Il a donc été choisi d'utiliser différents mécanismes de façon à ce que les points forts des uns recouvrent les faiblesses des autres. Cette section est dédiée à la présentation de l'ensemble de ces mécanismes.

4.3.2.1 Le protocole d'accord

Ce mécanisme, sans doute le plus efficace de ceux déployés dans cette architecture, est utilisé dans deux cas :

– pour valider les réponses des serveurs d'application dans les régimes autres que simplex (cf. 4.3.4). Les différents serveurs d'application qui traitent la même requête élaborent chacun une réponse et calculent une somme de contrôle cryptographique (MD5) sur cette réponse. Toutes ces sommes sont donc envoyées au meneur. Ce dernier effectue un protocole d'accord (par exemple, comparaison pour deux réponses, vote majoritaire pour trois, etc.). Si les réponses ne sont pas identiques, au moins l'un des serveurs est défaillant et une alerte est remontée aux autres mandataires.
– pour que les mandataires puissent prendre en commun des décisions lors de remontées d'alertes. En effet, lors de remontées d'alertes provenant des capteurs locaux, les opinions des différents mandataires concernant la corruption d'un composant dans le système peuvent ne pas être identiques. Dans ce cas, le protocole d'accord permet de faire en sorte que tous prennent la même décision concernant ce composant.

4.3.2.2 Les outils de détection d'intrusion

L'architecture DIT inclut deux outils de détection d'intrusions :

– Snort [Snort], logiciel libre, est un outil de détection d' attaques connues (*misuse detection* en anglais) basé sur un mécanisme efficace de signatures. Cette base très riche et mise à jour en permanence permet ainsi d'identifier un grand nombre de paquets réseaux suspects.
– Emerald [Valdes et Skinner 2000, Neumann et Porras 1999], est un outil développé à SRI International. Il effectue à la fois de la détection d'attaques connues et de la détection d'anomalies[1] (*anomaly detection* en anglais). Il est également capable d'effectuer des corrélations à partir de plusieurs capteurs d'information.

Ces deux outils ont été choisis pour leur complémentarité (Snort effectue de la détection d'anomalies et est plus efficace au niveau du réseau tandis qu'Emerald effectue de la détection d'attaques et est plus efficace au niveau du calculateur) et leur efficacité.

4.3.2.3 Le test d'intégrité des fichiers

Les outils de détection d'intrusion présentés ci-dessus peuvent être relativement inefficaces si on imagine une nouvelle attaque "lente". On peut ainsi imaginer un attaquant qui pourrait modifier tour à tour sur chaque serveur d'application un fichier auquel on accède très rarement. Si cette attaque est nouvelle, il est possible que Snort ne puisse identifier aucun paquet suspect et qu'Emerald ne détecte aucune attaque sur les serveurs. Le protocole d'accord de son côté ne peut détecter un problème que si le fichier en question est lu par le traitement d'une requête alors que l'attaque n'a encore modifié ce fichier que sur une partie des serveurs exécutant la requête. Mais si on imagine que ce fichier est rarement lu, il est possible que l'attaquant ait eu

[1]Qui correspond à la reconnaissance d'une déviation de comportement et non pas à la reconnaissance d'une signature représentative d'une attaque connue.

le temps de faire les mêmes modifications sur tous les serveurs, auquel cas le protocole d'accord ne détecte rien.

Aussi, un mécanisme consistant à faire des tests d'intégrité de fichiers à été ajouté dans l'architecture. Il vise à détecter des modifications non-autorisées apportées à ces fichiers. Il serait possible d'utiliser des outils existants tels que Tripwire [Kim et Spafford 1993] pour réaliser ces tests, mais ces outils sont faits pour être utilisés localement, dans un état sûr. Or, si une des machines de l'architecture est compromise, on peut imaginer qu'un logiciel malveillant puisse changer l'exécution du test d'intégrité pour l'empêcher de détecter une modification.

Il est donc nécessaire dans l'architecture DIT de réaliser ces tests à distance. Mais il est évident que télécharger les fichiers à partir des serveurs distants pour les vérifier n'est pas une solution envisageable en raison des coûts de communication et de stockage. Une solution originale a été développée, basée sur l'utilisation d'un défi [Deswarte et al. 2003].

Ce protocole est qualifié de protocole à défi-réponse ou PDR. Périodiquement, un défi D et un nom de fichier f est envoyé à chaque calculateur par le *vérifieur* (cf. fin de ce paragraphe). Le calculateur concatène le défi et le contenu du fichier en question et calcule S, une somme de contrôle cryptographique, à l'aide d'une fonction de hachage H sur ces données concaténées : $S = H(D|\ f)$. Cette réponse est ensuite retournée au vérifieur (cf. figure 4.2).

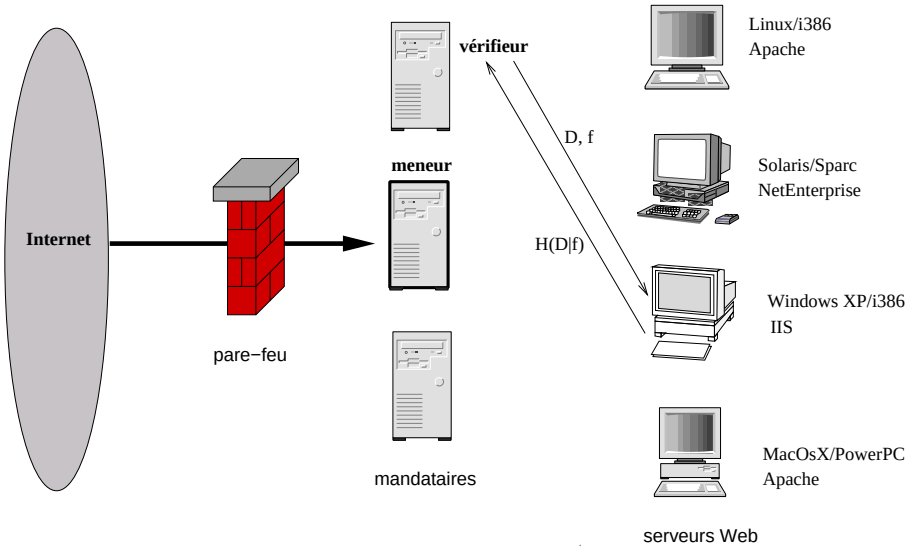


FIG. 4.2 – Protocole à défi-réponse

Ce dernier compare cette somme S avec la somme pré-calculée et stockée localement. Le défi a été ajouté de façon à garantir la "fraîcheur" de la réponse. En effet, si on calcule le résultat de la fonction H sur le seul contenu du fichier, il est possible à un intrus de précalculer toutes les réponses pour tous les fichiers, de les stocker et de les renvoyer tout en ayant modifié les fichiers

correspondants. Le défi permet donc de s'assurer que le calcul de la fonction H soit bien fait au moment de la demande. Il faut bien sûr assurer que le défi soit impossible à deviner ou d'observer pour l'attaquant et donc différent à chaque requête. Pour éviter cependant de stocker une trop grande quantité de réponses pré-calculées (une par défi), il suffit de choisir un nombre de défis tel que la fréquence de renouvellement des défis soit supérieure à la fréquence de redémarrage du serveur (utile pour le "rajeunissement du logiciel"). Cette fréquence de redémarrage peut par exemple être estimée à une fois par jour, ce qui donne donc un nombre de défis raisonnable.

On peut cependant imaginer que l'attaquant, pour contourner ce mécanisme, fasse des copies de tous les fichiers qu'il compte modifier. Dans ce cas, il peut calculer la somme de contrôle sur la copie du fichier à chaque fois. Mais ce comportement sera forcément détecté par d'autres mécanismes mis en place dans l'architecture : l'apparition de nouveaux fichiers dans le système sera détectée notamment par des outils de détection d'intrusion ou par vérification de l'intégrité des répertoires ; on peut également imposer des limitations strictes des ressources disque sur le serveur.

Les fichiers dont l'intégrité est ainsi testée dans le système sont bien sûr les fichiers gérés par les serveurs d'application (les pages *html* pour un serveur Web par exemple) mais aussi un certain nombre de fichiers du système, aussi bien sur les mandataires que sur les serveurs d'application. Le logiciel qui permet de faire ces tests est appelé *vérifieur* et est installé sur tous les mandataires. Chacun des mandataires effectue donc périodiquement des requêtes sur les serveurs et les autres mandataires.

4.3.2.4 La vérification en ligne

Le mécanisme de vérification en ligne teste le comportement du système pendant son exécution et permet de vérifier qu'un certain nombre de propriétés sont satisfaites (des propriétés de sécurité dans le cas de DIT). Toute violation de ces propriétés est ainsi détectée et indique probablement l'intrusion de code malicieux dans le système.

Dans le cas de DIT, les propriétés sont décrites par un modèle formel du comportement exprimé sous forme d'un automate. Un moniteur est chargé d'observer en temps réel les changements d'état du système et de vérifier s'ils sont compatibles avec le modèle. Dans le cas contraire, ce moniteur va déclencher une alarme.

Dans l'architecture DIT, le comportement de tous les mandataires est ainsi analysé en utilisant cette technique. Le code des mandataires a été instrumenté de façon à ce qu'on puisse suivre leur évolution pas-à-pas. On peut ainsi détecter tout comportement anormal. Ceci est particulièrement efficace contre l'intrusion de code malicieux comme les débordements de tampons par exemple.

4.3.3 Le gestionnaire d'alertes

Chaque mandataire a un rôle particulier : il peut être meneur, adjudicateur (cf. 4.3.5) ou auxiliaire. Aussi l'architecture des mandataires est composée de modules spécifiques pour les rôles de meneur et d'adjudicateur, et du gestionnaire d'alertes, qui est un module commun à tous les rôles. La figure 4.3 présente cette architecture.

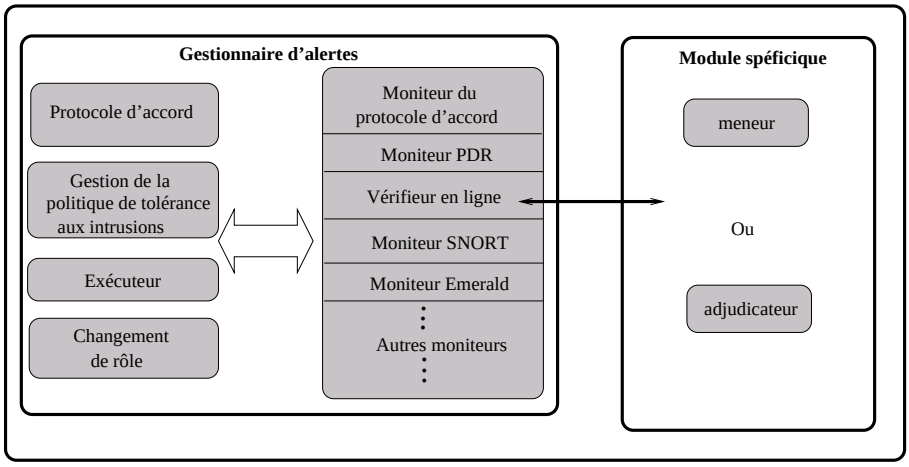


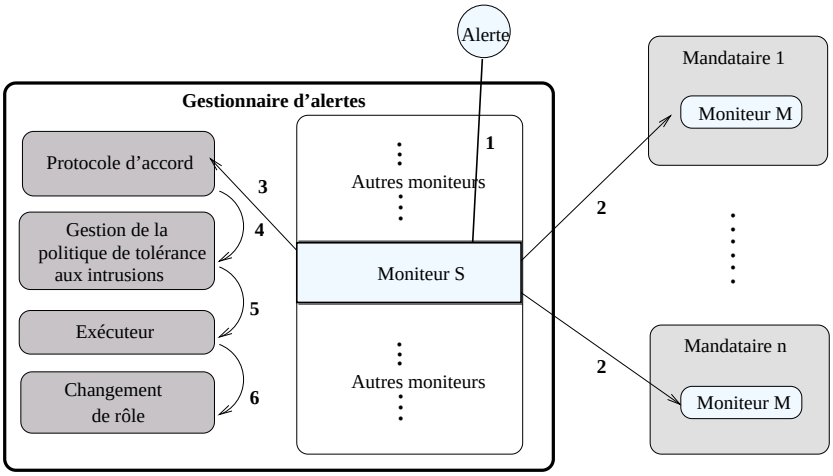
FIG. 4.3 – Architecture d'un mandataire

Le gestionnaire d'alertes inclut un moniteur pour chaque mécanisme de détection présent dans l'architecture et différents composants (un composant de protocole d'accord, un gestionnaire de la politique de tolérance aux intrusions, un composant d'exécution et un composant de changement de rôle). Le rôle principal du gestionnaire d'alertes est de traiter les alertes produites par les différents mécanismes de détection, en fonction de leur crédibilité.

Ainsi que le représente la figure 4.4, chaque moniteur attend des messages provenant du mécanisme correspondant. Par exemple, le moniteur Snort attend des alertes venant des capteurs Snort. Quand le moniteur reçoit une alerte (étape **1** de la figure 4.4), il vérifie l'état courant du processeur suspecté. Si ce processeur est déjà considéré localement comme corrompu, l'alerte est ignorée. S'il est considéré pour le moment comme correct, l'alerte est envoyée à tous les autres moniteurs similaires sur les autres mandataires (étape **2**) et une demande de protocole d'accord est envoyée au composant de protocole d'accord de façon à ce qu'un vote entre tous les mandataires soit exécuté sur l'état de ce processeur (étape **3**).

Le composant de protocole d'accord est chargé de gérer les votes entre les mandataires de façon à ce qu'une décision commune puisse être prise. Si une majorité de mandataires considère qu'un même processeur est corrompu, alors une requête est envoyée au composant de gestion de la politique de tolérance aux intrusions de façon à ce que ce dernier génère une liste de contre-mesures à appliquer dans le système (étape **4**). Cette liste est générée sur le mandataire qui a

initié le vote et elle est envoyée à tous les autres mandataires. Le composant d'exécution est quant à lui responsable de mettre en œuvre les contre-mesures. Par exemple, si le processeur corrompu est le meneur, il est nécessaire d'élire un autre mandataire pour jouer le rôle de meneur. Le composant d'exécution contacte alors le composant de changement de rôle qui va provoquer la reconfiguration du nouveau meneur.



Contenu des messages :
1 : IP_Corrompue
2 : Alerte#IP_Corrompue
3 : Alerte#IP_Corrompue
4 : IP_Corrompue (si une majorité est d'accord)
5 : action#action#action ...
6 : nouveau rôle (si meneur ou adjdicateur corrompu)

FIG. 4.4 – Étapes d'une gestion d'alerte

4.3.4 La redondance adaptative : les régimes de fonctionnement

Une redondance massive est déployée dans l'architecture DIT dans le but d'améliorer la disponibilité et l'intégrité du système. Une autre caractéristique importante de cette architecture est le souci de la performance. Pour faire en sorte que ce système soit le plus performant possible, même en présence d'attaques, l'architecture applique une redondance adaptative selon différents *régimes* de fonctionnement : si on dispose de `N` serveurs d'application et que le régime est d'ordre `n`, cela signifie que `n` parmi `N` serveurs d'application seulement vont traiter la requête. Ces `n` serveurs peuvent bien sûr être différents à chaque nouvelle requête. Le régime est choisi par les mandataires en fonction du niveau d'alerte dans le système. Lorsque le système démarre par exemple, on peut considérer qu'il est dans un état fiable et qu'on peut utiliser un seul serveur d'application (régime d'ordre 1 ou régime *simplex*). Dés lors que des alertes sont levées dans le système, ou que, par exemple, des avis de sécurité annoncent la propagation d'un nouveau type

d'attaque, on peut augmenter le régime (envoi de chaque requête à deux ou trois serveurs pour les régimes *duplex* ou *triplex*) pour améliorer la robustesse du système. De la même façon, si le niveau d'alertes diminue dans le système, les mandataires vont décider de baisser le régime. Bien sûr, plus le régime est élevé, plus les performances du système global vont être dégradées, mais dans la mesure où ce régime est constamment adaptable, on peut garantir une dégradation minimale des performances.

4.3.5 L'adaptation de l'architecture à des serveurs à contenu dynamique

L'architecture, telle que nous l'avons décrite jusqu'à présent est bien adaptée à des serveurs Internet à contenu statique. Si on considère par exemple le cas d'un serveur Web, cela signifie que les pages *html* qu'il gère ne peuvent pas être modifiées de façon dynamique par les requêtes des clients. En effet, il est difficile d'effectuer ces modifications de façon cohérente sur tous les serveurs d'application (qui gèrent tous exactement les mêmes données), puisque tous les serveurs n'exécutent pas les mêmes requêtes. Si une requête modifie des données, cette modification devrait être reportée sur les autres serveurs, ce qui poserait des problèmes d'atomicité et de cohérence. Aussi, avec une telle architecture, il est préférable d'arrêter le système tous les jours, de faire des modifications de certaines pages identiquement sur tous les serveurs, puis redémarrer les serveurs. Cette architecture est typiquement bien adaptée à des serveurs fournissant des informations "rarement" mises à jour comme par exemple un serveur mettant à disposition des alertes de sécurité.

Si on considère d'autres types de serveurs Internet, comme par exemple, le serveur Web d'une agence de voyage, pour lequel des données doivent être en permanence être mises à jour (réservations, modifications, annulations), il faut trouver une autre solution. Les serveurs d'application doivent pouvoir accéder à une base de données qu'ils puissent mettre à jour en temps réel. Si on décide d'utiliser une base de données par serveur d'application, les problèmes d'atomicité et de cohérence restent entiers.

La solution adoptée par DIT consiste à utiliser un seul serveur de base de données dans l'architecture. Ce serveur de base de données est un serveur sur étagère tolérant les fautes accidentelles. On suppose donc que tout accès à la base de données s'effectue toujours de façon correcte, même en cas de défaillance d'un de ses composants. Il faut dès lors contrôler l'accès des serveurs d'application à ce serveur de base de données. Deux composants ont ainsi été ajoutés à l'architecture : *l'adjudicateur* et le *médiateur* (cf. figure 4.5).

L'adjudicateur est l'un des mandataires. Il est chargé de contrôler les accès à la base de données : tout serveur d'application souhaitant effectuer une requête sur cette base doit passer par l'adjudicateur[2]. Il coordonne donc, vérifie et filtre les requêtes SQL destinées à la base. Comme chaque requête SQL est forcément liée à une requête *http* dans le système, l'adjudicateur doit

[2]Dans la suite, nous parlerons de requêtes SQL par commodité pour désigner les requêtes à la base de données, par opposition aux requêtes *http* exécutées par les serveurs Web.

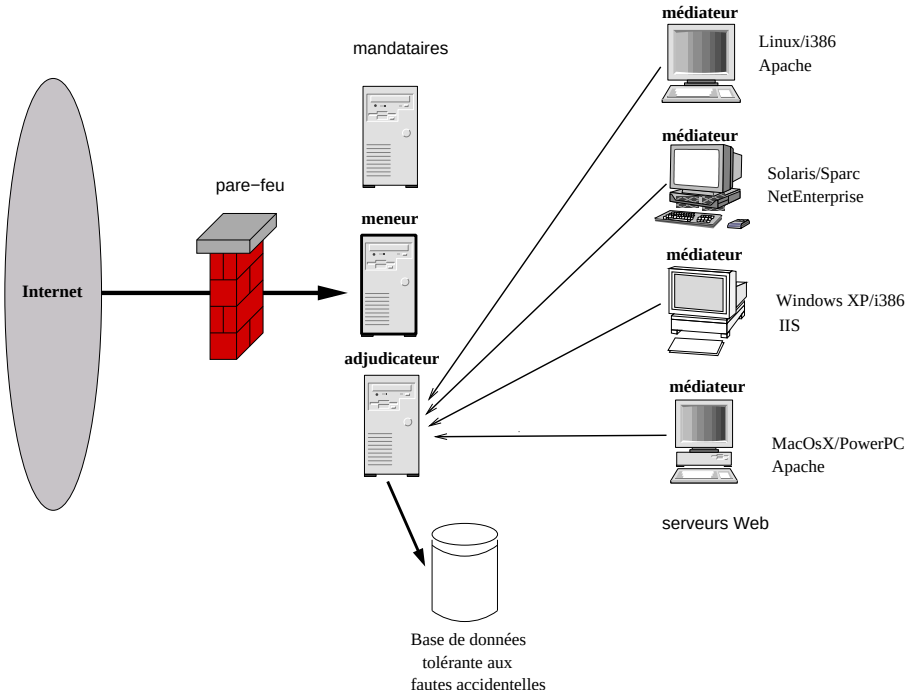


FIG. 4.5 – L'adjudicateur

donc vérifier que tous les serveurs d'application en charge de la même requête *http* `R` envoie les mêmes requêtes SQL et dans le même ordre à la base de données. Il effectue pour cela un protocole d'accord sur l'ensemble des requêtes SQL qu'il reçoit des différents serveurs. Il apparaît ici que pour établir ces vérifications, l'adjudicateur doit connaître à tout instant les requêtes *http* qui sont en cours d'exécution dans le système. Pour cela, le meneur est constamment en relation avec l'adjudicateur et l'informe de toute nouvelle requête client traitée. En fait, chacune de ces requêtes est associée à un identifiant (*IDR*) qui est transmis par le meneur à l'adjudicateur. Les requêtes SQL correspondant au même IDR sont donc comparées par l'adjudicateur et si elles sont cohérentes (utilisation d'un protocole d'accord sur l'ensemble de ces requêtes), l'accès à la base est effectué.

Le médiateur est un composant ajouté sur chaque serveur d'application. Ce composant est un intermédiaire entre le meneur et le serveur d'application et entre le serveur d'application et la base de données. Ainsi, chaque requête client envoyée par le meneur est interceptée par le médiateur qui stocke son IDR et fait suivre ensuite la requête au serveur d'application. En même temps, le médiateur joue localement le rôle du serveur de base de données. Ainsi, lorsque le serveur d'application nécessite un accès à la base de données, la requête SQL correspondante est reçue localement par le médiateur. Celui-ci effectue certaines vérifications puis envoie cette requête à l'adjudicateur.

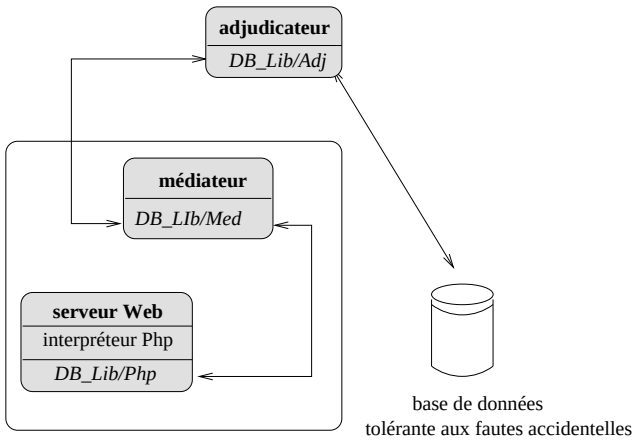


FIG. 4.6 – La librairie d'accès à la base de données

Pour que ce déroutement puisse être effectué, la librairie SQL standard incluse dans le serveur Web a été modifiée (ainsi que le montre la figure 4.6). Elle a été réécrite de façon à pouvoir rediriger les requêtes SQL vers l'adjudicateur par l'intermédiaire du médiateur. De façon à ce que cette librairie puisse être facilement appelée depuis les différents langages utilisés par les serveurs Web (PHP par exemple), cette librairie a été développée en langage C. Des outils tels que Swig [Swig], par exemple, peuvent être utilisés pour générer automatiquement l'interface d'accès à cette librairie dans différents langages.

4.3.6 Mesures de performance

L'architecture que nous avons présentée a été implémentée sous la forme d'un prototype de serveur Web pour une agence de voyage. Pour assurer la diversification matérielle, des processeurs à base de Pentium, de PowerPC et de Sparc ont été utilisées. De même, les systèmes d'exploitation Linux, MacOSX et Solaris ont été utilisés. Dans le prototype, trois mandataires et trois serveurs Web sont utilisés. Le serveur de base de données est un serveur MySQL.

Des mesures de performance ont été réalisées sur ce prototype afin de mesurer le coût des mécanismes de sécurité inclus dans cette architecture. Nous présentons ici quelques résultats (cf. [Saidane 2005] pour des résultats détaillés).

– Nous avons pu constater que le temps global de traitement d'une requête HTTP est plus sensible à la taille du fichier *html* qu'au régime de fonctionnement. Le temps utilisé par la fonction de hachage (pour générer les sommes de contrôles des réponses) et les vérifications effectuées par le meneur sont finalement moins coûteux que le temps utilisé par le serveur Web pour traiter la requête et renvoyer la réponse complète au client, et ce d'autant plus que le fichier demandé est gros. Si on prend l'exemple d'un fichier de 44Ko, le passage d'une architecture conventionnelle à l'architecture DIT en mode simplex représente un surcoût de

traitement de 26%. Ce résultat est acceptable compte tenu des mécanismes de tolérance aux intrusions inclus dans l'architecture. Le passage d'un régime donné à un régime plus sévère n'est quant à lui pas très coûteux et de moins en moins coûteux au fur et à mesure qu'on augmente le régime (le passage duplex → triplex est moins coûteux que le passage simplex → duplex).

- Le temps global de traitement d'une requête HTTP avec accès à la base de données dans notre architecture est environ 2 fois le temps global de traitement avec un accès direct à base de données SQL standard (sans utilisation de médiateur et adjudicateur, ni protocole d'accord). Compte tenu des mécanismes de sécurité mis en place, cette performance nous semble acceptable, d'autant plus que dans l'état actuel du prototype, l'accent n'a pas été mis sur l'optimisation du code, en particulier en ce qui concerne le développement de la librairie de gestion de base de données. Il semble donc raisonnablement possible d'améliorer encore ces performances.
- Nous avons mesuré la durée globale de réponse à une alerte (DGRA) qui correspond au temps écoulé entre la réception de l'alerte PDR et la notification de redémarrage du serveur corrompu. Ce DGRA est bien sûr élevé (en moyenne 73.2 sec) mais ceci est normal puisqu'il inclut le temps de redémarrage du serveur qui est de l'ordre de 68 sec. Nous avons également noté qu'il y a une grande différence entre la durée minimum et maximum du protocole d'accord (de 6ms à 1s). Cette différence peut s'expliquer par la latence de propagation de l'alerte entre les mandataires. En fait, un mandataire qui reçoit une requête de vote concernant la corruption d'un serveur qu'il juge localement correct n'envoie pas immédiatement une réponse négative mais attend un certain temps pour estimer à nouveau l'état de ce serveur avant de répondre : si une corruption est détectée par un mandataire, il est probable qu'elle sera détectée par les autres un peu après.

4.4 Conclusion

Compte tenu de la fréquence actuelle des attaques sur l'Internet et du grand nombre de vulnérabilités des systèmes informatiques classiques, la tolérance aux intrusions apparaît comme une technique réaliste pour améliorer la sécurité des serveurs sur Internet, en particulier en diversifiant les plateformes matérielles et logicielles. L'architecture DIT a montré la faisabilité de cette technique avec les technologies actuelles, avec des performances satisfaisantes. Cela a bien sûr un coût, puisqu'il est plus onéreux d'exploiter des systèmes informatiques hétérogènes. Mais c'est sans doute le prix à payer pour plus de sécurité dans un monde ouvert, donc incertain.

Bibliographie

[CC 1999] *Common Criteria for Information Technology Security Evaluation*, norme ISO/CEI 15408, version 2.1, août 1999.

[Laprie et al. 1996] J.-C. Laprie et al., *Guide de la sûreté de fonctionnement*, Cépaduès Éditions, 1995-1996, 368 p, ISBN 9782854283822.

[Rabejac 1995] C. Rabéjac, "Auto-surveillance logicielle pour applications critiques : méthode et mécanismes", *Thèse de Doctorat de l'Institut National Polytechnique de Toulouse*, n^o1095, novembre 1995, Rapport LAAS n^o95449.

[Deswarte et al. 1999] Y. Deswarte, K. Kanoun, J.-C. Laprie, "Diversity Against Accidental and Deliberate Faults", in *Computer Security, Dependability and Assurance : From needs to Solutions*, P. Amman, B.H. Barnes, S. Jajodia, E.H. Sibley Eds., IEEE Computer Society Press, 1999, pp. 171-181.

[Powell et Stroud 2003] D. Powell, R. Stroud (Eds.), "Malicious-and Accidental-Fault Tolerance for Internet Applications : Conceptual Model and Architecture", Final Version, Rapport LAAS n^o03011, *Projet IST-1999-11583 MAFTIA, Deliverable D21*, janvier 2003, 123 pp., disponible à (http ://www.research.ec.org/maftia/deliverables/).

[Valdes et al. 2002] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saïdi, V. Stavridou, T. Uribe, "An Adaptative Intrusion-Tolerant Server Architecture", *10th International Workshop on Security Protocols*, Cambridge (UK), avril 2002, LNCS n^o 2845, Springer h, 2003, pp. 158-178.

[Majorczyk et al. 2005] Frédéric Majorczyk, Eric Totel, and Ludovic Mé "COTS Diversity Based Intrusion Detection and Application to Web Servers", *Proceedings of the 8th International Symposium on the Recent Advances in Intrusion Detection (RAID)*, Springer Verlag, LNCS 3858, pp. 43-62, September 2005.

[Saidane 2005] A. Saidane, "Conception et réalisation d'une architecture tolérant les intrusions pour des serveurs Internet", *Thèse de Doctorat de l'Institut National des Sciences Appliquées de Toulouse*, janvier 2005, Rapport LAAS n^{o}05147.

[Snort] http ://www.snort.org

[Valdes et Skinner 2000] A. Valdes, K. Skinner, "Adaptive Model-based Monitoring for Cyber Attack Detection", Lecture Notes in Computer Science, Number 1907, *Recent Advances in Intrusion Detection (RAID 2000)*, Springer-Verlag, Toulouse, 2000, pp. 80-92.

[Neumann et Porras 1999] P. Neumann and P. Porras, "Experience with EMERALD to Date", *First USENIX Workshop on Intrusion Detection and Network Monitoring*, 1999.

[Kim et Spafford 1993] G.H. Kim, E.H. Spafford, "The design and Implementation of Tripwire : a File System Integrity Checker", *Technical Report CSD-TR-93-071*, Computer Science Dept, Purdue University, 1993.

[Deswarte et al. 2003] Y. Deswarte, J-J. Quisquater, A. Saidane, "Remote Integrity Checking", *6th IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Control in Information Systems (IICIS'2003)*, Lausanne, Suisse, 2003, pp. 1-11.

[Swig] http ://www.swig.org

Chapitre 5

Travaux de recherche : observation et caractérisation d'activités malveillantes sur Internet

5.1 Problématique

Durant la dernière décennie, les activités malveillantes ont proliféré sur le réseau Internet et ont augmenté de façon significative en volume et en diversité. Ces activités concernent différents types d'attaques incluant des vers, virus, dénis de service, tentatives d'hameçonnage (*phishing*), botnets, etc. [Jacobsson et Ramzan 2008]. Analyser comment les attaquants procèdent pour exploiter les vulnérabilités des systèmes peut fournir des informations intéressantes pour les concepteurs de systèmes de sécurité de façon à imaginer des mécanismes de protection efficaces qui prennent en compte de réelles menaces observées dans un contexte opérationnel. Il existe donc un réel besoin de méthodes pour collecter et enregistrer des données réelles d'attaques et un réel besoin de résultats expérimentaux basés sur l'analyse de ces données.

Un certain nombre d'initiatives et de mécanismes de collecte de données visant cet objectif sont déjà en cours depuis un certain nombre d'années. On peut citer, par exemple :

- les approches basées sur la centralisation et la fusion de fichiers de trace collectés depuis différentes sources (systèmes de détection d'intrusions, pare-feux, etc.)[1],
- les captures de trafic ciblant des adresses IP non attribuées à l'aide de différents mécanismes proposés par exemple dans [TeamCymru 2009, Moore 2002, Song et al. 2002, Bailey et al. 2005, Pang et al. 2004].
- le déploiement de *pots de miel*, c'est-à-dire de ressources dédiées à être attaquées et compro-

[1] http ://www.dshield.org

mises, qui peuvent être enregistrées pour analyser le comportement des attaquants [Spitzner 2002].

Ces techniques sont complémentaires et peuvent donc être utilisées ensemble.

Les pots de miel étant des ressources qui ne sont pas censées être utilisées, tout le trafic observé sur les machines correspondantes est a priori suspect et a de grandes chances de correspondre à des activités malveillantes. Deux types de pots de miel sont souvent présentés en fonction du niveau d'interactivité qu'ils peuvent offrir aux attaquants : les pots de miel basse et haute interaction [Spitzner 2002]. Les pots de miel basse interaction n'implémentent pas de réels services fonctionnels. Ils proposent des émulations de ces services, de piles réseaux ou de parties de systèmes d'exploitation au travers d'interfaces qui n'offrent pas aux attaquants la possibilité de réaliser de réelles opérations sur le système. A titre d'exemple, on peut citer `Tiny honeypot` [Bakos 2002], `honeyd` [Provos 2004] et `Nepenthes` [Baecher et al. 2006]. Ainsi, grâce à leur facilité d'installation et d'administration, des infrastructures de collecte de données utilisant des pots de miel basse interaction ont été déployées à une grande échelle pour fournir une vision globale des activités malveillantes observées à différents endroits de la planète. Nous pouvons mentionner par exemple le projet `leurre.com` [Pouget 2006] qui a déployé depuis 2004 plus de 50 pots de miel basés sur `honeyd` sur les cinq continents.

Les pots de miel basse interaction sont utiles pour fournir des statistiques et des informations de haut niveau concernant les menaces sur Internet et leurs évolutions. Cependant, ils ne sont pas adaptés pour observer l'activité des attaquants qui prennent le contrôle d'une machine cible et essaient de progresser dans leur processus d'intrusion pour obtenir de plus en plus de privilèges. Un tel objectif nécessite l'utilisation de pots de miel haute interaction qui offrent de réels services aux attaquants, ce qui en contre-partie rend ces pots de miel plus risqués. Un pot de miel haute interaction peut correspondre à une machine physique usuelle ou bien consister en une machine virtuelle mise en œuvre à l'aide de logiciels comme VMware[2], User-mode Linux UML [Dike 2006], ou Qemu [Bellard 2005]. Les implémentations sous forme de machine virtuelle offrent plus de flexibilité en terme de reconfiguration, déploiement et administration. Des exemples de tels pots de miel sont présentés dans [Provos et Holz 2007]. On peut par exemple citer Sebek [Honeywall 2005], Argos [Portokalidis et al. 2006], Honeybow [Zhuge et al. 2007] et Potemkin [Vrable et al. 2005].

Jusqu'à présent, les pots de miel haute interaction ont été principalement utilisés pour capturer et analyser des maliciels autonomes tels que les vers, virus et botnets. D'un autre côté, relativement peu de données expérimentales ont été publiées sur l'observation et l'analyse d'attaques non automatiques menées par des êtres humains, à l'aide de pots de miel haute interaction. Ceci est probablement du à la difficulté de mise en place de telles expérimentations.

Dans ces travaux, nous décrivons les leçons tirées du développement et du déploiement d'un pot de miel haute interaction visant cet objectif. Dans notre étude, nous nous focalisons sur la capture d'intrusions qui nécessite une connexion interactive via le service SSH. En effet,

[2]http ://www.vmware.com

des études récentes de tendances de vulnérabilités (voir par exemple le rapport publié par le *SANS Internet Storm Center* listant les principales menaces ciblant les systèmes informatiques sur Internet en 2007[3], qui a confirmé que SSH fait partie des services les plus ciblés par les attaquants). Notre pot de miel utilise le système d'exploitation Gnu/Linux. Il a été conçu de façon à autoriser la capture de tentatives de connexions SSH et des activités associées aux intrusions réussies. Trois types de données sont enregistrées par le pot de miel : 1) le mot de passe et le login testés par les attaquants pour obtenir l'accès au système, 2) les données échangées dans les connexions SSH (les commandes tapées) et 3) les appels systèmes générés par l'activité des attaquants.

Le pot de miel a été déployé sur Internet pendant plus d'un an (419 jours) pendant lesquels 552 362 connexions SSH ont été observées. Les résultats préliminaires basés sur l'analyse d'un sous-ensemble de ces données (les premiers six mois) ont été discutés dans [Alata et al. 2006]. Dans ce chapitre, nous présentons la méthodologie que nos avons développée pour analyser ces données et nous proposons une synthèse des principaux résultats obtenus. Deux étapes principales du processus d'intrusion ont été étudiées : 1) la première, généralement réalisée au moyen d'outils automatiques, concerne les attaques par dictionnaire destinées à obtenir l'accès sur le pot de miel, et 2) la seconde étape concerne l'activité réalisée par l'attaquant lorsqu'il est connecté sur le pot de miel. A l'issue de nos analyses, nous montrons que cette étape est généralement réalisée directement par des êtres humains, et non des outils automatiques.

La suite de ce chapitre est organisée comme suit. La section 5.2 présente la conception et l'implémentation du pot de miel. La section 5.3 donne une vue globale des données collectées et du processus d'intrusion observé. La section 5.4 est consacrée à l'analyse des attaques par dictionnaire et des logins et mots de passe testés par les attaquants. Les intrusions et les activités réalisées par les attaquants sont analysées dans la section 5.5. Enfin, la section 5.6 résume les principales conclusions de ces travaux.

5.2 L'architecture du pot de miel

Nos travaux ont pour but d'analyser principalement le comportement d'attaquants humains sur Internet (et non d'outils automatiques). Par conséquent, nous avons besoin de développer une plateforme permettant l'observation des activités réalisées par ce genre d'attaquants. Dans cette section, nous présentons l'architecture et les choix de conception de notre pot de miel ainsi que les détails d'implémentation.

[3] http ://www.sans.org/top20/2007

5.2.1 Principes

Notre but est d'observer et d'analyser le comportement d'êtres humains lorsqu'ils ont réussi à pénétrer un ordinateur. Pour cela, nous avons besoin de répondre à trois questions. Comment attirer des êtres humains ? Comment enregistrer leurs activités ? Comment contrôler leurs activités pour éviter qu'ils mettent en danger d'autres systèmes ?

Même s'il semble évident qu'un pot de miel à haute interaction semble désigné pour observer les attaquants de notre système, un tel outil va a priori capturer les activités réalisées par les attaquants humains mais aussi par les outils automatiques. Plusieurs articles ont analysé, grâce à des pots de miel, des maliciels tels que des vers, qui se propagent sur Internet. Dans [Dressler et al. 2007], les auteurs utilisent des pots de miel pour identifier des motifs d'attaques utilisés par des vers Internet. Des pots de miel ont également été utilisés pour analyser le comportement d'attaquants qui tentent de fabriquer des botnets ([Rajad et al. 2006, Zou et Cunningham 2006]).

Afin d'observer et enregistrer des attaques manuelles, nous avons de besoin de : 1) fournir aux attaquants une vulnérabilité à exploiter, et 2) d'assurer que les risques potentiels qui pourraient résulter de l'exploitation de cette vulnérabilité soient proprement contrôlés. Aussi, l'observation des activités liées aux attaquants nécessite des modifications du système opératoire. Pour cette raison, nous avons décidé d'utiliser le système Gnu/Linux pour notre pot de miel : nous sommes familiers avec ce système et la disponibilité du code source est un avantage indéniable. Dans nos expérimentations, nous sommes davantage intéressés par l'observation des activités menées par l'attaquant lorsqu'il s'est introduit dans le système plutôt que la façon dont il s'est introduit sur le système. Pour cette raison, nous avons choisi une vulnérabilité très classique : des comptes d'utilisateurs avec des mots de passe faibles. Grâce à cela, nous limitons les intrusions réalisées par des outils automatiques. En effet, il est peu probable pour un outil automatique de mener une attaque cohérente au travers du service SSH alors que c'est particulièrement intéressant pour un être humain puisqu'il obtient une connexion interactive avec le système[4].

Afin d'offrir une cible intéressante aux attaquants, nous avons décidé de mettre en oeuvre plusieurs pots de miel. Ceci permet également d'analyser le comportement d'un attaquant qui essaierait de faire des rebonds de site en site. Cependant, administrer plusieurs machines physiques susceptibles d'être mises à mal par les attaquants est une tâche fastidieuse. Ce problème peut être atténué grâce à la technique de virtualisation qui nous permet d'administrer une seule machine physique et d'exécuter plusieurs systèmes opératoires virtuels au-dessus. Nous avons donc décidé d'utiliser cette technique au travers du logiciel `VMware`. Ainsi, la copie, la reconfiguration et la modification de système opératoire devient plus aisée. De plus, si l'attaquant réussit à détruire une partie du système, la procédure de restauration est simplifiée.

Notre pot de miel doit être adapté à l'observation d'attaques réalisées par des êtres humains, mais

[4]On peut cependant noter que certains vers incluent des services de découverte de mots de passe, tels que décrits dans [Fsecure 2006].

doit rester aussi transparent que possible pour ne pas être facilement identifié en tant que tel. La modification du système opératoire virtuel est nécessaire pour collecter les informations nous permettant d'analyser le comportement des attaquants. Nous pouvons distinguer trois sources d'information à capturer : 1) les couples (login et mot de passe) testés par les attaquants pour obtenir l'accès au système ; 2) les données échangées dans chaque connexion SSH ; et 3) tous les appels système générés par l'activité de nos machines virtuelles. La section 5.2.2 présente les détails d'implémentation permettant de recueillir ces données. La stratégie adaptée pour sauvegarder ces données est décrite dans la section 5.2.3. La figure 5.1 présente une vue globale de l'implémentation de notre pot de miel.

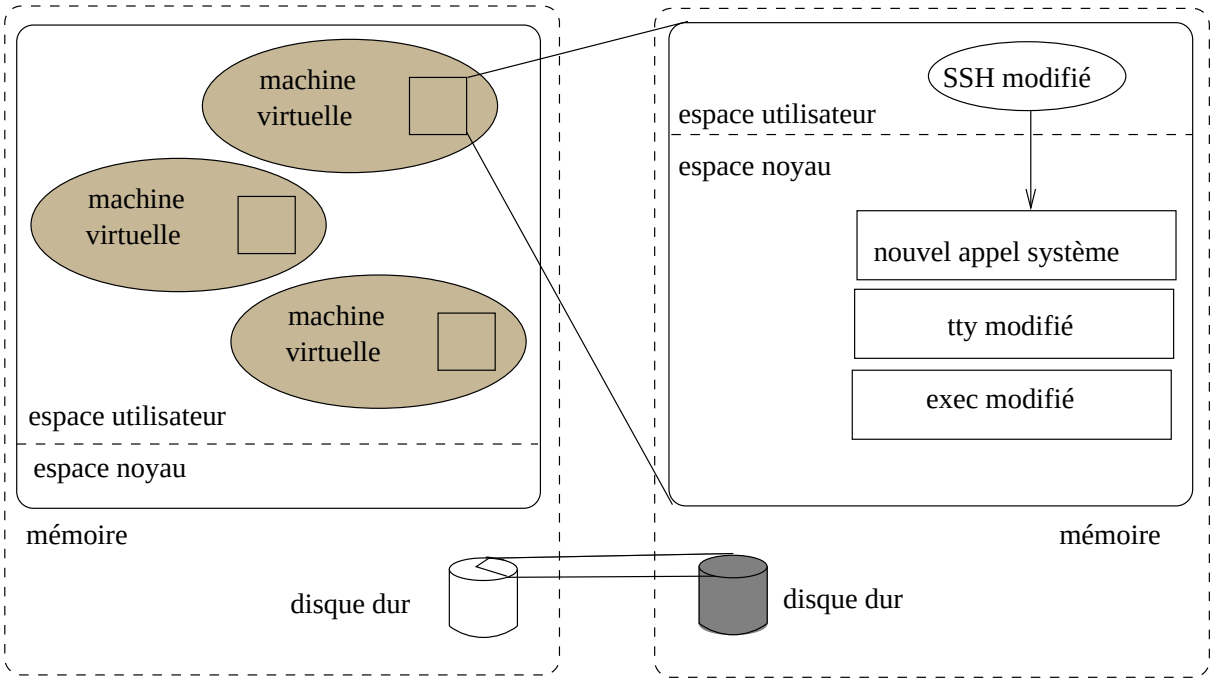


FIG. 5.1 – Implémentation du pot de miel haute interaction

5.2.2 Modification du système d'exploitation

Pour modifier le noyau d'un système Linux, deux principales approches existent. La première consiste à directement patcher le noyau et la seconde consiste à charger dynamiquement un module dans le noyau. La seconde approche est en général plus facile à détecter par les attaquants. Aussi, nous avons adopté la première approche, pour être le plus transparent possible. Les modifications effectuées au niveau du noyau de chaque machine virtuelle concernent : 1) la fonction qui permet d'intercepter chaque appel système exécuté par l'attaquant est instrumentée ainsi

que 2) les fonctions qui permettent de gérer tous les caractères tapés par l'attaquant lorsqu'il utilise un shell interactif. De plus, de façon à capturer les logins et mots de passe testés par les attaquants, nous avons créé un nouvel appel système dans le noyau et modifié le serveur SSH de façon à ce qu'il utilise cet appel.

La première modification du noyau concerne le pilote `tty`. Ce pilote contrôle tous les terminaux et pseudo-terminaux sur les machines Linux (et particulièrement tous les caractères entrés par les utilisateurs sur ces terminaux). Ce pilote définit les fonctions de lecture et d'écriture sur ces terminaux. La modification de ces fonctions nous permet donc d'intercepter tous les caractères saisis et affichés par l'attaquant sur son terminal.

La seconde modification est faite dans l'appel système `exec`, qui permet d'exécuter tout programme sur le système. En modifiant cet appel système, nous pouvons mémoriser la liste des fichiers exécutés par l'attaquant.

La dernière modification du noyau consiste à créer un nouvel appel système. Le serveur SSH utilise ce nouvel appel système pour mémoriser chaque nom et mot de passe testé par l'attaquant.

5.2.3 Sauvegarde des données collectées

Sauvegarder de façon sûre dans une base de données toutes les données collectées peut s'avérer consommateur de ressources. Des attaques statistiques peuvent être utilisées pour détecter une augmentation anormale de l'exécution des appels système et ainsi détecter cette sauvegarde, et donc l'utilisation de pots de miel. Pour faire face à ce problème, certains approches, par exemple dans [Alberdi et al. 2005], utilisent des mécanismes permettant de moduler la quantité d'information sauvegardée en fonction de la charge de la CPU. Pour notre implémentation, nous avons décidé de sauvegarder les informations capturées de façon périodique et indépendamment de la charge de la CPU. Que la quantité d'information à sauvegarder soit grande ou non, la sauvegarde est réalisée à heure fixe. Cette approche rend plus difficiles les attaques statistiques.

Cependant, notre stratégie de sauvegarde a un inconvénient : toutes les données collectées doivent être localement stockées sur la machine virtuelle en attendant l'heure de la sauvegarde. Comme les modifications du système d'exploitation que nous avons présentées dans le paragraphe 5.2.2 sont faites dans le noyau, nous avons donc décidé de stocker les informations collectées dans la mémoire du noyau (dans son espace d'adressage). Cette zone est préallouée statiquement de façon à éviter les allocations dynamiques, qui pourrait induire une surcharge de l'activité CPU. De façon à optimiser l'espace mémoire, les données collectées sont compressées à l'aide de l'algorithme LZRW1 [Williams 1991].

Les données capturées sont stockées dans la mémoire du noyau de la machine virtuelle et donc pourraient être identifiées par un attaquant. Cependant, ceci est difficile car cela implique que 1) l'attaquant possède les privilèges root ; 2) il est capable d'identifier la zone mémoire utilisée pour le stockage des données et 3) il identifie que l'algorithme LZRW1 est utilisé.

Les données sauvegardées dans la base de données ne sont pas envoyées via le réseau, comme cela est souvent réalisé dans beaucoup d'implémentations de pots de miel haute interaction (cf. [Alberdi et al. 2005]). Les données sont copiées sur la machine physique : tout le contenu de la mémoire d'une machine virtuelle est directement accessible depuis la machine physique. L'avantage de cette stratégie est qu'elle est difficile à identifier par l'attaquant. C'est une simple copie de données d'une région de mémoire vers le disque dur du système réel qui est réalisée une fois par jour à une heure précise (cette opération dure moins d'une seconde). Nous pensons que cette solution est meilleure qu'une sauvegarde par le réseau car l'activité du réseau est plus facilement observable par l'attaquant.

5.2.4 Déploiement et analyses préliminaires

La figure 5.2 présente la plateforme utilisée pour notre expérimentation. Elle est constituée de trois machines virtuelles (avec le système d'exploitation Gnu/Linux) qui s'exécutent au-dessus du logiciel VMWare. Ces machines virtuelles sont des machines standards incluant les outils de développement usuels (éditeur de texte, compilateur C, etc.), patchées avec nos modifications noyau. Seules les connexions entrantes sur le service SSH (port 22) sont autorisées par un pare-feu. Toutes les connexions sortantes sont interdites de façon à ce que les attaquants ne puissent utiliser notre pot de miel pour attaquer d'autres machines sur Internet (ce qui nous placerait dans une situation illégale).

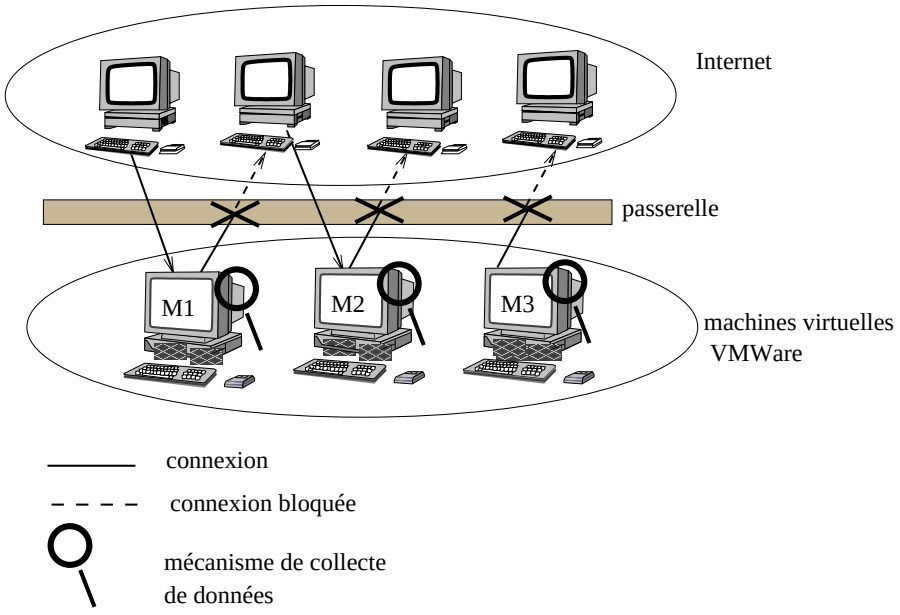


FIG. 5.2 – Plateforme d'expérimentation avec notre pot de miel

La connexion sur le service SSH est l'unité d'observation fournie par le pot de miel. Chaque connexion est caractérisée par cinq attributs : la date de la connexion, l'adresse IP qui est à l'origine de la connexion, le login et le mot de passe testés, un booléen indiquant si le mot de passe est correct et un booléen indiquant si la connexion contient des commandes interactives.

Chaque expérimentation a été menée en deux temps. Nous avons commencé durant le premier mois d'expérimentation par déployer le pot de miel avec le service SSH mais sans définir de compte d'utilisateur. Les connexions SSH infructueuses observées pendant cette période nous ont permis d'avoir une idée des couples (login/mot de passe) les plus testés. Nous avons ensuite créé 17 comptes d'utilisateurs.

La plateforme a été déployée sur Internet pendant 419 jours, période pendant laquelle 654 adresses IP ont essayé de se connecter au service SSH des pots de miel, pour un total de 552 362 connexions. Pour chaque connexion, un couple (login/mot de passe) a été fourni par l'attaquant. Certains couples ont été utilisés plusieurs fois. Nous en avons recensé 98347. Une analyse détaillée de ces couples est présentée dans la section 5.4.

La plupart des connexions SSH ont échoué. En considérant les connexions réussies pour chaque compte créé dans notre expérimentation, comme illustré dans la figure 5.3, nous pouvons distinguer deux étapes principales correspondant à deux types différents d'activité. La première étape identifiée par τ_1 correspond à la période observée entre la date de création du compte d'utilisateur et la première connexion réussie pour ce compte. La seconde étape, identifiée par τ_2, correspond à la période observée entre la première connexion réussie pour ce compte et la première connexion réussie pour ce compte avec saisie de commandes par l'attaquant.

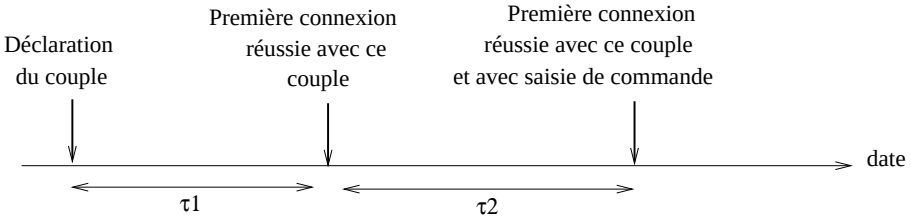


FIG. 5.3 – Relation entre τ_1 et τ_2

Le tableau 5.1 présente les durées correspondant à τ_1 et τ_2, observées pour chaque compte d'utilisateur. Il est intéressant de constater que τ_2 n'est jamais nul (excepté pour le premier compte). Ceci indique clairement qu'il y a deux étapes dans le processus d'intrusion (nous reviendrons plus en détails sur ce processus dans la section 5.3). La première étape consiste à deviner le mot de passe correct pour un compte particulier, mot de passe qui n'est pas utilisé immédiatement. La seconde étape, généralement quelques jours plus tard, consiste à utiliser ce mot de passe pour réaliser une connexion interactive sur le pot de miel avec saisie de commandes.

	τ_1	τ_2
UA1	1 jour	4 jours
UA2	1/2 journeée	4 minutes
UA3	15 jours	1 jour
UA4	5 jours	10 jours
UA5	5 jours	nulle
UA6	1 jours	4 jours
UA7	5 jours	8 jours
UA8	1 jour	9 jours
UA9	1 jour	12 jours
UA10	3 jours	2 minutes
UA11	7 jours	4 jours
UA12	1 jour	8 jours
UA13	5 jours	17 jours
UA14	5 jours	13 jours
UA15	9 jours	7 jours
UA16	1 jour	14 jours
UA17	1 jour	12 jours

TAB. 5.1 – Historique de la découverte des comptes

5.3 Le processus d'attaque

Cette section est dédiée à l'analyse du processus d'attaque sur notre pot de miel. Nous introduisons tout d'abord le concept de session d'attaque et nous décrivons comment nous avons classifiée ces sessions de façon à analyser le processus d'attaque correspondant. Ensuite, nous présentons, dans les sections 5.4 et 5.5, les deux étapes principales de ce processus d'attaque.

5.3.1 Identification des sessions d'attaques

Parmi les connexions SSH, certaines incluent des commandes. Nous appelons ces connexions *connexions d'action*. Les autres, qui constituent la majorité, n'incluent pas de commandes. Nous les appelons *connexions d'authentification*.

Les connexions d'authentification peuvent bien sûr provenir d'erreurs, telles qu'un utilisateur ayant saisi une mauvaise adresse IP. Il se connecte sur notre pot de miel, deux ou trois fois, réalise son erreur et abandonne. Mais, plus probablement, ces connexions correspondent à des *attaques par dictionnaire*. Une attaque par dictionnaire est une succession de connexions d'authentification rapprochées dans le temps, qui visent à découvrir des mots de passe faibles sur notre machine. La majorité de ces connexions échoue ; celles qui réussissent correspondent à

la découverte d'un mot de passe. Il est très probable que les mots de passe découverts vont servir à de futures attaques qui vont utiliser des connexions d'action, incluant des commandes interactives.

Afin d'obtenir un meilleur aperçu des processus d'attaque observés sur notre pot de miel et analyser les activités réalisées par les intrus, nous avons regroupé le grand nombre de connexions SSH que nous avons enregistrées en *sessions*. Une *session* est une séquence de connexions rapprochées dans le temps, réalisées depuis la même adresse IP et ciblant la même adresse IP (c'est-à-dire la même machine virtuelle). Une session représente le comportement d'un attaquant sur le pot de miel. Par exemple, si un attaquant réalise une séquence de connexions dans un temps très court le jour J, qu'il ne fait ensuite aucune connexion pendant 2 jours et qu'il réalise une nouvelle séquence de connexions dans un temps très court le jour $J + 3$; alors, nous identifions deux sessions d'un même attaquant.

Comme expliqué dans [Alata 2007], l'identification des sessions d'attaques peut être faite grâce à l'algorithme de la fenêtre glissante, en considérant un seuil qui définit la durée maximum séparant la réception sur le pot de miel de deux paquets consécutifs appartenant à la même session. La définition de ce seuil est généralement basée sur des heuristiques. La figure 5.4 représente le nombre de sessions obtenues en groupant les connexions SSH pour différentes valeurs du seuil (en secondes). On peut constater que le nombre de sessions ne change pas significativement lorsque le seuil dépasse 16 secondes. Dans ce cas, 1940 sessions sont obtenues en regroupant les 552 362 connexions SSH enregistrées par le pot de miel. Plus de détails sont fournis dans [Alata 2007].

5.3.2 Classification des sessions

Pour la classification des sessions, nous avons défini deux critères : la *dangerosité* et la *persévérance*. Une session est dangereuse si l'attaquant correspondant réussit à pénétrer dans le système et exécute des commandes. Autrement dit, une session est dangereuse si elle inclut des connexions d'action. Une session est persévérante si le nombre de connexions d'authentification inclues dans cette session est supérieur à un certain seuil. Nous avons positionné ce seuil à 9 car ce chiffre est suffisamment grand pour distinguer les réelles attaques par dictionnaire des erreurs. En appliquant ce premier critère à l'ensemble des attaques, nous avons identifié deux classes : la classe des sessions dangereuses (210) et celle des sessions non dangereuses (1730). En appliquant le critère de persévérance à cette seconde classe, nous avons identifié 1391 sessions qui ne sont pas dangereuses mais presévérantes et 339 qui sont non dangereuses et non persévérantes. Ces 1391 sessions non dangereuses mais persévérantes sont les attaques par dictionnaire que nous considérons dans la suite de ce chapitre. Les 210 sessions dangereuses sont ce que nous appelons dans ce chapitre les *intrusions*.

Cette classification est représentée dans la figure 5.5.

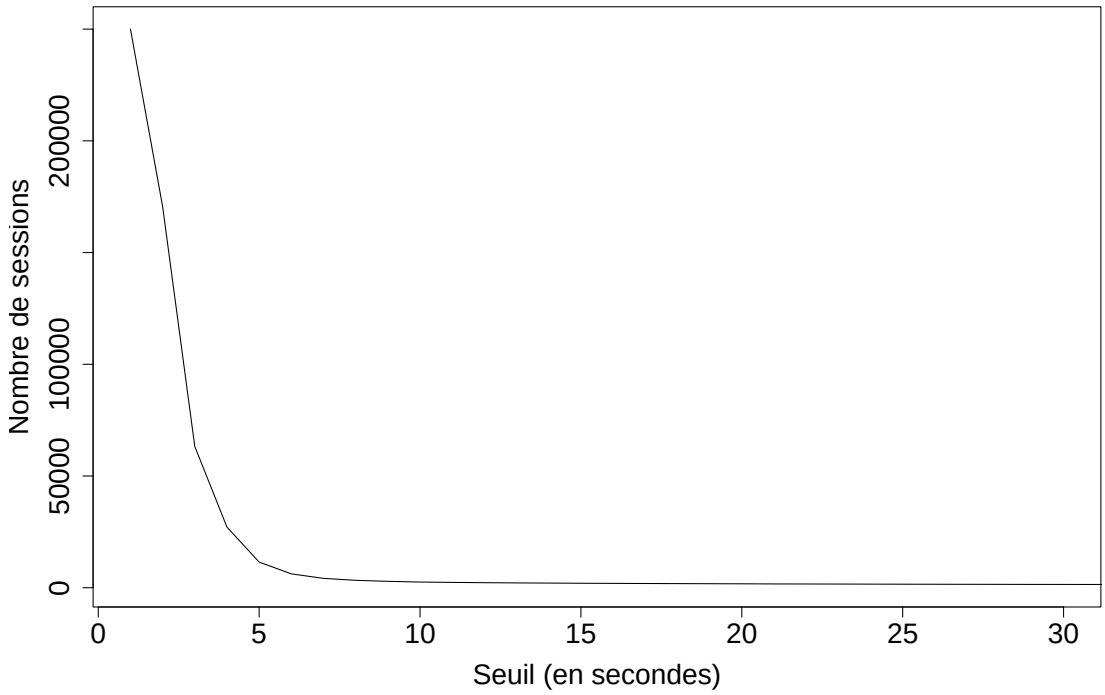


FIG. 5.4 – Evolution du nombre de sessions en fonction du seuil

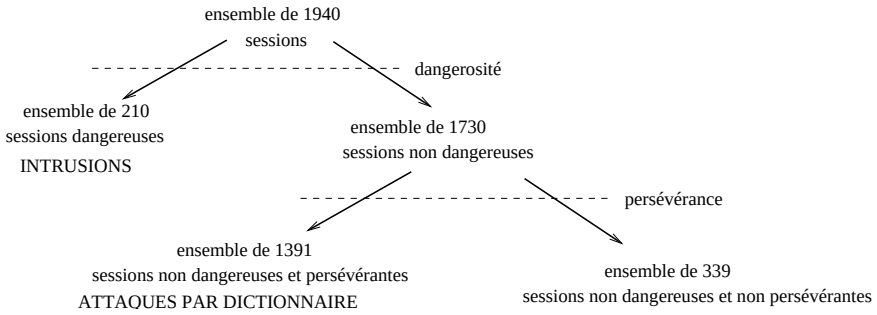


FIG. 5.5 – Classification des sessions

5.3.3 Les étapes principales du processus

A partir de l'ensemble des attaques par dictionnaire et des intrusions, nous avons fait des analyses de façon à identifier les relations entre ces différents types d'attaque, pour pouvoir mieux

appréhender le processus global d'attaque. Nous avons en particulier examiné les adresses IP des machines attaquantes ainsi que les dates de début des attaques.

Nous avons observé qu'une attaque par dictionnaire qui permet de découvrir un couple valide (login/mot de passe) précède **toujours** une intrusion réalisée sur notre pot de miel avec ce couple. De plus, en comparant les adresses IP des machines ayant réalisé des attaques par dictionnaire et les adresses IP des machines qui ont réalisé des intrusions ; nous avons constaté que **cette intersection est vide**. Cela signifie que les machines qui réalisent des attaques par dictionnaire sont spécialisées dans ce type d'attaque et ne sont pas utilisées pour réaliser des intrusions. Ce résultat est d'autant plus intéressant que nous l'avons vérifié sur une longue période (419 jours).

Nous avons également comparé les adresses IP qui réalisent les attaques par dictionnaire et les intrusions avec les adresses IP observées sur les pots de miel basse interaction de la plateforme `Leurre.com`. **Aucune** de ces adresses n'a été observée sur les pots de miel basse interaction. Ce résultat confirme la spécialisation des machines qui réalisent seulement un type d'attaque. De plus, ainsi que le montre [Pouget et al. 2004], les attaques par dictionnaire ne sont pas nécessairement réalisées aveuglément sur n'importe quelle adresse IP de l'Internet. Une étape préliminaire de scan, pour identifier les services disponibles sur les futures victimes, précède souvent l'attaque par dictionnaire. Cependant, l'étape de scan de ports n'est pas systématique dans tous les processus d'attaque connus. Ceci a été observé par exemple dans l'étude expérimentale présentée dans [Panjwani et al. 2005], dans laquelle les auteurs montrent que plus de 50% des attaques observées n'ont pas été précédées par des scans de port.

Dans la suite, nous nous focalisons sur les deux principales étapes du processus global d'attaque observé sur le pot de miel :

- les attaques par dictionnaire destinées à découvrir des comptes d'utilisateurs valides,
- le processus d'intrusion lui-même, qui consiste à exécuter des commandes sur la machine compromise.

Nous avons choisi dans ce résumé de nos travaux de nous attarder davantage sur la seconde étape de l'intrusion, aussi nous ne donnerons que les principaux résultats de l'étude de la première étape, sans entrer dans les détails que l'on peut trouver dans [Alata et al. 2008].

5.4 Les attaques par dictionnaire

L'objectif de l'étude de cette première étape est d'apporter des réponses à des questions telles que : y-a-t-il autant de dictionnaires que d'attaques par dictionnaire ou y-a-t-il simplement quelques dictionnaires partagés par la communauté des attaquants ?

Nous allons simplement ici présenter un résumé de la méthode permettant d'identifier les *coeurs de dictionnaire* et nous présenterons ensuite les résultats obtenus à partir de l'analyse ces coeurs. Des informations plus complètes peuvent être consultées dans [Alata et al. 2008].

Notre méthodologie est constituée des points suivants :

- Nous avons tout d'abord défini la notion de *vocabulaire* comme : l'ensemble des couples (login/mot de passe) utilisés par une attaque par dictionnaire. Puis nous avons identifié les similarités entre vocabulaires en introduisant une distance entre deux vocabulaires.
- A l'aide de cette distance, nous avons ensuite regroupé les vocabulaires en grappes. Deux vocabulaires sont regroupés au sein d'une même grappe quand au moins 75% des mots du plus petit sont partagés par le plus gros. Cette étape nous a permis d'identifier 434 grappes à partir de 1391 dictionnaires. Seulement 4% de ces grappes (16 grappes) représentent 40% des attaques par dictionnaire observés. Ce résultat nous permet de conclure que **peu de dictionnaires sont partagés par les attaquants sur Internet**.
- Nous avons également analysé, pour les trois plus grosses grappes (contenant respectivement 187, 161 et 23 vocabulaires), les durées pendant lesquelles les dictionnaires correspondants ont été utilisés lors de l'expérimentation. Nous avons constaté tout d'abord que ces dictionnaires ont été utilisés pendant une très longue période. Par exemple, la grappe incluant 187 attaques par dictionnaire est associée à 63 adresses IP différentes. Ces adresses ont été observées à plusieurs reprises sur le pot de miel, régulièrement durant les 419 jours de l'expérimentation. Une attaque par dictionnaire de cette grappe a été réalisée en moyenne tous les 2 jours. Il est important de noter qu'aucune des adresses IP de la grappe contenant 187 vocabulaires n'est associée aux deux autres grappes. En revanche, 6 parmi les 18 adresses IP de la grappe contenant 23 vocabulaires sont aussi associées à la grappe contenant 161 vocabulaires. Ainsi, ces analyses nous permettent de conclure que ces trois grappes identifient **des communautés différentes qui ont chacune utilisé le même dictionnaire tout au long de la période d'observation**.
- Nous avons ensuite défini la notion de *coeur de dictionnaire* associé à chaque grappe comme étant l'ensemble des mots partagés par au moins deux vocabulaires de la grappe. En effet, le coeur de dictionnaire n'est pas l'union de tous les dictionnaires parce qu'il inclurait des couples qui ont été testés simplement une fois, ce qui est hasardeux. De la même façon, nous n'avons pas considéré l'intersection de tous les dictionnaires parce qu'éliminer les couples testés par quelques dictionnaires mais pas tous serait trop restrictif.
- Nous avons ensuite analysé les 434 coeurs de dictionnaire et obtenu les résultats significatifs suivants :
  - 3/4 des coeurs de dictionnaire contient plus de 75% de mots sans signification
  - Les mots composant les coeurs de dictionnaire ne **tiennent pas compte de l'emplacement géographique de leur cible** : notre pot de miel est en France et la majorité des mots composant les coeurs de dictionnaire sont anglais
  - Les coeurs de dictionnaire **contiennent des couples particuliers**, correspondant aux comptes d'utilisateurs ou administrateurs **créés par défaut sur les systèmes Windows ou Unix**. En revanche, les attaquants ne semblent pas savoir ou ne pas s'intéresser au type de système qu'ils attaquent. Notre pot de miel est une machine Linux et le compte `guest`, spécificité du système Windows, est inclus dans 113 coeurs de dictionnaires.
  - Nous avons noté qu'environ un tiers des coeurs de dictionnaire contient **75% de couples**

composés de logins et mots de passe identiques. Ce résultat nous laisse penser que les attaquants considèrent toujours aujourd'hui que ce genre d'attaques est efficace, c'est-à-dire qu'encore aujourd'hui malheureusement, beaucoup d'utilisateurs sur Internet choisissent ce genre de mots de passe.

5.5 L'étape d'intrusion

Une attaque par dictionnaire est seulement une étape préliminaire qui permet ensuite à un attaquant de réaliser une *intrusion* (une session qui inclut des connexions d'action, cf. 5.3.2). Parmi les 1490 sessions que nous avons identifiées, 210 sont considérées comme des intrusions et sont étudiées dans cette section.

Le nombre d'adresses IP sources différentes à l'origine de ces 210 intrusions est de 57. Elles ciblent 21 différents comptes d'utilisateurs (après la première étude présentée dans le chapitre 5.2.4, nous avons ajouté 5 nouveaux comptes). Le tableau 5.2 présente, pour chaque compte, le nombre d'intrusions, les adresses IP et mots de passe associés. Tout d'abord, nous remarquons que plusieurs mots de passe peuvent être associés au même compte. Ceci résulte du fait que la première chose que fait un attaquant lorsqu'il est connecté sur le pot de miel est changer le mot de passe. Il est ensuite particulièrement intéressant de constater qu'**aucune adresse IP apparaissant dans ce tableau n'a été utilisée pour réaliser des attaques par dictionnaire**. On peut en déduire que la communauté des attaquants dédient leurs machines à un certain type d'attaques. Il semble aussi fortement probable que les attaquants soient organisés de façon à collaborer efficacement en utilisant différents types de machines pour réaliser différentes attaques.

Dans les paragraphes suivants, nous analysons le comportement des intrus. Tout d'abord, nous essayons de déterminer leur nature : êtres humains ou outils automatiques. Ensuite, nous donnons un aperçu des différents types d'activité que les intrus ont menés sur le pot de miel. Enfin, nous discutons leur compétence.

5.5.1 Nature des intrus : êtres humains ou outils automatiques

Les attaquants peuvent être des êtres humains ou des outils automatiques qui exécutent des tâches élémentaires. Nous avons considéré deux critères principaux pour identifier les activités réalisées par des êtres humains : 1) les fautes de frappe et 2) le mode de transmission des données entre l'attaquant et le pot de miel.

Les fautes de frappe constituent le premier élément qui permet de caractériser un être humain. Pour corriger ses fautes, l'attaquant appuie régulièrement sur la touche *backspace* (code ascii 127). Aussi, la première possibilité pour déterminer la nature des attaquants est d'analyser la séquence de touches tapées par les attaquants et de vérifier si la touche backspace est présente.

Compte	Nombre d'intrusions	Nombre de mots de passe	Nombre d'adresses IP
C_1	11	2	6
C_2	3	2	2
C_3	39	2	10
C_4	2	2	1
C_5	3	2	2
C_6	22	2	7
C_7	21	2	2
C_8	32	2	3
C_9	14	2	6
C_{10}	2	1	2
C_{11}	3	2	2
C_{12}	17	3	3
C_{13}	4	2	1
C_{14}	3	2	2
C_{15}	1	1	1
C_{16}	3	2	1
C_{17}	19	1	7
C_{18}	2	2	1
C_{19}	1	1	1
C_{20}	5	1	3
C_{21}	2	1	1

TAB. 5.2 – Nombre d'intrusions par compte

En faisant cette analyse, nous supposons qu'il est très peu probable que des outils automatiques essaient d'imiter le comportement humain en incluant délibérément des fautes de frappe.

Dans certains cas, l'intrus utilise très peu de commandes et a donc très peu de chances de faire des fautes de frappe. Il est également possible qu'un être humain faisant particulièrement attention puisse entrer ses commandes sans faute de frappe. Dans ces situations, le protocole utilisé pour envoyer les données à travers le réseau peut nous aider alors à faire la distinction. En effet, la transmission des données entre le client et le serveur d'une connexion SSH est asynchrone. La plupart du temps, les implémentations des clients SSH utilisent la fonction `select()` pour récupérer la touche saisie par l'utilisateur. Lorsque ce dernier appuie sur la touche, cette fonction se termine et le client envoie le caractère saisi au serveur. Dans le cas d'un copier/coller dans un terminal, un tampon mémoire contenant toutes les données issues du copier/coller est envoyé sur la connexion SSH. Dans notre implémentation, les données envoyées sur la connexion SSH sont reçues par la fonction `tty_read()`. Nous considérons donc que si cette fonction retourne plus d'un caractère, alors les données ont été envoyées par une opération copier/coller. En résumé,

nous considérons qu'une intrusion sans faute de frappe et dont les données sont envoyées par blocs est réalisée par un outil automatique. En revanche, si l'intrusion transfère des données à la fois caractère par caractère et bloc par bloc, nous considérons qu'elle est réalisée par un être humain qui réalise par moment des raccourcis clavier ou des copier/coller.

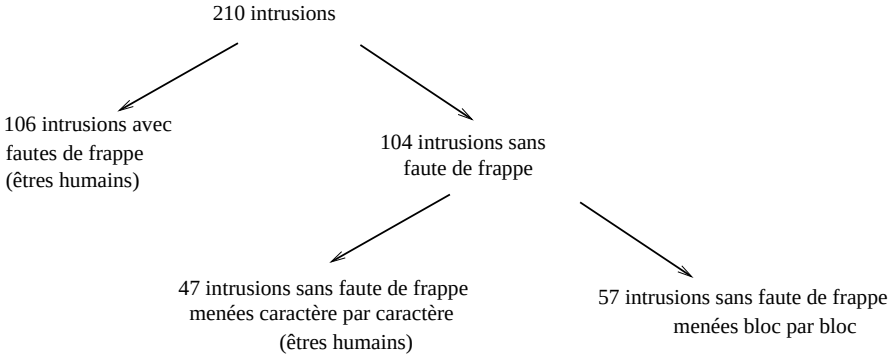


FIG. 5.6 – Nature des intrus

En appliquant ces deux critères (faute de frappe et mode de transfert de données), nous obtenons une classification des intrusions présentées dans la figure 5.6. 106 intrusions parmi 210 contiennent des fautes de frappe. Selon nos critères, ces 106 intrusions ont été très probablement réalisées par des êtres humains. En ce qui concerne les 104 intrusions qui n'incluent pas de fautes de frappe, 47 ont utilisé un mode de transmission caractère par caractère. Selon notre second critère, ces intrusions ont très probablement été menées par des être humains. Pour les autres intrusions, les activités menées ne nous permettent pas de conclure sur la nature de l'intrus. En effet, elles contiennent trop peu de commandes (parfois une seule, comme `w` par exemple). Pour conclure, plus de 75% des intrusions ont été réalisées par des êtres humains.

5.5.2 Les activités des intrus

Dans cette section, nous analysons les différentes commandes utilisées par les intrus de façon à identifier les différents types d'activités qu'ils ont menées, ceci en considérant les 210 intrusions.

Notons tout d'abord que tous les intrus commencent par changer le mot de passe du compte qu'ils utilisent sur le pot de miel. Changer ce mot de passe leur permet d'être la seule personne à pouvoir utiliser ce compte mais à la fois, cela trahit la présence de l'attaquant sur le pot de miel : si l'utilisateur légitime correspondant utilise régulièrement ce compte, il détectera facilement qu'il a été attaqué. En ce sens, le changement systématique du mot de passe nous paraît un peu étonnant.

La seconde activité réalisée par les intrus est le téléchargement de programmes malveillants. La commande la plus utilisée pour ce téléchargement est `wget`. 96 intrusions sur 210 l'ont utilisée.

Rappelons que les connexions sortantes sont bloquées par un pare-feu, aussi cette commande échoue systématiquement. Cependant, les intrus peuvent parfaitement télécharger leurs programmes malveillants depuis Internet sur le pot de miel grâce aux connexions SSH entrantes autorisées. Il suffit pour cela à l'intrus d'utiliser la commande `sftp` et d'utiliser la commande `put`. De façon surprenante, seulement 30% des intrus pensent à utiliser la commande `sftp` lorsqu'ils sont bloqués par la commande `wget`.

Deux explications semblent possibles. Certains intrus utilisent probablement des recettes de cuisine disponibles sur Internet. Ces recettes proposent certaines méthodes d'attaque, la plupart du temps basées sur la commande `wget` pour télécharger des fichiers. Les intrus qui utilisent ces recettes ne connaissent pas nécessairement d'autres méthodes pour télécharger les fichiers. En ce qui concerne la seconde explication possible, nous pouvons imaginer que les intrus suspectent quelque chose d'anormal lorsqu'ils réalisent que la commande `wget` échoue et qu'ils décident donc de stopper leur attaque. En fait, la première explication nous semble la plus probable. En effet, nous avons remarqué que les sessions correspondant aux activités des intrus qui abandonnent après l'échec de la commande `wget` contiennent beaucoup d'opérations de type copier/coller. Ceci nous laisse penser que ces intrus utilisent des recettes. De plus, nous avons remarqué que ces intrus en général reviennent sur le pot de miel plusieurs jours après et qu'ils essaient à nouveau la même séquence de commandes. Certains reviennent et réessaient plusieurs fois, comme s'ils étaient incapables d'imaginer d'autres méthodes.

Lorsque les intrus ont réussi à télécharger leur maliciel sur le pot de miel, ils décompressent leurs fichiers. Environ 75% d'entre eux n'utilisent pas le répertoire du compte d'utilisateur qu'ils ont exploité pour entrer sur le pot de miel mais plutôt des répertoires standards du système pour lesquels les droits d'écriture sont positionnés, tels que `/tmp`, `/var/tmp` ou `/dev/shm`. Ces répertoires étant utilisés et partagés par tous les utilisateurs légitimes d'un système Unix, il est difficile d'identifier une activité malveillante issue de ces répertoires. Dans ces répertoires standards, les intrus créent leur propre répertoire caché (répertoire dont le nom commence par le caractère '.') ou le répertoire " ". Ceci est étonnant parce-que cela montre qu'ils essaient de dissimuler leur activité sur le pot de miel alors que ces mêmes intrus ont changé le mot de passe du compte d'utilisateur qu'ils exploitent sur le pot de miel, trahissant par là-même leur présence.

Nous avons identifié trois types d'activité des intrus. La première est l'activité de scan. Les intrus n'ont pas scanné tous les ports mais uniquement le port SSH (22), de façon à trouver d'autres machines qui proposent ce service. Notons que les scans n'ont jamais été réalisés sur le réseau local lui-même. Il semble que les attaquants ne sont pas intéressés par l'exploitation d'autres machines sur le réseau local du pot de miel, probablement parce-que cela risquerait de trahir davantage leur présence. Ils utilisent en fait le pot de miel comme un rebond pour attaquer d'autres cibles sur d'autres réseaux. De plus, l'activité de scan est dédiée au service SSH, comme si les intrus étaient spécialisés dans ce type de tâche. La seconde activité que nous avons recensée est l'utilisation de client `irc`. Ces clients sont généralement inclus dans les outils de déploiement de botnets. En effet, le client `irc` est utilisé par les attaquants pour communiquer avec des "maîtres" qui envoient des ordres d'attaques ou de propagations aux machines com-

promises qu'ils contrôlent. Cette activité correspond donc clairement à la fabrication de botnets sur Internet.

La troisième activité consiste à essayer d'obtenir les privilèges du super utilisateur `root`. De façon surprenante, très peu d'intrus (seulement 3) ont essayé d'obtenir ces privilèges. Deux maliciels différents ont été utilisés pour cela. Le premier exploite une vulnérabilité de l'appel système `mremap` [Cert Mremap 2004] et une autre qui affecte le gestionnaire de la mémoire des processus [Cert Dobrk 2003]. Ces exploits ne pouvaient pas fonctionner sur notre pot de miel à cause d'un conflit de version du noyau. L'intrus aurait pu réaliser cette incompatilibité avant de lancer le maliciel à l'aide de la commande `uname -a` par exemple. Cependant, l'intrus a lancé son exploit, qui a donc échoué. Le second maliciel exploite une vulnérabilité dans le programme `ld`. Cet exploit a été testé par 3 intrus, mais n'a pas fonctionné.

5.5.3 Compétence et intention des intrus

Les intrus peuvent être classés en deux catégories : les *scripts kiddies*, personnes inexpérimentées qui téléchargent et utilisent des maliciels sans réellement les comprendre et les *black hats*, des experts de la sécurité qui sont réellement dangereux parce qu'ils maîtrisent parfaitement ce qu'ils font.

Notre expérimentation nous laisse penser que la plupart des attaquants que nous avons observés sur notre pot de miel sont inexpérimentés. Certains semblent même ne pas être réellement familiers du système Unix et de la gestion des droits d'accès ou des processus : ils essaient d'effacer des fichiers pour lesquels ils n'ont pas les privilèges correspondants, ils essaient de tuer des processus ne leur appartenant pas, etc. La plupart d'entre eux n'ont pas effacé les fichiers d'historique (tels que `.bash_history`, situé dans le répertoire du compte d'utilisateur) qui enregistrent les différentes commandes utilisées par l'attaquant.

La plupart des intrus ont identifié le système, en essayant d'obtenir de l'information concernant le matériel de la machine (grâce au fichier `/proc/cpuinfo` par exemple). Mais aucun d'entre eux n'a essayé, à l'aide de techniques pourtant connues, de vérifier si `VMware` était installé sur le système (ce qui aurait pu laisser soupçonner la présence d'un pot de miel).

5.6 Conclusion

Nous avons présenté les enseignements principaux tirés du développement et du déploiement d'un pot de miel haute interaction destiné à l'observation et l'analyse des activités réalisées par des attaquants sur Internet, incluant à la fois êtres humains et outils automatiques. Les résultats sont basés sur une expérimentation menée sur une longue période de temps (419 jours), pendant laquelle nous avons observé les différentes étapes qui amènent un attaquant à pénétrer dans la machine vulnérable et à réaliser des opérations une fois connecté. Nous avons considéré le

cas d'attaques perpétrées via le service SSH s'exécutant sur une plateforme Gnu/Linux. Deux classes d'attaques ont été analysées : 1) les attaques par dictionnaire visant à obtenir l'accès au système cible, et 2) les intrusions correspondant aux attaques interactives réalisées à l'aide des logins et mots de passe obtenus précédemment.

Les résultats de notre expérimentation sont de deux ordres. Premièrement, ils sont cohérents avec la connaissance informelle partagée par les experts en sécurité. Nos contributions résident dans la mise en œuvre d'une expérimentation et dans la réalisation d'analyses rigoureuses qui ont confirmé les suppositions informelles. Deuxièmement, les analyses détaillées de nos observations ont révélé des faits intéressants. Il apparaît que les attaquants utilisent et partagent relativement peu de dictionnaires, incluant un grand nombre de couples login/mot de passe qui sont couramment utilisés dans les système Unix et Windows, ainsi qu'un grand nombre de couples dont le login et le mot de passe sont identiques. De plus, les attaquants ne semblent pas adapter leurs dictionnaires à la localisation géographique de leur victime. Une autre observation intéressante concerne le fait que les adresses IP qui sont utilisées pour réaliser les attaques par dictionnaire sont complètement différentes de celles qui sont utilisées pour réaliser les intrusions. Ceci semble confirmer l'existence d'une organisation des machines des attaquants qui sont dédiées à des attaques bien spécifiques.

En ce qui concerne les analyses des activités capturées durant les intrusions, il apparaît qu'un bon nombre des intrusions ont été réalisées par des scripts kiddies. Ceci est très différent de ce que l'on peut observer sur certains ports, comme 445, 139 ou autres, pour lesquels des vers ont été conçus de façon à automatiquement infecter la machine et se propager. La seconde observation est que l'identification du pot de miel (pourtant possible) ne semble pas être une priorité des attaquants, puisqu'aucun d'entre eux n'a utilisé les techniques connues permettant de détecter la présence d'un pot de miel.

Bibliographie

[Jacobsson et Ramzan 2008] M. Jacobsson et Z. Ramzan, *Crimeware : Understanding New Attacks and Defenses*, isbn 978-0-321-50195-0, Symantec Press, Addison Wesley Professional, Boston, MA, USA, 2008.

[TeamCymru 2009] Team Cymru, "The darknet project", http ://www.cymru.com/Darknet/index.html, 2009.

[Moore 2002] D. Moore, "Network Telescopes : Observing Small or Distant Security Events", *Proceedings of the 11th Usenix Security Symposium, San Francisco*, CA, USA, 2002.

[Song et al. 2002] D. Song, R. Malan et R. Stone, "A Snapshot of Global Internet Worm Activity", *Proceedings of the FIRST Conference on Computer Security Incident Handling and Response*, Juin, 2002.

[Bailey et al. 2005] M. Bailey, E. Cooke, F. Jahanian, J. Nazario et D. Watson, "The Internet Motion Sensor : A Distributed Blackhole Monitoring System", *Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Fevrier, 2005.

[Pang et al. 2004] R. Pang, V. Yegneswaran, P. Barford, V. Paxson et L. Peterson, "Characteristics of Internet Background Radiation", *Proceedings of the 4th ACM SIGCOMM conference on internet Measurement*, pp. 27-40, ACM Press, 2004.

[Spitzner 2002] , L. Spitzner, *Honeypots : Tracking Hackers*, Paperback, isbn 0321108957, Addison-Wesley Professional, http ://www.amazon.ca/exec/obidos/redirect ?tag=citeulike04-20&path=ASIN/0321108957, 2002.

[Bakos 2002] G. Bakos, *Tiny Honeypot- resource consumption for the good guys*, http ://www.alpinista.org/thp, 2002.

[Provos 2004] N. Provos, "A Virtual Honeypot Framework", *Proceedings of the 13th Usenix Security Symposium*, San Diego, CA, 2004.

[Baecher et al. 2006] P. Baecher, M. Koetter, T. Holz, M. Dornseif et F. Freiling, "The Nepenthes Platform : An Efficient Approach to Collect Malware", *Proceedings of the 9th International Symposium On Recent Advances In Intrusion Detection, RAID06*, Hambourg, Allemagne, Septembre 20-22, 2006, Lecture Notes in Computer Science 4219, Springer, 2006.

[Pouget 2006] F. Pouget, "Distributed System of Honeypot Sensors : Discrimination and Correlative Analysis of Attack Processes", *Thèse de Doctorat*, Institut Eurecom, 2006.

[Dike 2006] J. Dike, *User Mode Linux*, 1st edition, 352 pages, Prentice Hall Ptr, 2006.

[Bellard 2005] , F. Bellard, "QEMU : A Fast and Portable Dynamic Translator", *Proceedings of the USENIX Annual Technical Conference*, FREE Track, Anaheim, CA, USA, pp. 41-45, Fevrier,2005.

[Provos et Holz 2007] N. Provos et T. Holz, *Virtual Honeypots : From Botnet Tracking to Intrusion Detection*, isbn 13 : 978-0-321-33632-3, 2007, Addison Wesley.

[Honeywall 2005] *Know Your Enemy : Honeywall CDROM Roo*, Honeynet Project, www.honeynet.org, 2005

[Portokalidis et al. 2006] G. Portokalidis, A. Slowinska et H. Bos, "Argos : An Emulator for Fingerprinting Zero-Day Attacks", *Proceedings of the 2006 EuroSys Conference*, ACM SIGOPS Operating System Review, 40(4), 2006, pp. 15-27.

[Zhuge et al. 2007] J. Zhuge, T. Holz, X. Han, C. Song et W. Zou, "Collecting Autonomous Spreading Malware Using High-Interaction Honeypots", *Proceedings of the 9th International Conference on Information and Communications Security (ICICS'07)*, Zhengzhou, Chine, Decembre, 2007, pp. 438-451.

[Vrable et al. 2005] , M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. Snoeren, G. M. Voekler et S. Savage, "Scalability, Fidelity, and Containment in the Potemkin Virtual Honeyfarm", *ACM SIGOPS Operating Systems review*, 39(5), pp. 148-162, 2005.

[Alata et al. 2006] E. Alata, V. Nicomette, M. Kaâniche, M. Dacier et M. Herrb, "Lessons Learned from the Deployment of a High-interaction Honeypot", *Proceedings of the 6th European Dependable Computing Conference (EDCC'06)*, pp. 39-44, Coimbra, Portugal, 2006.

[Dressler et al. 2007] F. Dressler and W. Jaegers et R. German, "Flow-based Worm Detection using Correlated Honeypot Logs", *15. Gi/Itg Fachtagung Kommunikation in Verteilten Systemen (KiVS 2007)*, Torsten Braun and Georg Carle and Burkhard Stiller editor, Bern, Switzerland, iVde, pp. 181-186, 2007, www7.informatik.uni-erlangen.de/ dressler/publications/-kivs2007b.pdf.

[Rajad et al. 2006] M. A. Rajad, J. Zarfoss, F. Monrose et A. Terzis, "A Multifaceted Approach to Understanding the Botnet Phenomenon", *Proceedings of the Internet Measurement Conference*, pp. 41-52, 2006.

[Zou et Cunningham 2006] C. C. Zou et R. Cunningham, "Honeypot-Aware Advanced Botnet Construction and Maintenance", *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, pp. 199-208, 2006.

[Fsecure 2006] F-Secure, *Malware information page : Allaple.A*, www.f-secure.com/v-descs/allaple_a.shtml, Decembre, 2006.

[Alberdi et al. 2005] I. Alberdi, J. Gabès et E. Le Jamtel, "UberLogger : un observatoire niveau noyau pour la lutte informatique défensive", *Proceedings du Symposium sur la Sécurité des Technologies de l'Information et des Communications SSTIC*, 2005.

[Williams 1991] R. Williams, "An Extremely Fast Ziv-Lempel Data Compression Algorithm" *Proceedings of the Data Compression Conference*, 1991, pp. 362-371.

[Alata 2007] Eric Alata, "Observation, caractérisation et modélisation de processus d'attaques sur Internet", *Thèse de doctorat*, LAAS-CNRS, 2007, Toulouse.

[Pouget et al. 2004] F. Pouget, M. Dacier et V.H. Pham, "Understanding Threats : A Prerequisite to Enhance Survivability of Computing Systems", *International Journal of Critical Infrastructures*, 4(1-2), pp. 153-171, 2008..

[Panjwani et al. 2005] S. Panjwani, S. Tan, K. Jarrin et M. Cukier, "An Experimental Evaluation to Determine if Port Scans are Precursors to an Attack", *Proceedings of the International Symposium on Dependable Systems and Networks (DSN-2005)*, pp. 602-611, Yokohama, Japon, 28 Juin-1er Juillet, 2005.

[Alata et al. 2008] E. Alata, M. Kaâniche et V. Nicomette, "Etude expérimentale d'attaques par dictionnaire", *Proceedings de la 3ème Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR/SSI'2008)*, pp. 301-315, Loctudy, France, 13-17 Octobre, 2008.

[Cert Mremap 2004] Us-Cert, "Linux kernel mremap(2) system call does not propertly check return value from do_munmap() function", www.kb.cert.org/vuls/id/981222, 2004.

[Cert Dobrk 2003] Us-Cert, "Linux kernel do_brk() function contains integer overflow", www.kb.cert.org/vuls/id/301156, 2003.

Chapitre 6

Travaux de recherche : plateforme expérimentale pour l'exécution contrôlée de maliciels

6.1 Problématique

Alors que l'Internet est en train de devenir un réseau multi-services pour des applications aux besoins divers, les attaques de dénis de service distribués (ou DDOS pour *Distributed Denial of Service*) représentent une menace d'importance. En effet, garantir la qualité de service est devenu aujourd'hui un des leitmotivs de l'Internet qui en plus de transporter des fichiers, transporte également de la voix et de la vidéo (pour ne citer que ces types de média) ayant des contraintes temporelles fortes. Ainsi, il suffit qu'une attaque ralentisse un peu le réseau pour que la qualité du service ne soit plus suffisante, et que, par conséquent, le service ne soit pas rendu (et donc pas facturable).

Ces types d'attaque sont extrêmement fréquents. L'exemple le plus célèbre est certainement l'attaque lancée le 17 Octobre 2002 contre les 13 serveurs DNS racines de l'Internet. Cette attaque avait réussi à rendre 7 de ces serveurs indisponibles, et une partie non négligeable de l'Internet s'est vue privée de ce service durant une période significative, engendrant des délais péniblement longs pour les utilisateurs. De façon générale, More *et al.* [More et al. 2001] a dénombré plus de 12000 attaques ciblant 5000 hôtes différents durant une période de 3 semaines en 2001. Selon Symantec ([Symantec 2006]), de janvier à juin 2006 il y a eu une moyenne de 66000 machines infectées qui ont lancé 6110 attaques par jour.

Protéger les réseaux contre ce type d'attaque est une des problématiques stratégiques principales à résoudre. Les solutions courantes de protection se basent généralement sur l'utilisation des

systèmes de détection d'intrusion. L'inconvénient de ces approches est d'être réactives et donc de ne fonctionner que lorsque les attaques ont lieu [Zhang et Parashar 2005]. Les systèmes de défense de l'Internet doivent donc évoluer de façon à être proactifs, c'est-à-dire d'être capables d'empêcher les attaques de se produire ou de limiter leurs effets au plus tôt.

C'est dans ce cadre que se situent les travaux présentés ici. Notre première contribution est d'avoir conçu et développé une plateforme expérimentale qui utilise la technologie des « pots de miel » pour télécharger des maliciels, et une infrastructure permettant l'observation et l'analyse du comportement des maliciels collectés (voir la section 6.2.1 pour la description du principe de la méthode et la section 6.2.2 pour son implémentation). Nous nous sommes essentiellement focalisés sur des maliciels qui pilotent des *botnets*[1] [Freiling et al. 2005] (nous reviendrons sur la définition du terme *botnet* dans la section suivante). Si nous sommes capables d'infiltrer des *botnets* et d'analyser au plus tôt les maliciels qui sont utilisés pour lancer des attaques massives de déni de service par exemple, nous pouvons être proactifs dans la défense des réseaux.

Ce genre de plateforme est confronté à des problèmes de légalité évidents concernant les risques de pollution des systèmes d'information d'autrui. En effet, si nous contrôlons mal l'exécution de maliciels au sein de notre laboratoire, il est possible que ces derniers lancent des attaques sur des machines extérieures, ce qui rendrait notre laboratoire responsable d'actes strictement interdits par la loi française. Cependant, afin de comprendre en détail comment fonctionnent les maliciels collectés, nous avons besoin de les exécuter le plus complètement possible. Il nous faut donc confiner les effets de bord de ces exécutions et notamment les attaques éventuelles qu'elles peuvent engendrer à l'extérieur. Notre seconde contribution est donc de proposer une plateforme qui permet de fonctionner dans un environnement restrictif, la principale contrainte étant d'interagir avec l'extérieur en réduisant au maximum le risque d'envoyer du trafic malveillant vers l'Internet. Pour cela, nous avons développé un nouveau filtre de paquets, son langage et son compilateur associé. Ce filtre est discuté dans la section 6.3.

6.2 Plateforme d'exécution et d'analyse de maliciels

Nous nous sommes focalisés dans un premier temps sur l'analyse des maliciels qui pilotent les *botnets*, parce que les *botnets* constituent probablement actuellement la menace la plus sérieuse sur le réseau Internet. Les attaques massives menées par ces milliers de machines corrompues peuvent être particulièrement efficaces.

[1] L'étude [Mashevsky 2006] affirme que la location de `botnets` (utilisé en particulier pour le lancement de DDOS ou de spam) est une activité très rentable, et selon leurs estimations, les profits engendrés par ces pirates seraient équivalents à ceux de l'industrie des logiciels antivirus.

6.2.1 Méthodologie et conception

Un *botnet* est un réseau de machines compromises [Holz 2005]. Ce réseau est en général contrôlé par une ou plusieurs personnes et est utilisé comme force de frappe pour lancer de façon massive des spams ou des attaques de déni de service par exemple. Chaque machine compromise est appelée *bot* ou zombie. Le maître du *botnet* envoie des ordres aux machines qui composent son *botnet* à l'aide d'un canal de communication en général appelé `Command & Conquer` (très souvent implémenté sous forme de communications `irc`).

Pour détecter et infiltrer les *botnet*s, on peut distinguer trois sources d'information :

1. L'observation du canal `Command & Conquer`, (C&C).
2. L'observation du trafic malveillant envoyé par les *bot*s (inondation (*flood*), spam, informations confidentielles récupérées sur les machines, etc).
3. La capture et l'analyse du maliciel qui corrompt un ordinateur et le transforme en *bot*.

Nos travaux visent à fournir des outils permettant de construire des mécanismes de défense pro-actifs. Il nous faut donc essayer de détecter au plus tôt l'occurrence d'attaques imminentes. La meilleure façon d'être proactif est de capturer et d'analyser les maliciels qui corrompent un ordinateur et les transforment en *bot*. Nous avons donc choisi de nous concentrer sur cette dernière solution.

Nous avons dans un premier temps mis en place une plateforme de collecte de tels maliciels. Nous ne détaillons pas cette plateforme ici. Brièvement, nous avons notamment utilisé des pots de miel et mis en place un système de téléchargement automatique de maliciels sur les réseaux `p2p` et `SMTP`. Sur `SMTP`, les mails temporaires offerts par les services tels que `pookmail.com` sont pour l'instant ceux qui nous ont donné le plus de binaires malveillants. Les résultats préliminaires sur les réseaux `p2p` sont très encourageants. Une simple requête de téléchargement d'exécutables sur le réseau à l'aide du logiciel *Limewire*, nous a permis de télécharger une dizaine de binaires malveillants différents.

L'objectif est ensuite d'exécuter les maliciels collectés afin d'en comprendre le fonctionnement. Pour cela, puisque nous ne pouvons pas légalement les exécuter sans restriction, nous avons créé un environnement de simulation. La qualité d'émulation des agissements de ce type de maliciel dépend du niveau d'interaction du système. En développant un client IRC particulier pour observer un *botnet* comme l'ont fait [Freiling et al. 2005], le niveau d'interaction obtenu est inférieur à celui que l'on peut espérer en exécutant ce maliciel. Un niveau d'interaction égal à celui des *bots* (c'est-à-dire une exécution sans contrôle) pose des problèmes quant à la maîtrise d'éventuelles attaques et au respect de la loi. Nous avons donc choisi de limiter ces effets de bord, en contrôlant l'exécution, mais en faisant en sorte de conserver le plus haut niveau d'interaction possible.

Les modifications que peut engendrer un tel maliciel sur un système d'exploitation peuvent être

conséquentes et influencer le comportement des processus s'exécutant a posteriori, en particulier celui d'autres maliciels. Nous pensons donc que le système dans lequel ce dernier va être exécuté doit être « propre », et avons choisi de n'exécuter qu'un seul maliciel à la fois, puis de réinitialiser le système.

La deuxième contrainte implique la maîtrise des communications sortantes. Le trafic du canal C&C d'un *bot* n'est pas en lui-même un danger pour le système d'information : il ne s'attaque en général aucunement au système sur lequel il s'exécute, il utilise simplement des ressources réseaux. Nous essayons donc de fournir un niveau d'interaction aussi haut que possible sur l'émulation de ces communications : laisser le maliciel communiquer avec ce canal lorsqu'il est opérationnel, et le simuler sinon[2]. Le trafic malveillant potentiellement émis par le *bot*[3] étant quant à lui illégal, nous nous contentons de simuler ce type d'interaction.

Il reste à savoir comment identifier les flux et simuler certaines interactions. Nous utilisons pour cela une approche itérative en redirigeant les flux vers des machines que nous maîtrisons. Lorsque nous identifions un nouveau flux, nous le redirigeons dans un premier temps vers une machine de notre réseau local qui simule le service demandé par ce flux. Nous observons et analysons ensuite ce flux pour décider de l'autoriser ou non lors de la prochaine exécution du maliciel. Nous relançons alors le maliciel après reconfiguration de la plateforme afin d'aller plus loin dans son exécution.

6.2.2 Implémentation

La grande majorité des systèmes installés sur les ordinateurs interconnectés à l'Internet étant `Windows XP`, nous avons choisi de baser notre environnement d'émulation sur ce système. Pour obtenir un système propre à chaque exécution, le logiciel de simulation d'architecture x86 `VmWare` a été utilisé. Ce logiciel permet de sauvegarder l'état courant de la machine virtuelle et d'y revenir quand bon nous semble. Pour avoir un système propre à chaque exécution, il nous suffit dès lors de mémoriser un état propre, et d'y revenir lors d'une nouvelle exécution.

Nous avons choisi le système `GNU/Linux` comme système hôte, essentiellement pour son pare-feu `netfilter` [Netfilter] et en particulier pour l'API de programmation offerte par la librairie `libipq` qui permet de déléguer certains traitements à l'espace utilisateur.

Le système global de simulation est schématisé sur la figure 6.1.

Il comprend :

- Une machine qui exécute les logiciels malveillants.
- Un simulateur de services ; cette machine est capable de simuler à la demande les services auxquels fait appel le logiciel malveillant : DNS, HTTP, FTP, etc. Lorsque celui-ci exécute

[2]Par exemple dans le cas IRC : url périmée, serveur indisponible...
[3]DoS, tentatives de propagation, spam.

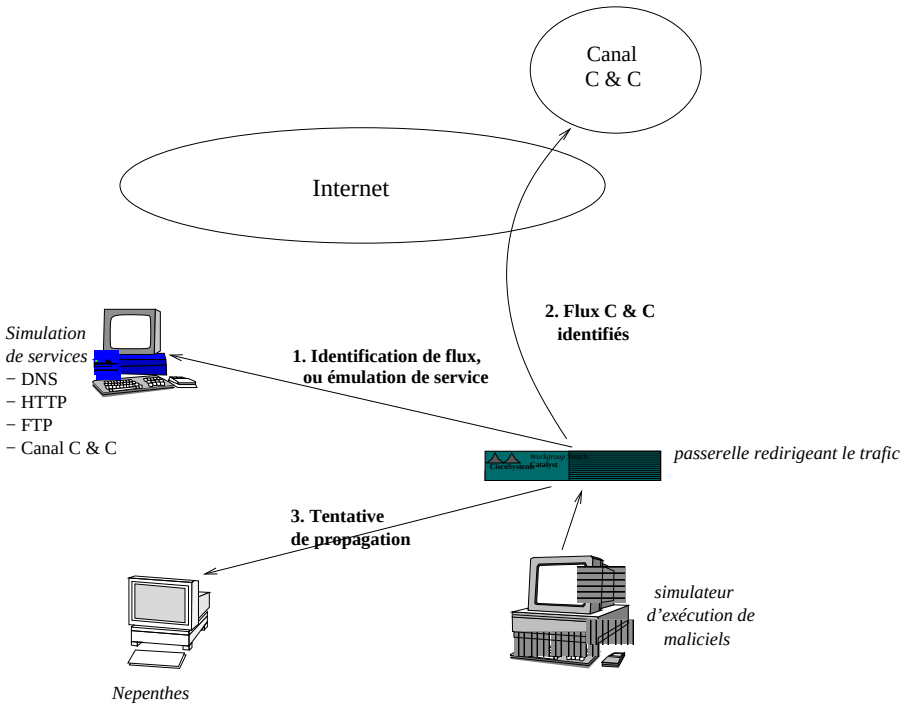


FIG. 6.1 – Environnement d'exécution

des requêtes sur des ports bien identifiés et associés à ces services, les flux sont redirigés vers ce simulateur pour tenter de poursuivre l'exécution.
- Pour les services Windows (Netbios, SMB) : l'attaque est redirigée vers une machine exécutant un pot de miel tel que *nepenthes* [Beacher et al. 2006], dont le but est de simuler une exploitation de faille connue, ou en d'autres termes d'émuler le succès d'une tentative de propagation.
- Pour tout autre service au port inconnu, les flux sont redirigés vers des serveurs TCP ou UDP, qui ne font que lire les octets en entrée.
- Au centre de la plateforme, la passerelle applicative identifie les flux réseaux (en fonction du port destination qui indique le service visé) et les redirige ou les laisse passer directement en fonction du service invoqué.

6.2.3 Un exemple d'exécution

Pour illustrer le fonctionnement de ce système, l'exemple suivant - correspondant au cas où le maliciel exécuté a pour objectif de scanner le réseau - détaille toutes les étapes de la simulation. Ce comportement qui était théorique au début a été confirmé par la suite au cours de la mise en service de la plateforme.

Dans un premier temps, la passerelle bloque tous les flux en sortie, mis à part les requêtes DNS qui sont redirigées vers un serveur DNS que nous maîtrisons. Le maliciel envoie alors une requête DNS pour obtenir l'adresse IP de `URL1`. Supposons que `URL1` existe[4]. Une fois cette adresse obtenue, le maliciel demande l'ouverture d'une connexion TCP (envoi d'un paquet TCP SYN) vers le port `p1` de la machine hébergeant `URL1`. Notre système va alors rediriger cette requête vers une de nos machines locales (flux 1 de la figure 6.1). Cette machine va accepter l'ouverture de la connexion TCP. Dès lors, comme la plupart du temps le canal C&C du maliciel est de type IRC, les premières requêtes envoyées après l'établissement de la connexion sur le canal IRC sont du type :

```
USER user1
NICK nick1
```

Cela nous renseigne également sur la localisation du canal C&C du maliciel qui se trouve à (`URL1,port1`). Nous pouvons alors relancer le maliciel en laissant cette fois passer le flux à destination de (`URL1,port1`) (flux 2 de la figure 6.1). Grâce à ces échanges sur le canal IRC C&C, le maliciel reçoit des instructions concernant la tâche qu'il doit effectuer : le scan d'autres machines de l'Internet dans notre cas d'étude. Le maliciel envoie donc des paquets SYN sur des adresses IP croissantes vers un port donné - le port 445 par exemple. Notre système, qui observe les communications, en déduit qu'il s'agit d'un scan réseau. Le maliciel est alors relancé après reconfiguration du module de redirection de façon à ce qu'une des adresses du scan soit redirigée vers une instance de *nepenthes*. On simule ainsi la réussite de propagation du maliciel (flux 3 de la figure 6.1).

6.3 Un nouveau filtre de paquets permettant l'exécution contrôlée de maliciels

6.3.1 Motivation et méthodologie

Dans l'architecture présentée dans la partie précédente, l'analyse des différents flux sur le réseau de façon à identifier s'il s'agit de protocoles connus (DNS par exemple), du canal C&C ou de propagation d'attaques, est faite manuellement après redirection de ces flux. En effet, dans la plateforme que nous avons présentée dans la partie précédente, la redirection des flux se fait sur identification du port destination et donc du service ciblé. Le flux redirigé est ensuite capturé et analysé manuellement. Le simulateur du service ne fait aucune analyse, il se contente de répondre aux requêtes dans la mesure du possible. De même, la passerelle ne fait que rediriger le flux après identification du service ciblé. L'analyse manuelle permet donc de déterminer la nature du flux

[4]Si `URL1` est périmée, nous changeons la configuration de notre serveur DNS pour qu'il désigne une de nos machines comme ayant ce nom.

et en fonction de cette nature, de relancer l'expérimentation avec éventuellement des modifications dans la plateforme pour que l'exécution du maliciel atteigne une phase supplémentaire comparé à l'exécution précédente (notre but est d'observer le plus en profondeur possible le comportement du maliciel). Cette analyse manuelle détermine aussi si le flux correspond à une tentative d'attaque ou non.

L'analyse manuelle est donc contraignante et surtout elle devient impossible si les flux de données analysés correspondent à des protocoles très complexes. C'est pourquoi nous avons tenté d'automatiser le mécanisme d'identification des flux de façon à automatiquement reconfigurer la plateforme pour la prochaine exécution du maliciel. L'identification doit également être capable de détecter si le flux contient une attaque ou non, de façon à ne pas laisser notre plateforme lancer des attaques sur Internet. Pour cela, nous avons conçu et implémenté un nouveau filtre de paquets qui :

- permet de rediriger des flux mais aussi de les analyser en profondeur de façon à déterminer si ces flux sont inoffensifs ou pas ;
- propose une configuration suffisamment souple (au travers de règles) pour pouvoir analyser n'importe quel type de compositions de protocoles utilisés.

Pourquoi avoir implémenté un nouveau filtre alors que des outils (IDS, IPS) existent déjà ? Les systèmes de détection/prévention d'intrusion et les filtres de paquets actuels se contentent de proposer à un utilisateur la configuration d'un ensemble de compositions protocolaires prédéfinies. Par exemple, la documentation du système de détection d'intrusion Snort[5], nous montre qu'un seul niveau d'encapsulation GRE (*Generic Routing Encapsulation*) est supporté. Une communication du type `Eth/IPv4/GRE/IPv4/GRE/IPv4/TCP/Pile_Applicative` ne peut donc pas être analysée par Snort. Dès lors, si on se contente, à l'instar de Snort et de ses semblables [Paxson 1998] ou des solutions commerciales [Juniper, Cisco, McAfee], de proposer la configuration d'un ensemble de compositions de protocoles prédéfinies, les développeurs de logiciels malveillants pourront assez aisément franchir ces barrières. En supposant que la composition IPv4/TCP/HTTP/A/B/C soit la plus profonde des compositions configurables, il leur suffit par exemple de développer un protocole IPv4/TCP/HTTP/A/B/C/M. Une telle composition leur permet alors de réaliser leurs agissements malveillants, tout en étant interopérables avec le pare-feu. Les canaux C&C des *bots* tels que [Holz et al. 2008, Chiang et Lloyd 2007] implementés au dessus de HTTP ou du protocole pair-à-pair décentralisé Kademlia illustrent ce problème, de même que les encapsulations ou *shellcodes* dans le protocole DNS [Miller 2008, Dns2tcp, Nussbaum et Richard 2008].

Les deux problèmes majeurs liés au développement de ce logiciel ont donc été 1) le développement d'analyseurs protocolaires et leur intégration dans le filtre et 2) la composition de ces analyseurs. Nous présentons rapidement dans la section suivante un certain nombre d'analyseurs que nous avons développés ainsi que des exemples de compositions de ces analyseurs.

[5] `http://www.snort.org/docs/snorthtmanuals/htmanual2832/node35.html`

6.3.2 Configuration

Notre filtre de paquets est basé sur un langage et un compilateur associé qui permet une expression de compositions de protocoles totalement générique. Il est possible à l'administrateur de ce filtre de préciser n'importe quel type de composition ; il est simplement limité par l'existence des analyseurs de chaque protocole intervenant dans la composition. De plus, ce filtre a été créé à l'aide d'un formalisme qui nous permet de le représenter sur la forme d'un arbre d'analyseurs. Nous pouvons vérifier grâce à ce formalisme que l'arbre généré par des règles est bien minimum et non ambigü. Cette notion est fondamentale car la gestion de la cohérence des règles n'est en général pas gérée dans les outils que nous connaissons. Ainsi, si l'administrateur entre des règles redondantes, notre compilateur factorise de façon à éliminer cette redondance. Si l'administrateur entre des règles incohérentes, une erreur est retournée.

Nous ne présentons pas ici ce formalisme, plus de précisions peuvent être trouvées dans [Alberdi et al. 2009]. En revanche, nous allons donner dans la suite des exemples d'analyseurs et de règles de notre langage de configuration.

6.3.2.1 Les analyseurs de protocoles

Nous avons développé les analyseurs suivants dans le prototype actuel. Ils constituent simplement un exemple. La conception de notre outil autorise le développement et l'intégration d'un nombre quelconque d'analyseurs.

- $IPv4$: Filtrage sur adresses source et destination. Des options permettent notamment de vérifier la somme de contrôle (*checksum*) IP ou non. On peut imaginer y insérer tout autre type de vérification IP, concernant les options, comme le bourrage (*padding*) par exemple.
- Tcp : Cet analyseur implémente une pile TCP et assure un suivi d'état TCP. Il vérifie donc l'établissement et la fermeture correcte et complète de connexion. Il vérifie également la cohérence des numéros de séquence, des numéros d'acquittements et des tailles de fenêtres. En particulier, l'approche formelle que nous avons adoptée pour le développement de l'analyseur TCP est celle proposée par [Bidder-Senn et al. 2007]. Contrairement à d'autres pare-feux tels que `netfilter` qui écrivent leur version de l'automate du milieu à la main, la méthode décrite dans [Bidder-Senn et al. 2007] permet, après avoir modélisé les extrémités à l'aide d'automates[6], de synthétiser l'automate du milieu qui prend en compte tous les scénarios possibles (perte de paquets, réordonnancement). Plus de précisions peuvent être trouvées dans [Alberdi et al. 2009].
- Udp : Cet analyseur implémente une pile UDP. Il est possible de le paramétrer pour vérifier le checksum UDP.
- $TPort$: Cet analyseur vérifie le port source ou destination d'une connexion TCP.
- $UPport$: Cet analyseur vérifie le port source ou destination d'une connexion UDP.

[6] Machine de Mealy

- *Dns* : Cet analyseur inspecte le protocole DNS. Il est possible de le paramétrer pour n'autoriser que certaines requêtes DNS, comme par exemple les requêtes A (requêtes liées au domaine DNS) ou AAA uniquement (requêtes liées à la résolution du nom d'une machine). C'est notre configuration par défaut.
- *Ftp* : Cet analyseur inspecte le protocole FTP. Il est capable de faire un suivi d'état des deux connexions TCP incluses dans ce protocole, mais aussi d'inspecter plus profondément le protocole en n'autorisant par exemple qu'un sous-ensemble des commandes du protocole et en vérifiant leur réponse.
- *CSend* : Cet analyseur vérifie qu'un client TCP ne fait qu'envoyer des données (il n'en reçoit pas).
- *CReceive* : Cet analyseur vérifie qu'un client TCP ne fait que recevoir des données (il n'en envoie pas).
- *Http* : Cet analyseur fait de l'inspection HTTP et est capable notamment de vérifier la présence de shellcode dans des URI.

L'acceptation ou le refus d'un paquet doit se faire en fonction de l'analyse réalisée par un certain nombre de ces analyseurs, telle que configurée par l'administrateur à l'aide de règles. Ces règles font appel à deux opérateurs et à des identificateurs représentant les analyseurs cités ci-dessus.

- `/` : opérateur de sérialisation
- `|` : opérateur de parallélisme

Chaque analyseur peut être configuré de façon à effectuer des contrôles précis, et ceci grâce à l'utilisation de paramètres. Par exemple, l'analyseur DNS peut être configuré ainsi : `Dns(AAAA_only)`. Cela signifie que l'on demande à l'analyseur de n'autoriser que les requêtes DNS particulières de type AAAA ou A L'administrateur de notre filtre peut donc, grâce à la souplesse de configuration des analyseurs autoriser le DNS tout en éliminant certaines requêtes suspectes. Un autre exemple est le paramètre `tcp_only` qui peut être utilisé pour configurer l'analyseur *Tcp*. En fonction de sa valeur (`true` ou `false`), l'analyseur peut gérer ou non le réassemblage de segments TCP. Nous verrons d'autres exemples dans les règles énoncées dans les paragraphes suivants.

6.3.2.2 Les règles : exemple 1

Soient les règles suivantes :

```
IF(idx=3,addrs=192.168.0.0/24,lo=192.168.0.254),
IF(idx=2,addrs=default,lo=192.168.1.254);;
SF(Ipv4(from=not (192.168.0.254 or 192.168.1.254),
        to=not (192.168.0.254 or 192.168.1.254),check_checksum=yes)/
   [Tcp)/
    TPort(sr_port=80)/
```

```
Http]);;
```

Les 2 premières lignes décrivent simplement les interfaces réseaux du filtre et les adresses IP associées. Les lignes suivantes représentent les datagrammes IP dont on vérifie 1) les adresses source et destination IP ainsi que le checksum IP (`check_checksum=yes`), 2) qu'ils correspondent au protocole `TCP`, port destination 80 (`sr_port=80`), et 3) qu'ils correspondent à des requêtes bien formées HTTP. Le composition des analyseurs est représentée par l'arbre de la figure 6.2. Seul l'opérateur séquentiel a été utilisé ici.

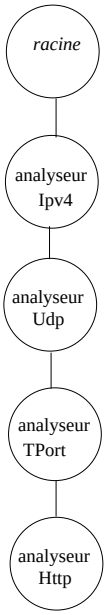


FIG. 6.2 – Exemple d'arbre simple HTTP

Dans l'exemple suivant, nous utilisons les deux opérateurs.

6.3.2.3 Les règles : exemple 2

L'exemple précédent n'est pas très réaliste. En effet, si un administrateur veut autoriser des requêtes HTTP à destination d'un serveur, il faut bien sûr que les requêtes DNS soient également autorisées. On peut dans ce cas obtenir les règles suivantes :

```
IF(idx=3,addrs=192.168.0.0/24,lo=192.168.0.254),
IF(idx=2,addrs=default,lo=192.168.1.254);;
SF(Ipv4(from=not (192.168.0.254 or 192.168.1.254),
```

```
    to=not (192.168.0.254 or 192.168.1.254),check_checksum=yes)/
[Udp(check_checksum=yes)/
 UPort(sr_port=53)/
 Dns] |
[Tcp/
 TPort(sr_port=80)/
 Http]);;
```

On voit bien ici l'utilisation de l'opérateur | qui permet de paralléliser des analyseurs. Dans le cas d'une requête DNS (et donc d'un datagramme UDP), c'est l'analyseur *Udp* qui va être invoqué par le filtre. Dans le cas de requête HTTP, c'est l'analyseur *Tcp* qui sera invoqué. La figure 6.3 représente l'arbre associé à cet exemple.

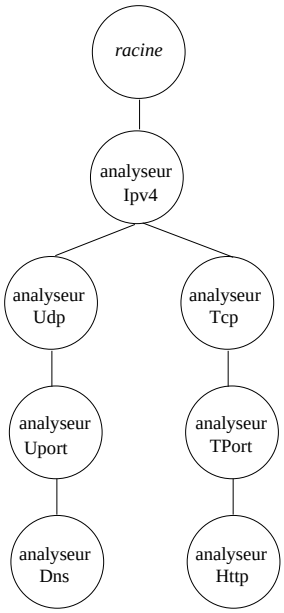


FIG. 6.3 – Exemple d'arbre simple HTTP et DNS

6.3.2.4 Les règles : exemple 3

Ce dernier exemple illustre mieux les possibilités de notre filtre. Nous avons choisi d'étudier le filtrage de canaux cachés dans le protocole DNS [Nussbaum et Richard 2008]. Ces canaux cachés exploitent la liberté sémantique offerte par les requêtes TXT de ce protocole[7] pour encoder des

[7] `http://www.isi.edu/in-notes/rfc1464.txt`

paquets TCP ou IP dans ce format, et ainsi encapsuler des protocoles de communications au-dessus de DNS. Il est ainsi possible d'utiliser ces canaux pour obtenir gratuitement un accès Internet sur des points d'accès wifi payants. En effet, certains points d'accès n'inspectant pas les requêtes DNS entrantes, un client malveillant peut, à l'aide de canaux créés dans un protocole DNS vers un site relai, surfer gratuitement sur le web. D'un autre côté, un client légitime voulant seulement consulter des sites web peut très bien se contenter des requêtes de type A/AAAA (résolutions de noms). Ainsi une inspection du protocole DNS qui n'autorise que les requêtes de type A/AAAA, et leurs réponses, permet d'empêcher l'établissement de ces canaux cachés. Voici, à titre d'exemple, des règles qui nous permettent ce filtrage. Elles ressemblent fortement à celles de l'exemple 2 et l'arbre des analyseurs est identique mais l'analyseur `Dns` est paramétré de façon à ne traiter que certaines requêtes que l'on considère inoffensives, à l'exclusion des autres. Les requêtes de type TXT dont donc filtrées.

```
IF(idx=3,addrs=192.168.0.0/24,lo=192.168.0.254)
IF(idx=2,addrs=default,lo=192.168.1.254);;

SF(Ipv4(check_checksum=yes,from=not (192.168.0.254 or 192.168.1.254),
        to=not (192.168.0.254 or 192.168.1.254))/
   [Tcp/
    TPort(sr_port=80) |
    Udp/
    UPort(sr_port=53)/
    Dns(AAAA_only=yes)
   ]);;
```

6.4 Conclusion

Ces travaux s'inscrivent dans la gestion proactive des techniques de défense des réseaux. Les deux principales contributions de ces travaux concernent la création d'une plateforme d'exécution et d'observation de maliciels destinés à transformer des machines en *bots*. Cette plateforme essaie d'offrir le maximum d'interaction au maliciel exécuté, de façon à pouvoir observer au mieux le comportement du matériel, tout en interdisant les éventuelles attaques lancées par ce maliciel. Au centre de cette plateforme, une passerelle décide des flux qui sont autorisés, interdits ou redirigés. Nous avons conçu un nouveau filtre de paquets qui permet de façon dynamique l'inspection de paquets sur les réseaux traversant cette passerelle. Ce filtre de paquets est nouveau dans le sens où sa configuration totalement générique permet l'analyse de n'importe quelle composition de protocoles réseaux.

Ce travail est en cours. Nous soumettons actuellement ce filtre de paquets à des tests de façon à évaluer plus précisément ses performances.

Bibliographie

[More et al. 2001] D. More, G. M.voelker et Stefan Savage, "Inferring Internet Denial-of-Service Activity", *Proceedings of the 10th USENIX Security Symposium*, Washington, D.C, USA, 2001, pp. 9-22.

[Symantec 2006] Symantec, "Symantec Internet Security threat report", September, 2006, http ://www.symantec.com/enterprise/threatreport/index.jsp.

[Zhang et Parashar 2005] G. Zhang et M. Parashar, "Cooperative Mechanism Against DDoS Attacks", *Proceedings of the International Conference on Security Management (SAM .05)*, Las Vegas, NV, USA, The Applied Software System Laboratory, Department of Electrical and Computer Engineering, Rutgers University, CSREA Press, 2005, pp. 86-96.

[Mashevsky 2006] Y. Mashevsky, Kaspersky Lab, "Les dessous de l'économie souterraine des codes malicieux : chevaux de Troie, virus et malware", VirusList.com, 2006.

[Freiling et al. 2005] F. Freiling, T. Holz et G. Wicherski, "Botnet Tracking : Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks", RWTH Aachen, AIB-2005-07, 2005, http ://pi1.informatik.uni-mannheim.de/publications/show/12.

[Holz 2005] T. Holz, "A Short Visit to the Bot Zoo", *IEEE Security & Privacy Magazine*, Volume 3, pp. 76-79, Mai-Juin 2005.

[Netfilter] `http://www.netfilter.org/`

[Beacher et al. 2006] P. Baecher and M. Koetter and M. Dornseif et F. Freiling, "The Nepenthes Platform : An Efficient Approach to Collect Malware", *Proceedings of the 9 th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2006, pp. 165-184, Springer.

[Paxson 1998] V. Paxson, "Bro : a System for Detecting Network Intruders in Real-time", *Proceedings of the 7th Conference on USENIX Security Symposium*, Volume 7 (San Antonio, Texas, January 26 - 29, 1998),

[Juniper] Juniper, "Safeguarding your Network IPS Overview Demo", `http://www.juniper.net/products_and_services/intrusion_prevention_solutions`.

[Cisco] Cisco, "Ips solutions", `http://www.cisco.com/en/US/netsol/ns785/networking_solutions_package.html`.

[McAfee] macAfe, "Host Intrustion Prevention for server", `http://www.mcafee.com/us/enterprise/products/hostintrusionprevention/hostintrusionpreventionserver`.

[Holz et al. 2008] T. Holz, M. Steiner, F. Dahl, E. Biersack, et F. Freiling, "Measurements and Mitigation of Peer-to-peer-based Botnets : a Case Study on Storm Worm", *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats* (San Francisco, California, April 15 - 15, 2008), F. Monrose, Ed., USENIX Association, Berkeley, CA, 1-9.

[Chiang et Lloyd 2007] K. Chiang, et L. Lloyd, "A Case Study of the Rustock Rootkit and Spam bot", *HotBots'07 : Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, 2007, pp. 10-10, Cambridge, MA, USENIX Association, Berkeley, CA, USA.

[Miller 2008] T. Miller, "Reverse DNS Tunneling Shellcode", Black Hat 2008.

[Dns2tcp] `http://www.hsc.fr/ressources/outils/dns2tcp/`

[Nussbaum et Richard 2008] L. Nussbaum, O. Richard, "Prototype de canal caché dans le DNS", *CFIP'08 : Colloque Francophone sur l'Ingénierie des Protocoles*, Les Arcs, France, 03, 2008, `http://perso.ens-lyon.fr/lucas.nussbaum`.

[Alberdi et al. 2009] I. Alberdi, V. Nicomette et P. Owezarski, "Generic and Composable Statefull Packet Filters", Rapport LAAS 09428, 12p.

[Bidder-Senn et al. 2007] D. Bidder-Senn, G. Basin, G. Caronni, "Midpoints Versus Endpoints : From Protocols to Firewalls", *5th International Confrence on Applied Cryptography and Network Security*, Zhuhai, China, pp. 46-64, 2007, `http://www.springerlink.com/content/p67300458q711043`.

[Alberdi et al. 2009] I. Alberdi, V. Nicomette et P. Owezarski, "Plateforme pour l'exécution contrôlée de logiciels malveillants", Rapport LAAS 09318, 12p.

Chapitre 7

Travaux de recherche : sécurisation de noyaux de système d'exploitation

7.1 Problématique

Les activités malveillantes visant les ordinateurs se multiplient régulièrement et essaient d'exploiter des vulnérabilités qui sont de plus en plus nombreuses en raison de la complexité toujours croissante des logiciels d'aujourd'hui. Ces maliciels ou *malware* peuvent s'attaquer aux applications installées sur le système mais aussi au système d'exploitation lui-même et en particulier, son noyau. Corrompre le noyau d'un système d'exploitation est particulièrement intéressant du point de vue d'un attaquant parce que cela signifie corrompre potentiellement tous les logiciels qui s'exécutent au-dessus de ce noyau. En particulier, les *rootkits en mode noyau* [Lacombe et al. 2008] sont des maliciels qui effectuent ce genre de corruption. Pour cela, ils profitent d'un certain nombre de vulnérabilités des noyaux de système eux-mêmes, présents en particulier dans les pilotes de périphériques[1].

Comme la corruption du noyau d'un système d'exploitation peut potentiellement provoquer la corruption de tous les logiciels s'exécutant au-dessus de ce noyau, ce dernier doit être fortement protégé. Mais, protéger un noyau de façon efficace est particulièrement délicat car il est extrêmement difficile de rendre les mécanismes de protection incontournables. En ce qui concerne le logiciel qui s'exécute dans l'espace de l'utilisateur, il est possible d'implémenter des mécanismes de sécurité efficaces dans la mesure où ces mécanismes sont dans le noyau et s'exécutent donc dans un mode plus privilégié que les entités qu'ils protègent. Pour protéger

[1] En ce qui concerne notamment le noyau Linux, les raisons principales pour cela sont que : (1) le noyau est constitué en majeure partie de ces pilotes ; (2) les règles qui régulent l'intégration des pilotes dans le noyau Linux par exemple, en ce qui concerne la qualité du logiciel, sont moins sévères que celles qui sont appliquées dans les autres parties du noyau.

efficacement un noyau contre des exécutions de code malveillant, il semble donc nécessaire de le faire depuis un mode plus privilégié que le noyau lui-même et de façon à ne pouvoir être contournés (par le noyau lui-même, l'espace de l'utilisateur ou les périphériques).

Nos travaux dans le domaine de la protection des systèmes d'exploitation couvrent deux axes. Dans un premier temps, nous avons analysé une catégorie particulière de maliciels, les *rootkits*, et plus particulièrement les rootkits noyau. Nous proposons donc dans la partie 7.2 un résumé de nos contributions. Dans un second temps, forts de notre connaissance des mécanismes de corruption utilisés par les rootkits au sein des noyaux, nous avons proposé des solutions qui permettent d'assurer efficacement la protection de ces noyaux grâce à l'utilisation de moyens matériels incontournables. Nous présentons nos contributions dans la partie 7.3.

Dans l'ensemble de ces travaux, nous nous sommes focalisés uniquement sur le système d'exploitation Unix et en particulier sur le noyau Linux.

7.2 Rootkits en mode noyau : application sous Linux

7.2.1 Historique des rootkits

Le premier réflexe d'un administrateur lorsqu'un événement éveille son attention sur son système est de consulter ses logs, d'appeler la commande `last` pour vérifier qui s'est connecté en dernier, d'appeler `netstat` pour examiner les connexions réseau, et ainsi de suite. Un pirate, connaissant la réaction de l'administrateur, va tenter de dissimuler sa présence sur le système en remplaçant les commandes usuelles : c'est la première forme des *rootkits*. Ainsi, lorsque l'administrateur en appelle à ses outils, ceux-ci cachent certaines informations sur l'intrus, rassurant par la même l'administrateur peu averti.

Cependant, cette approche n'est pas fiable pour le pirate :
- sur les premiers Unix, il était assez fréquent de recompiler les sources car la bande passante ne permettait pas de télécharger des Giga-octets de binaire, et l'intrus devait donc substituer ses programmes aux originaux compilés à chaque fois ;
- il est aisé d'obtenir la même information par le biais de plusieurs commandes (exemple : lister des fichiers avec `ls`, `find`, `grep -r`, etc), et le risque pour l'attaquant est d'en oublier une, voyant ainsi sa présence révélée ;
- bien souvent, et surtout dans les environnements sécurisés, des *empreintes* des binaires sont calculées à partir de fonctions de hachage, afin de détecter la modification de ces programmes, par exemple lors d'une analyse post-intrusion.

Afin de résoudre en partie ces problèmes, et en particulier les deux premiers, les rootkits ont évolué, cherchant à corrompre un maximum de programmes avec un minimum d'effort. Avec l'apparition des bibliothèques dynamiques, partagées par une majorité de binaires, les intrus ont vu là un bon moyen de régler leurs soucis : modifier une fonction dans une bibliothèque

affecte tous les programmes qui emploient cette bibliothèque. Néanmoins, les problèmes restent les mêmes, dans une moindre mesure.

La démarche de factorisation (modifier moins, corrompre plus) s'est donc naturellement poursuivie vers la dernière ressource partagée par tous les éléments : le noyau. En charge tant de la gestion du matériel que de l'ordonnancement des tâches ou de la gestion de la mémoire, le noyau est le point de passage obligatoire pour tous les éléments du système d'exploitation.

7.2.2 Architecture des rootkits

Après avoir donné un bref historique des rootkits, nous en proposons la définition suivante : un *rootkit* est un ensemble de modifications permettant à un attaquant de maintenir dans le temps un contrôle frauduleux sur un système d'information.

Plusieurs éléments sont caractéristiques d'un rootkit :
- *ensemble de modifications* : une première différence flagrante apparaît ici par rapport aux autres codes malveillants. D'une part, un rootkit est rarement un unique programme, mais est souvent constitué de plusieurs éléments. D'autre part, les multiples éléments d'un rootkit sont rarement des programmes autonomes, mais plutôt des modifications effectuées sur d'autres composants du système (programmes dans l'espace de l'utilisateur, partie du noyau, ou autres). Ce type de modifications nous fait penser à la notion de *parasitage*, associée aux codes malveillants qui propagent leur charge utile en fonction de leurs cibles.
- *maintien dans le temps* : les autres codes malveillants n'ont pas réellement de relation au temps, sauf dans quelques cas pour les bombes logiques. Dans le cas d'un rootkit, un attaquant a pris le contrôle du système pour y effectuer certaines opérations (vol d'information, rebond, déni de service, etc.) et doit fiabiliser son accès tant que son objectif n'est pas atteint.
- *contrôle frauduleux sur un système d'information* : cela signifie que l'attaquant dispose des privilèges dont il a besoin pour effectuer ses opérations alors qu'il ne devrait pas être en mesure d'utiliser le système. La plupart du temps, l'attaquant cherche à maintenir son contrôle à l'insu des utilisateurs légitimes du système, mais ce n'est pas une nécessité. Par ailleurs, cela suppose des interactions, et donc une communication, entre le système et le détenteur du rootkit.

La première étape pour l'attaquant est d'installer le rootkit puis de pérenniser l'accès au système compromis, d'où le besoin d'un module de protection. Une fois en place, l'attaquant utilise le rootkit comme intermédiaire avec le système. Il contient donc un module central servant d'interface entre le système et l'attaquant (c'est-à-dire une porte dérobée ou *backdoor*). L'intrus effectue les opérations nécessaires à l'accomplissement de ses objectifs, opérations qui caractérisent alors les services rendus par le rootkit. Un rootkit est donc composé des entités suivantes :

- **Un injecteur.**
  Il s'agit du mécanisme que l'attaquant emploie afin d'insérer le rootkit dans le système (infection de modules noyau, injection de code via `/dev/kmem`, etc.). En effet, que l'attaquant parvienne à rentrer sur le système en exploitant une faille logicielle ou un mot de passe faible,

il doit ensuite y placer son rootkit pour s'installer véritablement sur le système. Qu'il s'agisse d'un rootkit dans l'espace de l'utilisateur ou noyau, il a toujours besoin d'accéder puis de modifier certaines structures du système, une seule et unique fois.

- **Un module de protection.**
  Son objectif est de rendre le rootkit "tenace" sur le système tout le temps nécessaire à l'attaque. Plusieurs stratégies sont envisageables et combinables. Citons à titre d'exemples :
  - Dissimuler le rootkit.
  - Rendre résistant le rootkit (lui permettre de résister aux tentatives de suppression).
  - Rendre persistant le rootkit (faire en sorte qu'il résiste au redémarrage de la machine).
  - Camoufler l'activité de l'attaquant.
- **La porte dérobée.**
  La porte dérobée permet d'une part à l'intrus de conserver le contrôle (niveau de contrôle dépendant de ce qu'il cherche à faire sur le système[2]), et d'autre part d'accéder aux services. La porte dérobée est le point central du rootkit, elle joue le rôle d'interface entre l'extérieur (l'intrus) et les services fournis par le rootkit.
- **Les services.**
  Un rootkit fournit plusieurs services grâce auxquels l'attaquant effectue les opérations dont il a besoin sur le système compromis. Nous distinguons deux catégories de services :
  - Les services passifs ou d'espionnage (par exemple un *keylogger*).
  - Les services actifs (par exemple, dénis de service, destruction d'information, etc.).
- **Un module de communication.**
  Nous distinguons trois phases pendant lesquelles les communications avec le rootkit jouent un rôle principal :
  - Lors de l'intrusion à proprement parler, qui permet à l'intrus de compromettre la sécurité du système cible.
  - Lors du transfert et de l'installation du rootkit.
  - Enfin, lors de l'utilisation du système compromis, l'intrus envoie ses instructions et récupère éventuellement les résultats.

7.2.3 Vers l'évaluation d'un rootkit

L'emploi d'un rootkit par un attaquant traduit sa volonté de conserver le contrôle d'un système dans le temps une fois qu'il l'a compromis. Quelque soit la nature de l'attaquant (opportuniste, hacktiviste, mafieux, etc.), nous identifions en général quatre objectifs :

1. récupérer des informations contenues dans ou transitant par le système compromis ;
2. bloquer l'accès au système, c'est-à-dire provoquer un *déni de service* (DoS) (la notion de système incluant ici le réseau) ;

[2]Dans certains cas, l'attaquant n'a pas besoin du maximum de privilèges sur le système, par exemple il cherche à espionner un utilisateur donné.

3. prendre le contrôle du système afin de l'utiliser comme outil d'espionnage, pour le faire participer à un DDoS, ou pour le transformer en serveur de contenus illégaux ;
4. rebondir vers d'autres systèmes, auquel cas il sert uniquement de point de transfert vers une autre cible.

Dans la majorité des cas, l'intrus cherche à dissimuler sa présence aux utilisateurs légitimes. Il vient donc naturellement à l'esprit qu'un des critères essentiels caractérisant un rootkit concerne son degré d'*invisibilité* (notion sur laquelle nous revenons ci-dessous). Néanmoins, il est tout à fait envisageable que l'attaquant ne souhaite pas particulièrement dissimuler son rootkit mais qu'en revanche, il fasse en sorte que le retirer du système soit très difficilement réalisable sans mettre en péril le système lui-même. Cette notion fait donc intervenir un autre critère, concernant cette fois la *robustesse* du rootkit. Enfin, quelque soit le niveau d'invisibilité ou de robustesse associé au rootkit, ce dernier a pour but de modifier le système sur lequel il se trouve pour permettre à l'attaquant de maintenir de façon durable le contrôle sur ce système. Un troisième critère permettant d'exprimer l'importance de la modification faite par un rootkit sur le système compromis nous semble également pertinent pour évaluer un rootkit.

Nous avons donc proposé trois critères pour évaluer un rootkit :

L'*invisibilité* d'un rootkit exprime la difficulté avec laquelle l'utilisateur du système peut détecter le rootkit lui-même ainsi que les activités malveillantes exécutées à l'aide de ce rootkit.

La *robustesse* d'un rootkit exprime la difficulté de retirer un rootkit d'un système sur lequel il est installé.

Le *pouvoir d'infestation* d'un rootkit exprime le degré d'ingérence du rootkit dans le système, c'est-à-dire la quantité d'éléments qui sont affectés par le rootkit sur le système compromis. Ce critère est d'ailleurs proche de la notion de *virulence* qui est définie par É. Filiol [Filiol 2005] et qui s'applique habituellement aux virus.

Pour évaluer un rootkit à l'aide de ces critères, il faut bien sûr leur associer une mesure. Nous ne donnerons pas de détails ici concernant ces mesures mais nous invitons le lecteur à consulter [Lacombe et al. 2007] dans lequel nous avons proposé quelques pistes.

7.2.4 Un exemple de construction d'un rootkit "furtif"

En collaboration avec Frédéric Raynal (EADS puis Sogeti) qui a été à l'origine avec Eric Lacombe de ces travaux, nous avons conçu un rootkit en nous focalisant particulièrement sur la propriété d'invisibilité. Nous ne présentons pas en détail ce rootkit, une description complète peut être trouvée dans [Lacombe et al. 2007]. Nous donnons ici simplement un aperçu des techniques utilisées.

L'approche consiste à ne corrompre qu'un seul processus/*thread* dans le système. L'originalité

réside donc ici dans le fait que les autres processus s'exécutant sur le système ne voient aucune modification de leur environnement.

L'approche utilise l'appel système 0 de façon détournée. Il est normalement employé par le noyau pour relancer certains appels système interrompus avec de nouveaux paramètres de façon transparente pour l'espace de l'utilisateur. Ce cas se présente par exemple lorsqu'un processus endormi (via `sys_nanosleep`) doit être réveillé afin d'exécuter un gestionnaire de signal. Après avoir traité le signal, le processus doit être endormi de nouveau (si nécessaire) durant une période plus courte : la durée initiale moins la durée d'exécution du gestionnaire. Pour cela, la fonction `sys_nanosleep` est relancée via l'appel système 0 avec cette nouvelle durée. Étant donné que l'appel système à relancer est propre à chaque processus (ou *thread*) et peut varier en fonction du temps, une référence à cet appel est conservée pour chaque *thread*. Ainsi, au moment où un appel système (parmi ceux qui peuvent nécessiter un redémarrage) est exécuté, son adresse est stockée temporairement dans le descripteur du processus l'ayant appelé. Plus précisément, elle se trouve dans une structure `thread_info` liée au descripteur (fig. 7.1).

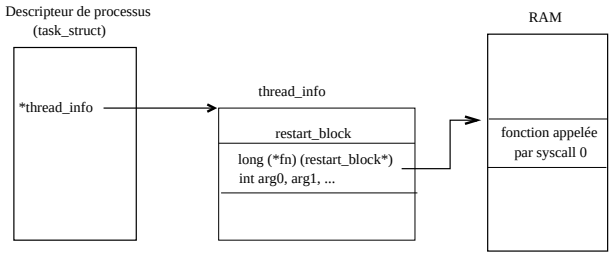


FIG. 7.1 – Le descripteur de processus et l'appel système 0

La technique de détournement consiste à modifier cette adresse. Ainsi, nous pouvons exécuter en *ring 0* n'importe quelle fonction de l'espace du noyau (ou code arbitraire préalablement injecté dans l'espace du noyau) depuis l'espace de l'utilisateur. Et cela est visible uniquement depuis le processus modifié. Ainsi nous sommes plus discrets que les approches courantes affectant le système globalement.

Des méthodes peuvent également être proposées pour dissimuler l'activité de l'attaquant réalisée grâce à ce rootkit. Brièvement, ces méthodes consistent en :

- cacher un processus chargé d'exécuter une activité malveillante ;
- cacher la descendance complète d'un processus destiné à réaliser des activités malveillantes ;
- effectuer des activités malveillantes sans même créer de nouveau processus mais en parasitant des *threads* du noyau déjà existants sur le système ; le principe est simplement de voler des cycles d'exécution à des *threads* du noyau de façon à leur faire exécuter du code malveillant pendant ces cycles et à leur faire exécuter le code légitime prévu pendant les autres cycles.

7.3 Protection de noyau de systèmes d'exploitation à l'aide de mécanismes de virtualisation matérielle

Après avoir présenté les rootkits en mode noyau et la façon dont ils opèrent, nous sommes arrivés au constat qu'il est particulièrement difficile de s'en protéger en utilisant les protections habituelles. En effet, la quasi totalité des protections de noyaux de systèmes est à l'heure actuelle effectuée par du logiciel s'exécutant au même niveau que le code du noyau lui-même. Un attaquant qui a donc réussi à installer un rootkit en mode noyau peut donc exécuter du logiciel qui s'exécute dans le même niveau de privilèges que le logiciel de protection. Par conséquent, il a la possibilité de détourner les mécanismes de protection mis en œuvre.

Aussi, nous avons décidé de nous focaliser sur des mécanismes matériels pour améliorer la protection des systèmes d'exploitation. En effet, de tels mécanismes matériels permettant à du logiciel de s'exécuter dans un niveau de privilège supérieur au code du noyau existent aujourd'hui : c'est la technologie de virtualisation matérielle, disponible sur les processeurs Intel et ADM notamment. Les mécanismes de protection implémentés par du logiciel plus privilégié que le noyau lui-même sont donc a priori incontournables, même par du logiciel s'exécutant dans le noyau, et peuvent donc protéger efficacement le noyau de systèmes d'actions malveillantes.

Nous allons brièvement présenter dans cette section cette technologie et donner des exemples simples montrant comment on peut empêcher certaines actions malveillantes de s'exécuter au sein même du noyau.

7.3.1 Support matériel pour la virtualisation – le cas d'Intel VT

Nous nous sommes intéressés à la protection de noyaux à l'aide de la technologie de virtualisation suite à la publication du rootkit baptisé *bluepill* [Rutkowska 2006], qui lui-même s'installe en tant qu'hyperviseur léger sur une CPU possédant la technologie de virtualisation matérielle et installe un système d'exploitation Windows dans une machine virtuelle. En profitant de cette technologie, ce rootkit est particulièrement redoutable car il s'exécute dans un niveau de privilèges plus élevé que le noyau. Nous avons donc décidé d'utiliser cette même technologie, non pas pour attaquer, mais pour défendre.

Les extensions pour gérer la virtualisation sur les processeurs Intel définissent en particulier un support de virtualisation au niveau du processeur sur la série IA32. Cette extension permet de supporter deux types de logiciels : (1) le moniteur de machine virtuelle (*Virtual Machine Monitor* ou VMM, c'est-à-dire l'hyperviseur) qui se comporte comme un hôte réel est qui a le contrôle complet du processeur et des autres parties matérielles ; (2) les systèmes invités, exécutés dans des machines virtuelles (VM). Chacune de ces machines virtuelles s'exécute indépendamment des autres et utilise la même interface pour le processeur, la mémoire, la mémoire de masse, la carte graphique et les entrées/sorties fournies par la plateforme physique.

Le support processeur pour la virtualisation est fourni par une forme d'opérations du processeur appelées opérations VMX. Il y a deux types d'opérations VMX : les opérations VMX *root*, qui sont disponibles pour l'exécution de l'hyperviseur et les opérations VMX *non-root* qui sont disponibles pour l'exécution du logiciel invité. Le comportement du processeur en mode VMX root est quasiment le même que le comportement en mode VMX non-root avec la différence qu'un ensemble d'instructions supplémentaires est disponible. Le comportement du processeur en mode VMX non-root est restreint et modifié pour faciliter la virtualisation. À la place de leur comportement habituel, certaines instructions et événements causent des transitions vers la VMM, appelées *VM-exits*. Comme ces VM-exists viennent se substituer au comportement habituel, les fonctionnalités du logiciel en mode VMX non-root sont donc limitées. Ces limitations permettent à l'hyperviseur de garder le contrôle des ressources du processeur.

Le cycle de vie d'un hyperviseur peut être résumé comme suit. Tout d'abord, le logiciel passe en mode VMX en exécutant l'instruction VMXON. Ensuite, à l'aide de *VM-entries*, l'hyperviseur lance les systèmes invités dans des machines virtuelles (pour réaliser une opération de type VM-entry, l'hyperviseur exécute les instructions VMLAUNCH et VMRESUME). Il récupère ensuite le contrôle à l'aide de VM-exits qui sont automatiquement déclenchés par le logiciel invité (sans que ce dernier puisse les empêcher). Ces instructions transfèrent le contrôle à un point d'entrée spécifié par l'hyperviseur. Ce dernier peut alors prendre une décision en fonction de la cause du VM-exit courant et redonner la main à la machine virtuelle à l'aide d'une VM-entry. Facultativement, l'hyperviseur peut décider de s'arrêter et de quitter les opérations VMX (en exécutant l'instruction VMXOFF).

Les opérations VMX non-root et les transitions VMX sont contrôlées par une structure de données appelée *Virtual-Machine Control Structure (VMCS)*. L'accès à cette structure est gérée par un composant de l'état du processeur appelé "pointeur VMCS" (il contient l'adresse de la VMCS). Ce pointeur est lu et écrit à l'aide des instructions VMPTRST et VMPTRLD. L'hyperviseur configure la VMCS à l'aide des instructions VMREAD, VMWRITE et VMCLEAR. Il est important de noter que ces instructions ne déclenchent des VM-exists que si elles sont exécutées en mode VMX non-root. La Figure 7.2 résume la façon d'utiliser ces instructions.

Dans la suite, avant de présenter comment nous avons utilisé cette technique pour protéger le noyau, nous présentons d'abord une classification des actions malveillantes en mode noyau que nous connaissons.

7.3.2 Actions malveillantes en mode noyau et protections associées

Nous considérons ici uniquement les actions malveillantes impliquant une perte d'intégrité d'un noyau de système d'exploitation en cours d'exécution. La perte d'intégrité du noyau peut provenir d'une altération 1) de la mémoire du noyau, 2) de l'environnement d'exécution du noyau (les registres du CPU, la mémoire interne du CPU) et 3) des périphériques (mémoire interne, registres).

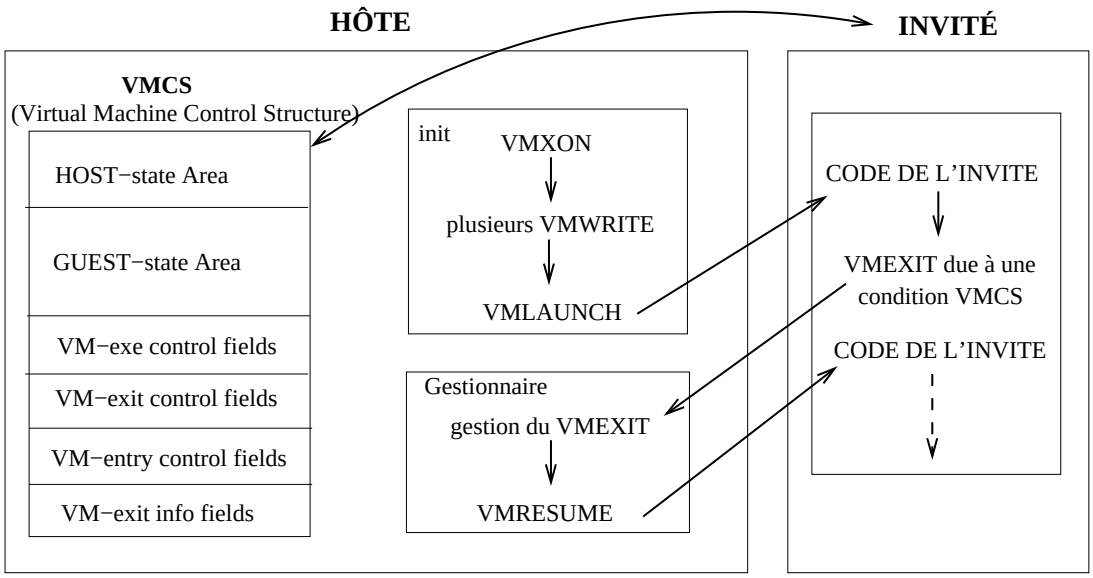


FIG. 7.2 – Bref aperçu de Intel VT-x

7.3.2.1 Les classes d'actions noyau malveillantes

Nous proposons ainsi une classification qui se découpe en trois classes principales correspondant à ces trois altérations possibles :

- **Classe 1 : altération de la mémoire du noyau**.
  - **Classe 1.1 : modification inappropriée du chemin d'exécution du noyau.** Cette classe est caractérisée par les actions malveillantes qui ont besoin d'injecter du code afin d'effectuer leur activité malveillante. Cette injection peut se faire par 1) ajout d'une région de code du noyau malveillante et détournement du flux d'exécution vers cette région [LKM 1999], 2) écrasement d'une région de code du noyau existante avec du code malveillant [Cesare 1999], 3) injection de code malveillant dans une région de données du noyau et détournement du flux d'exécution vers cette région et 4) injection de code malveillant au sein d'une région n'appartenant pas au noyau (typiquement, une région de l'espace de l'utilisateur) et détournement du flux d'exécution vers cette région.
  - **Class 1.2 : modification des variables d'état du noyau**. Cette classe regroupe les modifications des variables dont dépend l'exécution du noyau : par exemple, les variables telles que le *program counter* qui sont stockées dans la pile mais aussi les attributs des tables de pages du noyau par exemple (`Read/Write, No eXecution, ...`).
- **Classe 2 : altération de l'environnement d'exécution.** Cette classe regroupe les modifications des registres du CPU tels que sélecteurs de segments `cs, ds, ss`, le registre `idtr` contenant l'adresse de la table des interruptions ou le registre `gdtr` contenant l'adresse de la

table de pages, etc.
- **Classe 3 : altération des périphériques.** Cette classe regroupe les modifications qui peuvent être faites dans les registres ou la mémoire interne des périphériques. Comme le noyau communique avec les périphériques, ces modifications sont susceptibles d'engendrer de façon indirecte une altération du noyau lui-même.

Pour effectuer ces modifications, les logiciels malveillants ont besoin de vecteurs d'accès leur permettant d'atteindre la mémoire, ou l'environnement d'exécution ou les périphériques. Nous n'aborderons volontairement pas en détail ces vecteurs d'accès. Des informations complètes peuvent être consultées dans [Lacombe et al. 2009]. Pour en donner simplement un aperçu, l'accès peut se faire :

- Par l'intermédiaire de la MMU (*Memory Management Unit*) et peut provenir d'une fonctionnalité du système permettant l'accès direct à la mémoire (par exemple par les périphériques virtuels `/dev/mem` ou `/dev/kmem` [Phrack58 2001, Phrack61 2003]) ou d'une exploitation d'une vulnérabilité (débordement de tampon, chaînes de format, etc) [Phrack64 2007] qui permet de corrompre l'espace du noyau.
- Par l'intermédiaire du bus d'entrées/sorties supportant le DMA (*Direct Memory Access*). Dans ce cas, il s'agit des périphériques de type *Bus Mastering* DMA, lesquels sont aptes à prendre le contrôle du bus et à effectuer un transfert de données dans la mémoire principale sans aucune intervention de la CPU. Par exemple, le bus Firewire peut être utilisé pour lire ou écrire des données dans la mémoire physique sans le consentement du système d'exploitation [Piegdon 2008, Dornseif et al. 2005, Boileau 2006].

7.3.2.2 La protection du noyau vis-à-vis des actions malveillantes

Les moyens de protection connus permettant de se prémunir d'actions malveillantes dans le noyau sont toutes actuellement basées sur des moyens logiciels. Nous ne rentrons pas ici dans les détails de ces mécanismes mais nous pouvons citer à titre d'exemple le projet OpenWall (protection en mode utilisateur seulement), les patchs *grsecurity* [Grsecurity] (incluant *PaX* [Pax]), les mécanismes tels que *StackGuard* [Cowan 1998], PointGuard [Cowan 2003] ou *Propolice/SSP — Stack-Smashing Protection*. Même si ces approches sont effectivement efficaces, elles sont toujours contournables par du code malveillant tels que des rootkits qui s'exécutent déjà dans le mode noyau.

Nous proposons donc dans la suite l'utilisation de la virtualisation matérielle pour une protection efficace contre ce type de code malveillant.

7.3.3 La virtualisation matérielle pour la protection contre les maliciels en mode noyau

Dans cette approche, nous implémentons un hyperviseur qui contrôle certaines des actions qu'un noyau peut effectuer. Le concept majeur est d'essayer de garantir la préservation de contraintes du noyau.

Cette approche qui est décrite dans la suite de cette section se suffit à elle-même pour les classes 1.1.2, 1.1.3 et 1.14. Pour la classe 1.1.1 notre approche est complémentaire des solutions examinées précédemment. Enfin, en ce qui concerne la classes 1.2, 2 et 3, notre approche possède une aptitude unique à restreindre le mode ring-0 (c'est-à-dire le mode noyau) et ainsi peut enrayer les actions malveillantes de ces classes. Nous donnons dans le paragraphe 7.3.3.2 un exemple concernant cette classe 3.

7.3.3.1 Aperçu général de Hytux

Nous avons partiellement développé une preuve de concept pour une cible Linux x86 qui s'exécute sur une plateforme 64 bits supportant la technologie Intel VT-x [Intel] et facultativement Intel VT-d [Intel VT-d]. Notre preuve de concept est appelée Hytux et consiste en un hyperviseur léger qui dépend de ces technologies de virtualisation (cf. figure 7.3). Il s'installe lui-même en tant qu'hyperviseur sur un système faisant tourner Linux et place ensuite ce dernier dans une machine virtuelle qui est alors surveillée et contrôlée (au travers de la configuration d'une seule VMCS).

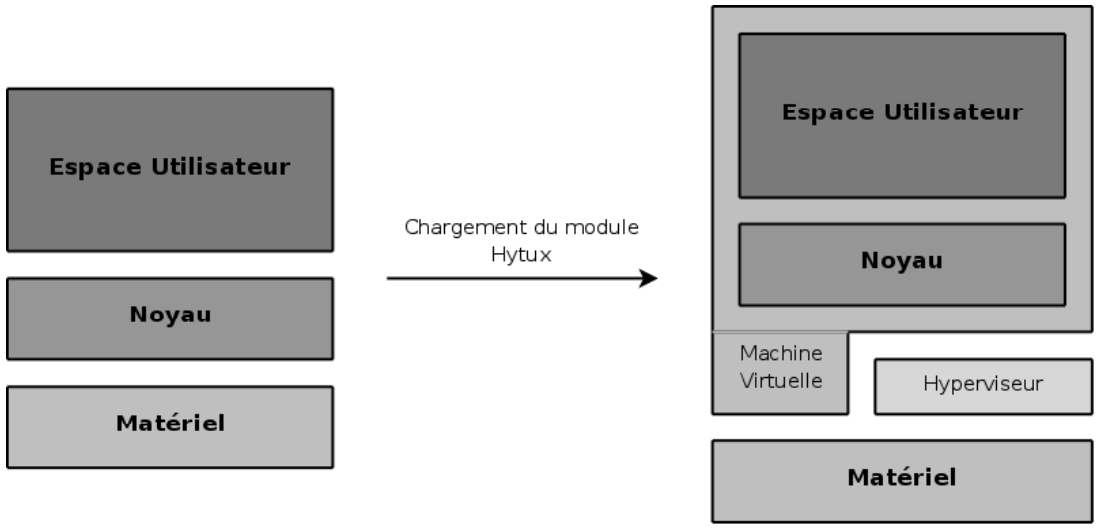


FIG. 7.3 – Hytux – un hyperviseur léger

Dans ce qui suit, nous présentons un exemple de mécanisme de protection implémenté par notre hyperviseur.

7.3.3.2 Protection des objets contraints par le noyau face à une altération depuis la CPU

Le raisonnement derrière ce mécanisme est de préserver les entités qui sont considérées contraintes par le noyau. Nous définissons le concept d'Objet Contraint par le Noyau (*Kernel-Constrained Object*) dans ce qui suit.

Un *Objet Contraint par le Noyau (KCO – Kernel-Constrained Object)* est une entité du système sur lequel le noyau s'exécute et qui devrait être *légitimement* dans un état fixe ou dans un état prédictible et déterministe, durant l'exécution du système.

Nous insistons dans cette définition sur le fait que l'entité est considérée être un KCO si elle est contrainte dans sa spécification, peu importe que l'implémentation soit incorrecte ou qu'une faille de conception existe.

7.3.3.3 La préservation des KCO expliquée au travers d'un exemple.

Le principe du mécanisme de protection est d'empêcher que les KCO soient altérés d'une quelconque manière. Notons que le premier état des KCO que notre hyperviseur (Hytux) voit est supposé être sûr. À partir de cette situation, Hytux essaie d'empêcher les KCO d'être altérés. Pour bien comprendre ce concept, nous prenons l'exemple du registre `idtr` qui est un KCO du point de vue du noyau Linux[3]. En effet, il est positionné lors de l'initialisation du système avec l'adresse de l'IDT (Interrupt Descriptor Table) et n'est pas supposé être modifié plus tard. Cependant, l'instruction du processeur `lidt` disponible depuis le mode ring-0 (c'est-à-dire le mode noyau) permet de charger une nouvelle adresse dans ce registre [Phrack59 2002]. Par conséquent, si le noyau contient un bogue qui peut être exploité ou une fonctionnalité (que nous appelons dans ce contexte une faille de conception) pour exécuter cette instruction avec un paramètre arbitraire, le KCO `idtr` pourrait être altéré. Néanmoins, puisque le registre `idtr` est un KCO, nous devons préserver la contrainte fixe qui régit le registre `idtr`. Afin de pourvoir à ce besoin, notre approche est d'émuler l'instruction `lidt` à l'intérieur de notre hyperviseur assisté par le matériel. Ainsi, quand le noyau exécute cette instruction pour la première fois, le comportement normal est émulé par Hytux, puis ce dernier passe de façon permanente à une émulation qui ne fait rien. De cette façon, le KCO est préservé. L'émulation de l'instruction `lidt` est

[3]Sur l'architecture IA32, les interruptions sont numérotées de 0 à 255. Chacune d'entre elles est associée à une routine si celle-ci a été positionnée par le noyau. Cette routine est une fonction qui est exécutée lorsque l'interruption est levée. Toutes ces routines sont accessibles depuis une table spécifique en mémoire : l'IDT (*Interrupt Descriptor Table*). Le noyau initialise cette table et charge son adresse dans le registre du processeur `idtr` via l'instruction `lidt`.

aisément accomplie : une VM-exit est rendu obligatoire en positionnant à la valeur 1, le champ `Descriptor-table exiting` de la VMCS. Nous procédons de même pour le registre `gdtr`, qui est également étiqueté KCO[4]. Au sujet des registres de contrôle `cr0` et `cr4`, nous agissons de la même façon mais seulement pour leurs bits qui peuvent être considérés comme des KCO[5].

Finalement, le registre de contrôle `cr3` est un cas particulier, il fait partie d'un KCO bien plus compliqué qui couvre les contraintes associées aux régions de mémoire de données et de code. Nous avons proposé une solution pour protéger l'agencement de l'espace de mémoire du noyau en tant que KCO : toute modification de cet agencement est interceptée et vérifiée. L'interception est réalisée par l'intermédiaire de VM-exits pré-configurés. En quelques mots, la vérification est implémentée grâce notamment à l'utilisation des attributs de pages `R/W`, à la modification de la fonction d'allocation VMALLOC du noyau et au contrôle d'utilisation du registre cr3. Plus de précisions peuvent être trouvées dans [Lacombe et al. 2009].

Pour conclure sur ce mécanisme de notre hyperviseur, il est nécessaire de remarquer que les KCO sur lesquels nous nous sommes concentrés ne constituent pas une liste exhaustive. Nous avons seulement souhaité montrer certains KCO du noyau Linux ainsi que la façon de les protéger contre n'importe quelle altération. Nous tenons à souligner le fait que tous les KCO ne peuvent pas être aussi facilement capturés. Cependant, la seule préservation de certains KCO bien choisis permet la protection du noyau contre de nombreux maliciels agissant au niveau noyau (tels que ceux qui dépendent de l'écriture soit dans la GDT, ou dans l'IDT ou encore dans la table des appels système ou bien dans des registres comme `idtr`, `gdtr`, etc.), et cela d'une façon globale.

7.4 Conclusion

Nos travaux dans le domaine de la protection des noyaux de systèmes d'exploitation nous ont tout d'abord mené à l'analyse de rootkits en mode noyau (définition d'une architecture et proposition de critères pour leur évaluation). Nous avons également conçu un rootkit de façon à ce qu'il soit le plus furtif possible.

Nous avons ensuite présenté des mécanismes de sécurité qui empêchent l'exécution de maliciels en mode noyau ou du moins tentent de les contenir. Afin de rendre ces mécanismes incontournables, ils doivent s'exécuter dans un mode plus privilégié que celui dans lequel s'exécute le noyau, et doivent donc employer du matériel dédié. C'est pourquoi nous proposons de les implémenter au sein d'un hyperviseur léger, appelé *Hytux*. Un tel hyperviseur accomplit différentes vérifications afin de prévenir la corruption d'entités cruciales contraintes par le noyau s'exécutant au dessus de l'hyperviseur. Nous avons enfin proposé une première preuve de concept pour un noyau Linux x86 sur un système 64 bits qui supporte la technologie de virtualisation d'Intel.

[4]Sur les processeurs Intel, il est possible d'utiliser la segmentation. Le noyau doit alors établir des segments en écrivant leurs descriptions en mémoire dans une table, appelée GDT (*Global Descriptor Table*). Ensuite, il charge l'adresse de cette table dans le registre `gdtr` de façon à indiquer au processeur où est cette table.

[5]Intel VT-x fournit des masques invité/hôte pour ces registres de contrôle, ce qui simplifie le procédé.

Le démonstrateur Hytux est pour l'instant en cours de développement, et nous avons l'intention de le publier en *open source* lorsqu'il sera terminé.

Bibliographie

[Lacombe et al. 2008] E. Lacombe, F. Raynal et V. Nicomette, "Rootkit Modeling and Experiments under Linux", *Journal in Computer Virology*, vol. 4, May 2008, pp. 137-157.

[Filiol 2005] E. Filiol, "Computer Viruses : From Theory to Applications", Springer Verlag France, IRIS International Series, 2005.

[Lacombe et al. 2007] E. Lacombe, F. Raynal et V. Nicomette, "De l'invisibilité des rootkits, application sous Linux", *Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC)*, Rennes (France), 30 Mai-1 juin 2007, 27p.

[Rutkowska 2006] J. Rutkowska, "Subverting Vista Kernel For Fun And Profit", *Black Hat* in Las Vegas 2006, 2006, http ://invisiblethings.org/papers.html

[LKM 1999] Pragmatic and THC, "(nearly) Complete Linux Loadable Kernel Modules. The definitive guide for hackers, virus coders and system administrators", http ://new-data.box.sk/raven/lkm.html, 1999.

[Cesare 1999] S. Cesare, "Kernel Function Hijacking", `http://www.l0t3k.org/biblio/kernel/english/kernel-hijack.txt`, 1999.

[Lacombe et al. 2009] E. Lacombe, V. Nicomette et Y. Deswarte, "Une approche de virtualisation assistée par le matériel pour protéger l'espace noyau d'actions malveillantes", *SSTIC 2009 (Symposium sur la Sécurité des Technologies de l'Information et des Communications)*, Rennes (France) 4-6 juin 2009, pp. 321-346.

[Phrack58 2001] Sd and Devik, "Linux On-the-fly Kernel Patching Without LKM", *Phrack*, Vol. 58, 2001.

[Phrack61 2003] C0de, "Reverse symbol lookup in Linux kernel", *Phrack*, Vol. 61, 2003.

[Phrack64 2007] Sqrkkyu and twzi, "Attacking the Core : Kernel Exploiting Notes", *Phrack*, Vol. 64, 2007.

[Piegdon 2008] D. R. Piegdon, "Hacking in physically addressable memory - a proof of concept", Easterhegg, 2008.

[Dornseif et al. 2005] M. Dornseif et al., "FireWire - All your Memory are Belong to Us", *Proceedings of CanSecWest/core05*, `http://md.hudora.de/presentations/firewire-cansecwest`, 2005.

[Boileau 2006] A. Boileau, "Hit by a Bus : Physical Access Attacks with Firewire", *Proceedings of Ruxcon 2006*, `http://www.security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf`.

[Grsecurity] B. Spengler et al., "Grsecurity features", `http://www.grsecurity.net/features.php`.

[Pax] B. Spengler et al., "PaX documentation", `http://pax.grsecurity.net/docs`.

[Cowan 1998] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang et H. Hinton, "StackGuard : Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks", *Proceedings of the 7th USENIX Security Symposium*, San Antonio, Texas, 26-29 janvier, 1998.

[Cowan 2003] C. Cowan, S. Beattie, J. Johansen et P. Wagle, "PointGuard : Protecting Pointers From Buffer Overflow Vulnerabilities", *Proceedings of the 12th USENIX Security Symposium*, 2003.

[Intel] Intel, "Intel 64 and IA-32 Architectures Software Developer's Manual - Volume 3B : System Programming Guide, Part 2", 2008.

[Intel VT-d] Intel, "Intel Virtualization Technology for Directed I/O - Architecture Specification", 2007.

[Phrack59 2002] kad, "Handling Interrupt Descriptor Table for fun and profit", *Phrack*, Vol. 59, 2002.

[Kivity et al. 2007] A. Kivity et al., "KVM : the Linux Virtual Machine Monitor", *Linux Symposium*, 2007.

Chapitre 8

Travaux de recherche : synthèse et perspectives

8.1 Synthèse

Les travaux présentés dans ce mémoire résument l'ensemble de nos activités dans le domaine de la protection des systèmes répartis. Ils s'orientent sur trois axes principaux :

- L'amélioration de la protection des différents services offerts sur les réseaux totalement distribués et ouverts, tels que le réseau Internet. Nous avons pour cela apporté deux contributions (résumées dans les chapitres 3 et 4 concernant 1) les mécanismes d'autorisation dans les systèmes répartis sur Internet et 2) la tolérance aux intrusions des serveurs Internet. Concernant la première contribution, nous avons proposé un nouveau schéma d'autorisation adapté à des services complexes tels que ceux qui sont qui sont à l'heure actuelle proposés sur Internet. Des nouveaux types de permissions ont été proposés de façon à pouvoir gérer des autorisations concernant des opérations composites faisant intervenir plusieurs participants répartis sur différents endroits du réseau. Concernant la seconde contribution, nous avons proposé une architecture de serveur Internet tolérant aux intrusions basée sur l'utilisation de la redondance adaptative associée à de la diversification logicielle et matérielle, ainsi qu'à des mécanismes diversifiés de détection d'intrusion. Cette architecture a donné lieu à un prototype d'un serveur Web.
- L'amélioration de la connaissance des attaquants, des techniques et des outils d'attaques utilisés principalement sur le réseau Internet. Pour cela, nous avons utilisé des mécanismes d'observation de deux types : 1) des pots de miel haute interaction de façon à observer des attaquants humains évoluant sur des systèmes une fois qu'ils ont réussi à les compromettre 2) une plateforme d'exécution de maliciels, préalablement collectés, nous permettant d'analyser précisément le comportement de ces maliciels. Ces deux contributions sont résumées dans les chapitre 5 et 6 de ce mémoire.

– L'amélioration de la sécurité des noyaux de systèmes d'exploitation, à l'aide de mécanismes matériels. Ce thème de recherche nous semble incontournable dans la mesure où assurer la sécurité des couches basses des systèmes est un pré-requis indispensable à une sécurisation globale de l'ensemble d'un système informatique. En particulier, nous nous sommes intéressés à la technologie de virtualisation disponible sur les derniers processeurs Intel (technologie Intel VT-d). Nous avons ainsi, en utilisant cette technologie, conçu et développé un hyperviseur léger dont le rôle est de vérifier la préservation d'un certain nombre d'invariants concernant le noyau Linux.

8.2 Perspectives

Les perspectives que nous envisageons concernant essentiellement les deux derniers axes des travaux présentés dans ce mémoire. Nous les présentons dans les deux sous-sections suivantes.

8.2.1 La connaissance des attaquants et des méthodes et outils d'attaques

Nos travaux concernant l'amélioration de la connaissance des attaquants, de leurs méthodes et outils d'attaques n'en sont qu'à leurs débuts. Nous envisageons notamment à court terme de conforter les résultats résumés dans le chapitre 5 de ce mémoire en déployant une expérimentation basée sur l'utilisation du pot de miel haute interaction présenté dans ce chapitre. Nous estimons indispensables de savoir si les conclusions que nous avons obtenues sur notre pot de miel sont généralisables. Nous sommes actuellement en train de déployer cette expérimentation sur plusieurs pots de miel de ce type sur des réseaux IP différents. Notre projet est de configurer tous ces pots de miel à l'identique et de les connecter sur Internet au même moment et d'observer et d'enregistrer l'activité des attaquants pendant une certaine durée. Nous pourrons alors, en fonction des résultats, généraliser ou non les conclusions de nos précédents travaux.

Cette amélioration de la connaissance des attaquants est fondamentale pour améliorer 1) nos processus de défense et 2) nos méthodes d'évaluation des systèmes informatiques répartis au sens large. Nous pouvons ainsi envisager à plus long terme, les travaux suivants :

– En partenariat avec le groupe OLC avec lequel le coencadrement des travaux de Ion Alberdi (chapitre 6) a déjà eu lieu, nous envisageons de poursuivre ces travaux afin de proposer des solutions originales de défense proactives capables de protéger le réseau, ses services et ses utilisateurs, c'est-à-dire capable d'anticiper sur le lancement d'attaques par les pirates informatiques. L'une des originalités du travail que nous envisageons consiste à concevoir, développer et mettre en œuvre des méthodes utilisant conjointement des outils de métrologie du réseau avec des pots de miel pour obtenir des informations sur le trafic et les activités illégitimes dans l'Internet (ces travaux sont la suite logique des travaux réalisés par Ion Alberdi). La métrologie permet d'étudier le trafic et les activités sur les réseaux de l'Internet et

les pots de miel permettent de collecter des traces de trafic malveillant mais aussi de capturer les logiciels malveillants qui génèrent ces traces. L'analyse de ces maliciels, la caractérisation et la corrélation des traces ainsi collectées permettront d'évaluer précisément le risque qui pèse sur l'Internet et ses utilisateurs. A partir de ces informations, l'objectif de notre travail est de concevoir et mettre en oeuvre des méthodes et des démonstrateurs d'architectures de défense contre les attaques qui soient proactives, c'est-à-dire qui soient capables de défendre le réseau contre des attaques qui se mettent en place, et donc avant que celles-ci ne soient effectivement lancées. Par exemple, la capture et l'analyse des maliciels qui pilotent les botnets peuvent nous permettre d'anticiper le déploiement imminent d'une attaque de type DOS de grande envergure.

- Dans le cadre des activités de recherche du groupe TSF, des travaux concernant l'évaluation quantitative de la sécurité ont été développés. Ces travaux ont abouti à la notion du *graphe des privilèges*. Chaque noeud de ce graphe représente un ensemble de privilèges et un arc reliant deux noeuds N_1 et N_2 représente le fait qu'il est possible à un utilisateur possédant les privilèges du noeud N_1 d'obtenir les privilèges du noeud N_2 grâce à l'exploitation d'une vulnérabilité. A partir de ce graphe, si l'on est capable d'associer une notion d'effort à un arc, c'est-à-dire si l'on est capable d'associer une mesure à l'exploitation d'une vulnérabilité particulière, on peut estimer l'effort nécessaire, en partant d'un certain noeud pour arriver à un noeud cible, c'est-à-dire pour obtenir les privilèges correspondant à ce noeud. Ce graphe est particulièrement intéressant mais lorsqu'on l'utilise de façon à calculer l'effort nécessaire à un attaquant pour atteindre une cible, on doit nécessairement faire des suppositions sur son comportement. En effet, dans un tel graphe, de multiples chemins sont souvent possibles pour accéder à un noeud en partant d'un autre noeud et il est impossible a priori de savoir comment un attaquant va se comporter. Dans ce cadre, l'observation des attaquants, à l'aide de technologies basées sur des pots de miel par exemple, peut être fort utile pour observer les chemins d'attaque que suivent réellement les attaquants et nous permettre ainsi par exemple d'enrichir notre graphe pour y ajouter ce type d'informations.

8.2.2 Protection des noyaux de systèmes d'exploitation à l'aide de mécanismes matériels

Les travaux que nous avons menés dans le cadre de la thèse d'Eric Lacombe (chapitre 7) n'en sont qu'à leur début et nous envisageons de poursuivre ces travaux à court et moyen terme[1]. Nous pouvons par exemple envisager les perspectives suivantes :

- Le démonstrateur Hytux est pour l'instant en cours de développement, et nous avons l'intention de le publier en *open source* lorsqu'il sera terminé. Il reste par exemple à évaluer précisément le coût en terme de performances induit par les mécanismes de protection mis en place. Nous pouvons pour l'instant simplement l'estimer à l'aide de quelques considérations. Notre hyperviseur n'effectue que peu de travail : il ne fait que vérifier certaines contraintes et

[1] Une thèse de doctorat qui débutera en octobre 2009 nous permettra de continuer à travailler sur ce thème.

rend ensuite directement la main au noyau. De plus, l'impact sur les performances du système dépend également de la façon dont se comportent les extensions matérielles de virtualisation (i.e. de la rapidité d'exécution des VM-exit, VM-entry et des injections d'évènements). À ce niveau, nous pouvons étudier les hyperviseurs existants qui mettent à profit la virtualisation matérielle (tel que KVM – Kernel Based Virtual Machine [Kivity et al. 2007]). Ces solutions ne causent pas de ralentissements majeurs du système et par conséquent des résultats similaires sont attendus avec notre approche.

- Protéger l'intégrité du noyau depuis des attaques provenant des couches logicielles est fondamentale mais il a également été montré que certains périphériques peuvent corrompre directement la mémoire même du noyau en utilisant des accès DMA (*Direct Memory Access*). Des attaques récentes ont été mises au point à l'aide par exemple de périphériques Firewire. Il est donc nécessaire de pouvoir contrôler ce type d'accès DMA. En particulier, dans les processeurs Intel et AMD récents, un composant, l'IOMMU, a été intégré, de façon à pouvoir contrôler l'accès à la mémoire depuis le bus d'entrées/sorties via le mécanisme du DMA. Ce composant doit donc permettre d'empêcher ou au moins de limiter les accès DMA malveillants. Il est actuellement en cours d'étude : nous avons analysé son fonctionnement et même identifié quelques failles. Cependant, nous n'en sommes qu'au début de l'analyse et elle nécessite d'être poursuivie.
- Protéger les accès à la mémoire du noyau, que ce soit depuis les couches logicielles ou depuis certains périphériques malveillants est fondamental mais il est tout aussi fondamental d'avoir des mécanismes nous permettant, depuis la mise sous tension d'un ordinateur, d'être certain que tous les éléments chargés en mémoire sont sains au départ. Encore une fois, des technologies matérielles peuvent venir à notre secours. La technologie TXT (*Trusted Execution Path*) des processeurs Intel récents permet d'établir une chaîne de confiance depuis la mise sous tension de la machine jusqu'au chargement du système d'exploitation et des applications. Cette confiance est notamment obtenue grâce à la vérification de signatures cryptographiques à l'aide d'un TPM (*Trusted Platform Module*). Cette technologie et les composants qui lui sont associés sont à étudier et nous venons simplement de commencer cette étude dans le cadre d'un stage de Master.

Enfin, nous avons pour objectif final d'utiliser conjointement ces trois technologies (hyperviseur, IOMMU et TXT) de façon à obtenir une solution de sécurisation relativement complète et efficace.