ENSC-2008-N°127

THÈSE

présentée à l'École Normale Supérieure de Cachan par

Elie Bursztein

pour obtenir le grade de

Docteur de l'École Normale Supérieure de Cachan

Spécialité : Informatique

Anticipation games

Théorie des jeux appliquée à la sécurité réseau.

Soutenue le 26 novembre 2008
devant le jury composé de

Pr. Jean Goubault-Larrecq	LSV, ENS-Cachan	directeur de thèse
Pr. Ludovic Mé	Supélec Rennes	rapporteur
Dir. Jeannette Wing	National Science Foundation et CMU, USA	rapporteur
Dir Claude Kirchner	INRIA Bordeau	président du jury
Pr. David Lubicz	Université Rennes 1 et DGA	membre du jury
Pr. Roberto Segala	Université de Vérone, Italie	membre du jury

Thèse Financée par la DGA.

Résumé

Les réseaux informatiques sous-tendent aujourd'hui l'ensemble des systèmes d'information majeurs du monde : Internet, la bourse, les systèmes bancaires, les réseaux téléphoniques, les centrales nucléaires et bien sûr les entreprises.

Assurer la sécurité des infrastructures réseau est donc critique pour la bonne marche des systèmes d'information. En raison de la taille gigantesque des réseaux et de l'hétérogénéité des protocoles et applications déployés, il est aujourd'hui nécessaire de développer des outils automatiques pour anticiper et prévoir les conséquences des attaques.

Pour répondre à ce besoin, cette thèse propose le premier modèle de théorie des jeux appliqué à la sécurité réseau basé sur la logique TATL. Dans ce modèle, l'administrateur et l'intrus sont vus comme deux joueurs ayant des objectifs opposés, le premier voulant protéger son réseau alors que le second cherche à le compromettre. Ce modèle s'avère efficace pour la prédiction des attaques et leurs conséquences car il permet de modéliser les effets collatéraux, la dimension temporelle des attaques ainsi que leur impact financier.

Afin d'expérimenter la mise en pratique de ce modèle, nous avons conçu l'outil open-source *NetQi* (http ://www.netqi.org). Cet outil est capable d'analyser et de trouver des stratégies de défense et d'attaque complexes (Honey-net) pour des réseaux ayant des milliers de services. De plus nous avons aussi réalisé l'outil open-source *NetAnalyzer* qui analyse les réseaux en temps réel pour construire automatiquement les modèles utilisés par *NetQi*.

Cette thèse a fait l'objet de 7 publications [Bur07b, BGL07, Bur08b, Bur08c, Bur07a, BM09, Bur08a]. La publication [Bur08c] a été élu meilleur papier de la conférence.

Remerciements

Cette thèse n'aurait jamais vu le jour sans la contribution et le soutien de nombreuses personnes. En particulier je remercie mon directeur de thèse *Jean Goubault-Larrecq* qui m'a appris ce que c'est que la recherche et m'a encadré et soutenu durant cette difficile transition du monde de l'ingénierie à celui de la recherche.

Merci à *Jeannette* qui, en dépit de ses énormes responsabilités et un planning démentiel a su trouver le temps de me conseiller tout au long de la thèse et de venir en France pour la soutenance. Merci à *Ludovic* pour avoir été si ouvert et encourageant et ce dès notre première rencontre à l'autre bout de la terre. Merci à *Claude* pour m'avoir fait l'honneur d'être président du jury et d'avoir trouver une place pour moi dans son emploi du temps surchargé. Merci à *Roberto* pour sa gentillesse infinie, ses précieux conseils tout au long de la thèse et d'être venu spécialement d'Italie pour moi. Merci à *David* pour avoir participé à mon jury mais surtout pour toutes les discussions passionnantes que nous avons eu ensemble.

Un grand merci à la DGA pour avoir financé cette thèse et fourni un encadrement et des opportunités qui furent essentielles à la réussite de celle-ci. En particulier merci à *Christine Couesnon* de s'être occupée de moi au cours de ces trois ans et merci à toute l'équipe des doctoriales.

Je remercie l'ensemble des membres de mon laboratoire le LSV et en particulier : *Florent, Stéphane, Philippe, Ralf, Dietmar, Sylvain, Pierre, Emmanuel, Ghassan, Fabrice, Pascal, Nathalie, Mathieu, Régis et Thomas, Virginie et Catherine* pour leur conseils, encouragements et discussions.

Je remercie aussi les membres de mon ancienne équipe à ULM avec qui je suis resté proche tout au long de la thèse et avec qui j'ai partagé de si agréables moments. Merci à *Julien, Xavier, Antoine, Jérome, Bruno, Laurent, David, Elodie et Patrick.*

Une thèse c'est aussi une aventure personnelle et je n'aurais jamais réussi sans ces trois personnes clé que sont *Elisabete, Audrey et Camille.* Elles furent mon rayon de soleil dans les mauvais jours et mes supportrices inconditionnelles. On ne peux rêver meilleur soutien qu'elles et je n'ai pas de mots pour exprimer tout l'amour et la gratitude que j'ai pour elles. Merci à mes amis pour avoir su m'aider à garder mon équilibre et me sortir la tête du "bocal". En particulier merci à *Pitoo, Emeline, Nicolas, Fred, Arnaud, Jean-Michel et Céline.*

Table des matières

1 Introduction **9**
- 1.1 Analyse de risques . . . 10
- 1.2 Méthodes formelles et analyse de risques . . . 13
- 1.3 Terminologie des méthodes formelles . . . 15
- 1.4 Terminologie du Model-checking . . . 17
- 1.5 Défis . . . 23
- 1.6 Contenu de la thèse . . . 25
- 1.7 Plan de la thèse . . . 26

I Modélisation **29**

2 Modèle pour l'analyse de risque **31**
- 2.1 Historique . . . 32
- 2.2 Quelques frameworks de quatrième génération . . . 35
- 2.3 Comparaison des frameworks . . . 46
- 2.4 Points Clés . . . 50

3 Anticipation Games, le framework **51**
- 3.1 Etude de cas : un service Web redondant . . . 52
- 3.2 Couche basse . . . 54
- 3.3 Modéliser l'évolution du graphe de dépendances . . . 60
- 3.4 Couche supérieure : les Anticipation Games . . . 66
- 3.5 Analyser la sécurité d'un réseau . . . 73
- 3.6 Exemple d'exécution et analyse automatique . . . 75
- 3.7 Points clés . . . 75

II Mise en pratique **77**

4 Anticipation Games, l'implémentation : NetQi **79**
- 4.1 *NetQi* Présentation . . . 80
- 4.2 Premier exemple : l'élément de surprise . . . 82

4.3 Deuxième Exemple : Vérification d'une propriété TATL 89
4.4 Points clés 94

5 Anticipation games, du réseau réel au modèle 95
5.1 Identification des services et vulnérabilités d'un réseau 96
5.2 Identification des dépendances 100
5.3 Méthodes avancées d'identification de protocoles 103
5.4 Exemples d'identification de technique de camouflage 109
5.5 Identifier les séquences de dépendances 115
5.6 Complexité de la construction automatique 121
5.7 Résumé 121

III Extensions 123

6 Stratégies pour l'administrateur et l'intrus 125
6.1 Ajout des coûts et récompenses aux Anticipation Games 127
6.2 Objectifs d'une Stratégie 135
6.3 Décidabilité et Complexité 140
6.4 Fichier de jeu *NetQi* correspondant à l'exemple 144
6.5 Mise en pratique 144
6.6 Points Clés 149

7 Emplacement, pénalités et timeline 151
7.1 l'Anticipation Game d'exemple 154
7.2 Emplacements 157
7.3 Utilisation d'une Timeline 162
7.4 Les coûts liés au temps 165
7.5 Décidabilité et complexité de l'extension 168
7.6 Stratégies de défense multi-sites 169
7.7 Evaluation de l'extension grâce à *NetQi* 173
7.8 Points Clés 176

8 Conclusion 177

Annexes 181

Index 181

Bibliographie 185

CHAPITRE 1

Introduction

"Security against defeat implies defensive tactics."

Sun Tzu, The Art of War. IV.1

11 septembre 2004 [dn04], depuis le début de la matinée l'ensemble des téléphones cellulaires utilisant le réseau téléphonique Bouygues sont inactifs. Pendant plusieurs heures, l'ensemble des 6 millions d'abonnés est dans l'impossibilité d'émettre ou recevoir un appel. A l'origine de cet incident majeur, la panne des deux serveurs d'authentification et un effet boule de neige imprévisible qui aboutira à la paralysie totale du réseau. Nul doute que si les conséquences de la panne de ces deux serveurs avaient pu être anticipées, cette paralysie aurait pu être évitée en prenant des mesures préventives. Cependant l'explosion de la complexité des réseaux et des interconnexions, rend la prédiction des conséquences d'une panne ou d'une attaque extrêmement ardue. Cet accroissement drastique de la complexité des réseaux est principalement dû à l'accroissement de la bande passante et la convergence des médias. Désormais, les réseaux téléphoniques, vidéo, et informatiques se confondent grâce aux avancées de la voix sur IP (VOIP), et les avancées du streaming vidéo. Il résulte de cette convergence, que la problématique de la sécurité réseau ne concerne plus uniquement les réseaux informatiques mais l'ensemble des communications planétaires. A tel point qu'en 2003, un virus informatique a mis en danger le fonctionnement d'une centrale nucléaire aux USA [kp03]. La complexité des réseaux est renforcée par l'accroissement de l'hétérogénéité des appareils qui dépendent de fonctionnalités réseau. Si les téléphones portables sont bien sur le fer de lance de cet accroissement avec l'utilisation de la 3G ou encore de la géolocalisation, de nombreux autres périphériques suivent cette tendance.

Ainsi, les consoles *"nextgen"* telles que la Xbox360 et la Wii utilisent un accès Internet pour être mises à jour et proposer l'achat de jeu directement téléchargeables. Les GPS, quant à eux, utilisent une connexion réseau pour obtenir les dernières informations relatives au réseau routier. Cette hétérogénéité des appareils connectés au réseau et de leurs besoins rend difficile la conception et la mise en place d'un réseau sécurisé. Ainsi, pour permettre l'accès au réseau Wifi à des Nintendo DS, il est nécessaire d'utiliser l'algorithme de chiffrement WEP puisque c'est le seul pris en charge par ces consoles, alors qu'il est notoirement connu pour être vulnérable à de nombreuses attaques [Tew07]. C'est de ce besoin pour les réseaux complexes de l'émergence d'un moyen efficace et fiable pour anticiper et limiter les conséquences des attaques et des pannes en prenant les actions préventives nécessaires qu'est né notre travail.

1.1 Analyse de risques

Un *risque* est un évènement redouté et la discipline analytique consacrée à leur traitement s'appelle l'*analyse de risques* (*Risk analysis*) . Elle regroupe l'ensemble des méthodes et modèles qui visent à les évaluer, les analyser et les gérer. Les deux grandes branches de cette discipline sont *l'évaluation des risques* (*Risk assessment*) et *la gestion des risques* (*Risk management*) . On définit ces termes [Hof89, Ana93] comme suit :

Définition 1.1. ***L'analyse de risques*** *est la discipline analytique destinée à permettre l'implémentation d'une politique de sécurité en tenant compte du ratio coût/bénéfice dans un environnement réel et complexe. Elle est composée de deux branches :* l'évaluation des risques *et* la gestion des risques.

Définition 1.2. ***l'évaluation des risques*** *consiste à identifier les risques présents au sein d'un système, leurs conséquences et à évaluer l'efficacité des parades associées.*

Définition 1.3. ***La gestion des risques*** *consiste à sélectionner les parades qui minimisent l'exposition aux risques tout en prenant en compte les contraintes fonctionnelles ainsi que les contraintes pragmatiques d'ordre social, politique, et financier.*

Notre recherche s'inscrit donc naturellement dans *l'évaluation des risques* qui est la phase amont de l'analyse de risques. *La gestion des risques* quant à elle est typiquement dévolue aux décideurs, qui utilisent les résultats de *l'évaluation des risques* pour décider quelles mesures préventives seront mises en œuvre. Cette dichotomie induit ce que l'on nomme la notion de *non-optimalité technique des parades choisies* [Dac94]. En effet, rien ne garantit que les décideurs suivront les recommandations fournies par l'évaluation des risques pour des raisons financières, fonctionnelles ou politiques. Cet amendement des recommandations techniques conduit à l'introduction dans le système de *vulnérabilités fonctionnelles*.

Un exemple de *vulnérabilité fonctionnelle* est le fait d'imposer un chiffrement Wifi WEP pour permettre aux Nintendo DS d'accéder au réseau, en dépit de la recommandation technique d'utiliser un chiffrement sûr comme le WPA.

Afin de limiter l'introduction de ce type de vulnérabilité, il est primordial que *l'évaluation des risques* tienne compte du maximum de contraintes possibles et en particulier des contraintes financières . Il est aussi vital qu'elle soit fondée sur des modèles rigoureux afin que ses conclusions ne puissent être remises en cause.

La non-optimalité technique des parades choisies n'est pas propre à l'informatique. Comme noté dans [Dac94], les constructeurs automobiles ont adopté en masse le système des *"airbags"* en raison de l'engouement du public alors que rien n'indique que les autres solutions disponibles ne soient pas techniquement meilleures. Ainsi, un constructeur a abandonné les colonnes de direction rétractables au profit des airbags alors que ce système était utilisé avec succès sur de nombreux modèles et ce depuis des années.

1.1.1 Les six étapes d'une analyse de risques

La discipline de l'analyse de risques regroupe un ensemble hétéroclite de techniques et méthodes, allant de la simple *liste de contrôle* (*check-list*) à la base de connaissances en passant par les modèles formels. En dépit de la variété des approches existantes, le résultat de l'application d'une analyse de risques reste toujours le même : prendre des décisions. Il est aussi possible de dégager six étapes communes à la plupart des méthodes [ZHM90] qui sont :

1. *Identification des actifs* du système d'information : on recense les actifs que l'on désire protéger.
2. *Identification des menaces et calcul des risques* : on liste les menaces qui pèsent sur chaque actif identifié à l'étape précédente.
3. *Analyse des conséquences* : cette étape vise à estimer l'impact sur le système d'information de la réalisation des menaces identifiées à l'étape précédente.
4. *Evaluation des parades existantes* : on estime les contre-mesures existantes en tenant compte de leur efficacité.
5. *Evaluation du niveau de sécurité existant* : en fonction des résultats des étapes précédentes, on estime le niveau de sécurité du système d'information et on décide des parades optimales du point de vue coût/efficacité à mettre en œuvre pour améliorer ce niveau.
6. *Formulation d'un plan de sécurité* : le résultat final de l'analyse débouche sur un compte-rendu des états précédents et un plan de mise en œuvre des mesures de protection retenues.

1.1.2 L'analyse de risques est un processus dynamique

L'analyse de risques est un processus dynamique car la sécurité du système d'information doit en permanence être réévaluée pour tenir compte des changements induits par les contraintes extérieures et les actions des utilisateurs. Ainsi, à chaque fois qu'une nouvelle vulnérabilité est découverte ou un nouveau service mis en place, il est nécessaire d'en évaluer les conséquences au niveau de la sécurité. Cet aspect dynamique de la sécurité est d'ailleurs un des points clés de l'ensemble des méthodologies récentes utilisées pour l'élaboration de politique de sécurité informatique, à l'instar de la méthode **Octave** [AD01] du CERT. D'un point de vue opérationnel, il est évident que cette réévaluation continuelle de la sécurité rend l'analyse de risques encore plus difficile : il faut non seulement que l'analyse soit efficace mais aussi qu'elle puisse être effectuée rapidement et de manière répétitive. Ces contraintes amènent naturellement à vouloir automatiser au maximum l'analyse afin de tirer profit des capacités de traitement des ordinateurs.

1.1.3 La place de l'analyse de risques pour les réseaux dans les politiques de sécurité

La couverture des *PSI* (politiques de sécurité des systèmes d'information) issue des méthodologies récentes est particulièrement étendue. Ainsi, la méthode Octave se veut une méthode qui se concentre sur "le cyber, le physique, le système et les évènements extérieurs" [AD01]. Une politique de sécurité réalisée en utilisant cette méthodologie couvrira donc les risques les plus divers, allant du tremblement de terre à l'intrusion informatique en passant par les risques terroristes. Cette généralité est vue comme un atout par les défenseurs de ces méthodes et comme une faiblesse par ses détracteurs. Sans vouloir entrer dans ce débat, reconnaissons que développer un modèle qui couvre l'ensemble de ces risques de manière rigoureuse est une tâche extrêmement ardue. Il semble en effet complexe de trouver un modèle qui soit capable de prendre en compte de manière précise les spécificités d'une attaque au fusil d'assaut et d'une attaque utilisant un programme informatique en même temps. C'est pourquoi notre travail vise plus modestement à fournir le modèle et les outils nécessaires pour couvrir les six étapes de l'analyse de risques concernant la partie réseau du système d'information. Assurer la sécurité du réseau est un objectif prioritaire car il sous-tend l'ensemble des activités d'un système d'information. Il n'est donc guère surprenant que de nombreux modèles aient été développés spécifiquement pour cette tâche comme nous le verrons au chapitre suivant.

1.2 Méthodes formelles et analyse de risques

Comme dit précédemment, les méthodes et les techniques d'analyse de risques forment un ensemble hétéroclite où le meilleur côtoie le pire. Au sein de ce corpus, une branche particulière émerge comme étant la plus rigoureuse et l'une des plus prometteuse : les méthodes formelles. Cette approche est considérée comme la plus rigoureuse car elle utilise un formalisme mathématique, qui permet d'obtenir une preuve irréfutable que la conclusion de l'analyse est correcte. Qui plus est l'utilisation d'un modèle mathématique permet d'assurer que l'analyse sera la plus impartiale possible. Cette capacité à fournir une preuve définitive et irréfutable combinée à la démocratisation des ordinateurs explique l'engouement extraordinaire que rencontre l'application des méthodes formelles aux diverses problématiques de sécurité ces dernières années.

En dehors de l'analyse de risques, de nombreux domaines de la sécurité ont bénéficié de l'apport des méthodes formelles. Ainsi, c'est sans surprise que l'un des premiers domaines à en avoir profité fut l'analyse de protocoles cryptographiques [DY83, DMTY97, GL00]. En effet, obtenir une preuve irréfutable qu'un protocole garantit le secret [CMR01] est un des objectifs majeurs de la cryptographie. Dans ce domaine, l'une des plus grande réussite des méthodes formelles est sans doute d'avoir su démontrer l'existence d'une attaque à l'encontre d'un protocole que l'on croyait sûr depuis 17 ans : le fameux protocole de Needham-Schroeder [Low95].

Un autre domaine où l'apport des méthodes formelles fut important est l'analyse de programmes. Elles servent dans ce cadre à garantir qu'un programme est exempt de certaines classes de bug [BBF+01]. La vérification de programmes est extrêmement ardue car indécidable dans le cas général (théorème de Rice) et à l'instar des réseaux, le passage à l'échelle sur des gros codes est problématique.
Parmi les techniques inventées pour faire face à ces problèmes, l'interprétation abstraite [CC77] se distingue comme étant l'une de celle qui apporte le plus de réponses. A tel point qu'elle est aujourd'hui devenue une branche à part entière des méthodes de vérification. Un de ses plus grand succès pour la vérification de code est l'outil ASTREE [BCC+03] qui a permis de vérifier des centaines de milliers de lignes de code embarquées dans les avions Airbus.

L'évolution de l'analyse de risques en informatique est indissociable des modèles formels puisque leur relation débute dès l'origine de la discipline. En effet, dès le départ, les modèles formels ont été utilisés pour structurer le cadre de raisonnement et apporter des preuves formelles de sécurité.

Ainsi, les premiers travaux sur la sécurité réseau eurent lieu en 1967 quand les Etats-Unis ont créé un groupe de travail afin d'étudier les moyens de mettre en œuvre les informations classifiées dans les systèmes informatiques. Le rapport final de ce groupe en 1970 [War70] donna lieu à de nombreuses initiatives dont la plus connue est le fameux rapport *Trusted Computer Security Evaluation Criteria* [oD83].

Ce document devint par la suite une norme du Département de la défense américaine *DOD* en 1985 pour les critères de sécurité après avoir subi une révision mineure [oD85]. Ce document, bien que critiqué pour son manque de souplesse notamment, resta l'unique référence pendant des années. Ce document marque le début de l'utilisation de modèles formels pour la sécurité, puisqu'il s'appuie sur le modèle formel de politique par mandats définis par Bell et La Padula [BL76] en 1976. Ce modèle permet de garantir de manière formelle que les règles de contrôle d'accès satisfont effectivement le problème de protection à résoudre.
Au fil du temps, de nombreux autres modèles formels ont été développés pour modéliser les critères d'évaluation de la sécurité, comme par exemple le modèle de Biba [Bib77] qui est le dual pour l'authenticité de celui de Bell-La Padula pour la confidentialité ou encore le modèle dit de *la muraille de chine* de Brewer et Nash [BN89].

Depuis le début des années 1990, on assiste à l'émergence de nombreuses méthodes formelles dédiées à l'analyse de risques pour les réseaux. Cet intérêt s'explique par le rôle clé des réseaux évoqués précédemment, le développement des méthodes permettant l'analyse automatique et leur apparente facilité à être modélisées. On verra en section 1.5 que c'est en fait beaucoup plus complexe qu'il n'y paraît. Notre travail s'inscrit dans cette lignée de méthodes formelles appliquées à l'analyse de risques pour les réseaux. On reviendra en détail sur l'historique des modèles formels pour la sécurité réseau au chapitre 2.

1.3 Terminologie des méthodes formelles

Puisque nous allons nous intéresser à l'application des méthodes formelles pour l'analyse de risques des réseaux, il est nécessaire de présenter les concepts de base des méthodes formelles avant d'aller plus loin.

1.3.1 Modèle formel

Le mot modèle synthétise les deux sens symétriques et opposés de la notion de ressemblance, d'imitation, et de représentation. On définit un modèle au sens mathématique et informatique comme suit :

Définition 1.4. *Un modèle est une abstraction ou un objet conceptuel qui représente un système réel.*

Ainsi le langage UML est considéré comme un langage de modélisation car il propose une représentation abstraite des données d'un système. Par extension, un modèle formel se définit comme suit :

Définition 1.5. *Un modèle formel est une abstraction mathématique qui représente un système réel.*

L'intérêt des modèles formels vient du fait qu'il est possible de prouver des vérités dessus car ils utilisent un formalisme mathématique. Le modèle étant une abstraction du système réel, cette approche permet de prouver des vérités sur un système réel.

1.3.2 Vérification d'un système

L'utilisation de modèles formels intervient lors de la conception d'un système, à l'étape cruciale de la vérification du système avant sa distribution. Les estimations montrent d'ailleurs que de 30% à 50% du budget développement d'un logiciel est consacré à cette étape.

La distinction entre la *vérification*, la *validation* et le *benchmarking* est la suivante : la *vérification* s'assure que l'on construit le système de la bonne manière. La *validation* s'assure que l'on construit le bon système. Le *benchmarking* s'assure que le système aura de bonnes performances quantitatives. On définit donc la vérification d'un système comme suit :

Définition 1.6. *La vérification d'un système vise à s'assurer que le système remplit les objectifs qualitatifs qui ont été identifiés.*

Par exemple, lors de la conception d'un four micro-ondes, s'assurer que celui-ci ne peut se mettre en marche avec la porte ouverte est un des objectifs qualitatifs que l'on peut identifier et vérifier.

1.3.3 Méthodes de vérification

Il existe quatre méthodes classiques pour vérifier que le système se comportera correctement :

1. **La simulation** : on conçoit un modèle du système et on essaye un ensemble de comportements sur ce modèle. Généralement, on simule les cas limites.
2. **Le test** : on conçoit le système ou un prototype et on fait une batterie de tests, pour valider que le système se comporte correctement.
3. **La vérification déductive** : on formalise le système sous forme d'axiomes et de règles d'inférences et l'on prouve que le système se comporte comme prévu grâce à un prouveur de théorème, tel que Coq [INR08] par exemple.
4. **Le model-checking** : on formalise le système sous forme d'un graphe de transition d'états, ou structure de Kripke, et l'on vérifie des propriétés temporelles dessus.

Les méthodes de vérification par simulation et test sont dites empiriques car elles ne testent qu'une partie des exécutions possibles. On n'a donc aucune garantie qu'il n'existe pas une exécution non testée où le système se comporte de manière incorrecte. On appelle cela le problème de couverture. Un exemple tragique d'exécution non prévue est l'explosion en vol de la fusée Ariane 5.

A l'opposé, les méthodes de vérification déductives et le model-checking sont dites formelles car elles garantissent, lorsqu'elles aboutissent, qu'il n'existe aucun cas où le système se comportera de manière incorrecte. La différence majeure entre vérification inductive et model-checking est que la vérification inductive demande une intervention manuelle mais permet de prouver des systèmes infinis là où le model-checking permet de traiter des systèmes finis de manière automatique. Cette différence usuelle tend à s'estomper car le développement de techniques de model-checking pour les systèmes infinis est un domaine en plein expension.
Les réseaux étant des systèmes finis (il y a un nombre de machines, services et failles fini), il est naturel de se tourner vers une approche à base de model-checking pour l'analyse de risques. Notons qu'avoir un système fini ne veut pas dire que celui-ci n'engendre pas des exécutions d'actions infinies. Par exemple : un service réseau tombe en panne, il est réparé, il tombe en panne ...

1.4 Terminologie du Model-checking

De manière classique, on décompose le processus de vérification par model-checking en trois étapes [BBF+01] :

1. La modélisation ou l'on abstrait le système réel sous forme de modèle.
2. La spécification de la propriété à vérifier.
3. La vérification de la propriété sur le modèle.

A la fin de la vérification, le model-checker , qui est l'outil qui effectue la vérification, répond soit que la propriété est satisfaite (vrai) pour le modèle. Soit il répond que la propriété n'est pas satisfaite et il fournit alors un contre-exemple d'exécution qui montre pourquoi elle ne peut être satisfaite. Notons que, si de manière usuelle, le model-checker s'arrête après avoir trouvé un contre-exemple pour des raisons de performances, il est possible qu'il existe plusieurs autres contre-exemples. On verra que pour l'analyse de risques, il est intéressant de fournir l'ensemble des contre-exemples puisqu'ils représentent chacun une attaque différente. Un schéma récapitulant le fonctionnement du model-checking est visible en figure 1.1.

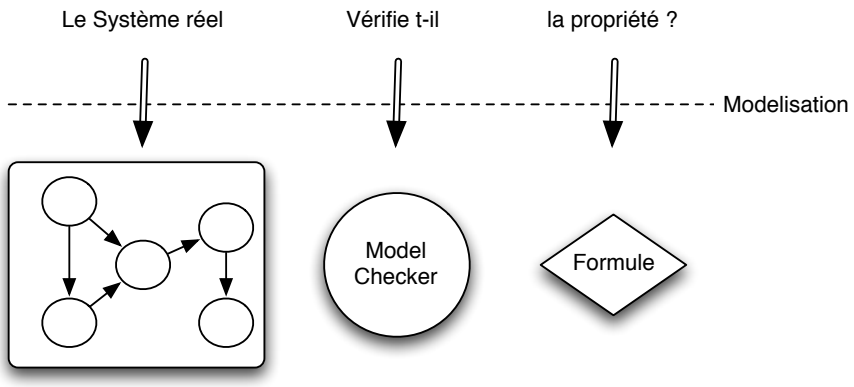

FIGURE 1.1 - *Principe de fonctionnement du model-checking*

1.4.1 Graphe

L'étape de construction du modèle vise à abstraire le système étudié vers une représentation formelle. Cette représentation est usuellement une machine abstraite que l'on nomme automate fini. Cet automate fini forme ce que l'on nomme un graphe orienté et étiqueté.

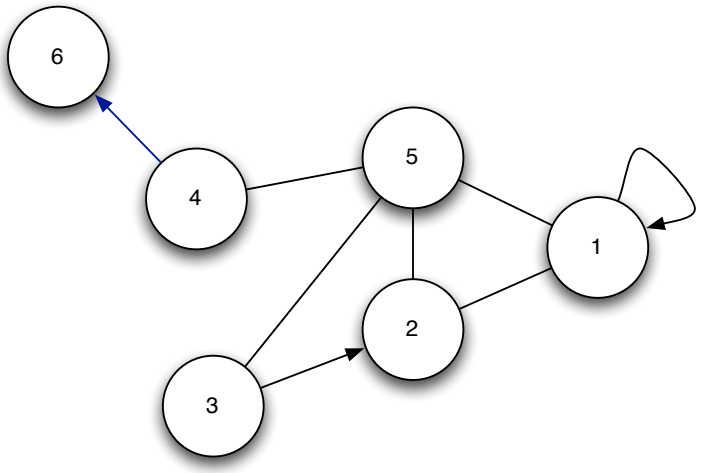

FIGURE 1.2 - *Exemple de graphe mathématique*

Un graphe G, dont un exemple est visible en figure 1.2, est constitué de deux types d'éléments : les *sommets* nommés parfois nœuds ou états, et les arêtes nommées parfois transitions ou arcs. Les sommets sont représentés sous forme de rond ou de point. L'ordre du graphe est le nombre de sommets que contient le graphe. Dans l'exemple de la figure 1.2, l'ordre du graphe est 6 puisque l'on dénombre 6 sommets.
Les arcs sont les flèches qui relient les sommets. Chaque arc a deux extrémités qui sont des sommets. On dit qu'un arc relie ou connecte deux sommets. On définit donc un arc par une paire de sommets, sur le graphe présenté en figure 1.2 on a par exemple, l'arête $(1,2)$.
On dit qu'une arête est *dirigée* si elle a un sommet de départ et un sommet d'arrivée. Dans ce cas, on la représente par une flèche entre les deux sommets. Le sommet initial est alors nommé queue et le sommet terminal tête. L'arête $(4,6)$ est dirigée, elle a pour queue le sommet 4 et pour tête le sommet 6. On appelle graphe dirigé, un graphe qui possède des arêtes dirigées. Le nœud 1 possède une boucle, c'est-à-dire une arête qui part et pointe sur lui.
L'action d'étiqueter un graphe correspond à assigner des *variables* et des *labels* aux sommets du graphe et des conditions aux arêtes du graphe que l'on nomme *gardes*. Ces gardes sont utilisées pour déterminer quelles sont les conditions à remplir pour franchir l'arête, en vérification, on nomme ce franchissement *exécuter la transition* puisque cela correspond à un changement d'état du système.

1.4.2 Structure de Kripke

Le graphe fini le plus utilisé pour modéliser les systèmes est la structure de Kripke, nommée d'après son inventeur. Une structure de Kripke est un graphe dont les sommets représentent les états du système et les arcs les transitions entre les états. Les transitions peuvent avoir des gardes qui déterminent les conditions pour lesquelles la transition est autorisée tel que un temps d'exécution maximal par exemple. Plus formellement une structure de Kripke est définie comme suit :

Définition 1.7. *Une structure de Kripke est le quadruplet* $M = (S, S_0, R, L)$ *Où* S *est l'ensemble fini des états,* $S_0 \subseteq S$ *est l'ensemble des états initiaux,* $R \subseteq S \times S$ *est la relation de transition et* $L : S \rightarrow 2^{(AP)}$ *est la fonction qui étiquette chaque état par un ensemble de propositions atomiques vrai pour cet état.*

Une proposition atomique est une proposition qui peut être uniquement soit vraie soit fausse (tiers exclu). Une proposition de manière générale est une combinaison de propositions atomiques et de connecteurs logiques. Usuellement, on note le vrai : $\top$ et le faux : $\bot$.

1.4.3 Un exemple de structure de Kripke

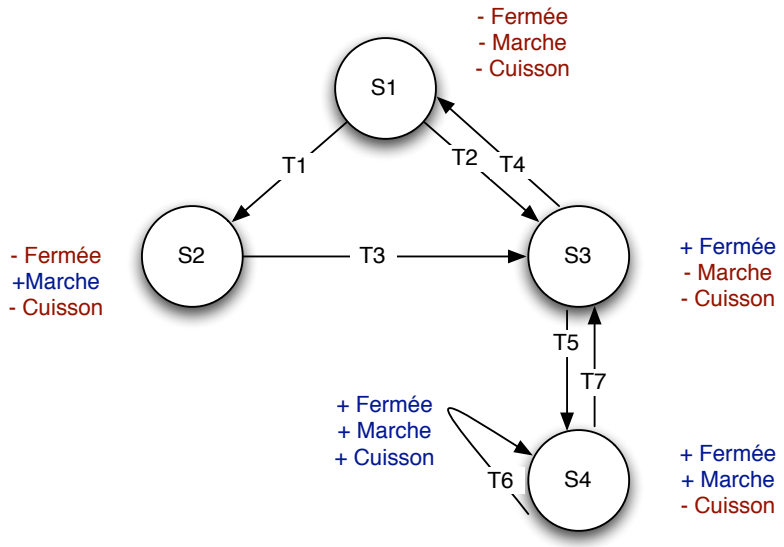

FIGURE 1.3 - *Exemple de structure de Kripke*

Si l'on reprend l'exemple du four à micro-ondes, on peut modéliser ce système par la structure de Kripke visible en figure 1.3.

On utilise trois propositions atomiques dans cet exemple :

1. **fermée** : pour indiquer si oui ou non, la porte est fermée.
2. **marche** : pour indiquer si le bouton marche a été pressé ou non.
3. **cuisson** : pour indiquer si l'aliment est cuit ou non.

La structure a quatre états $S1, S2, S3, S4$, et 7 transitions. L'état S1 est l'état initial, où les trois propositions atomiques sont fausses ($\neg fermée$ par exemple), ce qui modélise que le four est ouvert, le bouton marche est sur off ($\neg marche$) et que l'aliment n'est pas cuit. A partir de cet état, l'on peut mettre le bouton marche sur on ($marche$). Ceci est représenté par la transition T1 qui mène à l'état S2 ou le bouton marche est sur on ($marche$), la porte est ouverte ($\neg fermée$) et l'aliment n'est pas cuit ($\neg Cuisson$). De l'état S1, l'on peut aussi aller vers S3 par la transition T2 qui correspond à l'action de fermée la porte. Le franchissement de cette transition change l'état du système puisque à l'état S3 la proposition *fermée* devient vraie. Il est aussi possible d'arriver à l'état S3 en partant de S2 puisque la porte n'est pas fermée dans cet état. Ceci est modélisé par la transition T3 sur le schéma. Notons enfin que l'on peut revenir à l'état S1 depuis S3 en ouvrant la porte. Ce qui est modélisé par la transition T4.
De S2, on peut aller à l'état S3 en appuyant sur le bouton marche. Ceci est modélisé par la transition T5. La transition T6 qui est une boucle sur l'état S3 est utilisée pour modéliser la cuisson et mettre la proposition $Cuisson$ à vrai et enfin la transition T7 de S4 à S3 est utilisée pour modéliser que la cuisson est finie et que le bouton marche n'est plus enfoncé.

1.4.4 Spécifications

Une fois le système décrit, il faut spécifier quelle propriété l'on souhaite vérifier. En model-checking, les propriétés sont écrites sous forme de propriétés temporelles. Celles-ci permettent d'exprimer des propriétés qui ont besoin d'une notion d'ordre sur le temps. Ainsi, pour le micro-ondes, un exemple de propriété temporelle que l'on peut vouloir spécifier est que celui-ci ne démarre qu'**après** que la porte ait été fermée. Il y a bien une notion d'ordre temporel ici : d'abord la porte est fermée ensuite le micro-ondes démarre. Plus généralement, on définit les propriétés temporelles comme suit :

Définition 1.8. *Une **propriété temporelle** est un ensemble de comportements désirables dans le temps.*

On doit à A. Pnueli, la première spécification formelle d'une logique temporelle en 1977. Aujourd'hui, les logiques temporelles utilisent de manière classique 5 opérateurs pour parler du temps :

$$\begin{array}{rcll} \varphi & ::= & A & \text{atomic prop., dans } \mathcal{A} \\ & | & X & \text{Prochaine fois} \\ & | & F & \text{Dans le futur} \\ & | & G & \text{Globalement} \\ & | & U & \text{Jusqu'à} \\ & | & X & \text{Relâche} \end{array}$$

Ainsi que deux quantificateurs qui permettent de préciser la certitude de l'événement :

$$\begin{array}{rcll} \varphi & ::= & A & \text{atomic prop., dans } \mathcal{A} \\ & | & \Box & \text{Toujours} \\ & | & \Diamond & \text{Exist} \end{array}$$

Avec ces opérateurs et quantificateurs, il est possible de spécifier les propriétés que l'on souhaite vérifier sur le modèle. Par exemple, pour notre micro-ondes, l'on peut vouloir vérifier la propriété suivante :

$$\Box G(demarrer U Fcuisson)$$

Qui dit que, si le micro-ondes est démarré alors il finit toujours par cuire l'aliment.

1.4.5 Limitations du model-checking

Le model-checking souffre de deux limitations principales. La première, d'ordre calculatoire, est l'explosion du nombre d'états à vérifier (blow-up), l'autre d'ordre conceptuel, tient à la distorsion qui existe entre le système réel et le modèle qui le représente.

Explosion. L'analyse par model-checking impose de parcourir l'ensemble des exécutions générées par le système pour vérifier si la propriété à verifier est vraie pour chacune des exécutions. Or, cette approche est extrêmement coûteuse en terme de temps et d'espace mémoire. Comme on va le voir dans les prochains chapitres, en raison de la complexité des modèles, il arrive un moment où l'approche par model-checking échoue car il y a trop d'exécutions à vérifier. Sans rentrer dans les détails de la théorie de la complexité, on retient que plus un modèle est expressif, c'est-à-dire qu'il permet de modéliser de choses, plus il génère un grand nombre exécutions complexes à vérifier et est donc lent à vérifier. Cela revient à dire que plus un modèle est expressif, plus petits seront les systèmes formalisés dans ce modèle qui seront vérifiables. On considère aujourd'hui, que pour qu'un modèle soit utilisable en pratique, dans le pire cas, il doit avoir une complexité EXPTIME.

Ce qui correspond à une complexité de $0(2^{p(n)})$ où $p(n)$ est une fonction polynomiale de n [Pap94].

La classe EXPTIME rassemble les problèmes décidables par un algorithme déterministe en temps exponentiel par rapport à la taille de son instance. Elle est donc plus "grosse" que la fameuse classe *NP* des problèmes Non-déterministes Polynomiaux qui réunit les problèmes de décision qui peuvent être décidés sur une machine non déterministe en temps polynomial. On conjecture que la classe NP est plus "grosse" que la classe *P* des problèmes qui peuvent être décidés par une machine déterministe en temps polynomial par rapport à la taille de la donnée. Ce qui nous donne la hiérarchie suivante :

$$P \subseteq NP \subseteq EXPTIME$$

Intuitivement cette hiérarchie permet de prédire si un modèle passera à l'échelle ou pas en pratique. Ainsi, si un modèle appartient à la classe P, il passera mieux en théorie à l'échelle qu'un modèle EXPTIME. C'est pourquoi l'étude de la complexité des modèles à un rôle central dans ce travail. Notons que l'analyse de la complexité s'effectuant dans le pire des cas, la mise en pratique peut réserver de bonnes surprises : les performances sur des cas réels peuvent être meilleures que prévues (mais jamais pire). C'est par exemple le cas du langage OCaml qui, en dépit d'une borne de complexité pour le typage lors de la compilation très élevée, s'avère en pratique extrêmement rapide et efficace.

Cette différence qui existe entre la borne théorique et la mise en pratique, nous amène à la plus grande limitation du model-checking : la distorsion qui existe entre la réalité et les modèles.

Distortion. Plus la distorsion entre réel et modèle est importante et plus l'on court le risque de voir apparaître des comportements (attaques) réels non prévus par le modèle. En effet, si l'approche par model-checking permet de garantir que tout va bien dans le cadre du modèle, elle est évidemment aveugle aux comportements existants en dehors du modèle. C'est durant la phase d'abstraction du système réel vers le modèle, que l'ensemble des comportements possibles est réduit. Ainsi, pour l'exemple du micro-ondes, le modèle est impuissant à vérifier ce qui se passe si un utilisateur donne un coup de marteau sur la vitre pendant la cuisson puisque ce comportement n'est pas prévu par le modèle.

Cette notion de limitation des modèles face aux comportements imprévus (outlier) est bien connue depuis Russel ou Hume. Récemment cette notion d'événement inattendu qui invalide les résultats trouvés par un modèle a été nommée "*cygne noir*" par Taleb dans le livre éponyme [Tal07]. Ce nom vient du fait que jusqu'à la découverte de l'Australie et donc des cygnes noirs, les habitants de l' "ancien monde" étaient persuadés qu'il n'existait que des cygnes blancs.

Du point de vue de l'analyse de risques, l'existence de comportements non prévisibles nous amène à être extrêmement vigilants sur ce que nous devons considérer comme possible ou pas. En particulier, tout au long de l'élaboration du modèle, nous avons toujours pris soin de considérer que les attaquants pouvaient être en possession d'information ou de méthodes d'attaques inconnues jusqu'alors.

Ces deux limitations du model-checking sont à l'origine de la difficulté principale de ce travail, qui a été de trouver le juste compromis entre un modèle qui "doit" coller à la réalité pour limiter la distorsion qui existe entre modèle et réel et en limiter la complexité pour qu'il soit utilisable et analysable en pratique.

1.5 Défis

Comme dit en début de chapitre, l'extrême complexité des réseaux rend l'analyse de risques particulièrement difficile. Cette complexité est due aux facteurs présentés ci-dessous.

1.5.1 Le nombre d'équipements et de protocoles

Lorsque l'on s'attaque à la sécurité réseau, il faut d'abord être capable d'analyser un nombre massif d'équipements utilisant un nombre important de services. Ainsi, un réseau comportant une dizaine de milliers de machines utilisant des centaines de service, est aujourd'hui courant. Par exemple, le réseau de la Warner Bros possède plus de 35 000 machines. Manipuler un aussi grand nombre d'objets pose deux problèmes majeurs. Premièrement, cela fait exploser en termes de temps de calcul et d'espace mémoire les méthodes standards de vérification. Cette explosion combinatoire est à l'origine de l'échec de nombreux modèles basés sur les approches classiques comme on le verra au chapitre 2.
Deuxièmement, cela oblige à repenser la manière de formaliser les données de l'analyse, car il n'est pas possible, à l'instar de la vérification de protocoles cryptographiques, d'avoir un modèle explicite. En effet, il est tout simplement impossible d'écrire de manière explicite toutes les interactions entre les services car cela représenterait des dizaines, si ce n'est des centaines de milliers de lignes.

1.5.2 Acquisition des données

Le deuxième défi découle lui aussi de la taille des réseaux : la première phase de l'analyse de risques qui consiste à collecter des informations sur les actifs présents sur le réseau, et la deuxième phase qui consiste à identifier les vulnérabilités ne peuvent être accomplies à la main. Il est donc nécessaire de développer des méthodes de collecte et d'identification automatique pour espérer bâtir un modèle fiable.

1.5.3 Interactivite des réseaux

Le troisième défi provient du fait que c'est le résultat de l'interaction des utilisateurs, légitimes ou non avec ces services, qui conditionne l'état courant du réseau. Il est donc essentiel de pouvoir modéliser les actions bénéfiques et négatives des utilisateurs pour capturer la nature dynamique de l'état du réseau.

1.5.4 Dimension temporisé

Le quatrième défi découle du troisième : capturer la nature dynamique du réseau et les interactions utilisateurs nécessite d'ajouter une dimension temporisé au modèle. On parle ici de dimension temporisé et non temporelle car en model-checking, une contrainte temporelle est une contrainte de succession dans le temps alors qu'une contrainte temporisé utilise des valeurs de temps explicites. Il faut en effet prendre en compte que chaque interaction avec le réseau nécessite un temps minimum pour avoir un sens. Par exemple, l'application d'un correctif logiciel n'est pas instantanée car son application requiert, au minimum, le temps d'exécution du programme chargé de l'appliquer. On verra par la suite que tenir compte du temps permet de rendre naturellement compte de la course que se livre les administrateurs et les intrus pour la conquête du réseau (Chapitre 3).

1.5.5 Dimension relationnelle

L'interconnexion des services réseau nous amène au cinquième défi qui est de tenir compte de l'effet "boule de neige" qui survient lorsqu'un service tombe en panne ou est compromis. Comme évoqué en tout début de chapitre, ce n'est pas la panne en elle-même qui fait le plus de dégâts mais ses conséquences par *effets collatéraux*. Prendre en compte les dépendances entre les différents services est donc essentiel pour pouvoir modéliser les conséquences des attaques. Par exemple, les attaques par DDOS (Distributed Denial of Service / Déni de service distribué) , exploitent les dépendances qui existent au sein des réseaux : en rendant indisponibles les serveur DNS d'une société, elles empêchent par *effet collatéral* l'accès à l'ensemble des services de cette société tels que les services web et de messagerie en empêchant la résolution DNS dont dépendent les clients pour accéder à ces services (schéma 1.4). Cibler le point névralgique du réseau dont les autres services dépendent permet à l'assaillant de maximiser l'effet de son attaque.

1.5.6 Dimension financière

Pour un réseau donné, les services et équipements réseau n'ont pas tous la même valeur et leur indisponibilité ou compromission n'a pas le même impact. Il faut donc prendre en compte la dimension financière pour être en mesure d'évaluer l'impact sur l'entreprise des différentes attaques possibles.

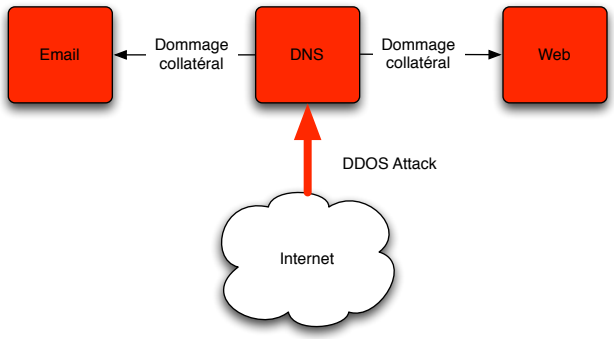

FIGURE 1.4 - *Attaque par DDOS*

Comme on le voit, les défis rencontrés lors de la création d'un modèle d'analyse de risques réseau sont multiples. Comme nous le verrons au chapitre 2, jusqu'à présent aucun framework [1] n'a été capable de relever l'ensemble des défis présentés dans cette section. En particulier, aucun framework ne prend en compte la nature dynamique du réseau.

1.6 Contenu de la thèse

Les contributions de la thèse se sont concentrées sur la création du framework des anticipation games [2]. Cette création s'articule autour de trois axes.

Sur le plan de la **modélisation**, nous avons créé un nouveau modèle d'analyse de risques pour les réseaux basés sur la théorie des jeux [Ras07] et la logique TATL [HP06] nommée **anticipation games**. Puisqu'il est le premier modèle d'analyse de risques réseau qui utilise la théorie des jeux, il se démarque des modèles précédents sur de nombreux points. D'une part, il est le premier à rendre compte de l'ensemble des dimensions évoquées précédemment. En particulier, il est le premier à être capable de modéliser la dynamique du réseau induite par l'interaction des utilisateurs.
D'autre part, il est le premier à utiliser une représentation compacte à base de règles. Cette approche offre une flexibilité et une facilité de modélisation sans précédent dans le domaine. Qui plus est, cette notion de règles est très proche de la notion de base de règles utilisées classiquement en sécurité pour les systèmes experts et les détecteurs d'intrusion, ce qui rend ce modèle intuitif aux professionnels de la sécurité ; et laisse entrevoir l'espoir d'une adoption rapide du modèle par la communauté.

1. Ce terme n'ayant pas d'équivalent en français nous avons préféré garder la version originale
2. Nous conservons le nom d'origine de notre framework car c'est son nom propre. Nous accordons le nombre pour respecter les conventions de la langue française.

Sur le plan de la **mise en œuvre**, nous introduisons les outils et techniques nécessaires pour mettre en pratique les anticipation games. Ainsi, nous avons développé le logiciel *NetQi* qui permet d'analyser automatiquement les réseaux modélisés sous forme d'anticipation game. Cet outil nous a permis de valider que notre modèle est utilisable en pratique. Nous avons aussi travaillé sur les étapes une et deux de l'analyse de risques en développant des techniques qui permettent de les automatiser. En particulier, l'on a résolu en grande partie la difficulté d'inférer les dépendances entre les services réseau de manière automatique. Ce domaine n'ayant jamais été exploré auparavant, nous avons dû inventer les techniques nécessaires pour les inférer et réaliser l'outil *NetAnalyzer* qui les implémente afin de valider l'approche. C'est un des algorithmes développé dans ce cadre qui nous a valu le "best paper award" à WISTP [Bur08c].

Enfin, nous n'avons eu de cesse d'apporter des **extensions** à notre framework et d'étudier des cas concrets pour s'approcher au plus proche de la réalité. Cet incessant travail a permis d'apporter deux extensions essentielles au framework d'origine. La première extension permet au framework de modéliser la dimension financière grâce à l'ajout de la notion de coût et de gain. Le bénéfice majeur de cet ajout est qu'il permet de raisonner, non plus en terme de propriété à prouver, mais en termes d'objectifs stratégiques à atteindre. On verra que cette notion permet d'exprimer naturellement les questions clés soulevées par l'analyse de risques. La deuxième extension majeure est l'ajout de la notion d'emplacement utilisée pour modéliser la coopération entre réseaux qui joue un rôle majeur aujourd'hui dans le combat contre les menaces inconnues, les botnets et les vers. Cette extension permet aussi de lier de manière plus étroite les dimensions temporelles et financières grâce à l'ajout des pénalités. Celles-ci permettent en effet d'exprimer les coûts liés au temps, et par extension de modéliser que certains coûts diminuent avec le temps.

Ce travail ne pourrait être complet sans les modélisations de cas concrets que nous avons réalisé afin de mettre en lumière l'adéquation de notre framework avec la réalité et sa capacité de passage à l'échelle. C'est pourquoi l'ensemble des exemples présentés dans cette thèse, sans exception, ont été réalisés grâce à notre outil *NetQi*.

1.7 Plan de la thèse

Cette thèse est composée de trois parties reprenant les trois axes de contribution. D'une part, la comparaison des frameworks existants et l'introduction du notre, d'autre part la mise en œuvre de notre framework pour des réseaux réels, et enfin les extensions que nous avons apportées à celui-ci pour coller à la réalité.

1.7.1 Modèles

Dans la partie *Modèles* de la thèse, nous revenons sur les modèles existants pour l'analyse de risques réseau ainsi que les fondements théoriques qui sont à la base de notre modèle, puis nous introduisons notre modèle à base de théorie des jeux.
Dans le **chapitre 2**, nous présentons plusieurs modèles représentatifs de ce qui est utilisés actuellement en analyse de risques réseau et discutons des avantages et inconvénients de chaque approche.
Le **chapitre 3** introduit notre modèle à base de théorie des jeux ainsi que les preuves de décidabilité associées, servant de fondement théorique à notre framework d'analyse de risques.

1.7.2 Mise en œuvre

La partie mise en œuvre est consacrée à la mise en pratique de notre framework sur des réseaux réels et les problèmes que cela engendre.
Ainsi le **chapitre 4** est consacré à la mise en pratique de la vérification des modèles de jeux grâce à notre outil *NetQi*. Dans ce chapitre l'on reviendra sur l'élément de surprise présent dans le modèle au travers d'un exemple écrit dans la syntaxe de *NetQi*.
Dans le **chapitre 5** nous revenons sur la mise en œuvre de l'étape un et deux de l'analyse de risques, à savoir, la collecte d'information sur les actifs et les vulnérabilités réseau. En particulier, nous introduisons les techniques qui permettent de collecter de manière automatique les dépendances existantes entre les différents services du réseau et l'on y présentera notre outil d'analyse de réseau *NetAnalyzer*.

1.7.3 Extensions et application

Cette dernière partie revient sur les extensions que nous avons apportées au framework tout au long de la thèse afin d'enrichir ses capacités d'analyse et présente quelques-unes des applications du framework les plus intéressantes.
Le **chapitre 6** est consacré à la recherche de stratégie optimale de défense et d'attaque. Ce chapitre détaille en quoi l'introduction de la dimension financière des attaques via la notion de coût et de gain permet de préciser des objectifs d'analyse qui sont bien plus en adéquation avec l'analyse de risques que la vérification de propriétés utilisées de manière classique en vérification.
Le **chapitre 7** est consacré à la coopération entre différents réseaux pour combattre les attaques inconnues. Ce chapitre revient aussi sur la relation qui existe entre le temps et les coûts en proposant l'ajout de pénalités au framework. Celles-ci permettent notamment de modéliser les coûts engendrés par l'indisponibilité des services.

Les chapitres 3, 6, 7 ont nécessité la réalisation de l'outil *NetQi* [Bur] qui fait environ 6 000 lignes de C sans compter l'interface graphique réalisée en Java. Le chapitre 5 a nécessité la réalisation de l'outil *NetAnalyzer* [Bur06] qui fait plus de 20 000 lignes de C sans compter les différents scripts utilisés pour la génération des règles.

Le chapitre 3 fait l'objet des publications [Bur07b, BGL07]. Le chapitre 4 fait l'objet de la publication [Bur08b]. Le chapitre 5 fait l'objet des publications [Bur08c, Bur07a], le chapitre 6 fait l'objet de la publication [BM09] et enfin le chapitre 7 a fait l'objet de la publication [Bur08a].

Première partie

Modélisation

CHAPITRE 2

Modèle pour l'analyse de risque

"If you know the enemy and know yourself, you need not fear the result of a hundred battles."

Sun Tzu, The Art of War. III.18

Dans ce chapitre nous revenons sur les frameworks précédant notre travail afin d'esquisser le contexte dans lequel il s'insère. Cet état de l'art va ainsi nous permettre de mettre en lumière son apport par rapport aux travaux existants. Un des travaux les plus pertinents pour comprendre l'évolution des modèles est la classification de Baskerville [Bas93]. Cette classification distingue trois générations successives de méthodes d'analyse de risques qui diffèrent par le niveau d'abstraction qu'elles offrent par rapport à la définition du système et de ses risques. Cette classification datant de 1993, nous avons rajouté une quatrième génération à cette classification qui correspond aux frameworks actuels.

Ce chapitre est composé de trois parties. Dans la première partie, nous présentons les quatre générations de frameworks et discutons de leurs avantages et inconvénients. Puis, dans la deuxième partie, nous détaillons les principaux frameworks de quatrième génération, ainsi que le modèle biologique SRI. Enfin, dans la troisième et dernière partie, nous comparons les différents frameworks afin de mettre en lumière les forces et les faiblesses des travaux précédant le nôtre. Nous concluons en récapitulant les points clés du chapitre.

2.1 Historique

2.1.1 Première génération

La première génération de modèles se fonde sur trois hypothèses. Premièrement, il existe une liste exhaustive de toutes les parades nécessaires à la protection des données. Deuxièmement, chacune de ces parades est applicable à tout système. Ce qui implique l'universalité des solutions à apporter aux objectifs de protection. Troisièmement, le but ultime de l'application d'une méthode est de mitiger la probabilité d'occurrence d'une attaque ou le coût résultant de l'attaque.

Cette génération s'applique à toutes les méthodes se basant sur des listes de contrôles telles que *SPAN* [ZHM90]. La principale limitation de cette génération de modèles est qu'elle est totalement dépendante d'un ensemble restreint de solutions techniques qu'elle essaye d'appliquer à tous les cas.

2.1.2 Deuxième génération

La deuxième génération est marquée par une volonté de décomposition du système en sous-systèmes plus facilement analysables. Trois principes fondateurs sont à l'origine de cette génération. Premièrement, les éléments d'un système étant complexes et interconnectés, il est nécessaire de partitionner le système pour pouvoir les analyser efficacement. Deuxièmement, les parades sont spécifiques à chaque sous-système. Troisièmement, la sécurité dépend de la maintenance ; elle est sensible à la modification du système.

Un changement majeur intervient avec cette nouvelle génération de modèles : il est désormais admis qu'il faut pouvoir ordonner les parades en fonction de leur efficacité. Ce changement va rendre nécessaire l'utilisation d'une algèbre et de méthodes d'ordonnancement. C'est celle qui, dans la classification de Baskerville, colle le mieux aux 6 phases de l'analyse de risques présentée en introduction (section 1.1.1). Le gros problème de cette approche est le manque de vision globale du système, qui empêche de prendre en compte les relations de causalités. Les méthodes *CRAMM* [Far91] et *RISKPAC* [Gar89] appartiennent à cette génération.

2.1.3 Troisième génération

La troisième et dernière génération présentée par Baskerville adopte un plus haut niveau d'abstraction par rapport au système. Elle corrige le point faible de la génération précédente, à savoir que le système est de nouveau traité globalement. Ce qui permet d'avoir une vision global du système. Cette génération se base sur les cinq hypothèses suivantes : premièrement, la solution de protection idéale ne peut être trouvée qu'à partir d'une compréhension du système global.

Deuxièmement, seul un niveau d'abstraction suffisant permet d'envisager le système dans sa globalité. Troisièmement, les solutions dérivées d'une telle vision sont plus flexibles et faciles à adapter en cas de modification du système. Quatrièmement, il n'existe pas de solution universelle. Cinquièmement, l'utilité de la notion de coût/bénéfice est inversement proportionnelle au niveau d'abstraction utilisé.

La méthode SSADM-CRAMM [Con93] appartient à cette génération. On peut aussi, comme le remarque [Dac94], ranger dans cette génération toutes les méthodes d'évaluation classiques utilisées pour la sécurité de fonctionnement comme les chaînes de Markov et les réseaux de Pétri stochastiques puisqu'elles offrent une méthode de modélisation abstraite et une capacité à générer des évaluations qualifiées sur ces représentations globales. La montée en niveau d'abstraction de la troisième génération de méthodes et l'ouverture vers les méthodes de modélisation abstraite vont être à l'origine de la quatrième génération de méthode, qui a vu le jour après la publication de cette classification.

2.1.4 Quatrième génération

La quatrième génération (4G) de modèles est aussi la génération actuelle. On peut la surnommer génération "modèle formel" car elle vise à appliquer les connaissances acquises pour la vérification et spécification des systèmes à l'analyse de risques. Ainsi, on retrouve à la base de cette approche les cinq fondements suivants : premièrement, les objectifs d'une analyse de risques peuvent être modélisés comme des propriétés à vérifier. En particulier, les buts de l'attaquant sont modélisés grâce à des propriétés dites de *sûreté (safety)*. Deuxièmement, la vérification des propriétés doit être effectuée automatiquement, notamment grâce au model-checking. Troisièmement, l'utilisation d'une logique pour décrire le modèle permet d'être flexible et général. Quatrièmement, la taille des réseaux augmentant de manière drastique, il devient nécessaire de construire le modèle de manière aussi automatique que possible. Cinquièmement, la complexité du modèle doit être prise en compte pour garantir le passage à l'échelle. Pour cette quatrième génération, on ne parle plus de méthode mais de framework car désormais les modèles phares de cette génération sont accompagnés de logiciels qui implémentent l'analyse du modèle et parfois même sa construction.

Définition 2.1. Framework *Un framework pour l'analyse de risque réseau est composé du modèle théorique et des outils permettant la mise en pratique de ce modèle.*

Les prémisses de la quatrième génération *4G* remonte à 1994 quand Baldwin, dans un rapport du MIT [Bal94], présente *Kuang* un outil pour vérifier la configuration d'une machine de manière automatique. Ce travail est repris en 1996 par Zerkle *et al.* [ZL96] dans l'outil NetKuang qui permet de vérifier plusieurs machines. Pour cela, ils utilisent *rsh* et traitent uniquement les configurations locales.

NetKuang utilise une analyse en arrière qui, à partir d'un objectif, recherche les étapes pour l'atteindre et retourne la liste des étapes à accomplir. Toujours en 1996, Dacier *et al.* [DDK96] proposent les *privilege graphs* pour l'analyse de sécurité. Ce modèle sera mis en pratique en 1999 [ODK99] par Ortalo *et al.*

En 1998, Phillips *et al.* [PS98] proposent d'utiliser une analyse en avant pour vérifier des propriétés de sûreté. En partant d'un nœud d'origine, on parcourt le graphe pour trouver tous les nœuds où la propriété est violée. Le graphe d'attaques est l'union de tous les chemins menant à cette violation. Swiler et al [SPEC00] utilisent cette méthode pour construire les attack graphs dans leur outil en 2000. Ce framework ne sera utilisé que sur de petits exemples. Toujours en 1998, un framework [RS98] plus général que celui de Phillips *et al.* basé sur le model-checking voit le jour, c'est lui qui popularisa le terme d'*attack graph.* On verra dans la section suivante que ce framework est l'un des frameworks majeurs de la 4G.

L'année 1999 verra Bruce Schneier proposer de manière informelle la notion d'attack tree [Sch99] pour l'attaque de coffre-fort. En 2000 Templeton *et al.* [TL00] proposent un modèle nommé *requires/provides* qui modélise des scenarios d'attaques enchaînant les exploits réseau. La partie *requires* représente les préconditions nécessaires à l'exécution de l'exploit et la partie *provides* représente les postconditions qui définissent les effets de l'attaque. De manière surprenante, ils ne proposent pas de combiner les scénarios générés pour construire un graphe d'attaques.

Utiliser la notion de scénario d'attaques pour la détection d'intrusion a donné lieu en 2000, d'une part à la création du langage LAMBDA [CO00] et d'autre part à la réalisation de l'IDS STAT [VEK00, Por92], en perl, qui utilise le langage STATL [EVK02]. Les recherches sur la possibilité de calculer le coût des attaques de manière automatique donna lieu à la création d'une première métrique en 2006 par Metha *et al.* [MBZ+06]. En 2007, Ou et Sawilla proposèrent une seconde métrique pour le calcul automatique de coût nommé AssetRank dans un rapport de recherche de la défense canadienne [SO07]. Celle-ci se base sur les relations de dépendances existantes dans le réseau.

Tous les frameworks de 4G sont confrontés au double défis d'avoir un fondement rigoureux tout en étant applicables en pratique. C'est pourquoi nos critères de comparaison s'articulent autour de ces deux axes : expressivité du modèle et mise en pratique. Ainsi notre comparaison est découpée en trois parties. Premièrement, les frameworks considérés sont présentés avec un accent mis sur leurs spécificités et en quoi ils ont contribué au domaine. Puis l'on s'intéressera à l'expressivité de leurs modèles, et enfin l'on étudiera leur capacité à être mis en pratique.

Notre comparaison ne prétend pas être exhaustive. En effet, comme nous l'avons vu de nombreux travaux tournent autour de la notion de chemin d'attaques, et peuvent être de près ou de loin assimilés à des frameworks.

C'est pourquoi, plutôt que de tenter un catalogage qui serait forcément incomplet, nous avons décidé de nous concentrer sur les frameworks qui ont contribué au domaine de manière significative en introduisant un nouvel aspect dans la modélisation, une nouvelle technique d'analyse ou une avancée en matière d'application pratique. En plus de ceux-ci, nous avons inclus le modèle SRI utilisé en biologie pour la modélisation des épidémies de virus car comme on le verra, sa modélisation des interactions est intéressante.

2.2 Quelques frameworks de quatrième génération

2.2.1 Attack Tree

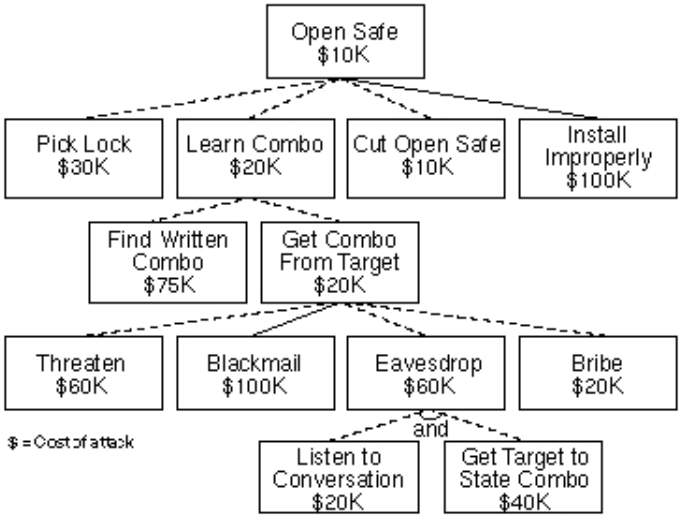

FIGURE 2.1 - *L'attack tree utilisé comme exemple par Schneier en 1999*

En 1999, Bruce Schneier, mieux connu pour ses travaux en cryptographie [Sch96], publie un article informel [Sch99] qui modélise l'attaque d'un coffre-fort sous forme d'un arbre nommé *Attack tree*. Cet arbre sert à modéliser les différentes étapes qui mènent à l'ouverture du coffre-fort. Parmi ces étapes on trouve des "attaques intermédiaires", telles que *écouter les conversations*, *percer le coffre*, ou encore *faire du chantage* (figure 2.1). Dans la dernière partie de l'article la notion de coût est abordée, Schneier propose de s'intéresser au chemin qui offre le plus grand profit et par là même rend compte de la dimension financière de l'attaque. Cet article est souvent considéré comme un précurseur [LI05] de la 4G car il rend l'ensemble des chemins explicites et ne se contente pas de proposer un seul chemin contrairement aux autres travaux de l'époque.

2.2.2 Privilege graph

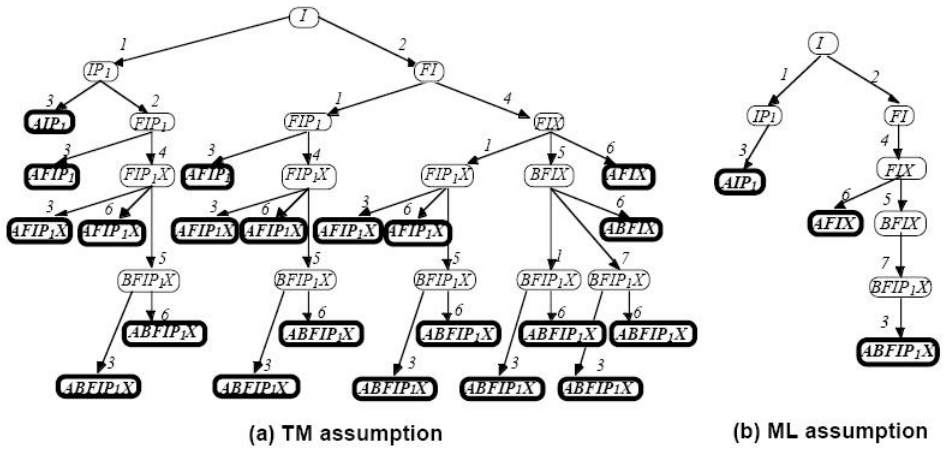

FIGURE 2.2 - *Le privilege graph utilisé comme exemple par Ortalo en 1999*

En 1999, Ortalo *et al.* [ODK99] proposent l'une des premières, si ce n'est la première, tentative de mise en pratique d'un modèle formel. Le modèle utilisé est une version avec des contraintes relâchées des *privilege graphs* inventés par Dacier [DDK96, Dac94] où les nœuds représentent les privilèges et les arcs les vulnérabilités. Ces graphes sont utilisés pour construire un *attack state graph* qui représente les différentes manières pour un attaquant d'atteindre un objectif particulier, tel que l'accès administrateur. Le *privilege graph* utilisé comme exemple dans le papier de 1999 est repris sur la figure 2.2. Ce modèle possède une notion de coût basé sur une métrique probabiliste appelée *mean effort to failure* ou METF.

On peut donc considérer ce Framework comme l'un des précurseurs du domaine avec NetKuang puisqu'il mêle à la fois modèle formel et application pratique. On trouve dans ce modèle les deux composants centraux des modèles à venir qui vont permettre l'automatisation des analyses : le modèle qui décrit les attaques et le but d'analyse à fournir à l'analyseur. Cette première tentative met en exergue l'une des difficultés majeures du domaine : les approches par méthodes formelles ont beaucoup de mal à passer à l'échelle. Ainsi, les auteurs reconnaissent que leur framework ne passe que sur de petits exemples (page 13 avant dernière ligne) et que parfois leur outil est incapable de construire le modèle.

2.2.3 Attack Graph

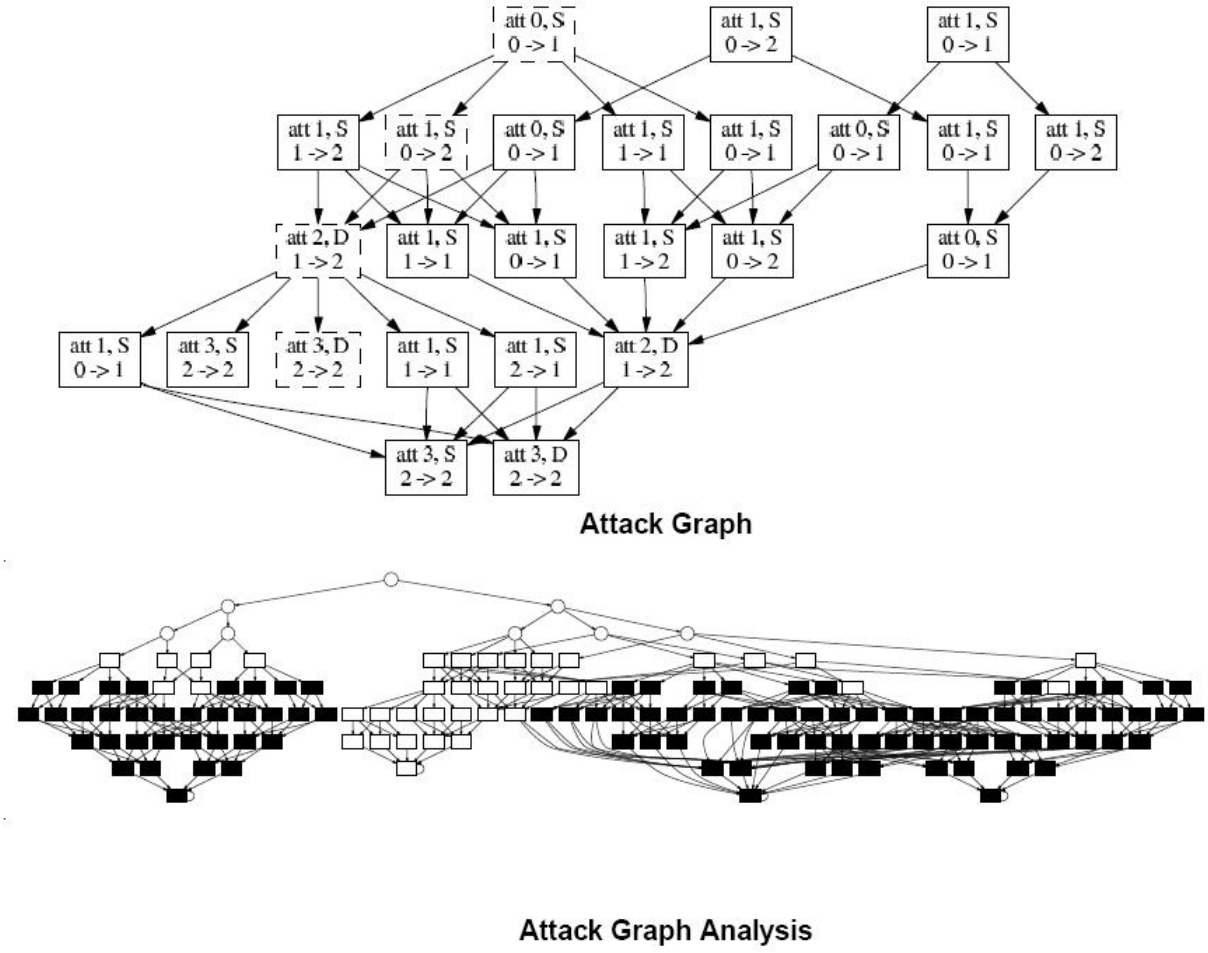

FIGURE 2.3 - *Exemple d'attack graph et de scenario graph extrait de la thèse de Sheyner 2004*

En 1998, Ramakrishnan et Sekar [RS98] utilisent le model-checking pour analyser la configuration d'une seule machine. La contribution majeure de ce premier papier a été d'abstraire le but d'une analyse de risques sous la forme de la vérification d'une *propriété d'accessibilité* , ce qui permet d'utiliser le model-checking pour l'analyser automatiquement.

En 2000, Ritchey et Ammann [RA00] vont plus loin et utilisent le model-checking pour construire un unique scenario d'attaques non limité à une seule machine. C'est ce papier qui popularisa le terme d'*attack graph*, qui fût par la suite repris par la communauté pour désigner les frameworks de 4G. Pour modéliser les attaques, ce modèle utilise les notions de *préconditions* et *postconditions*. Les préconditions servent à décrire les conditions nécessaires pour exécuter l'attaque et les postconditions servent à décrire les effets de l'attaque. Il est difficile de savoir quel modèle utilisa ces notions en premier. On la retrouve dans de nombreux papiers de cette époque tels que le modèle Lambda [CO00] qui s'intéresse à la corrélation entre modèle d'attaque et alerte de détecteur d'intrusion. Puisque un seul contre-exemple est généré par l'analyse, on ne parle pas encore de graphe d'attaques.

En 2002, Ammann *et al.* [AWK02] améliorent le passage à l'échelle de la méthode en proposant un algorithme de résolution ayant une complexité en $O(N^6E)$ où N est le nombre de machines et E le nombre d'exploits. Cet algorithme s'appuie sur une propriété de monotonie qui considère que les préconditions nécessaires à l'exécution des attaques ne sont jamais altérées. Il est important de noter ici que l'utilisation de cette propriété de monotonie élimine de facto toute possibilité de modéliser l'interaction des utilisateurs qui modifierait l'état du réseau de sorte à par exemple rendre impossible une attaque jusque la réalisable.

Toujours en 2002, Sheyner *et al.* [SHJ+02a, JSW02] présentent un framework complet basé sur les attack graphs. Ils utilisent le scanner de vulnérabilités Nessus [Der] pour construire le graphe d'attaques de manière automatique et le model-checker NuSmv [CCGR00, CCG+02] dont la version 2 vient de sortir pour analyser le graphe. On notera que le choix de NuSmv est probablement dû au fait que le Framework et NuSmv ont été développés à la même période dans le même laboratoire à Carnegie Mellon. La grande nouveauté, par rapport aux travaux précédents, est qu'au lieu de fournir un seul contre-exemple, cette fois l'ensemble des chemins d'attaques est généré. Ce framework travaille donc sur un attack graph complet aussi nommé *scenario graph* qui représente tous les chemins d'attaques possibles. Un exemple d'attack graph et de scenario graph issus de la thèse de Sheyner est reproduit en figure 2.3.

Si les performances, pour un petit modèle, sont acceptables : 5 secondes pour 4 machines/ 4 vulnérabilités, elles explosent rapidement, ainsi il faut 2 heures pour 5 machines et 8 vulnérabilités. Notons que le framework des attack graphs permet de tenir compte des contraintes topologiques d'accessibilité via les préconditions.

Les attack graphs constituent le premier framework complet à base de model-checking où la complexité est étudiée comme un facteur de réussite critique. C'est une approche modulaire utilisant des outils libres tels que Nessus ou NuSmv avec l'espoir de profiter de leurs avancées. C'est aussi le premier Framework à posséder une interface graphique.

L'ensemble des outils développés pour ce Framework est toujours maintenu puisque la dernière version date de février 2007. Enfin, notons que, conscient des limitations de son modèle, Sheyner propose dans les directions futures de sa thèse d'utiliser la théorie des jeux afin de rendre le modèle dynamique.

2.2.4 MultiVal

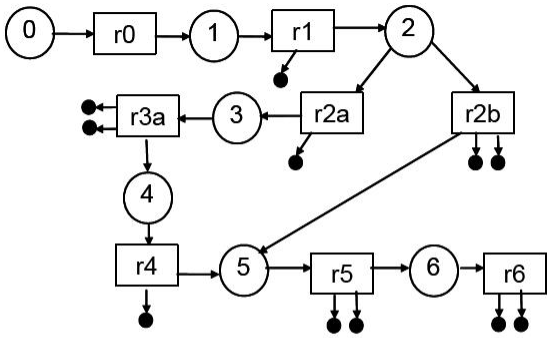

FIGURE 2.4 - *Exemple de logical attack graph extrait du papier de Ou* et al.

En 2005, Ou *et al.* présentent *MulVal* à la conférence Usenix Security [OG05]. Mulval a la particularité d'utiliser Datalog qui est un sous-ensemble de prolog destiné aux bases de données comme langage de modélisation. L'objectif des auteurs est de fournir un framework qui soit automatique et qui passe à l'échelle.

Ce framework possède, à l'instar du framework présenté précédemment deux logiciels. D'une part *Mulval scanner* , qui permet d'analyser la configurations de machines Linux et d'autre part *MulVal reasoning engine* qui raisonne sur les configurations. Le temps d'exécution pour l'analyse d'une machine est de 236 secondes et la *reasoning engine* est capable de traiter un exemple de 2 000 machines en 16 secondes environ. Notons que l'exemple utilisé comme benchmark est composé de 2 000 machines similaires construit en recopiant la même configuration et non d'un cas réel.

En 2006, Ou *et al.* proposent une évolution de MulVal nommée *logical attack graph* [OBM06] qui rend explicite les dépendances entre les attaques et la configuration des machines.

Ainsi, ce framework se démarque des attack graphs en exprimant de manière claire les relations de causalité qui existent entre les informations relatives à la configuration des systèmes et les droits potentiels que l'attaquant peut obtenir. Pour y arriver les auteurs utilisent deux types de sommets : les *facts nodes* et les *derivation nodes* . Les *facts nodes* étant eux-mêmes divisés en *primitive facts nodes* et *derivation fact nodes*. Les arcs entre les sommets représentent les dépendances entre les nœuds. L'exemple présenté dans le papier est repris en figure 2.4. Du point de vue pratique, la complexité de génération du graphe est annoncé par les auteurs comme étant de $O(N^2)$ mais en réalité elle est probablement de $O(N^m)$ où m est l'arité maximal des prédicats Datalog utilisés. Les études de cas du papier indiquent qu'analyser 1 000 machines et 100 vulnérabilités prennent environ 1 000 secondes, ce qui est beaucoup plus que le temps requis par la version précédente. Ce framework, en permettant de modéliser un gros réseau, fait apparaître un autre problème de passage à l'échelle : l'espace mémoire requis pour stocker la modélisation. Ainsi, pour 1 000 machines, leur outil a besoin d'1 GO de mémoire.

2.2.5 NetSPA : A Network Security Planning Architecture

En 2002, suite à la collaboration avec CMU pour développer les attack graphs, le Lincoln Laboratory du MIT démarre le projet NetSpa [Art02] qui vise à rendre les attack graphs applicables en pratique. L'outil NetSpa est réalisé en C++ et, dans sa première version, est capable d'analyser 17 machines / 21 vulnérabilités en 90 secondes. Par rapport aux attack graphs, ce framework introduit un certain nombre de nouveautés. Premièrement, les relations de confiance sont prises en compte partiellement, ce qui permet de modéliser les attaques par analyseur réseau (sniffer). Deuxièmement, NetSpa corrèle de multiples sources d'information pour construire ses attack graphs. Parmi ces sources, on retrouve les analyseurs de vulnérabilités mais aussi, pour la première fois, les règles de firewalls et les alertes des détecteurs d'intrusion. Enfin, l'analyse gère les buts multiples au lieu d'un but unique comme ce fut le cas pour les attack graphs. Notons que de l'aveu même de l'auteur, un certain nombre d'information doivent être entré à la main.

En 2006, une nouvelle version du framework est présentée à MILCOM [LIP+06] et à ASCAC [ILP06]. Cette version est capable d'analyser des réseaux simulés ayant jusqu'à 50 000 machines en moins de 4 minutes. Ils rapportent aussi l'analyse d'un réseau réel de 250 machines où un défaut de configuration a été trouvé grâce à NetSPA. Pour arriver à ce résultat, NetSpa utilise des *predictive graphs* qui sont des attack graphs simplifiés grâce à une hypothèse de monotonie similaire à celle utilisée par Ammann *et al.* [AWK02] pour les attack graphs évoqués précédemment.

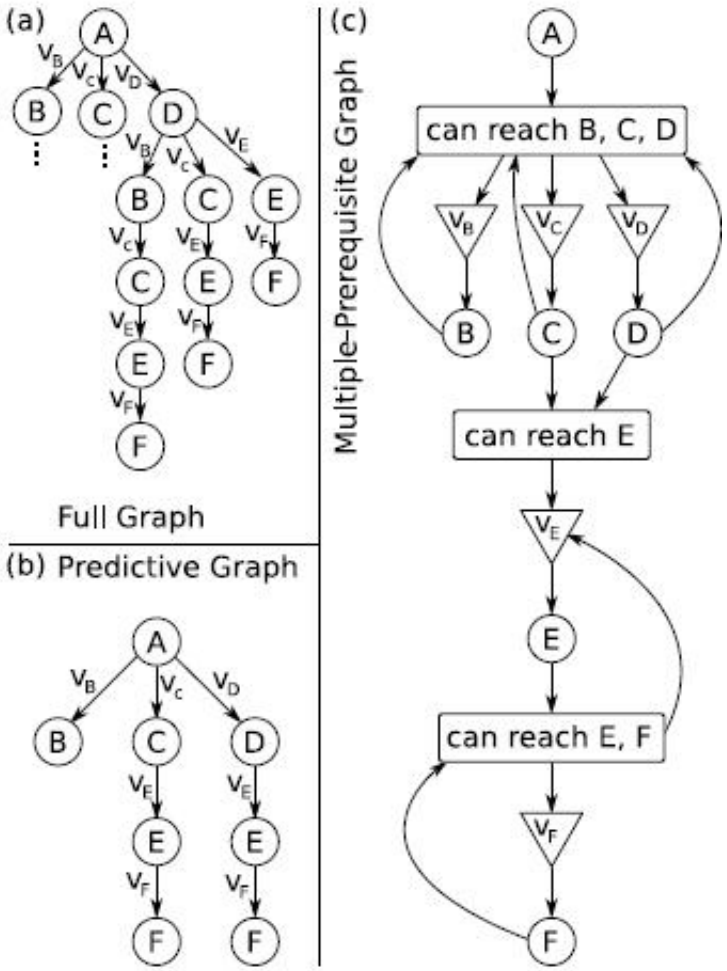

FIGURE 2.5 - *Exemple de predictive graphs extrait du papier de 2006*

Un exemple de *predictive graph* issue de l'article [ILP06] est visible en figure 2.5. On y voit les multiples étapes de simplification. A l'heure actuelle, NetSpa est l'analyseur le plus performant. Une des spécificités de NetSPA était d'utiliser une base de données pour stocker son modèle. Cela rendait la construction et la manipulation du modèle difficile. Récemment (2008) ce système à été remplacé par un stockage sur fichier qui de l'aveux des auteurs est bien plus efficace. NetSPA étant un produit du Lincoln Laboratory, il est impossible d'avoir accès au logiciel en dehors des USA pour des raisons de classification.

2.2.6 Un framework de biologie : le modèle SRI

FIGURE 2.6 - *Le modèle de propagation d'épidémie SIR*

Le modèle SRI a été développé en Angleterre entre 1927 et 1933 par Kermack et McKendrick [OM27, KM32, KM33]. Ce modèle propose d'étudier la propagation des virus en divisant la population en trois groupes : **S**usceptible, **I**nfecté, et **R**établi. Rétabli ici voulant dire soit guéri soit mort. Le passage pour chaque personne d'un groupe à l'autre étant dépendant d'un temps de transition (schéma 2.6). Ce modèle ne tient pas compte des vaccins puisque ceux-ci seront introduits en Angleterre en 1968. Le modèle SRI est dynamique puisque les sujets passent d'un état à l'autre.

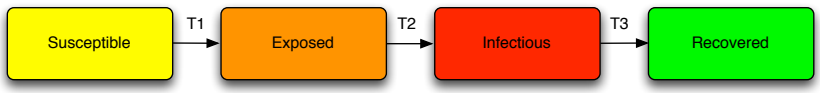

FIGURE 2.7 - *Le modèle de propagation d'épidémie SEIR*

Ce modèle va connaître de nombreuses évolutions. Parmi celles-ci, celle qui se rapproche le plus des attaques informatiques est le modèle dit SEIR : **S**usceptible, **E**xposé, **I**nfecté, et **R**établi (schéma 2.7). Un sujet est susceptible d'attraper le virus, puis il se retrouve exposé au virus, puis il devient infecté et enfin il se remet ou meurt. Ce cycle se traduit en sécurité réseau par : un service est vulnérable, l'attaquant utilise un exploit, le service est compromis par l'attaquant, et enfin l'administrateur rétablit le service (schéma 2.8).

On peut pousser l'analogie plus loin puisque de manière similaire à la biologie où se remettre d'une infection ne veut pas dire être immunisé, rétablir un service ne veut pas dire le rendre non vulnérable. Il est aussi possible de faire le parallèle entre les vaccins et les patchs informatiques : les vaccins protègent contre les virus et les patchs contre les vulnérabilités. Ce modèle semble donc pertinent pour modéliser l'interaction des attaquants et des administrateurs avec le réseau.

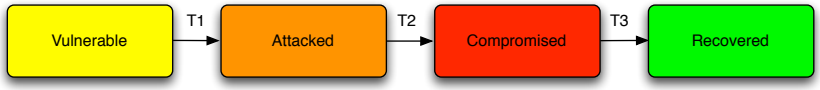

FIGURE 2.8 - *Le modèle SEIR adapté aux attaques réseaux*

Ce modèle va s'enrichir d'une dimension relationnelle en 1985, lorsque Revachev et al [RL85] démontrent qu'il est possible de reconstituer de manière rétrospective la propagation de la pandémie de myxovirus influenza (grippe) de 1968 au USA en tenant compte de son point d'origine (Hong Kong) et du réseau aérien. Le réseau aérien étant le facteur dominant de la propagation des épidémies de nos jours. On estime que cette pandémie a fait 500 000 morts et a infecté environ 50 millions de personnes aux Etats-Unis.

En 1994, ce modèle de prédiction est repris et adapté à l'Europe par Flahault *et al.* [FDV94]. Plus récemment, en 2006, ce modèle est généralisé et mis à jour grâce aux données provenant de la base de données de l'association de transport international et des recensements de population par Colizza *et al.* [CABV06]. On observe d'ailleurs, pour ce modèle, la même tendance que pour les modèles 4G, à savoir une volonté d'automatisation de l'analyse. Notons que ce modèle a également été exploité pour modéliser la propagation du SIDA [FV91] en 1991 et du virus SARS [HBG04] en 2004.

Ce modèle étendu est intéressant car il prend en compte les interactions entre les acteurs, la dimension temporelle et la dimension relationnelle [MPN+05]. Les interactions représentent le contact entre les groupes de population du modèle SRI. La dimension temporelle est présente pour modéliser deux facteurs, d'une part dans la durée des vols et d'autre part dans le temps d'incubation, de contagion et d'infection du virus. Enfin, le modèle utilise une dimension relationnelle pour modéliser la relation de dépendance qui existe entre la propagation du virus et les interconnexions aériennes qui sont représentées sous forme de graphe.

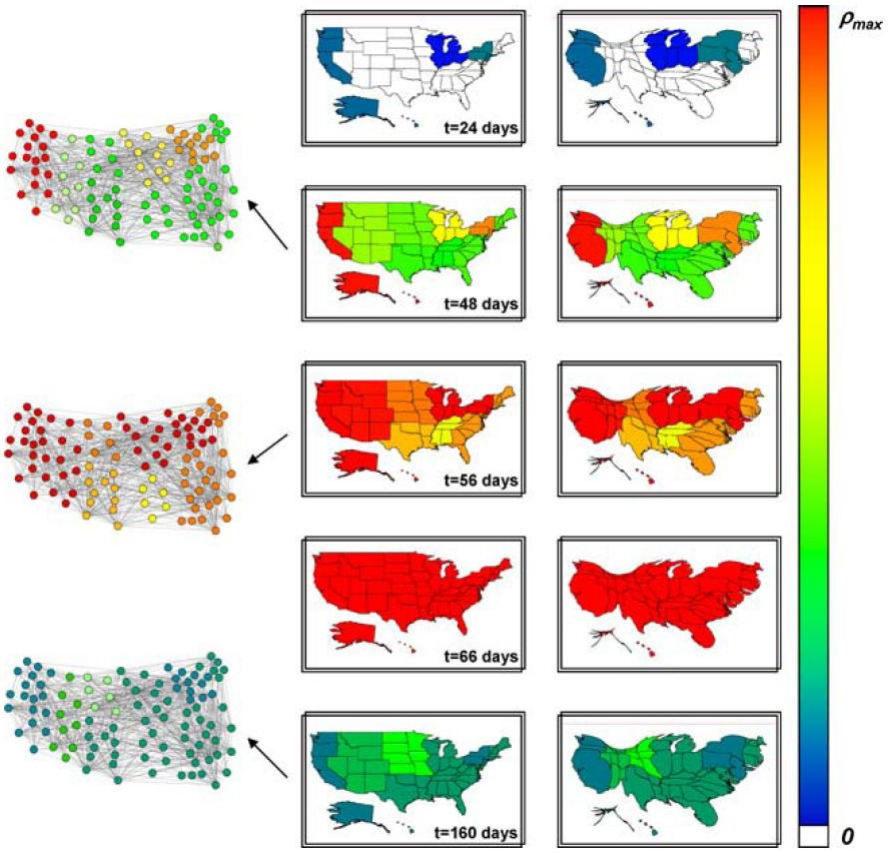

FIGURE 2.9 - *Modélisation de la propagation d'une épidémie de grippe aux Etats-Unis qui tient compte des interconnexions aériennes*

Un exemple de propagation d'épidémie de grippe au Etats-Unis tiré de [CABV06] est visible sur le schéma 2.9. Ce schéma est une *"heat map"*, ce qui veut dire que plus un état est chaud (rouge) plus le nombre de personnes infectées est grand. Ainsi, lorsqu'au jour 66 l'épidémie atteint son pic l'ensemble des états est coloré en rouge.

2.2.7 Récapitulatif

La liste ci-dessous résume les spécificités de chacun des frameworks présentés ci-dessus. On remarquera que la nature des contributions est relativement hétéroclite et que jusqu'à présent aucun framework n'a su s'imposer comme le framework de référence dans le domaine.

- **Attack trees [Sch99]** (1999) : Article informel sur l'enchaînement des attaques qui utilise une notion de coût.
- **Privilege graph [ODK99, DDK96]** (1994 – 1999) : Première tentative d'application d'un modèle formel.
- **Attack graph [RA00, AWK02, SHJ+02a, JSW02]** (2000 – 2004) : Premier framework à modéliser le but de l'analyse comme une propriété d'accessibilité, à utiliser le model-checking, à étudier formellement la complexité et à avoir une approche pratique modulaire basée sur des logiciels libres.
- **MulVal [OG05, OBM06]**(2005 – 2008) : framework rendant explicite les relations de causalité, et utilisant comme langage de spécification Datalog. C'est un des premiers framework à avoir des performances qui le rendent utilisable en pratique.
- **NetSpa [Art02, LIP+06, ILP06]** (2002 – 2007) : NetSpa est l'évolution des attack graphs. Réutilisant l'ensemble des travaux passés, en particulier les techniques de simplification basées sur les hypothèses de monotonie, ce framework se focalise sur les performances et le passage à l'échelle.
- **SRI [OM27, RL85, CABV06]** (1927 – 2008) : Le modèle SRI, bien que très ancien, offre un modèle de contagion particulièrement adapté aux épidémies et aux attaques informatiques. Ceci est dû au fait que c'est un modèle dynamique qui mêle interaction, dimension temporelle et dimension relationnelle.

2.3 Comparaison des frameworks

Nous allons maintenant comparer les frameworks présentés dans la section précédente sur les deux axes principaux de développement des frameworks de 4G : l'expressivité de leur modèle et leur capacité à passer à l'échelle en pratique.

2.3.1 Expressivité des modèles

Critères d'évaluation. Pour évaluer l'expressivité des différents modèles, on s'intéresse à la capacité de chacun d'eux à modéliser les différents aspects du réseau évoqué dans l'introduction à savoir :

1. **Le dynamisme du modèle** : est-il possible, comme le dit Baskerville, de modéliser le rôle de la maintenance sur l'état du réseau et de manière plus générale les résultats de l'ensemble des actions des utilisateurs légitimes et illégitimes ?
2. **La dimension relationnelle** : Le modèle prend-il en compte les relations de dépendances qui existent entre les services ? Est-il possible de modéliser les dommages collatéraux engendrés par une attaque ou une panne ?
3. **La dimension financière** : est-il possible de modéliser la notion de coût et de gain ?
4. **La dimension temporelle** : le modèle prend-il en compte le temps nécessaire pour effectuer une attaque ou une action défensive ?
5. **Le langage de spécification** : de quelle manière sont spécifiées les attaques : Celles-ci sont-elles aisément manipulables et flexibles ?

Evaluation. L'expressivité de chaque modèle est comparée dans le tableau 2.10. Parmi les points communs, l'ensemble des frameworks récents s'appuie désormais sur un langage de description basé sur des préconditions et postconditions. L'ensemble des langages étant explicites, il faut pour chaque vulnérabilité décrire précisément les préconditions et les postconditions.

La dimension relationnelle des réseaux est prise en compte à des degrés divers : au minimum, les frameworks tiennent compte de la succession des attaques et dans le meilleur des cas, des informations de routage (firewall) et de configurations. Cependant aucun ne tient compte des relations de dépendances qui permettraient d'inférer les dommages collatéraux.

L'enseignement principal de ce tableau est qu'aucun modèle ne permet aujourd'hui d'exprimer l'ensemble des aspects du problème. En particulier, il apparaît qu'en dehors du modèle SRI issu de la biologie aucun modèle ne rend compte de l'interaction des utilisateurs.

On note d'ailleurs que l'hypothèse d'un accroissement monotone des accès de l'intrus, utilisé par la plupart des frameworks pour passer à l'échelle, est incompatible avec la modélisation de cette interaction. Ce constat est surprenant car les modèles de deuxième génération tenaient compte de cette notion.

De même, aucun modèle ne tient compte de la dimension temporelle, or, celle-ci est nécessaire pour avoir un modèle dynamique et rendre compte de la vitesse de propagation des attaques. Par exemple, dans la propagation des vers comme CodeRed et Slammer, le temps est un facteur central. Ainsi, le pic de contagion de CodeRed [MS01] fût atteint après cinq heures alors que celui de Slammer [MPS+03] fût atteint en moins de 10 minutes. Cette disparité de vitesse de propagation ne peux être rendu sans l'utilisation d'une dimension temporelle.

2.3.2 Mise en application

Critères d'évaluation. La capacité d'un modèle à être mis en application est déterminée par deux éléments essentiels : d'une part sa capacité à être automatisé, car on l'a vu, construire un modèle et l'analyser à la main est impossible avec la taille actuelle des réseaux, et d'autre part sa capacité à passer à l'échelle c'est-à-dire à être appliqué à de gros exemples. On utilise donc les critères suivants pour évaluer les frameworks :

1. **Complexité théorique** : la complexité théorique permet d'avoir une estimation des performances du framework dans le pire des cas. On estime aujourd'hui qu'un modèle qui a une complexité supérieur à EXPTIME ne sera pas utilisable en pratique.
2. **Performance empirique** : il arrive à l'instar du problème de typage en CAML que les performances en pratique soient (bien) meilleures que le pire des cas. Il est donc nécessaire d'avoir une confirmation empirique de la complexité théorique pour savoir à quel point les cas réels sont éloignés du pire cas.
3. **Construction automatique** : avec la taille actuelle des réseaux, il est nécessaire de pouvoir construire le modèle de manière la plus automatique possible. C'est le seul moyen d'avoir un framework d'analyse qui s'adapte rapidement aux changements de situations telle que l'apparition d'une nouvelle vulnérabilité.
4. **Analyse automatique** : de manière similaire à la construction, il est nécessaire d'automatiser l'analyse du modèle. Cette partie étant la plus lente, c'est sur elle que se concentre les études de complexité théorique et les tests empiriques.

Framework	D. F.	D. T.	D. R.	Dyna	Language
Attack tree (Sec. 2.2.1)	**Oui**	Non	**Attaque**	Non	Informel
Privilege graph (Sec. 2.2.2)	Non	Non	**Privilèges** **Attaque**	Non	-
Attack graph (Sec. 2.2.3)	Non	Non	**Attaque** **Routage**	Non	Logique utilisant des préconditions et postconditions explicites
MulVal (Sec. 2.2.4)	Non	Non	**Attaque** **Routage** **Configuration**	Non	Logique : Datalog
NetSPA (Sec. 2.2.4)	Non	Non	**Attaque** **Routage** **Configuration**	Non	Langage explicite nommé REM utilisation de préconditions postconditions explicite
SRI (Sec. 2.2.6)	Non	Oui	**Dépendances**	Oui	N/A

Abréviation :

- *D. F.* : Dimension Financière.
- *D. T.* : Dimension Temporelle.
- *D. R.* : Dimension Relationnelle.
- *Dyna* : Modèle dynamique.

FIGURE 2.10 - *Comparaison de l'expressivité des différents frameworks*

Evaluation. La capacité de chaque framework à être mis en pratique est comparée dans le tableau 2.11. L'enseignement principal de ce tableau est que les frameworks passant le mieux à l'échelle sont ceux, comme NetSPA, qui ont su faire des compromis au niveau des hypothèses théoriques. La grande disparité des résultats, de 1 machines à 50 000, montre bien que si la complexité n'est pas prise en compte dès le début, les résultats en pratique risquent d'être médiocres. On note aussi une corrélation entre la complexité théorique et les résultats pratiques. Ainsi MulVal qui a une complexité quadratique $O(N^2)$ fait 50 fois moins bien que NetSPA qui a une complexité linéaire $O(N)$ ou N est le nombre de machines. Notons que cette différence s'explique aussi en partie par le matériel utilisé pour faire tourner les analyseurs.

Modèle	Complexité	Construction	Analyse	Meilleur exemple
Attack tree 2.2.1	N/A	Manuelle	Manuelle	10 noeuds
Privilege graph 2.2.2	N/A	Auto	Auto	1 machine 13 vulnerabilités
Attack graph 2.2.3	$O(N^6E)$ [AWK02] / Exptime [RA00]	Auto	Auto	5 machines 8 vulnerabilités
MulVal 2.2.4	$O(N^2)$	Auto	Auto	1 000 machines 100 vulnérabilités en 20 minutes
NEtSPA 2.2.5	$O(N)$	Auto	Auto	50 000 machines en 4 minutes
SRI 2.2.6	-	N/A	N/A	-

FIGURE 2.11 - *Comparaison de l'efficacité de la mise en pratique des différents frameworks*

La conception d'un framework 4G nécessite donc de trouver le subtil point d'équilibre entre expressivité du modèle et contraintes pratiques. Il est ainsi impensable aujourd'hui de construire un framework sans garder à l'esprit que sa mise en pratique se doit de passer à l'échelle.

2.4 Points Clés

- Aucun des framework de quatrième génération ne tient compte de la dimension temporelle des attaques.
- Le framework biologique SRI modélise la dimension temporelle de l'épidémie et tient compte de l'interaction des utilisateurs.
- La notion d'interaction des utilisateurs, présente dans les frameworks de deuxième génération, est absente dans tous les frameworks de quatrième génération.
- Les dépendances entre les différents services ne sont prises en compte par aucun framework.
- Les derniers frameworks de la quatrième génération passent à l'échelle puisqu'ils arrivent à analyser des réseaux possédant des dizaines de milliers de machines.
- L'hypothèse de monotonie utilisée par les attack graphs et NetSPA pour passer à l'echelle est incompatible avec la modélisation de l'interaction des utilisateurs puisqu'elle implique que toute action de l'intrus qui devient possible le reste à jamais.

CHAPITRE 3

Anticipation Games, le framework

"Rapidity is the essence of war : take advantage of the enemy's unreadiness, make your way by unexpected routes, and attack unguarded spots."

Sun Tzu, The art of war IX.19

A l'heure actuelle, comme constaté dans le chapitre précédent, aucun framework ne modélise la dimension temporelle des attaques, l'interaction des utilisateurs et les relations de dépendances qui existent dans les réseaux. C'est ce constat qui a servi de point de départ à l'élaboration de notre travail.

Une manière naturelle de modéliser l'interaction des utilisateurs légitimes et illégitimes, avec le réseau, est d'utiliser la théorie des jeux, puisque celle-ci fût développée pour représenter les situations stratégiques, où le succès d'un joueur dépend de l'action des autres [Ras07]. En effet, le but des attaquants est de maximiser leur impact sur le réseau, alors que le but de l'administrateur est de réduire cet impact au maximum. On a donc bien un jeu non-coopératif, où deux joueurs ont des objectifs diamétralement opposés.

On considère que la théorie des jeux fut inventée en 1944 par Von Neumann et Morgenstern [vNM44], bien que des travaux préliminaires datant des années 1930 existent.

Si au départ, l'application de la théorie des jeux est essentiellement centrée sur l'économie, huit prix Nobel d'économie ayant été décernés pour des travaux utilisant la théorie des jeux, elle a depuis été appliquée à de nombreux domaines tels que la biologie [Now06] et la stratégie militaire [B.97].

Elle joue un rôle important en vérification formelle, où elle est un moyen naturel de voir un système et son environnement comme deux joueurs ayant des objectifs différents [Chu62, PR89]. La logique ATL (*Alternating-time Temporal Logic*) [AHK02] est utilisée pour exprimer les propriétés à vérifier dans le cadre des jeux formels. Son extension nommée *timed alternating-time temporal logic* (TATL) [dAFH+03] est utilisée pour parler de jeu temporisé.

Dans le domaine de la sécurité informatique, la théorie des jeux a notamment été utilisée pour modéliser les dénis de services [MS05]. Pour l'analyse des risques réseau en particulier, on ne trouve qu'un seul travail antérieur au notre qui est du à Lye et Wing [LW05]. De l'aveu même de ces auteurs, cette utilisation n'est qu'une approche préliminaire visant à expérimenter les difficultés de l'approche.

Dans ce chapitre, nous commençons par présenter un exemple de services Web redondant, qui nous servira à illustrer le modèle. Puis nous décrivons la couche basse du modèle, le graphe de dépendances, utilisé pour décrire l'état du réseau. Ensuite nous présentons la couche haute du modèle, nommé à l'instar du framework anticipation game, qui est un automate de jeu temporisé. Finalement, nous prouvons que le model-checking de propriétés TATL sur les anticipation games est EXPTIME-Complet et nous concluons en fournissant des exemples de propriétés TATL que l'on souhaite vérifier pour s'assurer de la sécurité de son réseau. Comme pour tous les chapitres, les points clés du chapitre sont repris à la fin. Ce chapitre a fait l'objet des publications [Bur07b] et [BGL07].

3.1 Etude de cas : un service Web redondant

La fourniture de services Web fiables est devenue une préoccupation centrale pour l'activité de nombreuses sociétés. Citons par exemple, Amazon [Ama], Google [Goo98], ou encore le service MSN [Mica] de Microsoft. La fiabilité et la réactivité des services Web offerts est un enjeu majeur afin de satisfaire les clients. L'architecture la plus commune pour maximiser la disponibilité d'un service Web est de recourir à une architecture redondante. Ainsi, même si un des serveurs tombe en panne, la continuité du service est assurée. De plus, en utilisant un mécanisme de répartition de charge, ce type d'architecture permet d'améliorer la réactivité du service.

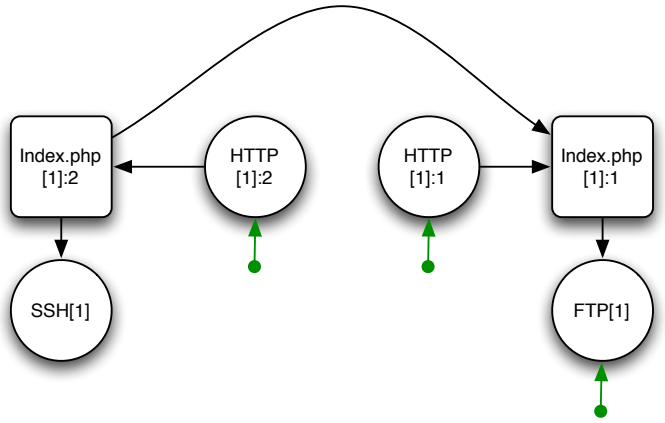

FIGURE 3.1 - *Exemple de graphe de dépendances : un service Web redondant*

Il est pertinent de considérer le cas d'un service Web redondant simplifié pour illustrer notre modèle. Le modèle présenté (Figure 3.1) a été simplifié pour des raisons de clarté. Les deux simplifications apportées par rapport à une architecture réelle sont : la non-modélisation du système de répartition de charge, et l'utilisation de pages Web statiques. Le système de répartition de charge ainsi que l'utilisation de bases de données pour la génération de pages dynamiques peuvent être modélisés dans notre modèle via un ou plusieurs services et dépendances (voir chapitre 6).

3.1.1 Equivalence entre les sommets

Les deux sommets HTTP *HTTP[1] :1* et *HTTP[1] :2* sont deux services Web qui servent respectivement les pages *index.php[1] :1* et *index.php[1] :2*. Dans notre approche, la charte de nommage n'as pas d'incidence. Cependant, nous avons choisi la convention de nommage suivante pour plus de clarté. Chaque nœud est composé du nom du service (HTTP par exemple) ou fichier (index.php par exemple) suivi de sa classe d'équivalence entre crochets, et d'un identifiant unique si la classe contient plusieurs services/fichiers équivalents. Dans les chapitres suivants pour des raisons de clarté, on omettra la classe d'équivalence si elle n'est pas utile. L'identifiant unique est précédé de deux points. Deux sommets ayant le même nom et la même classe sont considérés comme *équivalents*. Equivalent voulant dire ici qu'ils offrent le même service et peuvent donc être librement intervertis. Dans l'exemple, *HTTP[1] :1* et *HTTP[1] :2*, ont le même nom de service HTTP et la même classe : 1, et ils servent des fichiers équivalents. Plus spécifiquement, ils servent des fichiers qui sont équivalents car ils ont le même contenu. Du point de vue de l'utilisateur, récupérer le contenu de l'un ou de l'autre est équivalent. Pour la suite, pour représenter l'équivalence entre deux sommets, nous utiliserons la notation $\triangleq$ et plus tard l'opérateur $\Diamond_{\equiv}$.

3.1.2 Dépendances entre les sommets

Les arcs entre les sommets sont utilisés pour représenter les dépendances qui existent au sein du réseau. Modéliser les dépendances permet de remettre chaque service en contexte et de tenir compte des effets collatéraux induits par une panne ou une compromission du service/fichier.
Dans l'exemple, le fichier *index.php[1] :1* dépend du service *FTP[1]* pour être mis à jour. Le service *HTTP[1] :1* dépend du fichier *index.php[1] :1* car celui-ci contient les données destinées aux clients. Si le fichier n'existe pas ou est indisponible suite à un crash de disque dur, le service renverra une erreur 404 au client. Si le fichier est altéré par un attaquant, le service retournera des informations erronées voir un code malicieux au client. Le fichier *index.php[1] :2* dépend de *index.php[1] :1* pour sa mise à jour, *index.php[1] :2* étant une copie de *index.php[1] :1*. On considère, dans l'exemple, que la mise à jour du fichier *index.php[1] :2* se fait via une réplication sécurisée utilisant une connexion SSH. C'est une méthode communément utilisée par de nombreux logiciels de synchronisation de fichiers tels que *Unison, CVS, SVN, Rsync*. C'est pourquoi la bonne marche de la réplication dépend donc aussi de la disponibilité du service *SSH[1]*.

3.1.3 Un exemple de panne

L'importance des effets collatéraux dans l'analyse des pannes est mise en valeur par l'exemple suivant. Supposons que le service *SSH[1]* devienne indisponible suite à une panne. Le contenu de *index.php[1] :2* va devenir tôt ou tard différent de celui de *index.php[1] :1* car la réplication entre les deux ne peut plus avoir lieu. Par conséquence, les données renvoyées aux clients par *HTTP[1] :1* et *HTTP[1] :2* seront différentes. Il est important de noter que le temps joue un rôle essentiel dans notre exemple. Tant que la page *index.php[1] :1* n'est pas mise à jour, l'impact de la panne du service *SSH[1]* est limité. Si l'on considère que la page *index.php[1] :1* est mise à jour toutes les 5 minutes par un processus extérieur alors l'administrateur a une fenêtre de 5 minutes maximum pour remettre le service en ligne avant que l'impact de la panne ne se propage. Cet exemple illustre bien la relation qui existe entre la conséquence d'une panne, les dépendances existantes sur le réseau et le temps.

3.2 Couche basse

Dans notre framework, la couche basse est utilisée pour modéliser les informations réseau. Contrairement aux modèles d'attaques précédents, présentés au chapitre 2, les arcs entre les sommets n'indiquent pas un chemin d'attaque possible mais les relations existantes entre les services. Les chemins d'attaques sont modélisés par la couche haute du modèle. La couche basse est utilisée uniquement pour modéliser les dépendances entre les services ainsi que l'état des services et fichiers comme illustré précédemment (section 3.1).

3.2.1 Graphe de dépendances

Le graphe de dépendances est utilisé pour représenter les interactions entre les services. Formellement, un graphe de dépendances est défini comme un triplet $G = (V, \rightarrow, \equiv)$ où V est l'ensemble fini des états appelés *sommets*, $\rightarrow$ est une relation binaire sur V (relation de *dépendance*) , et $\equiv$ est une relation d'équivalence sur V. Par exemple, le service *HTTP[1] :1* dépend du fichier *index.php[1] :1*, donc il existe un arc entre *HTTP[1] :1* et *index.php[1] :1*.
La relation $\equiv$ est utilisée pour indiquer que, dans le modèle de dépendance que nous utilisons, deux services jouent le même rôle. Par exemple, dans la section 3.1, deux sommets sont équivalents si et seulement si ils ont le même nom et la même classe d'équivalence. Ainsi on a *HTTP[1] :1* $\equiv$ *HTTP[1] :2*, et *index.php[1] :1* $\equiv$ *index.php[1] :2*, mais *HTTP[1] :1* $\not\equiv$ *FTP[1]*.

En général, $\equiv$ est utilisé pour modéliser des configurations où n services sont utilisés pour la répartition de charge ou pour la tolérance de panne. La redondance est un des principes clés de la tolérance de panne : si une fraction des services fournissant les données tombe en panne, les clients sont toujours capables d'obtenir les données des serveurs survivants. De nombreux protocoles, comme le protocole *Simple Mail Transfert Protocol* (SMTP) [Kle01], utilisé pour le transfert de mail, implémentent nativement la gestion de plusieurs serveurs pour des besoins de tolérance de pannes et de répartition de charge.
La localisation et la priorité des différent serveurs SMTP pour un service donné est indiqué dans les enregistrements MX de sa zone DNS. Sous Unix, la commande **host** peut être utilisée pour obtenir la liste des serveurs SMTP associé à une zone DNS. Ainsi, comme on le voit sur la figure 3.2, le service de mail Gmail de Google utilise 5 serveurs SMTP. Lorsque le client désire relayer un mail à destination d'un compte Gmail, il essaye de contacter le serveur MX ayant la priorité la plus faible. Dans notre exemple : le serveur *gmail-smtp-in.l.google.com*. Si la connexion échoue, il essaiera le suivant et ainsi de suite.

```
gmail.com mail is handled by 10 alt2.gmail-smtp-in.l.google.com.
gmail.com mail is handled by 50 gsmtp147.google.com.
gmail.com mail is handled by 50 gsmtp183.google.com.
gmail.com mail is handled by 5 gmail-smtp-in.l.google.com.
gmail.com mail is handled by 10 alt1.gmail-smtp-in.l.google.com.
```

FIGURE 3.2 - *Exemple : les enregistrements MX pour le service gmail.com*

3.2.2 Les états

Le graphe de dépendances n'évolue pas au court du temps. On utilise un *état* pour modéliser les informations qui évoluent au court du temps. Intuitivement, un état décrit quels services et fichiers sont disponibles, compromis, en panne et ainsi de suite. Par exemple, un service peut être *public* ou non. Un service est typiquement non public quand il est derrière un firewall qui empêche l'accès au service depuis l'extérieur.

Un moyen évident de décrire les états du réseau serait d'énumérer toutes les propriétés possibles pour tous les fichiers et services à tout instant donné. Par exemple, un sommet peut être disponible ou pas, peut être un fichier ou pas, peut être vulnérable ou pas, peut être public ou pas et ainsi de suite. On pourrait alors réserver un drapeau pour chaque propriété de chaque état. Cependant, il est difficile et peu flexible de maintenir une telle liste d'état. En effet, il est évident qu'en fonction de l'analyse à effectuer, les aspects du réseau à considérer ne seront pas les mêmes. Par exemple, si l'on s'intéresse à l'analyse des conséquences d'une panne de disque dur, savoir si le serveur HTTP est vulnérable à une attaque de type buffer overflow n'est pas pertinent. C'est pourquoi nous optons pour une approche plus générique et flexible basée sur une logique modale pour la manipulation des états. Celle-ci est décrite en détail dans la section 3.3.1.

Définissons formellement ce qu'est un état . Soit $\mathcal{A}$ un ensemble fini de *propositions atomique* $A_1, \ldots, A_n, \ldots$, qui dénote chacune une propriété de base de l'état du réseau. Chaque proposition atomique est vraie ou fausse pour chaque sommet. Par exemple Avail est vrai pour chaque sommet qui est disponible, File est vrai pour tous les sommets qui sont des fichiers, Service est vrai pour chaque sommet qui est un service, Compr indique les sommets compromis, Vuln indique les services exploitables à distance, VulnLocal les services exploitables localement. Par exemple les services HTTP de la figure 3.1 sont exploitables localement. Patch est utilisé pour indiquer les services pour lesquels un correctif (patch) existe[LWS02], Pub représente les service publics. Les services publics sont donc les points de départ possibles d'une attaque à distance. Dans l'exemple de la section 3.1, tous les services sauf le SSH[1] sont considérés comme publics. MayDefunct identifie les sommets qui sont indisponibles (quelle que soit la raison, comme les bugs ou les pannes matérielles), Crypt est vrai quand les fichiers sont chiffrés par exemple les fichiers de hash de mots de passe. Il est raisonnable de considérer qu'un attaquant mettra beaucoup plus de temps à accéder aux données d'un fichier chiffré et à en exploiter le contenu que lorsque les données sont stockées en clair. Cette liste de propriétés n'est pas exhaustive et elle peut être étendue en fonction des besoins de l'analyse comme on le verra dans les chapitres 6 et 7. Ainsi, on utilise spécifiquement pour les besoins de l'exemple de la section 3.1, la proposition atomique Synced pour spécifier quels sont les fichiers qui sont le résultat d'une réplication des données. Ce qui correspond dans l'exemple à *index.php[1] :2*.

Les *états* sur G sont donc des fonctions $\rho : \mathcal{A} \rightarrow \mathbb{P}(V)$ qui associent à chaque proposition atomique l'ensemble des sommets qui la satisfont. On décrit ρ de manière finie comme un table de toutes les paires $(\mathcal{A}, v) \in \mathcal{A} \times v)$ telque $v \in \rho(\mathcal{A})$. Il en découle qu'il y a un nombre fini, arbitrairement grand, d'états. Comme il est difficile de manipuler de manière explicite de grandes tables d'états, nous allons utiliser une logique modale pour définir des propriétés plus complexes. Cette logique sera utilisé pour construire les règles d'évolution des états.

Si l'exemple de la panne du service SSH ne demande pas de spécifier des informations sur la topologie réseau, ce type d'information est nécessaire pour la modélisation d'attaques. Par exemple, pour identifier les points de départs possibles d'attaques externes, il est nécessaire de connaître les services accessibles depuis l'extérieur. C'est pourquoi l'ensemble des règles utilisées pour modéliser des attaques réseau et des pannes (section 3.3.2) nécessite que l'on précise les sommets qui sont vulnérables, patchables et accessibles depuis l'extérieur.

3.2.3 Exemple d'état

Pour illustrer le fonctionnement des états du graphe de dépendances, reprenons l'exemple de la section 3.1. Par convention, nous écrivons $\bot$ pour faux et $\top$ pour vrai. Le tableau 3.3 fournit un exemple d'un état possible pour le graphe de dépendance de l'exemple à partir des propositions atomiques décrites ci-dessous. Cet état sera utilisé comme état initial en conjonction avec l'ensemble des règles introduites à la section 3.3.2 pour créer un exemple d'anticipation game au chapitre 4. Notons ici que l'introduction de propositions atomiques additionnelles afin de décrire au plus près les spécificités d'une architecture réseau ne change pas le modèle, tant que leur nombre reste fini.

Par définition d'un état ρ, et suivant le sens intuitif donné à Service, ρ(Service) est l'ensemble des sommets qui sont des services réseaux. La figure 3.3 décrit cet état comme étant la réponse à la question "est-ce que v appartenant à ρ(Service)" pour chaque sommets v. Ainsi le résulat, ρ(Service) est représenté par la première ligne du tableau. La réponse consiste en l'ensemble des sommets qui sont indiqués comme étant $\top$ (Vrai), à savoir *HTTP[1] :1*, *HTTP[1] :2*, *FTP[1]*, *SSH[1]*. La troisième ligne indique qu'initialement tous les sommets sont disponibles. Par convention, nous appelons service disponible tout service qui fonctionne de manière normale, et fichier disponible tout fichier accessible. Pour des raisons de simplicité nous ne considérons pas les permissions des fichiers.
Initialement (voir la ligne ρ(Compr)), aucun sommet n'est compromis. Notons qu'il est possible qu'un sommet soit compromis et disponible. Un exemple de compromission est l'utilisation réussie d'un exploit qui retourne une ligne de commande distante via l'injection d'un shellcode .

Etat	*index.php[1] :1*	*index.php[1] :2*	*HTTP[1] :1*	*HTTP[1] :2*	*FTP[1]*	*SSH[1]*
ρ(Service)	⊥	⊥	⊤	⊤	⊤	⊤
ρ(File)	⊤	⊤	⊥	⊥	⊥	⊥
ρ(Avail)	⊤	⊤	⊤	⊤	⊤	⊤
ρ(Compr)	⊥	⊥	⊥	⊥	⊥	⊥
ρ(Vuln)	⊥	⊥	⊥	⊥	⊤	⊥
ρ(VulnLocal)	⊥	⊥	⊤	⊤	⊥	⊥
ρ(Patch)	⊤	⊥	⊤	⊤	⊤	⊥
ρ(Pub)	⊥	⊥	⊤	⊤	⊤	⊥
ρ(MayDefunct)	⊤	⊤	⊤	⊤	⊤	⊤
ρ(Crypt)	⊥	⊥	⊥	⊥	⊥	⊥
ρ(Synced)	⊥	⊤	⊥	⊥	⊥	⊥

FIGURE 3.3 - *Exemple d'un état*

L'ensemble ρ(Vuln) est l'ensemble des services qui sont susceptibles d'être exploités à distance à cause d'une faille. Si les failles sont généralement dues à un bug dans le code du service qui permet l'exécution de code arbitraire, environ 80% des vulnérabilités étant dues à des dépassement de tampons, il faut noter qu'il existe d'autres types de failles qui ne demandent pas d'attaquer directement le service. Par exemple, durant près de deux ans la distribution Debian [Dou08] et ses dérivées utilisa des clés SSH et SSL prédictibles en raison d'un bug de la génération aléatoire.
ρ(VulnLocal) est l'ensemble des sommets susceptibles d'être exploités de manière locale. Dans l'exemple, les services HTTP sont vulnérables localement. On peux imaginer par exemple que les fichiers *index.php[1]* sont interprétés via une version vulnérable de PHP. Pour exploiter ce type de vulnérabilité, l'attaquant doit pouvoir forcer le serveur à exécuter un script. C'est donc une vulnérabilité locale car l'attaquant doit trouver un moyen de télécharger sur le serveur un fichier contenant le code exploitant la vulnérabilité.
A l'opposé, ρ(Patch) est l'ensemble des services qui sont patchables. On sépare les notions de service vulnérables et service patchables car il arrive qu'un service soit vulnérable mais qu'aucun patch ne soit disponible. Ce phénomène est appelé la *fenêtre de vulnérabilité* [LWS02].

Un cas particulièrement préoccupant de fenêtre de vulnérabilités est lorsqu'un service est vulnérable à un exploit **0 day**, qui est rendu publique et qui prend le vendeur du logiciel de court. Cette situation est arrivée par exemple à Adobe pour le logiciel Acroread en 2008 : il suffisait d'ouvrir un document PDF pour ce faire compromettre sa machine. Détecter et contrer les exploits 0 day est difficile et fait appel à des techniques complexes comme nous le verrons par la suite (chapitre 7).

La fenêtre de vulnérabilité est une autre illustration du rôle central que joue le temps dans la sécurité réseau. D'ailleurs, certaines politiques de mise à disposition de patchs sont particulièrement adaptées à la modélisation par règles. C'est par exemple le cas de celle de Microsoft [Mic08] qui stipule que les correctifs, sont publiés une fois par mois. Cette politique de publication de patch à jour fixe, implique que dans le pire cas une vulnérabilité restera sans correctif pendant 30 jours. Implication qui a d'ailleurs été mise à profit par certains virus tel MyDoom [Kot04] pour maximiser leur propagation.

$\rho(\mathsf{Pub})$ est l'ensemble des sommets qui sont (pour le moment) publics, c'est à dire accessibles depuis l'extérieur. Dans notre exemple, tous les services, à l'exception du service *SSH[1]* sont publics. $\rho(\mathsf{MayDefunct})$ est l'ensemble des sommets qui peuvent tomber en panne. Par exemple, *index.php[1] :1* peut tomber en panne à cause d'une panne de disque dur (nous supposons ici que le serveur n'a pas d'architecture RAID). Le service *SSH[1]* peut tomber en panne à cause d'une panne matérielle ou d'un bug qui ferait crasher le service. La différence entre un bug lié à Vuln et un bug lié à MayDefunct est que le bug lié à MayDefunct n'est pas exploitable. Cette distinction est importante car de nombreux bugs ne sont pas exploitables. C'est-à-dire qu'un attaquant ne peut utiliser ces bugs pour détourner le logiciel de son fonctionnement normal à son profit. Utiliser deux ensembles distincts permet ainsi de capturer la différence entre les deux. A l'instar de cette distinction, il est aussi possible d'utiliser deux ensembles distincts pour faire la différence entre les bugs octroyant des droits utilisateurs et ceux octroyant des droits super utilisateurs.

Crypt est utilisé pour spécifier les fichiers chiffrés, ainsi $\rho(\mathsf{Crypt})$ est l'ensemble de tous les fichiers chiffrés. Il n'y en a aucun dans l'exemple. Cependant, le but de cette proposition atomique est de pouvoir écrire des règles qui tiennent compte du fait que si un attaquant a accès à un fichier chiffré, il lui faudra un très long moment avant de pouvoir utiliser les informations contenues dans le dit fichier.
Enfin $\rho(\mathsf{Synced})$ est l'ensemble des fichiers générés par réplication. Cet ensemble est utilisé pour introduire, dans le modèle, le délai nécessaire à la synchronisation de *index.php[1] :2*.

3.3 Modéliser l'évolution du graphe de dépendances

Nous avons besoin d'un moyen flexible pour modéliser les actions qui vont modifier l'état du réseau. La modification de l'état peut survenir suite à une panne (crash de disque dur par exemple), une attaque, ou une modification due à l'administrateur (interdire l'accès à un service). Pour se faire, on utilise des *règles* ayant des préconditions et des postconditions à l'instar de la logique de Hoare [Hoa69] et des frameworks précédents[SHJ+02b]. Nos règles sont de la forme :

$$\begin{aligned} \Gamma : \quad & \mathbf{Pre}\ \mathrm{Pre}\varphi_{\mathrm{pre}} \\ & \longrightarrow \Delta, p, a \\ & \mathbf{Effect}\ \mathrm{Post}\varphi_{\mathrm{post}} \end{aligned}$$

ou φ_{pre} sont les *préconditions*, qui définissent quand la règle s'applique, Δ est le temps minimal nécessaire à l'exécution de la règle, p est le nom du joueur à l'origine de la règle, a est le nom de la règle, et φ_{post} sont les *postconditions*, qui décrivent les effets de l'exécution. Pre et Post sont des formules de logique modale.
L'utilisation d'une durée d'exécution introduit une ambiguïté : il est demandé que les préconditions Pre soient vérifiées non seulement au lancement de la règle, mais durant toute la période nécessaire pour exécuter la règle, soit au minimum Δ unités de temps. Par exemple, une règle qui a pour but d'appliquer un correctif sur un sommet ne peut être exécutée que sur un nœud possédant un correctif , et à condition que celui-ci reste corrigable/vulnérable durant la totalité de l'exécution de l'application du patch.

Une règle se lit donc de la manière suivante : "Si l'assertion φ_{pre} est vraie pour l'état courant du graphe de dépendances alors la règle R peut être appliquée et si Pre reste vraie durant le délai d'exécution, d'au moins Δ unités de temps, alors φ_{post} devient vraie et l'état du graphe évolue". L'utilisation de préconditions et de postconditions permettent de restreindre la portée des actions des joueurs. Par exemple, il permet de restreindre les attaques à distance aux sommets qui ont une faille exploitable à distance et qui sont publics.

3.3.1 Logique modale

On utilise une logique modale pour spécifier les préconditions Pre et les post conditions $Post$. La modalité $\Diamond$ est utilisée pour parler du concept de dépendances, tandis que $\Diamond_{\equiv}$ est utilisée pour parler de l'équivalence entre les sommets. Les autres connecteurs logiques utilisés sont définis de manière standard : $F \vee F'$ est $\neg(\neg F \wedge \neg F')$, $F \Rightarrow F'$ est $\neg(F \wedge \neg F')$, et $\Box F$ est $\neg\Diamond\neg F$ par exemple. La liste des opérateurs utilisés est donc :

$$\begin{array}{rcll} F & ::= & A & \text{propositions atomique, dans} \mathcal{A} \\ & | & \top & \text{vrai} \\ & | & \neg F & \text{négation} \\ & | & F \wedge F & \text{conjonction} \\ & | & \Diamond F & \\ & | & \Diamond_{\equiv} F & \end{array}$$

La sémantique de notre logique est une sémantique de Kripke [BBF+01] avec un mélange d'opérateurs K ($\Diamond$) et S5 ($\Diamond_{\equiv}$). On définit le prédicat $G, \rho, \nu \models F$ par récurrence sur F : $G, \rho, \nu \models \Diamond F$ si et seulement s'il existe w dans G tel que $\nu \rightarrow w$ et $G, \rho, w \models F$; $G, \rho, \nu \models \Diamond_{\equiv} F$ si et seulement si il y a un sommet w dans G telque $\nu \equiv w$ et $G, \rho, w \models F$. La sémantique des autres connecteurs est standard, par exemple $G, \rho, \nu \models F_1 \wedge F_2$ si et seulement si $G, \rho, \nu \models F_1$ et $G, \rho, \nu \models F_2$.

Dans l'exemple de la section 3.1, si le fichier *index[1] :1* devient indisponible alors *HTTP[1] :1* deviendra indisponible après une certaine durée. Cet effet ce traduit par la règle : *"Si il y a un successeur de* HTTP[1] :1 *qui n'est pas disponible, alors* HTTP[1] :1 *devient indisponible après un délai d'au moins* Δ_2 *unités de temps"*. Le délai pouvant être plus ou moins long selon qu'un cache est utilisé ou non. Les préconditions de cette règle s'écrivent ainsi dans notre logique :

$$\Diamond\neg\texttt{Avail}$$

Elles sont interprétées sur le sommet *HTTP[1] :1*. L'effet de la règle est décrit quant à lui par les postconditions φ_{post} sous la forme d'une liste finie d'affectations : $A \leftarrow \top$ ou $A \leftarrow \bot$, ou A est une proposition atomique. Elles sont interprétées sur chaque sommet ν. De manière formelle, on définit la fonction $\rho[A \mapsto S]$ des états de A sur l'ensemble S. Les autres propositions atomiques A' étant l'objet d'une fonction $\rho(A')$. Définissons $\rho[A@\nu \mapsto \top]$ par $\rho[A \mapsto \rho(A) \cup \{\nu\}]$, et $\rho[A@\nu \mapsto \bot]$ par $\rho[A \mapsto \rho(A) \setminus \{\nu\}]$. Il suit que $\rho \models_\nu P \Rightarrow \rho'$ est défini par : $\rho \models_\nu \epsilon \Rightarrow \rho$ (ou ϵ est une commande vide), et $\rho \models_\nu A \leftarrow b, P \Rightarrow \rho'$ dès que $\rho[A@\nu \mapsto b] \models_\nu P \Rightarrow \rho'$.

3.3.2 Jeu de règles d'exemple

Illustrons maintenant le fonctionnement de notre logique modale via un jeu de règles. On utilise deux ensembles de règles : un pour modéliser les actions de l'incident/intrus et l'autre pour modéliser les actions des administrateurs.

Règles de l'intrus Les règles de l'intrus sont présentées en figure 3.4. Pour chaque règle, au lieu de spécifier le temps d'exécution, nous avons laissé Δ_n pour indiquer un nombre d'unité de temps arbitraire. A partir du prochain chapitre nous utiliserons des délais numériques. La première ligne de chaque règle spécifie les préconditions que l'état du jeu doit satisfaire pour permettre l'exécution de la règle. La deuxième ligne indique le nombre d'unités de temps nécessaire pour exécuter la règle (ici Δ), suivi du nom du joueur à qui appartient la règle (ici I pour intrus/incident), et du nom de la règle. Enfin la dernière ligne décrit, grâce aux postconditions, l'effet qu'a l'exécution de la règle sur l'état du réseau. Notons que par souci de concision, on écrit $Compr$ pour $Compr \leftarrow \top$ et $\neg Compr$ pour $Compr \leftarrow \bot$.

`Defunct` est le cas typique d'une panne accidentelle : n'importe quel fichier ou service susceptible de subir une panne (e.g, bugs) et qui est disponible deviendra indisponible. Il est naturel de faire varier les sommets qui sont susceptibles de tomber en panne en modifiant MayDefunct pour étudier l'impact d'un bug dans un logiciel particulier sur la santé du réseau. Par exemple, il est possible d'étudier l'impact d'un bug des services web en spécifiant qu'il n'y a qu'eux qui sont susceptibles de tomber en panne. La règle se lit de la manière suivante : (ligne 1) si un sommet est disponible ($Avail$) et susceptible de tomber en panne ($MayDefunct$) alors (ligne 2) en un temps Δ, l'intrus (I) peux exécuter la règle $Defunct$ qui (ligne 3) rendra le service non disponible ($\neg Avail$)

`DefunctProp` indique qu'un sommet disponible ($Avail$) tombe en panne ($\neg Avail$) s'il existe un sommet dont tous les équivalents tombent en panne ($\Diamond\neg\Diamond_{\equiv} Avail$), en prenant en compte les sommets équivalents. Par exemple, la plupart des connexions à Internet dépendent des serveurs DNS racine pour faire la translation entre les URL et les adresses IP. Tant que des serveurs DNS racines sont disponibles, les connexions à Internet restent au moins partiellement non affectées par la panne de certains d'entre eux. C'est pourquoi l'attaque du 6 Février 2007 contre 6 des 13 serveurs racines est restée quasi imperceptible pour les utilisateurs d'Internet [ICA07].

1) Γ : **Pre** $Avail \wedge MayDefunct$
$\longrightarrow (\Delta_1, I, Defunct)$
Effect $\neg Avail$

2) Γ : **Pre** $\Diamond\neg\Diamond_{\equiv} Avail \wedge Avail$
$\longrightarrow (\Delta_2, I, DefunctProp)$
Effect $\neg Avail$

3) Γ : **Pre** $Avail \wedge Vuln \wedge Pub \wedge \neg Compr$
$\longrightarrow (\Delta_3, I, Comp_1)$
Effect $Compr$

4) Γ : **Pre** $Avail \wedge Vuln \wedge Pub \wedge \neg Compr$
$\longrightarrow (\Delta_4, I, Comp_2)$
Effect $\neg Avail$

5) Γ : **Pre** $\Diamond Compr \wedge VulnLocal \wedge Avail \wedge Service \wedge \neg Compr$
$\longrightarrow (\Delta_5, I, CServProp_1)$
Effect $Compr$

6) Γ : **Pre** $\Diamond Compr \wedge VulnLocal \wedge Avail \wedge Service \wedge \neg Compr$
$\longrightarrow (\Delta_6, I, CServProp_2)$
Effect $\neg Avail$

7) Γ : **Pre** $\Diamond Compr \wedge Avail \wedge File \wedge \neg Crypt \wedge \neg Compr$
$\longrightarrow (\Delta_7, I, CFileProp_1)$
Effect $Compr$

8) Γ : **Pre** $\Diamond Compr \wedge Avail \wedge File \wedge Crypt \wedge \neg Compr$
$\longrightarrow (\Delta_8, I, CFileProp_2)$
Effect $Compr$

FIGURE 3.4 - *Jeu de règles de l'intrus*

Les règles $\texttt{Comp}_1$ et $\texttt{Comp}_2$ sont utilisées pour modéliser les attaques distantes contre des services disponibles, vulnérables, non compromis et publics ($Avail \wedge Vuln \wedge Pub \wedge \neg Compr$). $\texttt{Comp}_1$ modélise le cas ou l'attaque est un succès total et que le sommet ciblé est totalement compromis ($Compr$). $\texttt{Comp}_2$ modélise le cas typique ou l'attaque échoue et au lieu d'obtenir un accès, l'attaquant rend indisponible le service ($\neg Avail$). Cela arrive par exemple lorsque l'exploitation par débordement de tampon échoue parce que l'adresse de retour injectée ne correspond pas à l'adresse du shellcode. Ceci est courant, surtout depuis l'introduction de la randomisation de l'espace d'adressage dans le noyau Linux (2.6.13).

Les règles restantes sont utilisées pour modéliser la propagation des incidents. $CServProp_1$ dit que si (préconditions) un des sommets dont il dépend est compromis ($\Diamond Compr$), qu'il est vulnérable à une attaque locale ($VulnLoca$), disponible ($Avail$) et que c'est un service ($Service$) non compromis ($\neg Compr$) alors (postcondition) il devient compromis ($Compr$) sous l'action de l'administrateur après Δ unité de temps. Comme pour la compromission distante $\texttt{CServProp}_2$ décrit le cas ou le service devient indisponible ($\neg Avail$) au lieu d'être compromis.

$\texttt{CFileProp}_1$ indique que les fichiers non chiffrés qui dépendent d'un service compromis peuvent être compromis à leur tour. C'est le type de règle qui a typiquement un délai d'exécution très court. A l'opposé, la règle $\texttt{CFileProp}_2$ a un plus grand délai d'exécution car il représente le temps nécessaire à la compromission d'un fichier chiffré (Crypt) avec une clé faible.

Règles de l'administrateur Les règles de l'administrateur sont visible en figure 3.5. Ces règles illustrent comment l'administrateur (A) peut appliquer un correctif ($patch$) qui rendra un service non vulnérable (règle $Patch$ et $PatchLocal$), reconfigurer des équipements réseau pour interdire l'accès à un service (règle $Deny$) ou le rendre accessible (règle $Allow$). Il peut aussi rétablir un service compromis (règle $OReco$) c'est-à-dire rendre un service compromis ($Compr$) non compromis ($\neg Compr$). Enfin, il peut réparer les pannes (règle $ORest$) c'est-à-dire rendre un service non disponible ($\neg Avail$) disponible ($Avail$).

1) Γ : **Pre** $Pub \wedge Service \wedge Vuln$
$\longrightarrow$ (20, A, Deny)
Effect $\neg Pub$

2) Γ : **Pre** $\neg Public \wedge Service \wedge \neg Vuln$
$\longrightarrow$ (20, A, Allow)
Effect Pub

3) Γ : **Pre** $Avail \wedge Vuln \wedge Patch$
$\longrightarrow$ (300, A, Patch)
Effect $\neg Vuln$

4) Γ : **Pre** $Avail \wedge VulnLocal \wedge Patch$
$\longrightarrow$ (300, A, PatchLocal)
Effect $\neg Vuln$

5) Γ : **Pre** $Compr$
$\longrightarrow$ (30, A, OReco)
Effect $\neg Compr$

6) Γ : **Pre** $\neg Avail$
$\longrightarrow$ (30, A, ORest)
Effect $Avail$

FIGURE 3.5 - *Jeu de règles de l'administrateur*

3.4 Couche supérieure : les Anticipation Games

Les règles présentées dans la section précédente donne naissance à une sémantique à base d'automate de jeu temporisé [BHRP06], que nous allons maintenant rendre explicite. Nous appelons ces jeux des *anticipation games* car leur but comme nous l'avons dit est de permettre d'anticiper les conséquences des attaques réseaux et de renforcer la sécurité.

On considère que le nombre de règles utilisées dans chaque jeu est arbitrairement grand mais fini, et que chaque règle possède un nom qui l'identifie de manière unique, à l'instar des règles présentées en section 3.3.2. De plus l'on considère qu'il existe une table Trig telle que pour chaque règle R, $\mathsf{Trig}[R] = \Delta$ retourne le temps minimum pour déclencher l'effet de la règle R. De même, l'on considère qu'il existe une table Act telle que $\mathsf{Act}[R] = a$, une table Prog telle que $\mathsf{Prog}[R] = P$, et enfin une table $\mathbf{Pre}$ tel que $\mathbf{Pre}[R] = F$. Pour tout ensemble S, on écrit $S_\perp$ l'ensemble S union un nouvel élément $\perp$. Par exemple soit $\mathcal{R}$ l'ensemble (fini) de toutes les règles d'un anticipation game, $(\mathcal{R} \times V)_\perp$ indique soit l'absence d'une règle nommée $(\perp)$, soit le couple (R, v), qui est utilisé de manière standard pour spécifier une règle nommé R au sommet v. Ces paires (R, v) sont appelées des *règles ciblées* et les éléments de $(\mathcal{R} \times V)_\perp$ sont des règles ciblées *optionnelles* .

Un *automate de jeu temporisé* est le 8-uplet $\mathcal{T} = (L, \Sigma, \sigma, C, A_I, A_A, E, \gamma)$ qui satisfait certaines conditions [BHRP06, Section 2.2]. Nous allons récapituler ses conditions et montrer qu'elles définissent bien l'automate de jeu associé aux anticipation games. Cela nous donnera l'occasion de donner la sémantique de manière intuitive en même temps.

- L est un ensemble fini d' *emplacements*. Usuellement, l'on appelle ceci des états pour un automate ordinaire. Ici, le terme "emplacement" est plus adapté car nous avons déjà défini le terme *états* comme la fonction $\rho : \mathcal{A} \to \mathbb{P}(V)$ des propositions atomiques A vers les sommets ou A est satisfait. On prend L comme $(\mathcal{A} \to \mathbb{P}(V)) \times V \times (\mathcal{R} \times V)_\perp \times (\mathcal{R} \times V)_\perp$ constitué d'un quadriplet $(\rho, v, \mathrm{trg}_I, \mathrm{trg}_A)$ d'un état ρ, un sommet $v \in V$, et deux règles ciblées optionnelles trg_I, et trg_A qui indiquent quelles règles sont en cours d'exécution et quels sommets elles ciblent. Le sommet v ne joue aucun rôle dans la sémantique des graphes d'anticipation mais il sera utile pour la sémantique de la logique TATL que nous définirons ultérieurement. v est le *sommet d'attention* d'un observateur extérieur au graphe d'anticipation.
- Σ est l'ensemble fini des *propositions*. On prend Σ pour être $\mathcal{A}$.
- $\sigma : L \to \mathbb{P}(\Sigma)$ assigne à chaque emplacement un ensemble de proposition vrai à cet emplacement. On définit naturellement $\sigma(\rho, v, \mathrm{trg}_I, \mathrm{trg}_A) = \{A \in \mathcal{A} | v \in \rho(A)\}$.
- C est l'ensemble fini des *horloges* (aussi connu sous le nom de variables d'horloges). Il existe une horloge spécifique z, qui est utilisée pour mesurer le temps global. On définit $C = \{z, z_I, z_A\}$. L'horloge z_I mesure le temps écoulé depuis le début de la dernière attaque lancée par l'attaquant (ou le dernier événement qui a causé une panne de manière plus générale), s'il y en a une, qui est $\mathrm{trg}_I \neq \perp$.

Similairement, z_A mesure le temps écoulé depuis le début de la dernière action correctrice de l'administrateur, s'il y en a une. Comme pour les réseaux réels, on autorise I et A à lancer des actions concurrentes, avec des dates de départ et des délais différents.

- A_I et A_A sont deux ensembles disjoints d'événements pour l'attaquant I et l'administrateur A respectivement. Dans le langage usuel, l'on parle d'action mais cela on ne peut utiliser ce terme ici, car l'on aurait un conflit avec notre propre définition du terme action. On prend les éléments, A_p comme les paires $\langle p, \mathbf{Launch}\ a\rangle$ et $\langle p, \mathbf{Complete}\ a\rangle$, , and $\langle p, \mathbf{Drop}\ a\rangle$, ou a est un nom d'action quelconque, pour n'importe quelle $p \in \{I, A\}$. Un événement $\langle I, \mathbf{Launch}\ a\rangle$ indique que l'intrus viens juste de lancer une attaque via une action nommée a. Cette attaque pourra réussir, mais pas avant le délai imparti. Quand elle aura réussi, cela sera rendu explicite par une action $\langle I, \mathbf{Complete}\ a\rangle$. Elle peut aussi échouer et cela sera rendu explicite par une action $\langle I, \mathbf{Drop}\ a\rangle$.
  $E \subseteq L \times (A_I \cup A_A) \times \mathsf{Constr}(C) \times L \times \mathbb{P}(C \setminus \{z\})$ est la *relation de transition*, elle encapsule la sémantique d'un automate de jeu temporisé. $\mathsf{Constr}(C)$ est l'ensemble de toutes les *contraintes d'horloges* engendrées par la grammaire : $\theta ::= x \leq d | d \leq x | \neg\theta | \theta_1 \wedge \theta_2 | \texttt{TRUE}$ où x varie sur les horloges de C, et d varie sur $\mathbf{N}$. L'idée est que si $(l, \alpha, c, l', \lambda) \in E$, alors l'automate de jeu temporisé peut aller de l'emplacement $l \in L$ à l'emplacement l' en effectuant l'action α, à condition que toutes les horloges soit mises de façon à ce que c soit vrai ; A ce moment toutes les horloges de λ seront remises à zéro. Additionnellement, un automate de jeu temporisé peut décider de rester inactif durant un certain temps, ce qui revient à ne franchir aucune transition, à condition que l'invariant $\gamma(l)$ reste satisfait.
  Dans notre cas, E est l'ensemble de tous les tuples (i.e., *transitions*) qui sont d'une des formes suivantes :
  - $((\rho, \nu, \mathrm{trg}_I, \mathrm{trg}_A), \langle I, \mathbf{Launch}\ a\rangle, \texttt{TRUE}, (\rho, \nu, (R, \nu'), \mathrm{trg}_A), \{z_I\})$, ou ν' est un des sommets de G, et R est le nom de la règle $\Gamma : \mathbf{Pre}\ F \xrightarrow{\langle \Delta, I, a\rangle} P$ avec $G, \rho, \nu' \models F$. (et ν est choisi arbitrairement). En d'autres terme, l'intrus peut décider de lancer une règle nommée R sur n'importe quel sommet ν' à n'importe quel moment, à condition que ces préconditions F soit satisfaites à l'état courant ρ pour le sommet ν'. Lancer la règle ne modifie pas ρ, celui-ci changera uniquement quand la règle sera exécutée. Ceci ne pouvant survenir qu'après au moins Δ unités de temps. Notons que cette transition s'applique uniquement quand le composant trg_I de l'emplacement de départ I is $\perp$, ce qui veut dire qu'il n'y a pas d'évènement en attente de la part de l'attaquant. Notons que lorsqu'une nouvelle règle est lancée, l'horloge z_I est remise à zéro.
  - $((\rho, \nu, (R, \nu'), \mathrm{trg}_A), \langle I, \mathbf{Complete}\ a\rangle, z_I \geq d, (\rho', \nu, \perp, \mathrm{trg}'_A), \emptyset)$, où $a = \mathsf{Act}[R]$ et $d = \mathsf{Trig}[R]$, et ρ' est donné par $\rho \models_{\nu'} \mathsf{Prog}[R] \Rightarrow \rho'$. Alors, $\mathrm{trg}'_A = \mathrm{trg}_A$ si $\mathrm{trg}_A = \perp$, ou si trg_A est de la forme (R_A, ν_A) où $G, \rho', \nu_A \models \mathbf{Pre}[R_A]$; autrement $\mathrm{trg}'_A = \perp$.

En d'autres termes, la règle nommée R s'exécute pour le sommet v' à n'importe quel moment temps à condition qu'au moins d unités de temps se soient écoulées depuis que l'attaque ait été lancée ($z_I \geq d$). Alors l'état ρ est changé en ρ', qui est le résultat de l'exécution du programme $P = \mathsf{Prog}[R]$ pour l'état ρ sur le sommet v'. Le fait que trg_A puisse changer en trg'_A reflète l'élément de surprise : le résultat d'une action de l'attaquant I peut invalider les pré conditions $\mathbf{Pre}[R_A]$ de la règle qui est en cours d'exécution pour l'administrateur A rendant ainsi son exécution impossible.

  - Des règles similaires sont obtenues en échangeant les rôles I et A.
- $\gamma : L \to \mathsf{Constr}(C)$ est la fonction de mappage de chaque emplacement l vers un $\gamma(l)$ *invariant*. Quand ils sont à l'emplacement I, chaque joueur (I et A) doit proposé une transition partant de l avant que l'invariant $\gamma(l)$ expire. On prend $\gamma(l) = \text{TRUE}$ pour chaque l, car dans le modèle nous n'avons aucune transition urgente : attaques, pannes et réparations peuvent toujours prendre plus de temps que prévu.

Informellement [BHRP06], un jeu procède en sautant de configuration en configuration. Une *configuration* est une paire (l, κ), ou l est un emplacement et κ est une *valeur d'horloge*, ce qui est une fonction de mappage pour chaque horloge (ici z, z_I, z_A) vers un réel non négatif. Les automates de jeux temporisés peuvent procéder en exécutant une transition en un temps zéro. Ils peuvent aussi laisser le temps passer. C'est à dire rester au même emplacement l tout en incrémentant chaque horloge de la même valeur. Il est important de voir que pour n'importe quelle configuration, le jeu à plusieurs possibilités d'évolution, en particulier il est possible que deux transitions aient le même emplacement de départ. La sémantique d'un jeu temporisé précise que seule celle qui a le plus court temps d'exécution peut être déclenchée. S'il existe plusieurs transitions qui ont le plus court temps d'exécution alors une d'entre elle est choisie de manière non déterministe pour être déclenchée.

3.4.1 Propriétés

Il est pratique de définir une logique qui inclut à la fois les opérateurs TATL [HP06] et les opérateurs de la logique modale définis en section 3.3.1 pour exprimer les vérités à vérifier sur le modèle.

Pour définir la syntaxe de notre logique TATL on prend $w, x, y, \ldots$ pris parmi un nombre infini dénombrable d'ensemble de *variables d'horloges* ; distinctes des horloges z, z_I, z_A. On réserve les notations d, d', d_1, d_2, $\ldots$, pour les constantes entières non négatives. Et l'on définit $\mathfrak{P}$ comme appartenant au sous-ensemble $\{A, I\}$. La syntaxe de notre logique TATL$\Diamond$ est présenté ci-dessous. Les contraintes d'horloges $x + d_1 \leq y + d_2$, x et y peuvent être des variables d'horloges ou égale à zéro. Notons que l'on abrège $\langle\!\langle \mathfrak{P} \rangle\!\rangle \text{TRUE}\, \mathcal{U}\, \varphi$ par $\langle\!\langle \mathfrak{P} \rangle\!\rangle \blacklozenge \varphi$.

$$
\begin{array}{rcll}
\varphi & ::= & A & \text{atomic prop., in } \mathcal{A} \\
& | & \neg\varphi & \\
& | & \varphi \wedge \varphi & \\
& | & \Diamond\varphi & \\
& | & \Diamond_{\equiv}\varphi & \\
& | & x + d_1 \leq y + d_2 & \text{clock constraints} \\
& | & x \cdot \varphi & \text{freeze} \\
& | & \langle\!\langle \mathfrak{P} \rangle\!\rangle \blacksquare \varphi & \text{invariant} \\
& | & \langle\!\langle \mathfrak{P} \rangle\!\rangle \varphi_1 \, \mathcal{U} \, \varphi_2 & \text{eventually}
\end{array}
$$

La sémantique est à l'instar de tout automate temporisé de jeu donné en spécifiant quand $l, t, \kappa \models \varphi$ satisfait toutes les configurations (l, κ) et temps t. On rappelle que l est de la forme $(\rho, \nu, trg_l, trg_A)$. Soit $(\rho, \nu, trg_l, trg_A), t, \kappa \models A$ si et seulement si $\rho(A)$ est vrai. Comme pour la logique modale de la section 3.3.1, on définit $(\rho, \nu, trg_l, trg_A), t, \kappa \models \Diamond\varphi$ si et seulement s'il y a un sommet w dans G tel que $\nu \rightarrow w$ et $(\rho, w, trg_l, trg_A), t, \kappa \models \varphi$, et similairement pour $\Diamond_{\equiv}$, $\neg$, $\wedge$. Comme pour TATL, $l, t, \kappa \models x + d_1 \leq y + d_2$ si et seulement si $\kappa(x) + d_1 \leq \kappa(y) + d_2$, $l, t, \kappa \models x \cdot \varphi$ et si et seulement si $l, \kappa[x \mapsto t] \models \varphi$. $\langle\!\langle \mathfrak{P} \rangle\!\rangle \blacksquare \varphi$ est satisfait quand les joueurs de $\mathfrak{P}$ ont une stratégie qui garantie que φ va être satisfait pour tous les instants du futur quoique les autres joueurs fassent.

$\langle\!\langle \mathfrak{P} \rangle\!\rangle \varphi_1 \, \mathcal{U} \, \varphi_2$ est satisfait à chaque fois que le joueur de $\mathfrak{P}$ à une stratégie garantissant que φ_2 finira par être satisfait et, que φ_1 sera satisfait pour toutes les unités de temps qui précèdent, quoi que fasse l'autre joueur. De plus, une condition technique garantit que le temps diverge, de manière à empêcher un joueur de gagner en stoppant le temps grâce à une série infinie d'actions instantanées : voir [HP06, Section 3.1] pour les détails de la condition. Nous nous en servirons plus tard comme condition de victoire $WC_{\mathfrak{P}}$ dans la preuve du théorème 3.1.

Le model-checking d'une formule TATL est décidable [HP06, Theorem 1], et EXPTIME-complete. Il s'ensuit que le model-checking d'une formule TATL$\Diamond$ sur un anticipation game est aussi décidable. La preuve de ce résultat est basée sur l'observation que n'importe quelle formule φ de notre logique peut être encodée dans une formule TATL standard φ^*_ν such that $l, t, \kappa \models \varphi$ si et seulement si $l^*, t, \kappa \models \varphi^*_\nu$, ou $l = (\rho, \nu, trg_l, trg_A)$, et $l^* = (\rho, trg_l, trg_A)$ est un emplacement de notre automate de jeu modifié. L'automate de jeu temporisé est donné par l'ensemble des transitions E^*, constitué des transitions de la forme $((\rho, trg_l, trg_A), \alpha, c, (\rho', trg'_l, trg'_A), \lambda)$, ou $((\rho, \nu, trg_l, trg_A), \alpha, c, (\rho', \nu, trg'_l, trg'_A), \lambda)$ est une transition dans E pour un ν quelconque. On se sert ici du fait que la sémantique ne dépende pas de ν. L'encodage de φ en φ_ν est relativement transparente, e.g., $(\varphi_1 \wedge \varphi_2)^*_\nu = \varphi^*_{1\,\nu} \wedge \varphi^*_{2\,\nu}$. Le transcodage se fait de manière similaire pour tous les cas sauf quand φ est de la forme $\Diamond\varphi'$ ou $\Diamond_{\equiv}\varphi'$. On définit alors $(\Diamond\varphi')^*_\nu$ comme une disjonction de tous les w telque $\nu \rightarrow w$ de φ'^*_w, et $(\Diamond_{\equiv}\varphi')^*_\nu$ comme une disjonction de tous les w's tel que $\nu \equiv w$ of φ'^*_w.

On peut raffiner ceci comme suit :

Théorème 3.1. *Le model-checking de formule TATL$\Diamond$ sur un anticipation game est EXPTIME-complete.*

Démonstration. Le théorème s'appuie sur un encodage de TATL$\Diamond$ vers TATL, et de l'algorithme de model-checking de TATL. Cet algorithme est détaillé dans les papiers [HP06, dAFH+03, AHK02]. Nous écrivons $\langle\langle\mathfrak{P}\rangle\rangle$, $\blacksquare$ et $\blacklozenge$ pour les modalités ATL* standards. Elles sont usuellement écrites $\Box$ et $\Diamond$, Cependant nous utilisons $\blacklozenge$ afin de ne pas confondre la modalité TATL et notre modalité $\Diamond$. Pour des raisons de cohérence, nous utilisons $\blacksquare$ pour $\Box$ bien que cela ne soit pas nécessaire.

D'après [HP06, Lemma 1, 2], [dAFH+03, Theorem 5], et [AHK02, Section 3.2], on sait que le model-checking d'une formule TATL ϕ sur un automate de jeu temporisé $\mathcal{T}$, qui vérifie que quelque soit $s \models_{td} \phi$, pour un état quelconque s dans $\mathcal{T}$, peut être accomplie en un temps $T = O((|Q| \cdot m! \cdot 2^m \cdot (2c+1)^m \cdot h \cdot h_*)^{h_*+1})$.
Dans cette formule, Q est l'ensemble des emplacements dans $\mathcal{T}$, m est le nombre d'horloges dans $\mathcal{T}$ plus le nombre de quantificateurs de freeze dans ϕ, c est le plus grand délai de $\mathcal{T}$ et de ϕ. h est le nombre d'états, et h_* est l'ordre (c.a.d., La moitié du nombre possible de priorité assignée à un état) pour un automate déterministe et à parité totale $H_{\phi^\wedge}$ calculé à partir de $\phi^\wedge$, ou $\phi^\wedge$ est lui même obtenu à partir de ϕ en remplaçant chaque contrainte α de la forme $x + d_1 \leq y + d_2$ par une proposition atomique fraîche p_α.

Par le lemme [dAFH+03, Lemma 1], les valeurs de h et h_* sont polynomiales en la taille de ϕ. L'automate de jeu temporisé $\mathcal{T}$ qui sous tend un anticipation game $\mathcal{G}$ a exactement le même nombre d'horloges et la même borne supérieure c sur ses valeurs d'horloge, mais il a un nombre d'emplacements exponentiel nommé $|Q| = O(2^{|V|\cdot n} \cdot (r \cdot |V|)^2)$, tel que $|V|$ est le nombre de sommets dans le graphe de dépendances G, n est le nombre de propositions atomiques dans $\mathcal{A}$, et r est le nombre de règles. Donc l'expansion du temps T s'exprime toujours sous forme d'une simple exponentielle dans l'anticipation game. Cependant, l'encodage de TATL$\Diamond$ vers TATL construit une formule TATL $\phi = \varphi^*_\nu$ de taille exponentielle en la taille de φ : il en suit que h est exponentiel en la taille de φ. h_* compte le nombre d'usages imbriqués de condition de victoire nommé WC_1 dans [dAFH+03], ou plus généralement $WC_{\mathfrak{P}}$. Pour $\mathfrak{P} \subseteq \{I, A\}$: fixons trois nouvelles propositions atomiques distinctes, tick, bl_I et bl_A, Soit $bl_{\{I\}} = bl_I$, $bl_{\{A\}} = bl_A$, $bl_\emptyset$ est faux et $bl_{\{I,A\}}$ est vrai, alors pour chaque formule ATL* ψ, $WC_{\mathfrak{P}}(\psi) = (\blacksquare\blacklozenge tick \Rightarrow \psi) \wedge (\blacklozenge\blacksquare\neg tick \Rightarrow \blacklozenge\blacksquare\neg bl_{\mathfrak{P}})$ états (informellement) pour lesquels soit le temps diverge (on a un nombre infini de tics) et ψ est satisfaite, ou alors le temps est borné, on a alors un nombre fini de tics, et l'un des joueurs de $\mathfrak{P}$ peut être blâmé pour avoir bloqué le temps en déclenchant une infinité d'actions ayant un délai de 0.

Par [dAFH+03, Lemma 1], la valeur h_* d'une formule de la forme $WC_{\mathfrak{P}}(\psi)$ est un plus ψ. La valeur h_* d'une formule TATL sans variable d'horloges est $\phi^{\wedge}$ (d'après la formule TATL ϕ de [HP06, Lemma 2]) qui est donnée au travers de la formule ATL* $\psi = \mathsf{atlstar}(\phi)$ décrite dans [HP06, Lemma 1]. Il en résulte que les cas cruciaux de l'encodage de atlstar sont $\mathsf{atlstar}(\langle\langle\mathfrak{P}\rangle\rangle\blacksquare\phi_1) = \langle\langle\mathfrak{P}\rangle\rangle(WC_{\mathfrak{P}}(\blacksquare\mathsf{atlstar}(\phi_1)))$ et $\langle\langle\mathfrak{P}\rangle\rangle\phi_1 \mathcal{U} \phi_2 = \langle\langle\mathfrak{P}\rangle\rangle WC_{\mathfrak{P}}(\mathsf{atlstar}(\phi_1)\,\mathcal{U}\,\mathsf{atlstar}(\phi_2))$, et aucun autre cas n'utilise une modalité $\blacksquare$ ou $\blacklozenge$. Donc la valeur h^* de $\phi^{\wedge}$, ou de ϕ, est exactement la profondeur d'imbrication des quantifieurs dans ϕ. Dans notre encodage $\phi = \varphi^*_\nu$, bien que ϕ soit exponentiellement plus large que φ, la profondeur d'imbrication des quantifieurs reste la même. Donc le temps T est exponentiel en la taille de φ et en la taille de l'anticipation game considéré.

Cependant, la preuve d' EXPTIME-hardness ne provident *pas* de l'EXPTIME-hardness du model-checking TATL, contrairement à [HP06, Theorem 1] : dans cette preuve , il est crucial d'imposer des gardes de la forme $x = d$ sur l'automate (ou x est une horloge), alors que nous avons uniquement des gardes de la forme $x \geq d$. A la place, l'EXPTIME-hardness provient du fait que la formule modale peut être utilisé pour représentée un ensemble d'environnement de manière concise.

Afin de prouver cette borne, nous encodons directement le problème de l'accessibilité pour des machines de Turing alternante à espace polynomial $\mathcal{M}$. Sans perte de généralité, on peut considérer que $\mathcal{M}$ alterne strictement entre les états $\forall$ et $\exists$, ou les états $\exists$ amènent à l'acceptation si et seulement si au moins un des successeurs amène à l'acceptation et, les états $\forall$ sont les états qui amènent à l'acceptation si et seulement si tous les états amènent à l'acceptation.

Soit n la taille de l'entrée, et $p(n)$ l'espace polynomial disponible pour $\mathcal{M}$. L'on peut représenter les ID de $\mathcal{M}$ (bande, contenus, états de contrôle, position de la tête) en utilisant $O(p(n))$ atomique propositions A_i, une pour chaque bit d'ID. On considère que la position de la tête est représenté par $O(p(n))$ propositions et que parmi elles, une et une seule est vraie. On considère aussi qu'une proposition `all` est vraie quand l'état de contrôle est un état $\forall$ et faux si c'est un état $\exists$. Une proposition `accept` est vraie uniquement dans les états acceptants.

Construisons le graphe de dépendance trivial G avec exactement un sommet et pas de transition : sur G chaque variable est soit vraie soit fausse (pour cet unique sommet). Pour chaque transition $(q, \alpha, q', \alpha', \mathrm{dir})$ de $\mathcal{M}$. Pour l'état q qui est l'état qui lit la lettre α, va vers l'état q' tout en écrivant α' sous la tête de lecture et qui déplace la tête dans une direction $\mathrm{dir} \in \{-1, 0, 1\}$) on écrit $O(p(n))$ règles, une pour chaque position k de la tête de lecture. Si q est un état $\exists$, les règles sont de la forme :

$$\begin{aligned} \Gamma : \quad & \mathbf{Pre}\ F_{k,q,\alpha} \\ & \longrightarrow 1, \mathsf{I}, a \\ & \mathbf{Effect}\ P_k, q', \alpha', \mathrm{dir} \end{aligned}$$

Ou $F_{k,q,\alpha}$ est la formule construite à partir des propositions A_i qui test précisément à la position k, que l'emplacement est exactement q ; et que la lettre sous la tête est exactement α. De plus $P_{k,q',\alpha',\mathrm{dir}}$ assigne et remet à zéro des bits de manière à écrire α' sous la tête, change l'état de contrôle en q', et change la position de la tête. Si q est un emplacement $\forall$, les règles sont de la forme (A joue) :

$$\begin{aligned} \Gamma : \quad & \mathbf{Pre}\ F_{k,q,\alpha} \\ & \longrightarrow 1, \mathsf{A}, a \\ & \mathbf{Effect}\ P_k, q', \alpha', \mathrm{dir} \end{aligned}$$

On ajoute la règle additionnelle *rush* :

$$\begin{aligned} \Gamma : \quad & \mathbf{Pre}\ F_{k,q,\alpha} \\ & \longrightarrow 2, \mathsf{I}, \mathrm{rush} \\ & \mathbf{Effect}\ \texttt{accept} \leftarrow \top \end{aligned}$$

Celle-ci est jouée par I, mais demande 2 unités de temps au lieu de 1 pour les autres règles. Ensuite on model-check la formule $F = \langle\!\langle \mathsf{I} \rangle\!\rangle \blacklozenge \texttt{accept}$ sur ce graphe à partir de la position initiale qui code l'entrée de $\mathcal{M}$.

Si $\mathcal{M}$ est acceptable, alors il existe une stratégie non temporisée qui choisit des transitions depuis les états $\exists$ tel que quelque soit la transition choisie pour les états $\forall$, un état acceptant est atteint. Cela se traduit directement en une stratégie pour I contre A dans un anticipation game pour le cas d'états $\exists$. Pour chaque état $\forall$ q, l'argument est un peu plus subtile, sans la règle *rush*, A peut simuler de prendre n'importe quelle transition de q, mais il peut aussi décider d'attendre et ne jamais prendre de transition. Pour contrer ce comportement, la stratégie de I est de lancer la règle *rush*, ainsi si A attends plus de 2 unités de temps, alors la règle *rush* est exécutée et une position ou `accept` est vraie est atteinte. Il en découle que dans n'importe quel cas F est satisfait. Réciproquement, si F est satisfait, alors il existe une stratégie pour I qui va finir par mettre `accept` à vrai quoi que fasse A. Notons que dans tous les états $\forall$, Admin peut toujours exécuter en 1 unité de temps, car c'est un temps inférieur à celui nécessaire à I pour exécuter la règle *rush*, donc cette règle ne joue aucun rôle.

En particulier, comme les états $\exists$ et $\forall$ alternent dans $\mathcal{M}$, et parce que nos règles vérifient toujours que l'on est dans le bon état q dans leur preconditions, aucune règle *rush* d'un emplacement $\forall$ précédemment atteint ne peux rester en suspens. Une fois qu'une règle non-rush a été exécutée, les preconditions pour la règle fausse deviennent fausse et le champs $\mathrm{trg}_{\mathsf{I}}$ dans l'automate temporisé correspondant est réinitialisé à $\perp$. C'est pourquoi il est important que $\mathrm{Pre}[\mathrm{R}_{\mathsf{A}}]$ et $\mathrm{Pre}[\mathrm{R}_{\mathsf{I}}]$, soit testé lorsqu'un autre joueur prend une transition. Il est évident que cette stratégie mène I à choisir une transition dans les états $\exists$ de $\mathcal{M}$ qui mèneront à l'acceptation et ce quelque soit les transitions prises dans les états $\forall$. □

3.5 Analyser la sécurité d'un réseau

3.5.1 Adéquation du modèle à l'analyse de risques

Le découplage entre la modélisation des comportements et de l'état du réseau permet de minimiser les modifications à apporter au modèle lorsque un changement survient sur le réseau (vulnérabilité découverte, service ajouté..). Ce découplage qui permet de relancer rapidement une analyse a chaque changement intervenant sur réseau, fait que notre modèle est adapté au processus d'évaluation continuel qu'est l'analyse de risques.

De plus, l'utilisation de règles pour modéliser les comportements permet de réutiliser le même jeu de règles pour plusieurs architectures réseau. Cela permet d'envisager d'insérer notre framework dans les procédures d'audit de sécurité : l'auditeur rédige les règles et propriétés utilisées pour évaluer l'architecture. Il fournit et interprète les conclusions de la vérification et fournit le jeu de règles et de propriétés utilisé pour cette analyse. Celles-ci pourront être utilisées pour valider les changements dans la topologie réseau ou évaluer l'impact de nouvelles vulnérabilités.

3.5.2 Exemple de propriétés

Illustrons maintenant comme l'on utilise les propriétés TATL pour modéliser des objectifs de sécurité. Les propriétés présentées ici sont volontairement générales afin d'illustrer la puissance du modèle. On utilisera dans les chapitres suivants des propriétés beaucoup plus spécifiques et concrètes.

Une propriété simple la résilience Une première propriété que l'on peut vérifier sur le réseau est appelée *résilience à l'intrusion*. Elle est utilisée pour vérifier que l'administrateur a toujours accès à un sommet qui n'est pas compromis et ce quoique l'intrus fasse. Elle s'écrit :

$$\langle\!\langle \mathsf{A} \rangle\!\rangle \blacksquare \Diamond_{\equiv} \neg \mathsf{Compr}$$

Sa propriété jumelle pour la disponibilité est la *résilience aux pannes*

$$\langle\!\langle A \rangle\!\rangle \blacksquare \Diamond_{=} \mathsf{Avail}_{states}$$

Celle-ci permet de garantir que l'administrateur a toujours la possibilité de maintenir un sommet disponible quoique la panne provoque. C'est cette propriété qui aurait été désirable dans le cadre de la panne des serveurs de Bouygues mentionné dans l'introduction.

Une propriété complexe les SLA Il est aussi possible d'utiliser les anticipation games-pour analyser une partie proéminente des SLA *Service Level Agreement* : la garantie de disponibilité. Celle-ci peut être décrite informellement comme étant la garantie que le système ne sera pas indisponible plus de x% du temps durant l'année. Une garantie standard pour un service d'hébergement étant une disponibilité de 99.999% sur l'année soit 8.76h de non disponibilité. On modélise cette contrainte via la propriété suivante :

$$\langle\!\langle A \rangle\!\rangle \blacksquare x \cdot \neg \Diamond_{=} \mathsf{Avail} \Rightarrow [\langle\!\langle A \rangle\!\rangle \blacklozenge y \cdot y \leq x + d \wedge \langle\!\langle A \rangle\!\rangle \blacksquare z \cdot z \leq y + d' \Rightarrow \Diamond_{=} \mathsf{Avail}]$$

De manière formelle, l'on ne peut écrire directement cette garantie en TATL$\Diamond$, mais cela n'aurait de toute façon aucun sens. En effet l'on pourrait imaginer le cas extrême où le service est non disponible une milliseconde toutes les secondes. Ce qui violerait la spécification mais qui ne correspond à aucun comportement réel. En effet, il est physiquement impossible qu'un matériel tombe en panne et soit réparé durant un intervalle si court. Même le fait de débrancher et rebrancher un câble réseau prend un temps supérieur. C'est pourquoi plutôt que de modéliser directement la durée du temps de rétablissement, notre propriété SLA spécifie que quand un service tombe en panne et qu'aucun service équivalent n'est disponible, alors l'administrateur se doit de le remettre en service en au plus d unités de temps (ou d est typiquement très bref), et que de manière subséquente, ce service ou un service équivalent doit être disponible pendant au moins d' unités de temps, où d' est généralement grand. L'utilisation d'une durée d' grande empêche les comportements pathologiques du type : le service tombe une minute, revient cinq minutes et ainsi de suite toute l'année. Rendant le service inutilisable et cependant conforme à la garantie SLA.

3.6 Exemple d'exécution et analyse automatique

La démarche naturelle consisterait à illustrer dès maintenant notre modèle par un exemple d'exécution du jeu qui reprends, l'état initiale et les règles vues au cours de ce chapitre. Cependant ce jeu génère *327834* exécutions distinctes ce qui rend toute tentative d'analyse manuelle vaine. Bien sur on aurait pu utiliser une exécution réalisée à la main, mais cela est contraire à l'esprit de ce travail qui est d'offrir un framework facilement manipulable et automatisé.

On préférera donc repousser la présentation d'une exécution au chapitre suivant afin de pouvoir présenter simultanément des exécutions du jeu et l'outil que nous avons créé qui permet de les générer. Les lecteurs les plus impatients, peuvent trouver l'exemple tiré de ce chapitre en section.

3.7 Points clés

- La structure de jeu temporisé des anticipation gamespermet de modéliser l'interaction simultané de l'intrus et de l'administrateur avec le réseau.
- Les anticipation gamessont le premier framework qui modélise les dépendances entre les services.
- Les anticipation gamesrendent compte de la dimension temporelle des attaques puisque chaque action demande un temps d'exécution.
- La complexité du framework est plus élevée que celle des précédents frameworks de 4G.
- L'utilisation d'une représentation compacte permet de modéliser de gros réseaux.
- La modélisation des actions de l'intrus et de l'administrateur sous forme de règles permet de découpler la modélisation des comportements de la topologie réseau sous-jacent. Ce découplage permet de limiter les changements à introduire dans le modèle lorsque le réseau évolue.
- L'utilisation de règles pour modéliser les comportements ouvre la perspective de la constitution d'une base de données réutilisable par l'ensemble des utilisateurs du framework.

Deuxième partie

Mise en pratique

CHAPITRE 4

Anticipation Games, l'implémentation : NetQi

"In battle, there are not more than two methods of attack : the direct and the indirect ; yet these two in combination give rise to an endless series of maneuvers."

Sun Tzu, The art of war IV.10

Comme on l'a vu au chapitre 2, la création d'un framework complet pour l'analyse de risques réseau implique la création d'un modèle et la réalisation des outils nécessaires pour vérifier automatiquement les réseaux abstraits dans ce formalisme. Ce chapitre présente donc *NetQi*, notre outil, qui permet la vérification automatique des anticipation games. Cet outil représente un élément clé du framework, car sans lui il est quasi impossible de travailler sur les anticipation games, puisque même un exemple simple génère un très grand nombre d'états.

Ce chapitre est composé de trois parties. La première partie détaille le fonctionnement général de *NetQi* et le compare à l'autre model-checker TATL existant, Uppaal-Tiga [CDF+05]. La deuxième partie présente un exemple simple de jeu qui introduit la syntaxe de l'outil, et qui revient sur la notion d'élément de surprise. Enfin, la troisième partie présente un exemple plus complexe qui est la prolongation de l'exemple du chapitre précédent. Ce chapitre a donné lieu à la publication [Bur08b] et la réalisation de l'outil *NetQi*[Bur].

4.1 *NetQi* Présentation

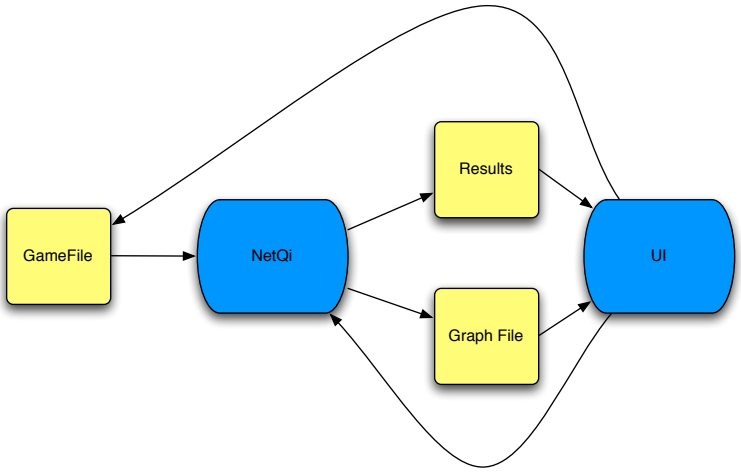

FIGURE 4.1 - *Vue d'ensemble du fonctionnement de* NetQi. *La couleur bleu dénote les programmes, la couleur jaune les fichiers.*

Le mot *NetQi* vient de la contraction du mot anglais *Net* qui signifie réseau et du mot chinois *Qi* qui peut se traduire comme le flux d'énergie. *NetQi* se comprend donc comme " le flux d'énergie du réseau ". Ce qui fait écho à la mission de l'outil qui est d'aider à anticiper et contrer les événements qui perturberaient le flux de données réseau généré par le trafic des utilisateurs.

NetQi est composé de deux parties, un moteur d'analyse écrit en C pour des raisons de performances et une interface graphique écrite en Java. L'analyseur prend en entrée un fichier de jeu et retourne le résultat sur la sortie standard ou sous forme de deux fichiers XML. Les fichiers XML de sortie sont utilisés pour interfacer *NetQi* avec d'autres outils. Ils servent aussi pour l'interface graphique, comme on peut le voir sur le schéma de la figure 4.1, qui présente une vue d'ensemble du fonctionnement de *NetQi*.

L'interface graphique fournit à l'utilisateur un moyen simple et pratique de maninupler les anticipation games(Voir figure 4.2 pour une capture d'écran).

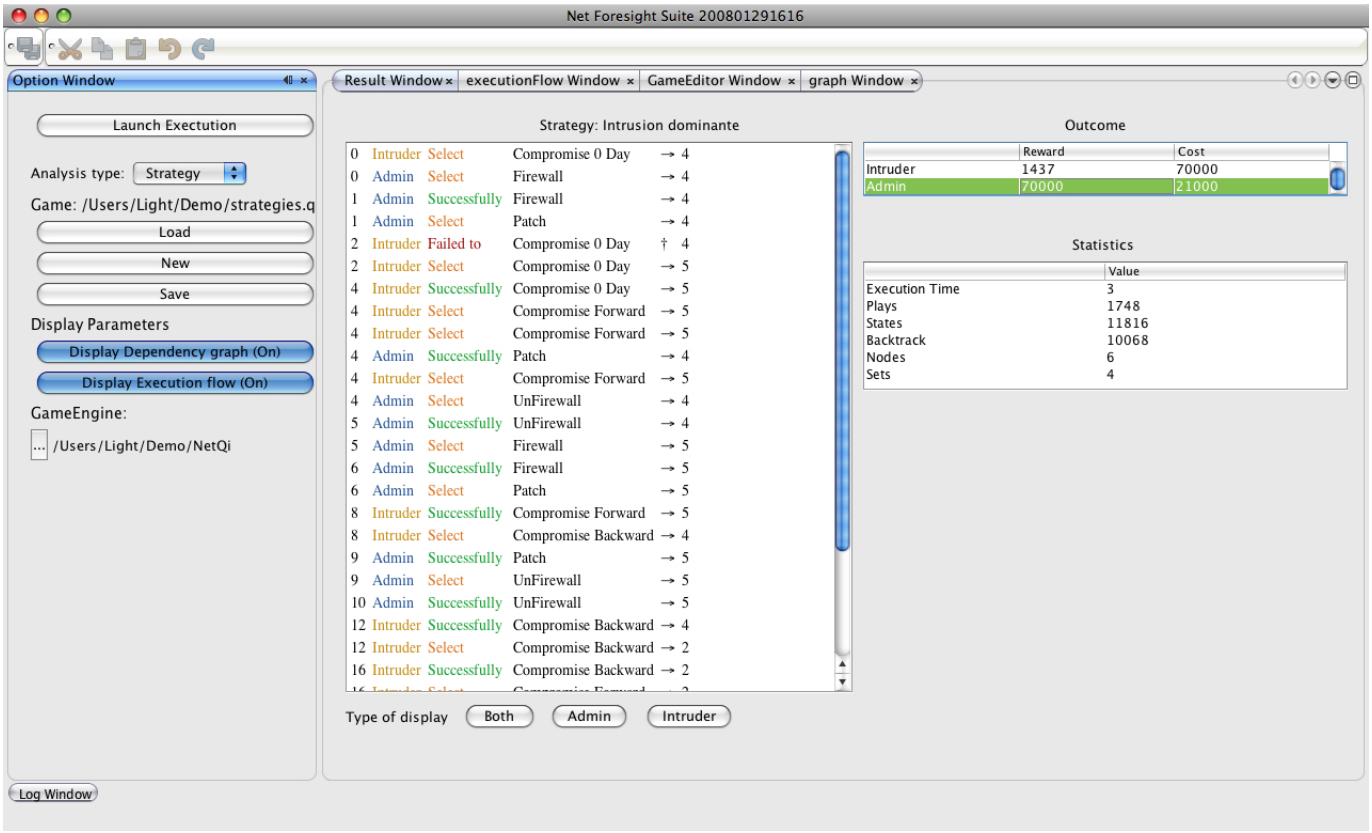

FIGURE 4.2 - *Capture d'écran de l'interface graphique de* NetQi.

Le moteur d'analyse de *NetQi* est un model-checker TATL spécialement adapté aux anticipation games. En particulier, le fait que les anticipation gamesutilisent une description compacte a rendu impossible la réutilisation de l'autre model-checker TATL existant Uppaal-Tiga [BCD+07, CDF+05]. Les différences majeures entre ces deux model-checkers sont :

- *NetQi* utilise une représentation compacte du modèle basée sur notre logique modale alors qu'Uppaal-Tiga utilise un modèle explicite.
- *NetQi* effectue une analyse online qui lui permet d'être très peu gourmand en mémoire. C'est-à-dire qu'au lieu de construire le jeu, puis de le vérifier, il construit et analyse le jeu en même temps. Cela lui permet "d'oublier" les branches d'exécution précédentes.
- *NetQi* supporte un mode d'analyse spécifique appelé Stratégie qui sera présenté au prochain chapitre.
- Uppaal-Tiga implémente l'ensemble des contraintes d'horloges disponibles en TATL alors que *NetQi* n'implémente que les contraintes autorisées par les anticipation gamesc'est-à-dire les contraintes de temps minimum.

Il est à noter que la version actuelle de *NetQi* ne gère pas les délais d'exécution. Autrement dit, il considère que lorsqu'une action peut être exécutée, elle l'est immédiatement. Cependant, cette limitation n'est pas bloquante pour l'analyse : il suffit d'augmenter le temps d'exécution de la règle pour introduire un délai dans l'analyse. *NetQi* et tous les exemples de cette thèse sont disponibles à l'adresse suivante : `http ://www.netqi.org`.

4.2 Premier exemple : l'élément de surprise

Ce premier exemple va permettre d'illustrer la structure d'un fichier de jeu, et d'un résultat d'analyse. Il va aussi nous permettre de mettre en valeur la notion d'élément de surprise présent dans le modèle.

4.2.1 Situation de départ

Le jeu que l'on cherche à analyser dépeint la situation de départ suivante : l'administrateur et l'intrus sont en compétition sur un seul service. L'administrateur dispose de deux moyens de patcher le service, un rapide et un lent, à l'opposé, l'intrus dispose d'un unique moyen pour compromettre le service. Cette situation est représentée sur le schéma 4.3.

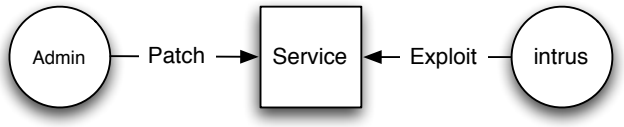

FIGURE 4.3 - *Schéma représentant le scénario de l'exemple 1*

4.2.2 Fichier de jeu

Structure d'un fichier de jeu Un fichier de jeu est composé de 6 parties :

1. Les options qui permettent d'indiquer le nombre de sommets du graphe de dépendances ou encore le timeout de l'analyse.
2. L'ensemble des variables d'états et leur situation initiale. Par exemple, c'est là que l'on définit les services compromis.
3. Les règles décrivant les actions que les joueurs (administrateur et intrus) peuvent entreprendre.
4. Le graphe de dépendances qui décrit le monde, c'est-à-dire les sommets et leurs relations.
5. Les pénalités en termes de coût. Celles-ci sont utilisées pour modéliser les coûts liés au temps comme on l'a présenté dans le chapitre 7.
6. Objectifs : ce que l'on cherche à prouver. Par exemple : une propriété de résilience comme proposé dans le chapitre précédent.

Le fichier Le fichier qui décrit le jeu est présenté en figure 4.4.

La section d'option précise uniquement le nombre de nœuds présents dans le réseau (ligne1) via l'option `nodes=1`. D'autres options sont disponibles, comme par exemple `timeout` qui permet de forcer *NetQi* à retourner après x secondes.

```
1) nodes=1

2) <sets>
3)    Vuln:1
4)    Compr:false
5) </sets>

6) <rules>
7)   I:3:Compromise:Vuln=>Compr
8)   A:6:Patch slow:Vuln=>!Vuln
9)   A:1:Patch fast:Vuln=>!Vuln
10) </rules>
11) tatlFormula=(|+Compr:1)
```

FIGURE 4.4 - *Exemple d'un fichier de jeu : l'élément de surprise*

Etat initial La deuxième partie du fichier représente l'état initial de l'état du graphe de dépendances commençant à la ligne 2 par la balise `<sets>` et finissant à la ligne 5 par la balise fermante `</sets>`. Pour définir une variable, il suffit d'écrire son nom. Lorsque *NetQi* interprète le fichier, il considère par défaut que cette variable d'état est fausse pour l'ensemble des sommets du graphe de dépendances. On peut rendre explicite ce comportement en suffixant le nom de la variable par le mot clé `false` comme dans l'exemple à la ligne 4. Réciproquement, si l'on souhaite que la variable d'état soit vraie pour l'ensemble des sommets alors l'on suffixe son nom par le mot clé `true`. Enfin, si l'on souhaite que ce set ne soit vrai que pour un ensemble de sommets donnés, on suffixe le nom de la variable par la liste des sommets pour laquelle le set est vrai comme à la ligne 3 de l'exemple. Dans l'exemple, la variable `Vuln` sert à modéliser qu'au début du jeu le sommet 1 est vulnérable à une attaque et la variable `Compr` pour modéliser les sommets compromis. Au début, évidement aucun ne l'est.

Les règles La troisième partie du fichier est constituée des règles des joueurs. Cette partie est délimitée par les balises `<rules>` (ligne 6) et `<rules/>` (ligne 10). Dans l'exemple on utilise trois règles : une pour l'intrus et deux pour l'administrateur. La syntaxe d'une règle est la suivante :

joueur :temps :precondition=>postcondition :cout@emplacement :emplacement du successeur

Les sept éléments constitutifs d'une règle sont :

1. **joueur** : le joueur qui a le droit d'utiliser la règle.
2. **temps** : le nombre d'unités de temps nécessaires pour exécuter la règle.
3. **préconditions** : l'ensemble de préconditions qui doivent être satisfaites pour que la règle puisse être utilisée.
4. **postconditions** : l'ensemble des effets que l'exécution réussie de la règle a sur l'état du graphe.
5. **coût** : définit le coût d'exécution de la règle. Le coût est notamment utilisé pour la recherche de stratégie comme on l'a introduit au chapitre suivant.
6. **emplacement** : les emplacements sont utilisés pour restreindre l'application de la règle à un ensemble de sommets.
7. **emplacement successeur** : utilisé pour restreindre l'application de la règle à un ensemble de successeurs.

Une version des règles de l'exemple écrites en utilisant la syntaxe du manuscrit est visible en figure 4.5. Dans ce chapitre, nous utilisons des règles qui n'ont ni coût ni emplacement. Des exemples les utilisant seront présentés dans les chapitres 6 et 7 en même temps que les extensions qui les utilisent.

La première règle (ligne 7) est pour l'intrus (I). Elle est nommée *compromise* et a besoin de *3* unités de temps pour être exécutée. Elle dit que si un service est vulnérable (précondition *Vuln*) alors l'intrus peut le compromettre (postcondition *Compr*). La deuxième règle (ligne 8) est pour l'administrateur (A). Elle est nommée *Patch slow* et dit que si un service est vulnérable (*Vuln*) alors l'administrateur peut le patcher (postcondition *!Vuln*)) en 6 unités de temps. Le caractère *!* est utilisé pour dénoter le non logique $\neg$, le tableau de correspondance complet entre caractère et symbole logique est visible en figure 4.6. La troisième règle (ligne 9) est très similaire à la deuxième règle. La différence se situant au niveau du temps d'exécution : 1 unité de temps au lieu de 6. L'utilisation de deux règles pour la même action permet de modéliser différents comportements. Ici, cela permet de modéliser une différence de réactivité de la part de l'administrateur.

1) Γ : **Pre** $Vuln$
$\longrightarrow$ (3, I, Compromise)
Effect $Compr$
2) Γ : **Pre** $Vuln$
$\longrightarrow$ (6, A, Patch slow)
Effect $\neg Vuln$
3) Γ : **Pre** $Vuln$
$\longrightarrow$ (3, A, Patch fast)
Effect $\neg Vuln$

FIGURE 4.5 - *Règles de l'exemple 1 écrites en utilisant la syntaxe du manuscrit*

Notons que, comme on l'a vu dans le chapitre précédent, il est possible de construire des préconditions et des postconditions complexes en utilisant notre logique modale comme nous le verrons dans l'exemple suivant (Section 4.3). Notre exemple n'a pas de section $<graph>$ car nous n'avons pas besoin de définir de relation d'équivalence, de dépendance ou d'emplacement.

Opérateur	Caractère
land	$\wedge$
lnot	!
$\Diamond$	$>$
$\neg\Diamond$	$\sim$

FIGURE 4.6 - *Correspondance entre les opérateurs logiques des règles et leurs caractères*

Propriété La dernière ligne (ligne 11) du fichier est la formule TATL que nous demandons à *NetQi* de prouver. Pour indiquer cette formule, on utilise le préfixe *tatlFormula=*. Le caractère *$* est utilisé pour représenter l'opérateur CTL $\Diamond$ et le caractère *+* indique que l'on souhaite que cela soit vrai pour la propriété. Ainsi, la formule que nous cherchons à prouver s'interprète comme suit : le sommet 1 (*:1*) est (*+*) à un moment (*$*) compromis (*Compr*) quoique fasse les deux joueurs. Notons que, comme l'exemple n'a qu'un seul sommet, l'on aurait pu écrire plus simplement la formule : *$+Compr*. Cette formule demandant à *NetQi* de prouver que tous les sommets sont compromis à un moment quoi que fassent les deux joueurs.

Résultat de l'analyse Il nous reste à lancer l'analyse en invoquant *NetQi* avec la ligne de commande suivante :

```
./NetQi -C -f scenario/these/exemple1.qi
```

L'option `-C` est utilisée pour indiquer que l'on souhaite utiliser le mode model-checking. L'option `-f` est utilisée pour spécifier le chemin de notre exemple. *NetQi* retourne le résultat visible en figure 4.7 suite à cette invocation. Celui-ci indique que la propriété n'est pas satisfaite et fournit la trace de contre-exemple.

```
1)######Property satisfiability is false ########
2)--- Associated trace ---
3)
4)------Start Game------------
5)    0:    Intruder chooses                Compromise on 1
6)    0:       Admin chooses                Patch slow on 1
7)    3:    Intruder executes                 Compromise  on 1
8)------End Game------------
```

FIGURE 4.7 - *Sortie de* NetQi *pour l'exemple 1 en mode model-checker, qui indique que la propriété n'est pas satisfaite avec le contre-exemple associé.*

La ligne numéro 1 nous indique que la propriété n'est pas satisfaite. Les lignes 4 à 6 donnent les étapes du contre-exemple qui invalide la propriété. La ligne 4 se lit de la manière suivante au temps 0 (*0*) l'intrus choisit d'exécuter la règle *Compromise* sur le sommet 1. La ligne 5 indique que l'administrateur au temps 0 choisit d'exécuter la règle *patch slow* sur le sommet 1. La ligne 6 indique qu'au temps 3 l'intrus compromet le sommet en exécutant la règle compromise. Puisque la propriété est violée, l'analyseur arrête ici l'exécution.

Une autre possibilité offerte par *NetQi* est de retourner l'exécution (play) du jeu qui utilise les règles les plus rapides pour les deux joueurs. Cette option va nous permettre d'illustrer l'élément de surprise des anticipation games. Pour utiliser cette option d'analyse sur notre fichier, l'on invoque *NetQi* avec la ligne de commande suivante :

```
./NetQi -G -f scenario/these/exemple1.qi
```

Seule l'option qui spécifie le mode d'analyse diffère par rapport à l'invocation précédente : on utilise cette fois `-G` à la place de `-C` pour demander une unique exécution. Le résultat de l'analyse est visible en figure 4.8.

```
1)------Start Game------------
2)    0:   Intruder chooses              Compromise on 1
3)    0:   Admin chooses               Patch fast on 1
4)    1:   Admin executes              Patch fast on 1
5)    3:   Intruder fails to execute   Compromise on 1
------End Game------------
```

FIGURE 4.8 - *Sortie de* NetQi *pour l'exemple 1 en mode exécution unique, qui montre l'élément de suprise*

Cette fois l'administrateur choisit d'exécuter la règle *fast patch* sur le sommet 1 (ligne 3). Il résulte de ce choix qu'au temps 1, l'administrateur patche le sommet 1 (ligne 4). L'intrus qui n'est pas au courant de l'action de l'administrateur échoue donc à compromettre le service au temps 3 (ligne 5). Il a été pris par surprise par l'administrateur. On le voit, l'élément de surprise intervient lorsqu'un joueur est plus rapide que l'autre. La surprise est donc basée sur la possibilité pour un joueur d'effectuer une action plus rapidement que l'autre.

La sortie standard de *NetQi* n'étant pas adaptée pour un manuscrit, nous utiliserons dorénavant la sortie LateX. Celle-ci a été conçue spécifiquement afin de pouvoir inclure de manière simple les résultats de l'analyse dans un rapport ou un article. Pour obtenir une sortie au format LateX, il suffit de rajouter l'option `-l`. Ainsi la sortie précédente au format LateX donne le résultat visible en figure 4.9

Time	Player	Action	Rule	Target	Succ
0	Intruder	choose	Compromise	1	$\perp$
0	Admin	choose	Patch fast	1	$\perp$
1	Admin	**execute**	Patch fast	1	$\perp$
3	Intruder	`fail`	Compromise	1	0

FIGURE 4.9 - *Exemple de la sortie latex de* NetQi.

4.3 Deuxième Exemple : Vérification d'une propriété TATL

4.3.1 Situation de départ

Comme promis, reprenons maintenant le graphe de dépendances, l'état initial et un jeu de règles tirées du chapitre précédent afin de construire un exemple plus conséquent. Comme visible en figure 4.10, le sous-ensemble de règles retenues reprend les règles utilisées pour modéliser l'intrusion sur un réseau. Par rapport aux règles de l'intrus du chapitre précédent, on substitue à la règle de compromission les règles `CFilePropNotSync` (règle numéro 2) et `CFilePropSync` (règle numéro 3) afin de tenir compte du délai introduit par la synchronisation périodique de *Index.php[1] :2*. Ainsi la règle numéro 3 a besoin de 5 unités de temps pour s'accomplir alors que la règle numéro 4 a besoin de 300 secondes. Le but du jeu est de prouver que l'administrateur est capable de contrer l'intrus et l'empêcher de modifier *Index.php[1] :2* et qu'il est par la même incapable de compromettre totalement le service web.

4.3.2 Fichier de jeu

Le fichier *NetQi* correspondant à ce jeu est présenté en figure 4.11. Le jeu ainsi constitué engendre, comme nous l'avons dit, 327 834 exécutions distinctes. Parcourir l'ensemble des exécutions demande environ 5 minutes à *NetQi* sur un core2 1.6Ghz portable. Trouver le contre-exemple ne demande, par contre, que moins d'une seconde. Des benchmarks effectués sur des exemples beaucoup plus gros, plus de 10 000 services, seront présentés dans les chapitres 6 et 7. Ils nous serviront à montrer que *NetQi*passe à l'échelle.
On obtient le nombre total d'exécutions engendrées en demandant à *NetQi* de prouver une propriété que l'on sait vraie pour le forcer à explorer l'ensemble des exécutions possible. Ainsi ici, l'on a demandé à *NetQi* de prouver que le serveur SSH[1] n'était jamais compromis ce qui est toujours vrai puisqu'il n'est ni vulnérable localement, ni vulnérable à distance.

L'état initial est modélisé par la section `<sets>` entre les lignes 2 et 13. Les règles sont modélisées entre les lignes 14 et 26. On remarque par rapport à l'exemple précédent, l'utilisation dans les préconditions de la conjonction logique $land : \wedge$ et de l'opérateur modal de successeur $\Diamond : >$ comme par exemple à la ligne 18.

On remarque aussi que cette fois, le fichier contient une section `<graph>` entre les lignes 27 et 42. Cette section permet de spécifier deux types d'information.
Premièrement, la correspondance entre les identifiants numériques des sommets du graphe de dépendances et leur nom usuel représenté par une chaîne de caractère (lignes 28 à 34). Les noms usuels sont utilisés pour rendre la sortie de *NetQi* plus intelligible, comme on le voit en figure 4.12.

1) Γ : **Pre** $Avail \wedge Vuln \wedge Pub \wedge \neg Compr$
$\longrightarrow$ (5, I, $Comp_1$)
Effect $Compr$

2) Γ : **Pre** $\Diamond Compr \wedge VulnLocal \wedge Avail \wedge Service \wedge \neg Compr$
$\longrightarrow$ (5, I, $CServProp_1$)
Effect $Compr$

3) Γ : **Pre** $\Diamond Compr \wedge Avail \wedge File \wedge \neg Crypt \wedge \neg Synced \wedge \neg Compr$
$\longrightarrow$ (5, I, CFilePropNotSync)
Effect $Compr$

4) Γ : **Pre** $\Diamond Compr \wedge Avail \wedge File \wedge \neg Crypt \wedge Synced \wedge \neg Compr$
$\longrightarrow$ (300, I, CFilePropSync)
Effect $Compr$

5) Γ : **Pre** $Pub \wedge Service \wedge Vuln$
$\longrightarrow$ (20, A, Deny)
Effect $\neg Pub$

6) Γ : **Pre** $\neg Public \wedge Public \wedge \neg Vuln$
$\longrightarrow$ (20, A, Allow)
Effect Pub

7) Γ : **Pre** $Avail \wedge Vuln \wedge Patch$
$\longrightarrow$ (300, A, Patch)
Effect $\neg Vuln$

8) Γ : **Pre** $Avail \wedge VulnLocal \wedge Patch$
$\longrightarrow$ (300, A, Patch)
Effect $\neg Vuln$

9) Γ : **Pre** $Compr$
$\longrightarrow$ (30, A, Oreco)
Effect $\neg Compr$

FIGURE 4.10 - *Jeu de règles utilisé pour l'exemple 2*

Deuxièmement, les dépendances entre les services entre les lignes 37 et 41. Par exemple, la ligne 37 se lit *2 dépend de 1* et elle représente la flèche entre le serveur `FTP[1]` et le fichier `Index[1] :1` visible sur le schéma 3.1 du chapitre 3. Notons que l'ordre choisi pour spécifier les informations de la section `<graph>` à l'instar de l'ordre des règles de la section `<rules>` est totalement arbitraire et qu'il n'a aucune incidence sur l'analyse.

Enfin, la ligne 43 spécifie la propriété que l'on souhaite prouver : quoi que fassent les deux joueurs, à chaque exécution donc, le fichier `Index[1] :2` est compromis à un moment ou un autre.

4.3.3 Résultat de l'analyse

L'exécution qui sert de contre-exemple à la propriété recherchée est présentée en figure 4.12. Il se lit de la manière suivante : au temps **0** l'intrus (`intruder`) choisit d'exécuter la règle `Comp1` sur le sommet 1 (`FTP[1]`). Au même moment (temps **0**), l'administrateur (`Admin`) choisit d'exécuter la règle `Deny` sur le même sommet (`FTP[1]`). Au temps **5** , l'intrus réussi à exécuter la règle `Comp1` sur le sommet `FTP[1]`. Il choisit de continuer sa compromission en choisissant d'exécuter la règle `CFilePropNotSync` sur le sommet `Index[1] :1` . Au temps **10**, il réussit à exécuter la règle `CFilePropNotSync`. Ce qui correspond au fait que l'intrus a fini de modifier la page et a pu y injecter un exploit qui va lui permettre d'exploiter la vulnérabilité locale de `HTTP[1] :1`. Il choisi donc logiquement de poursuivre au **temps 10** en utilisant le code injecté (règle `CServProp1`) pour compromettre le sommet 4 (`HTTP[1] :1`). Au temps **15**, l'intrus a réussi à compromettre le sommet `HTTP[1] :1`. Il décide de poursuivre son intrusion en tentant de compromettre `Index[1] :1` via la règle `CFilePropSync`. Pour cela, il doit attendre 300 unités de temps, le temps que le fichier `index.php[1] :1` soit répliqué. Pendant qu'il attend, au temps **20**, l'administrateur finit de firewaller `Deny` le sommet 1 `FTP[1]`. Voyant que celui-ci a été compromis, il décide de le restaurer en sélectionnant la règle `OReco`. Il finit d'exécuter la restauration du service `FTP[1]` au temps **50** et décide de restaurer le fichier `HTTP[1] :1` ; ce qu'il achève au temps **80**. L'administrateur choisit ensuite de restaurer `Oreco` le service `HTTP[1] :1` ce qu'il termine au temps 110. Une fois le réseau restauré, l'administrateur décide d'appliquer les patchs sur les services vulnérables en commençant par le service (`HTTP[1] :1`). Pendant que l'administrateur est en train de d'appliquer le correctif, au temps 315, l'intrus est pris par **surprise** : Alors qu'il croyait que la réplication allait lui permettre de modifier le code de `Index.php[1] :2`, cela échoue car l'administrateur a, entre temps, restauré le fichier `Index.php[1] :1`. A partir de ce moment, le service FTP n'étant plus accessible de l'extérieur et l'intrus ne possédant plus d'accès interne, il devient impuissant à effectuer la moindre action. L'exécution se termine par l'administrateur patchant les services vulnérables **FTP[1]**,`HTTP[1] :1`, et `HTTP[1] :2` et remettant le service `FTP[1]` public.

```
1) nodes=6
2) <sets>
3) Avail:true
4) Pub:1,4,5
5) Vuln:1
6) VulnLocal:4,5
7) Compr:false
8) File:2,3
9) Service:1,4,5
10) Crypt:false
11) Synced:3
12) Patch:1,4,5
13) </sets>
14) <rules>
15) //intruder
16) I:5:Comp1:Avail^Vuln^Pub^!Compr->Compr
17) I:5:CServProp1:VulnLocal^Avail^Service^>Compr^!Compr->Compr
18) I:5:CFilePropNotSync:>Compr^Avail^File^!Crypt^!Synced^!Compr->Compr
19) I:300:CFilePropSync:>Compr^Avail^File^!Crypt^Synced^!Compr->Compr
20) //admin
21) A:20:Deny:Pub^Service^Vuln->!Pub
22) A:20:Allow:!Pub^Service^!Vuln->Pub
23) A:300:Patch:Avail^Vuln^Patch->!Vuln
24) A:300:Patch:Avail^VulnLocal^Patch->!VulnLocal
25) A:30:Oreco:Compr->!Compr
26) </rules>
27) <graph>
28) //label
29) 1:FTP[1]
30) 2:Index.php[1]:1
31) 3:Index.php[1]:2
32) 4:HTTP[1]:1
33) 5:HTTP[1]:2
34) 6:SSH[1]
36) //dependencies
37) 2->1
38) 3->6
39) 3->2
40) 4->2
41) 5->3
42)</graph>
43)tatlFormula=($+Compr:3)
```

FIGURE 4.11 - *Fichier* NetQi *correspondant au jeu de l'exemple 2*

Time	Player	Action	Rule	Target	Succ
0	Intruder	choose	Comp1	1 (FTP[1])	$\perp$
0	Admin	choose	Deny	1 (FTP[1])	$\perp$
5	Intruder	**execute**	Comp1	1 (FTP[1])	$\perp$
5	Intruder	choose	CFilePropNotSync	2 (Index.php[1] :1)	$\perp$
10	Intruder	**execute**	CFilePropNotSync	2 (Index.php[1] :1)	$\perp$
10	Intruder	choose	CServProp1	4 (HTTP[1] :1)	$\perp$
15	Intruder	**execute**	CServProp1	4 (HTTP[1] :1)	$\perp$
15	Intruder	choose	CFilePropSync	3 (Index.php[1] :2)	$\perp$
20	Admin	**execute**	Deny	1 (FTP[1])	$\perp$
20	Admin	choose	Oreco	1 (FTP[1])	$\perp$
50	Admin	**execute**	Oreco	1 (FTP[1])	$\perp$
50	Admin	choose	Oreco	2 (Index.php[1] :1)	$\perp$
80	Admin	**execute**	Oreco	2 (Index.php[1] :1)	$\perp$
80	Admin	choose	Oreco	4 (HTTP[1] :1)	$\perp$
110	Admin	**execute**	Oreco	4 (HTTP[1] :1)	$\perp$
110	Admin	choose	Patch	4 (HTTP[1] :1)	$\perp$
315	Intruder	`fail`	CFilePropSync	3 (Index.php[1] :2)	$\perp$
410	Admin	**execute**	Patch	4 (HTTP[1] :1)	$\perp$
410	Admin	choose	Patch	5 (HTTP[1] :2)	$\perp$
710	Admin	**execute**	Patch	5 (HTTP[1] :2)	$\perp$
710	Admin	choose	Patch	1 (FTP[1])	$\perp$
1010	Admin	**execute**	Patch	1 (FTP[1])	$\perp$
1010	Admin	choose	Allow	1 (FTP[1])	$\perp$
1030	Admin	**execute**	Allow	1 (FTP[1])	$\perp$

FIGURE 4.12 - *Exécution correspondant à la trace infirmant la propriété vérifiée par l'exemple*

A aucun moment durant cette exécution, le fichier `Index.php[1] :2`, n'a été compromis. Cette exécution est donc bien un contre-exemple de la propriété que nous cherchions à prouver. On peut remarquer deux choses intéressantes à propos de cette exécution :
Premièrement, sa longueur et sa complexité rendent très difficile son dépliage à la main. Deuxièmement, si cette trace représente un contre-exemple valide, il en existe beaucoup d'autres, ne serait-ce que si l'on intervertit les actions de l'administrateur au temps 10 et 15. Il est donc naturel de se poser la question de savoir s'il existe un contre-exemple en particulier qui soit le plus efficace du point de vue de l'administrateur. C'est cette question de "comment trouver parmi un ensemble d'exécutions la plus satisfaisante ?", qui sera l'objet de l'extension proposée au chapitre 6.

4.4 Points clés

- *NetQi* permet de vérifier automatiquement des propriétés sur les anticipation games.
- L'élément de surprise arrive lorsqu'un joueur prend l'autre de vitesse.
- Les performances de *NetQi*montrent que le framework est utilisable en pratique.

CHAPITRE 5

Anticipation games, du réseau réel au modèle

"By altering his arrangements and changing his plans, he keeps the enemy without definite knowledge. By shifting his camp and taking circuitous routes, he prevents the enemy from anticipating his purpose.."

Sun Tzu, The Art of War. IX.37

Dans le chapitre précédent, nous avons présenté *NetQi*, l'outil que nous avons développé pour analyser automatiquement les anticipation games. Si celui-ci permet de faire le lien entre la théorie et la pratique, il reste encore une difficulté à surmonter pour pouvoir utiliser de manière effective les anticipation gamespour l'analyse de réseaux réels : la construction du modèle.

Ainsi le découplage entre les règles des joueurs et l'architecture du réseau rend l'écriture des règles à la main relativement facile, il n'en va pas de même pour la spécification de l'état initial du réseau. A mesure que la taille du réseau augmente, il devient en effet de plus en plus difficile de spécifier l'ensemble des services, dépendances, et vulnérabilités existants à la main. De plus, ces informations étant par nature dynamiques, il est encore plus difficile de maintenir ces données à jour de manière efficace sans recourir à un processus automatique.

Ce chapitre explore donc les méthodes utilisées pour construire cet état initial. La combinaison de plusieurs outils pour obtenir un diagnostic fiable du réseau étant un sujet de recherche à part entière [VVZK], nous ne pouvons prétendre avoir réalisé plus qu'un prototype qui démontre la faisabilité de l'approche. Ce prototype combine notre outil d'analyse réseau NetAnalyzer [Bur06] et plusieurs outils connus. La construction d'un graphe de dépendances et d'un état initial complet de manière automatique est un axe de recherche où de nombreuses questions restent ouvertes comme nous allons le voir.

Ce chapitre est articulé autour de deux axes qui seront traités tour à tour. D'abord les techniques utilisées pour l'identification des services et des vulnérabilités d'un réseau, puis les techniques utilisées pour l'identification des dépendances existantes dans un réseau. Ce chapitre a donné lieu aux publications [Bur07a] et [Bur08c] ainsi que la réalisation de l'outil [Bur06].

5.1 Identification des services et vulnérabilités d'un réseau

L'identification des services et vulnérabilités constitue la partie la plus facile en raison de la profusion de techniques et l'outil visant à la construction d'une cartographie réseau. La cartographie réseau est un domaine actif depuis les années 80, l'un des premiers outils développé dans ce domaine fut **traceroute** à Berkeley en 1987 par Van Jacobson. Plus récemment, les années 1990-2000 ont vu l'émergence d'outils intégrant l'ensemble des techniques nécessaires à l'identification précise des services [Fyo] et vulnérabilités [Der, Ten06] existants sur un réseau.

5.1.1 Identification des services

Le processus d'identification comprend 5 étapes principales schématisées en figure 5.1. Depuis le début des années 2000, NMap [Fyo] est devenu le logiciel de référence pour effectuer cette tâche. Son succès tient à son code libre sous licence GPL et au fait qu'il est l'unique outil actuel qui intègre l'ensemble des techniques, y compris les plus récentes, nécessaires pour accomplir les 5 étapes du processus d'identification. De plus, NMap dispose d'une sortie XML qui permet à des outils comme *NetQi* d'intégrer les résultats de son analyse de manière simple et robuste. D'un point de vue pratique, on utilise les techniques suivantes pour produire la liste des services existants sur un réseau donné :

1. **Listage des machines** : on utilise la fonction ping standard ICMP, mais aussi TCP pour lister les machines ne répondant pas à l'ICMP en raison des règles de firewall.

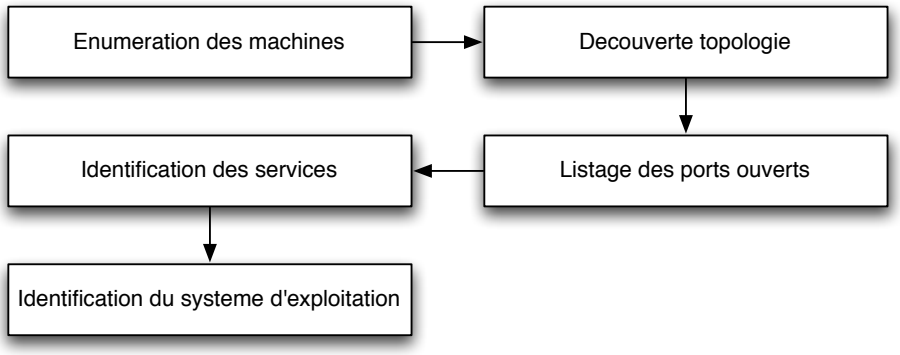

FIGURE 5.1 - *Schéma du processus d'identification des services d'un réseau*

2. **Découverte de la topologie réseau** : depuis la version 4, NMap implémente la fonction traceroute. Celle-ci permet d'identifier les routeurs. On les intègre dans le graphe de dépendances et l'état initial sous forme d'un service routage et d'un jeu de dépendances qui partent des services dépendants du routeur pour faire transiter leur trafic réseau.
3. **Listage des ports ouverts** : NMap implémente l'ensemble des méthodes de balayage de port existant. Cependant, pour des raisons de fiabilité et de rapidité, l'on n'utilise que deux méthodes pour détecter la présence de service, le half-open scan [Mai02] si l'utilisateur a les droits administrateur, et le scan Vanilla sinon. Ces deux méthodes ont, de plus, l'avantage de n'utiliser que des paquets standards et donc évitent à notre processus de construction de générer des alarmes IDS.
4. **Identification des services** : NMap, depuis sa version 4, possède une base de signatures qui permet d'identifier le service tournant sur un port donné. L'utilisation de cette base permet une identification plus fiable que de simplement se référer à la liste des ports standards. En effet, il est possible de faire tourner un service Web sur le port standard SNMP, par exemple.
5. **Identification de l'OS de la machine** : la dernière information nécessaire à la génération de la liste de services est de connaître l'OS [Fyo98, Zal06] de la machine faisant tourner les services. En effet, en fonction des OS, il est possible qu'un même service ne souffre pas des mêmes vulnérabilités. Cette information est donc nécessaire à l'établissement d'une classification précise des services.

Un exemple des informations retournées par NMap pour une machine donnée est visible en figure 5.2. Bien que NMap offre un framework de test extrêmement complet, un certain nombre de tests doivent être ajoutés pour répondre aux besoins spécifiques de *NetQi*.

```
PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 4.7 (protocol 2.0)
80/tcp   open  http       Apache httpd 2.2.8 ((Unix) mod_ssl/2.2.8
631/tcp  open  ipp        CUPS 1.2
3689/tcp open  rendezvous Apple iTunes 7.7.1
Service Info: OS: Mac OS X
```

FIGURE 5.2 - *Exemple d'information retourné par NMap*

Ainsi, il est pertinent d'ajouter un test qui identifie les DHCP présents sur le réseau et qui analyse les options renvoyées. Cela permet d'identifier au minimum quels sont les DNS et passerelles par défaut dont dépendent les clients du réseau. *NetQi* a aussi besoin de connaître les relations de dépendances qui existent entre les différents services DNS sur le réseau. En effet, il est important de savoir quel DNS dépend de quel autre pour résoudre une adresse. Pour découvrir ces dépendances on utilise l'outil GPL `dnstracer`. Un exemple de l'utilisation de cet outil est visible en figure 5.3.

```
Tracing to www.ens-cachan.fr[a] via 192.168.0.42, maximum of 3 retries
192.168.0.42 (192.168.0.42)
 |\___ ns2.nic.fr [ens-cachan.fr] (192.93.0.4) Got authoritative
 |\___ ariane.ens-cachan.fr [ens-cachan.fr] (138.231.176.4)
 |\___ rubis.cri.ens-cachan.fr [ens-cachan.fr] (138.231.110.2) * * *
  \___ dufy.aquarel.fr [ens-cachan.fr] (81.255.96.225) Lame server
```

FIGURE 5.3 - *Exemple d'information retourné par l'outil DNSTracer*

5.1.2 Identification des vulnérabilités

L'identification des vulnérabilités est plus problématique que l'énumération des services présents sur un réseau car la liste des vulnérabilités connues ne cesse de croître de jour en jour comme en témoigne le tableau de la figure 5.4. Maintenir la liste des vulnérabilités connue est un travail ardu qui occupe déjà à plein temps de nombreuses équipes de sécurité dans le monde. Parmi les plus connues, on citera le CERT [CER], le MITRE [MITb] et Bugtraq [Sec]. L'identification de vulnérabilités est encore compliquée par le fait qu'en dépit de son utilité évidente, il n'existe toujours pas de classification unifiée des vulnérabilités. Les deux classifications les plus reconnues à l'heure actuelle sont celles du CERT et du MITRE, nommée CVE [Mita].

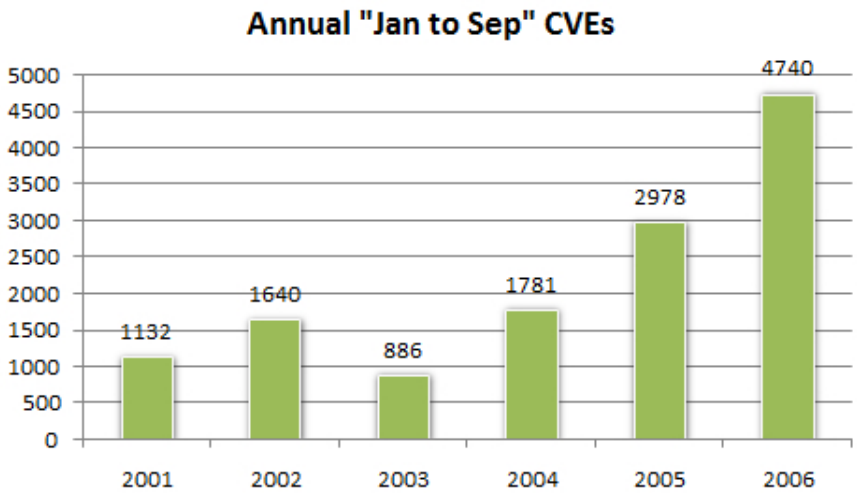

FIGURE 5.4 - *Nombre de vulnérabilités rapporté par le MITRE dans le cadre du projet CVE, par an*

Il existe une classe d'outils appelée scanner de vulnérabilités qui permet de tester si un service est vulnérable en utilisant une base de données de vulnérabilités. Pour ce faire, l'outil pratique un ensemble de tests, généralement un par vulnérabilité, et regarde comment le service réagit. La qualité et la précision de ces outils sont liées à la qualité et la fréquence de mise à jour de leur base de vulnérabilités et la qualité des tests effectués. L'un des premiers scanners de vulnérabilité, si ce n'est le premier, fut SATAN, créé par Farmer et Venema en 1995 [pu95].
Aujourd'hui, les outils les plus répandus pour effectuer une recherche de vulnérabilité sont : Retina [Eey] et Languard [GFI] pour les outils commerciaux et Nessus [Der] et SARA [Cor] pour les outils libres. Tous les quatre possèdent une base de données de vulnérabilités mise à jour quotidiennement.

A l'instar des outils précédents [SHJ+02a] qui construisaient des attack graphs à partir de réseaux réels, nous avons choisi d'utiliser Nessus pour les raisons suivantes : d'abord, son architecture client-serveur permet de le piloter via une interface en ligne de commande, rendant l'automatisation des tests possibles. Ensuite, Nessus permet de récupérer le résultat de son analyse sous forme de fichier XML, ce qui permet une intégration des résultats facile et robuste. Enfin, le moteur d'analyse de Nessus est particulièrement robuste et permet de suspendre puis reprendre une analyse ou encore d'effectuer une analyse différentielle, c'est-à-dire une analyse qui ne teste que les vulnérabilités sorties depuis la dernière analyse. Cette souplesse permet de rendre le processus de création de vulnérabilités flexible.

5.2 Identification des dépendances

La difficulté clé de la génération automatique du graphe de dépendances et de son état initial, est de pouvoir reconstruire le jeu de dépendances existant. La technique qui semble la plus prometteuse pour ce faire est de reconstruire ce jeu en analysant le trafic réseau. En effet, si une dépendance existe entre un service et un autre, alors tôt ou tard, du trafic entre les deux sera échangé. En analysant le trafic réseau pendant une période suffisamment grande, on finit donc par voir du trafic pour l'ensemble des dépendances existantes. Cette approche offre l'avantage de ne pas accroître le trafic du réseau puisqu'elle est entièrement passive. Cependant, la reconstruction du jeu de dépendances à partir d'une trace réseau est difficile pour deux raisons : les techniques de camouflage de trafic et l'identification des séquences de dépendances. Ainsi, la construction du jeu s'articule sur une analyse active du réseau pour les vulnérabilités et les services et une analyse passive pour les dépendances (voir schéma 5.5).

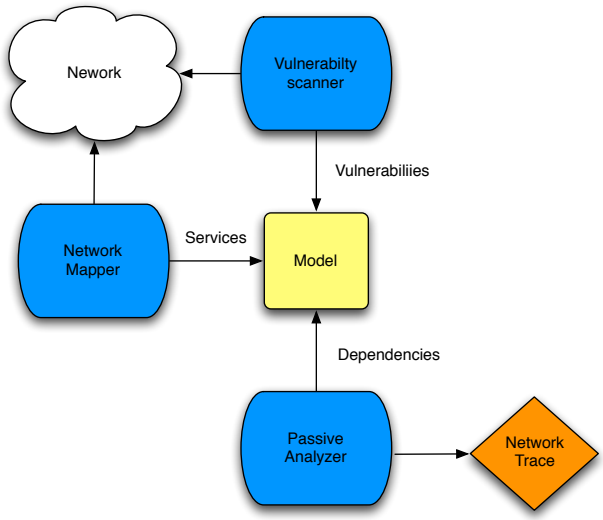

FIGURE 5.5 - *Vue d'ensemble du processus de construction automatique*

5.2.1 Impact des techniques de camouflage de protocole sur la construction du modèle

Un nombre croissant d'applications, telles que les applications de messagerie instantanée MSN messenger [Micb] (Windows Live Messenger) ou les applications p2p [BS06] (Edonkey, Bittorrent, Skype), utilisent des techniques de camouflage de trafic. Parmi ces techniques, les plus courantes sont le détournement de port connu, l'utilisation de port dynamique, le tunneling, et le chiffrement des données.

Ainsi, le bourrage (padding) de paquets est utilisé dans Emule [Pro], Les botnets utilisent des canaux cachés ICMP [Che] et DNS [Tur] et les clients BitTorrent le cryptosystème MSE "Message Stream Encryption" [Azu] conçus spécialement pour empêcher l'identification du protocole. Même pour des protocoles qui de base n'incorporent pas des mécanismes de camouflage, tels que l'IRC ou la messagerie instantanée, il est possible de mettre en place des tunnels pour contourner les restrictions imposées par la politique de sécurité réseau [Dvo]. L'importance croissante que joue l'identification des protocoles camouflés dans la sécurité réseau est soulignée par l'étude [MW06] qui montre que 40% du trafic d'une université n'est pas identifiable, sans une analyse approfondie de contenu.

Ces techniques rendent particulièrement difficile l'identification de la nature des connexions réseau. Or, l'on a absolument besoin de connaître la nature de chaque connexion pour deux raisons. Premièrement, si l'on se trompe sur la nature de la connexion, on court le risque que le résultat de l'analyse soit faussé car on introduit des informations erronées dans le modèle. Deuxièmement, connaître la nature des connexions est nécessaire pour distinguer si une nouvelle connexion est une instance d'une dépendance déjà connue ou si c'est une nouvelle dépendance.

Les techniques de camouflage font courir le risque que l'analyse aboutisse à une conclusion erronée car elles peuvent induire l'analyseur de trafic en erreur et fausser ses résultats. Par exemple, dans le cas d'une application comme Skype qui détourne le port 80, une analyse superficielle du trafic réseau se basant uniquement sur la liste des ports connus conclura à l'existence erronée d'une dépendance HTTP entre le client et le serveur WEB. Si les règles de l'intrus autorisent l'intrus à infecter un client via l'injection d'une page web malicieuse par cross-scripting [Joh06] ou subversion d'iframe [BJM08] alors l'analyse conclura à tort que l'intrus a une stratégie pour infecter le client à cause de la mauvaise caractérisation de dépendance (figure 5.6).

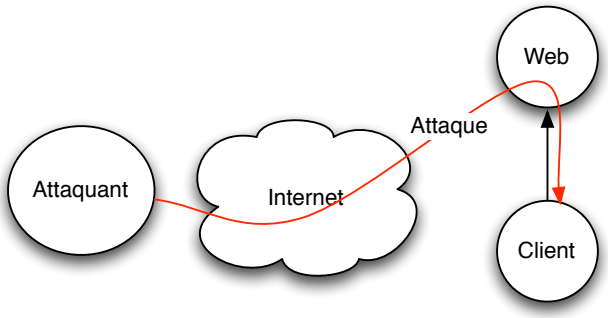

FIGURE 5.6 - *Utilisation d'une dépendance erronée dans la stratégie d'attaque*

5.2.2 Relation entre connexions et dépendances

La correspondance entre l'ensemble des connexions réseau et l'ensemble des dépendances est surjective. En effet, une dépendance pouvant être sollicitée plusieurs fois, il est naturel que plusieurs connexions correspondent à la même dépendance. Par exemple, la dépendance entre un client et un serveur DNS est sollicitée à chaque fois que le client a besoin d'effectuer une résolution DNS. Il est donc nécessaire de pouvoir identifier si une connexion appartient à une dépendance déjà existante ou non.

L'approche naïve consiste à considérer que deux connexions font partie de la même dépendance si et seulement si leur IP source, IP de destination et port de destination sont communs. On exclut le port source car il est aléatoire sur la plupart des architectures modernes. Cette approche est remise en question par la tendance des protocoles modernes à utiliser un port de destination dynamique. Ainsi, deux conversations vidéo MSN messenger n'auront ni le même port source, ni le même port de destination.

Qui plus est, il est impossible d'utiliser uniquement les deux IP comme critère d'agrégation car ce n'est pas assez discriminant. Il a donc fallu trouver une méthode d'agrégation qui soit robuste face à ce type de technique. Les dépendances étant liées à un service offert par un serveur à un ensemble de clients, la méthode d'agrégation retenue consiste à considérer que toutes les connexions de A vers B utilisant le protocole P correspondent à une dépendance unique.

Cette méthode d'agrégation n'étant pas liée à la notion de port, elle est robuste face à l'utilisation de port dynamique et autres techniques de camouflage, telles que l'utilisation de tunnels. En contrepartie, son implémentation implique que l'on soit capable d'identifier la nature de l'ensemble des connexions réseau.

C'est ce besoin d'identification de la nature des connexions réseau qui nous a poussé à travailler sur l'identification de protocole réseau et à développer notre propre outil d'analyse de flux réseau nommé *NetAnalyzer* [Bur06]. Nous avons dû développer notre propre outil d'analyse de flux car les outils existants tels que `NTop` n'offrent pas de méthode d'analyse approfondie des flux. *NetAnalyzer* représente environ 20 000 lignes de C.

Par rapport aux outils existants, ses principaux apports sont : d'une part, l'implémentation d'une analyse approfondie des flux réseau utilisant notre méthode probabiliste et d'analyse de phase de trafic. D'autre part, un moteur de signatures qui permet à l'administrateur de paramétrer l'inspection des payloads et de choisir les informations remontées sans toucher au code, et enfin une analyse des fichiers échangés sur le réseau, qui permet de combattre le trafic illégitime.

NetAnalyzer utilise une architecture multi-thread qui permet de tirer avantage des architectures modernes multicore. La sortie des résultats au format XML permet de réutiliser simplement les résultats. Enfin, le langage de signatures de *NetAnalyzer* permet de s'interfacer avec les bases de signatures de NMap [Fyo] et du projet l7 pour le firewall Netfilter [Qua], et ainsi de disposer de 4 000 signatures.

5.3 Méthodes avancées d'identification de protocoles

Les méthodes d'identification, aussi appelées classification de trafic ou encore inspection avancée (`Deep inspection`), peuvent être regroupées en deux grandes familles, l'identification au moyen de critères discriminants pour construire ce que l'on appelle un *classificateur* [MZ05] telles que la taille des paquets, et la reconnaissance de payloads (données niveau 7), grâce à une base de signatures [SEVS04].

L'identification de protocole utilisant une base de signatures est très proche dans son principe de fonctionnement de celui des NIDS utilisant une base de signatures d'attaques, tels que Snort [Roe] ou Bro [DFM+06]. Ce type d'identification est utilisée pour imposer la Q.O.S (Quality Of Service) dans les routeurs Cisco [Bab05] et le firewall Linux Netfilter [Qua].

Ces dernières années, un important travail de recherche a été effectué sur la génération automatique de signatures car c'est une tâche fastidieuse et difficile [KC03, SEVS04, HSSW05, WCS05]. Il existe aujourd'hui des outils tels que Polygraph [NKS05] et Hamsa [LSC+06] qui sont capables de générer automatiquement des signatures. Des attaques contre ces générateurs de signatures ont été développés, en particulier les "*Allergy attacks*" [CM06].

Les classificateurs utilisent des critères calculés à partir du payload ou des en-têtes des paquets. Les classificateurs les plus courants sont l'entropie [Sha51] du payload, la fréquence des caractères [WCS05], la taille des paquets et l'intervalle entre les paquets [BTA+06]. Il existe aussi des classificateurs propres à un protocole donné, ainsi dans [KV03], 6 critères sont utilisés pour détecter les intrusions dans les CGI HTTP.

5.3.1 Un exemple d'identification

L'exemple présenté en figure 5.7 est le résultat de l'analyse par *NetAnalyzer* d'une session HTTP conduite sur un port non standard. Cet exemple illustre les bénéfices fournis par une analyse approfondie : l'identification précise du protocole utilisé par la session indépendamment de son port et la fourniture d'information avancée sur la session tels que les logiciels utilisés. Le rapport généré par NetQi pour chaque session est composé de 4 parties : les informations principales (ligne 1), les informations de trafic (ligne 2), les informations récupérées grâce à la base de signatures (ligne 3 à 10) et le détail de l'algorithme d'identification (ligne 11 - 12).

```
(1)http (94%): x:1052 -> y.:2080 F:0/2 R::2/0 R:0/2 E:0/0 TCP CLOSED RST
(2)[Traffic] I:1 Kb/s (3pkt) O:37 Kb/s (20pkt) [Distance] C:local S:7
(3)[Protocol]:http (1.1) HyperText Transfer Protocol - RFC 2616
(4)[File] request:www.xxx.org/vip.html Ref:"http://xxx"
(5)[File] content: extension:.html familly:text  (X)HTML
(6)[File] request:www.xxx.org/hello.gif Ref:"http://xxx"
(7)[File] content: extension:.jpg familly:image
(8)[Server] Apache httpd 2.0.52
(9)[Client] browser Internet Explorer 6.0 Windows XP
(10)[Client] proxy squid 2.5.STABLE4-20031106
(11)[Guessed protocol] http:94% Port:0% Class:100% Patt:100%
(12)[Guessed protocol] autodesk:9% Port:100% Class:n/a Patt:0%
```

FIGURE 5.7 - *Exemple de connexion analysée par* NetAnalyzer

Le protocole identifié avec la probabilité la plus haute est affichée dans la partie la plus à gauche de la ligne 1. Ici, c'est le protocole *http* avec une probabilité de 94%. Le score n'est pas de 100% car le port du serveur (2080) n'est pas celui standard assigné au protocole HTTP 80 par l'IANA. Il y a donc un conflit entre le résultat de l'heuristique basé sur les numéros de port standard et l'analyse approfondie.

Ce conflit est visible à la fin du rapport (lignes 11 et 12), où l'ensemble des protocoles possibles sont rapportés. Chaque ligne indique le nom du protocole, sa probabilité, et son score pour chacune des méthodes d'identification utilisée. Ainsi, la ligne 11 indique que la probabilité de 94% pour le protocole HTTP est due au fait qu'il y a eu six signatures reconnues (100% des signatures), que le résultat de l'heuristique est négatif et que le classificateur affiche 100% de résultats positifs. Le score de 100% au classificateur, indique que les 23 paquets de la session ont été reconnus par les profils HTTP, car la valeur des différents discriminateurs est située à l'intérieur des intervalles d'acceptation. La probabilité d'identification par le classificateur de chaque protocole étant indépendante, la somme des probabilités des protocoles peut excéder 100%.

En effet les bornes d'acceptance des profils de certains protocoles se chevauchent. C'est notamment le cas lorsque deux protocoles sont très proches, comme par exemple POP3 et IMAP. Le protocole `Autodesk` n'a qu'une probabilité de 9% car seule l'heuristique est en sa faveur. Le résultat du classificateur pour ce protocole indique n/a car *NetAnalyzer* n'a pas de profil pour ce protocole. Dans ce cas, seul le résultat des signatures et l'heuristique sont pris en compte.

5.3.2 Algorithme d'Identification probabiliste du protocole réseau utilisé par une connexion

Comme illustré dans la section précédente, nous avons développé un algorithme d'identification probabiliste [Bur08c] qui combine les différentes approches existantes afin d'obtenir l'identification la plus fiable possible. Pour ce faire, nous nous sommes appuyés sur les recherches menées dans [HSSW05], qui démontrent que l'identification par signature est plus fiable que l'identification par classificateur qui est elle-même plus fiable que l'heuristique consistant à identifier un protocole en comparant son port de destination avec la liste de ports standards fournis par l'IANA [IAN].

Afin de refléter la différence de précision entre les différentes méthodes lors de la corrélation d'information on utilise une moyenne arithmétique pondérée. Ainsi, la probabilité que la connexion utilise le protocole $\mathbb{P}_x$ est calculée de la manière suivante :

$$\mathbb{P}_x = \frac{\alpha\mathbb{H}_x + \beta\mathbb{C}_x + \gamma\mathbb{S}_x}{\alpha + \beta + \gamma}$$

où $\mathbb{H}_x$ est la probabilité que ce soit le bon protocole d'après l'heuristique basée sur les numéros de ports et α représente le coefficient de confiance que l'on a dans l'heuristique. Comme le port correspond ou non au protocole, $\mathbb{H}_x$ vaut soit 0 soit 100. $\mathbb{C}_x$ est la probabilité que ce soit le bon protocole d'après le classificateur et β représente le coefficient de confiance que l'on a dans le classificateur. $\mathbb{C}_x$ varie entre 0 et 100. Enfin, $\mathbb{S}$ est la probabilité que ce soit le bon protocole d'après l'analyse de payload par signatures et σ est le coefficient de confiance que l'on a dans la base de signatures.

Pour notre analyseur *NetAnalyzer*, on utilise les valeurs $\alpha = 1, \beta = 5, \sigma = 10$. Ces valeurs sont cohérentes avec [HSSW05] et fontionnent bien en pratique. Ainsi, la probabilité du protocole HTTP de l'exemple 5.7 est :

$$\mathbb{P}_{\text{http}} = \frac{1 \times 0 + 5 \times 100 + 10 \times 100}{16} = 93.75$$

5.3.3 Probabilité d'identification liée au classificateur

La probabilité du classificateur est calculée en comparant chaque paquet à l'ensemble de profils générés par clustering [PM00] .*NetAnalyzer*, en raison de la difficulté qu'a le classificateur à distinguer deux protocoles proches, ne rapporte les protocoles identifiés uniquement grâce à celui-ci que lorsque aucune autre identification n'est disponible. Les protocoles ICMP et UDP identifiés sont par contre systématiquement rapportés.

La pratique montre que l'identification par classificateur est en revanche très efficace pour infirmer un résultat, c'est-à-dire détecter lorsqu'un protocole tente de se faire passer pour un autre en détournant son port ou en injectant de faux payloads. La probabilité du classificateur pour un protocole $\mathbb{P}_x$ est le ratio du nombre de paquets reconnu par les profils de ce protocole sur le nombre total de paquets :

$$\mathbb{C} = \frac{success}{total} * 100$$

Dans l'exemple précédent (Figure 5.7), les 23 paquets de la session ont été reconnus par les profils HTTP.
A l'instar de [WCS05], les profils des flux clients et serveurs sont séparés pour améliorer leur précision. Cette séparation est motivée par le fait qu'une session est rarement symétrique : ainsi un des flux est utilisé pour demander des données et l'autre pour les envoyer. Dans l'exemple de la section précédente, le client HTTP demande une image qui est renvoyée par le serveur.

Le classificateur utilise quatre critères de discrimination : la *taille* des paquets, l'*intervalle de temps entre deux paquets* d'un même flux, l'*intervalle de temps séparant l'envoi d'un paquet venant du client et la réception d'un paquet venant du serveur* et enfin *l'entropie du payload* du paquet. Notons que les paquets TCP ACK ne sont pas pris en compte par les profils car ils ne contiennent pas de données niveau 7. L'entropie est calculée en utilisant l'estimateur de Paninsky [Pan03] qui est connu pour être efficace sur une faible quantité de données.

A l'instar de [BTA+06], l'ensemble des profils utilisés pour identifier un paquet est lié à sa position dans le flux car la position détermine souvent le type de données contenues dans le paquet et ce, en particulier, pour les premiers paquets d'un flux. Ainsi, une session POP3 commence par un échange "hello" suivi d'une authentification. Pour chaque protocole, on utilise donc un ensemble de profils distincts pour les 10 premiers paquets et un profil agrégé pour les paquets subséquents.s On a ainsi 22 ensembles de profils distincts par protocole.

Pour chaque discriminateur, une borne supérieure et la borne inférieure sont déterminées à partir de la moyenne et de l'écart-type. Si la valeur du paquet est entre ces bornes, alors le paquet est accepté par le discriminateur. Un paquet correspond au profil s'il est accepté par les quatre discriminateurs. On utilise un ensemble de profils plutôt qu'un simple profil avec des bornes très larges afin d'accroître la précision de l'identification.

Ces profils sont générés automatiquement grâce à un programme Perl que nous avons réalisé. Ce programme utilise l'algorithme classique de clustering par k-mean [PM00]. Les données de base sont extraites automatiquement de la trace réseau en utilisant l'option `-D` de *NetAnalyzer*, qui calcule et affiche les informations nécessaires à la constitution d'un profil pour chaque paquet présent dans la trace réseau. Afin d'éviter des profils qui sont "sur-adaptés" (over-fitness), le nombre maximum de profils par ensemble est limité à 5. Durant le processus de clustering, les discriminateurs qui n'ont pas de sens sont éliminés du profil. L'algorithme considère qu'un discriminateur n'a pas de sens, lorsque l'écart-type reste trop large en dépit du processus de clustering. Par exemple, le discriminateur d'entropie est supprimé du profil si l'écart-type est supérieur à 3.

```
Ping :ICMP :2 :2 :64 :64 :995229 :1004962 : : :7.54564 :7.65728 :
```

FIGURE 5.8 - *ICMP ping packet 1 profile*

La figure 5.3.3 est un exemple de profil utilisé par *NetAnalyzer*. Le premier champ est le nom du protocole : Ping. Le second champ est le nom du protocole niveau 3 : ICMP. Le troisième champ est le flux visé : le nombre 1 dénote le flux serveur, et le nombre 2 le flux client. Le quatrième champ est la position du paquet dans le flux : ici le profil est pour le deuxième paquet du flux. Le reste du profil est constitué des bornes inférieures et supérieures de chaque discriminateur. En premier la taille du paquet, ici les bornes inférieure et supérieure sont égales : 64 octets. En deuxième, l'intervalle de temps entre paquets du même flux qui est d'environ 1 seconde pour l'exemple. En troisième, l'intervalle de temps entre deux paquets de flux opposés, qui a été supprimé de l'exemple car considéré comme n'ayant pas de sens par l'algorithme. Ce qui était prévisible puisque ce délai varie de quelques millisecondes pour un ping entre deux machines d'un réseau local à plusieurs secondes de délai.

Cette variation des délais de réponse n'ayant pas de coupure nette, l'algorithme échoue à trouver des clusters. Enfin, l'entropie du paquet est le dernier discriminateur. Ici, elle oscille entre 7.54 et 7.65. On verra en section 5.4.1 que les profils sont extrêmement efficaces pour identifier les canaux de communication cachés.

5.3.4 Probabilité d'identification liée à l'analyse du contenu des paquets

Intuitivement, l'analyse de payload peut être induite en erreur par l'utilisation d'un tunnel car les premières signatures vont correspondre au protocole de tunnel et non pas au protocole encapsulé. En conséquence, la première chose à faire lorsque l'on s'attache à identifier les tunnels est de faire une identification continue. C'est-à-dire de continuer à inspecter les paquets même lorsqu'une signature est détectée. Ceci n'est pas suffisant car il reste encore à l'analyseur à déterminer quel est le véritable protocole de la session. Pour ce faire, l'on tient compte de l'ordre dans lequel les signatures ont été reconnues. On ne peut pas se baser uniquement sur la dernière signature reconnue car cela ouvrirait la porte à des attaques par injection qui ajouteraient des fausses signatures à la fin de la session. L'analyse de payload prend donc en compte l'ensemble des signatures reconnues en accordant aux dernières un poids plus significatif qu'aux premières. Pour ce faire, l'on utilise une *moyenne mobile pondérée* :

$$\mathbb{P} = \frac{D_{x_i} \times n + D_{x-1} \times (n-1) + ... + D_{x_1} \times (1)}{n + (n-1) + ... + 1}$$

Où D_{x_i} est la valeur de confiance associée à la signature reconnue en position i et n le nombre total de signatures reconnues. La valeur de confiance associée par défaut à une signature est de 100. Elle peut être modifiée très simplement grâce à l'option *confidence* du langage de signature de *NetAnalyzer*. Cette valeur de confiance est particulièrement utile lorsque l'on sait qu'une signature peut engendrer des faux positifs. C'est par exemple le cas de certaines signatures utilisées pour détecter le trafic P2P edonkey qui sont connues pour produire de temps à autre, un faux positif car elle sont très courtes. Le format du protocole edonkey rendant leur amélioration impossible, la seule manière de limiter l'impact des faux positifs générés est de baisser le niveau de confiance que l'on a dans ces signatures.

5.4 Exemples d'identification de technique de camouflage

5.4.1 Identification d'une session utilisant un tunnel

Utiliser un protocole pour en encapsuler un autre est une technique populaire pour passer les restrictions de politique de sécurité imposées par les firewalls. Ainsi, de nombreux clients de messagerie instantanée comme Pidgin ou client IRC proposent cette fonctionnalité en natif. Comme on le présente en section 5.3.4, la moyenne mobile pondérée permet de détecter le protocole réel d'une session utilisant un tunnel en assignant la valeur la plus importante à la dernière signature reconnue.

```
(1) CONNECT irc.********.org:6667 HTTP/1.0
(2) HTTP/1.0 200 Connection established
(3) USER 0 0 0 ::
(4) NICK ****
(5) :irc.********.org 001 **** :Welcome to the **  **!0@xxx
```

FIGURE 5.9 - *An exemple of a tunneled IRC session in a HTTP one*

La figure 5.9 présente une session IRC qui utilise un tunnel HTTP. Les deux premières signatures vont identifier la session comme étant une session HTTP (ligne 1 et 2). Si le processus d'identification s'arrête après avoir reconnu la première signature, cette session est identifiée incorrectement comme étant une session HTTP. Avec l'inspection continuelle, les deux signatures reconnues pour l'IRC (ligne 3 et 5) seront prises en compte. Ainsi l'analyse termine avec quatre signatures reconnues : deux pour le protocole HTTP et deux pour l'IRC. Comme l'algorithme d'identification utilise une moyenne mobile pondérée, les deux signatures reconnues pour l'IRC, auront un poids supérieur et la session sera donc correctement identifiée comme une session IRC. Ainsi, les scores de l'identification par analyse de payload, en sachant que la valeur de confiance de chaque signature est de 100, sont :

$$\mathbb{S}_{\text{http}} = \frac{100+200}{100+200+300+400} = 30\% \qquad \mathbb{S}_{\text{irc}} = \frac{300+400}{1000} = 70\%$$

Ce qui se traduit par les scores d'identification suivants :

$$\mathbb{HTTP} = \frac{100\times 1+30\times 10}{11} = 36,3\% \qquad \mathbb{IRC} = \frac{0\times 1+70\times 10}{11} = 66,6\%$$

Supposons maintenant que la session n'utilise pas un tunnel mais soit, à l'opposé, le résultat du téléchargement de la RFC IRC. Dans ce cas, l'identification a été induite en erreur. Cependant, le calcul précédent ne tient pas compte de l'identification par classificateur. Or, les sessions HTTP et IRC ont des profils très différents.

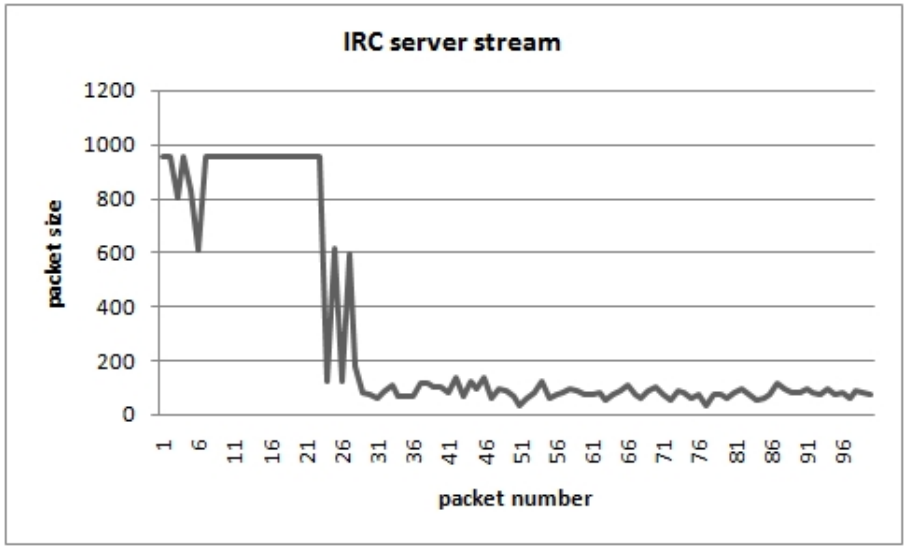

FIGURE 5.10 - *Evolution de la taille des paquets pour un flux IRC serveur*

La figure 5.10 représente l'évolution de la taille des paquets d'un flux serveur IRC, et la figure 5.11 représente l'évolution d'un flux serveur HTTP. Pour le flux HTTP, la taille des paquets est constante après le premier paquet car lorsqu'un fichier est envoyé au client, le comportement standard est de maximiser le débit en envoyant les paquets les plus gros possibles. Ce comportement est connu sous le nom de *"TCP bulk transfert"*. A l'inverse, le protocole IRC étant un protocole dit *"interactif"*, la taille des messages varie énormément en fonction des messages envoyés par les utilisateurs. Il est extrêmement rare que ceux-ci atteignent la taille maximum. En conséquence, les paquets de notre session ne seront reconnus que par les profils HTTP et non les profils IRC. Ce qui mène à la probabilité d'identification suivante :

$$\mathbb{HTTP} = \frac{100 \times 1 + 100 \times 5 + 30 \times 10}{16} = 56,25\% \qquad \mathbb{IRC} = \frac{0 \times 1 + 0 \times 5 + 70 \times 10}{11} = 43,75\%$$

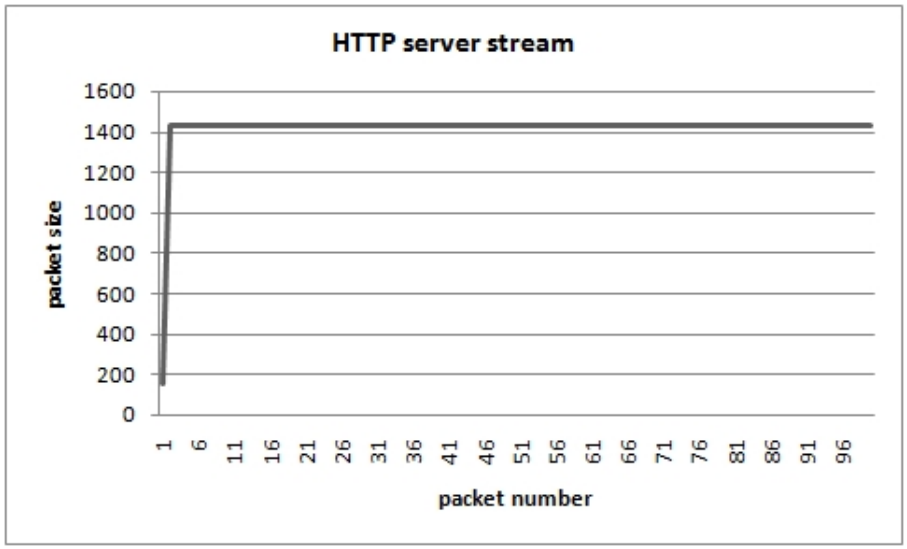

FIGURE 5.11 - *Evolution de la taille des paquets pour un flux HTTP serveur*

5.4.2 Identification d'un canal de communication caché

Un autre problème important est d'identifier les canaux de communication cachés. Les Botnets et malwares les utilisent pour masquer leur présence et contourner les restrictions des firewalls. L'outil de DDOS Stacheldraht [Che] utilise un canal de communication caché ICMP et le rootkit [Tur] utilise en partie un canal de communication caché DNS.

Les canaux cachés ICMP sont difficiles à détecter car d'après la RFC [Pos81] le contenu du payload est arbitraire. Il est donc impossible d'écrire une signature de payload pour ce protocole. De la même manière, les canaux cachés DNS ne peuvent être détectés par signature car ils cachent les informations dans les enregistrement TXT ou encore dans n'importe quel champ DNS de longueur indéfinie. C'est pourquoi, la détection de canaux cachés fait principalement appel au classificateur.

Illustrons comment le classificateur est à même de détecter les canaux cachés en prenant l'exemple d'un canal ICMP. Pour ce faire, on utilise le logiciel PTunnel [Stø] afin d'encapsuler une session SSH dans un canal caché ICMP. *NetAnalyzer* est capable de détecter ce canal car le comportement du canal ne correspond pas au comportement d'un ping ICMP.

Ainsi, deux discriminateurs ont des comportements très différents lorsqu'il s'agit d'un ping réel ou d'un canal caché. Le premier discriminateur à avoir un comportement différent est la taille des paquets. Comme l'on remarque en section 5.3.3, la taille des paquets ICMP reply est constante (schéma 5.12) alors que la taille des paquets du tunnel fluctue de manière importante (schéma 5.13).

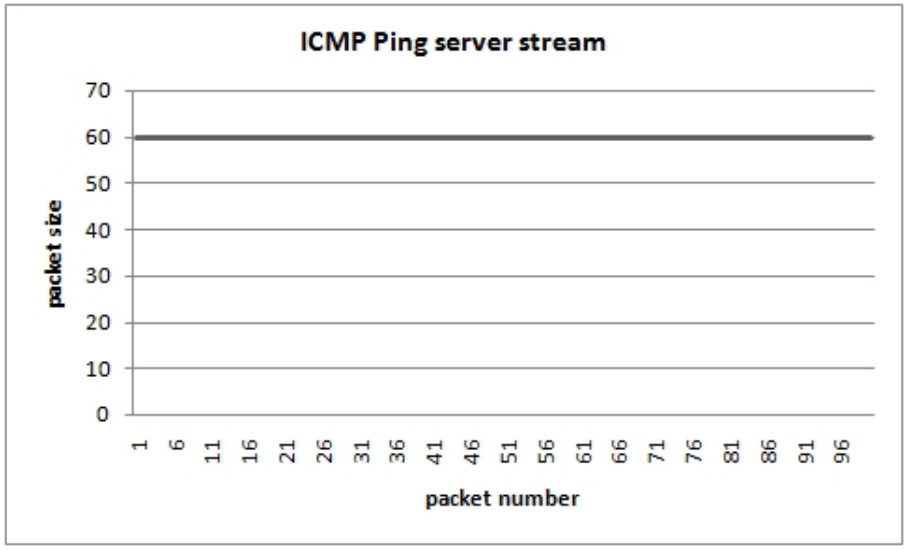

FIGURE 5.12 - *Evolution de la taille des paquets pour un flux écho ICMP reply*

Le second discriminateur qui a un comportement différent est l'intervalle de temps entre deux paquets d'un même flux. Cette valeur est extrêmement stable pour la réponse à un ping ICMP, puisque les requêtes sont envoyées à intervalle régulier (schéma 5.14) alors que l'intervalle du canal caché est totalement imperfectible (schéma 5.15).

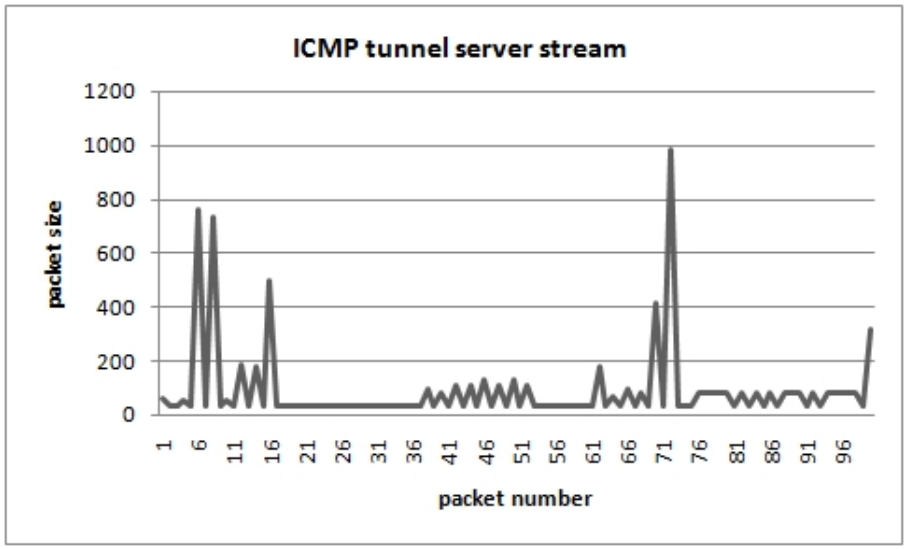

FIGURE 5.13 - *Evolution de la taille des paquets pour un flux Ptunnel serveur*

L'utilisation d'une identification par classificateur est aussi efficace contre les canaux cachés DNS comme l'on montre dans [Kam05]. Le discriminateur basé sur l'entropie étant particulièrement efficace pour les détecter. Cette efficacité vient du fait que les sessions DNS légitimes utilisent un jeu de caractères limité [Moc87], contrairement aux sessions encapsulées. Il résulte de cette différence de comportement que les sessions DNS ont une faible entropie et les sessions encapsulées, une entropie forte.

L'identification par classificateur est efficace parce qu'il est très difficile de contrefaire efficacement tous les aspects du comportement d'un protocole. Cependant, des techniques de masquage utilisant par exemple le bourrage de paquet [FGBZ03] peuvent être utilisée pour induire le classificateur en erreur. Certains protocoles P2P utilisent d'ailleurs une combinaison de techniques de masquage [Azu] tels que le bourrage et l'insertion de fausses données pour plus d'efficacité. Face à cette profusion de techniques, la seule solution est de rajouter un nouveau discriminateur qui n'est pas la cible de ces techniques. Ainsi, pour faire face à la montée en puissance des techniques de camouflage employées par les protocoles P2P, l'introduction d'un discriminateur utilisant la détection de session TCP et UDP simultanées a été proposée dans [KBFc04].

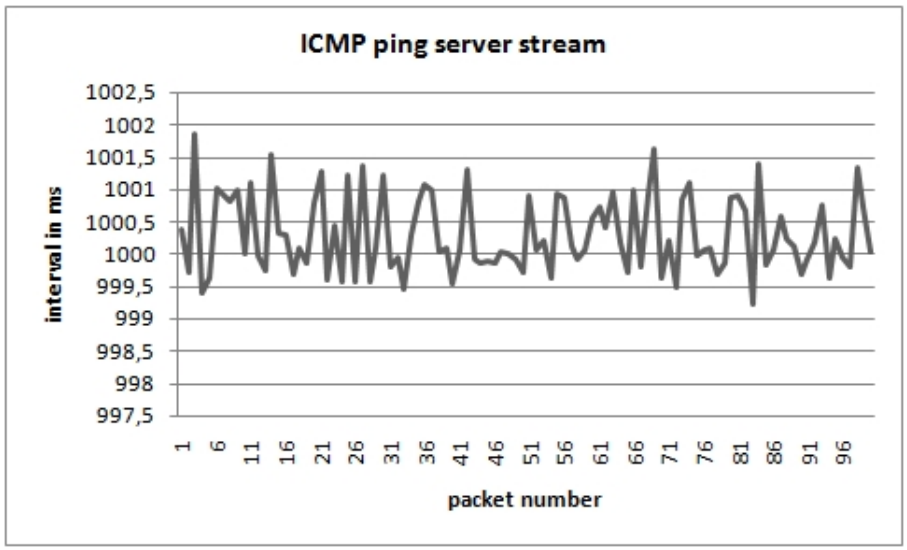

FIGURE 5.14 - *Intervalle entre deux paquets ICMP reply*

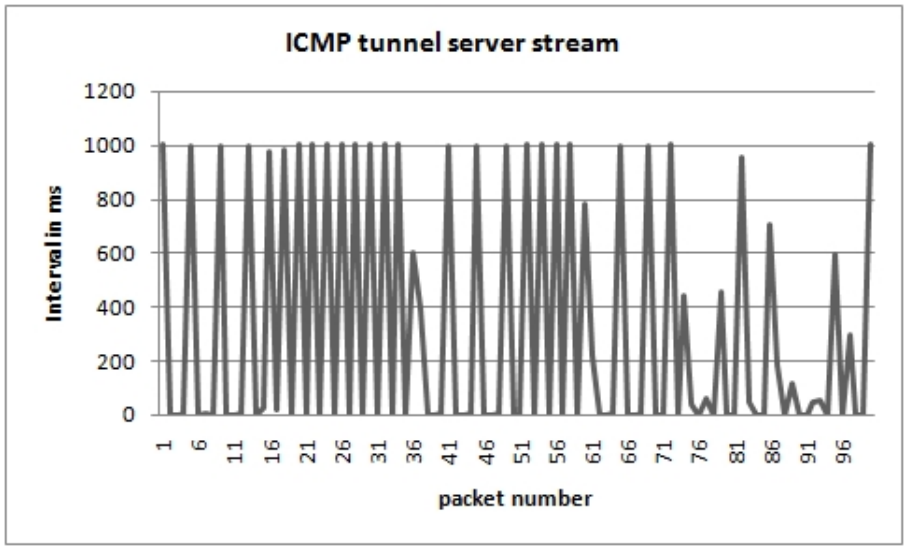

FIGURE 5.15 - *Intervalle entre deux paquets d'un flux serveur PTunnel*

5.5 Identifier les séquences de dépendances

Identifier les séquences de dépendances était, jusqu'à présent, un domaine inexploré que nous avons défriché. On définit une séquence de dépendances de la manière suivante :

Définition 5.1. (Séquence de connexions) *Une séquence de connexions est une succession de connexions réseau qui se déroulent successivement et qui sont toutes nécessaires à l'exécution d'une action utilisateur unique.*

L'identification des séquences de connexion est nécessaire afin de faire apparaître ce que l'on nomme les dépendances "cachées" :

Définition 5.2. *Une dépendance cachée est une dépendance dont l'origine n'apparaît pas directement dans la trace réseau.*

5.5.1 Illustration des dépendances cachées

Illustrons l'existence de ces dépendances cachées en considérant l'exemple de l'affichage d'une page web (Schéma 5.16). Comme on le voit sur le schéma, pour afficher la page web, le client dépend du DNS (`DNS`) pour récupérer l'IP correspondante à l'URL qu'il souhaite afficher, puis il dépend du service web (`www`) pour afficher la page web. A son tour, le service web (`www`) dépend du service de base de données (`DB`) pour générer la page web demandée (schéma 5.17). C'est cette dernière dépendance qui est "cachée", car lorsque l'on regarde l'ensemble des connexions réseau l'on voit une connexion du client vers le service web et plus tard, éventuellement après d'autres connexions sans rapport, une connexion de la machine hébergeant le service web vers le service base de données. Rien dans l'ensemble des connexions observées ne permet de dire directement qu'il existe une relation entre la connexion du client au service web et la connexion du serveur au service base de données (schéma 5.18). Pour mettre en lumière ces dépendances qui ne sont pas matérialisées par une connexion, il faut reconstruire les séquences de dépendances (schéma 5.18). On a une dépendance cachée à chaque fois que dans une séquence deux connexions qui se suivent n'ont pas le même initiateur (schéma 5.18).

5.5.2 Méthode d'identification

Puisque qu'aucun travail sur le sujet n'existait, nous avons cherché à nous inspirer de problèmes similaires ayant déjà été traités dans d'autres domaines. Il apparaît, au terme de cette démarche, que les problèmes les plus proches sont l'identification de gènes à partir d'un corpus ADN en recherche génétique [SBT01] et l'extraction automatique de mots à partir d'un corpus [NM]. Ces deux méthodes utilisent des chaînes de Markov d'ordre supérieur, mieux connues sous le nom de *N-Grams*, introduites par Shannon dans [Sha51]. La définition générale d'un N-Gram est :

Définition 5.3. *Un N-Gram est une sous-séquence de* n *objets consécutifs d'une séquence donnée.*

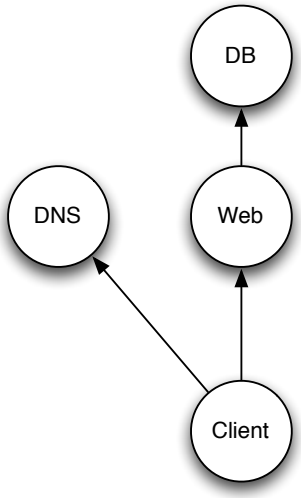

FIGURE 5.16 - *Etapes nécessaires pour la consultation d'une page Web*

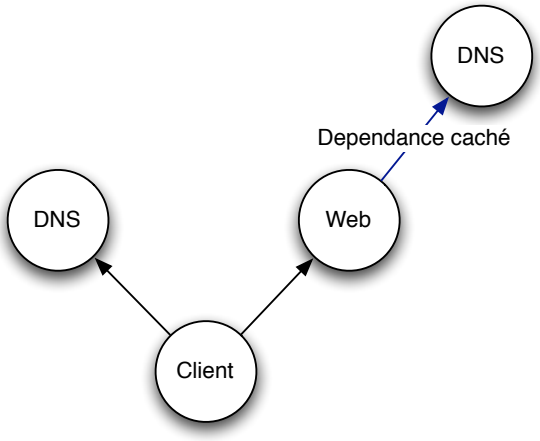

FIGURE 5.17 - *Dépendances associées à la consultation d'une page web*

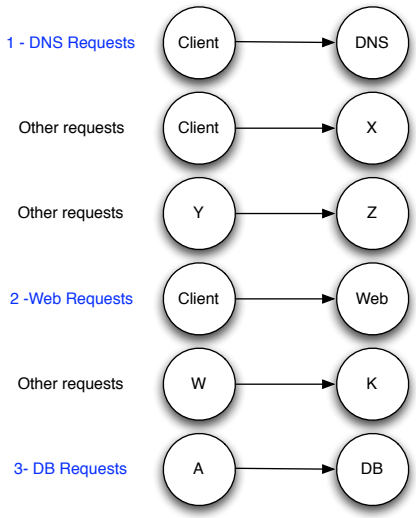

FIGURE 5.18 - *Schéma symbolisant la trace réseau capturée durant la consultation de page web*

Remarquons que les connexions réseau peuvent être considérées comme strictement ordonnées par le temps car les médium réseaux n'autorisent la transmission que d'un paquet à la fois. Ainsi, même si deux connexions semblent concurrentes, en réalité leurs paquets réseaux sont émis de manière séquentielle et entrelacée. Cette observation nous conduit à définir les N-Grams, dans le contexte de connexion réseau, comme suit :

Définition 5.4. *Un N-Gram est une sous-séquence de* n *connexions réseau consécutives présentes dans l'ensemble des connexions réseau ordonnées par le temps.*

A première vue, l'analogie entre le problème d'identifier des gènes à partir d'une succession de base ADN et le problème d'identifier les séquences de connexion à partir d'une succession de connexions réseau laisse entrevoir la possibilité de réutiliser les mêmes algorithmes d'identification. Cependant, l'analogie entre ces deux problèmes a une limite majeure, les séquences ADN sont strictement séquentielles, alors que les séquences de connexions réseau sont entrelacées en raison de la nature conccurente du réseau. Il a donc fallu inventer une méthode pour désentrelacer les séquences réseau.

Il est intéressant de noter que là encore, l'analogie entre ADN et réseau nous a servi de point de départ. En effet, on peut remarquer qu'à l'instar des gènes qui possèdent un codon start, c'est-à-dire une suite de bases ADN dénotant le début d'un gène, nos séquences de connexion possèdent elles aussi une connexion de démarrage. Cette connexion de démarrage correspond à l'action initiée par l'utilisateur ou le planificateur de tâche de son PC. Si on est capable d'isoler les connexions de démarrage, il devient facile de dévoiler les dépendances cachées en utilisant la cible de la dépendance d'origine.

Pour détecter les N-Grams commençant par une connexion de démarrage, il a fallu trouver un critère discriminatoire qui permet de faire la différence entre une connexion de démarrage et une connexion subséquente. Ce critère s'appuie sur le facteur temporel suivant : les ordinateurs réagissent plus vite que les humains.

En effet, il est raisonnable de considérer qu'un humain effectue, au plus, une action toutes les 100 millisecondes. Cette estimation est conservatrice puisqu'elle s'appuie sur la mesure du nombre d'actions par minute qu'effectue un joueur d'esport professionnel lors de parties acharnées de Warcraft 3. Cette mesure montre qu'au maximum ceux-ci effectuent 150 actions par minute soit une action toutes les 400 ms. A l'opposé, nous avons mesuré que le temps de réaction d'un ordinateur est de l'ordre de la dizaine de millisecondes. Il existe donc un facteur d'au moins 10 entre le temps d'exécution d'un être humain et le temps d'exécution d'un ordinateur. Cependant, ces temps ne sont que les bornes inférieures, en effet, il est probable que l'intervalle entre deux actions humaines soit beaucoup plus espacé, mais ceci ne change rien à notre méthode d'identification. En revanche, ce qui est plus problématique, c'est qu'en fonction de la congestion du réseau, du débit de la machine ou encore de la charge du serveur, il arrive que l'intervalle entre deux connexions dépendantes soit supérieur. Afin de gérer la fluctuation de l'intervalle de temps séparant deux connexions dépendantes, on fixe donc une borne supérieure d'acceptation de manière empirique à 5 secondes. Trouver une méthode d'évaluation plus précise de cette borne supérieure est un sujet ouvert.

5.5.3 Cas réel

Afin d'illustrer la disparité qui existe entre le temps d'exécution humain et le temps d'exécution machine, on a choisi de présenter les séquences de connexions générées par l'enregistrement (login) d'un client Pidgin [Pid] au service de messagerie instantanée de Microsoft, plus connu sous le nom de MSN messenger.

Cet exemple met aussi en valeur le problème de fluctuation de l'intervalle de temps entre deux connexions dépendantes. Un listing partiel des connexions réseau remontées par NetAnalyzer présentant nos tentatives réussies d'enregistrement est présenté en figure 5.19.

Pour des raisons de clarté, nous avons identifié les machines par leur IP plutôt que leur nom d'hôte DNS. De même, toutes les connexions du listing utilisant le protocole de transport TCP étant dans l'état fermé au terme de la capture, ces indications ont aussi été supprimées du listing.

Une ligne de ce listing se lit de la manière suivante. Le nombre sur la gauche représente l'heure en secondes et microsecondes depuis le 1er janvier 1970. Il est suivi de l'ip du client, qui est suivi du symbole de séparation -> , qui est suivi de l'ip du serveur, qui est suivi du symbole de séparation :. Enfin, la chaîne la plus à droite de la ligne représente l'acronyme du protocole utilisé par la connexion. Il y a trois protocoles représentés sur le listing HTTPS, MSNM (MSN Messenger) et enfin WHO, un protocole UNIX sans rapport avec nos tentatives de connexions.

```
...
1157724894.631646  138.x.y.72 ->   65.54.179.228:  HTTPS
1157724896.388296  138.x.y.72 ->   207.46.26.121:  MSNM
1157724896.388728  138.x.y.72 ->   207.46.26.120:  MSNM
1157724896.388880  138.x.y.72 ->   207.46.26.122:  MSNM
1157724896.389004  138.x.y.72 ->   207.46.26.119:  MSNM
1157724896.544729  138.x.y.72 ->   207.46.26.110:  MSNM
1157724896.935850  138.x.y.72 ->   207.46.26.171:  MSNM
...
1157724927.268604  138.x.y.72 ->   65.54.179.228:  HTTPS
1157724929.963989  138.x.y.72 ->   207.46.26.121:  MSNM
1157724929.964110  138.x.y.72 ->   207.46.26.120:  MSNM
1157724930.128004  138.x.y.72 ->   207.46.26.111:  MSNM
1157724930.128125  138.x.y.72 ->   207.46.26.113:  MSNM
...
1157724943.810032  138.x.y.72 ->   65.54.179.228:  HTTPS
1157724944.461102  138.x.y.72 ->   138.x.y.255:    WHO
1157724944.486470  138.x.y.72 ->   207.46.26.121:  MSNM
```

FIGURE 5.19 - *Listing des connexions générées lors de l'enregistrement au service MSN messenger*

On voit sur le listing dans les deux premières parties que la même sous-séquence de connexion se répète lorsqu'un client s'enregistre sur le service MSN messenger mais qu'à partir de la 4ème connexion, les IP contactées divergent. Cette divergence est causée par le mécanisme de répartition de charge entre plusieurs serveurs implémentés dans le protocole.

On observe aussi que le délai entre la connexion HTTPS et la première connexion MSNM est bien loin de la borne inférieure puisqu'il fluctue entre 1.7 et 2.7 secondes. La troisième partie du listing permet, quant à elle, de mettre en évidence l'entrelacement entre les séquences de connexions. Ici une broadcast WHO vient s'entrelacer avec la séquence d'enregistrement au service MSN messenger.

On utilise un algorithme d'extraction de N-Gram modifié pour extraire les séquences qui apparaissent plusieurs fois et qui correspondent à nos bornes inférieures et supérieures. Pour chaque N-Gram, l'algorithme retourne sa fréquence d'apparition, c'est-à-dire le nombre de fois où il a été vu.

C'est cette fréquence d'apparition qui va permettre de désentrelacer les séquences en éliminant les N-Grams qui sont le résultat de l'entrelacement de plusieurs séquences. Ce filtrage permet aussi d'éliminer le "bruit" réseau telles que les séquences de connexion dues au balayage de machines. En effet, il y a peu d'intérêt à balayer une machine plusieurs fois, et de plus, l'ordre des ports testés est généré de manière aléatoire par la plupart des outils de balayage à l'instar d'NMap.

Si l'on se base sur le postulat qu'un utilisateur effectuera toujours des actions plus lentement qu'un ordinateur, il est alors possible de calculer la probabilité d'entrelacement entre deux séquences de connexion de la manière suivante :

$$\frac{L}{N}$$

où L est l'intervalle entre deux connexions dépendantes et N est l'intervalle entre deux actions utilisateurs. Ainsi, si l'on reprend notre borne supérieure d'acceptation pour L (5 secondes) et que l'on estime qu'un utilisateur effectue une action toutes les 15 secondes ($N = 15$) alors la probabilité qu'un N-Gram vu 3 fois soit le résultat d'un entrelacement est de 3%. En pratique, la probabilité est bien plus faible que cela car pour atteindre ces 3%, il faudrait que l'utilisateur déclenche les deux actions entrelacées exactement à la même période 3 fois de suite.

Il est d'ailleurs probable que si un tel comportement existe, il est l'indicateur d'une relation de causalité qui dépasse le cadre du réseau (et donc de notre travail). On pense par exemple, aux gestes routiniers de l'utilisateur se connectant le matin, qui clique sur son navigateur et son client mail pratiquement au même moment, et ce, tous les jours.

L'utilisation d'un algorithme de N-Gram pour l'extraction de séquences de connexions est donc une heuristique qui est relativement fiable et qui s'avère être efficace en terme de temps de calcul comme nous allons le voir.

5.6 Complexité de la construction automatique

Si, au cours de ce chapitre, nous avons montré que via un ensemble de techniques il était possible de construire un graphe de dépendances et un état initial de manière automatique, il nous reste à aborder la question de la complexité du processus. Le nombre de machines, de services par machine et de vulnérabilités par machine étant fini, la complexité de l'identification des services et vulnérabilités est linéaire. Plus précisément, il est dans le pire cas de :

$$O(131070 \times m \times v)$$

où m est le nombre de machines et v le nombre de vulnérabilités. La constante 131070 vient du fait qu'il y a au plus 65 535 services TCP et 65 535 services UDP par machine.

La construction de la liste des connexions à partir d'une trace réseau est linéaire dans le nombre de paquets présents dans la trace. L'identification du protocole de chaque connexion fait appel à du pattern matching d'expression régulière et à un classificateur basé sur les entêtes réseau. Cette identification est donc linéaire en le nombre d'expressions régulières testées et le nombre de paquets. Pour les N-Grams, l'extraction ce fait en $O(2n\log(n))$[NM]. La substring réduction est aussi linéaire $(O(n/2))$. Enfin le parcours des N-Gram pour le filtrage et la reconstruction des dépendances est linéaire en la taille du nombre des N-Gram $O(m)$.

5.7 Résumé

- Il est possible de construire de manière automatique le graphe de dépendances.
- On utilise des méthodes actives pour énumérer les services et les vulnérabilités.
- On utilise une analyse passive du trafic réseau pour les dépendances.
- Les protocoles de nouvelle génération (P2P) demandent une analyse avancée du trafic réseau pour être identifiés correctement.
- La reconstruction de dépendances utilise les N-grams.

Troisième partie

Extensions

CHAPITRE 6

Stratégies pour l'administrateur et l'intrus

"All men can see the tactics whereby I conquer, but what none can see is the strategy out of which victory is evolved."

Sun Tzu, The art of war VII.27

Bien que, comme on l'a vu dans les chapitres précédents, les anticipation games offrent une amélioration substantielle par rapport aux attack graphs en termes d'expressivité, il reste cependant un aspect de la modélisation qui rend l'analyse difficile à mettre en œuvre pour la sécurité réseau : la spécification du but de l'analyse. Jusqu'à présent, toutes les approches à base de model-checking visent à prouver qu'une propriété de sécurité est satisfaite pour un modèle donné. Ainsi, si l'on reprend l'exemple 4.2 du chapitre 4, le but a été de prouver qu'à aucun moment le service 1 n'a été compromis.

Cependant, définir le but de l'analyse sous forme de la vérification de propriété de sécurité pose les problèmes suivants.
Premièrement, lorsque l'on analyse un grand réseau, il est extrêmement difficile de savoir pour quels services les propriétés de sécurité doivent être prouvées, il est probable que certains services clés seront sous-estimés parce que la topologie est trop complexe.
Deuxièmement, même s'il était possible de définir des propriétés à vérifier qui couvriraient l'ensemble des services clés de manière fiable, un obstacle majeur subsisterait : si la propriété n'est pas vérifiée par exemple, s'il existe une attaque, alors il est probable qu'il existe une multitude de contre-exemples c'est-à-dire d'exécutions qui contredisent la propriété.

Or, la plupart des questions que l'on se pose en sécurité ne demande pas de trouver un contre-exemple quelconque dans l'ensemble des contre-exemples existants, mais un bien spécifique.

Par exemple, une des questions clés que se pose tout administrateur est : "*Dans le pire des cas, combien de services seront compromis ?*". Cette question traduite en termes de model-checking revient à choisir, parmi l'ensemble des exécutions qui sont des contre-exemples, celle qui touche le plus de services. Or, il est impossible de spécifier un tel objectif en termes de propriétés de sécurité.

C'est ce besoin qui nous amène à introduire une nouvelle forme de but que nous appelons *objectifs de stratégie*. Ceux-ci permettent de définir le but de l'analyse en termes de contraintes mais aussi en termes de coûts, récompenses et temps. Ceci va permettre que l'analyse retourne non plus une exécution quelconque parmi l'ensemble des contre-exemples mais le contre-exemple le plus intéressant, nommé *stratégie*, que l'on définit comme suit :

Définition 6.1. Stratégie *: pour un ensemble d'objectifs numériques de stratégie et un modèle donné, on appelle stratégie optimale l'exécution qui satisfait les contraintes et qui maximise les objectifs numériques.*

Le but de l'analyse passe donc de la vérification d'une propriété à la recherche d'une stratégie qui est la meilleure réponse à un problème donné. Notons que l'on reste dans le cadre du model-checking car l'on continue à explorer toutes les exécutions possibles du jeu pour déterminer quelle exécution est la meilleure stratégie.

L'introduction des coûts et des récompenses dans le modèle permet de prendre en compte la dimension financière de l'attaque, qui est un aspect essentiel de l'analyse. Avec cette extension, l'anticipation game est le premier Framework à prendre en compte la dimension économique et temporelle des attaques en même temps. Prendre en compte la dimension financière permet par exemple de calculer le coût d'une attaque, les pertes engendrées et l'investissement nécessaire pour la prévenir.

Ce chapitre est organisé comme suit : d'abord nous introduisons la notion de coûts et de récompenses dans le modèle. Puis, nous présentons la notion d'objectifs de stratégie. Enfin nous étudions la décidabilité et la complexité de cette extension. Nous concluons en évaluant, grâce à *NetQi*, l'effectivité de l'approche en pratique. Ce chapitre a fait l'objet de la publication [BM09].

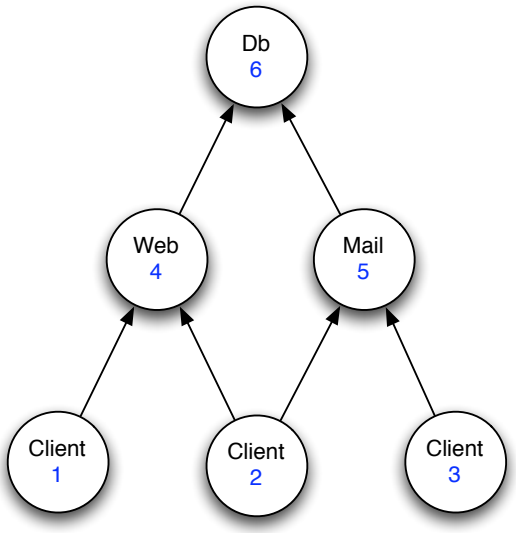

FIGURE 6.1 - *le graphe de dépendances utilisé pour l'exemple*

6.1 Ajout des coûts et récompenses aux Anticipation Games

Dans cette section, on va détailler les extensions apportées au modèle pour pouvoir rechercher des stratégies. Ces modifications concernent principalement l'ajout de la notion de récompenses et de coûts au niveau des règles et du graphe de dépendances. Ces changements sont illustrés au travers d'un exemple dont le graphe de dépendances est présenté en figure 6.1. Comme dans les exemples précédents, les arcs entre les sommets sont utilisés pour représenter les dépendances entre les différents services. Par exemple, l'arc orienté qui relit le sommet *Mail (5)* au sommet *DB (6)* est utilisé pour indiquer que le serveur de courriel (*Mail*) dépend de la base utilisateur (*user database*) pour identifier ses clients.

Le mappage de variables qui définit l'état initial du réseau est présenté sur la figure 6.2. Ce mappage indique que les services *Mail* et *Web* doivent être publics, sont vulnérables et publics car (ρ(NeedPub)), ρ(Vuln), et (ρ(Vuln)) renvoient vrai ($\top$) pour chacun d'eux. Cela indique aussi qu'aucun sommet n'est compromis au départ puisque ρ(Compr) renvoit faux ($\bot$) pour chaque sommet. Enfin, l'ensemble ρ(NeedPub) est utilisé par la règle `Unfirewall` pour savoir quel sommet doit être rendu public.

States	1	2	3	4	5	6
$\rho(\mathsf{Vuln})$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\bot$
$\rho(\mathsf{Public})$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\bot$
$\rho(\mathsf{Compr})$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\rho(\mathsf{NeedPub})$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\bot$

FIGURE 6.2 - *L'état initial du réseau utilisé dans l'exemple*

La manière dont les actions de joueurs sont décrites ne change pas. On utilise toujours un jeu de règles par joueur. Les règles sont toujours de la forme :

$$\begin{aligned} \Gamma: \quad & \mathbf{Pre} \ : F \\ & \Longrightarrow \Delta, p, a, c \\ & \mathbf{Effect} \ : P \end{aligned}$$

où F est l'ensemble de préconditions, Δ le nombre d'unités de temps nécessaire pour exécuter la règle, p le nom du joueur, a le nom de l'action, $c \in \mathbb{N}$ le coût de la règle, et P est l'ensemble de postconditions qui décrivent l'effet de la règle. La seule différence par rapport au chapitre précédent est l'introduction de la variable c utilisée pour spécifier le coût associé à chaque action comme nous allons le voir ci-après. Rappelons que l'ensemble de préconditions F ne doit être seulement satisfait quand la règle est lancée mais durant l'ensemble de la durée nécessaire à l'exécution de la règle soit Δ unités de temps.

6.1.1 Coût et récompense

L'ajout de la dimension financière se fait en introduisant la notion de coût et de récompense dans le modèle. La manière la plus naturelle de faire cela est d'associer un coût à chaque règle et une récompense à chaque sommet du graphe de dépendances.
En effet, lier les coûts aux règles est naturel car il est évident que certaines actions demandent des ressources plus importantes que d'autres. Par exemple : analyser le code source d'un logiciel pour trouver une faille et coder l'exploit associé est beaucoup plus coûteux que d'utiliser un exploit déjà existant.

De manière similaire, il est naturel que les récompenses soient associées aux sommets du graphe de dépendances car certains services ont plus de valeur que d'autres. Ainsi, dans notre exemple (figure 6.1), il est évident que la bonne marche du service de base de données *DB* est essentiel. De manière formelle, on définit la fonction $\mathsf{Value}(x) \rightarrow y/y \in \mathbb{N}^*$ qui retourne la valeur y associée au sommet x du graphe de dépendancespour retourner la valeur des récompenses.

Il faut aussi prendre en compte le fait que toutes les règles ne donnent pas forcément droit à une récompense. Par exemple, si l'objectif de l'administrateur est de sécuriser son réseau en supprimant les vulnérabilités existantes alors firewaller un service (l'enlever de l'ensemble `Public`) va effectivement empêcher le service d'être compromis de l'extérieur, mais ceci est une mesure temporaire et ne doit pas offrir de récompense car le problème n'est pas résolu. A l'inverse, patcher le service (l'enlever de l'ensemble `Vuln`) est une mesure permanente qui supprime la vulnérabilité. Cette action permet d'accomplir les objectifs de l'administrateur (au moins en partie) et se doit donc d'offrir une récompense. Pour ce faire, nous utilisons deux types de règles : les règles dites *règles standard* qui ont un coût d'exécution mais n'offre pas de récompense et les règles dit *règles d'octroi* qui ont un coût d'exécution et offre une récompense.

6.1.2 Le jeu de règles

Le jeu de règles utilisé pour notre exemple se concentre sur l'analyse d'intrusion. Le coût et la durée d'exécution associés à chaque règle de notre exemple sont de l'ordre de grandeur de ce qui est communément admis [LWS02]. Cependant, ils ne doivent pas être considérés comme précis car ils dépendent de nombreux paramètres propres à chaque analyse et réseau considérés, telle que la classe de l'attaquant (individu, gouvernement...).

Les sept règles utilisées pour l'exemple sont présentées en figure 6.3. On prend la convention que les règles d'octroi utilisent une flèche double $\Longrightarrow$ et que les règles standard utilisent la flèche simple : $\longrightarrow$. Les règles `Compromise 0day` (1) et `Compromise public` (2) disent que si le sommet est vulnérable (`Vuln`), public (`Public`) et non compromis (`Compr`) alors il peut être compromis. La différence entre les deux est le temps requis pour compromettre le service, deux ou sept unités de temps, et le coût requis pour le faire, 20 000 ou 5 000. L'utilisation de ces deux règles permettent d'exprimer que l'utilisation d'un exploit 0day offre un avantage en terme de temps en échange d'un désavantage en terme de coût. Cela permet de mettre en évidence la relation existante entre l'investissement requis et l'efficacité de l'attaque. Plus une attaque est efficace, plus elle requiert un investissement certain. La règle (7) de l'administrateur `patch` est cohérente avec la notion de 0day car elle est plus lente que la règle de l'intrus `Compromise 0day` et plus rapide que la règle `compromise public`. Ces trois règles modélisent donc ce que l'on nomme la fenêtre de vulnérabilité [LWS02] : l'exploit inconnu est disponible pour l'attaquant, puis l'administrateur a accès au patch et finalement un exploit est rendu public.

La règle `Compromise backward` dit que l'intrus peut utiliser à son avantage une relation de dépendance pour compromettre un sommet qui dépend d'un sommet déjà compromis. Cette règle modélise une attaque qui exploite les relations de confiance. Par exemple, quand un service DNS est compromis, l'intrus peut l'utiliser pour rediriger ses clients vers des sites spoofer (imposteur). Il exploite alors la relation de confiance qui existe entre le service DNS et les clients.

1) Γ : $\mathbf{Pre}$: $Vuln \wedge Public \wedge \neg Compr$
$\Longrightarrow$ 2, I, Compromise 0day, 20000
$\mathbf{Effect}$: $Compr$
2) Γ : $\mathbf{Pre}$: $Vuln \wedge Public \wedge \neg Compr$
$\Longrightarrow$ (7, I, Compromise public, 5000)
$\mathbf{Effect}$: $Compr$
3) Γ : $\mathbf{Pre}$: $\neg Compr \wedge \Diamond Compr$
$\Longrightarrow$ (4, I, Compromise backward, 5000)
$\mathbf{Effect}$: $Compr$
4) Γ : $\mathbf{Pre}$: $Compr \wedge \Diamond \neg Compr$
$\Longrightarrow$ (4, I, Compromise forward, 5000)
$\mathbf{Effect}$: $\Diamond Compr$
5) Γ : $\mathbf{Pre}$ $Public \wedge Vuln$
$\longrightarrow$ (1, A, Firewall, 10000)
$\mathbf{Effect}$ $\neg Public$
6) Γ : $\mathbf{Pre}$ $\neg Public \wedge \neg Vuln \wedge NeedPub$
$\longrightarrow$ (1, A, UnFirewall,0)
$\mathbf{Effect}$ $Public$
7) Γ : $\mathbf{Pre}$ $Vuln \wedge \neg Compr$
$\longrightarrow$ (3, A, Patch, 500)
$\mathbf{Effect}$ $\neg Vuln \wedge \neg Compr$

FIGURE 6.3 - *Jeu de règles*

De manière similaire, la règle `Compromise forward` (4) dit que l'intrus peut tirer avantage d'une relation de dépendance pour compromettre le successeur d'un sommet compromis. Dans notre graphe de dépendances exemple (figure 6.1) si l'intrus est capable de compromettre le serveur d'intranet web, il peut inspecter ses fichiers de configuration et ainsi récupérer les login et mots de passe utilisés pour accéder aux informations stockées en base de données.

La règle `Firewall` (5) dit que si un service est vulnérable (`Vuln`) et public (`Public`), alors l'administrateur peut l'isoler du réseau. Le coût de la règle est très élevé (10 000) comparé à la règle de patch (500) car isoler un service public va effectivement empêcher l'intrus d'y accéder, mais ceci va aussi empêcher l'accès aux utilisateurs légitimes. Ainsi, cette action introduit une perturbation dans l'activité normale du réseau et entraîne une possible perte financière. Notons la flèche simple $\longrightarrow$ utilisée pour cette règle afin de dénoter qu'aucune récompense ne sera octroyée lors de l'exécution de cette règle. Enfin la règle `Unfirewall` (6) est utilisée pour rendre à nouveau public des services qui ne sont pas vulnérables et qui ont besoin d'être publics (`NeedPub`).

6.1.3 Exemple d' exécution du jeu

Comme il est précisé dans le chapitre 4, toutes les exécutions présentées sont générées grâce à notre outil *NetQi*. L'exemple présenté ici semble simple à première vue, mais ne peut à l'instar des exemples des précédents chapitres être analysé à la main, car il génère 4011 exécutions distinctes.

Cet exemple se lit comme suit : Au *temps 0* l'intrus choisit d'utiliser un exploit 0day contre le service web (Target 4). En même temps, l'administrateur décide de d'isoler le service web. Puisque le firewalling est plus rapide que l'exploitation d'une vulnérabilité 0day, l'administrateur est à même de finir d'isoler le service web (*temps 1*) avant que l'exploitation via un exploit 0day ait lieu. Une fois le firewalling effectué, l'administrateur commence à patcher le service web. Au (*temps 2*), l'intrus est pris par surprise par l'administrateur car au moment où il doit compléter son attaque, le service est déjà firewallé ainsi l'exécution de sa règle échoue. Il décide alors de lancer une autre attaque 0day mais cette fois contre le service de mail (cible 5, *temps 2*).
Au *temps 4* l'administrateur a fini de patcher le service web et décide d'enlever le firewall puisque ce dernier n'est plus vulnérable et doit être public. Pendant ce temps, l'intrus compromet le service de mél et décide d'utiliser son nouvel accès pour compromettre la base de données (cible 6) en inspectant les fichiers de configuration du service de mél par exemple. Au *temps 5* l'administrateur décide de patcher le service de mél (cible 4). Au *temps 8* l'intrus à compromis la base de données utilisateurs (cible 6). Au même moment l'administrateur a fini de patcher le service de mél (cible 5). Ainsi au *temps 12* l'intrus échoue à compromettre le client 2 à partir du serveur de mél (Succ 4) car le serveur de mél n'est plus vulnérable et compromis.

Time	Player	Action	Rule	Target	Succ	Payoff	Cost
0	I	choose	Comp 0 Day	4	$\perp$	-	-
0	A	choose	Firewall	4	$\perp$	-	-
1	A	**execute**	Firewall	4	$\perp$	0	10000
1	A	choose	Patch	4	$\perp$	-	-
2	I	`fail`	Comp 0 Day	4	$\perp$	0	20000
2	I	choose	Comp 0 Day	5	$\perp$	-	-
4	I	**execute**	Comp 0 Day	5	$\perp$	31	40000
4	I	choose	Comp For	5	6	-	-
4	A	**execute**	Patch	4	$\perp$	21	10500
4	A	choose	UnFirewall	4	$\perp$	-	-
5	A	**execute**	UnFirewall	4	$\perp$	21	10500
5	A	choose	Patch	5	$\perp$	-	-
8	I	**execute**	Comp For	5	6	1382	45000
8	I	choose	Comp Back	2	5	-	-
8	A	**execute**	Patch	5	$\perp$	52	11000
12	I	`fail`	Comp Back	2	5	1382	50000
12	I	choose	Comp Back	4	6	-	-
16	I	**execute**	Comp Back	4	6	1403	55000
16	I	choose	Comp Back	1	4	-	-
20	I	**execute**	Comp Back	1	4	1404	60000
20	I	choose	Comp Back	2	4	-	-
24	I	**execute**	Comp Back	2	4	1405	65000
24	I	choose	Comp For	2	5	-	-
28	I	**execute**	Comp For	2	5	1436	70000
28	I	choose	Comp Back	3	5	-	-
32	I	**execute**	Comp Back	3	5	1437	75000

FIGURE 6.4 - *Exemple d'exécution qui maximise le gain de l'intrus*

Cependant, l'intrus a toujours accès à la base de données utilisateurs (sommet 6) et il utilise cet accès pour compromettre le serveur web au *temps 16*. A partir de là, il va compromettre le client 1 au *temps 20* et le client 2 au *temps 28*. Il utilise son accès au client 2 pour compromettre le service web à nouveau (sommet 5, *temps 28*) et il finit de compromettre totalement le réseau en compromettant le client 3.

6.1.4 Calculer la valeur des sommets

Il existe de nombreuses manières de calculer la valeur des sommets d'un graphe, c'est même la problématique centrale des moteurs de recherche web : comment calculer la valeur autoritaire d'une page web. C'est pourquoi, nous ne pouvons prétendre donner une solution simple et générale à ce problème. Dans notre travail, on préfère donc se restreindre à présenter trois manières de calculer ces valeurs pour illustrer les approches possibles, en étant bien conscient qu'il existe un vaste champ de recherche à explorer dans cette direction.

La première approche possible pour calculer le coût des sommets consiste à utiliser la même valeur pour l'ensemble des sommets. Bien que cette approche semble naïve et évidente, elle permet en réalité à l'analyse de répondre à la fameuse question : « Dans le pire cas combien y a-t-il de services compromis et quels sont-ils ? ». En effet, lorsque l'on assigne la même valeur à tous les services, maximiser son gain revient à maximiser le nombre de services compromis. Il devient donc possible d'exprimer cette question en recherchant la stratégie de l'intrus qui maximise ses gains.

La deuxième approche possible est de faire appel à l'administrateur pour donner une valeur à chacun de ses services en fonction de sa connaissance du réseau. Cette approche pose, bien sûr, des soucis de passage à l'échelle et induit le risque d'une erreur d'appréciation ou d'oubli.

La troisième et dernière approche est celle employée par les moteurs de recherches web actuels. Elle vise à trouver un ensemble de critères mesurables qui permettent de calculer de manière automatique la valeur supposée des services. Dans ce cadre, nous avons développé un algorithme de classement inspiré des algorithmes de classement de page web tels que le PageRank [PBMW98] (Google) ou HITS [Kle99] (Msn). Cette approche est proche dans l'esprit des métriques développées pour les attack graphs [MBZ+06]. Ainsi, à l'instar de ces algorithmes, nous nous basons sur l'idée que plus un sommet a un degré entrant élevé, plus il a une valeur importante pour le réseau. On considère aussi que plus une dépendance est sollicitée, plus elle est importante. Cela vient de l'idée que les services les plus sollicités sont les plus importants.

Bien que ces choix soient discutables et puissent être remplacés par d'autres approches, ils produisent en pratique des résultats intéressants. On utilise donc, pour calculer la valeur d'un sommet la formule suivante :

$$value = \sum_{1}^{i} vertexValue(i) \times dependencyValue(i)$$

Où $i \in \mathbb{N}$ est le nombre de sommets qui sont des prédécesseurs, $vertexValue(i)$ est la fonction qui retourne la valeur du prédécesseur i et $dependencyValue(i)$ est la fonction qui retourne la valeur associée à la dépendance entre i et le sommet.
La fonction `dependencyValue()` est calculée en utilisant la formule suivante :

$$dependencyValue_i = (occur_i / totalOccur) \times 100$$

Où $occur_i$ est le nombre d'occurrences observées pour la dépendance i pour une période t donné et `totalOccur` est le nombre total d'occurrences observé pour l'ensemble des dépendances durant la période t. Ce qui, appliqué à notre graphe de dépendances d'exemple (figure 6.1) avec les fréquences de dépendances détaillées en figure 6.5, donne le résultat présenté en figure 6.6. On note que, afin d'obtenir une valeur positive pour l'ensemble des sommets, on assigne à chacun d'entre eux une valeur de départ de 1 avant l'application de l'algorithme. De plus, afin d'éviter les boucles l'ensemble des dépendances sortantes du sommet ciblé ne sont pas prises en compte.

Dependence	Valeur
Client 1 → Web server	10
Client 2 → Web server	10
Client 2 → Email server	15
Client 3 → Email server	15
Web server → database server	20
Email server → database server	30

FIGURE 6.5 - *Valeur des fréquences pour chaque dépendance*

Sommet	Valeur
Client 1	1
Client 2	1
Client 3	1
Web server	21
Email server	31
Database server	1351

FIGURE 6.6 - *Valeur associée à chaque sommet*

6.2 Objectifs d'une Stratégie

Comme on l'a dit en introduction du chapitre, on appelle stratégie l'exécution du jeu qui remplit au mieux les objectifs d'un joueur donné. C'est donc la succession optimale d'actions que peut entreprendre ce joueur pour atteindre ses objectifs. Un exemple classique d'objectifs pour une stratégie administrateur est de vouloir patcher le réseau pour un coût et un temps minimum, tout en ayant la garantie qu'à la fin de l'exécution l'ensemble des sommets ne soient plus vulnérables. Les objectifs de stratégie sont donc à la fois un ensemble de contraintes sur l'état du graphe de dépendances tel que : "ne pas être vulnérables" et en même temps un ensemble de critères numériques. S'il existe plusieurs exécutions qui satisfont les contraintes sur le graphe de dépendances alors les critères numériques permettent de les départager.

On extrait l'exécution correspondant aux objectifs de stratégie en model-checkant l'ensemble des exécutions possibles du jeu et en retournant celle qui satisfait les contraintes et qui maximise/minimise le plus les contraintes numériques spécifiées. Formellement, on définit les objectifs de stratégie comme suit :

Définition 6.2 (Objectifs de Stratégie). *Les objectifs de stratégie sont le tuple* S : $(\mathtt{name}, \mathsf{P}, \mathcal{O}, \mathcal{R}, \mathcal{C})$ *où* `name` *est la chaîne qui représente le nom de la stratégie,* P *est le joueur qui cherche la stratégie,* $\mathcal{O}$ *est l'ensemble des objectifs numériques de la stratégie,* $\mathcal{R}$ *est l'ordre strict de priorité sur les objectifs numériques et,* $\mathcal{C}$ *est l'ensemble de contraintes que doit satisfaire l'exécution.*

On a choisi de considérer les stratégies sans mémoire car elles permettent de garantir que l'intrus et l'administrateur n'ont pas de connaissance préalable de la situation ou de leur adversaire. Cette approche est cohérente avec l'idée que lors d'une attaque et particulièrement lors de l'utilisation de 0day, l'intrus et l'administrateur se retrouvent face à une situation nouvelle. Il est aussi extrêmement rare que l'administrateur connaisse ou sache reconnaître les personnes qui attaquent son réseau.

Notons que la recherche d'objectifs de stratégie s'applique aussi dans le cadre des jeux à mémoire, et qu'il n'y a donc pas de perte de généralité à considérer les jeux sans mémoire. En effet, que l'on considère les jeux avec ou sans mémoire cela ne change pas les résultats de décidabilité. L'argument court étant qu'un jeu avec mémoire peut être vu comme une succession de jeux sans mémoire. Or le nombre de jeux itérés engendré par un anticipation game est fini car le nombre d'exécutions possible est lui aussi fini. Par réccurence, il est trivial de montrer que si un jeu sans mémoire est décidable, une succession fini de jeux est décidable.

6.2.1 Objectifs numériques

Les objectifs numériques $\mathcal{O}$ sont assignés sur les résultats ϕ :

Définition 6.3 (Résultats d'exécution). *Les résultats d'exécution sont un ensemble non ordonné d'entiers naturels*
$\phi : \{\texttt{payoff}, \texttt{cost}, \texttt{opayoff}, \texttt{ocost}, \texttt{time}\}$ *ou* `payoff` *est le gain du joueur,* `cost` *est le coût du joueur,* `opayoff` *est le gain de son opposant,* `ocost` *est le coût de son opposant et* `time` *est la durée de l'exécution.*

Le gain du joueur P pour l'exécution ρ est la somme de toutes les récompenses octroyées par les exécutions réussies de ses règles d'octroi. Le coût du joueur P pour l'exécution ρ est la somme de tous les coûts de règles qu'elles aient eu une exécution réussie ou pas, car quelque soit l'issue de l'exécution, le joueur doit investir la même quantité de ressources. Il est pratique de décrire les objectifs numériques dans le langage concis suivant :

$\mathcal{O}$	::=	O	Objectif $\in \phi$
	|	$\mathcal{O} \wedge \mathcal{O}$	
	|	MAX(O)	maximise la valeur de l'objectif O
	|	MIN(O)	minimise la valeur de l'objectif O
	|	$O < x$	$x \in \mathbb{N}$
	|	$O > x$	$x \in \mathbb{N}$

Ainsi dans ce langage, les objectifs numériques de la stratégie de patchage qui vise à minimiser le temps et le coût s'écrivent :

$$\text{MIN}(\text{cost}) \wedge \text{MIN}(\text{time}).$$

On a deux ordres possibles sur les objectifs :

$$\mathcal{R} : \text{cost} > \text{time} \text{ ou } \mathcal{R} : \text{time} > \text{cost}.$$

Time	Player	Action	Rule	Target	Succ	Payoff	Cost
0	I	choose	Comp 0 Day	4	$\perp$	-	-
0	A	choose	Patch	4	$\perp$	-	-
2	I	**execute**	Comp 0 Day	4	$\perp$	21	20000
2	I	choose	Comp 0 Day	5	$\perp$	-	-
3	A	**execute**	Patch	4	$\perp$	21	500
3	A	choose	Patch	5	$\perp$	-	-
4	I	**execute**	Comp 0 Day	5	$\perp$	52	40000
4	I	choose	Comp For	5	6	-	-
6	A	**execute**	Patch	5	$\perp$	52	1000
8	I	`fail`	Comp For	5	6	52	45000

FIGURE 6.7 - *La stratégie où aucun service n'est compromis à la fin*

6.2.2 Contraintes

Les contraintes sur l'état du graphe de dépendances sont utilisées pour ne considérer que les exécutions pertinentes pour une stratégie. Dans notre exemple de stratégie de patch, les exécutions pertinentes sont celles qui assurent que l'ensemble des services vulnérables sont patchés. Il est pertinent d'ajouter une deuxième contrainte à cette stratégie pour tenir compte de l'interaction de l'intrus avec le réseau : aucun service ne doit être compromis.

Cette contrainte a deux interprétations possibles qui mènent à deux résultats très différents. Premièrement, elle peut être interprétée comme voulant dire qu'à la fin de l'exécution aucun service n'est compromis, mais qu'à un moment de l'exécution un service a pu être compromis puis restauré. Deuxièmement, elle peut être interprétée comme voulant dire qu'à aucun moment de l'exécution un service a été compromis.

Ces deux comportements sont illustrés en figure 6.7 pour l'exécution correspondant à la contrainte qu'aucun service n'est compromis (à la fin) et en figure 6.8 pour la stratégie correspondante à la contrainte aucun service n'est **jamais** compromis. Comme on le voit, dans le premier cas un service est compromis (temps 2 sommet 4) puis restauré (temps 3 sommet 4). Dans le deuxième cas, aucun service n'est jamais compromis.

Pour exprimer le second type de contrainte, l'opérateur CTL [BBF+01] $\Box$ est nécessaire. Cet opérateur est utilisé pour exprimer qu'une contrainte doit être vraie pour l'ensemble des états de la stratégie. Réciproquement on peut utiliser $\Diamond$ pour demander qu'une contrainte soit vraie à un moment donné. Par exemple, cela est utile pour les stratégies de l'intrus afin de modéliser une fuite d'information : le service n'a pas besoin de rester compromis, il doit juste l'être à un moment donné. Ainsi les contraintes sont exprimées dans le fragment CTL suivant :

$$\begin{array}{rcll} \varphi & ::= & A & \text{Proposition atomique} \\ & | & \neg\varphi & \\ & | & \varphi \wedge \varphi & \\ & | & \forall A & \\ & | & \exists A & \\ & | & \Box\varphi & \\ & | & \Diamond\varphi & \end{array}$$

Lorsqu'une contrainte est exprimée sans les opérateurs $\Diamond$ et $\Box$, l'on prend la convention que cela indique qu'elle doit être vraie uniquement pour le dernier état de la stratégie. Dans ce fragment exprimé, l'ensemble des contraintes qui assurent qu'aucun sommet n'est jamais compromis (appartient à l'ensemble `Compr`) et qu'à la fin tous les sommets vulnérables sont patchés s'écrit :

$$\Box\neg Compr \wedge \neg Vuln$$

6.2.3 Stratégie dominante

La question naturelle qui se pose est quelle classe de stratégies est intéressante à considérer pour la sécurité réseau. Une approche naïve consiste à considérer les stratégies qui minimisent/maximisent les gains/coûts d'un joueur et qui s'assure par un ensemble de contraintes que les buts du joueur sont atteints. Ainsi, si l'on reprend les objectifs numériques et les contraintes présentées plus haut, on obtient des objectifs de stratégie qui minimisent le coût et le temps et s'assurent par contraintes qu'aucun sommet n'est jamais compromis et qu'à la fin de l'exécution tous les sommets sont non vulnérables. Elle s'écrit :

$$S : (patch, Admin, MIN(Cost) \wedge Min(Time), Cost > Time, \Box\neg Compr \wedge \neg Vuln)$$

Cependant, cette approche naïve mène à trouver une stratégie incorrecte car lorsque les coûts sont minimaux, l'opposant fait des "erreurs" comme on peut le voir sur l'exécution suivante :

Time	Player	Action	Rule	Target	Succ	Payoff	Cost
0	I	choose	Comp Public	4	$\perp$	-	-
0	A	choose	Patch	5	$\perp$	-	-
3	A	**execute**	Patch	5	$\perp$	31	500
3	A	choose	Patch	4	$\perp$	-	-
6	A	**execute**	Patch	4	$\perp$	52	1000
7	I	`fail`	Comp Public	4	$\perp$	0	5000

Ici, l'intrus aurait pu être plus efficace en utilisant un exploit 0day par exemple au début. Ainsi, une classe de stratégies plus intéressante pour la sécurité est celle qui minimise/maximise les coûts /gains et est efficace quoi que l'opposant fasse. Une stratégie de ce type est la meilleure série d'actions contre le pire cas. Cette classe de stratégies est celle qui en plus des objectifs du joueurs vise aussi à maximiser/minimiser les coûts/gains de l'opposant. On les appellent stratégies (strictement) dominantes [Ras07]. Une stratégie dominante est efficace contre le pire cas et les cas moins effectifs. En d'autres termes, cela revient à dire que si un joueur a une stratégie strictement dominante et qu'il l'applique, il a la garantie qu'il remplira ses objectifs quoique fasse son opposant. La notion de stratégie imbattable a été introduite dans [Ham67] par Williams Hamilton.

Du point de vue de la sécurité réseau, cela revient à dire que si l'administrateur a une stratégie strictement dominante pour un ensemble d'objectifs alors peu importe la connaissance que l'intrus a du réseau et des erreurs qu'il fait (ou pas), l'administrateur peut toujours atteindre ses objectifs en suivant cette stratégie. Pour notre exemple d'objectif de patchage, il existe une stratégie strictement dominante pour l'administrateur. Elle est présentée en figure 6.8. Elle lui garantit que quoi que fasse l'intrus, il sera capable de patcher les deux services vulnérables et qu'aucun d'entre eux ne sera jamais compromis.

Les stratégies strictement dominantes n'existent pas toujours. Par exemple, si l'administrateur n'est pas en mesure s'isoler les services publics, alors l'intrus a une stratégie dominante qui utilise des attaques 0day. Cette stratégie est visible en figure 6.9. Cela se traduit par le fait que quelles que soient les actions de l'administrateur, l'intrus peut toujours compromettre des services, tant qu'il utilise des exploits 0day. Dans cette configuration, l'administrateur a une stratégie dominante faible qui est capable de contrer toutes les stratégies de l'intrus qui ne font pas appel à des attaques 0days (Figure 6.10). C'est la meilleure stratégie possible pour l'administrateur dans cette situation. Elle permet de minimiser le nombre de stratégies que son opposant (l'intrus) peut employer pour la vaincre. En d'autres termes, c'est la stratégie qui lui permet de vaincre les intrus ayant une connaissance imparfaite du réseau ou qui font une erreur. Du point de vue de l'analyse de risques c'est donc la parade technique optimale pour minimiser la menace.

Time	Player	Action	Rule	Target	Succ	Payoff	Cost
0	I	choose	Comp 0 Day	4	⊥	-	-
0	A	choose	Firewall	4	⊥	-	-
1	A	**execute**	Firewall	4	⊥	0	10000
1	A	choose	Firewall	5	⊥	-	-
2	I	`fail`	Comp 0 Day	4	⊥	0	20000
2	I	choose	Comp 0 Day	5	⊥	-	-
2	I	choose	Comp 0 Day	5	⊥	-	-
2	A	**execute**	Firewall	5	⊥	0	20000
2	I	choose	Comp 0 Day	5	⊥	-	-
2	A	choose	Patch	4	⊥	-	-
4	I	`fail`	Comp 0 Day	5	⊥	0	40000
5	A	**execute**	Patch	4	⊥	21	20500
5	A	choose	UnFirewall	4	⊥	-	-
6	A	**execute**	UnFirewall	4	⊥	21	20500
6	A	choose	Patch	5	⊥	-	-
9	A	**execute**	Patch	5	⊥	52	21000
9	A	choose	UnFirewall	5	⊥	-	-
10	A	**execute**	UnFirewall	5	⊥	52	21000

FIGURE 6.8 - *Stratégie dominante de l'administrateur*

6.3 Décidabilité et Complexité

Il reste à étudier la décidabilité de la recherche de stratégie et donner la complexité associée à cette recherche. La difficulté majeure de la preuve est que certaines exécutions engendrées par un jeu donné sont infinies. L'argument clé pour résoudre cette difficulté est que même si une exécution est infinie, elle est ultimement périodique. Pour décider si une exécution remplit les objectifs de stratégie demandés, il faut connaître deux choses : les gains, coûts et temps des joueurs pour l'exécution et si l'exécution satisfait les contraintes de stratégie. Pour les résultats numériques de l'exécution, on a le résultat suivant :

Lemme 6.1. *Les résultats numériques d'une exécution infinie peuvent être calculés en examinant un préfixe court.*

Démonstration. Par définition le nombre de règles et de sommets d'un anticipation game est fini. Si bien qu'il n'existe qu'un nombre fini possible d'application des règles. Il en découle qu'une exécution même infinie n'a qu'un nombre fini d'états distincts.

Time	Player	Action	Rule	Target	Succ	Payoff	Cost
0	I	choose	Comp 0 Day	4	$\perp$	-	-
0	A	choose	Patch	5	$\perp$	-	-
2	I	**execute**	Comp 0 Day	4	$\perp$	21	20000
2	I	choose	Comp For	4	6	-	-
3	A	**execute**	Patch	5	$\perp$	31	500
3	A	choose	Patch	4	$\perp$	-	-
6	I	**execute**	Comp For	4	6	1372	25000
6	I	choose	Comp Back	1	4	-	-
6	I	choose	Comp Back	1	4	-	-
6	A	**execute**	Patch	4	$\perp$	52	1000
10	I	`fail`	Comp Back	1	4	1372	30000
10	I	choose	Comp Back	4	6	-	-
14	I	**execute**	Comp Back	4	6	1393	35000
14	I	choose	Comp Back	1	4	-	-
18	I	**execute**	Comp Back	1	4	1394	40000
18	I	choose	Comp Back	2	4	-	-
22	I	**execute**	Comp Back	2	4	1395	45000
22	I	choose	Comp For	2	5	-	-
26	I	**execute**	Comp For	2	5	1426	50000
26	I	choose	Comp Back	3	5	-	-
30	I	**execute**	Comp Back	3	5	1427	55000

FIGURE 6.9 - *Stratégie strictement dominante pour l'intrus lorsque la règle firewall n'existe pas*

Donc, si une exécution est infinie, elle a une boucle. Les fonctions de gains et de coûts sont monotones croissants, donc la valeur du gain et du coût ne peut qu'augmenter ou rester la même à mesure que l'exécution progresse. Le temps, quant à lui, est strictement monotone croissant et donc ne fait qu'augmenter à mesure que l'exécution progresse. Par conséquent, il existe deux cas possibles lorsqu'une exécution est infinie. Cas un, si la valeur est augmentée durant l'exécution de la première itération de la boucle, elle le sera à chaque itération de la boucle et par conséquent à l'infini sa valeur diverge. Deuxième cas, la valeur reste constante durant la première itération de la boucle, par conséquent elle restera constante quel que soit le nombre d'itérations, par conséquent à l'infini sa valeur est égale à sa valeur lors de l'entrée dans la boucle. On en déduit qu'il est possible de calculer les valeurs d'une exécution infinie en examinant un court préfixe de cette exécution qui contient seulement la première itération de la boucle.

Time	Player	Action	Rule	Target	Succ	Payoff	Cost
0	I	choose	Comp Public	5	$\perp$	-	-
0	A	choose	Patch	5	$\perp$	-	-
4	A	**execute**	Patch	5	$\perp$	31	500
4	A	choose	Patch	4	$\perp$	-	-
7	I	`fail`	Comp Public	5	$\perp$	0	5000
7	I	choose	Comp 0 Day	4	$\perp$	-	-
8	A	**execute**	Patch	4	$\perp$	52	1000
10	I	`fail`	Comp 0 Day	4	$\perp$	0	25000

FIGURE 6.10 - *Stratégie faiblement dominante pour l'administrateur lorsque la règle firewall n'existe pas*

Pour la satisfaction des contraintes on a le résultat suivant :

Lemme 6.2. *Pour une exécution infinie, la décidabilité de la satisfaction des contraintes d'objectifs de stratégie peut être réduite à vérifier la satisfaction des contraintes sur un préfixe court.*

Démonstration. Comme on l'explique dans la preuve : 6.3, les exécutions infinies ont une boucle. Il en découle qu'à la fin de la première itération de la boucle, l'ensemble des états distincts de l'exécution aura été énuméré. Il en découle que si les contraintes d'objectifs sont satisfaites par la première itération, elles seront satisfaites pour l'ensemble des itérations. Donc, il est possible de décider si une exécution infinie satisfait l'ensemble des contraintes en ne regardant qu'un court préfixe.

Cela nous amène au théorème central pour la décidabilité :

Théorème 6.1. *Décider si une exécution est une stratégie valable pour un ensemble d'objectifs de stratégie est décidable en ne regardant qu'un nombre fini d'états de l'exécution.*

Démonstration. Dans le cas de l'exécution finie, c'est trivial. Lorsque l'exécution est infinie, c'est possible car vérifier les contraintes et les résultats numériques se font sur un court préfixe d'après les lemmes 6.1 et 6.2.

De plus, on peut prouver que décider si une exécution est un candidat valide pour des objectifs de stratégie se fait en temps polynomial :

Théorème 6.2. *Décider si une exécution satisfait les contraintes des objectifs de stratégie se fait en temps polynomial :* $O(s \times |\varphi|)$ *où* s *est le nombre d'états et* φ *l'ensemble des contraintes à vérifier.*

Démonstration. Par le théorème 6.2, on sait que le nombre d'états à vérifier est fini. De plus, comme les contraintes de stratégie sont exprimées dans un fragment de CTL, le théorème 3.1 de [MS03] est applicable. Ce théorème dit que le model-checking d'une contrainte CTL sur une exécution peut être effectué en temps polynomial si le nombre d'états est fini.

Finalement, on a le théorème de décidabilité général suivant :

Théorème 6.3 (Decidability)**.** *Trouver un ensemble fini de stratégies dans les anticipation gamesest décidable.*

Démonstration. Le nombre de sommets du graphe de dépendances, de règles et d'ensemble d'états est fini. Donc le nombre d'états de jeu possible est fini. Il en découle que le nombre d'exécutions est fini. Par le théorème 6.1, chaque exécution est décidable en examinant un nombre fini d'états. De plus, par hypothèse, le nombre de stratégies à trouver est fini, il en découle que chaque état est évalué un nombre fini de fois. Ainsi, trouver un nombre fini de stratégies pour un anticipation game donné est décidable.

6.3.1 Complexité en terme d'espace mémoire

Une propriété clé du modèle pour l'implémentation est que l'espace mémoire nécessaire pour trouver une stratégie est linéaire.

Théorème 6.4. *Le pire cas en termes d'espace mémoire est :* $WC = Set \times V \times R \times (S + 1)$ *où* Set *est la mémoire finie nécessaire pour stocker les variables d'états,* V *le nombre de sommets du graphe de dépendances,* R *le nombre de règles et* S *est le nombre de stratégies.*

Démonstration. La plus longue exécution où le préfixe fini dans le cas d'une exécution infinie possible est celui où toutes les règles sont exécutées sur tous les sommets du graphe de dépendances deux fois puisqu'une règle peut avoir une exécution réussie ou échouée. Par le théorème 6.1 dans le cas des exécutions infinies, ne regarder que le préfixe est suffisant car rajouter un état au préfixe est équivalent à ajouter le premier état de la deuxième itération de la boucle ce qui est inutile pour décider si une exécution est une stratégie valide. Par conséquent, le pire cas est lorsque la stratégie recherchée et l'exécution courante sont égales à cette plus longue exécution.

6.4 Fichier de jeu *NetQi* correspondant à l'exemple

Le fichier de jeu qui implémente l'exemple présenté tout au long du chapitre est présenté en figure 6.11. Par rapport aux exemples précédemment, l'on note l'apparition des *règles d'octroi* qui utilisent la syntaxe => pour symboliser la double flèche $\Longrightarrow$, comme par exemple, la règle *Compromise 0day* à la ligne 9. On remarque aussi que les règles sont suffixées par le coût. Par exemple, la règle *Compromise Public* (ligne 10) a son coût de 5 000 matérialisé par le suffixe `:5000`. Entre les lignes 33 et 38, la valeur des sommets du graphe de dépendances est spécifiée. Enfin les différents objectifs de stratégie utilisés au cours du chapitre sont spécifiés entre les lignes 41 et 48. La syntaxe des objectifs de sécurité est très semblable à celle de la définition. On notera que `Cost` représente le coût du joueur, `OCost` celui de son adversaire comme utilisé à la ligne 47 pour la recherche de la stratégie : `Patch Dominant strategy` du joueur administrateur `A`. Pour les contraintes CTL, l'opérateur $\Box$ est symbolisé par $\sim$. A l'opposé | est utilisé pour demander l'évaluation uniquement à la dernière étape de l'exécution comme à la ligne 48 qui demande qu'à la fin de l'exécution au moins un service soit compromis. Le + étant utilisé pour indiquer que pour un sommet au moins, la variable d'états `Compr` doit être vraie. Réciproquement − est utilisé pour indiquer qu'une variable d'état est fausse pour au moins un sommet. Enfin $\star$ est utilisé pour indiquer que la variable d'état doit être vraie pour tous les sommets et ! qu'elle doit être fausse pour tous les sommets comme utilisée à la ligne 47.

6.5 Mise en pratique

Nous avons, grâce à *NetQi*, mené une série d'évaluations visant à démontrer que les anticipation games avec des objectifs de stratégie pouvaient être utilisés en pratique, et que les bornes de complexité théoriques se retrouvaient bien en pratique. Les évaluations ont été conduites sur une machine standard, à savoir un Intel 2.9 GHz core 2 sous Linux 2.6.x avec 2 Go de mémoire vive. Chaque test a été effectué trois fois et les mesures reportées ci-dessous représentent la moyenne de ceux-ci. Nous avons effectué deux jeux de tests. Le premier jeu de tests, visant à évaluer si l'approche peut être mise en pratique, confronte *NetQi* à un exemple de gros réseau. Le deuxième jeu de test vise à évaluer l'impact de la recherche de stratégie sur les performances de *NetQi* et à s'assurer que ces performances sont cohérentes avec les bornes de complexités théoriques.

```
1)  nodes=6
2)  <sets>
3)  Vuln:4,5
4)  Public:4,5
5)  Compr:false
6)  NeedPub:4,5
7)  </sets>
8)  <rules>
9)   I:2:Compromise 0 Day:Vuln^Public^!Compr=>Compr:20000
10) I:7:Compromise Public:Vuln^Public^!Compr=>Compr:5000
11) I:4:Compromise Backward:!Compr^>Compr=>Compr:5000
12) I:4:Compromise Forward:Compr^~Compr=>>Compr:5000
13) A:1:Firewall:Public^Vuln->!Public:10000
14) A:1:UnFirewall:!Public^!Vuln^NeedPub->Public
15) A:3:Patch:Vuln=>!Vuln^!Compr:500
16) </rules>
17) <graph>
18) //label
19) 1:client1
20) 2:client3
21) 3:client3
22) 4:www
23) 5:email
24) 6:database
25) //dependencies
26) 1->3
27) 2->3
28) 3->4
29) 3->4
30) 4->6
31) 5->6
32) //costs
33) 1=1
34) 2=1
35) 3=1
36) 4=21
37) 5=31
38) 6=1351
39) </graph>
40) //subsection play example
41) strategy(Intrusion example,I,MAX(Reward)^MIN(OCost),Reward>OCost,|+Compr)
42) //subsection Constraints examples
43) strategy(Patch bogus strategy,A,MIN(Cost),Cost,|!Compr^|!Vuln)
44) strategy(Patch strategy,A,MIN(Cost),Cost,~!Compr^|!Vuln)
45) //subsection : Dominant strategies
46) strategy(Patch simple minimize strategy,A,MIN(Cost),Cost,~!Compr)
47) strategy(Patch Dominant strategy,A,MIN(Cost)^MAX(OCost),OCost>Cost,~!Compr^|!Vuln)
48) strategy(Intrusion dominante,I,MAX(Reward)^MAX(OCost),OCost>Reward,|+Compr)
```

FIGURE 6.11 - *Fichier de jeu de* NetQi *implémentant l'exemple du chapitre 6*

6.5.1 Jeu de test 1 : mise en pratique

Le premier jeu de tests vise à confronter *NetQi* à un exemple de gros réseau. Cet exemple réutilise le jeu de règle présente en section 6.1.2 avec un graphe de dépendances et un état initial composé de $\mathbf{5200}$ sommets, $\mathbf{27000}$ dépendances aléatoires, et $\mathbf{3}$ vulnérabilités aléatoires. Les 5 200 sommets sont répartis comme suit : 200 sommets dits serveurs et 5 000 sommets dits clients. Chaque sommet serveur possède 10 dépendances tirées aléatoirement et chaque sommet client possède cinq dépendances tirées aléatoirement. Un petit script écrit en perl a été utilisé pour générer cet exemple. Afin de garantir que les performances n'étaient pas dues à un cas particulier de l'état initial, nous avons utilisé 10 instances de graphes différentes. Les tests prouvent que quelle que soit la configuration initiale, les résultats sont les mêmes.

Afin de gérer des exemples aussi larges, *NetQi* implémente de nombreuses optimisations qui permettent de réduire le nombre d'exécutions à explorer ou de réduire la profondeur de l'exécution explorée. Ainsi, l'analyse statique du fichier de jeu avant son exécution permet de réduire le nombre de règles à tester à chaque étape de chaque exécution, en supprimant les règles dont les préconditions sont en opposition avec les contraintes d'objectifs. Par exemple, si les objectifs contiennent la contrainte $\neg \mathrm{compr}$, alors toutes les règles ayant pour précondition : compr sont enlevées du jeu de règles. Ceci est une optimisation importante car plus de 60% du temps d'exécution de *NetQi* est passée dans la boucle qui teste quelle règle peut être appliquée à chaque étape du jeu. Ainsi, supprimer ne serait-ce qu'une règle permet de parcourir des milliers d'étapes en plus par seconde.

L'optimisation la plus efficace lors de la recherche de stratégie est le *early cut* qui arrête l'exploration d'une exécution dès que l'une des contraintes $\Box$ est violée. Cette optimisation ne fonctionne que dans l'exemple de la stratégie de défense. Afin de gérer les cas où la recherche de stratégie déclenche une explosion combinatoire, *NetQi* dispose d'une recherche heuristique de stratégie correcte mais non complète : elle retourne une stratégie satisfaisant les contraintes mais qui n'est pas forcément optimale. Cependant, sur de petits exemples de tests, elle a toujours retourné la stratégie optimale. C'est cette heuristique qui a été utilisée pour la recherche d'une stratégie d'attaque. En effet, cette recherche déclenche une explosion combinatoire due à l'entrelacement des règles d'abus de confiance. Les résultats de ce premier jeu de tests sont synthétisés dans le tableau 6.12. Ils démontrent que les anticipation games sont utilisables en pratique.

En effet, même face à un gros réseau possédant un grand nombre de relations de dépendances, *NetQi* est capable de trouver une stratégie qui remplit les objectifs recherchés en quelques secondes.

Nb services	Nb Dépendances	Stratégie	type	Temps
5200	27000	Defense	Exacte	2.4 sec
5200	27000	Intrusion	Approchée	55 sec

FIGURE 6.12 - *Résultats du jeu de tests 1*

6.5.2 Jeu de tests 2 : vérification des bornes de complexité théorique

Afin d'évaluer si les bornes de complexité se retrouvent bien en pratique, nous avons conduit deux types de tests. Le premier type visait à mesurer l'impact sur les performances en terme de vitesse. Le second type de test visait, quand à lui, à mesurer l'impact sur l'espace mémoire nécessaire. Afin d'avoir les mesures les plus précises possibles, l'ensemble des optimisations de *NetQi* a été désactivé. Le fichier de jeu utilisé pour nos tests est celui présenté en exemple 6.11 étendu à 10 sommets clients. Sans aucune optimisation, l'ajout de clients augmente significativement le temps d'exécution en raison de l'entrelacement entre les règles `compromise forward` et `compromise backward`.

Pour l'impact sur les performances en terme de vitesse d'exécution nous avons utilisé, comme temps de référence, l'exécution du fichier de jeu sans recherche de stratégie. Ainsi le temps de base qui est la moyenne de trois exécutions est de $\mathbf{6.8}$ secondes. Nous avons ensuite généré des fichiers de jeu avec un nombre croissant d'objectifs de stratégie dont nous avons mesuré le temps d'analyse en utilisant la commande Unix *time*. Nos mesures vont d'un fichier contenant 0 stratégie à un fichier contenant 100 stratégies. Afin d'être dans le pire cas, c'est à dire que les contraintes de stratégie soient évaluées à chaque étape de chaque exécution, les stratégies ajoutées avaient toutes une contrainte $\square$, à l'instar de la stratégie de défense présentée en exemple. Les résultats de ces tests sont récapitulés sous forme d'une courbe présentée en figure 6.13. Cette courbe quasi linéaire montre que les résultats obtenus en pratique sont cohérents avec le théorème de complexité 6.2.

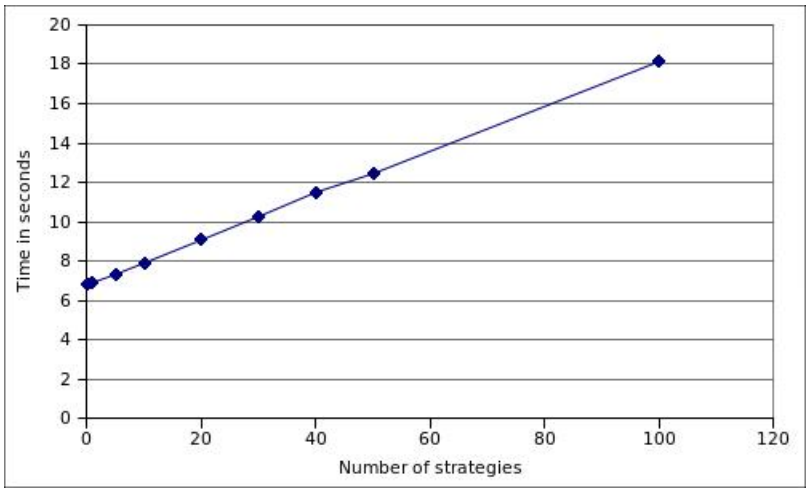

FIGURE 6.13 - *Evaluation de l'impact de la recherche de stratégie en termes de vitesse d'exécution*

Pour le deuxième test, nous avons utilisé le profileur de mémoire **massif** issu de la suite logicielle **valgrind** [NWF06]. Pour vérifier que l'espace mémoire nécessaire augmente de manière linaire en fonction du nombre de stratégies à trouver comme prévu par le théorème 6.4, nous avons réutilisé les fichiers de jeu générés pour le jeu de tests précédent. En raison de la durée conséquente de chaque test et l'impossibilité d'automatiser la lecture du résultat, nous nous sommes restreints à l'intervalle $0 - 30$. En conséquence, après chaque test, nous avons analysé le graphique d'utilisation mémoire généré par massif et reporté le pic de mémoire consommé par *NetQi* sur la courbe 6.14. Comme prédit par la théorie, l'espace mémoire requis augmente bien de manière linaire en fonction du nombre de stratégies.

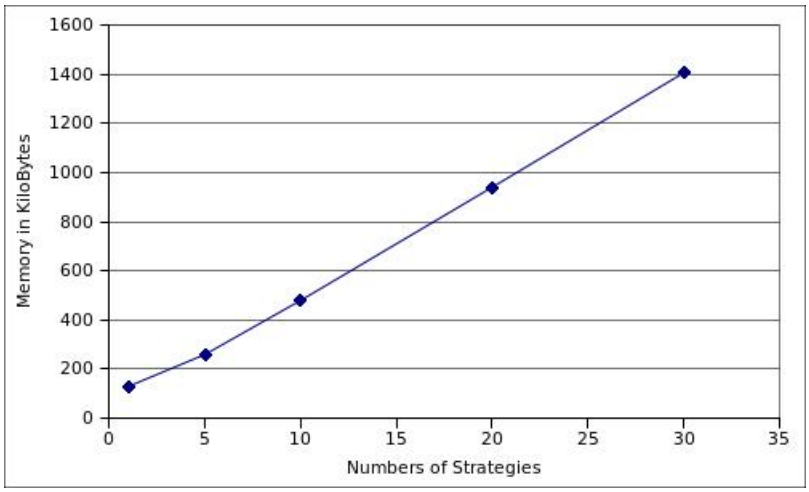

FIGURE 6.14 - *Evaluation de l'impact de la recherche de stratégie en terme d'espace mémoire*

6.6 Points Clés

- Les objectifs de stratégie permettent de trouver l'exécution du jeu qui correspond le mieux à la demande de l'utilisateur.
- L'introduction des coûts et des gains permet d'introduire la dimension financière dans le modèle.
- La recherche de stratégie n'ajoute qu'un facteur linéaire de complexité.
- La recherche de stratégie ne demande qu'un espace mémoire linéaire.
- Les expérimentations montrent que l'on peut analyser des réseaux ayant des milliers de services et dépendances en quelques secondes.

CHAPITRE 7

Emplacement, pénalités et timeline

"The art of war teaches us to rely not on the likelihood of the enemy's not coming, but on our own readiness to receive him ; not on the chance of his not attacking, but rather on the fact that we have made our position unassailable."

Sun Tzu, The art of war VII.11

Lorsque l'on évalue la sécurité d'un réseau, il n'est pas suffisant de considérer la présence ou l'absence de vulnérabilités publiques car à mesure que les réseaux se complexifient et que le nombre de services et protocoles mis en jeu augmente, l'existence de vulnérabilités inconnues ou d'erreurs de configuration devient un risque de plus en plus important. Inévitablement, un réseau étendu offrant de multiples services sera la cible d'attaque 0day.

Il est donc essentiel que les anticipation games soient capables de couvrir ce type de scénario. Pour cela, il est nécessaire d'étendre les anticipation games afin qu'ils puissent modéliser les méthodes destinées à contrer ce type d'attaque. Les méthodes de contre-attaques inconnues font appel à la coopération entre les réseaux pour recouper les informations. En effet, puisque les attaques 0day ne peuvent pas être contrées par les mécanismes de défense classiques comme les détecteurs d'intrusion par misuse, ou l'application de patch, il a fallu inventer de nouveaux mécanismes de défense.

L'idée de base de ces mécanismes est que lorsqu'un site (réseau) est la cible d'une attaque inconnue, il alerte les autres. Il est alors possible de mettre en œuvre des contre-mesures telles qu'interdire l'accès au réseau à l'attaquant potentiel en interdisant son IP grâce au firewall. Cette idée a donné naissance à un mécanisme de défense dédié à cette tâche appelé honey-pot qui vise à mettre en place une machine ou un réseau leurre (honey-net). Ce réseau leurre apparait à l'intrus comme étant réel et possédant des services semblables à ceux utilisés sur les réseaux de production. Cependant, étant des leurres aucun trafic légitime n'est jamais à destination de ceux-ci. Il en découle que lorsqu'une connexion vers l'un des services leurres est enregistrée, elle est forcément malicieuse. Ainsi, l'honey-net envoit une alerte lorsqu'un de ses services est attaqué et ce quelle que soit la nature de l'attaque (connue ou pas). Ces alertes sont utilisées par les sites réels pour mettre en place des contre-mesures. L'utilité de la coopération intersites ne se limite pas à la détection d'attaques inconnues : elle permet aussi de détecter les attaques de grande envergure telles que les vers [CAL+08] et à tracer les botnets [RWJ08].

Les stratégies de coopération ne peuvent pas être modélisées dans la version actuelle des anticipation games car certaines règles et stratégies doivent être restreintes à un sous-ensemble spécifique de services, alors que d'autres règles doivent être globales pour modéliser la coopération inter-services. Par exemple, il est évident que les règles utilisées pour décrire l'application de patchs visant à corriger les vulnérabilités des services ne doivent pas s'appliquer aux services du honey-net. C'est pourquoi, il est nécessaire d'introduire la notion de l'*emplacement* dans le framework.

Modéliser l'existence d'attaques 0day suppose de tenir compte de la durée de vie de l'attaque, aussi appelée fenêtre de vulnérabilité. Modéliser cette notion de fenêtre de vulnérabilité implique de pouvoir modéliser une chronologie d'événements qui décrit à quel moment elle s'ouvre et à quel moment elle se ferme. Cette notion de chronologie d'événements, que nous appelons *timeline d'événements* , est nécessaire pour modéliser de nombreux aspects de la sécurité. Par exemple, le cycle de vulnérabilité [LWS02] implique de modéliser une timeline d'événements qui retrace les différentes étapes de la vie d'une vulnérabilité depuis sa découverte à son éradication, en passant par l'existence d'exploit 0day et la sortie d'un correctif. Pour la coopération multi-site, l'utilisation d'une timeline d'événements est aussi nécessaire pour retracer les différentes étapes de la coopération depuis la détection d'une attaque inconnue jusqu'à la mise en place de contre-mesure.

La modélisation d'attaque par DDOS (Distributed Denial Of Service) et la mise en place d'honey-net ont en commun qu'elles nécessitent de pouvoir exprimer un coût par unité de temps. Dans le cas du honey-net, ce coût correspond au coût de maintenance du honey-net ; dans le cas de l'attaque par DDOS des pertes financières engendrées par l'indisponibilité du service. Nous nommons cette notion de coût par unité de temps à l'instar de la dénomination classique en théorie des jeux : *pénalité* . Intuitivement, une pénalité est un coût ajouté pour chaque unité de temps pour laquelle une condition est satisfaite. Par exemple que le service n'est pas disponible.

L'introduction de pénalités va permettre d'intégrer au framework une relation essentielle qui existe entre le temps et les coûts : la notion de diminution des coûts en fonction du temps. Cette notion apparaît quand une action est exécutée plusieurs fois. Par exemple, lorsque l'attaquant utilise plusieurs fois le même exploit. Elle apparaît aussi lorsqu'un processus continu existe. Par exemple, le coût de monitoring d'un service diminue au cours du temps.

Enfin, cette extension va permettre de modéliser l'exécution simultanée de plusieurs actions par le même joueur et le branchement d'événements. Modéliser l'exécution simultanée d'actions est essentiel pour modéliser que l'ensemble des sites prennent des contremesures simultanément lorsqu'une attaque inconnue est détectée par le honey-pot ; elle est aussi nécessaire pour modéliser le déploiement massif de correctifs. Le branchement événementiel, est utilisé pour modéliser qu'à un certain point, plusieurs possibilités sont offertes à un joueur, comme par exemple, utiliser un patch ou un autre.

La contribution principale de ce chapitre est donc l'extension du framework pour permettre la modélisation de la coopération inter-sites, de la notion de coût lié au temps, et la gestion des timelines d'évènements. Afin d'illustrer le fonctionnement de l'extension, nous allons nous appuyer sur un exemple qui décrit la modélisation d'une stratégie multi-site utilisant un honey-net pour combattre différentes attaques, y compris les attaques 0day.

Le chapitre est organisé de la manière suivante. D'abord nous introduisons l'exemple qui nous servira de fil conducteur. Ensuite nous présentons la notion d'emplacement et nous prouvons que l'ajout des emplacements ne change pas la complexité des anticipation games. Puis nous détaillons la notion de timeline et illustrons son utilisation en modélisant le cycle de vie d'une vulnérabilité. Ensuite nous développons la notion de pénalité et montrons comment elle permet de modéliser les coûts en fonction de leur durée. Enfin nous analysons les stratégies produites par l'exemple pour montrer comment l'efficacité des honey-net à contrer les attaques 0-day se traduit dans le framework. Nous concluons ce chapitre en détaillant comment l'exemple du chapitre est codé dans le langage de *NetQi*, et en évaluant l'efficacité de *NetQi* pour la recherche de stratégies multi-sites.

Ce chapitre a fait l'objet de la publication [Bur08a].

7.1 l'Anticipation Game d'exemple

Le but de l'analyse présentée en exemple sera, à l'instar du chapitre précédent, exprimé sous forme d'objectifs de stratégies. On cherche à répondre à la question suivante : "*Comment contrer une attaque utilisant un exploit 0day*".

Rappelons que les objectifs de stratégies sont composés de deux parties principales : un ensemble de contraintes CTL et un ensemble d'objectifs numériques sur les résultats des exécutions. Les contraintes sont utilisées pour définir quelles conditions l'état du réseau doit satisfaire pour qu'une exécution soit considérée comme une stratégie valable. Les objectifs numériques permettent de sélectionner parmi l'ensemble des exécutions qui satisfont les contraintes celle qui est la meilleure pour le joueur en terme de coût, gain et temps. Ainsi une stratégie typique de défense contre les intrusions minimisera les coûts.

7.1.1 Graphe de dépendances

Le graphe de dépendances utilisé pour l'exemple est représenté en figure 7.1, et l'état initial correspondant en figure 7.2. Rappelons que le graphe de dépendances est le graphe utilisé pour représenter la topologie du réseau et en particulier les dépendances entre les services. Ce graphe est statique et n'évolue pas durant les exécutions du jeu. A l'opposé, l'état du réseau est fait pour évoluer. Cet état, représenté par un ensemble de propositions atomiques, est utilisé pour décrire les conséquences des actions des joueurs sur le réseau.

Le graphe de dépendances est composé de six sommets dits *réels* et d'un sommet dit *virtuel* (sommet 1). Les arcs sont les dépendances qui existent entre les services. Les dépendances concrètes sont représentées par des lignes pleines et les dépendances virtuelles, utilisées pour la timeline d'événements, sont représentées par des lignes en pointillés. Le rôle du nœud virtuel et des arcs entrants est de permettre la modélisation de la timeline d'événements comme détaillée à la section 7.3.

Comme dans les chapitres précédents, les dépendances concrètes sont utilisées pour modéliser qu'un service dépend d'un autre. Dans notre exemple, le sommet *www (5)*, qui est un service web, dépend du sommet *DB (6)* qui est une base de données, pour authentifier les utilisateurs. D'un point de vue de la sécurité, cette dépendance implique qu'il existe une relation de confiance entre le service web et la base de données qui peut être exploitée par un intrus.

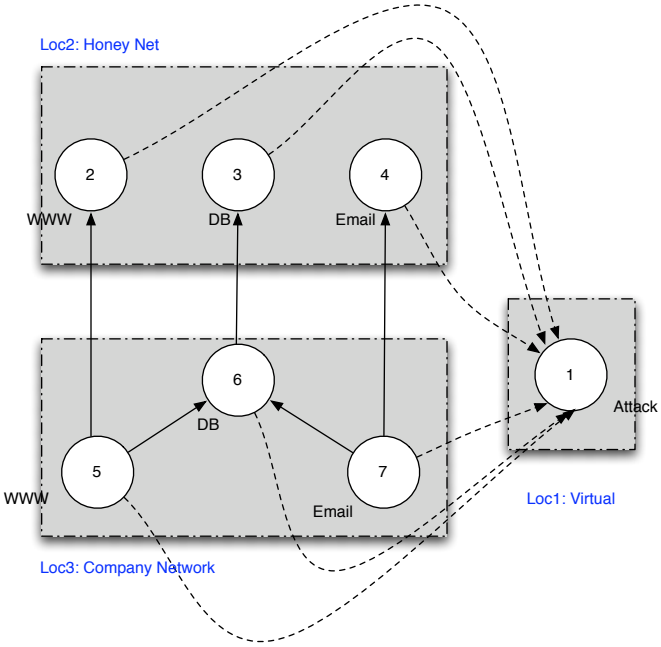

FIGURE 7.1 - *Graphe de dépendances*

Les trois dépendances qui partent des services réseau de la société vers leur homologue du honey-net sont utilisées pour la mise en place de la stratégie de défense. Les faux services du honey-net sont utilisés pour appâter et détecter l'intrus quand celui-ci essaye de les attaquer. L'utilisation du honey-net permet de détecter les attaques inconnues en comparant le trafic à destination du honey-net avec l'ensemble des attaques connues. Ainsi, si le honey-net voit du trafic qui ne correspond pas à la base d'attaques connues, il alertera le service correspondant du réseau réel afin que celui-ci puisse prendre les contre-mesures nécessaires.

L'ensemble des variables mappées pour cet exemple peut être divisé en trois parties. La première partie est constituée des variables `0DayAvail`, `CustomAvail`, `PubAvail` et `PatchAvail` utilisée pour modéliser la timeline d'événements comme détaillée en section 7.3.
La seconde partie est constituée des variables `Detected` et `Monitored` qui sont utilisées pour la coordination de la défense multi-sites comme détaillée en section 7.6.
Enfin, la troisième partie est utilisée pour décrire les conditions initiales proprement dites : la variable `Vuln` est utilisée pour modéliser que le service web et le service d'emails et leurs homologues du honey-net sont vulnérables à une attaque inconnue (0day). La variable, `Compr` est utilisée pour indiquer qu'au départ aucun service n'est compromis. Enfin, la variable `Public` indique qu'au départ tous les services sont publics (non firewallés).

States	1	2	3	4	5	6	7
ρ(0DayAvail)	⊥	⊥	⊥	⊥	⊥	⊥	⊥
ρ(CustomAvail)	⊥	⊥	⊥	⊥	⊥	⊥	⊥
ρ(PubAvail)	⊥	⊥	⊥	⊥	⊥	⊥	⊥
ρ(PatchAvail)	⊥	⊥	⊥	⊥	⊥	⊥	⊥
ρ(Detected)	⊥	⊥	⊥	⊥	⊥	⊥	⊥
ρ(Monitored)	⊥	⊤	⊤	⊤	⊥	⊥	⊥
ρ(Vuln)	⊥	⊥	⊤	⊥	⊥	⊤	⊥
ρ(Compr)	⊥	⊥	⊥	⊥	⊥	⊥	⊥
ρ(Public)	⊤	⊤	⊤	⊤	⊤	⊤	⊤

FIGURE 7.2 - *Etat initial*

7.1.2 Règles

Comme toujours, les actions des joueurs sont décrites par un ensemble de règles temporisées de la forme :

$$\Gamma_{x:y} : \mathbf{Pre}\ F \xrightarrow{\Delta,\ p,\ a,\ c} P$$

où F est l'ensemble des préconditions qui doivent être satisfaites pour que la règle puisse être exécutée. Δ est le nombre d'unités de temps nécessaire pour exécuter la règle, p est le joueur qui utilise la règle, a et le label de la règle (chaîne) et c est le coût de la règle. P est l'ensemble des postconditions, qui décrivent les effets de la règle. Par rapport aux chapitres précédents, l'unique différence dans la syntaxe des règles est l'introduction de $\Gamma_{x:y}$ pour indiquer l'emplacement de la règle. x spécifie l'emplacement du sommet cible et y spécifie l'emplacement de l'éventuel successeur. Comme précédemment, nous utilisons deux types de règles : les *règles d'octroi* utilisant la double flèche $\Longrightarrow$ qui octroient au joueur une récompense en cas d'exécution réussie et les `règles standard` utilisant la simple flèche $\longrightarrow$ qui n'octroient pas de récompense. Le gain associé à la règle est toujours lié à la valeur du sommet cible. On utilise une nouvelle fois la règle suivante pour modéliser un abus de confiance :

$$\Gamma : \mathbf{Pre}\ \Diamond \mathrm{Compr} \wedge \neg \mathrm{Compr} \xRightarrow{2,\mathrm{I},\mathrm{Trust\ abuse},\ 200} \mathrm{Compr}$$

qui dit que, l'intrus (I) peut compromettre un sommet non compromis ($\neg$Compr) en exploitant une relation de confiance, si un de ses successeurs est compromis ($\Diamond$Compr) en deux unités de temps pour un coup de 200$. L'opérateur modal $\Diamond$ est, comme défini au chapitre 3, utilisé pour parler des successeurs.

7.2 Emplacements

Dans sa forme originale (chapitre 3), le framework autorise l'utilisation d'une règle sur n'importe quel sommet du graphe de dépendances tant que l'ensemble des préconditions est rempli pour celui-ci. De manière similaire, les objectifs de stratégies englobent l'ensemble des sommets existants. Or, dans bon nombre de cas, un tel comportement n'est pas souhaitable. En particulier, pouvoir restreindre la portée des règles à un ensemble de sommets donnés est essentiel lorsque l'on modélise des réseaux qui ont des politiques et des méthodes d'action différentes. Par exemple, il est évident que l'action de firewaller un service vulnérable n'a pas de sens pour un réseau honey-net. Afin de pouvoir gérer les différents types de restrictions de portée, nous allons utiliser trois types de règles distinctes :

1. Les règles **locales** utilisées pour les actions spécifiques à un site.
2. Les règles **globales** utilisées pour les actions qui s'appliquent à l'ensemble des sites.
3. Les règles **transversales** utilisées pour modéliser la coopération entre les sites.

Dans notre exemple, ces trois type de règles vont être utilisés : la règle, pour modéliser l'abus de confiance, a besoin d'être restreinte au réseau de la société ; ce sera donc une règle `locale`, les règles pour la coopération intersites seront `transversales` et enfin l'attaquant peut cibler l'ensemble des services, les règles d'attaques sont donc `globales`. A l'instar des règles, les objectifs de stratégies doivent aussi pouvoir être restreints à un sous ensemble de sites. Ainsi dans notre exemple, pour les objectifs de la stratégie de défense, il est évident que la contrainte qu'aucun service ne soit jamais compromis ne s'applique pas aux services du honey-net.

Ainsi afin de pouvoir restreindre la portée des règles, l'on va étendre les anticipation games avec le concept d'emplacement. Intuitivement, un emplacement est un groupement de sommets appartenant au même site ou ayant la même fonction. De manière plus formelle, on définit comme suit un emplacement

Définition 7.1. (Emplacement) *Un emplacement est un ensemble non vide de sommets du graphe de dépendances représenté par un entier* $\mathtt{i}$.

Notons que pour des raisons de clarté et de facilité d'utilisation, *NetQi* autorise l'utilisation de chaînes de caractères pour nommer les emplacements. Cependant, dans sa représentation interne du modèle, il utilise bien un entier. Au moment du parsage du fichier de jeu, *NetQi* construit une table de correspondance entre les chaînes de caractères et leur identifiant numériques. Ainsi lorsque l'on écrit :

```
<graph>
1@mysite
</graph>
```

NetQi fait la translation et associe le nœud 1 à l'identifiant numérique associé à la chaîne `mysite`. L'utilisation des emplacements dans *NetQi* est illustrée au travers de l'exemple de ce chapitre en section 7.7.

7.2.1 Type de Règles

L'exemple utilise le jeu de règles dites *règles opérationnelles* présentées en figure 7.3. On parle de règles opérationnelles car elles sont utilisées pour modéliser les actions d'attaque et de défense. A l'opposé, on parle de *règles chronologiques* pour parler de l'ensemble des règles utilisées pour modéliser la timeline d'événements. Les règles chronologiques sont décrites en section 7.3.

L'ensemble des règles opérationnelles combinent les trois types de règles évoqués précédemment. On définit de manière plus formelle les trois types de règles de la manière suivante :

Définition 7.2. (Règle globale) *Une règle est globale, si et seulement si, aucune restriction d'emplacement n'est spécifiée.*

Définition 7.3. (Règle locale) *Une règle est locale si la même restriction d'emplacement est spécifiée pour la cible de la règle et son successeur.*

Définition 7.4. (Règle transversale) *Une règle est transversale si un emplacement différent est spécifié pour la cible et son successeur.*

7.2.2 Règles globales

Les trois premières règles sont comparables, car elles modélisent toutes les trois la même action : l'intrus (I) utilise un exploit pour compromettre un service public vulnérable. Les préconditions de la règle s'assurent que le service est bien vulnérable (Vuln doit être vrai) et accessible à distance (Public doit être vrai). La règle a pour effet, lorsque son exécution est réussie, que le sommet devient compromis (Compr devient vrai). Puisque par définition, l'intrus ne doit pas pouvoir faire la différence entre les services réels et ceux du honey-net, ces règles sont globales (Γ n'a pas d'indice).

1) $\Gamma_{:}$: **Pre** : $\Diamond 0DayAvail \wedge Vuln \wedge Public \wedge \neg Compr$
$\Longrightarrow$ 3, I, 0 day exploit, 20000
Effect : $Compr$
2) $\Gamma_{:}$: **Pre** : $\Diamond CustomAvail \wedge Vuln \wedge Public \wedge \neg Compr$
$\Longrightarrow$ 4, I, Custom exploit, 2000
Effect : $Compr$
3) $\Gamma_{:}$: **Pre** : $\Diamond PubAvail \wedge Vuln \wedge Public \wedge \neg Compr$
$\Longrightarrow$ 7, I, Public exploit, 200
Effect : $Compr$
4) $\Gamma_{3:3}$: **Pre** : $\neg Compr \wedge \Diamond Compr$
$\Longrightarrow$ 2, I, Trust Abuse, 200
Effect : $Compr$
5) $\Gamma_{1:1}$: **Pre** $Monitored \wedge Compr \wedge \neg Detected$
$\longrightarrow$ 1, A, Attack Detected, 2000
Effect $Detected$
6) $\Gamma_{2:2}$: **Pre** $\neg Vuln \wedge \neg Public$
$\longrightarrow$ 1, A, Unfirewall, 100
Effect $Public$
7) $\Gamma_{3:2}$: **Pre** $\Diamond Detected \wedge Vuln \wedge Public$
$\longrightarrow$ 0, A, Firewall, 100
Effect $\neg Public$
8) $\Gamma_{3:1}$: **Pre** $\Diamond PatchAvail \wedge Vuln$
$\longrightarrow$ 6, A, Patch, 500
Effect $\neg Vuln$

FIGURE 7.3 - *Ensemble de règles opérationnelles utilisées pour décrire les actions des joueurs*

Elles diffèrent à cause de la timeline d'événements, elles sont disponibles pour l'intrus à des moment différents. Ainsi l'exploit 0day est disponible en premier, suivi de l'exploit custom, et enfin l'exploit public. Ainsi, $0DayAvail$ est mis à vrai pour le sommet virtuel par une règle de timeline après 48h. Cette précondition est utilisée pour s'assurer que l'intrus ne pourra l'utiliser plus tôt dans le jeu. Simililairement, l'exploit custom ne peut pas être utilisé avant que la contrainte $CustomAvail$ soit rendue vraie par une règle de timeline car $CustomAvail$ est mise à faux pour l'ensemble des sommets dans l'état initial. Le coût des trois règles diffèrent également pour rendre compte des ressources nécessaires pour la création et l'utilisation des trois types d'exploits. En effet, il est plus coûteux de rechercher et mettre en œuvre un exploit 0day, que d'écrire un exploit custom une fois la vulnérabilité connue, que de simplement réutiliser le code d'un exploit public.

7.2.3 Règles Locales

Les règles 4, 5, et 6 sont des règles locales. Leur indice Γ est de la forme $n : n$, où le premier n est l'emplacement du sommet cible et le second n est celui du successeur du sommet cible. La règle 4 dit que si un service est non compromis ($\neg Compr$) et que l'un de ses successeurs est compromis ($\Diamond Compr$), alors il peut être compromis par l'intrus (I) en 2 heures pour $200. Cette règle doit être restreinte à un contexte local car autrement des actions erronées deviennent possibles : comme visible sur le schéma 7.1, il existe une dépendance entre chaque service de la société et son homologue du honey-net. Lorsque la règle d'abus de confiance n'est pas restreinte à un contexte local, ces relations deviennent exploitables pour des attaques par abus de confiance. Ainsi, un service de honey-net compromis peut être utilisé pour compromettre un service réel. Ceci est clairement une action erronée. C'est pourquoi on la restreint au contexte du réseau de la société où des relations de confiance entre services existent réellement.

La règle 5 est locale au réseau honey-net. Elle dit que si un service est monitoré ($Monitored$), compromis ($Compr$) et qu'une alarme n'a pas encore été déclenchée ($\neg Detected$), alors une alarme est déclenchée. Le temps requis pour exécuter la règle inclut aussi le temps que l'alerte met à être propagée afin de pouvoir modéliser l'exécution simultanée de firewalling de services comme expliqué en section7.3. La variable d'état $Monitored$ est utilisée comme détaillée en section 7.4 pour calculer les coûts du processus continu de surveillance.

Etant donné que la règle de firewall ne concerne que le réseau de la société, il est naturel que la règle 6 utilisée pour enlever le firewall soit locale au réseau de la société. Cette règle dit que si un service est non public ($\neg Public$) et non public ($\neg Vuln$), alors il peut être rendu public ($Public$).

7.2.4 Règles transversales

Les règles 7 et 8 sont transversales. Leur indice Γ est de la forme $n : m$ où n est l'emplacement du sommet cible et m est l'emplacement de l'éventuel sommet successeur. Notons le cas particulier où l'indice Γ est de la forme $n :$ dans ce cas l'emplacement du sommet est restreinte à n et le successeur n'a pas de restriction d'emplacement. Ces règles sont utilisées pour modéliser les interactions entre les sites. Dans notre exemple, il y a deux types d'interactions distinctes.
Première interaction, celle entre le honey-net (loc 2 sur le schéma 7.1) et le réseau de la société (loc 3 sur le schéma 7.1). Cette interaction permet au réseau de la société de se défendre contre les attaques inconnues : les alertes remontées par le honey-net permettent de firewaller les services du réseau de la société de manière préventive. Cette interaction est décrite par la règle 7 qui dit que si une attaque est détectée sur le sommet de l'emplacement distant ($\Diamond Detected$) et que le sommet est public ($Public$) et vulnérable ($Vuln$) alors il peut être firewallé par l'administrateur ($\neg Public$).

La restriction d'emplacement permet de s'assurer que seul le réseau de la société sera affecté par la règle. La restriction sur le successeur permet de s'assurer que le successeur appartient bien au honey-net.

Deuxième interaction, celle entre le réseau de la société et le sommet virtuel. Cette règle est restreinte au réseau de la société car les services du honey-net ne sont pas faits pour être patchés. Le successeur doit être le sommet virtuel car c'est là où la timeline d'événements évolue. L'utilisation de la timeline est nécessaire pour savoir quand le patch est disponible. Cette règle ne peut être que transversale : si elle est globale, elle peut être appliquée aux services du honey-net, et si elle est locale elle ne marchera pas car l'évolution de la timeline d'événements se fait sur le sommet virtuel.

7.2.5 Objectifs de Stratégie avec emplacement

Nous étendons maintenant la définition des objectifs de stratégies pour tenir compte des emplacements

Définition 7.5. *(***Objectifs de Stratégie avec emplacement***) Un ensemble d'objectifs de stratégie est un sextuplet* $S : (\mathtt{name}, \mathsf{P}, \mathcal{O}, \mathcal{R}, \mathcal{C}, \mathcal{L})$ *où* `name` *est le nom de la stratégie,* P *son possesseur,* $\mathcal{O}$ *est l'ensemble des objectifs numériques,* $\mathcal{R}$ *est l'ordre strict sur les objectifs numériques,* $\mathcal{C}$ *est l'ensemble de contraintes CTL sur l'exécution et,* $\mathcal{L}$ *est l'ensemble de contraintes sur les emplacements.*

Pour notre exemple, les objectifs de stratégies utilisés sont les suivants :

$$S : (\text{Defense strategy}, Admin, \text{MIN}(\text{Cost}) \wedge \text{MAX}(\text{OCost}), \text{OCost} > \text{Cost}, \Box\neg\text{Compr}, \neg 2)$$

Notons que ces objectifs sont très proches de ceux utilisés au chapitre précédent. Ils sont utilisés pour trouver l'exécution qui maximise le coût de l'intrus ($\text{MAX}(\text{OCost})$), et qui ensuite minimise le coût de l'administrateur ($\text{MIN}(\text{Cost})$). Les contraintes CTL et de l'emplacement spécifient que seules les exécutions où aucun service n'est compromis ($\Box\neg\text{Compr}$) à l'exception des services du honey-net ($\neg 2$) sont des stratégies possibles. Ajouter la maximisation du coût de l'adversaire permet de trouver la **stratégie dominante (faible)** de l'administrateur.

Pour la complexité et décidabilité des anticipation games étendues avec les emplacements, on a le résultat suivant :

Lemme 7.1. *Le model-checking des objectifs de stratégies sur les anticipation games étendus avec les emplacements est décidable.*

Démonstration. Supposons qu'à chaque emplacement λ_x, on associe un ensemble de somment σ_x du graphe de dépendances. Les emplacements sont spécifiés pour trois composants du jeu : les sommets du graphe de dépendances, les règles, et les objectifs de stratégies. Chaque sommet du graphe de dépendances est associé à un unique emplacement. On peut encoder cette association en une variable d'état $sigma_x/x < n$ initialisée à vrai pour chaque sommet qui appartient à l'emplacement λ_x et à faux autrement.
Pour les règles, la restriction d'emplacement s'applique sur le sommet cible et sur le sommet successeur. On peut restreindre une règle à un emplacement λ_x et rajoutant aux préconditions que la contrainte que le set $sigma_n$ doit être vrai pour les règles locales et transversales. Ajouter cette précondition est possible car la logique modale utilisée pour écrire les règles permet l'utilisation de l'opérateur standard de conjonction. De manière similaire, il est possible de restreindre le successeur à l'emplacement λ_x en utilisant σ_n set. Ainsi, pour restreindre le successeur à l'emplacement λ_x, on rajoute à la règle la précondition $\Diamond\sigma_x$. Restreindre les objectifs de stratégies à l'emplacement λ_x peut être accompli en testant pour chaque sommet que $\rho(\lambda_x)$ est vrai avant de vérifier les contraintes. Si $\rho(\lambda_x)$ renvoit faux, les contraintes ne sont pas testées.
Il est donc suffisant de rajouter λ_n variables d'état pour encoder les emplacements. Ce nombre de variable d'états n est fini car le nombre d'emplacements est fini. En effet, par définition, un emplacement contient au moins un sommet, et le nombre de sommets du graphe de dépendances est lui aussi fini. Par le théorème 3.1, l'on sait que le model-checking d'un anticipation game avec un nombre fini arbitrairement grand de variables d'états est décidable. On en conclut donc que le model-checking des anticipation games étendu avec les emplacements est décidable.

7.3 Utilisation d'une Timeline

Etre capable de modéliser une timeline d'événements est essentiel car de nombreux scénarios de sécurité réseau en ont besoin. Par exemple, le classique cycle de vulnérabilités [LWS02] suit une timeline d'événements : le patch pour une vulnérabilité est développée *uniquement après* que celle-ci ait été reportée ou qu'un exploit l'utilisant soit attrapé et ait subit un processus d'ingénierie inversée (reverse engineering) . De manière similaire, une attaque est détectée par un IDS basé sur la misuse *uniquement après* que la signature de l'attaque a été ajoutée à sa base de données. Ce type de timeline d'événements peut être modélisé dans les anticipation games en utilisant une combinaison de règles, états et dépendances. L'idée clé est d'ajouter un sommet virtuel qui servira de "mémoire" à l'évolution de la timeline d'événements grâce à l'ajout de variables d'états dédiées.

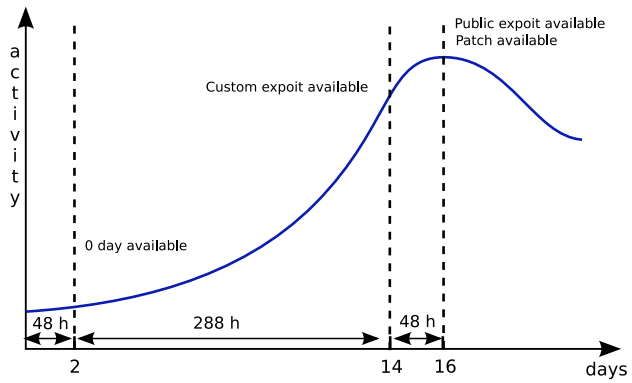

FIGURE 7.4 - *timeline d'événements du cycle de vulnérabilité en temps relatif*

On rajoute aussi un ensemble de dépendances entre les sommets réels et le sommet virtuel pour permettre d'utiliser les informations de la timeline d'événements dans les préconditions et l'effet des règles. Un exemple de jeu de dépendances, utilisé à cet effet, est visible sur la figure 7.1. Les restrictions d'emplacements sont utilisés pour s'assurer que les évolutions de la timeline d'événements s'effectuent bien sur le sommet virtuel. Sans elles, chaque règle d'évolution de la timeline d'événements s'exécuteraient sur l'ensemble des sommets de manière successive et cela mènerait à des comportements erronés.

7.3.1 Illustration

Notre exemple de défense multi-site utilise une timeline d'événements inspirée du cycle standard de vulnérabilité évoqué précédemment. La différence par rapport au cycle standard tient au fait que l'on a converti les durées en temps absolues en durées relatives par rapport à l'évènement précédent comme visible en figure 7.4. Cette conversion est nécessaire pour exprimer ce cycle dans le framework car le temps d'exécution des règles est relatif par rapport au moment du choix, on ne peut donc pas exprimer de temps absolu par rapport au début de l'exécution.

Afin de modéliser cette timeline d'événements, quatre variables d'états et quatre règles sont nécessaires. Les règles sont présentées en figure 7.5. Intuitivement, les variables d'états sont utilisées pour modéliser les points de la timeline qui ont été atteints jusqu'à présent et les règles sont utilisées pour faire progresser la timeline. Une variable d'état distincte est nécessaire pour chaque point de la timeline car ce sont des variables booléennes. En conséquence, l'ensemble des variables d'états utilisées pour la gestion de la timeline est mis à faux. Le temps d'exécution de la règle correspond à l'intervalle de temps entre deux événements consécutifs.

1) $\Gamma_{1:1}$: **Pre** $\neg 0DayAvail$
$\longrightarrow$ 48, I, O day exploit Available, 0
Effect $0dayAvail$
2) $\Gamma_{1:1}$: **Pre** $\neg CustomAvail \wedge 0DayAvail$
$\longrightarrow$ 288, I, Custom exploit available, 0
Effect $CustomAvail$
3) $\Gamma_{1:1}$: **Pre** $\neg PubAvail \wedge CustomAvail$
$\longrightarrow$ 48, I, Public exploit available, 0
Effect Pub
4) $\Gamma_{1:1}$: **Pre** $\neg PatchAvail \wedge CustomAvail$
$\longrightarrow$ 48, I, Patch available, 0
Effect $0dayAvail$

FIGURE 7.5 - *Ensemble de règles utilisées pour modéliser l'évolution de la timeline d'événements*

Par exemple, l'exploit custom est disponible 14 jours après que la vulnérabilité a été découverte (temps absolu), et 12 jours après que l'exploit 0day est disponible (temps relatif). La disponibilité de l'exploit custom est modélisé par la règle 2. Cette règle dit que si un exploit custom n'est pas disponible ($\neg CustomAvail$) et que l'exploit 0day l'est ($0DayAvail$), alors après 288 unités de temps (12 jours), l'attaquant peut avoir accès à l'exploit custom.

7.3.2 Branchement

Utiliser un temps relatif permet aussi de modéliser la notion de branchement. Par exemple, si la timeline présentée au-dessus n'est pas suffisante, car l'on souhaite modéliser les différentes manières dont une vulnérabilité est révélée, alors il est possible d'utiliser plusieurs règles qui ont le même effet mais des précondition, durée et coût différents. Par exemple, pour modéliser qu'une vulnérabilité est révélée suite à une attaque sur un honey-net qui a été reverse-engineerée, il suffit d'ajouter la règle suivante à notre exemple :

$\Gamma_{1:2}$: **Pre** $Detected$
$\longrightarrow$ 288, A, Reverse Engineering , 500
Effect $CustomAvail$

Cette règle transversale dit que si un honey-net détecte une attaque, en 12 jours, l'administrateur est capable de la reverse-engineerer pour un coût de 500$. L'exemple du chapitre n'inclut pas de branchement pour des raisons de simplicité.

7.3.3 Actions simultanées

Un autre type de timeline apparaît lorsque plusieurs actions doivent avoir lieu au même moment. Dans notre exemple de défense multi-site, ceci survient lorsqu'une attaque à l'encontre du honey-net est détectée : l'ensemble des sites utilisant ce honey-net comme mécanisme d'alerte doivent prendre une contre-mesure (firewaller) de manière simultanée. Sans action simultanée, le temps nécessaire à x sites pour prendre leurs contre-mesures est égal a $x \times t$ où t est le temps requis pour firewaller un site. Afin d'avoir un temps constant pour l'exécution des contre-mesures et ce quelque soit le nombre de sites, on utilise donc une variable d'état comme point de validation.

Dans notre exemple, cette variable d'état est $\mathrm{Detected}$. Le temps requis pour firewaller un site est inclus dans le temps de la règle 5 de la figure 7.3. Une fois cette règle exécutée, l'administrateur peut simultanément utiliser autant de règles de firewall qu'elle souhaite. Ceci est achevé en ayant un temps d'exécution pour la règle firewall de 0.

7.4 Les coûts liés au temps

Jusqu'à présent, les coûts sont liés à l'exécution des règles : à chaque fois qu'un joueur exécute une règle, son coût augmente. C'est comme nous l'avons vu, la manière naturelle de modéliser un coût pour chaque action. Cependant, cette approche a une limitation importante : elle ne permet pas de tenir compte des coûts dépendants du temps. De tels coûts existent pour les processus durables qui jouent un rôle essentiel en sécurité réseau.

Deux exemples fameux de processus durables sont les DDOS de service [MS05] et la détection d'intrusion. Plus ces processus durent dans le temps, plus le coût est élevé. Pour modéliser ce type de coût, les anticipation games doivent être étendus avec la notion de pénalité. Intuitivement, une pénalité est un coût qui est ajouté au total du joueur pour chaque unité de temps où une contrainte est satisfaite pour un sommet donné. De manière formelle, une pénalité est définie comme suit :

Définition 7.6. *(**Pénalité**) Une pénalité est u quadruplet* $P : (\mathsf{P}, \mathsf{N}, \mathsf{C}, \mathsf{F})$ *où* P *est le joueur ciblé par la pénalité,* $\mathsf{N} \in \mathbb{N}^*$ *est le sommet du graphe de dépendances où la contrainte doit être satisfaite,* C *est la contrainte qui doit être satisfaite pour activer la pénalité, et* $\mathsf{F}(x)$ *est la fonction* $\mathsf{F}(x) : \mathbb{N}^* \to \mathbb{N}^*$ *qui prend en paramètre l'entier* x *qui est le nombre d'unités de temps depuis que la pénalité est déclenchée et retourne le coût de pénalité associé à la dernière unité de temps. Le coût total engendré par la pénalité est donc la somme de tous les coûts retournés par la fonction de pénalité.*

7.4.1 Exemple de Pénalité

Les pénalités peuvent être utilisées pour modéliser le coût induit par un DDOS . Supposons que le sommet 5 du graphe de dépendances d'exemple (figure 7.1) soit utilisé par la société pour vendre des produits. Chaque heure, le revenu généré par ce service est de 1 000$. Par conséquent, pour chaque unité de temps où le service est indisponible ($\neg$ *Avail*), la société perd 1 000$ de revenus. Ceci est modélisé par la pénalité :

$$P : (\text{Administrator}, 5, \neg\text{Avail}, f(x) \rightarrow 1000)$$

qui dit que pour chaque unité de temps où le sommet 5 (*www*), est non disponible, les coûts de l'administrateur sont augmentés de 1 000. L'utilisation de ce type de pénalité permet, lors de la recherche de stratégie de minimisation des coûts d'une attaque, de tenir compte de la relation qui existe entre la perte de revenu et le temps écoulé. Dans notre exemple, nous utilisons le même type de pénalité pour calculer le coût induit par l'action de firewaller un service public. L'utilisation d'une fonction basée sur le nombre d'unités de temps écoulées depuis le déclenchement de la pénalité permet de modéliser des coûts exponentiels ou un amortissement des coûts.

7.4.2 Amortissement des coûts

L'autre relation entre temps et coût qu'il faut incorporer aux anticipation games, afin d'avoir un modèle réaliste, est la notion de diminution des coûts [DDK96]. Cette relation existe lorsqu'une action est répétée de nombreuses fois ou lorsqu'un processus a une longue durée.

Effectuer la même action encore et encore est une pratique commune en sécurité réseau. Par exemple, l'administrateur applique le même correctif sur de nombreux services similaires et l'intrus réutilise le même exploit pour compromettre de nombreuses instances du même service. A chaque fois, le coût de la première exécution est supérieur aux suivantes.

Dans le cas de l'application de correctif, la première exécution demande de récupérer le correctif, de le tester et faire la procédure d'installation alors que les exécutions suivantes ne demandent que l'application de la procédure. Dans le cas de l'intrus, la mise au point de l'exploit implique de trouver la vulnérabilité puis de développer l'exploit alors que les utilisations subséquentes ne demandent que de lancer ledit exploit.

La notion de réduction des coûts est modélisée dans les anticipation games en utilisant n règles qui ont des coûts différents et une timeline d'événements pour s'assurer que les règles les moins chères ne sont utilisées que lorsque les plus chères ont été exécutées. Cette modélisation est très similaire à la modélisation des exécutions simultanées sauf qu'elle concerne la réduction des coûts et non du temps.

De nombreux systèmes de sécurité tels que les détecteurs d'intrusion et les honey-net demandent une surveillance constante. Ce type de coût peut être divisé en deux parties : les coûts d'implémentation et les coûts de supervision. Les coûts d'implémentation couvrent l'achat du matériel et du logiciel, la mise en place et la période de test, soit l'ensemble des coûts qui n'interviennent qu'une fois. Le coût de supervision couvre la partie récurrente de l'activité de surveillance telle que la mise à jour des bases de signatures, le traitement des alertes, le coût en électricité... On modélise le coût d'implémentation en utilisant une règle et le coût de supervision en utilisant une pénalité. Afin de modéliser que le coût de supervision diminue au cours du temps avec un coût minimum, l'on utilise une pénalité de la forme suivante :

$$P : (\text{Administrator}, 5, \text{Monitored}, f(x) \rightarrow \mathit{int}(1000/x) + y)$$

où x est le nombre d'unités de temps écoulées, $y \in \mathbb{N}$ est le coût minimum et $\mathit{int}(x)$ est la fonction classique qui retourne la partie entière arrondie d'un nombre flottant.

Dans l'exemple, on utilise deux types de pénalités. Le premier type est utilisé pour modéliser les coûts induits par le monitoring des services du honey-net. On suppose que monitorer un service honey-net coute 10$ par heure. En conséquence, on ajoute une pénalité pour chaque service surveillé soit les trois pénalités suivantes :

$$P : (\text{Administrator}, 2, \text{Monitored}, f(x) \rightarrow 10)$$
$$P : (\text{Administrator}, 3, \text{Monitored}, f(x) \rightarrow 10)$$
$$P : (\text{Administrator}, 4, \text{Monitored}, f(x) \rightarrow 10)$$

Le deuxième type de pénalité est utilisé pour rajouter un coût pour chaque unité de temps où un service public n'est pas accessible ($\neg public$) afin de modéliser la perte de revenu. Par souci de simplicité dans l'exemple, on considère que cette perte est de 1 000$ par unité de temps quel que soit le service. En conséquence, on ajoute les trois pénalités suivantes au modèle :

$$P : (\text{Administrator}, 5, \neg\text{Public}, f(x) \rightarrow 1000)$$
$$P : (\text{Administrator}, 6, \neg\text{Public}, f(x) \rightarrow 1000)$$
$$P : (\text{Administrator}, 7, \neg\text{Public}, f(x) \rightarrow 1000)$$

7.5 Décidabilité et complexité de l'extension

Si l'ajout des pénalités permet de modéliser une nouvelle classe de coûts et de mettre en relation les notions de temps et de coût, du point de vue de la décidabilité, étendre le modèle ne change rien.

Lemme 7.2. *Le model-checking d'objectifs de stratégies sur un anticipation game étendu par les pénalités est décidable.*

Démonstration. Une pénalité peut être appliquée quand une variable d'état est vraie ou fausse pour un sommet donné. Si deux pénalités pour la même contrainte et le même sommet existent, elles peuvent être réduites à une pénalité unique dont la fonction de coût est la combinaison des deux. Il y a donc au plus $2 \times n \times m$ pénalités où n est le nombre de sommets du graphe de dépendances et m le nombre de variables d'états. Puisque par définition, les nombre de sommets du graphe de dépendances et de variables d'états sont finis, il y a un nombre fini de pénalités. Il est donc possible de calculer l'ensemble des coûts induits par les pénalités pour un état donné. Le nombre d'états distincts de n'importe quelle exécution du jeu est fini puisque les nombre de règles, et de sommets du graphe de dépendances sont finis. Il s'ensuit que si une exécution est infinie, elle a une boucle. Le coût ajouté par les pénalités est strictement positif, donc, à chaque fois que la pénalité est appliquée, le coût du joueur augmente. Dans le cas d'une exécution infinie, il y a donc deux cas. Premier cas, si le coût du joueur est augmenté par la pénalité durant la première itération de la boucle, alors il le sera à chaque tour de boucle et donc le coût du joueur diverge à l'infini. Deuxième cas, la pénalité n'accroît pas le coût durant la première itération de la boucle, dans ce cas le coût du joueur ne sera jamais augmenté par la pénalité quel que soit le nombre d'itérations de la boucle. On en déduit qu'il est possible de calculer le coût de l'ensemble des pénalités pour n'importe quelle exécution en regardant au plus un court préfixe d'exécution.

Du lemme 7.1 et du lemma 7.2, il suit que :

Théorème 7.1. *Le model-checking d'objectifs de stratégie sur un anticipation game, étendu par les emplacements et les pénalités, est décidable.*

D'un point de vue pratique, le résultat essentiel est que l'extension du modèle ne change pas la borne de complexité :

Théorème 7.2. *Le model-checking d'objectifs de stratégie sur un anticipation game, étendu par les emplacements et les pénalités, est EXPTIME-Complete*

Démonstration. L'ajout des emplacements ajoute au plus deux tests à chaque choix et exécution d'une règle. Donc tester les emplacements n'ajoute qu'un facteur constant à l'évaluation des règles. Trouver la stratégie correspondante à des objectifs ne demande que d'inspecter un nombre fini d'états par le théorème 6.1. Il en découle que restreindre les stratégies à un ensemble d'emplacements demande d'effectuer n tests pour chaque état. n étant le nombre de sommets du graphe de dépendances. Donc, ajouter les emplacements aux objectifs de stratégie, ajoute $n \times s$ tests ou n est le nombre de sommets du graphe de dépendances et s le nombre d'états parcourus durant le model checking de l'anticipation game. Donc, restreindre les objectifs de stratégies à un ensemble d'emplacements ne rajoute qu'un facteur de complexité linéaire. Il en découle que la recherche de stratégie sur les anticipation games étendue par les emplacements reste EXPTIME-complete.
Le nombre de pénalités est fini. En conséquence, les pénalités ajoutent au plus $x \times s$ tests ou x est le nombre de pénalités et s le nombre d'états. Il en découle que les pénalités n'ajoutent qu'un facteur de complexité linéaire et donc la recherche de stratégies sur les anticipation games étendue par les pénalités reste EXPTIME-complete.
Comme les pénalités et les emplacements n'ajoutent chacune qu'un facteur de complexité linéaire, on en conclut que l'ajout des deux n'ajoutent qu'un facteur linéaire. Il en découle que la recherche de stratégie sur les anticipation games étendue par les emplacements et les pénalités reste EXPTIME-complete.

7.6 Stratégies de défense multi-sites

Nous allons voir maintenant comment l'extension du modèle permet de trouver des stratégies multi-site en utilisant le graphe de dépendances, les variables d'états, et les règles présentées ci-dessus.

Pour mettre en lumière l'apport du honey-net on va considérer deux cas. Premier cas, le réseau de la société n'utilise pas le honey-net comme mécanisme d'alerte et par conséquent, on enlève le honey-net et ses règles du jeu. Deuxième cas, le honey-net est utilisé comme mécanisme d'alerte. C'est la configuration exacte que l'on a décrite dans ce chapitre. Dans les deux cas, les objectifs de stratégies de l'administrateur sont les mêmes :

$$S : (\text{Defense strategy}, Admin, \text{MIN}(\text{Cost}) \wedge \text{MAX}(\text{OCost}), \text{OCost} > \text{Cost}, \Box\neg\text{Compr}, \neg 2)$$

Ces objectifs visant à trouver la stratégie dominante ont été détaillés en section 7.2. On va utiliser les objectifs de stratégies suivants pour l'intrus :

$$S : (\text{Attack strategy}, \text{Intruder}, \text{MAX}(\text{Reward}) \wedge \text{MAX}(\text{OCost}), \text{Reward} > \text{OCost}, \text{Compr})$$

Ces objectifs sont utilisés pour trouver la stratégie dominante qui compromet le plus de services possible.

Time	Player	Action	Rule	Target	Succ	Reward	Cost
0	I	sel	0day avail	2	$\perp$	-	-
48	I	**exec**	0day avail	2	$\perp$	0	0
48	I	sel	Custom avail	2	$\perp$	-	-
336	I	**exec**	Custom avail	2	$\perp$	0	0
337	I	sel	Public avail	2	$\perp$	-	-
337	A	sel	Patch avail	2	$\perp$	-	-
385	I	**exec**	Public avail	2	$\perp$	0	0
385	I	sel	Compr public	7	2	-	-
385	A	**exec**	Patch avail	2	$\perp$	0	2700
385	A	sel	Patch	7	2	-	-
391	A	**exec**	Patch	7	2	1	3500
392	I	`fail`	Compr public	7	2	0	200

FIGURE 7.6 - *Stratégie de défense sans honey-net et sans action préventive*

7.6.1 Défense mono-site

Lorsque le honey-net n'est pas présent, la seule stratégie d'attaque pouvant être contrée est celle utilisant un exploit public. Comme visible sur la figure 7.6, cette stratégie implique de patcher le service vulnérable dès que le patch est disponible.

7.6.2 Defense mono-site avec action préventive

La stratégie de défense mono-site peut être améliorée en utilisant la règle suivante :

$$\begin{aligned}\Gamma_{3:1} : \quad & \mathbf{Pre}\ \Diamond \mathit{CustomAvail} \wedge \mathit{Vuln} \wedge \mathit{Public} \\ & \longrightarrow 0,\ \mathrm{A},\ \mathrm{Firewall},\ 100 \\ & \mathbf{Effect}\ \mathit{Public}\end{aligned}$$

Cette règle modélise que l'administrateur peut prendre des actions préventives en firewallant les services vulnérables dès que la vulnérabilité est révélée. Avec cette règle, la stratégie de l'administrateur s'améliore puisque il devient capable de contrer les exploits customs comme illustrés en figure 7.7

.

L'intrus possède toujours une stratégie strictement dominante qui implique d'utiliser un exploit 0day. Firewaller un service dès qu'une vulnérabilité est révélée n'est pas dans la plupart des cas une politique de sécurité effective car elle empêche l'accès aux utilisateurs légitimes.

Time	Player	Action	Rule	Target	Succ	Reward	Cost
0	I	sel	0day avail	2	$\perp$	-	-
48	I	**exec**	0day avail	2	$\perp$	0	0
48	I	sel	Custom avail	2	$\perp$	-	-
336	I	**exec**	Custom avail	2	$\perp$	0	0
337	I	sel	Public avail	2	$\perp$	-	-
337	A	sel	Patch avail	2	$\perp$	-	-
385	I	**exec**	Public avail	2	$\perp$	0	0
385	I	sel	Compr custom	7	2	-	-
e 385	A	**exec**	Patch avail	2	$\perp$	0	3600
385	A	sel	Firewall	7	2	-	-
388	A	**exec**	Firewall	7	2	0	5600
388	A	sel	Patch	7	2	-	-
389	I	`fail`	Compr custom	7	2	0	2000
394	A	**exec**	Patch	7	2	1	6100
394	A	sel	UnFirewall	7	$\perp$	-	-
395	A	**exec**	UnFirewall	7	$\perp$	1	6403

FIGURE 7.7 - *Stratégie de défense mono-site avec action préventive*

7.6.3 Stratégie de Défense Utilisant le Honey-net

Lorsque le honey-net est utilisé, la stratégie de défense permet de contrer les attaques 0day tant que celles-ci ciblent en premier le honey-net comme présenté en figure 7.8. Cependant, même lorsque le honey-net est utilisé, l'intrus possède une stratégie dominante qui consiste à attaquer les services réels en premier comme détaillé en figure 7.9.

Ce résultat est cohérent avec les honey-nets réels dont le but est de mitiger les attaques en piégeant les menaces inconnues mais sans garantie de pouvoir toutes les attraper, puisqu'au final il n'existe aucun moyen de forcer l'intrus à viser une cible plutôt qu'une autre.

7.6.4 Réduire le risque des attaques 0day

Comme on l'a vu, utiliser un honey-net ne protège que partiellement des attaques 0day. Afin d'améliorer cette protection, il est possible d'étendre le modèle à n sites. En effet, si l'on considère que les attaques 0day ne ciblent que des sites très spécifiques en raison de leur coût et que l'attaquant n'est pas capable de faire la distinction entre un service réel et un service de honey-net, alors on peut calculer la probabilité de contre-mesure introduite par le honey-net. Si l'on a un service honey-net, alors l'intrus a 50% de chance de cibler en premier le service réel.

Time	Player	Action	Rule	Target	Succ	Reward	Cost
0	I	sel	0day avail	2	$\perp$	-	-
48	I	**exec**	0day avail	2	$\perp$	0	0
48	I	sel	Compr 0 day	4	2	-	-
51	I	**exec**	Compr 0 day	4	2	1	20000
52	I	sel	Compr 0 day	7	2	-	-
52	A	sel	Attack catched	4	$\perp$	-	-
52	A	**exec**	Attack catched	4	$\perp$	0	2000
52	A	sel	Firewall	7	4	-	-
52	A	**exec**	Firewall	7	4	0	4800
54	I	`fail`	Compr 0 day	7	2	1	40000
54	I	sel	Custom avail	2	$\perp$	-	-
342	I	**exec**	Custom avail	2	$\perp$	1	40000
343	I	sel	Public avail	2	$\perp$	-	-
343	A	sel	Patch avail	2	$\perp$	-	-
390	I	**exec**	Public avail	2	$\perp$	1	40000
391	A	**exec**	Patch avail	2	$\perp$	0	4000
391	A	sel	Patch	7	2	-	-
397	A	**exec**	Patch	7	2	1	4500
397	A	sel	UnFirewall	7	$\perp$	-	-
398	A	**exec**	UnFirewall	7	$\perp$	1	4803

FIGURE 7.8 - *Stratégie de défense avec un honey-net*

Si l'on ajoute un second service, l'intrus a 33% de cibler en premier le service réel et ainsi de suite. Ajouter des services honey-net contribue à la réduction de ce que l'on appelle *la menace absolue*.

Une autre approche est de généraliser la coopération à un ensemble de sites pour réduire ce que l'on nomme la *menace relative*. La réduction de la menace relative ne se concentre pas sur la réduction de la menace pesant sur un service en particulier, mais la réduction de la menace qui pèse en moyenne sur chaque service. Si l'on ajoute une coopération entre deux sites qui ont chacun un service réel et un service honey-net, la probabilité de la menace absolue reste inchangée (50%). Cependant, d'un point de vue de la menace relative, le risque d'attaque sur chaque service chute à 25%. Cette notion de menace relative est utilisée pour améliorer la résilience aux attaques. Cette notion s'applique aux sites ayant de multiples miroirs et au réseau P2P où un nœud peut alerter les autres pour contrer la propagation de l'attaque.

Time	Player	Action	Rule	Target	Succ	Reward	Cost
0	I	sel	0day avail	2	$\perp$	-	-
48	I	**exec**	0day avail	2	$\perp$	0	0
48	I	sel	Custom avail	2	$\perp$	-	-
336	I	**exec**	Custom avail	2	$\perp$	0	0
337	I	sel	Public avail	2	$\perp$	-	-
337	A	sel	Patch avail	2	$\perp$	-	-
385	I	**exec**	Public avail	2	$\perp$	0	0
385	I	sel	Compr 0 day	4	2	-	-
385	A	**exec**	Patch avail	2	$\perp$	0	5700
385	I	sel	Compr custom	7	2	-	-
385	A	sel	Patch	7	2	-	-
389	I	**exec**	Compr custom	7	2	1	2000
389	I	sel	Trust abuse	8	7	-	-
391	I	**exec**	Trust abuse	8	7	2	2200
391	I	sel	Trust abuse	6	7	-	-
391	A	**exec**	Patch	7	2	1	10400
393	I	**exec**	Trust abuse	6	7	3	2400
393	I	sel	Compr public	4	2	-	-
400	I	**exec**	Compr public	4	2	4	2600
401	A	sel	Attack catched	4	$\perp$	-	-
401	A	**exec**	Attack catched	4	$\perp$	1	14200

FIGURE 7.9 - *Stratégie d'attaque strictement dominante*

7.7 Evaluation de l'extension grâce à *NetQi*

Afin d'évaluer l'efficacité des anticipation games à être utilisés pour analyser des scénarios complexes requérant la coopération de multiples sites, l'extension présentée dans ce chapitre a été codée dans la styntaxe de *NetQi*.

7.7.1 Codage

En guise de benchmark, l'on a modélisé l'exemple présenté ci-dessus. Le fichier de jeu utilisé par *NetQi* est présenté ci-dessous. Par rapport aux exemples présentés dans les chapitres précédents, un certain nombre d'éléments ont été rajoutés pour gérer les emplacements et les pénalités.

Ainsi ligne 20, l'emplacement de la règle est ajouté en suffixant la règle par la chaîne : `@virtual`. Le symbole `@` est utilisé pour indiquer que l'on souhaite préciser une emplacement pour la règle et le nom `virtual` est le nom de l'emplacement cible. Comme indiqué précédemment, *NetQi* fait une traduction entre les chaînes de caractères utilisés pour les emplacements et un indice numérique de manière transparente. Ligne 34, la règle utilise l'emplacement `@company :company`. Le premier `company` est utilisé pour restreindre le sommet cible à l'emplacement *company* et le deuxième `company` est utilisé pour restreindre le successeur à l'emplacement *company*. Ligne 49, le nœud 1 est associé à l'emplacement `virtual`. On utilise comme pour les règles, le symbole `@` pour spécifier l'emplacement des sommets du graphe de dépendances.

La ligne 75 contient le tag ouvrant pour la section des pénalités `<penality>` et la ligne 82 le tag fermant. Ligne 76, on indique qu'une pénalité doit être ajoutée pour l'administrateur (`A`) si pour le sommet `2` la variable d'états `Monitored` est vraie. Cette pénalité rajoute `ADD` un coût de `10` par unité de temps. Notons que le mot-clé `ADD` correspond à la fonction d'addition simple. Pour des raisons de simplicité, les différentes lois de pénalités sont représentées par des mots-clé dans le fichier de jeu. Leur fonctionnement explicite est codé dans le fichier *penalty.c*. Enfin ligne 83, on remarque, à la fin des objectifs de stratégie, que la portée de ceux-ci est limitée à l'emplacement *company*.

```
1)    ### General ###
2)   nodes=10

3)  <sets>
4)   //sets used for timeline
5)   0DayAvail:false
6)   CustomAvail:false
7)   PubAvail:false
8)   PatchAvail:false
9)   Detected:false
10)  //set used to specify which host are monitored : the honeynet
11)  Monitored:2,3,4
12)  //sets used used for network state
13)  Vuln:3,6
14)  Compr:false
15)  Public:true
16)  </sets>

17) <rules>
18) ###vulnerability cycle (not granting rules)###

19 ) //Exploit availability cycle

20) I:48:0day available:!0DayAvail->0DayAvail@virtual
21) I:288:Custom exploit available:!CustomAvail^0DayAvail->CustomAvail@virtual
22) I:48:Public exploit available:!PubAvail^CustomAvail->PubAvail@virtual

23) //patch availability cycle (assume that patch and public exploit are
24) // available at the same time if not reduce one or the other time)
25) A:48:Patch available:!PatchAvail^CustomAvail->PatchAvail@virtual


26) ###attack rules (Incident granting rule)####

27) //0day attack
28) I:3:Compromise 0 day:>0DayAvail^Vuln^Public^!Compr=>Compr:20000
29) //Custom attack
30) I:4:Compromise custom:>CustomAvail^Vuln^Public^!Compr=>Compr:2000
31) //Public attack
32) I:7:Compromise public:>PubAvail^Vuln^Public^!Compr=>Compr:200

33) //propagation
34) I:2:Trust abuse:!Compr^>Compr=>Compr:200@company:company
35) ###detection rules###

36) //catch attack on honeynet
37) A:1:Attack catched:Monitored^Compr^!Detected->Detected:2000@honeynet
```

```
38) //defense rules

39) //firewall triggered by honeynet attack
40) A:0:Firewall:>Detected^Vuln^Public->!Public:2000@company:honeynet
41) //firewall triggered by vulnerability disclosure
42) A:3:Firewall:>CustomAvail^Vuln^Public->!Public:2000@company:virtual
43) A:1:UnFirewall:!Vuln^!Public->Public@company
44) //restricted to prod  network (or it will patch the honeynet which is not a good idea)
45) A:6:Patch:>PatchAvail^Vuln=>!Vuln:500@company:virtual
46) </rules>

47) <graph>

48) //node emplacements
49) 1@virtual
50) 2@honeynet
51) 3@honeynet
52) 4@honeynet
53) 5@company
54) 6@company
55) 7@company

56) //dependency to attack to use timeline event
57) 2->1
58) 3->1
59) 4->1
60) 5->1
61) 6->1
62) 7->1

63) //dependency between real service and honey service
64) 5->2
65) 6->3
66) 7->4

67) //dependency between real services
68) 5->6
69) 7->6


70) //node label
71) 5:www
72) 6:database
73) 7:email
74) </graph>

75) <penality>
76) A:2:Monitored:10:ADD
77) A:3:Monitored:10:ADD
78) A:4:Monitored:10:ADD
79) A:5:!Public:1000:ADD
80) A:6:!Public:1000:ADD
81) A:7:!Public:1000:ADD
82) </penality>

83) strategy(Defense,A,MAX(OCost)^MIN(Cost)^MIN(Time),OCost>Cost>Time,~!Compr, company)
84) strategy(Intrusion example,I,MAX(Reward)^MAX(OCost),Reward>OCost,|+Compr)
```

7.7.2 Evaluation

Afin d'évaluer l'efficacité de l'approche à l'exemple, on a rajouté, grâce à un script perl, des sites supplémentaires qui dépendent eux aussi du honey-net. On a fait varier le nombre de services par site, le nombre de sites, et le nombre de vulnérabilités par site. Comme dans les évaluations précédentes, les tests ont été réalisés sur un linux core 2. Le résultat de ce test est reporté en figure 7.10. Comme on le voit sur ces résultats, il est possible de trouver une stratégie optimale pour 50 services répartis en 4 sites et un honey-net. En dépit des optimisations implémentées dans *NetQi*, quand d'autres services sont ajoutés, le temps d'exécution explose, comme prédit par la complexité théorique du modèle. C'est pourquoi, pour les réseaux larges, nous avons développé une heuristique capable de trouver une stratégie approchée.

Strategy	Service	Network	Timeissec
Exact	30	2	0.03
Exact	40	3	0.1
Exact	50	4	1020
Appro	2000	1	0.48
Appro	3000	2	0.51
Appro	4000	3	0.65
Appro	5000	4	0.82
Appro	10000	3	2.26

FIGURE 7.10 - *Résultat du test*

Cette heuristique utilise un ordonnancement dynamique des règles afin d'effectuer un parcours du jeu mi en largeur, mi en profondeur. Cette heuristique est correcte car elle retourne une stratégie qui satisfait les contraintes CTL et d'emplacement, mais il n'y a aucune garantie qu'il n'existe pas une stratégie qui fasse mieux en termes d'objectifs numériques. Cependant, sur des exemples courts, il apparaît que l'heuristique trouve bien la stratégie optimale. Cette évaluation prouve qu'analyser les mécanismes de coopération intersites pour contrer les attaques réseau en utilisant les anticipation games est une approche viable.

7.8 Points Clés

- Les emplacements permettent de restreindre la portée des règles et des contraintes de stratégies.
- L'utilisation d'une timeline permet d'introduire une relation de causalité entre les événements.
- Les pénalités permettent de lier les dimensions financière et temporelle des attaques.
- L'ajout des localités et des extensions ne change pas la complexité du modèle.
- L'exemple du chapitre montre que cette extension permet de modéliser la coopération entre les sites, les honey-nets et le cycle de vulnérabilités.
- L'évaluation des performances de *NetQi* montre qu'il est possible de trouver des stratégies multi-sites pour des réseaux ayant des milliers de services.

CHAPITRE 8

Conclusion

"Hence the saying : If you know the enemy and know yourself, your victory will not stand in doubt ; if you know Heaven and know Earth, you may make your victory complete."

Sun Tzu, The art of war VIII.31

L'application de la théorie des jeux pour l'analyse de risques pour les réseaux se heurte à deux principales difficultés : la modélisation des réseaux et de leurs propriétés, et la mise en pratique de l'analyse. Nous avons apporté une contribution pour chacun de ces problèmes mais des améliorations et prolongements sont possibles

Modélisation. Nous avons proposé un modèle très expressif pour modéliser les attaques et pannes réseaux. En particulier, ce modèle est le premier de la quatrième génération à permettre d'exprimer l'interaction des utilisateurs, la notion de temps et la relation qui existe entre le temps et les coûts. Cette expressivité a un coût : la vérification de formules TATL est EXPTIME-complet. Cependant, au vu des résultats obtenus lors de la mise en pratique, il apparaît que ce modèle offre un bon compromis entre expressivité et complexité.

Au chapitre 3, nous avons introduit l'anticipation game qui est notre modèle de jeu temporisé pour les réseaux. Nous avons montré que la vérification de propriétés TATL sur ce modèle est EXPTIME-complet. Au chapitre 6, nous avons développé la notion d'objectifs de stratégies qui permet, en combinant des contraintes CTL et des objectifs numériques, de trouver parmi l'ensemble des exécutions du jeu, celle qui est la plus intéressante pour l'utilisateur. Nous avons montré qu'évaluer si une exécution est une stratégie valide peut être fait en un temps linéaire et que la recherche de stratégies ne réclame qu'un espace linéaire en mémoire. Enfin, au chapitre 7, nous avons étendu les anticipation games avec la notion de localité et de pénalité afin de pouvoir modéliser les coûts en fonction du temps et la coopération multi-site. Nous avons montré que cet ajout ne modifie pas la complexité du modèle.
Les trois prolongements naturels de ce modèle sont, d'une part, de s'intéresser à la notion de coût dynamique, d'autre part, la recherche de point névralgique et enfin modéliser les différentes classes d'attaquant.

Introduire une notion de coût dynamique permettrait de tenir compte que, en fonction des services déjà compromis, la valeur et le coût pour compromettre les services restants sont différents. L'exemple canonique est qu'une fois le service de base de données compromis, la valeur associée à l'application web qui en dépend chute drastiquement puisque l'intrus a déjà accès aux informations contenues dans cette application grâce au contenu de la base de données. De manière similaire, il est probable que le coût pour compromette l'application web soit amoindri car l'intrus possède déjà les logins et passwords pour l'utiliser.
La recherche de points névralgiques est une direction intéressante pour renforcer la sécurité du réseau car cela permet de mieux cerner l'imprévisible. On cherche à isoler les vulnérabilités et pannes potentielles qui pourraient causer le plus de dommages afin de déterminer les services clé du réseau qui demandent une surveillance accrue.
Dans le modèle actuel, l'attaquant ne fait pas d'erreur, ce qui est une hypothèse conservatrice puisqu'elle correspond au pire cas. Cependant, en réalité, l'efficacité d'un intrus dépend de sa connaissance du réseau et de son niveau de compétence. Il semble donc pertinent d'essayer de modéliser différents niveaux de compétences et de connaissance pour proposer un jeu de parades adapté à chaque classe d'attaquant.
Enfin, trouver des domaines d'application des anticipation games autre que la sécurité des réseaux semble une piste prometteuse, car de nombreux domaines, à l'instar de la biologie, ont besoin de modèles mêlant les notions de dépendances, de temps et d'interaction. On pense ici, en particulier, à la modélisation de l'évolution en biologie qui combine théorie des jeux et graphes [Now06].

Mise en pratique. La mise en pratique du modèle a donné lieu à des contributions dans deux directions : d'une part, la réalisation d'un outil pour vérifier des propriétés et rechercher des stratégies et d'autre part, le développement d'outils et de techniques d'analyse réseau pour la construction du modèle.

Au chapitre 4, nous avons présenté *NetQi*, l'outil que nous avons développé pour analyser les anticipation games. Au chapitre 5, nous avons présenté notre outil *NetAnalyzer*qui est utilisé pour construire le jeu de dépendances d'un réseau à partir d'une trace réseau. Cet outil utilise de nombreuses optimisations pour tenter de faire passer le modèle à l'échelle des gros réseaux. Nos expérimentations ont montré qu'il est possible d'analyser des réseaux ayant jusqu'à 15 000 nœuds. Dans le cadre de la construction automatique de modèle, nous avons aussi développé des techniques d'analyse de flux réseau pour gérer les protocoles qui utilisent des méthodes de camouflage.

Les directions futures dans la mise en pratique semblent s'articuler autour de deux axes : d'une part, améliorer le passage à l'échelle en développant des abstractions pour réduire la taille du modèle analysé par *NetQi*. Une direction prometteuse serait de faire une abstraction qui comprime les services existants qui sont semblables. D'autre part, améliorer l'interface utilisateur permettrait de rendre ce framework plus accessible. En particulier, bien que *NetQi* possède une interface graphique, celle-ci est encore rudimentaire et demande à être étendue. En particulier, l'ajout de nombreuses techniques de visualisation comme les treemaps permettraient de rendre l'analyse des anticipation games beaucoup plus intuitive.

Index

timeline d'événements, 152
4G, 33

, 14

abstraction, 15
arête, 18
arcs, 18, 19
arête
    boucle, 18
arête dirigée, 18
AssetRank, 34
ATL, 52
Attack graph, 38
Attack state graph, 36
attack tree, 35
automate de jeu temporisé, 66
automate fini, 17

benchmarking, 15
blow-up, 21
botnets, 26

CERT, 12
classificateur, 106
clustering, 106
contrainte financière, 11
contre-exemple, 17
cygne noir, 22

Datalog, 39
DDOS, 166
Déni de service, 24
Derivation fact nodes
    Logical attack graph, 40
Derivation nodes
    Logical attack graph, 40
distortion, 22

effet collatéral, 24
Emplacement, 157
emplacement, 152
Etat, 56
Etat, 56
état, 18
étiqueter un graphe, 18
Complexité
    EXPTIME, 21

Fact nodes
    Logical attack graph, 40
fenêtre de vulnérabilité, 58
Framwork, 33

garde, 18
gardes, 19
graphe, 18
Graphe de dépendances, 56
Graphe de dépendances, 55
graphe dirigé, 18

IDS, 162
interprétation abstraite, 13
irc, 109

Kripke, 19
kripke, 16, 19

label, 18
liste de contrôle, 11
Logical attack graph, 39

mean effort to failure, 36

méthodes formelles, 13
model-checker, 17
model-checking, 16
modèle, 15
modèle
    formel, 15
MulVal, 39
Reasoning Engine
    MulVal, 39
Scanner
    MulVal, 39

N-Gram, 115
Needham-Schroeder, 13
netqi, 80
NetSpa, 40
noeud, 18
Complexité
    NP, 22
    P, 22

octave, 12

pénalité, 152
parade
    non-optimalité, 11
parades, 10
Postcondition, 38
postconditions, 60
Précondition, 38
préconditions, 60
predictive graph, 42
Primitive facts nodes
    Logical attack graph, 40
Privilege graphs, 36
proposition, 19
proposition
    atomique, 19
proposition atomique, 20
propositions atomique, 56
Propriété d'accessibilité, 37
propriété temporelle, 20
protocole cryptographique, 13
PSI, 12

Chronologique
    Règle, 158
Globale
    Règle, 158
Locale
    Règle, 158
Opérationnelle
    Règle, 158
transversale
    Règle, 158
Relation d'équivalence, 55
Relation de dépendance, 55
Rice, 13
risque, 10
risque
    analyse, 10
risque
    évaluation, 10
    analyse, 10, 11
    gestion, 10

Scenario graph, 38
SEIR, 42
simulation, 16
sommet, 18
SRI, 42
Structure de Kripke, voir Kripke

test, 16
théorie des jeux, 51
transition, 18
transition, 18
tunnel, 109
typage, 22

UML, 15

validation, 15
vérification, 15
vérification de programme, 13
vérification déductive, 16
vers, 26
VOIP, 9
vulnérabilité

fonctionnelle, 11

Bibliographie

[AD01] Christopher Alberts and Audrey Dorofee. An inroduction to the octave method. Technical report, CERT, 2001. 12

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5) :672–713, 2002. 52, 70

[Ama] Amazon. Amazon online shopping : http ://www.amazon.com. website. 52

[Ana93] Z. Anastovski. Risk analysis in computer networks - a review. Technical report, Center for Computer Security Research, Wollongong (NSW, Australia), 1993. 10

[Art02] M. Artz. *NetSPA : a Network Security Planning Architecture*. PhD thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science., 2002. 40, 45

[AWK02] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *CCS '02 : Proceedings of the 9th ACM conference on Computer and communications security*, pages 217–224, New York, NY, USA, 2002. ACM Press. 38, 40, 45, 49

[Azu] Azureus. Message stream encryption : http ://www.azureuswiki.com/index.php/message_stream_encryption. 101, 113

[B.97] Myerson R. B. *Game Theory : Analysis of Conflict*. Harvard University Press, 1997. 52

[Bab05] Babachiz. Managing peer-to-peer traffic with cisco service control technology. Technical report, CISCO, Febuary 2005. 103

[Bal94] R. Baldwin. Kuang : Rule based security checking. Technical report, MIT/LCS, 1994. 33

[Bas93] R. Baskerville. Information system security design methods : Implication for information system development. In *ACM Computing Surveys*, volume 25, pages 375–414, 1993. 31

[BBF+01] B Bérard, M Bidoit, A Finkel, F Laroussinie, A Petit, L Petrucci, and P Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001. 13, 17, 61, 138

[BCC+03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI'03)*, pages 196–207, San Diego, California, USA, June 7–14 2003. ACM Press. 13

[BCD+07] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Uppaal-tiga : Time for playing games ! In Werner Damm and Holger Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125. Springer, 2007. 82

[BGL07] Elie Bursztein and Jean Goubault-Larrecq. A logical framework for evaluating network resilience against faults and attacks. In *12th annual Asian Computing Science Conference (ASIAN)*, pages 212–227. Springer-Verlag, Dec. 2007. 3, 28, 52

[BHRP06] T. Brihaye, T. A. Henzinger, J.F. Raskin, and V. Prabhu. Minimum-time reachability in timed games. In *FORMATS 06*, Lecture Notes in Computer Science. Springer-Verlag, 2006. 66, 68

[Bib77] K. J. Biba. Integrity, considerations for secure computer systems. Technical Report MTR 3153, MITRE, 1977. 14

[BJM08] Adam Barth, Collin Jackson, and John C. Mitchell. Securing browser frame communication. In *17th USENIX Security Symposium*, 2008. 101

[BL76] D. E. Bell and L. J. LaPadula. Secure computer systems : Unified exposition and multics interpretation. Technical Report MTR 2997, MITRE, 1976. 14

[BM09] Elie Bursztein and J. C. Mitchell. Using strategy objectives for network security analysis. In *5th International Conferences on Information Security and Cryptology INSCRYPT*. Springer-Verlag, 2009. 3, 28, 126

[BN89] D. F. C. Brewer and M. J. Nash. The chinese wall security policy. In *IEEE Symposium on Security and Privacy*, pages 206–214, 1989. 14

[BS06] S.A Baset and H. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. In *IEEE infocom 2006*, 2006. 101

[BTA+06] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *SIGCOMM Comput. Commun. Rev.*, 36(2) :23–26, 2006. 104, 107

[Bur] Elie Bursztein. Netqi http ://www.netqi.org. 28, 79

[Bur06] Elie Bursztein. Netanalyzer homepage : http ://code.google.com/p/netanalyzer/, 2006. 28, 96, 103

[Bur07a] E. Bursztein. Time has something to tell us about network address translation. In *NordSec 2007 : The 12th Nordic Workshop on Secure IT Systems*, Nov. 2007. 3, 28, 96

[Bur07b] Elie Bursztein. Network incidents anticipation. In Christopher Kruegel, Richard Lippmann, and Andrew Clark, editors, *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID'07)*, volume 4637 of *Lecture Notes in Computer Science*, Gold Coast, Australia, September 2007. Springer. Poster presentation. 3, 28, 52

[Bur08a] Elie Bursztein. Extending anticipation games with location, penalty and timeline. In Pierpaolo Degano, Joshua Guttman, and Fabio Martinelli, editors, *Proceedings of the 5th International Workshop on Formal Aspects in Security and Trust (FAST'08)*, Lecture Notes in Computer Science, Malaga, Spain, october 2008. Springer Verlag. 3, 28, 153

[Bur08b] Elie Bursztein. NetQi : A model checker for anticipation game. In Moonzoo Kim and Mahesh Viswanathan, editors, *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis (ATVA'08)*, Lecture Notes in Computer Science, Seoul, Korea, October 2008. Springer. 3, 28, 79

[Bur08c] Elie Bursztein. Probabilistic protocol identification for hard to classify protocol. In Jose A. Onieva, Damien Sauveron, Serge Chaumette, Dieter Gollmann, and Konstantinos Markantonakis, editors, *Proceedings of the 2nd International Workshop on Information Security Theory and Practices (WISTP'08)*, volume 5019 of *Lecture Notes in Computer Science*, pages 49–63, Sevilla, Spain, May 2008. Springer. Best paper award. 3, 26, 28, 96, 105

[CABV06] V. Colizza, A.Barrat, M. Barthelemy, and A. Vespignani. The modeling of global epidemics : stochastic dynamics and predictability. *Bulletin of Mathematical Biology*, 68 :1893–1921, 2006. 43, 45

[CAL+08] Senthilkumar G. Cheetancheri, John-Mark Agosta, Karl N. Levitt, Felix Wu, and Jeff Rowe. Optimal cost, collaborative, and distributed response to zero-day worms - a control theoretic approach. In Richard Lippmann, Engin Kirda, and Ari Trachtenberg, editors, *Recent Advances in Intrusion Detection RAID 08*, pages 231–250, 2008. 152

[CC77] P. Cousot and R. Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL'77*, LA, Ca, 1977. ACM. 13

[CCG+02] Ro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and O Tacchella. Nusmv 2 : An opensource tool for symbolic model checking. In *Computer Aided Verification*, pages 359–364. Springer, 2002. 38

[CCGR00] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv : a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2 :2000, 2000. 38

[CDF+05] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR 2005 – Concurrency Theory*, volume 3653 of *LNCS*, pages 66–80. Springer-Verlag, 2005. 79, 82

[CER] CERT. Cert : http ://www.cert.org/. 98

[Che] G. Cheng. Analysis on ddos tool stacheldraht v1.666 : http ://www.sans.org/resources/malwarefaq/stacheldraht.php. 101, 111

[Chu62] A. Church. Logic, arithmetics and automata. In *Congress of Mathematician*, pages 23–35. Institut Mittag-Leffler, 1962. 52

[CM06] S. P. Chung and A. K. Mok. Allergy attack against automatic signature generatio. In *RAID Recent Advances in Intrusion Detection*, 2006. 103

[CMR01] V. Cortier, J. Millen, and H. Ruess. Proving secrecy is easy enough. In *In 14th IEEE Computer Security Foundations Workshop*, 2001. 13

[CO00] Fr&#233 ;d&#233 ;ric Cuppens and Rodolphe Ortalo. Lambda : A language to model a database for detection of attacks. In *RAID '00 : Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, pages 197–216, London, UK, 2000. Springer-Verlag. 34, 38

[Con93] GBT Conseils. Les méthodes d'analyse de risques. Technical report, GBT Conseils, 1993. 33

[Cor] Advanced Research Corporation. Security Auditor Resarch Assistant : http ://www-arc.com. 99

[Dac94] M. Dacier. *Vers une Evaluation Quantitative de la Securite Informatique*. PhD thesis, Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS, Dec. 1994. 10, 11, 33, 36

[dAFH+03] L. de Alfaro, M. Faella, T. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *14th International Conference on Concurrency Theory*, volume 2761 of *LNCS*, pages 144–158. Springer-Verlag, 2003. 52, 70, 71

[DDK96] Marc Dacier, Yves Deswarte, and Mohamed Kaaniche. Models and tools for quantitative assessment of operational security. In *12th International Information Security Conference*, pages 177–186, May 1996. 34, 36, 45, 166

[Der] Renaud Deraison. nessus : security scanner. 38, 96, 99

[DFM+06] Holger Dreger, Anja Feldmann, Michael Mai, Vern Paxson, and Robin Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In *Usenix*, 2006. 103

[DMTY97] M. Debbabi, M. Mejri, N. Tawbi, and I. Yahmadi. Formal automatic verification of authentication cryptographic protocols. In *1st IEEE International Conference on Formal Engineering Methods*, 1997. 13

[dn04] Journal du net. Bouygues telecom privé de réseau, 2004. 9

[Dou08] Chad R Dougherty. Debian and ubuntu openssl packages contain a predictable random number generator. Vulnerability Note VU 925211, CERT, May 2008. 58

[Dvo] D. Dvoinikov. Htthost (http proxy) :http ://www.htthost.com/htthost.boa. 101

[DY83] D. Dolev and A. C. Yao. On the security of pubic key protocols. In IEEE, editor, *Transactions on Information Theory*, pages 198–208, 1983. 13

[Eey] Eeye. Retina vulnerability scanner : http ://www.eeye.com. 99

[EVK02] S.T. Eckmann, G. Vigna, and R.A. Kemmerer. STATL : An Attack Language for State-based Intrusion Detection. *Journal of Computer Security*, 10(1/2) :71–104, 2002. 34

[Far91] B Farquhar. One approach to risk assessment. *Computers and Security*, 10 :21–23, 1991. 32

[FDV94] A. Flahault, S. Deguen, and J. Valleron. A mathematical model for the european spread of influenza. *European Journal of Epidemiology*, 10 :471–474, 1994. 43

[FGBZ03] Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. On effectiveness of link padding for statistical traffic analysis attacks. In *ICDCS '03 : Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 340, Washington, DC, USA, 2003. IEEE Computer Society. 113

[FV91] A. Flahault and A.-J. Valleron. A method for assessing the global spread of hiv-1 infection based on air-travel. In *Math. Pop. Studies 3*, 1991. 43

[Fyo] Fyodor. Nmap : free open source utility for network exploration or security auditing http ://www.insecure.org/nmap/. 96, 103

[Fyo98] Fyodor. Remote os detection via tcp/ip stack fingerprinting. *phrack*, volume 8, october 1998. 97

[Gar89] P. E. Gardner. Evaluation of five risk assessment programs. *Computer and Security*, 8 :479–485, 1989. 32

[GFI] GFI. Languard network security scanner : http ://www.gfi.com/lannetscan/. 99

[GL00] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification (extended abstract). In Springer Verlag, editor, *Proceedings of the Workshop on Formal Methods in Parallel Programming, Theory and Applications (FMPPTA'2000)*, volume LNCS 1800, pages 977–984, 2000. 13

[Goo98] Google. search engine : www.google.com. website, 1998. 52

[Ham67] W. D. Hamilton. Extraordinary sex ratios. *Science*, 1967. 139

[HBG04] L. Hufnagel, D. Brockmann, and T. Geisel. Forecast and control of epidemics in a globalized world. In *Natl. Acad. Sci.*, 2004. 43

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10) :576–585, Oct 1969. 60

[Hof89] L. J. Hoffman. Smoking the bad actors : risk analysis in the age of microcomputer. In *Computers and Security*, volume 8, pages 299–302, 1989. 10

[HP06] T. Henzinger and V. Prabhu. Timed alternating-time temporal logic. In *Formats 06*, volume 4202, pages 1–18. Springer-Verlag, 2006. 25, 68, 69, 70, 71

[HSSW05] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Acas : automated construction of application signatures. In *MineNet '05 : Proceeding of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 197–202, New York, NY, USA, 2005. ACM Press. 103, 105, 106

[IAN] IANA. Matrix for protocol parameter assignment/registration procedures http ://www.iana.org/numbers.html. 105

[ICA07] ICANN. Dns attack factsheet. Technical report, ICANN, Mar 2007. 62

[ILP06] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *ACSAC '06 : Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference*, pages 121–130, Washington, DC, USA, 2006. IEEE Computer Society. 40, 42, 45

[INR08] INRIA. Le système coq : http ://coq.inria.fr, 2008. 16

[Joh06] Martin Johns. Sessionsafe : Implementing xss immune session handling. In *In Proc. ESORICS*, 2006. 101

[JSW02] S. Jha, O. Sheyner, and J. Wing. Two formal analysis of attack graphs. In *CSFW '02 : Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, pages 49–63, Washington, DC, USA, 2002. IEEE Computer Society. 38, 45

[Kam05] D. Kaminsky. Attacking distributed systems the dns case study. In *Black Hat Europe*, 2005. 113

[KBFc04] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc claffy. Transport layer identification of p2p traffic. In *IMC '04 : Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134, New York, NY, USA, 2004. ACM Press. 113

[KC03] C. Kreibich and J. Crowncroft. Creating intrusion detection signatures using honeypot. In *Second Workshop on Hot Topics in Networks*, Nov. 2003. 103

[Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5) :604–632, 1999. 133

[Kle01] J. Klensin. Rfc 2821 - simple mail transfer protocol. Technical report, IETF Network Working Group, 2001. 55

[KM32] W. O. Kermack and A. G. McKendrick. Contributions to the mathematical theory of epidemics. ii. the problem of endemicity. In *Proceedings of the Royal Society of London*, volume 138 of *A*, pages 55–83, 1932. 42

[KM33] W. O. Kermack and A. G. McKendrick. Contributions to the mathematical theory of epidemics. iii. further studies of the problem of endemicity. In *Proceedings of the Royal Society of London*, volume 141 of *A*, pages 94–122, 1933. 42

[Kot04] Munir Kotadia. Viruses exploit microsoft patch cycle. Technical report, Zdnet, Nov 2004. 59

[kp03] kevin poulsen. Slammer worm crashed ohio nuke plant network. Technical report, Security focus, 2003. 9

[KV03] C. Kruegel and G. Vigna. Anomaly Detection of Web-based Attacks. In *Proceedings of the* 10^{th} *ACM Conference on Computer and Communication Security (CCS '03)*, pages 251–261, Washington, DC, October 2003. ACM Press. 104

[LI05] R. P. Lippmann and K. W. Ingols. An annotated review of past papers on attack graphs. Technical Report IA-1, Lincoln Laboratory MIT, March 2005. 35

[LIP+06] R. Lippmann, K Ingols, K Piwowarski, K Kratkiewicz, M. Artz, and R Cunningham. Validating and restoring defense in depth using attack graph. In *MILCOM*, 2006. 40, 45

[Low95] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. In Bird, editor, *Information Processing Letters*, volume 15, 1995. 13

[LSC+06] Zhichun Li, Manan Sanghi, Brian Chavez, Yan Chen, and Ming-Yang Kao. Hamsa : Fast signature generation for zero-day polymorphic worms with provable attack resilience. In *IEEE Symposium on Security and Privacy*, May 2006. 103

[LW05] Kong-wei Lye and Jeannette M. Wing. Game strategies in network security. *Int. J. Inf. Sec.*, 4(1-2) :71–86, 2005. 52

[LWS02] R. Lippmann, S. Webster, and D. Stetson. The effect of identifying vulnerabilities and patching software on the utility of network intrusion detection. In *RAID '02 : Proceedings of the 5th International Workshop on Recent Advances in Intrusion Detection*, pages 307–326. Springer-Verlag, Oct 2002. 56, 58, 129, 152, 162

[Mai02] Uriel Maimon. Port scanning without the syn flag. *Phrack*, 49, 2002. 97

[MBZ+06] Vaibhav Mehta, Constantinos Bartzis, Haifeng Zhu, Edmund Clarke, and Jeannette Wing. Ranking attack graphs. In *Proceedings of Recent Advances in Intrusion Detection*, 2006. 34, 133

[Mica] Microsoft. Msn portal : http ://www.msn.com/. website. 52

[Micb] Microsoft. Windows live messenger http ://en.wikipedia.org/wiki/windows_live_messenger 101

[Mic08] Microsoft. Microsoft patch cycle information : http ://www.microsoft.com/protect/computer/updates/bulletins/200809.mspx, Sep. 2008. 59

[Mita] Mitre. Common Vulneralilities and Exposures : http ://cve.mitre.org. 98

[MITb] MITRE. the mitre corporation : http ://www.mitre.org. 98

[Moc87] P. Mockapetris. Rfc1035 : Domain names - implementation and specification. Technical report, Network Working Group, 1987. 113

[MPN+05] L.A. Meyers, B. Pourbohloul, M.E.J. Newman, D.M. Skowronski, and R.C Brunham. Network theory and sars : Predicting outbreak diversity. In *J. Theor. Biol.*, volume 232, pages 71–81, 2005. 43

[MPS+03] D. Moore, V. Paxson, S. Savage, C Shannon, S. Staniford, and N. Weaver. The spread of the sapphire/slammer worm. Technical report, CAIDA and ICIR and LBNL and UCSD CSE and Silicon Defense, 2003. 47

[MS01] David Moore and Colleen Shannon. The spread of the code-red worm (crv2). Technical report, CAIDA, Jul 2001. 47

[MS03] N Markey and P Schnoebelen. Model checking a path. In Roberto M. Amadio and Denis Lugiez, editors, *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *Lecture Notes in Computer Science*, pages 251–265, Marseilles, France, August 2003. Springer. 143

[MS05] Ajay Mahimkar and Vitaly Shmatikov. Game-based analysis of denial-of-service prevention protocols. In *18th IEEE Computer Security Foundations Workshop (CSFW), Aix-en-Provence, France, June 2005, pp. 287-301. IEEE Computer Society, 2005.*, pages 287–301. IEEE Computer Society, Jun 2005. 52, 165

[MW06] Alok Madhukar and Carey Williamson. A longitudinal study of p2p traffic classification. In *MASCOTS '06 : Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pages 179–188, Washington, DC, USA, 2006. IEEE Computer Society. 101

[MZ05] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *SIGMETRICS '05 : Proceedings of the 2005*

ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pages 50–60, New York, NY, USA, 2005. ACM Press. 103

[NKS05] J. Newsome, B. Karp, and D. Song. Polygraph : Automatically generating signatures for polymorphic worms. In *IEEE Symposium on Security and Privacy*, May 2005. 103

[NM] Makoto Nagao and Shinsuke Mort. A new method of n-gram statistics for large number of n and automatic extraction of words and phrases from large text data of japanese. 115, 121

[Now06] M. A. Nowak. *Evolutionary Dynamics : Exploring the Equations of Life*. Harvard University Press., 2006. 52, 178

[NWF06] N. Nethercote, R. Walsh, and J. Fitzhardinge. Building Workload Characterization Tools with Valgrind. *Workload Characterization, 2006 IEEE International Symposium on*, pages 2–2, 2006. 148

[OBM06] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. In *CCS '06 : Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345, New York, NY, USA, 2006. ACM Press. 39, 45

[oD83] Departement of Defense. Departement of defense trusted computer system evaluation criteria. Technical Report CSC-STD-001-83, DOD, 1983. 14

[oD85] Departement of Defense. Departement of defense trusted computer system evaluation criteria. Technical Report 5200.28-STD, DOD, 1985. 14

[ODK99] Rodolphe Ortalo, Yves Deswarte, and Mohamed Kaâniche. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Trans. Software Eng.*, 25(5) :633–650, 1999. 34, 36, 45

[OG05] Xinming Ou and Sudhakar Govindavajhala. Mulval : A logic-based network security analyzer. In *In 14th USENIX Security Symposium*, 2005. 39, 45

[OM27] Kermack W. O. and A. G. McKendrick. A contribution to the mathematical theory of epidemics. In *Proceedings of the Royal Society of London*, volume 115 of *A*, pages 700–721, 1927. 42, 45

[Pan03] Liam Paninski. Estimation of entropy and mutual information. *Neural Comput.*, 15(6) :1191–1253, 2003. 106

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994. 22

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking : Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998. 133

[Pid] Pidgin. Pidgin (gaim) messenger : http ://www.pidgin.im/. 118

[PM00] Dan Pelleg and Andrew W. Moore. X-means : Extending k-means with efficient estimation of the number of clusters. In *ICML*, pages 727–734, 2000. 106, 107

[Por92] P.A. Porras. STAT – A State Transition Analysis Tool for Intrusion Detection. Master's thesis, Computer Science Department, University of California, Santa Barbara, June 1992. 34

[Pos81] J. Postel. Rfc792 : Internet control message protocol. Technical report, Network Working Group, 1981. 111

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL '89 : Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190, New York, NY, USA, 1989. ACM Press. 52

[Pro] Emule Project. http ://www.emule-project.net/. 101

[PS98] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *NSPW '98 : Proceedings of the 1998 workshop on New security paradigms*, pages 71–79, New York, NY, USA, 1998. ACM Press. 34

[pu95] CERIAS purdue university. Satan scanning tool history : http ://www.cerias.purdue.edu/about/history/coast/satan.php, 1995. 99

[Qua] Quadong. Linux layer 7 packet classifier http ://sourceforge.net/projects/l7-filter/. 103

[RA00] Ronald W. Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *SP '00 : Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 156–165, Washington, DC, USA, 2000. IEEE Computer Society. 38, 45, 49

[Ras07] E Rasmusen. *Games and Information*. Blackwell publishing, 2007. 25, 51, 139

[RL85] L.A. Revachev and I.M. Longini. A mathematical model for global spread of influenza. *Math Biosci*, 75 :3–22, 1985. 43, 45

[Roe] Marty Roesch. The open source network intrusion detection system. 103

[RS98] C. R. Ramakrishnan and R. Sekar. Model-based vulnerability analysis of computer systems. In *In Proceedings of the 2nd International Workshop on Verification, Model Checking and Abstract Interpretation*, 1998. 34, 37

[RWJ08] Daniel Ramsbrock, Xinyuan Wang, and Xuxian Jiang. A first step towards live botmaster traceback. In Richard Lippmann, Engin Kirda, and Ari Trachtenberg, editors, *Recent Advances in Intrusion Detection RAID 08*, pages 59–77, 2008. 152

[SBT01] Yevgeny Seldin, Gill Bejerano, and Naftali Tishby. Unsupervised sequence segentation by a mixture of switching variable memory Markov sources. In *Proc. 18th International Conf. on Machine Learning*, pages 513–520. Morgan Kaufmann, San Francisco, CA, 2001. 115

[Sch96] Bruce Schneier. *Applied Cryptography*. John Wiley, 1996. 35

[Sch99] B. Schneier. Attack trees : Modeling security threats. *Dr. Dobb's journal*, Dec. 1999. 34, 35, 45

[Sec] SecurityFocus. Bugtack mailling list : http ://www.securityfocus.com/archive/1. 98

[SEVS04] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Operating Systems Design and Implementation*, Dec. 2004. 103

[Sha51] C. E. Shannon. Prediction and entropy of printed english. *Bell System Technical Journal*, 1951. 104, 115

[SHJ+02a] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *SP '02 : Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 273 – 284, Washington, DC, USA, 2002. IEEE Computer Society. 38, 45, 99

[SHJ+02b] Oleg Sheyner, Joshua W. Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *IEEE Symposium on Security and Privacy*, pages 273–284, 2002. 60

[SO07] Reginald Sawilla and Xinming Ou. Googling attack graphs. Technical Report TM 2007-205, Defence R and D Canada, 2007. 34

[SPEC00] L. P. Swiler, C Phillips, D Ellis, and S Chakerian. Computer attack graph generation tool. In *DARPA Information Survivability Conference and Exposition*, 2000. 34

[Stø] D. Stødle. Ping tunnel. 111

[Tal07] Nassim Nicholas Taleb. *The Black Swan*. Penguin Books, 2007. 22

[Ten06] Tenable. Passive vulnerability scanning : Gw university case study. Technical report, Tenable network security, 2006. 96

[Tew07] Erik Tew. Attacks on the wep protocol, 2007. 10

[TL00] Steven J. Templeton and Karl Levitt. A requires/provides model for computer attacks. In *NSPW '00 : Proceedings of the 2000 workshop on New security paradigms*, pages 31–38, New York, NY, USA, 2000. ACM Press. 34

[Tur] J. Turkulainen. Backdoor spotcom analysis http ://www.securiteam.com/securityreviews/6z00n208uc.html. 101, 111

[VEK00] Giovanni Vigna, Steve T. Eckmann, and Richard A. Kemmerer. The stat tool suite. In *In Proceedings of DISCEX 2000*. IEEE Computer Society Press, 2000. 34

[vNM44] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Pres, 1944. 51

[VVZK] Giovanni Vigna, Fredrik Valeur, Jingyu Zhou, and Richard A. Kemmerer. Composable tools for network discovery and security analysis. 96

[War70] W. H. Ware. Security controls for computers systems : Report of defense science board task force on computer security. Technical Report AD-A076617/0, Rand Corporation, 1970. 14

[WCS05] Ke Wang, G. Cretu, and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Recent Advance in Intrusion Detection*, 2005. 103, 104, 106

[Zal06] Michal Zalewski. P0f2 : "dr. jekyll had something to hyde" passive os fingerprinting tool. Web, 2006. 97

[ZHM90] M. Zviran, J. C. Hoge, and V. A. Micucci. Span - a dss for security plan analysis. *Computers and Security*, 9(2) :153–160, Apr 1990. 11, 32

[ZL96] Dan Zerkle and Karl Levitt. Netkuang : a multi-host configuration vulnerability checker. In *SSYM'96 : Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, pages 195–201. Usenix, 1996. 33