Définition et évaluation d'un mécanisme de génération de règles de corrélation liées à l'environnement.

Erwan Godefroy

N° d'ordre : 2016-31-TH

CentraleSupélec

Ecole Doctorale MATISSE
« Mathématiques, Télécommunications, Informatique, Signal, Systèmes Electroniques »

IRISA/CIDRE

THÈSE DE DOCTORAT

DOMAINE : STIC
Spécialité : Informatique

Soutenue le 30 Septembre 2016

par :

Erwan GODEFROY

Définition et évaluation d'un mécanisme de génération de règles de corrélation liées à l'environnement

Composition du jury :

Directeur de thèse :	Michel Hurfin	Chargé de recherche (Inria)
Co-directeur de thèse :	Frédéric Majorczyk	Docteur-Ingénieur (DGA-MI)
Co-directeur de thèse :	Eric Totel	Professeur (CentraleSupélec)
Président du jury :	Isabelle Chrisment	Professeur (Télécom Nancy)
Rapporteurs :	Hervé Debar	Professeur (Télécom SudParis)
	Vincent Nicomette	Professeur (INSA Toulouse)

Remerciements

Je souhaite remercier globalement toutes les personnes qui ont contribué à la réalisation de cette thèse.

Je tiens à remercier tout d'abord tous les membres du jury pour avoir accepté d'évaluer mes travaux. En particulier, je remercie les rapporteurs Hervé Debar et Vincent Nicomette pour avoir accepté d'examiner ce mémoire et Isabelle Chrisment pour avoir accepté de présider le jury.

Je remercie ensuite mes encadrants, Éric Totel, Michel Hurfin et Frédéric Majorczyk, qui ont non seulement lu et relu ce manuscrit pour qu'il prenne sa forme finale mais qui m'ont également guidé et aidé à clarifier, développer et valider mes idées tout au long de ma thèse.

Je remercie également de manière globale tous les membres de l'équipe CIDRE (qu'ils soient permanents ou non permanents) pour l'accueil et l'ambiance au sein du labo. Côté DGA, je remercie les personnes des différents laboratoires, départements ou divisions avec qui j'ai eu l'occasion de travailler, en particulier pour les retours sur mes travaux. Je tiens à remercier plus particulièrement Marie-Thérèse pour m'avoir laissé le temps nécessaire pour que je puisse travailler sur ma thèse.

Table des matières

Table des figures

Liste des tableaux

Introduction

Les systèmes d'information sont aujourd'hui omniprésents dans tous les secteurs d'activité. Ces systèmes sont constitués de différents nœuds (serveurs, postes clients, routeurs) organisés en sous-réseau au sein d'une infrastructure. La complexité de ces systèmes les rend souvent vulnérables à des attaques. Une attaque correspond à un ensemble d'actions menées pour atteindre un objectif déterminé et qui viole la politique de sécurité du système visé. Les attaques se différencient par leurs complexités, les techniques mises en jeu et l'enchaînement des actions nécessaires à leur réalisation. Il peut s'agir de la prise de contrôle d'un site Web, d'un déni de service distribué visant à rendre inaccessible un service ou d'un vol de données sensibles.

La sécurité de ces systèmes est ainsi un enjeu fondamental et passe par la mise en place d'une politique de détection adaptée. Cela se traduit par le déploiement d'outils de détection d'intrusions (IDS) permettant de détecter des comportements suspects sur le système. Ces IDS peuvent laisser reposer la détection sur une base de signatures dans laquelle le comportement malveillant à détecter est explicitement défini. Il existe également des IDS utilisant des approches de détection comportementales cherchant à détecter des déviations entre le comportement normal du système et le comportement constaté. Un IDS lève ainsi une alerte lorsque l'observation réalisée correspond à la description d'une attaque ou dévie par rapport au comportement normal du système.

Les problèmes du fonctionnement de ces IDS sont les faux positifs et les faux négatifs d'une part et la granularité des alertes d'autre part. Un faux positif correspond à une alerte levée qui ne correspond pas à une attaque réelle. Ce phénomène peut être causé par une mauvaise spécification des signatures de détection dans le cadre des IDS par signatures ou d'une déviation bénigne du comportement normal du système dans le cadre des IDS comportementaux. À ces deux causes s'ajoutent la notion plus floue de tentative d'attaque qui correspond à un échec de l'action d'attaque. Cet échec peut être lié au caractère non vulnérable du système par rapport à l'attaque en question. Selon les circonstances, la détection et la levée d'alertes liées à ces tentatives d'attaques sont pertinentes ou non. Lorsque ces alertes identifiant des tentatives d'attaques génèrent un bruit de fond incessant qui risque de noyer des alertes plus pertinentes, elles peuvent être classées dans la même catégorie que les faux positifs. Par exemple, un serveur directement accessible depuis l'Internet subit des tentatives d'attaques incessantes, alors qu'un serveur interne isolé ne devrait pas du tout en subir. Dans le premier cas, les alertes liées à ces tentatives d'attaques noient des alertes potentiellement plus pertinentes, alors que dans le second cas, du fait de la situation du serveur au sein du système, la détection d'une tentative d'attaque doit être considérée comme un événement important.

Les faux négatifs sont des attaques réelles non détectées par un IDS. Cette non détection peut être liée à une mauvaise spécification des signatures de détection dans le cas des IDS par signatures (ou à une absence de signature pertinente) ou encore à une déviation non significative du comportement normal du système dans le cas d'IDS comportementaux.

Le niveau de granularité des alertes levées est souvent faible. Les IDS lèvent des alertes correspondant à une action élémentaire de l'attaquant, par exemple la réalisation d'une injection de code sur un service réseau. Lorsqu'une suite spécifique d'actions déterminées est nécessaire pour réaliser une attaque, il est alors important de regrouper les alertes élémentaires qui constituent le scénario d'attaque afin d'identifier la cible potentielle et la progression de cette attaque. Cette suite d'actions de la part de l'attaquant pour atteindre son objectif est appelé scénario d'attaque. Les scénarios d'attaques peuvent prendre différentes formes. Il peut s'agir d'un enchaînement d'exploitations de diverses vulnérabilités sur plusieurs machines permettant de prendre progressivement le contrôle

d'un système pour atteindre par exemple des informations intéressantes. Lors de la réalisation de ce scénario, certaines actions de l'attaquant correspondent à des actions qui sont sans ambiguïté identifiées comme des attaques (par exemple, la réalisation d'une injection SQL à partir d'un formulaire Web ou l'utilisation d'un dépassement de tampon dans une application vulnérable). D'autres actions ne sont, prises individuellement, pas immédiatement identifiables comme des attaques. Par exemple, la création d'un nouvel utilisateur privilégié, la mise en place d'un nouveau service sur une machine, la création d'un fichier. Cependant, c'est l'enchaînement de ses actions qui donne au scénario son caractère malveillant.

L'introduction de techniques de corrélation d'alertes permet de réduire les conséquences de ces différents problèmes. La corrélation d'alerte est définie comme étant l'ensemble des techniques permettant de réduire le nombre d'alertes et de créer des notifications d'un niveau d'abstraction élevé (appelées méta-alertes). Cette corrélation est rendue nécessaire par la faiblesse des sondes de détection d'intrusion qui génèrent un nombre important de faux positifs et ne détectent pas l'intégralité des attaques, ainsi que par le besoin de regrouper les alertes liées logiquement entre elles. La corrélation peut impliquer des événements ne disposant de différents niveaux d'abstraction. Un événement est la manifestation d'une action sur le système. Un événement brut est un événement physique ou logique issu directement d'une source de données physique et qui n'a pas subi d'interprétation ou d'analyse spécifique (par exemple, une capture réseau brute). Un message est un événement qui constitue une interprétation de la signification d'un événement ou d'un ensemble d'événements. Les événements interprétés peuvent inclure des événements bruts analysés ou bien d'autres messages. Un message est généré par un observateur dans un format donné. Les observateurs regroupent à la fois les outils fonctionnels natifs permettant de générer des journaux et des outils plus spécialisés tels que des IDS. Les messages qui constituent ainsi un premier niveau d'interprétation sont représentés par exemple sous la forme de fichiers de journalisation (logs). Lorsqu'un message est généré par un IDS et correspond à l'interprétation d'une attaque, il s'agit alors d'une alerte (ou alarme).

La mise en œuvre d'un processus de corrélation pertinent demande la disponibilité d'informations sur le système surveillé. Ces informations permettent à la fois de normaliser les alertes et événements issus de sources différentes et de déterminer la pertinence de certaines attaques par rapport à l'importance des éléments ciblés. Une base de connaissances rassemble ces informations. Une base de connaissances est constituée à la fois d'un ensemble de faits (ou de biens) qui caractérise la composition d'un système donné et également d'une base de règles permettant d'exprimer des relations entre les faits.

La reconnaissance de scénario d'attaques est une composante de la corrélation d'alertes consistant à rassembler au sein d'un ensemble cohérent un ensemble d'alertes ou d'événements liés logiquement entre eux et traduisant l'avancement de l'attaque vers un objectif donné. Cette étape vise à aider un administrateur à reconstruire le cheminement d'un attaquant en regroupant la succession des alertes liées à l'attaque, ou bien à corréler des événements qui, s'ils sont pris isolément, ne sont pas nécessairement identifiés comme relatifs à une attaque.

Les contributions apportées par cette thèse sont les suivantes :

— La définition d'un processus de génération de règles de corrélation. Ce processus est semi-automatique et permet de transformer progressivement un scénario d'attaque exprimé de manière générique en règles de corrélation adaptées au système surveillé.
— La construction d'une base de connaissances permettant de fournir les informations pertinentes au processus de génération de règles de corrélation. En particulier, le modèle comprend un ensemble de classes d'attaques et d'actions normales permettant de spécifier la visibilité des outils de détection.
— L'évaluation de l'applicabilité de la méthode sur un cas représentatif permettant d'évaluer la pertinence des règles de corrélation générées.
— L'évaluation de la résistance du processus de génération à certaines fautes concernant la base de connaissances.
— Une étude théorique et pratique de la complexité spatiale de l'algorithme de génération proposé.

L'organisation du mémoire est définie par le plan qui suit.

Le chapitre 1 décrit les problématiques de la corrélation d'alertes ainsi que les travaux qui s'y rapportent. Il est divisé en trois parties. La première détaille le processus de corrélation. La seconde présente les bases de connaissances utilisées comme support au processus de corrélation. La troisième propose une étude sur la classification des types attaques.

Le chapitre 2 consiste en une comparaison de deux formalismes permettant d'exprimer des scénarios d'attaque : les arbres d'attaque et les graphes d'attaques. La première partie est dédiée aux arbres d'attaque et la seconde partie aborde les graphes d'attaques. Dans chaque partie, les liens et points communs entre ces formalismes et la structure d'une règle de corrélation sont étudiés.

Le chapitre 3 présente la base de connaissances servant de support à notre processus de génération de règles de corrélation et constitue une première contribution. Ce chapitre est organisé en trois parties. La première revient sur la structure d'une base de connaissances existante nous servant de point de départ et que nous avons fait évoluer. Une seconde partie décrit les éléments introduits ou modifiés pour construire une base de connaissances adéquate par rapport à nos objectifs. La troisième étudie les problématiques de mise en œuvre et d'utilisation de cette base de connaissances.

Le chapitre 4 est notre principale contribution et présente le procédé de génération des règles de corrélation. Ce chapitre détaille les différentes étapes permettant de transformer progressivement un arbre d'attaque en règles de corrélation. Un langage d'actions permettant de spécifier de manière explicite la sémantique des feuilles d'un arbre d'attaque est introduit dans un premier temps. Son utilisation permet d'effectuer une transformation manuelle d'un arbre d'attaque en un arbre d'actions. Cet arbre d'actions subit par la suite une succession de transformations automatiques qui aboutissent à une représentation générique de règles de corrélation appelée arbre de corrélation. Une partie de ce chapitre est dédiée à la traduction de cette structure en règles de corrélation exprimées dans le langage ADeLe.

Le chapitre 5 présente une évaluation de notre approche. Ce chapitre est organisé en trois parties. Dans la première, une évaluation de la méthode sur une infrastructure représentative d'une petite entreprise est effectuée. Dans une seconde partie, l'influence de fautes survenant au sein de la base de connaissances sont mesurées. La synchronisation de la base de connaissances avec le système réel est en effet une caractéristique dont cette évaluation cherche à mesurer l'importance. Dans un troisième temps, nous réalisons une étude sur le passage à l'échelle de l'approche dans le but de déterminer les cas dans lesquels une telle méthode est applicable ou non.

Les travaux effectués sont récapitulés dans la conclusion et les perspectives qui en découlent y sont également présentées.

Chapitre 1

État de l'art sur la corrélation d'alertes

Dans ce chapitre consacré à l'état de l'art, une première partie aborde le thème de la corrélation d'alertes afin de positionner les travaux de cette thèse. La seconde partie présentera les outils et approches utilisés pour modéliser l'environnement d'exécution et les informations externes servant de support à la corrélation d'alertes. Une troisième partie aborde les problématiques liées à l'identification et à la classification des types d'attaques.

1.1 Processus de corrélation d'alertes

Cette première partie propose dans un premier temps d'introduire la notion de corrélation d'alertes, puis dans un second temps d'étudier les différentes caractérisations possibles du processus de corrélation. Dans un troisième temps, les spécificités des différentes approches proposées dans la littérature sont abordées.

1.1.1 Corrélation d'alertes

A Contexte

La corrélation d'alertes est un processus rendu nécessaire pour gérer le volume de données produit par les différents IDS du système et pour fournir une vue d'ensemble sur l'état de ce système. En effet, dans un système d'information générant un volume de données important, le nombre de notifications de sécurité peut être très élevé. Ce nombre élevé est la conséquence de plusieurs phénomènes. Une même attaque peut provoquer l'émission de plusieurs alertes de la part d'un ou plusieurs IDS. Des attaques répétitives ou séquentielles (ex : scan de ports, scan de machines, déni de services, attaque par force brute) peuvent mener à la génération de plusieurs centaines d'alertes par IDS. De plus, une partie importante de faux positifs est présente dans les alertes générées. Ces faux positifs sont causés par des règles de détection de mauvaise qualité, non adaptées au système (dans le cas des IDS utilisant une base de signatures) ou des irrégularités bénignes dans le comportement du système ou encore une évolution dans l'utilisation du système (dans le cas des IDS comportementaux). Les opérateurs en charge de la supervision ont ainsi à gérer un volume de données important constitué de notifications apportant un faible niveau d'abstraction.

B Objectifs de la corrélation d'alertes

La corrélation d'alertes pallie ces problèmes avec deux objectifs principaux qui sont (a) la réduction du nombre d'alertes et (b) la création de notifications disposant d'un niveau d'abstraction élevé afin de proposer une vue haut niveau sur les intrusions détectées.

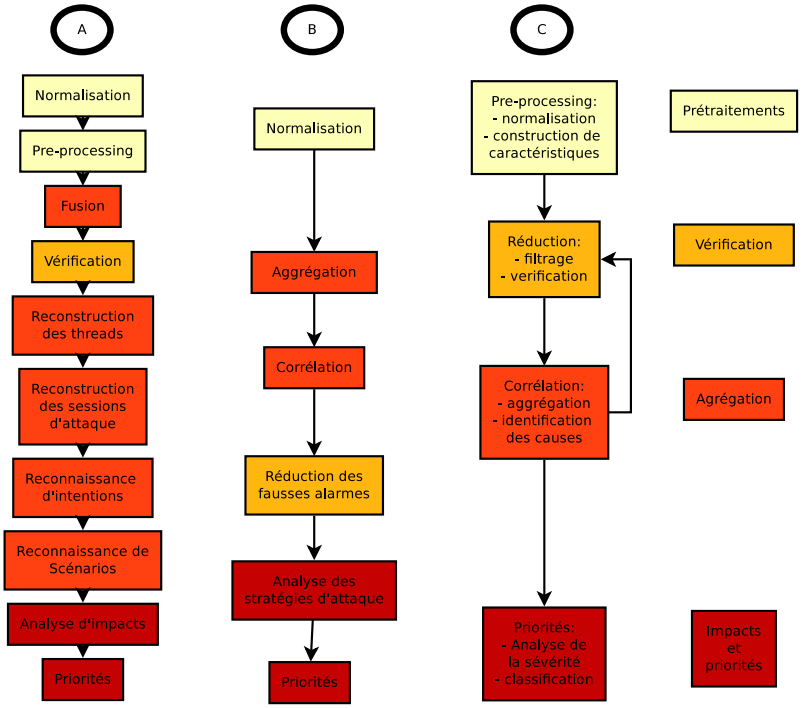


FIGURE 1.1 – Différentes visions du processus de corrélation. A : [Val06], B : [SG06], C : [SMFDV13]. A droite, les grandes fonctions qui ressortent dans tous ces travaux.

Réduction du nombre d'alertes Cet objectif consiste essentiellement à éliminer les fausses alertes (faux positifs) et à regrouper les alertes redondantes ou périodiques. Le regroupement des alertes similaires suppose de connaître le format et la sémantique des différentes notifications, alors que l'élimination de faux positifs demande des connaissances précises sur le système attaqué.

Création d'une vue haut niveau sur les intrusions détectées Cet objectif consiste à reconstruire des scénarios d'attaque à partir des notifications de bas niveau et à attribuer des scores, des indications de pertinence ou de criticité aux méta-alertes générées qui sont destinées aux opérateurs en charge de la supervision.

1.1.2 Architecture du processus de corrélation

Une première proposition de découpage fonctionnel du processus de corrélation d'alertes est présentée dans [ZU04] où les trois parties (prétraitement, l'analyse et la corrélation) sont identifiées. Par la suite, le modèle exposé dans [Val06] (A sur la Figure 1.1) sert généralement de référence pour le découpage du processus de corrélation en fonctions élémentaires. Ce dernier est repris et étendu dans quelques travaux ([SG06] et [SMFDV13] respectivement B et C sur la Figure 1.1) qui ajoutent, renomment, ou généralisent certaines étapes.

Malgré les différences entre les nomenclatures utilisées par différents auteurs, il est possible de ressortir un schéma général des fonctions se présentant sous la forme suivante :

(1) les prétraitements (normalisation et enrichissement)

(2) la vérification (fonction d'élimination des messages)

(3) l'agrégation (fonction de création de méta-alertes)

(4) l'analyse d'impacts et attribution de priorités (fonction de mise en valeur)

Le principe des différentes fonctions est explicité dans les paragraphes qui suivent.

A Les prétraitements

Avant de réaliser le cœur du travail de corrélation, des étapes de prétraitement des alertes brutes sont nécessaires. Les phases de prétraitement sont usuellement découpées en une phase de normalisation et une phase d'enrichissement.

La normalisation Les alertes sont générées par différents IDS utilisant des formats et des conventions différentes. L'étape de normalisation vise à homogénéiser la structure et la sémantique des alertes pour les exprimer dans un format unifié. La normalisation est à la fois syntaxique et sémantique (définition de dictionnaires de termes équivalents, utilisation d'une représentation unifiée des identifiants des différents objets).

L'enrichissement Cette étape vise à ajouter des informations externes à l'alerte, par exemple des attributs manquant en fonction de l'IDS émetteur de l'alerte (ajout de l'adresse IP dans le cas d'un IDS hôte (HIDS) ou bien d'une information temporelle comme suggéré dans [SMFDV13]) ou bien d'informations contextuelles sur des identifiants de vulnérabilités. Elle peut survenir à plusieurs reprises dans le processus global de corrélation. Cet enrichissement suppose de disposer d'une base de biens contenant les informations requises (topologie, informations complémentaires sur les IDS déployés, base de vulnérabilités).

B La vérification

Cette étape peut permettre de rejeter les alertes qui référencent des attaques exploitant des vulnérabilités absentes du système (vérification statique). Une vérification des conséquences observables d'une attaque sur le système peut également être réalisée afin de déterminer si l'attaque est réussie (vérification dynamique).

C L'agrégation

L'agrégation rassemble toutes les phases qui visent à réduire le nombre d'alertes en utilisant des regroupements d'alertes. Ces regroupements peuvent être réalisés sur la base d'une cause commune ou d'un objectif commun.

C.1 Fusion La fusion a pour but de regrouper au sein d'une méta-alerte (également appelée hyper-alerte) les alertes issues d'IDS similaires et référençant la même instance d'attaque [Val06]. Il est possible de raffiner en considérant que la fusion intervient dans les trois cas qui suivent [Mor04] :

(a) plusieurs alertes produites pour une même instance d'attaque et par un même IDS (scission)

(b) plusieurs alertes produites pour une attaque récurrente et par un même IDS (récurrence)

(c) plusieurs alertes produites par une attaque vue par plusieurs IDS de même type (redondance)

Les travaux traitent généralement les problèmes de récurrence et de redondance (la scission pouvant être considérée comme liée à une mauvaise configuration de l'IDS).

C.2 Identification d'attaques élémentaires Ce processus vise à regrouper des alertes produites par des IDS de types différents faisant référence à la même attaque ou à regrouper un ensemble d'alertes caractéristiques d'une attaque spécifique (par exemple un scan du réseau ou un scan de ports). Cette étape se rapproche de la reconstruction des sessions proposée par F. Valeur dans [Val06]. Contrairement à la fusion, cette étape peut demander une mise en relation entre messages de différents types (messages de différentes IDS).

C.3 Identification de scénarios d'attaque Cette étape regroupe les alertes correspondant à des attaques élémentaires et qui forment un enchaînement caractéristique d'un scénario d'intrusion (découverte, exploitation, élévation de privilèges). Cette fonction est au centre des travaux de la thèse.

D L'analyse d'impact et les priorités

Le but est d'analyser l'impact potentiel de chaque attaque sur le système. Les alertes (ou méta-alertes) qui correspondent à ces attaques peuvent ensuite être classées selon différents critères (criticité de l'attaque, impacts potentiels, confiance...). Cette étape suppose d'avoir à disposition une base de biens du système exprimant les dépendances entre ses constituants.

E Dimensions et caractéristiques supplémentaires

L'ordre d'enchaînement de ces différentes phases n'est pas immuable et certains auteurs proposent de placer des cycles (par exemple, l'approche de [SMFDV13] suggère l'ajout d'une boucle de rétroaction entre les modules d'agrégation liés à l'identification des scénarios et à la fusion). De même, l'approche [ZYZ08] propose par exemple de placer le module de vérification après l'étape de fusion.

De plus, comme le soulignent les auteurs de [HS14], le terme générique *corrélation* est utilisé dans la communauté pour caractériser à la fois (1) la fusion d'alarmes (liées à un IDS unique) (2) la fusion d'alarmes de plusieurs IDS de même type (3) la reconstruction d'attaque élémentaire ou encore un mélange de ces différents composants. Des étapes supplémentaires sont parfois identifiées, par exemple les étapes de réduction des faux positifs ou bien les étapes de reconnaissance d'intention ([SG06]), mais ces dernières peuvent se rapporter à une ou plusieurs des étapes identifiées ci-dessus (respectivement la vérification et l'identification de scénarios d'attaque). Tous les articles ne se préoccupent pas nécessairement de toutes les étapes et se focalisent souvent sur l'une d'entre elles en la nommant étape de corrélation.

Pour chacune des étapes préalablement identifiées, différentes approches et outils ont été proposés. Ces approches se différencient en fonction des disciplines utilisées. L'état de l'art (dans [SG06]) fait l'association entre différentes approches et algorithmes utilisés lors de chaque étape de la corrélation (intelligence artificielle, fouille de données, apprentissage automatique...). Pour permettre une vue plus synthétique, de grandes tendances sont extraites concernant les algorithmes utilisés. Ainsi, les travaux [SMFDV13] proposent une classification dans laquelle les méthodes utilisées sont réparties en trois grandes catégories : les méthodes basées sur les similarités, les méthodes séquentielles et les méthodes reposant sur des cas de référence. En plus des fonctions élémentaires du processus de corrélation et des types de techniques mises en œuvre, il est possible de faire ressortir d'autres dimensions caractéristiques. Il est par exemple possible d'effectuer une différenciation en fonction de la diversité de la source de données (collecte sur un type d'IDS unique ou avec des IDS de différents types) ou encore sur le type d'architecture (centralisée, hiérarchisée ou totalement répartie) comme proposé dans [SMFDV13].

De même, le processus de corrélation (en particulier les phases d'agrégation) nécessite également l'utilisation de fenêtres temporelles dans laquelle les événements sont sélectionnés pour être corrélés. La gestion de cette fenêtre peut se faire de différentes manières [BB13], [FAK15] : fenêtre temporelle fixe, glissante, fenêtre fixée par un nombre maximal d'événements, fenêtre glissante pondérée, fenêtre à repère.

La corrélation d'alertes peut ainsi être caractérisée dans de multiples dimensions. Les sous-sections qui suivent ont pour but de recenser les principaux travaux traitant de chacune des étapes de la corrélation d'alertes.

1.1.3 Travaux sur la normalisation

Les alertes sont générées par différents IDS utilisant des formats et des conventions différentes. L'étape de normalisation vise à homogénéiser la structure et la sémantique des alertes pour les exprimer dans un format standard [HG07]. Cette normalisation peut affecter la syntaxe des champs (normalisation syntaxique). La normalisation peut également être sémantique lorsqu'elle contrôle les valeurs et la signification des champs des notifications (dictionnaire d'équivalents entre des noms d'attaques).

Le format IDMEF [1](Intrusion Detection Message Exchange Format) propose une structure

1. http ://tools.ietf.org/html/rfc4765

normalisée pour la génération des alertes. Ce format prend en charge les aspects syntaxiques (nom des champs), mais pas l'aspect sémantique. Le format IDMEF constitue un format de référence dans une grande partie des travaux qui demandent une manipulation ou une modélisation d'alertes. En particulier, l'approche [KBN05] met en œuvre une normalisation syntaxique en convertissant dans un premier temps les valeurs des champs en chaînes de caractères, en convertissant ensuite au format IDMEF puis en effectuant une normalisation sémantique en recherchant des noms équivalents (CVE-Bugtrack) dans une base de données. De même, l'approche [LXP08] normalise également les alertes au format IDMEF et dispose d'une solution sous forme de modules de normalisation sémantique pour chaque type d'IDS afin de rendre cohérentes les valeurs des champs.

Cette phase de normalisation est explicitée dans [LWC08] avec une simplification et une réduction des alertes à 8 champs (Source, cible, port source, port cible, type d'attaque, horodatage, motif de l'attaque, poids). La normalisation sémantique est ici représentée par le champ indiquant le motif général (reconnaissance, scan de ports, intrusion) à laquelle l'attaque appartient. Cette opération se situe à la limite entre la normalisation et l'enrichissement.

La normalisation est nécessaire mais elle est souvent traitée de manière implicite dans les travaux. Les solutions utilisent ou s'inspirent généralement d'IDMEF pour définir l'ensemble des champs utiles. La normalisation sémantique est le plus souvent centrée sur la caractérisation du type d'alertes. Un dictionnaire peut être utilisé pour substituer des noms équivalents ou bien une solution de généralisation peut également être mis en œuvre afin de faire correspondre un ensemble d'identifiants d'attaques spécifiques à un IDS avec une catégorie d'attaque plus générique (scan, exploitation).

1.1.4 Travaux sur l'enrichissement

L'enrichissement consiste à ajouter des informations aux alertes pour faciliter les étapes ultérieures du processus de corrélation. Cet enrichissement peut être réalisé dynamiquement ou statiquement.

Un exemple d'approche dynamique est illustrée dans [SP03] par l'utilisation d'une correspondance entre les hôtes du réseau et leur distance en nombre de sauts pour aider les NIDS (Network-based IDS) à détecter les tentatives d'évasion. Cette correspondance est recherchée de manière dynamique sur le réseau.

Le système utilisé dans [DW01] se fonde sur une base de biens statique pour ajouter des informations manquantes aux alertes (sur les cibles, les sources et les IDS). Cette base de biens est constituée d'informations renseignées au préalable dans des fichiers statiques de configuration sur le système.

Un autre exemple d'enrichissement statique est présentée dans les travaux [SCS11] mais contrairement aux travaux cités auparavant, l'enrichissement est sémantique. Les alertes issues de l'IDS Snort sont associées à une sémantique permettant de regrouper toutes celles ayant la même signification, par exemple celles faisant référence à un scan du réseau ou à la compromission d'une machine. Il est possible d'identifier cette opération comme un mélange d'enrichissement et de normalisation sémantique.

Les travaux sur l'enrichissement ajoutent ainsi principalement des informations contextuelles sur l'état du système ou des informations pour classer les alertes par type.

1.1.5 Travaux sur la vérification

Cette étape a pour objectif de rejeter les alertes qui référencent des attaques exploitant des vulnérabilités absentes du système, ou bien de vérifier les conséquences observables d'une attaque sur le système.

[KRV04] montre que cette vérification peut être active (obtention dynamique des informations de configuration par connexion sur la machine potentiellement compromise pour détecter une conséquence de l'attaque, utilisation d'un scanner de vulnérabilité pour vérifier la présence de la vulnérabilité), ou passive (utilisant uniquement une base de biens).

Des règles de vérification peuvent également être appliquées et générées automatiquement à partir du changement éventuel d'état des entités attaquées : dans [MLB08], l'état d'un port (ouvert ou fermé) ou d'une machine sont utilisés pour inférer le succès ou l'échec d'une attaque. Il est également établi qu'une vérification exhaustive pour chaque attaque détectée demande beaucoup de ressources ([RD11]).

1.1.6 Travaux sur la fusion (agrégation)

L'étape de fusion a pour objectif de regrouper les alertes caractérisant une attaque élémentaire qui se répète, qui donne lieu à plusieurs alertes produites par un même IDS ou bien qui est détectée par plusieurs IDS de même type. Cette étape de fusion fait appel le plus souvent à une recherche de similarités entre attributs. Les attributs similaires peuvent être spécifiés ou bien déduits statistiquement.

A Regroupement par similarité

La fusion peut reposer sur un regroupement des alertes par similarité. En effet, des alertes comprenant des attributs similaires et qui sont observées dans un intervalle de temps donné tendent à avoir la même cause. Les méthodes reposant sur ces similarités mettent en œuvre des comparaisons entre les attributs des alertes en utilisant différentes métriques. Les attributs des alertes sont alors organisés de manière à permettre la définition de la distance qui sépare la valeur de deux attributs. Cette distance sert alors d'indicateur pour réaliser ou non une fusion. Les travaux de Julish ([Jul03]) en sont un exemple. La distance entre deux alertes y est définie à partir de la mesure de la distance entre différents attributs des alertes dans une hiérarchie. La différence entre deux attributs est définie comme la longueur du chemin le plus court qui relie ces deux attributs à un parent commun. Ensuite, la différence entre deux alarmes se calcule comme la somme des différences de tous leurs attributs. Cette approche suppose de disposer d'une hiérarchie permettant de classer les attributs de manière cohérente et dans laquelle la notion de distance a un sens, ce qui n'est pas toujours simple (par exemple, dans le cas du calcul de distance entre deux ports).

Une autre approche ([VS01]) consiste à regrouper les alertes similaires, où la similarité est calculée à partir de la somme des distances entre chaque attribut des alertes. L'approche suppose d'avoir au préalable défini un indice de similarité attendu entre différents types d'alarmes. Ainsi, si la similarité réelle est inférieure à un seuil fixé (le seuil peut concerner deux alertes ou bien deux champs d'alertes), les alertes ne sont pas fusionnées. Cette problématique du regroupement des alertes en fonction de leurs similarités est également traitée dans [RCM10] et [MIS12].

D'autres méthodes font évoluer cette notion de similarité. Par exemple, une méthode de fusion reposant sur la comparaison des entropies des alertes est proposée dans [GGB13]. Un autre exemple est décrit dans [SCS11]. Il s'agit d'un système de corrélation qui procède à une étape de fusion de toutes les alertes d'une fenêtre temporelle donnée visant la même cible et ayant la même portée sémantique (classification des types d'alertes : scan, exploitation...). Une approche similaire est utilisée dans [RSG10] où les alertes sont regroupées en fonction de la similarité de leur attributs dans une fenêtre de temps glissante.

B Élimination des alertes récurrentes

D'autres travaux sur la fusion se sont concentrés sur l'élimination des alertes récurrentes. Les travaux [VDM+09] proposent une méthode implicite qui utilise les séries temporelles pour éliminer les phénomènes périodiques du flux d'alertes en considérant que ces phénomènes périodiques sont issus d'un comportement normal. La fouille de données est utilisée dans [Vaa09] pour repérer les attributs des alertes revenant de manière récurrente.

Les travaux [DW01] présentent une étape de fusion permettant d'éliminer les doublons. Ces doublons peuvent être produits par deux NIDS différents qui contiennent les mêmes champs (adresse source, adresse destination, port source, port destination). De même, la partie *corrélation basique* de [SPB12] correspond à une étape de fusion (élimination des doublons, éliminations des alertes de même type dans une fenêtre temporelle).

1.1.7 Travaux sur l'identification d'attaques élémentaires (agrégation)

Cette étape vise à regrouper les alertes levées par des IDS de type différent qui caractérisent une même attaque élémentaire ou à regrouper les alertes complémentaires liées à la réalisation d'une attaque élémentaire de type scan ou déni de service. Les travaux abordant la reconstruction d'attaques élémentaires utilisent en grande partie des algorithmes d'apprentissage automatique.

L'article [SG09] présente une méthode de fouille de données permettant d'identifier les motifs qui correspondent à une attaque ciblant plusieurs machines, plusieurs attaques ciblant une machine unique et plusieurs attaques qui visent plusieurs machines. MS2IFS ([CYP10]) permet également de reconstruire des attaques élémentaires à l'aide de méthodes de partitionnement.

L'approche présentée dans [SJDM08] met en œuvre un regroupement des alertes correspondant à une attaque élémentaire. Pour cela, les auteurs utilisent plusieurs dizaines d'attributs caractéristiques des paquets IP, TCP et UDP (mais n'incluent pas les adresses IP source et destination) qui servent d'entrées et de sorties à un réseau de neurones de type auto-associateur. Lors de l'entraînement, ce réseau cherche à faire correspondre la valeur de chacune de ses sorties à la valeur de l'entrée correspondante (il dispose pour cela du même nombre d'entrées que de sorties).

Dans [CNTBB04], les auteurs utilisent trois algorithmes (addition simple d'un paramètre de sévérité, un réseau de neurones et les k plus proches voisins) pour corréler les alertes issues d'un NIDS (Snort), d'un HIDS (Samhain) et des journaux Syslog. En particulier, ils disposent d'une étape de reconnaissance des attaques de type scan de port.

Cette étape d'identification d'attaques élémentaires est centré sur la construction de méta-alertes spécifiques à de grandes classes d'attaques telles que des dénis de service ou des scans de ports.

1.1.8 Travaux sur la gestion de l'impact et des priorités

L'objectif de cette étape est de déterminer et de classer les attaques selon leurs impacts potentiels (en termes de service, de coût, de sécurité) sur le système. Emerald M-correlator [PFV02] attribue différents niveaux de criticités aux attaques possibles en fonction de leur impact potentiel sur le système. Un score d'impact est alors calculé pour chaque alerte. Toutes ces valeurs sont calculées à l'aide d'un réseau bayésien modifié. Un système à base de logique floue est utilisé dans [AASB08] pour attribuer un score aux alertes puis les prioriser. Une approche similaire est employée dans [SJ01].

Dans [PTC+14], la confiance dans les alertes générées par les IDS est prise en compte. Cette confiance traduit non seulement la tendance de certains IDS à émettre un trop grand nombre d'alertes mais également la possibilité pour un attaquant d'intercepter les alertes levées ou d'en faire émettre pour tromper la supervision (faux-positifs malveillants). Cette notion de confiance est également évoquée dans [YF07] (au niveau de la fiabilité généralement plus élevée des HIDS par rapport aux NIDS) et dans [TL07].

1.1.9 Travaux sur l'identification de scénarios d'attaque (agrégation)

Lorsqu'un scénario d'intrusion est constitué d'un enchaînement de plusieurs étapes élémentaires (reconnaissance, exploitation d'une faille, mise en place d'une porte dérobée, élévation de privilèges), il est souhaitable d'agréger les alertes (ou méta-alertes) identifiant chaque étape du scénario au sein d'une nouvelle méta-alerte.

Les méthodes utilisées pour l'identification de scénarios d'attaque peuvent être regroupées en trois grandes catégories dépendant de la manière dont le ou les scénarios d'attaque à détecter sont exprimés. Les scénarios d'attaque sont exprimés entièrement dans les méthodes explicites, partiellement dans les méthodes semi-explicites à l'aide de pré-conditions et de post-conditions et à l'aide de métriques spécifiques dans le cas des méthodes implicites.

A Méthodes implicites

Les méthodes implicites pour identifier les scénarios d'attaque font appel à des statistiques ou bien à des similarités entre différents attributs des alertes. Il est cependant mentionné dans [OMSH03] que les techniques reposant sur un apprentissage sont en général complexes à appliquer car elles demandent souvent un apprentissage avec des jeux de données comprenant un nombre suffisant de scénarios d'attaque complexes. Or ces attaques sophistiquées sont beaucoup plus rares en comparaison aux attaques de plus faible complexité.

A.1 Utilisation des similarités La référence [DC01] utilise une méthode de corrélation implicite basée sur un indice de similarité : ce dernier est calculé à partir de l'écart temporel entre alertes, de la probabilité d'enchaînement des deux types d'alertes et des valeurs des attributs des alertes. Dans le cas où l'indice est trop faible par rapport aux scénarios déjà identifiés, un nouveau scénario est créé.

A.2 Réseaux bayésiens et chaînes de Markov L'approche décrite dans [BKM08] utilise des réseaux bayésiens pour la reconnaissance de scénarios. Un plan d'attaque est défini comme un ensemble d'actions et un objectif d'intrusion. Ces actions peuvent avoir plusieurs influences sur l'objectif d'intrusion : une influence négative, positive ou critique (si cette action permet de réaliser directement l'objectif d'intrusion). Ces différentes influences d'une action A sur l'objectif O sont mesurées par la comparaison entre $P(O|A)$ et $P(O)$. Dans [TBLM11] des réseaux bayésiens sont utilisés en parallèle d'une base de connaissances représentant la topologie et les vulnérabilités du système. L'approche [RSG10] utilise également des réseaux bayésiens pour reconstruire les scénarios d'attaque à la volée. Il existe également des méthodes reposant sur des chaînes de Markov ([OMSH03], [ZGHS09], [ZY10], [FAK15]) dans lesquels le système renvoie la probabilité d'occurrence d'une attaque d'un type donné pour une séquence des alertes observées.

A.3 Combinaison de plusieurs techniques Un enchaînement de plusieurs outils d'apprentissage est proposé dans les travaux suivants :

Les auteurs de [ZG05] mettent en œuvre un réseau de neurones indiquant en sortie la probabilité avec laquelle deux alertes sont corrélées. Les entrées de ce réseau de neurones sont calculées à partir de la similarité entre deux adresses IP sources, entre deux adresses IP cibles, entre deux ports cibles ou la correspondance entre l'adresse IP source et une ancienne IP cible. L'entraînement est réalisé à partir de données synthétiques et l'évaluation (réalisée à partir des données DARPA 2000) ne précise pas le taux de faux positifs.

Les auteurs de [MMC11] proposent une méthode implicite faisant appel à un enchaînement de plusieurs techniques. Les alertes sont traitées successivement par une méthode de partitionnement hiérarchique, une carte auto-organisatrice, l'algorithme k-means puis les partitions d'alertes sont comparées au moyen d'un index de corrélation pour déterminer leur appartenance au même scénario. Les auteurs proposent simplement une évaluation de l'algorithme en termes de ressources mémoire utilisées.

Une autre méthode d'apprentissage non supervisée à base de réseaux de neurones suivie d'une méthode de partitionnement est proposée dans [SJDM08].

Dans [BBGR13] et [BBG13], les auteurs mettent en œuvre une succession de trois techniques issues des systèmes immunitaires artificiels. Il s'agit respectivement de logique floue pour simuler des règles immunitaires innées, d'une sélection de règles basées sur les similarités (simulation de la réponse adaptative de première ordre) et enfin d'un algorithme de système immunitaire artificiel modifié.

B Méthodes semi-explicites

Avec les méthodes semi-explicites, l'objectif est de corréler les conséquences d'une attaque élémentaire avec les prérequis d'autres attaques afin de reconstruire les scénarios d'intrusions probables. Le langage JIGSAW ([TL00]) constitue le point de référence des approches semi-explicites. Le langage JIGSAW permet en effet de décrire un scénario d'attaque construit autour des prérequis et des conséquences de chaque attaque élémentaire.

B.1 Systèmes de pré-conditions et de post-conditions Le langage lambda est utilisé par le système de corrélation semi-explicite décrit en [CM02]. Dans ce système, l'étape de corrélation consiste à déterminer si une alerte peut constituer un maillon d'un scénario d'attaque ou bien commencer un nouveau scénario. L'approche permet de corréler un enchaînement d'attaques (correspondance entre les post-conditions d'une attaque et les prérequis d'une autre) ou plusieurs stratégies permettant d'obtenir un même résultat (correspondance entre les post-conditions de différentes attaques). La reconnaissance d'intention a pour objectif de déterminer par transitivité des étapes non détectées dans le scénario.

L'approche de [KDP+12] utilise également le langage Lambda pour décrire les scénarios en y ajoutant une mesure de la vraisemblance du succès d'une attaque ainsi que l'état d'avancement du plan de l'attaquant. Ce plan étant représenté sous forme d'un graphe d'attaque, le processus consiste à le découper dans un premier temps en plusieurs sous graphes (correspondant à des objectifs disjoints) qui seront ensuite transformés en modèles de Markov pour lesquels la probabilité de passer d'une étape à une autre est donnée par le niveau d'expérience de l'attaquant, la cohérence avec les étapes précédentes et la difficulté de l'attaque (mesurée en évaluant les conditions à remplir pour que l'attaque soit possible). Il est également proposé un système de contre-mesures dans lequel chaque action permet d'inhiber certaines pré-conditions nécessaires à la réalisation d'une phase du scénario d'attaque.

B.2 Satisfaction partielle de pré-conditions P. Ning et al. [NCR02] présentent un système de corrélation améliorant l'approche de JIGSAW ([TL00]) qui ne permettait pas de corréler plusieurs tentatives d'une même attaque. Ce système est basé sur l'expression de prérequis et des conséquences liées à chaque attaque élémentaire. Dans cette approche, la satisfaction partielle de prérequis est autorisée, ce qui permet de corréler des alertes même si toutes les étapes normalement nécessaires à sa préparation n'ont pas été détectées. Ce système est formalisé à l'aide du concept d'hyper-alerte. Cette dernière consiste en un triplet (fait, prérequis, conséquences). Cette hyper-alerte peut être instanciée sous la forme d'un ensemble fini de tuples associés à un estampillage sous forme d'intervalle temporel. La relation *prépare à* définie entre deux hyper-alertes $H1$ et $H2$ permet d'exprimer le fait qu'une attaque de type $H1$ permet de faciliter une attaque de type $H2$ sans pour autant remplir tous les prérequis nécessaires à sa réalisation complète. Des contraintes temporelles peuvent également être exprimées afin de réaliser des agrégations significatives. Les scénarios d'attaque sont représentés par un graphe de corrélation d'hyper-alertes dont les nœuds sont des hyper-alertes et les arêtes représentent la relation *prépare à*. Cependant, cette méthode a un taux de faux positifs et de faux négatifs élevé. En effet, elle va corréler des alertes qui semblent préparer certaines attaques même si ce n'est pas réellement le cas. De plus, le système ne permet pas de corréler deux attaques liées entre lesquelles aucune relation explicite de préparation n'existe.

Cette méthode est améliorée dans [ZNIR04] et [NXHA04] par l'introduction de dépendances entre les alertes et l'état du système. La notion de preuve d'intrusion est alors définie comme un événement observé par un IDS ou par un changement d'état du système résultant des actions de l'attaquant. L'état du système est décrit à l'aide d'attributs à valeurs booléennes reliés entre eux par des relations logiques. Par exemple, le fait pour un attaquant d'avoir les privilèges administrateur implique qu'il possède également les droits de lecture dans un fichier local. Une notion de confiance dans la détection est également définie : une alerte a une probabilité donnée d'être liée à une attaque réelle, et si les prérequis de l'attaque (définis à l'aide des attributs) sont vérifiés, alors la probabilité de réussite de cette attaque est fixée à 1. L'inconsistance dans les valeurs des attributs permet d'émettre une hypothèse sur des attaques manquées par des IDS.

Une autre approche ([NRJ04]) propose elle aussi d'inférer les attaques manquées par les IDS en utilisant un graphe d'attaque, mais c'est la distance entre deux nœuds correspondant à deux alarmes qui fait office d'indicateur de corrélation. L'approche de P. Ning et al. ([NX04]) propose une extension de cette démarche en tentant d'inférer non seulement les types d'attaques manquées mais également la valeur des attributs, en faisant l'hypothèse que les attaques manquées sont des variations inconnues d'attaques répertoriées ou bien des attaques équivalentes.

Des réseaux de Petri colorés cachés sont mis en œuvre dans [YF07] et permettent de spécifier des pré-conditions et post-conditions des actions de l'attaquant et également d'associer des probabilités aux transitions entre différentes actions en fonction des alarmes observées.

Enfin, MARS [AAA+10] est une architecture de corrélation utilisant un système de pré et post-conditions couplée à une méthode statistique.

C Méthodes explicites

Les méthodes explicites reposent sur une description des scénarios d'attaque attendus. Il faut distinguer d'une part la spécification du scénario d'attaque (graphe, langage, arbre) qui constitue la règle de corrélation et la stratégie mise en œuvre pour la détection (algorithme de détection).

C.1 Langages de corrélation Un certain nombre de langages permettent d'exprimer un scénario d'attaque à détecter sous la forme d'une règle de corrélation. Cette règle est ensuite fournie au corrélateur pour réaliser la détection. La règle de corrélation est alors souvent transposée sous la forme d'un automate de reconnaissance. Les approches utilisent en général des automates pour reconnaître les scénarios et un langage de description d'attaque pour les décrire. Plus concrètement, ces langages sont les suivants :

(a) STATL [EVK00]

(b) Sutekh [PD02]

(c) CAML [CLF03]

(d) ADeLe [TVM04]

(e) OrchIDS [GLO08]

Ces différents langages font l'objet d'une section spécifique dans le chapitre 2. Cette problématique de détection explicite de scénarios d'attaque à partir d'une spécification de scénario d'attaque sous la forme d'une règle de corrélation est le point central de la thèse. Plus particulièrement, notre centre d'intérêt se focalise sur une méthode permettant de générer des règles de corrélation exprimées dans un de ces langages (à savoir : ADeLe). Cette méthode est présentée au chapitre 4.

C.2 Approches sans langages de corrélation entièrement définis Une spécification d'un scénario d'attaque sous la forme d'un arbre d'attaque amélioré est proposé dans les travaux [cY07] Il s'agit d'un arbre d'attaque disposant d'un opérateur ET séquentiel et qui permet de définir un automate spécifique modélisant la progression de l'attaquant. Cependant, ces travaux se centrent surtout sur la formalisation d'un automate de détection à partir d'un arbre sans réellement se soucier de la sémantique des actions élémentaires définies dans les feuilles de l'arbre, des capacités des outils de détection (c'est à la charge du créateur de l'arbre de vérifier que les événements spécifiés peuvent effectivement être générés par le système) ni de l'environnement d'exécution.

Dans [RC07], quatre motifs statiques définissent des scénarios d'attaque : LinkToSession (attaque par force brute suivie d'une réussite d'authentification), LinkBackToSession (authentification suivie d'un déni de service), Island-Hoping (la machine attaquée devient la source de la nouvelle attaque) et Recon-Breaking-Escalate (séquence d'une phase de reconnaissance suivie d'une phase d'intrusion puis d'une élévation de privilèges). Cette approche définit donc seulement quatre scénarios d'attaque.

D'autres approches peuvent également être classifiées dans cette catégorie. Par exemple, les travaux [QL04b] et [QL04a] introduisent un réseau bayésien permettant de corréler les attaques à partir de la connaissance de la probabilité d'enchaînement des alertes référençant des types d'attaques déterminées. Des grammaires non contextuelles sont utilisées pour générer des scénarios d'attaque ([AMZ09]). Une méthode d'extraction de séquences d'alertes est utilisée dans [LLWL07].

C.3 Utilisation de graphes Les graphes d'attaques peuvent être utilisés comme une structure pour spécifier les enchaînements possibles d'attaques à détecter. Cependant, les travaux se concentrent essentiellement sur les algorithmes de détection permettant d'exploiter efficacement le graphe. Par exemple, les travaux [WLJ05] et [WLJ06] utilisent un graphe d'attaque pour définir le scénario d'attaque à détecter. Le graphe utilisé comprend deux types de nœuds : les exploits et les pré-conditions et post-conditions liées à ces exploits. Au lieu de comparer une alerte à toutes les précédentes qui satisfont la relation *prépare à* d'une même fenêtre temporelle pour la reconstruction du scénario d'attaque, l'introduction d'une structure nommée *Queue Graph* permet d'augmenter

les performances de l'algorithme de reconnaissance. Cela consiste à conserver de manière implicite les alertes correspondant à l'exploitation d'un même exploit. Si les estampilles temporelles de deux alertes sont inférieures à une valeur seuil, ces deux alertes sont traitées comme étant concurrentes : dans ce cas, l'ordre conservé est toujours celui permettant de réaliser une corrélation avec les précédentes alertes. Cette approche est destinée à permettre de corréler des alertes espacées d'intervalles de temps importants.

L'algorithme de corrélation utilisé par [RCM11] utilise également des graphes d'attaques et tente d'améliorer les performances de l'approche [WLJ05]. Dans cette approche, il est proposé de faire correspondre des alertes aux nœuds du graphe d'attaque (à partir des attributs sources, cibles et du type d'alerte), d'agréger les alertes similaires créées dans un laps de temps inférieur à un seuil donné, de construire un graphe de dépendance d'alertes et d'utiliser l'algorithme de Floyd-Warschall pour identifier des sous-ensembles d'alertes appartenant à un même scénario.

Les graphes d'attaques sont utilisés également dans [AJA11] et permettent de corréler des alertes même lorsque certaines attaques élémentaires du scénario n'ont pas été détectées. Ceci est réalisé en faisant des hypothèses sur des attaques manquées dont les pré-conditions sont remplies par la réalisation d'attaques déjà détectées et dont les conséquences satisfont les prérequis d'attaques également détectées. Les auteurs de [LWC08] proposent la construction d'un APG (attack path graph) basé sur un motif récurrent (exploration, scan, intrusion, objectif), puis une phase de génération du scénario à partir de ce graphe de chemins d'attaques.

C.4 Conclusion sur les méthodes explicites En conclusion, les travaux traitant de la reconnaissance de scénarios avec des méthodes explicites se concentrent largement sur les algorithmes de détection permettant d'obtenir de meilleures performances et de prendre en compte de possibles erreurs dans la détection sous-jacente (alertes manquantes dans une séquence d'action par exemple). Dans la plupart des cas, la création des règles de corrélation est considérée comme un acquis. Par conséquent, peu de travaux s'intéressent à la manière d'obtenir des règles de corrélation. Un autre problème important est lié au niveau d'abstraction de l'expression des scénarios d'attaque. Pour permettre la détection, les notifications attendues doivent être spécifiées. Ceci mène à deux cas : dans le premier, les règles de corrélation font directement intervenir les caractéristiques des notifications attendues. Ces règles sont donc très dépendantes du système et des IDS. Dans le second cas, c'est un scénario d'attaque faisant intervenir des actions de l'attaquant qui est décrit (par exemple dans le cas d'utilisation des graphes d'attaques comme règle de corrélation). Le niveau d'abstraction y est ainsi plus élevé, et il faut alors réaliser le lien entre les notifications possibles et les actions de l'attaquant. Cette étape est souvent omise ou simplifiée (en considérant par exemple uniquement des NIDS).

1.1.10 Discussion

Cette section a présenté le fonctionnement et l'organisation générale de la corrélation d'alertes. Les travaux existants se focalisent généralement sur l'étape de fusion ou de reconnaissance des scénarios d'attaque. En particulier, différentes approches dans le domaine de la reconnaissance de scénarios d'attaque utilisent des méthodes explicites. Les travaux se concentrent cependant sur la performance des algorithmes de détection. La création de règles de corrélation semble être un point qui n'est pas abordé dans la littérature.

Cependant, la création de règles de corrélation demande de disposer d'un ensemble d'informations sur le système. La section suivante propose une revue des différents éléments nécessaires à l'élaboration d'une base de connaissances permettant de servir de support à la création de règles de corrélation.

1.2 Base de connaissances pour la corrélation d'alertes

La section précédente a évoqué le fait que le processus de corrélation nécessite, en plus des événements et des alertes à corréler, des informations contextuelles sur le système surveillé (que ce soit pour la vérification, les priorités ou l'agrégation). Ces informations sont fournies par une base de connaissances contenant les informations pertinentes pour la corrélation. Pour servir de

support au processus de corrélation, cette base doit inclure non seulement les éléments du système potentiellement vulnérable et les relations entre ces mêmes éléments, mais également une modélisation du comportement des IDS présents. Ce comportement inclut les capacités de détection des observateurs. Dans le cas des IDS, il est intéressant de pouvoir identifier les classes d'attaques effectivement détectables, ce qui demande de disposer d'une classification cohérente et efficace des différents types d'attaques. Une telle taxinomie peut être centrée sur l'attaquant, sur la cible, sur le type d'action réalisée ou encore sur les vulnérabilités.

Cette section présente les différents domaines couverts par les bases de connaissances (spécifiques à la corrélation d'alertes ou non).

1.2.1 Topologie et cartographie

La topologie représente la position et les relations entre les nœuds du système alors que la cartographie est l'ensemble des logiciels, processus et services associés à chaque nœud [Mor04]. Ces éléments sont primordiaux pour la corrélation car ils permettent de connaître les interconnexions entre les machines, les services exposés et les identités des machines. Ces éléments constituent les éléments minimaux d'une base de connaissances. Ces éléments sont présents dans [GHH+01] (services locaux, services réseaux, OS), [PFV02], [GMHD12]. Ces mêmes éléments se retrouvent également dans les travaux traitant des graphes d'attaques ([JSW02], [SHJ+02]). Certaines approches introduisent également une modélisation des utilisateurs et des comptes utilisateurs ([Lag10], [SEH13]).

Les éléments qui différencient les approches sont le niveau de détail de la description ainsi que les moyens mis en œuvre pour le peuplement de cette base de biens (manuelle, semi-automatique). Des logiciels dédiés placés sur chaque nœud peuvent fournir des informations sur les configurations et la topologie ou bien des méthodes heuristiques (actives ou passives) permettent de déduire ces mêmes informations. TRINETR [YRS+04] est un exemple d'approche qui met en œuvre ce peuplement de la base à l'aide d'agents.

Une autre caractéristique permettant de différencier les principales approches est le formalisme utilisé pour la description de la base de connaissances (langage descriptif, relations, expressions logiques). Le système M2D2 ([MMDD02] et [Mor04]) permet de regrouper les informations pertinentes sur le système à protéger pour l'enrichissement des alertes. Chaque produit est associé à un nom, un type et une version. M4D4 [MMDD09] reformule les concepts de M2D2 dans une logique de premier ordre (ceci permet de définir par exemple des règles récursives) et permet de décrire la topologie réseau en termes de routes, sous-réseaux, hôtes, résolution de noms et de la cartographie qui lie les nœuds aux logiciels (nom, version, architecture...). Le modèle permet d'exprimer l'exécution d'un processus par un utilisateur, l'écoute d'un service sur un port. Une logique de description nommée IDDL pour *Intrusion Detection Description language* est introduite dans [TBLM11] pour la représentation des connaissances utiles au processus de corrélation. Le contenu est inspiré de M4D4 pour la modélisation de la topologie et de la configuration logicielle des nœuds.

Topologie	nœuds, sous-réseaux, routage
Cartographie	services réseau, logiciels, systèmes d'exploitation, versions des logiciels, utilisateurs

TABLE 1.1 – Composants de la topologie et de la cartographie.

Le tableau 1.1 reprend les caractéristiques qui nous semblent essentielles pour décrire la topologie et la cartographie d'un système.

Les éléments précédents constituent les dimensions minimales permettant une première représentation d'un système. Cependant, des éléments supplémentaires sont indispensables pour compléter et affiner la modélisation. En particulier, les regroupements de machines, l'accessibilité et les règles de filtrage sont des éléments importants permettant de prendre en compte les communications possibles entre machines.

Certains travaux introduisent également des regroupements de machines qui disposent de fonctionnalités communes ou bien qui disposent de connectivités identiques. Cela peut permettre une

représentation plus compacte de l'environnement et de limiter la complexité de la base de connaissances. Par exemple, le système de corrélation présenté en [PTC+14] utilise une base de connaissances qui divise le système en domaines (correspondant à des segments de réseau), chaque domaine imposant des contraintes dépendantes des services disponibles. Ces contraintes correspondent à l'ensemble des classes d'attaques susceptibles d'être utilisées contre les services présents. Un impact en cas de compromission est également associé à chaque contrainte.

Les regroupements de machines peuvent également être modélisés pour faciliter la résolution de l'accessibilité entre différents nœuds du réseau. Par exemple, la représentation de l'accessibilité entre machines dans NetSPA [LIS+05] est réalisée à l'aide de domaines d'accessibilité consistant à rassembler les machines qui ont une matrice d'accessibilité similaire. Les problèmes de connectivités liés au NAT sont résolus par l'introduction de pseudo-machines en amont ou en aval des pare-feux. L'approche de [Lag10] utilise une base de biens permettant de représenter une topologie des réseaux sous forme hiérarchique. La modélisation des zones logiques et physiques présente dans le langage CySeMoL [SEH13] peut également être incluse dans cette catégorie.

Cette notion d'accessibilité est importante pour déterminer les flux d'informations possibles lors d'une attaque. Cette accessibilité dépend non seulement de la topologie du système mais également des équipements de filtrage comme les pare-feux qui peuvent mettre en œuvre des règles de complexité arbitraire. Les modèles les plus simples représentent les adresses et ports bloqués ou autorisés. Par exemple, dans M4D4, le prédicat $deny(F, S, SP, T, TG)$ exprime le blocage par le pare-feu F des connexions entrantes entre le port SP de l'adresse source S et le port TP du nœud T. Dans [BCTS14], une règle de filtrage qui modélise également le protocole utilisé s'exprime sous la forme *(<source, destination, port, protocole>)*. À côté de ce modèle simple de représentation des règles de filtrage, la modélisation de FIREMAN ([YCM+06]) permet de représenter certaines règles ou chaînes de règles de filtrage complexes qui sont mises en place dans des pare-feux. Ces règles sont utiles pour permettre d'établir les communications possibles entre différents nœuds du système. Une variante de ce système est utilisée dans [ICL+09] et permet également de modéliser la translation d'adresses pour calculer l'accessibilité entre machines.

Des standards fournissent également des primitives permettant de modéliser certaines parties d'un système d'information. CIM [2] (Common information model) propose un méta-modèle permettant de modéliser les éléments administrables d'un SI de manière unifiée. CybOX [3] permet également de modéliser des éléments d'un système d'information avec une granularité très fine.

Réseaux hiérarchiques	Regroupements de machines	Règles de filtrage
[Lag10]	[PTC+14]	[YCM+06]
	[LIS+05]	[BCTS14]
		[MMDD09]

TABLE 1.2 – Caractéristiques avancées de la topologie et de la cartographie.

Le tableau 1.2 reprend quelques caractéristiques avancées intéressantes pour la modélisation : des réseaux hiérarchiques, qui permettent de représenter une inclusion de sous-réseaux les uns dans les autres, l'introduction de zones regroupant des machines équivalentes et une utilisation de règles de filtrage.

1.2.2 Base de vulnérabilités

La connaissance des vulnérabilités qui affectent le système permet de prendre en compte les attaques les exploitant. Lorsqu'une attaque exploite une vulnérabilité (ce qui n'est pas toujours le cas, par exemple dans le cas d'attaques par force brute ou de certaines violations de la politique de sécurité), certaines caractéristiques de l'attaque dépendent de la vulnérabilité elle-même (par exemple, si une vulnérabilité permet d'obtenir des droits particuliers). M2D2 comprend une partie dédiée aux vulnérabilités exploitées et leurs identifiants. Une relation d'équivalence entre les vulnérabilités ainsi qu'une relation entre les vulnérabilités et des configurations vulnérables sont définies. Dans M4D4, les vulnérabilités sont liées à des configurations avec la relation $affects$ et

2. http ://www.dmtf.org/standards/cim/
3. http ://cybox.mitre.org

il est possible d'affecter à ces vulnérabilités un niveau de sévérité, un niveau de privilège minimal requis pour leur exploitation, les conséquences d'une exploitation réussie, les liens entre les références de la vulnérabilité dans différentes conventions de nommage. Dans [Zar14], les vulnérabilités sont simplement liées à des vecteurs d'exploitation (Hameçonnage, Drive-by-Download, mail, ingénierie sociale, média amovible). Le choix de cette liste de vecteurs n'est pas justifié et peut être critiqué : par exemple, le phishing est un cas particulier d'ingénierie sociale. Les approches utilisant des graphes d'attaques disposent généralement d'informations sur les vulnérabilités présentes ([JSW02]). Les vulnérabilités affectant les nœuds du système peuvent être connues ou suspectées avec une certaine probabilité ([BCTS14]).

L'état de l'art [JWCY09] répertorie les travaux de classification des vulnérabilités réseau en comparant le nombre de dimensions utilisées dans chaque classification, le domaine d'application (orienté attaque, système d'exploitation, générique, réseau sans fils) et les principaux types d'attributs utilisés au sein de chaque classification. La NVD [4] et OSVDB [5] caractérisent les vulnérabilités par différents attributs permettant de les classer. Ces attributs sont constitués du type d'accès permettant de réaliser l'exploitation (local, distant), l'impact en termes d'intégrité, de confidentialité ou de disponibilité ainsi que des scores liés à la facilité d'exploitation et à l'impact global. VULDA est une base de vulnérabilités présentée dans [AD99]. Chaque vulnérabilité comprend une description de la faute associée, une description de l'attaque nécessaire pour l'activer, les types de systèmes et protocoles vulnérables. La description de chaque faute comprend sa difficulté de compréhension, la manière dont elle a été introduite dans le système (par exemple lors de la conception, de l'implémentation ou de la configuration), sa localisation et la manière de vérifier sa présence. Une attaque est définie par sa difficulté de réalisation, la disponibilité d'exploits potentiels, le niveau d'exploitabilité (distant, local), les privilèges requis, le principe d'attaque, les moyens de détection, la latence et les impacts directs ou indirects.

Configurations vulnérables	Prérequis	Conséquences
version de logiciels	accès (local/disant)	exécution de code
	droits administrateur	élévation de privilège
		déni de service

TABLE 1.3 – Trois dimensions caractérisant une vulnérabilité.

Le tableau 1.3 résume les propriétés essentielles des vulnérabilités d'un point de vue de la corrélation. Il s'agit de la configuration logicielle vulnérable, des prérequis pour l'attaquant en termes de privilèges et des conséquences d'une exploitation réussie pour l'attaquant.

1.2.3 Déploiement et caractérisation des IDS

Un premier niveau de caractérisation consiste à différencier les IDS selon leur nature : IDS applicatif, IDS hôte (HIDS) et IDS réseau (NIDS). Cette caractérisation sommaire est utilisée par exemple dans les travaux [GMHD12], [Zar14] et [SZFL13]. La modélisation peut également être plus précise. Il est possible de différencier la visibilité topologique d'un IDS, définie comme la vision physique possible étant donné son positionnement dans le système, et la visibilité opérationnelle qui dépend de la configuration interne de l'IDS définissant les informations à surveiller et les événements réellement observables. Ces notions de visibilités topologiques et opérationnelles sont introduites dans M2D2 puis reprises et utilisées dans M4D4.

Cette séparation entre visibilité topologique se retrouve également dans d'autres travaux. Dans [PTC+14], chaque IDS est affectée à un domaine (qui correspond à la visibilité topologique) ainsi qu'à une capacité de détection (couche transport, couche application, protocole HTTP). Cette dernière capacité peut à la fois être considérée comme une visibilité topologique (l'IDS surveille une localisation particulière dans la couche réseau) ou opérationnelle (car cette capacité de détection ne dépend pas de l'emplacement de l'IDS dans le système, mais de l'utilisation de certains protocoles réseaux). De plus, chaque IDS dispose d'une réputation qui caractérise la fiabilité et la pertinence des alertes levées. Les travaux [JSW02] et [SHJ+02] associent aux IDS une visibilité qui est fonction

4. http ://nvd.nist.gov
5. http ://osvdb.org

de trois paramètres : la machine source, la machine destination et le type d'attaque. Il s'agit donc à la fois de visibilité topologique et opérationnelle. La fonction de visibilité indique si une attaque d'un certain type entre les deux machines concernées est détectable, non détectable ou bien détectable sous certaines conditions.

Le modèle exposé en [Ale04] est conçu dans une optique d'évaluation et de comparaison des performances de différents types d'IDS. Les IDS sont modélisés via leurs visibilités topologiques qui s'expriment à un niveau de granularité très fin. Ainsi, un IDS peut voir des éléments à trois niveaux principaux (réseau, utilisateur, hôte). Le niveau réseau se divise en 6 catégories (qui correspondent aux couches TCP/IP) et le niveau hôte est divisé entre firmware, noyaux, modules noyaux, appels systèmes, système de fichier, signaux inter-processus, environnement et processus.

Nature (HIDS/NIDS)	Visibilité topologique	Visibilité opérationnelle
[GMHD12]	M4D4	M4D4
[Zar14]	[Ale04]	[PTC+14]
M4D4	[JSW02]	[JSW02]
	[SHJ+02]	[SHJ+02]
	[PTC+14]	

Table 1.4 – Dimensions permettant de caractériser les IDS.

Le tableau 1.4 résume les différentes dimensions permettant de modéliser les IDS. Il s'agit tout d'abord de son type (IDS hôte, IDS applicatif, IDS réseau), de sa visibilité topologique et de sa visibilité opérationnelle. La principale différence entre toutes ces approches est le niveau de détail apporté pour définir la visibilité topologique ou opérationnelle.

1.2.4 Synthèse sur les bases de biens

Les bases de connaissances présentées sont toutes conçues pour répondre à une problématique spécifique qui peut être la vérification de la pertinence des alertes en fonction des services installés dans le cas de la corrélation d'alertes ou bien les chemins d'attaques possibles sur un système dans le cas de l'analyse de risques. Outre le formalisme utilisé, ce qui différencie principalement ces bases de biens est le niveau de détail disponible pour décrire le système, la modélisation des IDS et la prise en compte d'éléments topologiques complexes (routage, pare-feu avancé, NAT). Les bases de biens présentées ici ne sont de plus pas conçues spécifiquement pour répondre à notre objectif (la création de règle de corrélations). Dans notre optique de génération de règles de corrélation, il est par exemple indispensable de disposer d'une modélisation des capacités de détection des IDS ainsi que du lien entre les actions de l'attaquant et les messages issus de la détection de ces actions par les IDS déployés. Il est donc nécessaire de construire ou d'adapter une base de connaissances à partir des travaux disponibles. Cette base de connaissances fait l'objet du chapitre 3.

1.3 Attaques et événements

Deux éléments distincts sont à différencier en ce qui concerne la modélisation des attaques et des événements. Il s'agit d'une part des actions possibles sur un système (malveillantes ou non) et, d'autre part, la manifestation de ces actions sous forme de messages générés par des observateurs. Cette sous-section se concentre sur la manière de représenter et d'organiser les différents types d'attaques ou d'actions légitimes qui peuvent survenir sur un système d'information. Cette représentation joue un rôle central dans la modélisation de la visibilité opérationnelle des observateurs.

1.3.1 Événements, messages et alertes

Ces éléments sont les manifestations d'actions réalisées sur un système. Dans M4D4, les messages sont différenciés des événements bruts. Un message est défini comme étant une interprétation d'un événement brut par un outil d'analyse et une alerte est un message généré par un IDS lorsqu'une attaque est détectée à partir d'informations contenues dans les événements analysés. Dans

M4D4, les événements bruts peuvent être représentés sous une forme hiérarchique : par exemple, la représentation d'un paquet IP peut inclure la représentation d'une trame TCP encapsulée.

Des travaux s'intéressent à établir un lien entre actions de l'attaquant et alertes. Par exemple, les travaux de [LT09] proposent une taxinomie permettant d'associer des actions (attaques spécifiques ou exploitations de vulnérabilités) aux alertes observées en se basant sur les identifiants CVE.

1.3.2 Classification des attaques

Nous présentons ici les méthodes de classification des attaques. Les classes d'attaques définies sont importantes pour la reconstruction de scénario d'attaques. Une alerte est ainsi la manifestation d'une attaque qui appartient à une catégorie identifiée.

Howard propose dans ([How97]) les caractéristiques d'une bonne taxinomie. Celle-ci doit contenir des catégories à la fois mutuellement exclusives, exhaustives et sans ambiguïtés. L'article [IW08] présente également un état de l'art général sur les taxinomies des attaques. Les auteurs concluent qu'une taxinomie efficace doit être spécifique à un système donné et hiérarchique. Les dimensions doivent prendre en compte l'impact de l'attaque, les types d'attaques spécifiques au système, les cibles de l'attaque ainsi que les origines de la vulnérabilité.

Ces différentes catégorisations sont présentées dans les paragraphes qui suivent.

A Classification en fonction du type de violation

Cette première catégorisation s'applique à un niveau d'abstraction élevé et consiste à différencier les attaques selon leur nature : exploitation de vulnérabilité, action violant une politique de sécurité, action anormale... Par exemple, la base de connaissances Intrusion Reference Model du système SCYLLARUS ([GHH+01]) dispose d'un dictionnaire qui sépare les événements normaux des événements d'attaque. Les événements sont représentés sous forme arborescente avec héritage multiple. Un modèle similaire est utilisé dans DIDS (Distributed Intrusion Detection System) [SBD+91]. Il s'agit d'une fonction de classification des menaces en trois classes : les attaques (définies comme une modification de l'état d'une machine), les violations de la politiques de sécurité ne modifiant pas l'état d'une machine et les actions suspectes (par exemple un scan de ports). La taxinomie VERIS [6] de Verizon propose un équilibre entre la facilité d'utilisation et le niveau des détails disponibles pour décrire des incidents de sécurités. Un incident de sécurité est constitué de quatre éléments (acteurs, actions, biens, attributs). Les acteurs sont ceux qui effectuent des actions sur les biens. Les attributs caractérisent les conséquences des actions sur les biens (conséquences sur l'intégrité ou la disponibilité). VERIS permet à la fois de décrire des exploitations de vulnérabilités logicielles, des violations de la politique de sécurité, des attaques physiques, de l'ingénierie sociale ou des accidents.

B Classification en fonction de la technique utilisée par l'attaquant

Cette partie représente le cœur de la classification car c'est elle qui caractérise et différencie la manière dont l'attaque se déroule. Certains travaux proposent une simple liste de classes d'attaque et d'autres tentent de les hiérarchiser de différentes manières.

Des travaux utilisent des énumérations de classes d'attaques en fonction de leurs besoins. C'est par exemple le cas d'Emerald M-correlator ([PFV02]) qui propose une liste des types d'attaque (Déni de service, reconnaissance, élévation de privilèges, corruption du système...). Dans [SH04], le vecteur de l'attaque peut être un ensemble hétérogène mélangeant la technique (dépassement mémoire), le vecteur (virus, ver) et la conséquence (déni de service). Le système de corrélation [AMZ09] utilise les classes d'attaques suivantes : énumération, scan, compromission de service, obtention d'accès utilisateur, accès à des données sensibles. Ces différentes classes sont présentées à plat et manquent de généralisation (par exemple, le scan et l'énumération pourraient être représentées comme deux sous-attaques d'une même classe plus générique appelée *reconnaissance*). Dans la taxinomie VERIS, il est fait une distinction entre 7 types d'actions (*malware, hacking,*

6. http ://veriscommunity.net/

social, misuse, physical, error, environmental actions). Chaque groupe est ensuite raffiné en une énumération.

Il existe ainsi des sous-ensembles de techniques communes aux différentes listes mises en œuvre. Cependant, le point critiquable est le manque de transparence concernant le choix et la couverture des techniques référencées.

Un autre ensemble d'approches classifient les techniques d'attaque de manière plus structurée. L'article [LJ97] met en avant l'importance de la séparation de la technique d'attaque d'une part et du résultat de cette dernière d'autre part en proposant une taxinomie à deux dimensions. Les techniques d'attaques sont regroupées en 9 classes reflétant les grands types génériques de technique utilisées par l'attaquant. Ces techniques regroupent les attaques physiques, les attaques au niveau matériel, les usurpations d'identités, le contournement des contrôles, l'utilisation malveillante de ressources, attaques indirectes... L'approche [Lou01] reprend la taxinomie précédente pour l'appliquer aux réseaux sans fils. Dans [WOL11], les techniques se subdivisent en 7 catégories (infection, reconnaissance, triche (qui inclut un ensemble hétérogène allant de l'usurpation d'identité à l'injection de paramètres), les attaques par brute force, les attaques distribuées et la classe *autre*). Le problème de ces types de classification réside dans la trop grande généricité et la difficulté pour lier des attaques réelles aux classes d'attaques proposées.

Alors que les travaux précédents prennent le point de vue de l'attaquant pour classifier les techniques d'attaques, des travaux se positionnent en proposant une classification du point de vue du défenseur. La taxinomie introduite dans [KMT04] est centrée sur la défense dans le but de regrouper les attaques pour lesquelles des stratégies de défenses similaires peuvent être mises en place. Il y est proposé une classification des attaques en trois catégories : présence de symboles étrangers ; présence de séquences étrangères ; présence de séquences dormantes. Cette taxinomie est ainsi plutôt destinée à caractériser des méthodes et des stratégies de détection pour des IDS. La taxinomie à 5 dimensions proposée dans [GKAD07] met en avant une autre vision en caractérisant les attaques selon le type de vulnérabilité exploitée (vulnérabilité liée à la configuration, à l'implémentation ou à la spécification). [Ken99] présente également une différentiation entre les méthodes (exploitation d'une mauvaise implémentation, d'un défaut de conception, d'une mauvaise configuration). Le principal intérêt de ce type de classification est d'identifier les faiblesses potentielles lors de la création d'un système.

CAPEC La classification CAPEC [7] répertorie un grand nombre de techniques d'attaques. Ces attaques sont classifiées sous forme arborescente : une racine représente une catégorie générique d'attaque (ex : injection) qui est ensuite raffinée en nœuds intermédiaires (appelés *méta attack pattern*) puis en feuille (*standard attack pattern* ou *detailed standard pattern* selon le niveau de précision de la description). Chaque attaque dispose d'une description, d'un niveau de sévérité, des étapes nécessaires à sa réalisation (chaque étape contient une description, un résultat et les contrôles permettant de la prévenir ou de la détecter), des prérequis nécessaires pour l'exploitation, des exemples d'instances, des ressources nécessaires, des indices permettant de détecter l'attaque et des solutions pour la prévenir. Le point fort de cette taxinomie est la présence d'un grand nombre de classes d'attaques (463 en octobre 2015) ainsi que son caractère extensible. Par contre, les hiérarchies ne sont pas toujours intuitives et sont organisées en fonction de la technique d'attaque (et non de la technique de détection). Certaines distinctions sont intéressantes du point de vue théorique mais peuvent rendre le choix d'une classe spécifique difficile. Par exemple, une distinction est faite entre l'inclusion de code et l'injection de code (l'inclusion de code implique l'existence préalable de code de référence, ce qui n'est pas le cas de l'injection de code). La figure 1.2 illustre le fonctionnement de l'organisation hiérarchique de CAPEC (que l'on a ici centré sur le mécanisme d'attaque *injection*).

C Classification des privilèges requis par l'attaquant

Les prérequis à satisfaire par l'attaquant pour réaliser une attaque donnée représentent une dimension importante permettant de déterminer si une attaque est possible ou non.

7. http ://capec.mitre.org

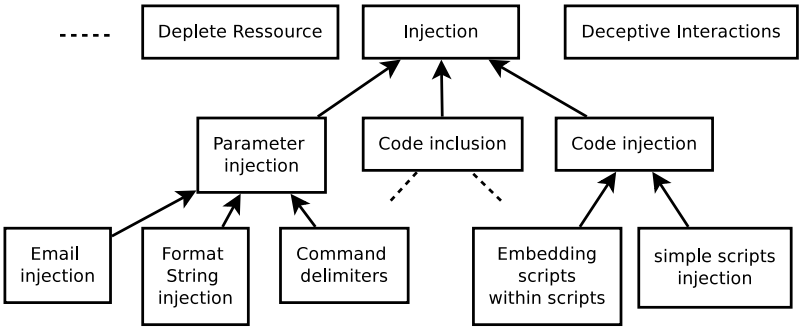


FIGURE 1.2 – Extrait de CAPEC montrant une partie de la hiérarchie des mécanismes d'attaque.

Une première série de travaux divisent ces privilèges en deux ou trois classes. Dans [WOL11], ces privilèges sont représentés par une localisation consistant en deux sous-catégories (locale, distante). Cette notion de privilège ou position locale ou distante se retrouve dans la taxinomie [GKAD07]. [How97] différencie également les privilèges distants et locaux et également les utilisateurs privilégiés (simple utilisateur, root).

Un découpage plus complexe est entrepris dans [Ken99]. Les niveaux de privilèges requis sont alors divisés en cinq catégories : accès depuis internet, accès direct au réseau local, accès utilisateur, accès administrateur et accès physique. Le tableau 1.5 résume les choix effectués dans chaque cas.

local et distant	différents droits utilisateurs	accès physique
[WOL11]	[How97]	[Ken99]
[GKAD07]	[Ken99]	
[How97]		

TABLE 1.5 – Types de privilèges définis dans les travaux.

D Classification des conséquences de l'attaque

Généralement, les conséquences d'une attaque sont un impact sur l'intégrité, la confidentialité ou la disponibilité. Par exemple, dans [LJ97] ou [Ken99] les résultats d'une attaque sont une atteinte à la confidentialité, à la disponibilité ou à l'intégrité. L'une des quatre dimensions exposées dans [SH04] est la charge de l'attaque qui correspond aux conséquences (attaque sur l'intégrité, la confidentialité, les droits d'accès). Dans ONTIDS, les objectifs d'une attaque sont un déni de service, du SPAM, une élévation de privilèges, une exécution de code ou un vol d'informations.

Les travaux raffinent généralement ce premier niveau de description pour obtenir un niveau de détail plus fin. Par exemple, une notion de privilèges gagnés est introduite dans la classification à 5 dimensions proposée dans [GKAD07]. Elle comprend une branche consacrée à la conséquence d'une attaque en termes de privilèges obtenus. Ces derniers peuvent prendre les valeurs {None, user, root, system}. Dans [WOL11], le niveau de granularité est beaucoup plus fin. Les conséquences peuvent être une fuite d'information à différents niveaux (fichier, utilisateurs, machines, réseau...), une escalade de privilèges (trois niveaux sont identifiées : absence de privilège vers un privilège anonyme, utilisateur ou root, anonyme vers utilisateur ou root et utilisateur vers root) et une nuisance correspondant à une atteinte à l'intégrité ou à la disponibilité.

Les conséquences d'une attaque sont également abordées dans la classification des vulnérabilités (en trois axes) issue du système d'analyse de risques Haruspex et GVscan ([BCTG13], [BCTS14]). Les trois axes classifient les vulnérabilités selon les conséquences possibles des attaques qui les exploitent en termes de privilèges sur un nœud (contrôle total, contrôle total en utilisant des informations supplémentaires ou contrôle partiel).

E Classification des cibles des attaques

La cible d'une attaque peut être spécifiée avec une précision adaptée. Un premier niveau est par exemple défini dans [SH04] : la cible peut être une machine, un système d'exploitation ou une application. Le niveau de détail augmente légèrement dans [WOL11] : les cibles potentielles incluent alors les fichiers, les utilisateurs, les machines, les réseaux ou les services.

Une granularité beaucoup plus fine est proposée dans [Ale04]. Les paramètres mis en jeu sont classés en attributs statiques et dynamiques. Les attributs statiques caractérisent l'objet affecté (mémoire, processeur, fichier, processus...) et l'objet interface permettant d'accéder à l'objet affecté (connexion, appel système, variable d'environnement...). Les variables dynamiques servent à caractériser les moyens de communication (canal unidirectionnel ou bidirectionnel), le type de méthode invoquée sur l'objet (création, suppression, modification, lecture...) et les attributs utilisés pour la détection (donnée spécifique, répétition d'une activité, origine...). L'approche [GKAD07] est à mettre sur le même plan. Il y est référencé l'élément porteur de l'attaque qui peut être une couche du réseau dans le modèle TCP/IP ou bien une action locale (appel système, environnement, commande, communication entre sockets) et la cible de l'attaque (la mémoire, le système d'exploitation, la pile réseau, un objet du système de fichier ou un processus).

1.3.3 Synthèse

L'importance d'une classification des attaques dans la base de connaissances a été mise en avant. Les taxinomies des attaques sont toutes construites pour répondre à un objectif précis. Cet objectif peut être la compréhension des vulnérabilités exploitées par l'attaquant, une classification des attaques selon les prérequis à satisfaire pour l'attaquant ou une classification basée sur la méthode de détection permettant d'observer l'attaque. Dans notre cas, il est déjà possible d'éliminer les dimensions qui ne nous intéressent pas pour réaliser des règles de corrélation. Par exemple, connaître la nature d'une vulnérabilité (faille de spécification, d'implémentation ou de configuration) n'est pas indispensable : cette information est importante lors de l'implémentation d'un système ou à titre didactique, mais ne donne pas d'indication sur les conséquences de l'attaque ni sur la manière de l'observer. Les privilèges de l'attaquant (local / distant) et la cible de l'attaque (fichier, service, machine, processus...) sont importantes pour déterminer le caractère détectable de l'attaque. Cependant, dans notre cas, ces informations seront contenues dans un langage dédié (voir chapitre 4).

Une autre dimension qui nous intéresse est la classe d'attaque utilisée par l'attaquant. L'identification de cette classe d'attaque a pour but de lier l'action malveillante et les notifications levées par les IDS. Ces classes d'attaque sont bien représentées par CAPEC du fait de son nombre d'attaques élémentaires représentées et de sa structure hiérarchique. Notre utilisation de CAPEC est expliquée au chapitre 4 et son intégration dans la base de connaissances est abordée au chapitre 3.

1.4 Conclusion

Ce chapitre a répertorié les caractérisations possibles du processus de corrélation d'alertes et recensé les principales techniques et stratégies adoptées pour traiter les différentes étapes de ce processus. Une grande partie des travaux se concentre sur les phases de fusion et de reconnaissance de scénarios en se basant sur des méthodes pouvant être explicites, implicites ou semi-explicites. Dans la suite, nous nous concentrons exclusivement sur la reconnaissance de scénarios d'attaque utilisant une spécification du scénario d'attaque à détecter (méthode explicite).

La seconde partie donne un aperçu des formalismes utilisés pour modéliser un environnement d'exécution qui peuvent servir de support à la corrélation d'alertes. Les éléments de modélisation récurrents (topologie, vulnérabilités) sont ainsi identifiés. Comme aucune base de connaissances n'est spécifiquement conçue pour créer des règles de corrélation, il est nécessaire d'ajouter les éléments indispensables à prendre en compte, comme la capacité de détection des observateurs et les formats des notifications produites par ces dernières. De plus, une taxinomie des attaques est nécessaire pour lier l'attaque élémentaire de l'attaquant aux capacités de détection des observateurs. La revue des taxinomies existantes nous permet d'identifier CAPEC comme un candidat intéressant pour tenir ce rôle, principalement du fait de sa structure et du nombre de motifs d'attaques

référencés. L'intégration de ces éléments dans une base de connaissances adaptée fait l'objet du chapitre 3.

Chapitre 2

Relations entre point de vue du défenseur et celui de l'attaquant

Le chapitre précédent a dressé un panorama sur la corrélation d'alertes. Ce chapitre se focalise sur un point précis de la corrélation d'alertes qui est l'identification de scénarios d'attaque avec des méthodes de corrélation explicite. Cette approche suppose d'exprimer le scénario d'attaque à détecter sous la forme d'une règle de corrélation. Cette règle de corrélation représente le déroulement du scénario d'attaque du point de vue du défenseur car cette règle référence les différentes notifications pouvant survenir lors de la réalisation de l'attaque. Il est alors intéressant de comparer cette expression des scénarios d'attaque du point de vue du défenseur avec les formalismes de représentation centrés sur l'attaquant. L'objectif final est d'identifier les niveaux d'abstraction utilisés dans chacun des cas et de définir une méthode permettant de passer d'un scénario d'attaque exprimé dans un formalisme de haut niveau vers une règle de corrélation contenant des détails de niveau plus bas.

Ce chapitre se compose de deux parties. La première traite des langages de corrélation d'alerte permettant de réaliser l'identification de scénarios d'attaque. La seconde étudie les relations entre ces langages et deux formalismes utilisés en analyse de risque pour exprimer des scénarios d'attaque : les arbres d'attaque et les graphes d'attaques.

2.1 Expression d'un scénario d'attaque pour la corrélation d'alertes

Des langages ont été spécifiquement conçus pour décrire des scénarios d'attaque. Ces scénarios d'attaque directement interprétables par un moteur de corrélation constituent des règles de corrélation. Il est important de différencier cette règle de corrélation (spécification statique) de l'algorithme utilisé par le corrélateur pour détecter les attaques correspondantes (voir figure 2.1). Ces algorithmes reposent sur l'utilisation d'un automate dont l'avancement est lié aux événements faisant avancer le scénario d'attaque. Selon les approches, le langage dispose d'un niveau d'abstraction proche de la définition d'un automate ([CLF03]) ou bien plus proche d'une description de haut niveau (comme dans [TVM04]).

Les mécanismes de reconnaissance font intervenir des instances de scénarios partiellement détecté, parfois appelés plans ou threads. Par exemple, supposons qu'une règle de corrélation exprime une séquence attendue de messages de type A, puis B, puis C. Un flux de messages contenant une succession de deux messages de type A, puis un de type B, puis deux messages de type C ($AA'BCC'$) mène à plusieurs combinaisons possibles et stratégies de détection. Deux stratégies possibles sont par exemple de conserver uniquement la première occurrence de chaque type de message ou de conserver toutes les combinaisons possibles. Dans le premier cas, une seule alerte est levée alors que dans le second quatre sont levées. Chaque choix à une incidence en termes de ressources consommées et de possibilité de manquer certaines attaques. Par exemple, dans le cas de l'exemple ci-dessus, s'il existe une contrainte entre les champs des messages A et C et que seuls A'

et C les satisfont, la stratégie consistant à conserver uniquement le premier message de chaque type mène à un blocage de la détection et potentiellement à un faux négatif. Cette gestion de reconnaissance des instances partiellement reconnue est désignée sous le nom de gestion des coupures. Les différentes stratégies de gestion des coupures sont présentées dans les références [GLO08] et [PD02]. Les coupures constituent un élément important de l'algorithme de détection et des indications sur la manière dont certaines d'entre elles doivent être réalisées peuvent se retrouver spécifiées dans les règles de corrélation.

La suite de cette partie propose de passer en revue les différents outils et langages permettant d'exprimer des règles de corrélation.

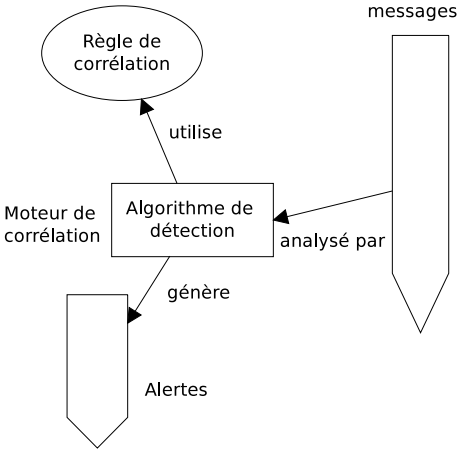


FIGURE 2.1 – Distinction entre la règle de corrélation (spécification) et la détection réalisée par un corrélateur. Les résultats de détection dépendent à la fois des algorithmes utilisés par le corrélateur et de la qualité de la règle de corrélation.

2.1.1 Présentation des langages de corrélation

Cette sous-section présente brièvement les différents langages permettant d'exprimer des règles de corrélation avant de passer en revue leurs caractéristiques communes.

ADeLe Le langage ADeLe a été défini dans [TVM04], [VTM05]. La signature du scénario d'attaque exprimée dans ce langage est automatiquement transformée en automate à états finis. Les règles de corrélations sont exprimées dans trois sections spécifiques : la section Filter, Enchain et Context.

La section Filter sert à définir des filtres pour conserver uniquement les messages qui peuvent contribuer à la reconnaissance du scénario. Cette section consiste en trois éléments : le nom du filtre (définissant un type de message), le type d'observateur qui doit produire le message et les valeurs des attributs constants de ce type de message. Lors de la reconnaissance, si un message correspond à l'un des filtres, le corrélateur associe un nom à ce message pour accéder à ses attributs.

La section Enchain définit l'ordre dans lequel les occurrences des différents messages sont attendus. Les opérateurs permettent de modéliser une séquence de messages (Opérateur Sequence), un choix (OneAmong), une concurrence (NonOrdered), une absence (Without) ou une répétition (Repeat).

La section Context permet d'exprimer les contraintes liant différents messages à l'exécution. Ces contraintes peuvent lier des attributs de plusieurs instances de messages ou bien être des indications temporelles (un temps d'expiration, un délai minimal ou maximal entre deux messages).

Lors de la détection, les attaques partiellement reconnues sont stockées dans une structure appelée plan. Ce dernier contient l'état courant de l'automate ainsi que l'historique des éléments déjà acceptés.

CAML Correlated Attack Modelling Langage [CLF03] utilise une décomposition dans laquelle un module représente une étape d'un scénario d'attaque. Le module est divisé en trois parties : l'activité, les pré-conditions et les post-conditions. L'activité décrit l'étape en termes de nombre d'événements, de la source et de la cible de chaque événement et de la classification de l'événement (concepts issus d'IDMEF). Les pré-conditions et les post-conditions sont exprimées à l'aide de prédicats pouvant porter sur des conditions temporelles, sur l'existence d'un service, d'un fichier, le statut de la machine ou le changement d'un utilisateur.

Lambda Le langage LAMBDA (proposé en [CO00]) permet d'exprimer la corrélation entre événements. Il consiste à définir les éléments suivants :

— les prérequis et les post-conditions de chaque événement
— la combinaison d'événements nécessaires à la détection de l'attaque
— la combinaison d'événements réalisés par l'attaquant
— les actions à générer pour vérifier le succès de l'attaque (étape de vérification)

Même s'il s'agit avant tout d'un langage utilisé pour réaliser de la corrélation semi-explicite, il est également possible de spécifier un enchaînement complexe d'actions. La modélisation est basée sur des transitions d'états (réalisées par l'occurrence d'événements). Les événements peuvent être composés de différentes manières (séquences, en parallèle, absence, choix non déterministe ou synchronisation).

Orchids Cet outil ([GLO08]) permet de détecter des actions spécifiées par des signatures. L'écriture d'une règle de détection revient à définir un automate reconnaissant une sous-séquence spécifique dans un flux d'événements. L'un des points forts de l'approche est l'utilisation de coupures implicites permettant d'éliminer des threads d'événements redondants ou qui ne peuvent pas mener à une détection complète.

STATL STATL [EVK00] est un langage de bas niveau permettant de définir des automates (états et transitions). La transition de type déroulement (unwinding) permet un retour à un état précédent et permet ainsi de modéliser une négation. En STATL, les coupures peuvent provenir des transitions *consuming* qui déplacent définitivement l'automate dans un nouvel état, sans possibilité de créations de différentes instances d'attaques. Le langage STATL permet également de définir des délais pour exprimer des temps d'expiration entre certaines transitions. Avec STATL, une contrainte inter-événements est construite explicitement par l'introduction d'une variable intermédiaire affecté à la valeur de chaque champ à comparer.

Sutekh Sutekh [PD01], [PD02] est un langage de spécification de règles de corrélation permettant de générer un diagramme de transition d'états à partir d'un scénario d'attaque. Ce langage permet d'exprimer des séquences, choix, ordres partiels, négations pour corréler des événements. La signature d'un événement unique est appelée filtre : un filtre exprime des contraintes entre les différents champs d'un événement et un terme (qui peut être un ensemble de constantes ou bien une variable). Les opérations d'égalité, d'inégalité, ainsi que la correspondance à une expression régulière sont définies. Les règles de corrélation sont composées à partir de filtres liés par les opérateurs séquence, choix, conjonction et négation. Il est possible de spécifier des prédicats externes ou des procédures externes pour lever des alertes partielles. Le niveau de granularité le plus faible est représenté par un événement (modélisé comme un ensemble de paires liant un nom de champ à une valeur). Une règle de corrélation correspond alors à la définition d'une grammaire.

2.1.2 Comparaisons des langages de corrélation

Après une brève présentation de chaque langage de corrélation, cette sous-section s'intéresse à comparer leurs principales caractéristiques. Le but est ainsi de comparer leur expressivité. Les dimensions qui interviennent dans la comparaison sont la définition et le filtrage d'événements, les définitions de contraintes entre événements, de contraintes temporelles, l'expression de l'ordre et les indications de traitement des instances de scénarios d'attaque partiellement reconnues (coupures).

A Filtrage et sélection d'événements

Chaque langage permet de définir les événements élémentaires qui composent un scénario. Ces événements se composent d'un ensemble de couples champ/valeur et constituent généralement les éléments fixes d'un événement particulier, comme le type d'événement ou le nom de l'événement.

Langage	Nom de la section	Filtrage (type)	Nom des champs
ADeLe	FILTER	ensemble champs-valeurs	arbitraire
CAML	Event	" "	basé sur IDMEF
Lambda	Event	prédicats	arbitraire
Orchids	Event	prédicats	arbitraire
STATL	State	EventSpec / variables	arbitraire
Sutekh	Filter	ensemble champs-valeurs	arbitraire

TABLE 2.1 – Sélection d'événements pour les différents langages.

Le récapitulatif présenté dans le tableau 2.1 reprend les caractéristiques de la sélection d'événements pour chaque langage. Les langages diffèrent sur la manière de spécifier les filtres, mais cette différence reste d'ordre syntaxique. Le langage ADeLe donne la possibilité de définir les événements comme des types d'événement à instancier. Un événement A représente un type à partir duquel des instances (A_1, A_2) sont définies. Ces instances sont caractérisées par un ensemble de valeurs de champs fixées par le type d'événement A.

B Contraintes

Contraintes inter-événements Le processus de reconnaissance d'un scénario d'attaque peut nécessiter la comparaison entre deux champs d'événements distincts. Il doit donc être possible d'exprimer des contraintes entre les valeurs de certains champs d'événements différents.

Langage	Nom de la section	Contraintes	Opérateurs de contraintes
ADeLe	CONSTRAINTS	contraintes entre champs	$= < > \neq$
CAML	/	contraintes entre variables	$= < >$
Lambda	detection	contraintes entre prédicats	$=$
Orchids	/	Substitutions	$=$
STATL	transition	contraintes entre variables	$= \neq$
Sutekh	signature constraints	contraintes entre variables	$= \neq$

TABLE 2.2 – Contraintes inter-événements pour les différents langages.

Comme le montre le tableau 2.2, tous les langages permettent au moins de définir une relation d'égalité entre les champs de deux messages. Ces contraintes s'expriment de différentes manières : la réutilisation d'une même variable (CAML, Orchids) ou de manière explicite en exprimant les liens entre les champs de deux instances d'événements (ADeLe, Lambda, Sutekh, STATL).

Contraintes temporelles L'enchaînement de deux événements est lié à une contrainte temporelle qui peut être un délai minimum ou un délai maximum. Ces contraintes temporelles peuvent être calculées à partir d'un timer spécifique (STATL, ADeLe) ou bien d'utiliser les champs d'horodatage inclus dans les événements. Les contraintes temporelles peuvent également être utilisées pour réaliser des coupures.

Langage	Nom de la section	Type de contrainte
ADeLe	CONSTRAINTS	Timeout, MinDelay, MaxDelay
CAML	/	Comparaison entre les champs dates
Lambda	/	prédicat *date*
Orchids	Gardes	temps ou nombre d'événements
STATL	transition	contraintes entre variables
Sutekh	Filter	Comparaison entre les champs dates

TABLE 2.3 – Contraintes temporelles des différents langages.

Le tableau 2.3 recense les différentes méthodes permettant de spécifier de telles contraintes pour chaque langage. Même si la méthode pour exprimer les propriétés temporelles sont différentes, il est possible d'exprimer le même type de contraintes. Par exemple, en ADeLe, *MaxDelay(A,B,10)* peut s'écrire $B.date < A.date + 10$ dans les langages utilisant le champ horodatage.

Contraintes externes Certains événements ne sont pertinents que si le système est dans un état donné. Par exemple, une attaque contre un service ne peut réussir que si ce dernier est actif. Ces contraintes supposent la possibilité de recevoir ces informations au moment de la détection. Ces contraintes sont souvent présentées comme prérequis à chaque étape du scénario d'attaque. Les pré-conditions des langages lambda et CAML, le *such_ that* du langage Sutekh, les annotations de STATL et les gardes d'Orchids peuvent servir à exprimer de telles contraintes.

C Ordonnancement des événements

Une fois les événements élémentaires définis, il est possible de les ordonnancer pour exprimer l'ensemble des enchaînements possibles de ces événements qui mènent à la réalisation du scénario d'attaque.

Langage	Section	Choix	Sequence	Concurrence	Absence
ADeLe	ORDER	X	X	X	X
Lambda	/	X	X	X	X
Sutekh	Signature	X	X	X	X
		Autre moyen d'expressions disponibles			
CAML	precondition	Opérateurs de logique temporelle (Allen)			
Orchids	/	Opérateurs de logique temporelle (Wolper)			
STATL	transition	Définition directe d'un automate			

TABLE 2.4 – Ordonnancement dans les différents langages.

La tableau 2.4 illustre la présence des différents opérateurs pour chaque langage.

Les opérateurs primaires qui se retrouvent dans tous les travaux sont la séquence, le choix entre deux événements et la concurrence (attente d'un ensemble d'événements non ordonnés). Des opérateurs supplémentaires sont également disponibles comme la répétition (ADeLe) permettant de factoriser une séquence d'événements qui se répètent où l'exécution parallèle de deux événements (qui peut être vu comme un cas particulier de concurrence).

D Gestion des plans et des coupures

C'est sans doute le point qui permet de différencier le plus les différents langages. Un plan représente une instance partielle d'un scénario d'attaque. Idéalement, il est souhaitable de conserver uniquement les plans qui mènent à la réalisation du scénario en éliminant les redondances possibles. Conserver l'intégralité des plans mène à une explosion combinatoire. Le rôle des coupures est de pallier cette explosion. Cette gestion est liée à l'algorithme de reconnaissance utilisé. Cependant, le langage de corrélation peut introduire des directives permettant de gérer certaines coupures.

Les coupures se distinguent également par leurs effets sur la détection :

— Si l'introduction de la coupure engendre potentiellement des faux négatifs,

— Si l'introduction de la coupure ne provoque pas de faux négatifs

Les premières sont parfois nommées coupures rouges et les secondes coupures vertes. De plus, elles sont explicites si elles peuvent être spécifiées dans la description du scénario et implicites si elles sont réalisées par un mécanisme interne au corrélateur.

La détection de scénarios peut suivre trois stratégies ([PD02]). Il est possible de détecter l'instance de scénario qui commence la première et qui finit également la première, celle qui est la dernière à commencer parmi celles qui finissent en premier, ou bien la plus courte. Les coupures implicites sont liées à l'algorithme de corrélation et non au langage. Ainsi, le moteur de corrélation d'Orchids [GLO08] détecte l'instance de signature la plus courte parmi les différentes possibilités (la relation *est plus court que* est exprimée avec l'ordre lexicographique sur les sous-séquences d'événements pris en compte). Il s'agit d'une coupure verte implicite.

Langage	Événement déclencheur	délais temporels	Coupure arbitraire
ADeLe	Without	(min/max)_delays	First
CAML	/	startime/endtime	/
Lambda	$\bar{e}$	définition manuelle	/
Orchids	via l'automate		
STATL	unwinding	timers	consuming
Sutekh	if_not	définition manuelle	/

TABLE 2.5 – Différents types de coupures explicites au sein des langages de corrélation.

La tableau 2.5 référence les possibilités de définir des coupures explicites dans les langages.

Les coupures par événement déclencheur sont liées à la détection d'un événement qui annule un ou plusieurs plans. La spécification d'une absence d'un événement particulier dans une règle de corrélation permet de réaliser ce type de coupure. Dans STATL, cette coupure est réalisée par les transitions *unwinding* qui permettent de revenir en arrière et d'annuler toutes les instances créées depuis le point de retour.

Les délais temporels entre deux événements peuvent également servir de coupure. Les choix des délais est arbitraire et peut engendrer des faux négatifs. Il s'agit donc d'une coupure rouge. Encore une fois, la principale différence sur ce point entre les différents langages reste au niveau syntaxique.

L'opérateur *First* du langage ADeLe appliqué à un événement permet de conserver uniquement la première occurrence de ce dernier et constitue ainsi une coupure rouge explicite. Dans STATL, la création d'instances de reconnaissance est contrôlée par les transitions de type *nonconsuming*. Une transition de type *consuming* ne crée pas de nouvelle instance de scénario et est comparable à l'opérateur *First* du langage ADeLe.

2.1.3 Discussion

Les langages de corrélation diffèrent sur certains points comme la syntaxe ou bien le niveau d'abstraction dans la description du scénario. Le couplage avec l'algorithme de détection peut être

également visible sous la forme de coupures explicites. Outre ces points, l'expressivité des différents langages reste similaire.

Aucune approche ne semble prendre en compte la gestion des règles de corrélation qui doivent être définies à la main. En effet, la définition des règles de corrélation nécessite une connaissance fine à la fois des scénarios d'attaque et du système. Il est ainsi nécessaire de connaître les différents types de messages générés par les observateurs pour pouvoir spécifier les enchaînements de notifications attendues.

À ce stade, il serait intéressant de disposer d'un formalisme permettant d'exprimer des règles de corrélation en faisant abstraction des détails d'implémentation du système. Pour cela, il est nécessaire de pouvoir représenter une règle de corrélation de manière générique. L'ordonnancement des événements peut être exprimé sous la forme d'un arbre dans lequel les nœuds représentent des opérateurs d'ordonnancement et les feuilles les événements (ou instances d'événements) à détecter. La figure 2.2 illustre cette représentation. Cet arbre exprime une règle de corrélation qui doit reconnaître une séquence constituée des événements A et B dans un ordre quelconque, mais sans apparition de C, suivis des événements D ou E.

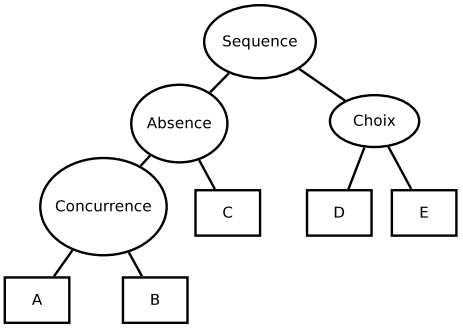


FIGURE 2.2 – Exemple de structure de règle de corrélation sous forme arborescente. Les événements (feuilles) sont ordonnancés en fonction de la sémantique des divers opérateurs

Cette représentation fait uniquement ressortir l'ordonnancement entre les événements. Les autres dimensions de la règle de corrélation (les différentes contraintes entre les valeurs des champs, les contraintes temporelles, les filtres) restent à spécifier. Cette représentation d'une règle de corrélation permet de faciliter la comparaison avec les formalismes de plus haut niveau abordés dans la partie suivante.

Dans la suite, la structure d'une règle de corrélation est comparée avec des formalismes utilisés en analyse de risques pour spécifier également des scénarios d'attaque mais avec un niveau d'abstraction différent.

2.2 Comparaison des règles de corrélation avec d'autres formalismes

Nous avons montré qu'il était possible de représenter une règle de corrélation sous une forme arborescente. Il est donc intéressant de voir dans quelle mesure la représentation d'un scénario d'attaque représenté sous la forme d'un arbre d'attaque ou bien d'un graphe d'attaques peut être liée à une règle de corrélation. Le but de cette section est donc de caractériser les liens possibles entre les règles de corrélation et ces deux formalismes. Nous proposons de regrouper les principales approches utilisant des arbres d'attaque dans une première partie puis de présenter dans une seconde partie les travaux traitant des graphes d'attaques. Une troisième partie synthétise les liens entre ces trois formalismes.

2.2.1 Les arbres d'attaque

Dans ce formalisme introduit dans [Sch99], un objectif d'attaque, représenté par la racine de l'arbre, est décomposé itérativement en sous-objectifs modélisés par les nœuds de l'arbre. Différents opérateurs binaires (*AND* et *OR*) sont associés aux nœuds internes de l'arbre pour exprimer des dépendances entre sous-objectifs. Ainsi, un nœud *AND* est réussi lorsque tous les sous-objectifs associés le sont, alors que la réussite d'un nœud *OR* nécessite seulement la réussite de l'un des sous-objectifs associés (voir figure 2.3). De nombreuses représentations et extensions des arbres d'attaque ont été proposées. Ces dernières peuvent être classées de différentes manières [KPCS13] : prise en compte des actions de l'attaquant ou du défenseur, modèle statique ou dynamique (dépendances temporelles), possibilité de définir des mécanismes de métrique et de quantification. Nous proposons par la suite une vision en deux axes : les extensions qui portent sur les métriques et celles qui apportent une nouvelle sémantique à la structure, par exemple par l'ajout de nouveaux types d'opérateurs ou de nœuds.

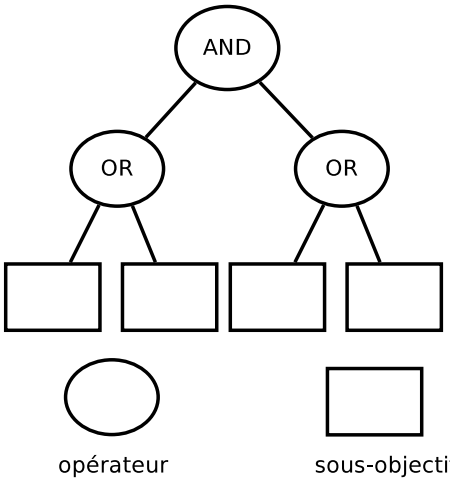


FIGURE 2.3 – Exemple de structure d'arbre d'attaque.

A Extensions portant sur les métriques

Une partie des travaux s'intéresse à l'ajout de métriques aux arbres. Ces métriques permettent de quantifier le temps mis par un attaquant pour réaliser son objectif, de déterminer les probabilités pour l'attaquant de réaliser un objectif donné sans se faire détecter ou encore de quantifier les gains potentiels d'une attaque réussie pour l'attaquant ainsi que les coûts engendrés. Le tableau 2.6 résume les dimensions usuellement développées.

temps d'exploitation	probabilité d'atteinte de l'objectif / de détection	gain / coût
[VJ03]	[BLP$^+$06]	[BLP$^+$06]
[WPWP10]	[cY07]	
[PC10]	[WpWm11]	[PDR12]
	[PC10]	

TABLE 2.6 – Classification des extensions portant sur les métriques.

Par exemple, des métriques ([VJ03]) peuvent être ajoutées pour modéliser la complexité et le temps d'exploitation nécessaire à la réalisation de chaque nœud. Cette notion est élargie au temps de détection (c'est-à-dire le temps que met un mécanisme de détection à rendre compte d'une étape de l'attaquant) dans les arbres développés dans [WPWP10] en plus du temps restant avant que l'attaquant n'atteigne son objectif.

Dans [BLP+06], les auteurs s'intéressent aux paramètres des gains pour l'attaquant, du coût de l'attaque, de la probabilité de succès de l'attaque, de la probabilité pour l'attaquant de se faire détecter et des pénalités attendues en cas de détection. Les graphes d'attaques bayésiens utilisés dans [PDR12] prennent également en compte la difficulté d'exploitation et les dommages potentiels résultant de chaque action de l'attaquant. L'arbre d'attaque amélioré [WpWm11] permet de calculer un score basé sur le coût de chaque attaque, sa difficulté et la probabilité pour l'attaquant d'être découvert.

Dans [cY07], un paramètre de durée de vie peut également être attribué à un nœud pour exprimer que l'étape considérée n'est valide que dans une fenêtre temporelle précise. Il est ainsi possible de calculer la probabilité qu'un objectif donné soit atteint sachant qu'un des sous-objectifs est également atteint.

Les Boolean Driven Markov Processes (BDMP) [PC10] sont un formalisme graphique introduits par Bouissou dans le domaine de la sûreté de fonctionnement et adaptés par Pietre-Cambacedes à des problématiques de sécurité. Il s'agit d'une combinaison d'arbres de défaillance et de chaînes de Markov. Les BDMP associent aux feuilles de l'arbre de défaillance des processus de Markov modélisant leur comportement selon des modes de fonctionnement. Ainsi, chaque feuille peut se retrouver dans un mode sollicité (le composant ainsi modélisé contribue au fonctionnement du système) ou non sollicité. Le processus de Markov modélise les possibilités de défaillance ou de réparation pour chaque mode du composant. Un couplage entre les modes de certaines feuilles et l'état d'autres feuilles est également introduit, ce qui permet de modéliser des dépendances séquentielles entre différentes branches de l'arbre. Dans le cadre de la sécurité, les deux modes sont *Actif* et *Inactif* selon la réalisation ou non de l'événement. Le choix du mode d'un processus de Markov piloté dépend d'une fonction booléenne de l'état des autres processus, ce qui permet de construire des séquences d'activation des différentes feuilles du BDMP. Ce formalisme permet de calculer le temps moyen global pour qu'un attaquant atteigne son objectif ou bien la probabilité qu'il atteigne un objectif avant un temps donné. Différentes stratégies de détection des actions de l'attaquant sont également modélisées. La discrimination s'opère sur l'instant de la détection effective. Cette détection peut ainsi s'opérer au commencement de l'action (détection initiale), lorsque l'action est en cours de réalisation (détection en cours), à l'instant où l'action se termine (détection finale) ou bien a posteriori.

B Extension portant sur la structure et les opérateurs

Outre les métriques ajoutées aux différents nœuds de l'arbre d'attaque, des extensions supplémentaires apportent de nouveaux opérateurs logiques (en plus du *ET* et du *OU* traditionnel) ou bien introduisent de nouveaux types de nœuds permettant de définir d'autres aspects que les sous-objectifs de l'attaquant.

B.1 Extensions des opérateurs

En plus des opérateurs *AND* et *OR* traditionnels, de nouveaux opérateurs apparaissent pour étendre l'expressivité de ces derniers. Le tableau 2.7 référence ces nouveaux opérateurs.

Travaux	Opérateur Séquence	Autres opérateurs
[BP03]	PAND	inhibit, XOR, transfert
[Yag06]		OU et ET généralisé
[Kha09]	PAND, SEQ	CSUB
[cY07]	O-AND	
[WpWm11]	SAND	
[PC10]	Gâchette	

TABLE 2.7 – Nouveaux opérateurs introduits pour étendre le ET et le OU et travaux associés.

Les portes supplémentaires définies dans [BP03] sont la porte XOR, la porte Priority AND

correspondant à une séquence temporelle (PAND), inhibit correspondant à une porte AND entre un événement et une condition sur l'environnement, modélisée par un événement conditionnant. Les notions de porte de transfert (in et out) permettent de hiérarchiser et de scinder les arbres trop volumineux mais n'apporte pas de sémantique supplémentaires.

Il existe également une différence entre un objectif intermédiaire (événement intermédiaire qui correspond à des sous-objectifs) et des événements basiques (qui correspondent au plus haut niveau de résolution possible). La figure 2.4 illustre une partie de ces éléments.

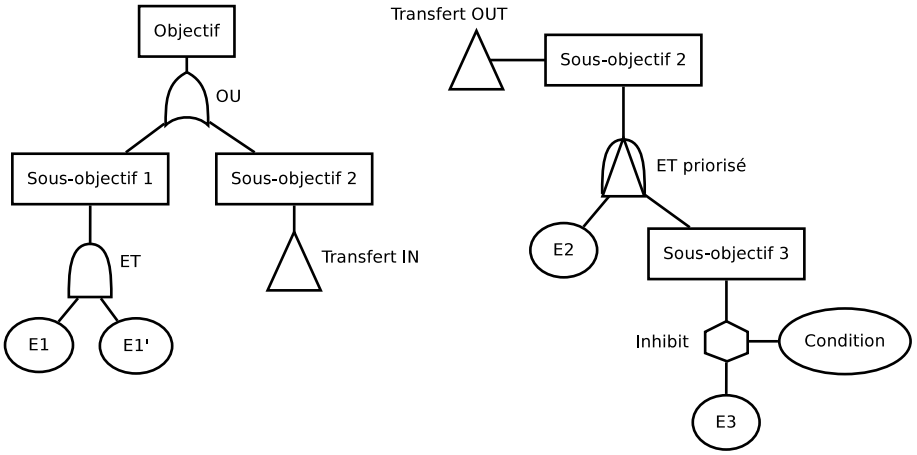


FIGURE 2.4 – Dans cette structure d'arbre d'attaque, les portes de transferts (représentées par des triangles) permettent de découper l'arbre en sous-structures. Le *ET priorisé* contraint l'ordre de réalisation des sous-objectifs (de gauche à droite) et l'opérateur *Inhibit* conditionne la réalisation de l'événement basique E3 .

Les auteurs de [Yag06] introduisent avec les arbres OWA une généralisation des opérateurs OU et ET des arbres d'attaque traditionnels permettant d'exprimer par exemple que la moitié des sous nœuds doit être vérifiée pour satisfaire l'objectif du nœud parent.

Les arbres de fautes dynamiques ([Kha09]) sont introduits pour dépasser les limites statiques des arbres d'attaque classiques. Ils ajoutent les portes logiques suivantes : PAND (porte ET à priorité), SEQ (porte séquentielle) et CSUB (Subordination conditionnelle) qui permet de définir une réussite automatique d'une sous-branche lorsqu'une autre est réalisée. La figure 2.5 donne un exemple d'utilisations de ces nœuds.

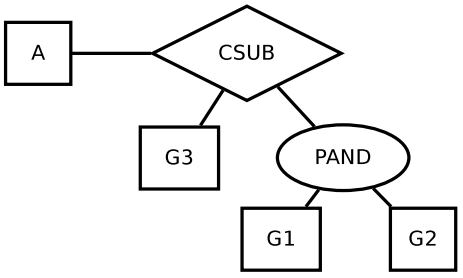


FIGURE 2.5 – Dans cet arbre de fautes dynamique, la porte CSUB exprime le fait que la réalisation du sous-objectif A réalise automatiquement les autres sous-objectifs dépendants de cet opérateur. Le PAND indique que le sous-objectif G1 doit être réalisé avant G2.

L'opérateur ET séquentiel se retrouve également dans [cY07] sous la forme d'un ET ordonné. Les arbres d'attaque améliorés ([WpWm11]) introduisent la porte SAND (ET séquentiel) pour la modélisation. La gâchette des BDMP ([PC10]) joue également un rôle similaire à l'opérateur séquentiel.

B.2 Extension des autres nœuds Ces extensions concernent plus particulièrement les feuilles de l'arbre. Des différences de sémantiques sont identifiées dans certains travaux. Le formalisme des BDMP [PC10] apporte également de nouveaux types de feuilles : ces dernières peuvent en effet être une action de l'attaquant qui peut être active si elle est en cours de réalisation par l'attaquant ou inactive, un événement de sécurité instantané qui modélise un événement dont la réalisation est instantanée, ou encore un événement de sécurité temporisé. Ce dernier modélise une action qui n'est pas directement sous le contrôle de l'attaquant, comme l'exécution d'une pièce jointe malveillante par un utilisateur.

L'approche [BP03] définit cinq types d'événements pouvant jouer le rôle de nœuds de l'arbre d'attaque : événement basique si aucun raffinement supplémentaire n'est possible, événement non développé lorsque le raffinement est arrêté par manque d'information, événement externe pour un événement arrivant au cours d'un fonctionnement normal, événement intermédiaire et événement conditionnant. Un événement peut représenter une faute ou bien être un événement qui participe simplement à la réalisation d'une faute de plus haut niveau.

Certains formalismes se sont intéressés à enrichir les arbres d'attaque avec les réponses possibles du défenseur. Par exemple, les nœuds des arbres sont annotés avec des contre-mesures potentielles. Dans les références [KMRS11] et [KMRS12], les nœuds de défense et d'attaque sont mêlés et les contre-mesures sont potentiellement contrées par de nouvelles attaques. Ainsi les contre-mesures ne sont donc pas nécessairement les nœuds terminaux de l'arbre.

C Construction de scénarios d'attaque et conversions

Usuellement, les arbres d'attaque sont des structures construites manuellement. Cependant, quelques travaux proposent d'automatiser ce processus. La génération automatique d'arbres d'attaque à partir de motifs d'attaque est par exemple proposée dans [LM01]. Une autre méthode de construction automatique d'arbres d'attaque de taille minimale est également proposée dans [RP05].

Des liens entre la structure d'arbre d'attaque et d'autres types de représentations sont parfois évoqués, par exemple les auteurs de [DH04] proposent la construction d'un arbre d'attaque à partir des chemins d'attaque décrits sous forme de graphes d'attaques. Qin et Lee ([QL04a]) proposent une méthode de conversion des arbres d'attaque vers un réseau bayésien : les nœuds sont conservés, les portes ET sont transformées en une séquence.

D Arbres d'attaque et règles de corrélation

Un arbre d'attaque qui représente un scénario d'attaque du point de vue de l'attaquant fait intervenir des sous-objectifs de différents niveaux de granularité. Une action de l'attaquant permettant de réaliser un sous-objectif donné peut être déterminée comme observable (ou non observable) selon les propriétés du système d'informations visé. L'association des observations possibles à ces sous-objectifs peut donner une structure qui permettrait de réaliser le lien entre les actions de l'attaquant et les observations possibles du côté du défenseur.

La structure d'arbre présentée en 2.6 décrit un scénario d'attaque du point de vue de l'attaquant. Les feuilles (représentées en gras sur la figure) peuvent être associées aux observations potentiellement réalisables lors de leur réalisation.

La figure 2.7 représente un arbre des observations de la réalisation du scénario décrit précédemment. Les feuilles en pointillés correspondent à des messages potentiels levés par les actions de l'attaquant. Ici, le type de notification (alerte ou événement) est indiqué et suivi du message ou de l'identifiant de la notification. Les nœuds (opérateurs) qui traduisent directement l'ordonnancement attendu des différents messages sont conservés. Cet arbre peut être considéré comme une ébauche ou un squelette de règle de corrélation associé au scénario présenté dans la figure 2.6. Pour disposer d'une règle de corrélation fonctionnelle, il est nécessaire de disposer des informations sur les autres champs des notifications (source, cible, date et autres informations permettant d'identifier les éléments impactés). De plus, les notifications présentées dans la figure 2.7 supposent l'existence d'observateurs capables de les générer.

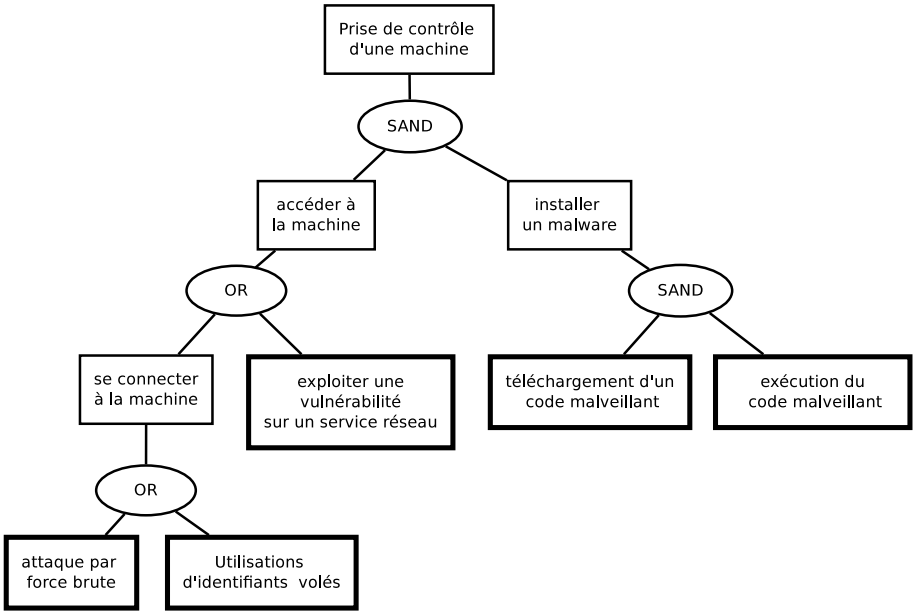


FIGURE 2.6 – Un arbre d'attaque (partiel) pour prendre le contrôle d'une machine.

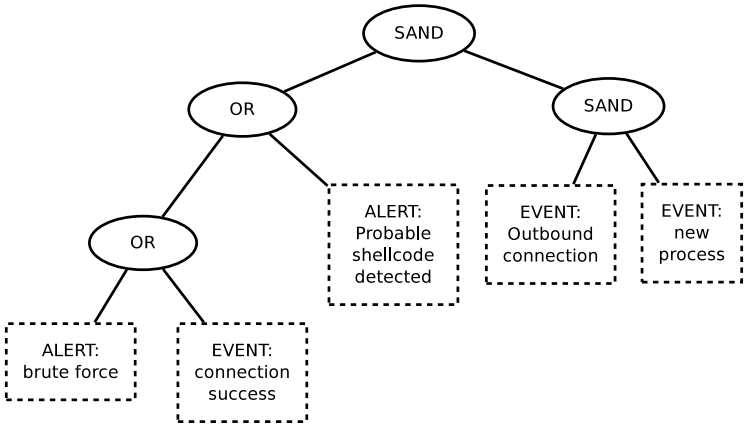


FIGURE 2.7 – Cet arbre représente des observations possibles lors de la réalisation du scénario décrit par la figure 2.6.

E Synthèse sur les arbres d'attaque

Les arbres d'attaque présentent ainsi de nombreuses variantes et extensions. Les arbres d'attaque enrichis avec un opérateur modélisant la séquence peuvent servir à exprimer des scénarios d'attaque génériques. Cependant, les arbres et les informations contenues dans les feuilles restent souvent informels ou bien modélisent des grandeurs (évaluation des dommages, probabilité de réussite en un temps donné) qui ne sont pas primordiales dans une optique de création de règles de corrélation.

Il a été mis en évidence l'existence d'un lien entre l'expression d'un scénario d'attaque sous la forme d'un arbre d'attaque et l'expression d'un squelette de règle de corrélation permettant de détecter ce scénario. Cependant, ce lien reste à approfondir pour permettre la création d'une règle fonctionnelle. L'un des points clefs qui n'a pas été abordé est l'identification des différents chemins d'attaques possible pour un même scénario d'attaque. Cela implique d'identifier les différentes combinaisons de nœuds potentiellement attaquables pour un scénario donné. Cette problématique nous amène à nous pencher sur les graphes d'attaques.

2.2.2 Les graphes d'attaques

Un graphe d'attaques représente la séquence des actions nécessaires à un attaquant pour atteindre son objectif [LI05].

Ce formalisme a deux usages principaux [SMC09], [KA13] : il peut s'agir d'un moyen de modélisation informel au même titre que les arbres d'attaque ([GW07], [MBFB06]) ou bien d'une structure de données complexe générée automatiquement et représentant de manière exhaustive les différents chemins d'attaques réalisables par un attaquant sur un système donné. Ce graphe peut être obtenu à partir d'une modélisation de la topologie du réseau et les informations sur les vulnérabilités de chaque machine.

La seconde dimension qui intervient est la sémantique des nœuds et des arcs du graphe d'attaques. Trois catégories principales de graphes se distinguent ainsi : les graphes représentant des transitions de l'état du système, les graphes à pré-conditions et les graphes à pré-conditions multiples. Ces types de graphe se différencient par la sémantique de leurs nœuds et de leurs arcs.

Il est également possible de catégoriser les graphes selon leur mode de génération. La construction peut être réalisée à partir d'outils de model-checking ou à partir d'une construction logique.

A Graphes à transition d'états du système

Les premiers travaux [RA00], [SHJ+02] concernant cette catégorie décrivent une méthode de génération de graphes d'attaques basée sur du model-checking. Les nœuds du graphe représentent un état possible du système et les arcs les exploits utilisés par l'attaquant. Cette génération est réalisée à partir d'une description du système permettant d'exprimer les liens physiques entre les machines, les services en écoute sur les machines ainsi que les vulnérabilités connues présentes sur chaque nœud. Chaque vulnérabilité dispose de pré-conditions et de post-conditions décrivant l'état du système avant et après l'exploitation. Pour générer les chemins d'attaque, il suffit de préciser la position et les privilèges initiaux de l'attaquant ainsi que la propriété de sécurité (assurant par exemple que l'attaquant ne peut pas obtenir de privilèges sur une machine donnée). Un automate va alors générer un contre-exemple présentant le scénario d'attaque.

Cependant, cette approche passe difficilement à l'échelle. En effet, de nombreux chemins générés diffèrent uniquement dans l'ordre dans lequel des attaques indépendantes sont réalisées. Ce type de graphes d'attaques grossit exponentiellement avec la taille du système et le nombre de vulnérabilités. La taille du graphe augmente exponentiellement avec le nombre de vulnérabilités et factoriellement avec le nombre de machines ([ILP06]). Ce problème peut être partiellement réduit en regroupant les machines ayant des configurations équivalentes et en éliminant les redondances du graphe ([SPEC01]). [Art02] et [APRS05] proposent également une méthode permettant de simplifier les graphes d'attaques générés en sélectionnant uniquement les chemins d'attaques les plus pertinents. La figure 2.8 présente la structure d'un graphe à transition d'états. Chaque état contient l'attaque qui est utilisée pour accéder à l'état suivant. Par exemple *sshd_ bof(0,1)* indique que l'attaque menée est un dépassement de buffer initié depuis la machine 0 (l'attaquant) vers la machine 1.

B Graphes d'attaques à prérequis

Cette catégorie de graphes d'attaques représente les états du système de manière implicite. Les nœuds du graphe représentent désormais les exploits et les liens les pré-conditions et post-conditions d'exploitation de ces exploits. De plus, il est fait l'hypothèse que les attaques suivent une propriété monotone ([AWK02]), ce qui signifie que l'attaquant ne peut pas perdre de privilèges en lançant une attaque. Ainsi, les privilèges de l'attaquant ne peuvent qu'augmenter et cela permet de réduire le nombre d'états. Cependant, cette hypothèse ne permet pas de modéliser les effets d'un déni de service ou de l'échec d'un exploit sur un service rendant impossible toute exploitation. Une approche similaire est utilisée par [WLD06], mais une hiérarchie plus stricte y est utilisée pour catégoriser les pré-conditions et post-conditions (Système d'exploitation, Application, Accès). De plus, l'étude propose d'utiliser les graphes obtenus pour optimiser le déploiement des IDS.

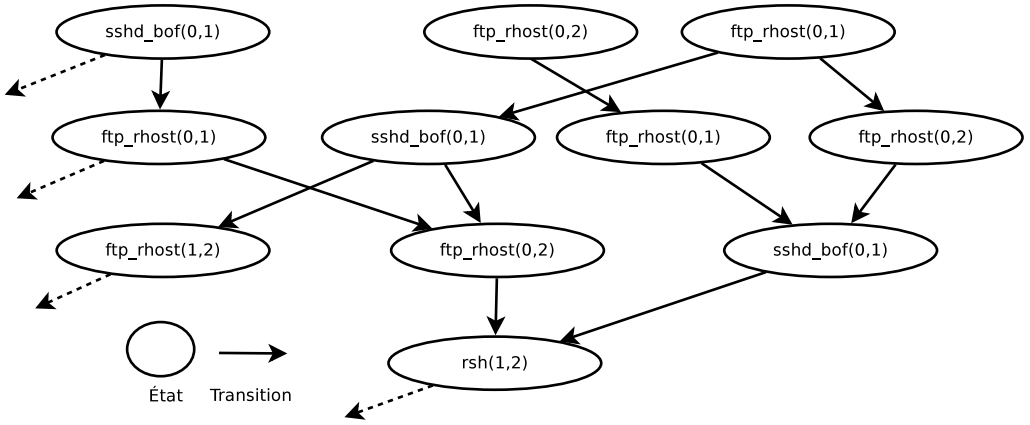


FIGURE 2.8 – Exemple d'un graphe à transition d'états.

C Graphes à prérequis multiples

Ces types de graphes disposent de plusieurs types de nœuds afin de différencier plusieurs aspects de l'exploitation. Par exemple, les graphes d'attaques utilisés dans [ILP06] sont constitués de trois types de nœuds. Les nœuds qui représentent l'état (les privilèges) d'un attaquant sur une machine donnée, les nœuds qui représentent les pré-conditions (en termes de connectivité) et les conséquences (en termes de privilèges) de chaque attaque et les nœuds représentant une vulnérabilité. Trois approches principales présentent des graphes d'attaques appartenant à cette catégorie : MulVAL, NetSPA et TVA.

C.1 MulVAL MulVAL ([OBM06]) est un outil permettant de construire des graphes d'attaques logiques dont la complexité est quadratique en temps et polynomiale en espace en fonction de la taille du système. Pour cela, un nœud du graphe ne représente plus l'état complet du système mais un prédicat logique. Les arcs spécifient alors des relations de causalité. Cependant, cette approche est sémantiquement équivalente à énumérer les pré- et post-conditions de chaque exploit (cette approche est utilisée dans les graphes de dépendance d'exploits). MulVAL est couplé à un scanner qui lui permet d'être informé des vulnérabilités et configurations des différentes machines du système pour pouvoir construire des graphes d'attaques spécifiques au système. Des règles génériques permettent de modéliser des comportements génériques d'enchaînement d'attaques.

Une adaptation de l'algorithme pagerank pour le calcul de priorité dans le graphe est proposé dans [SO08].

C.2 NetSPA Les travaux de Lippmann et Ingols [LIS+05] se basent sur le système NetSPA proposé initialement par [Art02]. Les graphes produits contiennent trois types de nœuds : les états, les prérequis et les instances de vulnérabilités. Un état atteignable par l'attaquant est la combinaison d'une machine et d'une action (user, root, DOS ou autre). Les pré-conditions des attaques peuvent prendre la forme d'identifiants (login, clef SSH ou autre) et de connectivités entre interfaces. Une vulnérabilité est définie par sa localité, qui correspond à une exploitation distante ou locale, et par son effet correspondant au niveau de privilège gagné. La modélisation du système prend en compte les interfaces réseaux pouvant avoir un ou plusieurs ports ouverts, des instances de vulnérabilités associées à ces ports ouverts, les configurations des pare-feux et des routeurs et enfin la valeur des biens. La figure 2.9 donne un exemple d'un graphe d'attaques pouvant être généré par NetSPA.

C.3 TVA Les travaux [JNO05], [JN07], [NJ08],
[NEJ+09], [JN10b] et [JN10a] proposent une approche topologique de l'analyse de vulnérabilité. L'approche retenue permet d'analyser l'intégralité des chemins d'attaque sans pour autant avoir à les énumérer. Les composants principaux du système sont un ensemble de modèles d'exploits

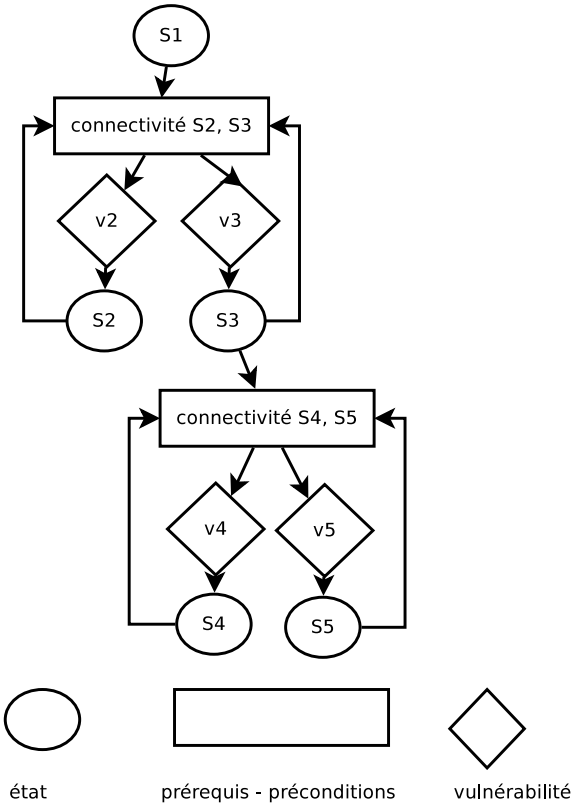


FIGURE 2.9 – Exemple d'un graphe à prérequis multiples de type NetSPA.

stockés dans une base de connaissances et indépendants de tout réseau particulier, une modélisation du réseau à étudier et la spécification d'un scénario d'attaque qui comprend la cible de l'attaque, l'état initial et les modifications de configurations réseau. La construction de la base de vulnérabilités se fait avec les résultats issus par des scanners de vulnérabilité comme Nessus. Les vulnérabilités doivent également être définies avec un ensemble de pré et post-conditions. Ces conditions de sécurité sont exprimées par des variables booléennes et servent à modéliser le succès d'une exploitation. Le modèle du réseau est peuplé automatiquement, la base d'exploits semi automatiquement (l'ajout des conditions est manuel). Le modèle du réseau associe une table de connectivité à chaque machine.

En plus de la phase d'analyse du risque, une partie traitant de détection est également abordée. La phase de détection associe les événements détectés aux éléments correspondant dans le graphe d'attaques. Cette détection est possible uniquement si chaque alerte est associée explicitement aux identifiants des vulnérabilités ou exploits dont l'exploitation peut être détectée. Les auteurs soulignent qu'ils ne modélisent pas les attaques de types Man-in-the-middle (trois machines sont impliquées à la place de deux usuellement) [NEJ+09]. La figure 2.10 présente un graphe d'attaques de type TVA. Les prérequis sont présentés au niveau des arcs.

D Variantes et problématiques connexes aux graphes d'attaques

De nombreuses variantes et modifications de graphes d'attaques existent. De plus, la création d'un graphe demande une représentation du système cible qui conditionne la précision du graphe d'attaque. Dans la suite, quelques variantes ou choix de sémantiques spécifiques concernant les graphes sont abordés, puis la problématique de vérification des chemins d'attaques générés, puis une présentation de certains choix de modélisation des systèmes d'informations dans le cadre de la génération de graphes d'attaques.

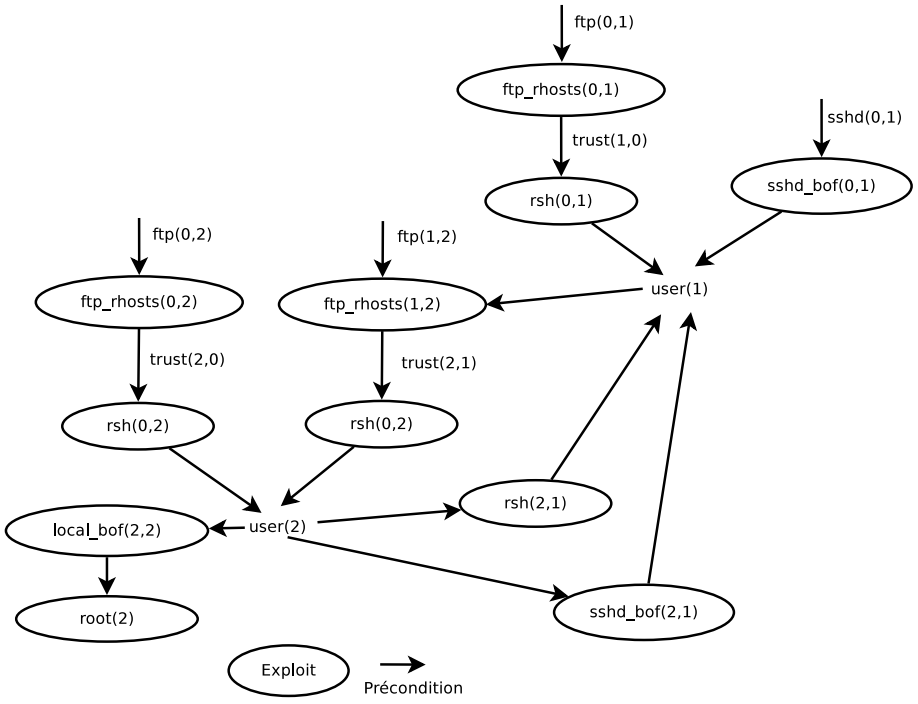


FIGURE 2.10 – Exemple d'un graphe à prérequis de type TVA.

Certains travaux font évoluer la sémantique des nœuds et des arcs des graphes d'attaques. Par exemple, les auteurs de [ND07] mettent en place un graphe dans lequel les nœuds représentent des vulnérabilités et les arcs des actions. Cela suppose que tous les éléments du système soient accessibles. Dans [LFXX10], les nœuds représentent une machine et le privilège obtenu par l'attaquant alors que les arcs représentent les exploits (vulnérabilités). Dans [WLJ06] le graphe d'attaques contient à la fois des nœuds qui représentent des exploits (identifiant de la vulnérabilité, source et cible de l'attaque) et des alertes pour les actions qui ne correspondent pas à l'exploitation d'une vulnérabilité (par exemple un PING ICMP).

CAPEC est utilisé dans [ZYL09] comme moyen d'exprimer les actions d'attaques. Ce sont les pré-conditions, post-conditions ainsi que poids de chaque attaque qui entrent en jeu. Nous avons montré au Chapitre 1 que CAPEC pouvait servir de référence pour nommer et organiser les attaques élémentaires lors de la création de scénario d'attaque. Une approche similaire est utilisée dans [XCT+09].

D'autres dimensions sont parfois recherchées pour caractériser les attaques constituant le graphe. Il est proposé dans [FWSJ08] une représentation d'un graphe d'attaques sous la forme d'un réseau bayésien dynamique incorporant des facteurs temporels liés à la facilité d'obtention d'exploits ou de patchs. À chaque exploit sont affectés différentes métriques : un score de base pour quantifier ses propriétés intrinsèques, des scores temporels divisés en trois métriques (exploitabilité, restauration et confiance).

D'autres travaux se sont concentrés sur la comparaison entre différents structures de graphes d'attaques. C'est ce qui est réalisé dans [AJN12] où les auteurs mettent en avant un système d'équivalence entre deux représentations d'un même graphe d'attaques. Le premier est un graphe classique, le second est un graphe arborescent dans lequel la racine constitue l'objectif final de l'attaquant.

La validation des chemins d'attaques exprimés dans un graphe d'attaques est également un axe de recherche. L'approche proposée en [OSR13] décrit une architecture permettant d'exécuter les chemins d'attaques inclus dans des graphes d'attaques sur des réseaux de taille moyenne. Le système est constitué d'une architecture (Core impact) comportant des modules exploits et un modèle de l'environnement d'application comprenant des informations sur les machines, les systèmes

d'exploitation, les ports utilisés, les services réseau disponibles et les machines compromises, d'un planificateur prenant en entrée une description du domaine et du scénario en PDDL. L'approche décrite permet de générer des chemins d'attaque sur le système et de les valider automatiquement sur le système cible.

Des aspects de topologie et de dépendances entre composants fonctionnels sont pris en compte dans certains graphes d'attaques. L'outil MsAMS (Multi-step Attack Modelling and Simulation, [Fra09], [FVEWL09]) est un outil de graphes d'attaques permettant de représenter les relations et dépendances liées aux identifiants de connexion qui sont nécessaires pour accéder à certains comptes ou services. Il permet également de prendre en compte les VLANs. L'idée de dépendances à différents niveaux du système est également développée avec les graphes de dépendances introduits dans [AJPS11]. Ces représentations permettent de faire le lien entre graphes d'attaques classiques, les nœuds physiques du système et les dépendances fonctionnelles entre ces nœuds. Les graphes générés dans [ICL+09] prennent en compte les contraintes liées aux NAT et aux règles de filtrage complexes des pare-feux. Ces problématiques ne sont pas propres aux graphes d'attaques et sont également rencontrées dans le domaine de la corrélation. Il est en effet nécessaire de connaître les flux potentiellement visibles par des NIDS ainsi que les correspondances entre IP avant et après translation d'adresses et de ports.

Nous proposons dans la suite de mettre en relation les approches basées sur les graphes d'attaques avec la structure des règles de corrélation.

E Liens entre graphes d'attaques et règles de corrélation

La présentation des problématiques et des approches liées aux graphes d'attaques montre que les graphes d'attaques peuvent être utilisés pour réaliser une analyse du risque sur un système donné. Les graphes permettent l'obtention de chemins d'attaques constitués généralement d'un enchaînement d'exploits permettant à un attaquant de compromettre les éléments clefs d'un système. L'objectif est ici de réaliser les liens possibles entre ces graphes d'attaques et les règles de corrélation.

Les graphes d'attaques sont utilisés dans certains travaux pour servir de support à la détection de scénarios d'attaque ([RCM11], [NRJ04], [WLJ06]).

L'approche [RCM11] offre un exemple d'utilisation de graphe d'attaques pour l'identification de scénarios d'attaque en utilisant la correspondance suivante : une alerte a est un tuple $a = (t, s, d, c)$ avec t l'estampille temporelle de l'alerte, c la classification de l'alerte et s et d respectivement la machine source et destination. Ces alertes peuvent être associées aux nœuds d'un graphe d'attaques (généré en utilisant MulVAL). Ces nœuds sont définis par le triplet (im, h, r), im étant l'impact de l'attaque, h le nœud sur lequel elle est effectuée et r la référence de la vulnérabilité exploitée. La correspondance entre une alerte et un nœud du graphe peut être établie en vérifiant les correspondances entre la classification de l'alerte et la référence de la vulnérabilité ($c = r$), la correspondance de la cible ($d = h$) et l'appartenance de la source s à un autre nœud du graphe. De même dans [NRJ04] les identifiants de Snort sont associés aux identifiants des vulnérabilités découvertes par Nessus.

En s'inspirant de ces approches qui reposent sur l'utilisation de graphes d'attaques pour exprimer les scénarios à détecter, nous proposons de réaliser une opération similaire à celle présentée en 2.2.1.D pour développer le scénario. Le but est d'appliquer à l'exemple développé à la figure 2.6 une transformation d'un graphe d'attaques vers une structure permettant de prendre en compte les problématiques de détection. La figure 2.12 représente la première étape de cette adaptation. Il s'agit d'un graphe d'attaques dans lequel un nœud contient l'identifiant de l'attaque réalisée par l'attaquant, l'identifiant du nœud attaquant et du nœud attaqué. Ici, la machine initiale de l'attaquant est identifiée par *1*. Les identifiants *2* et *3* correspondent à deux machines qui peuvent être la cible de l'attaquant pour ce scénario d'attaque. La figure 2.11 schématise le système de référence utilisé pour construire le graphe d'attaques.

L'attaquant dispose de trois entrées possibles pour la compromission (qui correspondent au développement des deux branches *OR* de l'arbre d'attaque 2.6). Au premier niveau se trouvent les possibilités de force brute, de vols d'identifiants ou d'exploitation d'une vulnérabilité d'un service

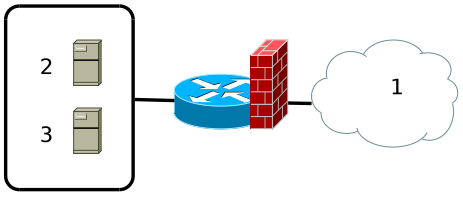


FIGURE 2.11 – Système de référence utilisé pour la construction du graphe d'attaque. L'attaquant est à la position 1 et les machines 2 et 3 sont des cibles potentielles.

web (pour la machine *2*) ou une exploitation d'une vulnérabilité shellschock (pour la machine *3*). Les actions suivantes correspondent au téléchargement et au lancement du code malveillant (développement du *SAND* de la figure 2.6).

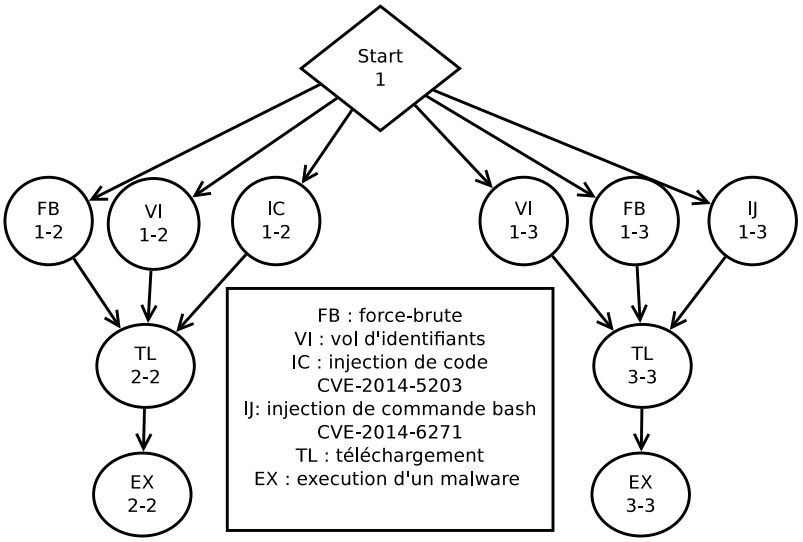


FIGURE 2.12 – Graphe d'attaques présentant le scénario de la figure 2.6.

Il est ensuite possible d'associer aux nœuds de ce graphe les messages correspondant à la réalisation de ces actions, comme illustré à la figure 2.13. L'identifiant de l'action utilisé dans le graphe initial est remplacé par l'identifiant de l'alerte qui doit être levée lors de la réalisation de l'étape du scénario par l'attaquant.

Les informations permettant d'identifier la source et la cible de l'attaque sont également à inclure dans les messages. Cependant, des informations manquent pour déterminer si cet identifiant apparaît au niveau du champ source ou cible de l'alerte ou du message. Par exemple, dans le cas de l'action de téléchargement, le graphe d'attaque renseigne une cible et une source identifiant uniquement le nœud à l'origine du téléchargement au lieu d'identifier la machine distance avec laquelle la transaction a lieu.

La transformation des nœuds en message suppose également une connaissance précise des moyens de détection et du lien avec les actions de l'attaquant.

F Discussion sur les graphes

Les graphes d'attaques sont des outils permettant de générer des chemins d'attaques, le plus souvent à partir d'une base de connaissances représentant le système d'information à défendre. Certains graphes d'attaques sont également directement utilisés pour la corrélation d'alertes. Des outils comme MulVAL permettent déjà de générer des graphes d'attaques à partir de règles génériques. Cependant, dans tous ces travaux, le lien entre les nœuds du graphe qui correspondent le plus souvent à une exploitation de vulnérabilité, et la détection est pris en compte de manière très restrictive. Par exemple, dans TVA, le modèle considère uniquement des NIDS avec une corres-

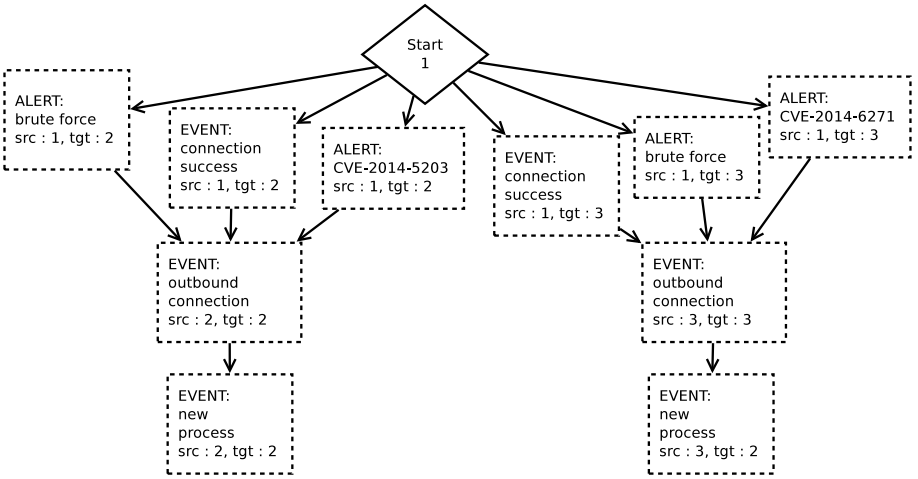


FIGURE 2.13 – Graphe d'alertes issu du graphe de la figure 2.12.

pondance directe entre identifiant de vulnérabilité et identifiant d'alertes. De plus, aucun scénario d'attaque n'est spécifié dans un formalisme de haut niveau. Le plus souvent, les scénarios générés sont constitués d'un enchaînement d'exploitations de vulnérabilités locales et distantes renseignées dans la base de connaissances.

2.2.3 Identification des liens entre les graphes d'attaques, les arbres d'attaque et les règles de corrélation

Nous avons présentés successivement trois formalismes génériques permettant d'exprimer des scénarios d'attaque : les arbres d'attaque, les graphes d'attaques et les règles de corrélation. Les comparaisons respectives entre les règles de corrélation et les arbres d'attaque d'une part et entre les règle de corrélation et les graphes d'attaques ont été effectuées. Cette sous-section propose d'étudier brièvement les liens possibles entre ces trois représentations.

A Convergence entre les graphes d'attaques, les arbres d'attaque et les règles de corrélation

Le tableau 2.8 résume des caractéristiques des trois structures étudiées. L'unité élémentaire correspond aux nœuds du graphe d'attaques, aux feuilles de l'arbre d'attaque et aux filtres des règles de corrélation. Le niveau d'abstraction de la structure cible (la règle de corrélation) doit être choisi comme référence à appliquer dans les autres structures. Ce niveau d'abstraction permet d'exprimer des messages à un faible niveau d'abstraction, c'est-à-dire un niveau d'abstraction qui fait intervenir les détails du système cible.

Formalisme	Unités élémentaires	Structure	Niveau d'abstraction
graphe d'attaques	exploit	graphe	faible
arbre d'attaque	sous-objectif / action	arbre	variable
règle de corrélation	événement / message / alerte	arbre	faible

TABLE 2.8 – Éléments de comparaison macroscopique entre graphes d'attaques, arbres d'attaque et règles de corrélation.

Les sections précédentes ont déjà présenté des adaptations respectives des structures d'arbres d'attaque et de graphes d'attaques permettant de faire apparaître des messages attendus lors de la détection d'un scénario d'attaque (figure 2.7 et figure 2.13). Il reste à établir les liens entre ces deux structures pour compléter cette comparaison et faire ressortir leur utilité pour nos travaux.

B Liens entre représentation sous la forme de graphes et sous la forme d'arbres

L'arbre de la figure 2.7 et le graphe de la figure 2.13 représentent un même scénario d'attaque. Les différences entre ces deux représentations sont de deux types. D'une part, la structure est différente car le graphe ne comporte pas les opérateurs logiques présents dans l'arbre d'attaque. D'autre part, le graphe est instancié, ce qui signifie que les chemins d'attaques sont explicitement développés avec les identifiants des machines qui interviennent à chaque étape.

En supposant que le niveau d'abstraction des nœuds du graphe et des feuilles de l'arbre est similaire, l'ensemble des transformations qui suivent permettent de passer d'une structure à l'autre.

— transformer les enchaînements simples en *ET Séquentiel* (*SAND*), comme illustré en (1) dans la figure 2.14 ;
— transformer les jonctions à l'aide d'opérateurs *OR*, comme dans le cas (2) de la figure 2.14 ;
— factoriser les combinaisons à l'aide d'un opérateur *AND* lorsque cela est possible (cas (3) sur la figure 2.14).

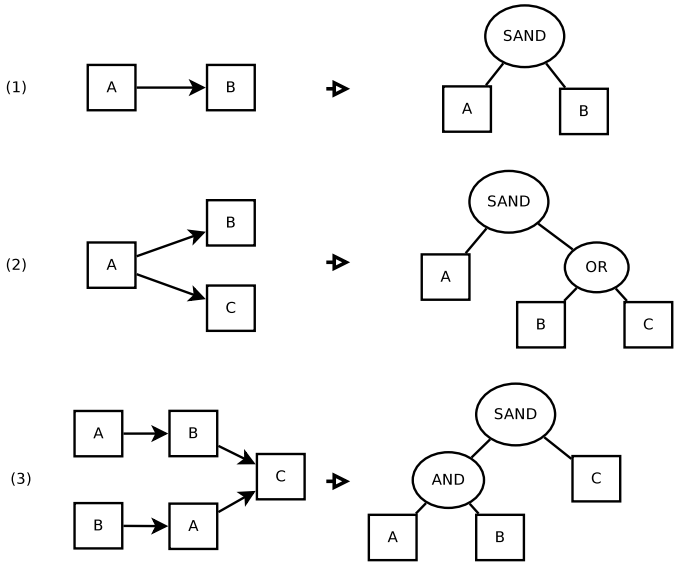


FIGURE 2.14 – La première transformation illustre la traduction d'une séquence d'un graphe d'attaques vers un arbre d'attaque. La seconde transformation illustre la traduction d'un embranchement du graphe d'attaques vers un arbre d'attaque. La troisième transformation est une factorisation possible à l'aide d'un opérateur *AND*.

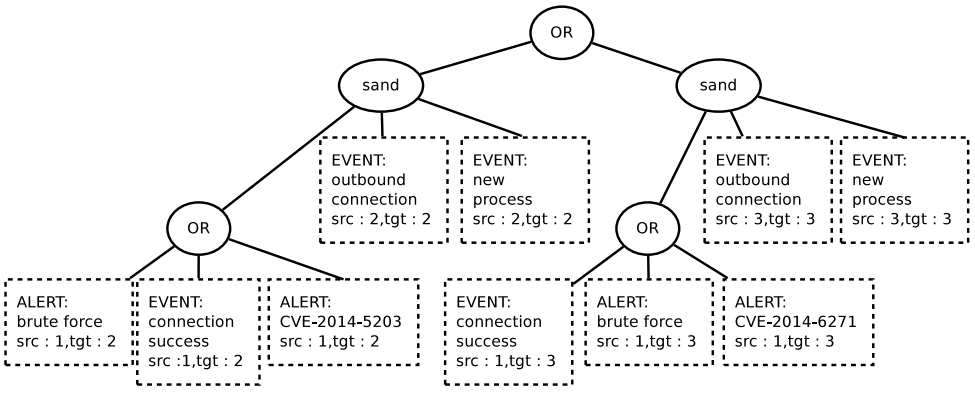


FIGURE 2.15 – Transformation du graphe de la figure 2.13 en arbre.

Il est possible d'obtenir une instanciation de l'arbre représenté à la figure 2.7 en spécialisant chaque feuille de l'arbre sur un chemin d'attaque spécifique et réitérer cette opération pour tous les chemins possibles. L'ensemble des arbres est ensuite agrégé via l'ajout d'un nouvel opérateur *OR*. Il est également possible de transformer directement le graphe de la figure 2.13 en arbre en appliquant les règles de transformations présentées au paragraphe précédent. La figure 2.15 illustre le résultat de ces transformations.

Cette structure reste un embryon de règle de corrélation qui explicite principalement l'ordonnancement général des messages attendus. L'obtention d'une règle de corrélation utilisable nécessite des éléments supplémentaires, en particulier les contraintes entre les champs de différents messages et le contenu précis des champs générés par les observateurs.

De plus, il est fait l'hypothèse que chaque action était observable par un observateur et mène à la production d'un unique type de message. Ces informations dépendent fortement de l'environnement et de la configuration des observateurs.

C Discussion

Cette comparaison des approches utilisant des graphes d'attaques, des arbres d'attaques et des règles de corrélation a mis en avant plusieurs points. Non seulement un graphe d'attaques peut servir de squelette pour la création de règles de corrélation, mais en plus il est possible de réaliser une correspondance avec une structure arborescente qui peut être également interprétée comme une règle de corrélation. Un graphe d'attaques peut être généré automatiquement pour donner un ensemble de chemins d'attaques possible sur un système. La fusion de ces deux structures pourrait permettre de disposer d'un point de départ pour l'obtention de règles de corrélation adaptée à l'environnement. L'un des objectifs de la thèse est de proposer un langage permettant de spécifier un scénario d'attaque avec un niveau d'abstraction suffisamment élevé.

Peu de travaux s'intéressent à la génération des règles de corrélation. L'approche qui semble se rapprocher le plus de notre objectif est [STPRS15] dans laquelle il est proposé une architecture de corrélation auto-adaptative reposant sur une génération de règles de corrélation par un algorithme de programmation génétique. Cependant cette approche suppose un apprentissage avec des données d'attaques représentatives et demande également une normalisation sémantique élevée pour chaque événement. Un jeu de données contenant des messages classés en fonction de leur caractère malveillant est nécessaire pour réaliser la phase de sélection des scénarios lors de la mise en œuvre de l'algorithme génétique. De plus, les exemples mis en avant correspondent plus à des reconstructions d'attaques élémentaires (DOS, HTTPFlood) plutôt qu'à une identification de scénarios d'attaque.

2.3 Conclusion

Ce chapitre a mis en parallèle l'expression de scénarios d'attaque dans le domaine de la corrélation et dans le domaine de l'analyse de risques. Les caractéristiques communes des langages de corrélation ainsi que leurs limites (niveau d'abstraction très bas) sont ainsi ressorties. D'un côté, les arbres d'attaque (et leurs variantes) sont un formalisme dont la structure peut être adaptée pour décrire des scénarios d'attaque génériques spécifiés sous une forme analogue à celle des règles de corrélation. Un graphe d'attaques présente un ensemble de chemins d'attaques faisant intervenir des machines et des vulnérabilités spécifiques au système.

L'idée est de mettre en relation ces trois éléments : exprimer un scénario d'attaque générique sous la forme d'un arbre d'attaque puis utiliser une méthode de génération (ou plutôt d'instanciation) qui se rapproche de la construction des graphes d'attaques pour obtenir des règles de corrélation exprimées dans un langage cible. La principale différence avec l'approche des graphes d'attaques est que la génération est contrainte par la spécification initiale du scénario et le résultat est une règle de corrélation complète qui prend en compte les capacités de détection déployées au sein du système.

Chapitre 3

Construction d'une base de connaissances pour la génération de règles de corrélation

Dans ce chapitre, nous décrivons notre première contribution qui consiste en la définition d'une base de connaissances servant de support à la génération des règles de corrélation. Le premier chapitre a donné un aperçu des fonctionnalités de diverses bases de connaissances utilisées comme support à la corrélation, à la détection d'intrusion ou pour la génération de graphes d'attaques. L'objectif de ce chapitre est de proposer une base de connaissances offrant un niveau d'abstraction adapté pour la construction de règles de corrélation. Les informations contenues dans cette base doivent permettre la transformation d'une spécification générique de scénario d'attaque en un scénario d'attaque instanciant les spécificités du système modélisé. Cette base de connaissances doit remplir plusieurs objectifs : (1) fournir une modélisation de la topologie du système suffisamment fine pour permettre une sélection des nœuds impliqués dans un scénario d'attaque ; (2) exprimer la capacité de détection des différents éléments du système qui permettent d'observer des actions et exprimer également le contenu des messages générés par ces éléments lors de la détection ; (3) permettre de lier les actions de l'attaquant et leur détection.

La base de connaissances présentée dans ce chapitre s'appuie sur M4D4 ([MMDD09]) et propose une évolution [1] de cette base de connaissances pour permettre de résoudre les problématiques adaptées à nos travaux. La première section justifie ce choix et récapitule la structure et le contenu de M4D4. La section suivante présente les éléments de notre base de connaissances en soulignant les modifications apportées par rapport à M4D4. La troisième partie traite des problématiques spécifiquement liées à la mise en œuvre et l'utilisation de cette base de connaissances.

3.1 Choix d'une base de référence

La construction de cette base de connaissances est réalisée en se basant sur les spécifications d'une base existante. Le choix de M4D4 [MMDD09] comme fondation paraît justifié pour plusieurs raisons. Tout d'abord sur le fond, une grande partie des éléments de modélisation recherchés sont présents (cartographie, vulnérabilités, attaques) et en particulier le modèle de visibilité des IDS. Ensuite, sur la forme, la base est conçue pour être facilement implémentée dans un langage logique (Prolog), ce qui donne certains avantages de développement et de souplesse, au prix de limitations en performances. Cependant, la base M4D4 a été initialement conçue comme une architecture permettant de fédérer des informations contextuelles pertinentes pour remettre les notifications de sécurités dans leur contexte et permettre de réaliser des requêtes pour tester la corrélation entre événements et alertes reçues. Dans notre approche, les informations pertinentes concernent la description d'un système d'information ainsi que la modélisation du comportement des outils de

1. Comme les évolutions proposées ne sont pas entièrement compatibles avec la description originale de M4D4, on parle d'une évolution ou d'une adaptation plutôt que d'une extension.

détection. La partie de M4D4 dédiée à la fédération des événements générés par le système est par exemple en dehors de notre périmètre.

La base de connaissances à construire doit contenir toutes les informations contextuelles indispensables à la création des règles de corrélation. Ces informations sont organisées de la manière suivante :

— une première partie modélise la topologie du système ;
— une seconde partie est dédiée à la cartographie (les services et logiciels installés) ;
— une partie dédiée aux actions, aux attaques et aux vulnérabilités qui décrit une hiérarchie de classes d'attaques élémentaires et d'actions standards pouvant être réalisées par un attaquant sur le système ;
— une partie dédiée aux observateurs et à la supervision inclut les positionnements et configurations des observateurs.

La section qui suit propose un passage en revue des différentes parties de la base de connaissances en rappelant les concepts issus de M4D4 et introduisant les évolutions apportées.

3.1.1 Rappels sur M4D4

A Topologie et cartographie

Cette partie de M4D4 met à disposition des prédicats permettant de décrire l'infrastructure fonctionnelle d'un système d'information. Le tableau 3.1 liste les prédicats liés à la topologie du système et les prédicats permettant de spécifier la cartographie, c'est-à-dire les processus et services instanciés au niveau de chaque machine, sont regroupés dans le tableau 3.2. La topologie décrite dans M4D4 repose sur la définition de réseaux et de nœuds connectés à un réseau.

Prédicats	signification
network(N)	N est un réseau
netaddress(N, A)	A est l'adresse du réseau N
node(H)	H un est nœud
nodeaddress(H, Ah)	Ah est une adresse de H
gateway(H)	H est un routeur
nodenet(H, N)	Le nœud H appartient au réseau N
nodegw(H, Hg)	Le routeur Hg est directement accessible par H
routes(Hg1, Hg2, N)	Hg1 route les paquets vers le réseau N par Hg2
path(Hgs, Hgd, L, N)	L est la liste de routeurs composant la route vers le réseau N depuis Hgs vers Hgd
route(Hs, Hd, L)	L est une route entre la source Hs et la destination Hd
nat(G, A1, P1, A2, P2)	L'adresse A1 et le port P1 sont translatés respectivement A2 et P2 par le routeur G
nodename(H, Name)	Name est le nom d'hôte de H
resolve(A, Name)	Le FQDN de l'adresse A est Name

TABLE 3.1 – M4D4 : topologie.

Des précisions sont apportées concernant la lecture du tableau 3.1. Le prédicat *nodenet(H, N)* peut correspondre à un fait ou bien est déduit (via une règle) par vérification de l'appartenance d'une adresse du nœud H au sous-réseau N. Lorsque le noeud H est en dehors du système administré, le prédicat *nodegw(H, Hg)* est vrai si *Hg* est un routeur de bordure permettant l'interconnexion avec un réseau externe. Le prédicat *nodegw(H, Hg)* associe à un noeud externe au système administré un routeur en bordure du système. Le prédicat *path* est défini par des règles qui mettent en jeu un prédicat auxiliaire *travel* chargé de construire la liste des routeurs traversés de manière

récursive :

$$path(Hgs, Hgd, L, N) \leftarrow travel(Hgs, Hgd, [Hgs], L, N)$$
$$travel(Hgs, Hgd, R, [Hgd|R], N) \leftarrow routes(Hgs, Hgd, N)$$
$$travel(Hgs, Hgd, R, L, N) \leftarrow routes(Hgs, Hgi, N) \wedge Hgi \neq Hgd$$
$$Hgi \notin R \wedge travel(Hgi, Hgd, [Hgi|R], L, N)$$

Le prédicat *route(Hs, Hd, L)* permet de déterminer la liste L des routeurs constituant une route entre *Hs* et *Hd*. Cette route est déterminée en identifiant les routeurs respectifs *Hgs* et *Hgd* des nœuds *Hs* et *Hd* en faisant appel au prédicat *path* :

$$route(Hs, Hd, L) \leftarrow nodegw(Hs, Hgs) \wedge nodegw(Hd, Hgd)$$
$$nodenet(Hd, N) \wedge path(Hgs, Hgd, L, N)$$

Prédicats	signification
software(N, V, T, A)	Logiciel de nom N, version V, de type T et d'architecture A
hosts(H, S)	Le nœud H héberge le produit S
process(S, U)	Le processus S est exécuté avec l'identité U
exec(H, P)	Le nœud H exécute le processus P
service(P, Q)	Le processus P écoute sur le port Q
listens(H, Ser)	Le nœud H exécute le service Ser

TABLE 3.2 – M4D4 : cartographie.

B Vulnérabilités et attaques

Les prédicats introduits dans M4D4 pour décrire les vulnérabilités et les classes d'attaques sont présentés dans le tableau 3.3. Une vulnérabilité est définie comme une faiblesse au niveau de l'architecture, l'administration ou l'implémentation d'un système ou d'un logiciel. Les vulnérabilités modélisées dans M4D4 reposent sur une configuration vulnérable constituée d'un ensemble de logiciels.

Prédicats	signification
vulnerability(V)	V est une vulnérabilité
affects(V, C)	La vulnérabilité V affecte la configuration C
takespartin(S, C)	Le logiciel S est un élément de la configuration vulnérable C
refersto(V, O, Vn)	La vulnérabilité V est nommée Vn par l'organisation O
equiv(O, V, O', V')	V et V' sont des noms d'une même vulnérabilité (respectivement de l'organisme O et O')
severity(V, S)	La sévérité de la vulnérabilité V est S
requires(V, W)	W est un prérequis pour exploiter la vulnérabilité V
losstype(V, C)	Les conséquences de l'exploitation de la vulnérabilité V sont C
published(V, D)	D est la date de publication de la vulnérabilité V
attackclass(C)	C est une classe d'attaque
inherits(C1, C2)	C1 est une classe fille de C2
attacksubclass(C1, C2)	C1 est une sous-classe de C2
exploits(C, V)	La classe d'attaque C exploite la vulnérabilité V

TABLE 3.3 – M4D4 : vulnérabilités et classes d'attaques.

En plus des éléments apportés dans le tableau 3.3, il faut noter que le prédicat *not_ vulnerable(H, V)* exprime le fait que le nœud H n'est pas affecté par la vulnérabilité V :

$$\begin{aligned} not\_vulnerable(H,V) \leftarrow & vulnerability(V) \wedge node(H) \wedge \forall C.affects(V,C) \\ & \wedge (\exists Sv.takespartin(Sv,C) \\ & \quad \wedge (\forall Sn.hosts(H,Sn) \wedge Sv \not\approx Sn)) \end{aligned}$$

Le nœud H n'est pas vulnérable si l'intégralité des configurations vulnérables mettent en jeu au moins un produit qui diffère de tous les produits hébergés par H.

Le prédicat *attacksubclass(C1, C2)* est défini par la règle :

$$attacksubclass(C1,C2) \leftarrow inherits(C1,C2)$$
$$attacksubclass(C1,C2) \leftarrow inherits(Ci,C2) \wedge attacksubclass(C1,Ci)$$

C Analyseurs

Les prédicats qui permettent une description des outils de sécurité qui génèrent des messages (IDS) et de leurs capacités de détection sont regroupés dans le tableau 3.4

Prédicats	signification
analyser(A)	A est un analyseur
ids(A)	A est un IDS
hids(A)	A est un IDS hôte
monitors(A, H)	L'HIDS A surveille le nœud H
monitors(A, H, S)	L'IDS applicatif A surveille l'application S du nœud H
nids(A)	A est une IDS réseau
ips(A)	A est un IPS (Intrusion Prevention System)
monitors(A, Hg)	L'IPS A est positionné sur le routeur Hg
monitors(A, G1, G2)	Le NIDS A surveille le lien G1, G2.
can_detect(A, Hs, Hd)	L'IDS A peut être capable d'analyser un paquet allant du nœud Hs au nœud Hd
kbids(A)	A est un IDS fonctionnant par signatures
abids(A)	A est un IDS comportemental
signature(S)	S est une signature d'IDS
active(A, S)	La signature S est active pour l'IDS A
detects(C, S)	La signature S est prévue pour détecter la classe d'attaque C
func_vis(A, C)	L'IDS par signatures A peut détecter des attaques qui sont des sous-classes de C
detects(A, C)	L'IDS comportemental A peut détecter l'attaque C ainsi que ses sous-classes
scanner(A)	A est un scanner de vulnérabilité
monitors(A, V, H)	A recherche la vulnérabilité V sur le nœud H
deny(Hg, As, Ps, Ad, Pd)	Hg bloque les connexions entrantes venant de l'adresse As et du port Ps vers le port Pd de l'adresse Ad

TABLE 3.4 – M4D4 : analyseurs.

Le prédicat *can_ detect(A, Hs, Hd)* distingue les cas dans lesquels A est un IPS placé en coupure au niveau d'un routeur G ou un NIDS :

$$can\_detect(A,Hs,Hd) \leftarrow ips(A) \wedge route(Hs,Hd,L) \wedge monitors(A,G) \wedge G \in L$$
$$can\_detect(A,Hs,Hd) \leftarrow nids(A) \wedge route(Hs,Hd,L) \wedge monitors(A,Gi,Gj) \wedge [Gi,Gj] \subset L$$

La visibilité opérationnelle *func_ vis(A, C)* se dérive selon le type d'analyseur (IDS par signature ou IDS comportemental). Un analyseur peut détecter une classe d'attaque C si cette dernière est une sous-classe d'une classe d'attaque C' qui est détectable par l'analyseur :

$$\begin{aligned} func\_fis(A,C) \leftarrow & kbids(A) \wedge active(A,Sig) \wedge detects(Sig,C') \\ & attacksubclass(C,C') \\ func\_fis(A,C) \leftarrow & abids(A) \wedge detects(A,C') \\ & attacksubclass(C,C') \end{aligned}$$

D Événements et alertes

Les prédicats permettant de modéliser les événements et alertes produits à l'exécution sont regroupés dans le tableau 3.5. La principale raison d'être de la modélisation des événements est liée à la volonté de ne pas limiter M4D4 à une simple base de connaissances mais d'en faire une plateforme de corrélation vers laquelle les événements du système sont directement renvoyés.

Prédicats	signification
event(E, T)	L'événement E se produit au temps T
rawevent(E)	E est un événement brut
ippacket(E, S, D, P, ...)	E est un paquet IP
includes(E1, E2)	L'événement E1 inclut l'événement E2
message(E, A)	E est un message produit par l'analyseur A
evidence(M, R)	Le message M est associé à l'événement brut R
alert(E, AI)	E est une alerte portant sur l'instance d'attaque AI
attacktype(AI, C)	L'instance d'attaque AI appartient à la classe C
attackorigin(AI, O)	L'origine de l'instance d'attaque AI est O
attacktarget(AI, T)	La cible de l'instance d'attaque AI est T
origin(O, F)	F fait partie de l'origine O
target(T, F)	F fait partie de la cible T
impact(AI, S, C, T)	S, C et T sont respectivement la sévérité, le statut (réussite ou échec) et le type de l'instance d'attaque T
subsumes(A, M)	L'alerte A intègre le message M dans le diagnostic de corrélation
report(E, VI)	E est un rapport sur l'instance de vulnérabilité VI
vulninstance(VI, H, V)	L'instance de vulnérabilité VI exprime la présence de la vulnérabilité V sur la machine H.
fwlog(E)	Le message E est un log de pare-feu

TABLE 3.5 – M4D4 : événements et alertes.

3.2 Enrichissement des éléments de la base de connaissances

Cette section présente les adaptations et modifications apportées à la base de connaissances M4D4. Les parties de cette section sont respectivement dédiées à la topologie, la cartographie, aux vulnérabilités et actions exploitables par un attaquant et aux outils de détection. À la fin de chaque partie, un tableau récapitulatif résume les transformations réalisées. Ces transformations peuvent prendre la forme d'ajouts de prédicats adaptés ou de modifications de prédicats existants (ajout d'un argument, modification de la règle, modification de la sémantique).

3.2.1 Topologie

La topologie du système englobe la connaissance des nœuds, des sous-réseaux ainsi que des communications possibles entre ces nœuds. Cette sous-section est organisée de la manière suivante.

La modélisation des nœuds est tout d'abord évoquée. Il est ensuite question de la problématique du routage entre les nœuds. Ce point est important pour déterminer si un flux peut être observé entre différents points du réseau. La modélisation des règles de filtrage joue également un rôle important pour permettre de déterminer si une connexion est possible ou non entre deux points du système. La principale question concerne le niveau de détails et de précision voulu pour modéliser de telles règles. Ces problématiques de filtrage amènent également à étudier les spécificités introduites par les translations d'adresses et de port. Ces mécanismes entraînent en effet une modification des informations d'identifications du flux qui compliquent la détermination de l'accessibilité entre différentes machines.

A Modélisation des nœuds

Les différents nœuds qui constituent le système sont représentés dans M4D4 avec le prédicat *node(N)*. Les routeurs permettent la communication entre sous-réseaux et disposent de leur propre prédicat : *gateway(G)*. Dans M4D4, une machine est directement associée à une (ou plusieurs) adresses via le prédicat *nodeaddress(Node,Addr)*. Il est cependant intéressant de disposer d'informations explicites sur les interfaces réseau pour les raisons qui suivent :

(a) dans certains cas, les observateurs identifient les machines par leur adresse physique ou font référence à un nom d'interface ;

(b) cela permet de modéliser plus facilement la présence d'un service en écoute sur une interface particulière ;

(c) cette information permet de lever certaines ambiguïtés (en particulier pour modéliser le routage).

Par conséquent, nous proposons d'enrichir cette notion d'adresse par l'utilisation d'interface réseau. Le prédicat **nodeiface***(N,eth1,PhyAddr)* est ainsi utilisé pour spécifier l'adresse physique (*PhyAddr*) associée à l'interface *eth1* de la machine *N*. L'appartenance d'un nœud N à un réseau *Net* sur l'interface réseau *eth1* se spécifie alors par *nodenet(N,eth1,Net)*. Un routeur G directement accessible par le nœud N en utilisant l'interface réseau *eth1* s'exprime alors par *nodegw(N,G,eth1)*. Le prédicat original de M4D4 *nodegw/2* est lié à ce nouveau prédicat *nodegw/3* par la relation $nodegw(N, G) \leftarrow nodegw(N, G, \_)$, ce qui permet d'exprimer simplement l'association entre un nœud et un routeur directement accessible lorsqu'il n'existe pas d'ambiguïté sur les interfaces utilisées.

Modification	Type de modification	Justification
nodeiface	ajout	explicitation de l'interface
nodenet	ajout d'un argument	lien entre interface et réseau
nodegw	ajout/ ajout d'un argument	interface réseau utilisée

TABLE 3.6 – Évolution de la spécification des nœuds.

Le tableau 3.6 résume les modifications apportées par cette partie.

B Modélisation du routage

M4D4 propose un mécanisme permettant de calculer les routes possibles lors de la communication entre deux machines. La route y est modélisée sous la forme de la liste des routeurs traversés. Cependant, cette modélisation est ambiguë dans certaines configurations et ne permet pas de déterminer les réseaux traversés. Or cette information sur les sous-réseaux traversés est indispensable pour déterminer la visibilité d'un NIDS en écoute uniquement sur un sous-réseau (ou sur une interface particulière d'un routeur).

La figure 3.1 illustre un de ces cas. Dans cette configuration, l'utilisation du prédicat de M4D4 donne *route(a,b,[r2,r1])*, ce qui est vrai (la route de a à b est passe bien par les routeurs $r1$ et $r2$). Cependant cette information sur la route ne permet pas de déterminer si un NIDS en écoute sur le sous-réseau *net1* peut voir le flux.

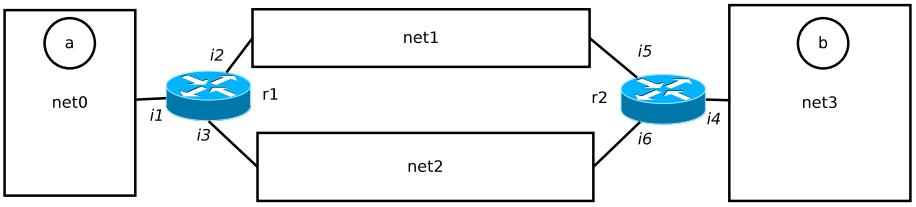


FIGURE 3.1 – Configuration topologique permettant d'illustrer l'importance de la connaissance des réseaux traversés ou des interfaces réseau (notées ix pour $x \in \{1 \dots 6\}$) utilisées.

Pour pallier ce manque, nous proposons l'utilisation des informations qui concernent les interfaces réseau utilisées pour le routage. Ainsi, le prédicat *routes(G1,G2,Net,iface)* exprime le fait que le routeur *G1* envoie les paquets à destination du réseau *Net* vers le routeur *G2* en utilisant son interface réseau *iface*. Une interface réseau étant liée à un sous-réseau, il est alors possible de connaître les sous-réseaux traversés.

Le prédicat *route(Hs,Hd,L)* exprimant le fait que la route L permet au nœud Hs de communiquer avec le nœud Hd est alors défini comme indiqué au listing 3.2.1. Dans cette règle, les routeurs Hgs et Hgd directement accessibles par les nœuds respectifs Hs et Hd sont déterminés avec le fait *nodegw* (3.2.1a). Le nœud Hd et le routeur Hgd sont tous deux rattachés au sous-réseau N respectivement par les interfaces Ifd et $Ifgd$ (3.2.1b).

$$
\begin{aligned}
route(Hs, Hd, L) \leftarrow & nodegw(Hs, Hgs) \wedge nodegw(Hd, Hgd, Ifd) && \text{(3.2.1a)}\\
& \wedge nodenet(Hd, Ifd, N) \wedge nodenet(Hgd, Ifgd, N) && \text{(3.2.1b)}\\
& \wedge Gws.n = Hgs \wedge Gwd.n = Hgd \wedge Gwd.out = Ifgd && \text{(3.2.1c)}\\
& \wedge path(Gws, Gwd, L, N) && \text{(3.2.1d)}
\end{aligned}
$$

Les variables Gws et Gwd (3.2.1c) représentent des dictionnaires comprenant l'identifiant d'un routeur (n) et l'interface de sortie servant au routage (out).

Les prédicats auxiliaires *path(Gws, Gwd, L, N)* (3.2.1d) et *travel(Gws, Gwd, R, L, N)* (3.2.2a) sont alors utilisés. Les trois cas de résolution du prédicat *travel* correspondent respectivement au cas où Gws et Gwd sont confondus (3.2.2b), le cas où une route directe[2] existe entre les deux routeurs (3.2.2c), et au cas générique faisant intervenir un routeur intermédiaire Hgi (3.2.2e).

$$
\begin{aligned}
path(Gws, Gwd, L, N) \leftarrow & travel(Gws, Gwd, [Gws], L, N). && \text{(3.2.2a)}\\
travel(Gws, Gws, R, R, N) \leftarrow & routes(Gws.n, Gws.n, N, Gws.out). && \text{(3.2.2b)}\\
travel(Gws, Gwd, R, [Gwd|R], N) \leftarrow & Gws \neq Gwd && \text{(3.2.2c)}\\
& \wedge routes(Gws.n, Gwd.n, N, Gws.out). && \text{(3.2.2d)}\\
travel(Gws, Gwd, R, L, N) \leftarrow & \wedge Gwi.n = Hgi \wedge Hgi \neq Gwd.n && \text{(3.2.2e)}\\
& \wedge Gwi \notin R && \text{(3.2.2f)}\\
& \wedge routes(Gws.n, Hgi, N, Gws.out) && \text{(3.2.2g)}\\
& \wedge travel(Gwi, Gwd, [Gwi|R], L, N). && \text{(3.2.2h)}
\end{aligned}
$$

Le tableau 3.7 déroule cet algorithme sur l'exemple de la figure 3.1 pour déterminer la route entre les nœuds a et b.

Cependant, comme souligné dans [MMDD09], cette modélisation ne permet toujours pas de modéliser le routage asymétrique dans lequel les paquets d'un même flux sont routés différemment.

C Les réseaux et sous-réseaux

Dans M4D4, un sous-réseau est défini par le prédicat *network(N)*. Il est également possible de lui associer une adresse (via *netaddress(N,A)*). Cependant, le principal intérêt de cette associa-

2. Cette route est renseignée avec le prédicat *routes/4* qui indique une route connue alors que le prédicat *route/3* utilise une règle pour déterminer une route entre deux nœuds arbitraires

étape	prédicats	Commentaires
3.2.1a	$route(a, b, L)$	
3.2.1a	$nodegw(a, r1), nodegw(b, r2, ifb)$	ifb est l'interface réseau reliant b au réseau $net3$
3.2.1b	$nodenet(b, ifb, net1), nodenet(r2, i4, net3)$	L'interface $i4$ permet à $r2$ de communiquer avec b
3.2.1c	$Gws.n = r1, Gwd.n = r2, Gwd.out = i4$	
3.2.2a	$path(Gws, Gwd, L, net3)$	
3.2.2c	$travel(Gws, Gwd, [Gws], [Gwd, Gws], net3)$	cas correspondant à la configuration
3.2.2d	$routes(r1, r2, net3, i2)$	$r1$ utilise l'interface $i2$ pour router les paquets vers $net3$
	$L = [Gwd, Gws], Gws.out = i2$	route entre a et b

TABLE 3.7 – Mise ne pratique de l'algorithme de routage sur l'exemple de la figure 3.1.

Modification	Type de modification	Justification
routes	ajout d'un argument	interface de sortie
route, *travel*	modification des règles	prise en compte des interfaces de sortie

TABLE 3.8 – Évolution de la spécification des routes.

tion est de permettre l'association par déduction d'une machine à un sous-réseau via le prédicat *matches(A,NA)*, où *A* représente une adresse IP et *NA* l'adresse du réseau à laquelle *A* appartient. Par exemple, une machine dont l'adresse est 10.0.0.1 appartient au réseau 10.0.0.0/24 si *matches(10.0.0.1,10.0.0.0/24)* est vrai. Cette association peut cependant être ambiguë dans des cas où plusieurs sous-réseaux ont des plages d'adresses qui se chevauchent. Par exemple, en reprenant l'exemple précédent, 10.0.0.1 appartient également aux sous-réseaux 10.0.0.0/8 ou 10.0.0.0/28. Un autre exemple illustre la limite de ce choix en considérant un réseau d'une grande entreprise dans laquelle chaque site géographique dispose d'un adressage privé identique et un système complexe de NAT pour les communications internes entre les différents sites. Ces cas ambigus sont évités en considérant le prédicat *nodenet* comme un fait et non comme une règle déduisant l'appartenance à un sous-réseau en fonction de l'adresse IP.

Il n'y a pas de prédicat spécifique dédié aux VLANs. Les VLANs sont modélisés en utilisant les sous-réseaux et routeurs standards, au prix d'une perte de la possibilité de modélisation des attaques exploitant des caractéristiques propres aux VLAN.

Contrairement à certains modèles ([Lag10]), nous ne donnons pas d'éléments de représentation des sous-réseaux hiérarchiques. L'utilisation d'une hiérarchie (inclusion de différents sous-réseaux) est intéressante dans des problématiques de visualisation.

D Règles de filtrage

Les règles de filtrage constituent un élément important. En effet, ces règles permettent de limiter les connexions possibles entre les différents nœuds du système. Cependant, dans un système réel, les règles de filtrage sont potentiellement complexes ou contradictoires. L'objectif de la modélisation n'est pas de spécifier le comportement exact d'un pare-feu mais de disposer d'un formalisme pour identifier un sur-ensemble des connexions possibles afin de limiter au maximum la prise en compte d'interactions impossibles entre plusieurs machines du fait des règles de filtrage. La présence au niveau du modèle de flux de communications impossibles dans le système réel a pour conséquence de mener à la génération de chemins qui ne pourront pas être utilisés par un attaquant et ne doivent pas avoir d'influence sur la détection. La seule conséquence négative est l'augmentation de

la complexité de la règle de corrélation générée.

M4D4 définit un pare-feu avec le prédicat *fw(F)* et les règles de filtrage sont exprimées avec *deny(F, SA, SP, DA, DP)* pour modéliser l'interdiction des connexions de l'adresse SA et du port source SP vers l'adresse DA sur le port DP.

Dans nos travaux, la politique par défaut est inversée par rapport à M4D4. Par défaut, tout nœud n'étant pas un pare-feu autorise toutes les connexions et tout pare-feu a une politique de blocage par défaut. Ce choix est justifié en se reposant sur le rôle initial du pare-feu qui est de laisser passer uniquement les connexions autorisées. L'autorisation d'une connexion est alors exprimée par le fait **accept***(F, SA, SP, DA, DP)*. Pour des raisons de facilité d'usage et d'expression des règles, SA et DA peuvent être des adresses de nœuds, ou bien des identifiants ou adresses de sous-réseaux. Ces règles de filtrage agissent non pas au niveau des paquets mais au niveau d'un flux de communication (requête et réponse). Il faut donc comprendre F autorise les connexions entre les couples adresses/ports $SA : SP$ et $DA : DP$.

Les limites de ce modèle de pare-feu sont les suivantes :

(a) le niveau de granularité est binaire (autorisé/interdit) et ne permet pas de différencier les politiques au niveau des protocoles autorisés ; Par exemple, si seul le protocole UDP est autorisé dans la configuration réelle, la modélisation autorise n'importe quel protocole.

(b) les règles contradictoires ou qui s'annulent ne sont pas prises en compte ;

(c) toutes les règles utilisant des mécanismes avancés (inspection en profondeur, port knocking, filtrage basé sur des plages horaires) ne sont pas modélisables.

Le modèle de pare-feu est très simple contrairement à des modèles plus complexes comme FIREMAN ([YCM+06]) et ses dérivés qui modélisent les interactions entre les différentes tables de filtrage d'un pare-feu. Ces modèles sont utilisés pour déceler des erreurs de configuration ou des redondances entre les règles de filtrage. L'objectif de la modélisation du pare-feu n'est pas de détecter une erreur de configuration dans les règles de filtrage ou d'obtenir un simulateur de pare-feu, mais simplement d'obtenir un sur-ensemble représentatif des accessibilités possibles entre les nœuds du système.

E Gestion du NAT

Les pare-feux utilisent souvent des translations d'adresses et de ports en plus du filtrage. La modélisation de cette translation est importante à la fois pour permettre de contrôler l'accessibilité entre machines mais également pour connaître les adresses sources et cibles visibles dans chaque segment du système. Le NAT destination (DNAT) et le NAT source (SNAT) sont différenciés par l'introduction des prédicats **snat***(F,A,P,A',P')* et **dnat***(F,A,P,A',P')*.

De plus, il est probable que du NAT soit utilisé en conjonction de règles de filtrage. Il est alors nécessaire de déterminer si les règles de NAT sont appliquées avant ou après les règles de filtrage. En distinguant le DNAT et le SNAT, il est possible de s'inspirer du fonctionnement de netfilter : le DNAT est appliqué suivi des règles de filtrage puis des règles de SNAT.

Le prédicat *connectivity(S,T,P,C)* (3.2.3a) exprime l'accessibilité entre le nœud S et le port P du nœud T. C correspond alors à la liste des triplets (R_i, R_{i-1}, Cs_i) où Cs_i représente les métadonnées du flux entre deux routeurs consécutifs R_i et R_{i-1} (3.2.3h). La liste de ces métadonnées CS est calculée par le prédicat *track_all_segments(Seg, R, Cs)*, où Seg correspond aux métadonnées du lien entre le dernier routeur et le nœud cible T (3.2.3d). L'information sur la route R (3.2.3b) joignant S et T est utilisée pour déterminer la liste des routeurs traversés. *node_addr_to_gw(N,NA,G)* exprime le fait que la machine N communique avec le routeur G en utilisant son adresse NA (3.2.3c

et 3.2.3g).

$$
\begin{aligned}
connectivity(S,T,P,C) \leftarrow & && \text{(3.2.3a)}\\
& \wedge route(S,T,R) && \text{(3.2.3b)}\\
& \wedge R = [Gwf|\_] \wedge node\_addr\_to\_gw(T,DA,Gwf.n) && \text{(3.2.3c)}\\
& \wedge Seg.da = DA \wedge Seg.dp = P && \text{(3.2.3d)}\\
& \wedge track\_all\_segments(Seg,R,Cs) && \text{(3.2.3e)}\\
& \wedge Cl = lastelement(Cs), Gwl = lastelement(R) && \text{(3.2.3f)}\\
& \wedge node\_addr\_to\_gw(S,Cl.sa,Gwl.n) && \text{(3.2.3g)}\\
& \wedge C = [(S,R_n,Cs_{n+1}),(R_n,R_{n-1},Cs_n)\ldots,(R_1,T,Cs_1)] && \text{(3.2.3h)}
\end{aligned}
$$

Le prédicat auxiliaire *track_ all_ segments(F,R,Cs)* détermine l'ensemble de ces métadonnées *Cs* pour tous les segments situés entre deux routeurs consécutifs sur la route *R*. Le prédicat *connection_ segment(GW,Cb,Ca)* (3.2.4e) exprime que *Cb* sont les métadonnées de la connexion avant le passage par le routeur *GW* et *Ca* sont celles après passage du routeur, après application des règles de DNAT, de filtrage et de SNAT. Dans cet algorithme, *GW* est un dictionnaire comprenant l'identifiant d'un routeur et l'interface réseau de sortie utilisée pour le routage.

$$
\begin{aligned}
track\_all\_segments(C,[],[C]). & && \text{(3.2.4a)}\\
track\_all\_segments(Clast,[Ri|L],[Clast|Prec]) \leftarrow & && \text{(3.2.4b)}\\
& \wedge connection\_segment(Ri,Cbefore,Clast) && \text{(3.2.4c)}\\
& \wedge track\_all\_segments(Cbefore,L,Prec) && \text{(3.2.4d)}\\
connection\_segment(Gw,Cb,Ca) \leftarrow & && \text{(3.2.4e)}\\
& \wedge apply\_dnat(Gw.n,Cb,Ci) && \text{(3.2.4f)}\\
& \wedge not\_filtered(Gw.n,Ci) && \text{(3.2.4g)}\\
& \wedge apply\_snat(Gw.n,Ci,Ca) && \text{(3.2.4h)}\\
& && \text{(3.2.4i)}
\end{aligned}
$$

Les prédicats présentés au listing 3.2.5 permettent de résoudre les translations d'adresses et de ports, que ce soit pour du DNAT ou du SNAT. Le prédicat *apply_ snat(R,Ci,Cf)* (3.2.5a) est vrai lorsque les métadonnées *Ci* sont translatées en *Cf* par le routeur *R*.

$$
\begin{aligned}
apply\_snat(R,Ci,Co) \leftarrow & snat(R,Ci.sa,Ci.sp,Co.sa,Co.sp) && \text{(3.2.5a)}\\
& \wedge Ci.da = Co.da \wedge Ci.dp = Co.dp && \text{(3.2.5b)}\\
apply\_snat(\_,C,C). & && \text{(3.2.5c)}\\
apply\_dnat(R,Ci,Co) \leftarrow & dnat(R,Ci.da,Ci.dp,Co.da,Co.dp) && \text{(3.2.5d)}\\
& \wedge Ci.sa = Co.sa \wedge Ci.sp = Co.sp && \text{(3.2.5e)}\\
apply\_snat(\_,C,C). & && \text{(3.2.5f)}\\
& && \text{(3.2.5g)}
\end{aligned}
$$

Les limites de la modélisation du NAT sont discutées dans la sous-section 3.3.1.

Le tableau 3.9 résume les modifications apportées par cette partie.

Cette sous-section a passé en revue les éléments permettant de modéliser la topologie du système (nœuds, routeurs, sous-réseaux) ainsi que les mécanismes associés (routage, filtrage, translation d'adresses). La sous-section suivante est centrée sur la modélisation de la configuration logicielle d'un nœud donné.

Modification	Type de modification	Justification
accept	ajout	inversion de la politique par défaut
snat, *dnat*	ajout	modélisation du NAT
connectivity	ajout	détermination de l'accessibilité

TABLE 3.9 – Évolution de la spécification liée aux pare-feux.

3.2.2 Cartographie

La cartographie représente les configurations logicielles propres à chaque nœud du système. La connaissance de ces paramètres joue un rôle important pour déterminer si un nœud peut être la cible de certaines catégories d'attaques. Cette section présente la modélisation des logiciels, processus et services qui sont liés à un nœud sont présentés. La question du regroupement des machines disposant de configurations équivalentes est également traitée. Cette partie se termine en abordant la question de la modélisation des machines virtuelles et des comptes utilisateurs.

A Logiciels, processus

Les prédicats restent inchangés par rapport à M4D4. Un logiciel est exprimé par *software(N,V,T,A)*, où N représente le nom du logiciel, V son numéro version, T son type et A l'architecture d'exécution. Un processus *process(S,U)* est constitué d'un logiciel (S) exécuté par un utilisateur U. L'exécution d'un processus est modélisé avec *exec(H, P)* pour traduire le fait que la machine H exécute le processus P.

B Service

La principale différence avec M4D4 est l'ajout de la prise en compte de l'interface réseau sur laquelle le service écoute. Le prédicat *listens(N,service(Process, Port, Iface))* signifie que le service dépendant du processus *Process* est en écoute sur le port *Port* sur l'interface réseau *Iface* de la machine N. Cette information permet de modéliser un service en écoute uniquement sur une interface donnée.

Modification	Type de modification	Justification
listens	ajout d'un argument	prise en compte de l'interface d'écoute

TABLE 3.10 – Évolution de la spécification des services.

Le tableau 3.10 résume les évolutions apportées.

C Classes d'équivalence

Dans des systèmes, il peut être commun de trouver une ensemble de machines au sein d'un même sous-réseau qui disposent de configurations équivalentes, d'une accessibilité équivalente et qui diffèrent uniquement par leurs identifiants (adresse IP et mac). Cette méthode de regroupement des machines en classes d'équivalence est similaire à celle mise en œuvre pour la simplification des graphes d'attaques ([ZOH11]). Deux nœuds sont équivalents s'ils disposent de la même configuration et de la même accessibilité.

$$equivalent\_node(N, N') \leftarrow \tag{3.2.6a}$$
$$\wedge\, same\_configuration(N, N') \tag{3.2.6b}$$
$$\wedge\, same\_reachability(N, N') \tag{3.2.6c}$$
$$same\_configuration(N, N') \leftarrow \tag{3.2.6d}$$
$$\wedge\, same\_processes(N, N') \tag{3.2.6e}$$
$$\wedge\, same\_services(N, N'). \tag{3.2.6f}$$
$$same\_processes(N, N') \leftarrow \{S \mid \forall S.exec(N, S)\} = \{S \mid exec(N, S)\} \tag{3.2.6g}$$
$$same\_services(N, N') \leftarrow \{S \mid \forall S.listens(N, S)\} = \{S \mid listens(N, S)\} \tag{3.2.6h}$$
$$same\_reachability(N, N') \leftarrow reach\_from(N, X), reach\_from(N', X) \tag{3.2.6i}$$
$$reach\_to(N, Y), reach\_to(N', Y) \tag{3.2.6j}$$
$$reach\_from(N, X) \leftarrow X = \{(S, P, R) \mid \tag{3.2.6k}$$
$$connectivity(S, N, P, \_) \tag{3.2.6l}$$
$$\wedge\, route(S, N, R)\}. \tag{3.2.6m}$$
$$reach\_to(N, X) \leftarrow X = \{(S, P, R) \mid \tag{3.2.6n}$$
$$connectivity(N, S, P, \_) \tag{3.2.6o}$$
$$\wedge\, route(N, S, R)\}. \tag{3.2.6p}$$
$$\tag{3.2.6q}$$

Le listing 3.2.6 exprime la définition de l'équivalence de deux nœuds N et N'. Ces nœuds doivent disposer de la même configuration (3.2.6b) et de la même accessibilité (3.2.6c). L'équivalence des configurations s'exprime comme l'égalité entre l'ensemble des processus et des services présents sur chaque nœud (3.2.6d).

Les prédicats $reachFrom(N, X)$ (3.2.6k) et $reachTo(N, X)$ (3.2.6n) donnent respectivement l'ensemble des connexions possibles vers et depuis le nœud N, où X représente l'ensemble des triplets (S, P, R) où S est un nœud, P un port et R la route entre les nœuds N et S.

La condition sur les configurations équivalentes implique également l'équivalence de la visibilité topologique des observateurs applicatifs ou locaux installés sur les machines d'une même classe d'équivalence. Les observateurs réseaux disposent également d'une visibilité topologique sur l'ensemble des nœuds d'une même classe d'équivalence du fait de la propriété d'accessibilité qui contraint une égalité des routes empruntées.

Cette définition est valable si les observateurs disposent d'une visibilité opérationnelle équivalente sur chaque machine. Cela exclut par exemple la présence de règles spécifiques à seulement une sous-partie des machines d'une classe d'équivalence. Cette limite au modèle de classes d'équivalence ne sont pas traitées, en estimant que ces cas ne sont pas courants et sont en contradiction avec une surveillance homogène d'un parc de machines équivalentes.

Détermination des classes d'équivalence Le listing 3.2.6 présente les propriétés dont doivent disposer deux machines pour caractériser leur appartenance à une même classe d'équivalence. Il reste à déterminer la manière dont ces classes sont déterminées en pratique.

Une classe d'équivalence C est définie par le fait **equivalence_class***(C)* et l'appartenance d'un nœud N à une telle classe C est caractérisée par le fait **belongs_to_class***(N,C)*. Le listing 3.2.7 illustre le fonctionnement du test d'appartenance d'un nœud à une classe d'équivalence : N appartient à la classe d'équivalence N s'il existe un nœud X équivalent à N et qui appartient à la classe d'équivalence C.

$$belongs\_to\_class(N, C) \leftarrow equivalence\_class(C) \tag{3.2.7a}$$
$$\wedge\, belongs\_to\_class(X, C) \wedge X \neq N \tag{3.2.7b}$$
$$\wedge\, equivalent\_node(N, X) \tag{3.2.7c}$$

Il suffit alors de définir une classe d'équivalence et un nœud de référence pour en déduire l'ensemble des nœuds équivalents.

Modification	Type de modification	Justification
equivalence_class	ajout	définition d'une classe d'équivalence
belongs_to_class	ajout	appartenance d'une machine à une classe d'équivalence
equivalent_node	ajout	définition de deux machines équivalentes

TABLE 3.11 – Spécification des classes d'équivalence.

Le tableau 3.11 rassemble les nouveaux prédicats utilisés pour la modélisation des classes d'équivalence.

D Machines virtuelles

Dans un cas simple, une machine virtuelle est modélisée comme un nœud physique. Cependant, cette approche ne permet pas de modéliser des attaques spécifiques aux machines virtuelles. En effet, un attaquant pourrait prendre le contrôle de la machine hôte pour accéder aux machines virtuelles qui y sont hébergées. Inversement, l'attaquant peut s'évader de l'environnement virtuel pour passer sur la machine physique ou sur d'autres applications hébergées sur le même nœud. Afin de modéliser cette dépendance, nous proposons d'utiliser les prédicats *hosts(Node1, VirtualNode)* pour indiquer que la machine représentée par *Node1* héberge une machine virtuelle représentée par *VirtualNode*.

E Utilisateurs

Dans M4D4, les utilisateurs existent uniquement lorsqu'ils sont associés à un processus. Il est intéressant d'étendre la modélisation pour permettre de spécifier des interactions liées à un utilisateur (par exemple une réception d'un courrier électronique de hameçonnage suivie d'une infection via une pièce jointe ou un lien piégé par exemple). Pour cela, le fait **is_user***(U)* exprime qu'*U* est un utilisateur et le fait **has_account***(U,M)* permet de spécifier que l'utilisateur *U* dispose d'un compte sur la machine *M*.

Modification	Type de modification	Justification
is_user(U)	ajout	modélisations des comptes utilisateurs
has_account(U,M)	ajout	modélisations des comptes utilisateurs
hosts	modification de la sémantique	modélisation des machines virtuelles

TABLE 3.12 – Utilisateurs et machines virtuelles.

Le tableau 3.12 liste les prédicats introduits.

Cette sous-partie a présenté les éléments permettant de modéliser les détails liés à un nœud et à son environnement. La topologie et la cartographie se contentent de décrire les éléments fonctionnels du système. La partie suivante propose une modélisation des actions possibles pour un attaquant sur un tel système.

3.2.3 Classes d'attaques, vulnérabilités et actions normales

L'un des objectifs de la base de connaissances est de permettre de lier les actions de l'attaquant à leurs observations. Il est donc indispensable de disposer de l'ensemble des actions à observer. Ces actions sont divisées en deux catégories qui sont les actions d'attaques et les actions normales. Nous proposons donc une description des choix de modélisation qui concernent les classes d'attaques

ainsi que les vulnérabilités qui peuvent être exploitées par ces attaques. Ensuite, les possibilités de modélisation des actions normales sur un système sont étudiées.

A Classes d'attaques

Concernant les classes d'attaque, M4D4 fournit uniquement un mécanisme d'héritage de classes d'attaque sans proposer de peuplement de cette hiérarchie par des classes d'attaques concrètes. Il reste cependant à peupler cette hiérarchie. La hiérarchie est construite autour du prédicat *attack-class(K)* et *inherits(K',K)* pour exprimer le fait que *K'* est une sous-classe directe de la classe *K*. Le prédicat *attacksubclass(K1, K)* permet de caractériser *K1* comme une sous-classe (directe ou indirecte) de *K*. Dans M4D4, une classe d'attaque *C* peut être associée à une vulnérabilité exploitée *V* via *exploits(C,V)*. Cette relation permet de lier les classes d'attaques et les vulnérabilités.

Comme indiqué au paragraphe précédent, un peuplement de la hiérarchie de classes d'attaques manque. CAPEC [3] (Common Attack Pattern Enumeration and Classification) est un point de départ pour l'établissement d'une hiérarchie d'action malveillantes. Les attaques y sont en effet présentées hiérarchiquement. CAPEC sert de point de départ et de référence, mais ne limite pas nos choix de modélisation. Il est possible d'ajouter ou de raffiner certaines classes, d'ajouter de nouvelles classes (aux racines ou aux feuilles), d'utiliser de l'héritage multiple pour caractériser plusieurs aspects d'une même classe d'attaque.

Le listing 3.1 donne un aperçu d'une partie des faits traduisant certaines classes d'attaques importantes issues de CAPEC. La mise en œuvre du mécanisme d'héritage est illustrée par le listing 3.2. La réalisation de ce travail est longue (CAPEC référence environ 400 classes d'attaques) mais il est seulement nécessaire de le réaliser une seule fois car cette description est indépendante du système. Par contre, il est possible d'ajouter de nouvelles catégories ou de définir des nouvelles relations d'héritage au cas par cas.

Listing 3.1 – Faits Prolog définissant des classes d'attaques adaptées de CAPEC.

```
attack_class(code_injection).
attack_class(file_system_function_injection_content_based).
attack_class(command_line_execution_through_sql_injection).
attack_class(data_excavation_attacks).
attack_class(data_interception_attack).
attack_class(sniffing_attacks).
attack_class(functionality_misuse).
attack_class(abuse_of_communiation_channels).
attack_class(software_integrity_attacks).
attack_class(malicious_software_download).
attack_class(gather_information).
attack_class(footprinting).
attack_class(port_scanning).
```

Listing 3.2 – Faits Prolog définissant les relations d'héritage entre les différentes classes d'attaques.

```
inherits(port_scanning,footprinting).
inherits(footprinting,gather_information).
inherits(ressource_location_attacks,ressource_manipulation).
inherits(input_data_manipulation,ressource_manipulation).
inherits(path_traversal,ressource_manipulation).
inherits(infrastructure_manipulation,ressource_manipulation).
inherits(file_manipulation,ressource_manipulation).
inherits(buffer_attacks,data_structure_attack).
inherits(integer_attacks,data_structure_attack).
inherits(pointer_attack,data_structure_attack).
inherits(privilege_escalation,exploitation_of_privilege_trust).
```

3. http ://capec.mitre.org/

B Vulnérabilités

Les vulnérabilités sont caractérisées dans M4D4 par le prédicat *vulnerability(V)*, qui affecte une configuration logicielle C (*affects(V, C)*). Cette configuration logicielle vulnérable C est définie par les logiciels S qui la composent (*takespartin(S, C)*). M4D4 définit la règle *not_ vulnerable(H,V)* qui exprime le fait que le nœud H n'est pas vulnérable à la vulnérabilité V si toutes les configurations affectées par V mettent en jeu au moins un produit non présent sur H. Cette règle implique une comparaison qui dépend fortement de la sémantique des numéros de versions de logiciels.

Dans notre approche, la connaissance de ces vulnérabilités permet principalement d'éliminer les signatures d'IDS qui détectent des attaques ciblant des vulnérabilités n'affectant pas une machine donnée.

C Actions normales

Une action normale correspond à une action qui, lorsqu'elle est prise individuellement, est légitime mais peut également constituer un maillon d'un scénario d'attaque. Il est donc important de les modéliser au même titre que les actions qui correspondent à des classes d'attaques. Différents dictionnaires ou taxinomies existent pour décrire les événements normaux qui peuvent survenir dans un système.

C.1 CEE (Common Event Expression) [4] est un standard ayant pour objectif la normalisation des journaux d'événements. Une taxinomie est définie et contient une liste des différentes actions que l'on retrouve usuellement sur des systèmes. Ces noms d'actions permettent de caractériser des actions de type lectures/ écritures (*read/ write/ modify/ copy/ execute/ access/ move*), des actions d'authentifications (*login/ logout*), des actions mettant en jeu les connexions réseaux (*upload/ download/ initiate / connect/ disconnect*), des actions sur l'environnement (*install/ uninstall/ update/ suspend*) ou sur des services (*start/ stop*). Il faut cependant noter que toutes ces actions n'ont pas la même granularité. Par exemple, l'action *access* peut être un *read*, *write* ou un *exec*, un *move* peut être décomposé en un *copy* et *delete*.

C.2 CYBOX (Cyber Observable Expression) [5] propose également une énumération d'actions standard (*cyboxVocabs :ActionNameEnum*). Les noms d'actions proposés sont à un niveau d'abstraction très bas (*getSystemTime, getLibraryHandle, getFileAttributes, create Thread...*) qui ne sont pas nécessairement pertinents pour un processus de corrélation d'alertes de haut niveau. De plus, ces actions ne sont pas hiérarchisées et sont donc présentées à plat. Par contre, il existe également une metaclasse *cyboxVocabs :ActionTypeEnum* qui recense des types d'actions. Ces actions sont plus génériques et sont comparables à celles issues du dictionnaire d'action de CEE. Le type *cyboxVocabs :EventTypeEnum* énumère les types d'événement d'un point de vue le leur origine (*Registry Ops, Service Mgt, DNS lookup, HTTP Traffic, Authentication Ops...*). Le modèle recense également (dans *cyboxVocabs :EventTypeEnum*) les types d'outils pouvant générer des observables (NIDS, NIPS, HIPS, Firewall, Router, Proxy, Gateway, SIM, Vulnerability Scanner...).

C.3 XDAS (Distributed audit service) [6] est un standard ayant pour objectif de définir un format pour les rapports d'événements. Ce standard propose une taxinomie incluant des actions de gestions de comptes (création, activation, suppression, modification), des actions liées à la gestion des sessions (création, fermeture), à la gestion d'objets (création, suppression, requête, modification), à la gestion des services (installation, activation, désactivation, invocation, requête...), à la gestion des accès à une ressource ou encore aux événements exceptionnels (démarrage du système, corruption, épuisement d'une ressource).

C.4 Choix du dictionnaire d'actions Au vu des différents formalismes et dictionnaires disponibles pour caractériser les actions normales d'un système, nous proposons d'utiliser la liste des actions du dictionnaire de CEE qui semble être à la fois la plus pertinente du point de vue du

4. http ://cee.mitre.org
5. https ://cybox.mitre.org
6. http ://openxdas.sourceforge.net/

niveau d'abstraction utilisé et qui offre un bon compromis entre exhaustivité et complexité. De plus, ce choix permet de construire une base d'actions normales et ne doit pas être limitatif. Il est donc possible d'étendre ce dictionnaire avec de nouvelles actions si nécessaire. Une action normale est définie par le prédicat *action_ class(A)*. Le listing 3.3 illustre le résultat de cette modélisation pour quelques actions.

Listing 3.3 – Faits Prolog définissant des actions normales issues de CAPEC.

```
action_class(write).
action_class(read).
action_class(initiate).
action_class(login).
action_class(logout).
action_class(logout).
action_class(connect).
action_class(install).
action_class(uninstall).
action_class(start).
action_class(stop).
action_class(upload).
action_class(download).
```

Discussion Le choix de CAPEC comme référence pour constituer une hiérarchie de classes d'attaques a comme avantage de fournir un ensemble de données important. Cependant, certains choix de classification sont critiquables. Par exemple, les classes racines peuvent représenter des domaines d'attaques ou bien des mécanismes d'attaques, ce qui rend la recherche d'une classe d'attaque donnée non intuitive. De même, il est intéressant de pouvoir identifier des classes d'attaques non pas selon leurs domaines d'appartenance ou selon les types de mécanismes mis en œuvre, mais par leurs effets (exécution de code arbitraire, élévation de privilège). Il faut alors considérer CAPEC comme un point de départ qui peut être étendu en fonction des besoins. Par exemple, l'ajout d'une nouvelle classe racine modélisant les *attaques permettant d'exécuter du code arbitraire* permet d'ajouter une relation d'héritage entre cette classe nouvellement introduite les classes d'attaques pertinentes.

Cette sous-section a présenté les éléments de modélisation des actions possibles pour un attaquant sur un système. Certaines actions sont usuellement considérées comme normales en dehors de tout contexte alors que d'autres sont directement identifiables comme malveillantes. Dans chacun des cas, un dictionnaire de référence a été sélectionné pour permettre de nommer une action. La sous-partie suivante présente la modélisation des éléments de détection susceptibles de détecter la réalisation de ces différentes actions.

3.2.4 Éléments de détection

Cette sous-section présente la modélisation ainsi que les problématiques liées aux équipements de détection. Les différents types d'équipements sont tout d'abord présentés suivis de leurs capacités de détection. Il est ensuite question des messages levés par ces équipements. Enfin, une partie est dédiée aux problématiques de modélisation de la détection en présence de flux chiffrés.

A Nature des observateurs

Dans M4D4, tout composant de sécurité du système pouvant générer des messages est modélisé par le prédicat *analyser(A)*. Un raffinement permet ensuite de différencier les IDS, les pare-feux et les scanners de vulnérabilités. Par contre, nous cherchons également à prendre en compte des composants générant des événements utiles à la corrélation mais qui ne sont pas nécessairement considérés comme des outils de sécurité. Le prédicat **observer***(O)* permet alors de caractériser un tel élément, qu'il soit une sonde fonctionnelle ou un IDS spécialisé.

IDS Un observateur qui est un IDS est caractérisé en plus par le fait *ids(O)*. La hiérarchie de type d'IDS proposée dans M4D4 est conservée (HIDS, NIDS ou AIDS respectivement pour IDS hôte, IDS réseau et IDS applicatif).

B Visibilités des observateurs

La capacité d'un observateur à détecter une action est modélisée par deux types de visibilités : la visibilité topologique et la visibilité opérationnelle.

B.1 Visibilité topologique La visibilité topologique caractérise la capacité de détection liée à la position de l'observateur au sein d'un système et à sa source de données. Ainsi, un observateur applicatif O peut surveiller un produit S spécifique sur une machine M (*monitors(O,M,S)*), un observateur hôte O peut potentiellement surveiller une machine M donnée (*monitors(O,M)*). Un observateur réseau O peut surveiller un sous-réseau Net (*monitors(O,Net)*) ou bien être placé sur un routeur R et ainsi voir le trafic qui y est routé sur une interface $Iface$ particulière (*monitors(O, R, Iface)*). La visibilité topologique des IDS réseaux de M4D4 est définie au niveau d'un routeur ou bien entre deux routeurs. Comme expliqué par la figure 3.1, cette information peut dans certains cas ne pas être suffisante pour modéliser un observateur écoutant sur le port mirroring d'un switch.

Le but de cette modélisation est de déterminer les observateurs placés de manière adéquate pour la détection d'une action entre deux nœuds (distincts ou non) du système. Si les nœuds sont confondus, alors un observateur hôte ou applicatif est potentiellement pertinent. Si les nœuds sont distincts, les observateurs pertinents sont :

- les observateurs hôtes et applicatifs de chacun des deux nœuds
- les observateurs situés sur la (ou les) routes reliant ces deux nœuds. Ceci inclut :
  - les observateurs placés au niveau des routeurs (ou des pare-feux) et qui écoutent sur une des interfaces réseau utilisées par le routage
  - les observateurs qui surveillent un sous-réseau traversé

Les observateurs réseaux O dont la visibilité topologique permet de surveiller une communication entre S et T sur le port P sont sélectionnés par le prédicat *can_detect(O, S, T, P)* comme illustré au listing 3.2.8. Ce prédicat fait appel à *detects_in_path(O, Cs)* pour permettre de déterminer la capacité de l'observateur O à surveiller au moins un des segments de la liste Cs, où chaque segment définit les métadonnées du flux (ports et adresses) entre deux interfaces de routeurs particuliers.

Les trois cas correspondent respectivement à la présence d'un observateur sur un sous-réseau traversé (3.2.8c), sur l'interface de sortie d'un routeur traversé (3.2.8e) et sur l'interface d'entrée d'un routeur (3.2.8f). Pour rappel, la liste Cs contient des triplets (Gw_i, Gw_{i+1}, C), où C représente les métadonnées du flux entre Gw_i et Gw_{i+1}, avec $Gw_i = (G, Iface)$ qui représente un couple identifiant du nœud et interface réseau de sortie utilisée pour le routage. Les _ représentent des variables muettes.

$$can\_detect(O,S,T,P) \leftarrow connectivity(S,T,P,Cs) \tag{3.2.8a}$$
$$\wedge\, detects\_in\_path(O,Cs) \tag{3.2.8b}$$
$$detects\_in\_path(O,Cs) \leftarrow monitors(O,Net) \wedge nodenet(G,iface,Net) \tag{3.2.8c}$$
$$\wedge\, (\_,(G,iface),\_) \in Cs \tag{3.2.8d}$$
$$detects\_in\_path(O,Cs) \leftarrow monitors(O,G,iface) \wedge (\_,(G,iface),\_) \in Cs \tag{3.2.8e}$$
$$detects\_in\_path(O,Cs) \leftarrow monitors(O,G,iface) \wedge nodenet(G,iface,Net) \tag{3.2.8f}$$
$$\wedge\, nodenet(G',iface',Net) \wedge ((G',iface'),(G,\_),\_) \in Cs \tag{3.2.8g}$$

La figure 3.2 illustre les différentes configurations possibles concernant la position d'un observateur. Dans ce système, un pare-feu jouant également le rôle de routeur connecte deux sous-réseaux *net1* et *net2*. L'observation la plus limitée est réalisée au sein d'une unique application de la machine *n*. La visibilité d'un tel observateur est limitée aux seules actions qui impliquent le service ou le processus local surveillé. Un HIDS placé au sein de la machine n dispose d'une visibilité par

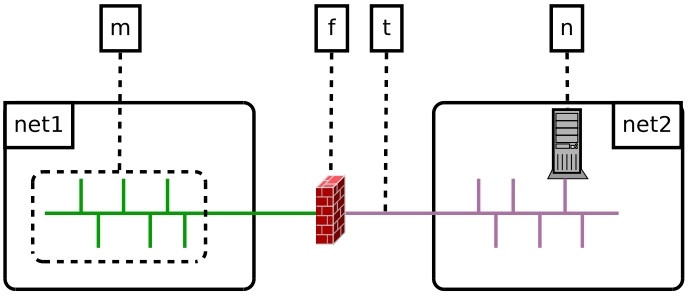


FIGURE 3.2 – Différentes configurations possibles pour illustrer la visibilité topologique.

défaut sur l'ensemble de ce nœud. Il est cependant possible de restreinte la visibilité à un sous-ensemble des processus ou services de ce nœud. Le pare-feu f est un observateur réseau lorsqu'il journalise des événements de connexion. Sa visibilité topologique englobe les flux échangés entre les sous-réseaux *net1* et *net2*. Un IDS placé au niveau d'un tap réseau t ou bien en coupure au même emplacement dispose d'une visibilité topologique équivalente à f. Cet observateur ne peut cependant pas surveiller les flux entre deux machines du même sous réseau (sauf dans le cas où des hubs sont utilisés). Cette surveillance des échanges entre deux machines d'un même sous-réseau est possible pour l'observateur m dont la visibilité englobe l'intégralité d'un sous-réseau. L'observateur m est par exemple un NIDS relié au port d'écoute d'un switch. Le tableau 3.13 résume ces possibilités.

positionnement sur la figure 3.2	Observateur potentiels	visibilité topologique
n	AIDS	une application de n
n	HIDS	(Sous)-ensemble des processus et services n
f, t	pare-feu, NIDS, NIPS	Flux entre *net1* et *net2*
m	NIDS	Flux entre *net1* et *net2*, flux intra-*net1*

TABLE 3.13 – Observateurs potentiels du système de la figure 3.2.

B.2 Visibilité opérationnelle (ou fonctionnelle) La visibilité opérationnelle correspond aux actions visibles pour un observateur du fait de sa configuration et de sa méthode de détection. Dans notre cas, un observateur peut détecter des actions normales ou des actions d'attaque. La visibilité opérationnelle s'exprime de la manière suivante : chaque observateur O dispose au moins d'une configuration de détection *Conf* (**detection_conf***(O,Conf)*) lui permettant de détecter un ensemble de classes d'attaques ou d'actions normales. Une configuration de détection peut être directement associée à des actions détectables ou bien passer par l'intermédiaire de signatures selon le mode de fonctionnement de l'observateur.

(a) KBIDS : Ces observateurs reposent sur l'utilisation de signatures pour détecter des exploitations de vulnérabilités connues. Le prédicat *active(Conf,S)* exprime l'activation de la signature S dans la configuration de détection *Conf*. Dans M4D4, une signature est associée à une classe d'attaque (*detects(Sig,AC)*). Le défaut de cette modélisation est qu'il est nécessaire de disposer d'autant de classes d'attaques que de vulnérabilités différentes afin de pouvoir sélectionner de manière fine les signatures appropriées. Un raccourci d'écriture consistant à affecter directement une signature à une vulnérabilité (*detects(Sig,Vuln)*) devient alors un choix de modélisation pratique. Il faut alors comprendre que la signature permet de détecter non pas la vulnérabilité

mais son exploitation. Cela évite d'avoir à créer une sous-classe d'attaques spécifique pour chaque vulnérabilité.

(b) ABIDS : les observateurs O disposent de configuration permettant de détecter certaines classes d'attaques. Ils sont identifiés par le fait *abids(O)*. La configuration de détection d'un ABIDS est directement associée à une classe d'attaque susceptible d'être détectée. Toutes les sous-classes sont alors considérées comme détectables. Par exemple, un IDS O capable de détecter tout type d'injection SQL se spécifie par *detects(O, sql-injection)*.

Cette introduction de la configuration de détection ajoute une couche d'abstraction supplémentaire (dans M4D4, un IDS est directement associé à des signatures ou à des classes d'attaques détectables) et permet de définir simplement plusieurs instances d'un même type d'observateur utilisant des configurations identiques et pouvant ainsi détecter le mêmes types d'actions.

L'exemple donné au listing 3.4 met en jeu un IDS identifié par *nids1* qui dispose d'une configuration de détection pour laquelle deux règles de détection sont actives. Ces règles permettent de détecter l'exploitation d'une vulnérabilité spécifique nommé ici *php_cgi_command* qui correspond à une injection de commande PHP. Cette vulnérabilité est associée à la classe d'attaque *command_delimiter* (qui est elle-même une sous-classe de la classe injection).

Listing 3.4 – Exemple de configuration de détection pour un IDS.

```
detection_conf(nids1, snort_config).
active(snort_config,'22063').
active(snort_config,'22064').
detects('22063',php_cgi_command).
detects('22064',php_cgi_command).
exploits(command_delimiters,php_cgi_command).
```

B.3 Visibilités des observateurs autres que des IDS Ces types d'observateurs (ex : les loggers associés à des services fonctionnels) disposent d'une modélisation similaire aux IDS. Leur visibilité topologique est similaire à celle des IDS : elle se limite à une application (ex : serveur SSH ou serveur web), à un nœud (auditd) ou à un ensemble de flux potentiel (pare-feu). La visibilité fonctionnelle fait référence aux actions normales détectables avec la configuration dont dispose les observateurs. Par exemple, dans une configuration standard, un processus SSH journalise les connexions, ce qui se modélise par *detection_conf(sshd, ssh_info), detects(ssh_info, connection)*.

Modification	Type de modification	Justification
observer	ajout	notion générique d'observateur
monitors	ajout d'un argument	prise en compte de l'interface
detection_conf	ajout	factorisation des configurations communes

Table 3.14 – Évolution des prédicats concernant les observateurs.

Le tableau 3.14 récapitules les prédicats introduits ou modifiés.

C Informations générées par les observateurs

La spécification des messages attendus par le corrélateur de la part des différents observateurs est un élément important qui n'est pas présent dans M4D4. Chaque observateur génère un message dont le format et la structure (sémantique et présence des différents champs) sont connus. Un observateur O génère un message contenant le champ F lors de la détection d'une action A, ce qui est modélisé par le prédicat **provides_field***(O,F,A)*. Par exemple, un observateur *s1* pour lequel le champ indiquant le port cible est toujours présent dans ses messages est représenté par le prédicat *provides_field(s1,'tgt.port',_)*.

Une autre information importante à spécifier est le format dans lequel le message est généré. Le nom du format *Form* utilisé par l'observateur O est donné par le prédicat **message_format***(O, Form)*. Cette information est pertinente dans les cas où les observateurs n'utilisent pas de format

unifié pour leurs messages. Il est important de noter que l'on ne définit ici que le nom du format. L'organisation réelle des champs au sein d'un message doit être connue par le corrélateur utilisant les règles de corrélation. Le nom du format lui permet ainsi de sélectionner le décodeur adapté.

Le listing 3.5 illustre ces aspects en décrivant un NIDS capable de fournir des informations sur les champs identifiants et ports (cibles et sources) pour tous les types d'actions et un HIDS pouvant affecter le champ correspondant au nom du fichier cible d'une attaque sur l'intégrité. Ces observateurs génèrent des messages dans des formats différents spécifiques à chaque IDS.

Listing 3.5 – Quelques exemples de faits modélisant les champs disponibles et le format de messages généré par des observateurs.

```
provide_field(nids1,'src.id',_).
provide_field(nids1,'tgt.id',_).
provide_field(nids1,'tgt.port',_).
provide_field(ossec,'tgt.filename',software_integrity_attacks).
message_format(nids1,snort_fast).
message_format(ossec,ossec_xml).
```

Modification	Type de modification	Justification
provides_field	ajout	présence de champs dans les messages
message_format	ajout	format utilisé par l'observateur pour générer le message

TABLE 3.15 – Évolution des prédicats concernant les observateurs.

Le tableau 3.15 contient les prédicats introduits dans cette partie.

3.2.5 Résumé

Cette section a développé les différents éléments permettant de construire une base de connaissances regroupant les informations nécessaires à la génération de règles de corrélations. Ces éléments permettent de disposer d'une vision globale du système à surveiller. Entre particulier, tous ces éléments permettent de déterminer si la réalisation de certaines actions est possible et lorsque cela est le cas, de quelle manière ces actions seront visibles étant donné les caractéristiques des observateurs placés dans le système. Cependant, certains éléments de modélisation sont simplifiés (en particulier les pare-feux) et ne permettent donc pas de reproduire fidèlement l'intégralité des comportements du système réel. Ces éléments sont importants pour réaliser une simulation réaliste du système. Comme notre objectif n'est pas de réaliser une telle simulation, les choix réalisés permettent d'atteindre un point d'équilibre entre la simplicité d'utilisation et la modélisation de cas complexes.

3.3 Problématiques liées à la base de connaissances

Cette section a pour objectif de mettre en avant certains points importants directement liés à la base de connaissances. Le premier concerne la convention de nommage utilisée pour identifier les machines. Le second point concerne la création et la mise à jour constante d'une telle base de connaissances.

3.3.1 Problématiques liées à la normalisation des identifiants des machines

Cette sous-section a pour objectif de mettre en avant une problématique importante concernant l'identification des nœuds du réseau. Même si le cœur du travail porte sur la reconnaissance de scénario d'attaque par des méthodes explicites, il dépend de la normalisation des messages et des alertes effectuée en amont. Une première partie expose l'importance et les problèmes liés à l'identification des machines, une seconde traite du problème d'identification particulier posé par

les serveurs mandataires et une dernière partie propose un compromis pour résoudre ce problème d'identification.

A Identification non ambiguë des nœuds

La capacité d'identification des machines de manière fiable est un prérequis important à l'identification de scénarios d'attaque. Un message généré par un observateur peut en effet identifier un nœud de différentes manières : par l'une de ses adresses (IP, MAC), par un identifiant propre à l'observateur ou encore par un nom de machine. Certains dispositifs (serveur applicatif ou NAT notamment) peuvent modifier les adresses source ou cible des paquets qui transitent sur le réseau. Sans mise en place de normalisation, il peut devenir difficile de déterminer l'origine ou la destination d'un événement. Cette identification doit être réalisée dans un (ou plusieurs) dispositif(s) de normalisation. Il s'agit de l'étape de normalisation qui fait partie du processus de corrélation introduit en 1.1.2. L'objectif est de substituer à chaque terme se référant à une machine particulière un identifiant unique permettant de connaître sans ambiguïtés les nœuds du système concerné. Dans notre cas, cet identifiant peut par exemple être celui utilisé avec le prédicat *node* pour définir les nœuds du système (*node(Id)*). Il est également possible d'utiliser le nom d'une machine avec le prédicat *nodename(H, Name)* et d'associer une adresse à un nom avec *resolve(Addr,Name)* (ces deux prédicats sont définis dans M4D4). Cette normalisation est particulièrement utile dans le cas de réseaux utilisant des dispositifs qui modifient de manière dynamique les adresses des nœuds (que ce soit directement comme c'est le cas avec le protocole DHCP ou lors du routage avec du NAT dynamique ou des serveurs mandataires).

L'objet de la thèse n'est pas de s'attarder sur ce module de normalisation, mais il est fait l'hypothèse de son existence dans les cas où le système à surveiller est complexe et modifie dynamiquement les adresses des machines.

Pour le NAT dynamique (masquerading par exemple), l'association entre les adresses initiales et translatées est connu du dispositif qui réalise cette opération et il est nécessaire que le module de normalisation récupère cette information.

B Serveurs mandataires

L'utilisation d'un serveur mandataire a plusieurs conséquences sur les flux et leur vision par des observateurs : (1) il modifie les adresses du flux (2) il restreint potentiellement les connexions (3) le serveur mandataire peut également jouer le rôle d'observateur applicatif ou réseau selon les cas.

Un serveur mandataire est modélisé partiellement en considérant qu'il s'agit d'un routeur réalisant du SNAT et du DNAT et réalisant également des fonctions d'un pare-feu en acceptant uniquement certaines connexions. Les échanges possibles se résument ainsi : un client envoie explicitement ses requêtes au serveur mandataire qui transfère la requête vers la ressource demandée ou la renvoie si cette dernière est en cache, ce qui complexifie d'autant plus la modélisation des flux. Pour une communication entre un client d'adresse C et un serveur d'adresse S, l'utilisation d'un serveur mandataire explicite d'adresse *Proxy* se traduit par :

```
AMONT du serveur mandataire :   (Source : C, Destination : Proxy)
AVAL du serveur mandataire :  (Source : Proxy, Destination : S)
```

Cela peut donc se modéliser par du DNAT et du SNAT.

Dans le cas d'un serveur mandataire transparent, les sources et destinations des flux d"'informations deviennent :

```
AMONT du serveur mandataire :  (Source : C, Destination : S)
AVAL du serveur mandataire : (Source : Proxy, Destination : S)
```

Le serveur mandataire peut alors être modélisé comme un routeur réalisant du SNAT.

L'origine (le client réel) d'une requête relayée par un serveur mandataire est difficilement identifiable par un observateur placé en aval de ce même serveur mandataire. Les informations nécessaires à cette identification sont normalement connues du serveur mandataire et visibles dans ses journaux. Dans le cas des serveurs mandataires HTTP, cette indication peut être fournie par le champ

x-forwarded-for. Cependant, cette caractéristique ne saurait être une solution entièrement satisfaisante au problème de l'identification de l'origine de la requête. En effet, ce champ n'est pas obligatoirement renseigné par le serveur et il est de plus possible de se retrouver dans une configuration constituée d'un enchaînement de serveurs mandataires ayant pour conséquence un masquage de l'identité du client à l'origine de la requête.

C Choix de modélisation

Cette partie liée aux identifiants des nœuds est rendue complexe principalement à cause des équipements réalisant des associations entre différents identifiants de manière dynamique (NAT dynamique, serveur mandataire), éliminant la possibilité d'une modélisation statique. Par conséquent, dans notre approche, les hypothèses suivantes sont posées :

— dans le cas où un équipement permet de normaliser les identifiants des nœuds en tout point où des observateurs sont présents, c'est cet identifiant normalisé qui est utilisé pour identifier les nœuds dans la règle de corrélation ;
— si un tel dispositif n'est pas mis en place, l'adresse IP des machines est utilisée. Pour les machines qui disposent de plusieurs adresses IP, il est nécessaire d'en privilégier une.

3.3.2 Initialisation et maintenance de la base de connaissances

La mise en place du peuplement et de la maintenance automatique de la base de connaissances n'est pas traitée dans cette thèse et est donc à mettre sur la liste des travaux futurs. La base de connaissances rassemble des données reflétant différents niveaux de granularité et de volatilité. En pratique, les données peu volatiles (celles dont la validité est peu dépendante de la configuration du système) peuvent être rentrées manuellement. Cependant, ce n'est pas le cas des données plus volatiles qui doivent être idéalement renseignées de manière automatique pour une intégration des évolutions fréquentes de l'état du système.

L'initialisation de la base de connaissances demande dans un premier temps une définition de l'infrastructure générale du système (sous-réseaux et routeurs). Vient ensuite le peuplement avec l'ensemble des nœuds qui sont placés au sein de cette infrastructure. Une automatisation partielle de cette étape est théoriquement possible en réalisant une exportation des informations contenues dans la base de données d'un logiciel de gestion de parc. Ce dernier contient usuellement la liste des machines déployée, des versions des logiciels installées et des services mis en place.

Les composants de la base de connaissances ne sont pas tous affectés de la même façon par des opérations de maintenance. Le tableau 3.16 donne une classification possible de la manière dont différentes catégories d'éléments sont affectées.

Volatilité	éléments
haute	services, logiciels, vulnérabilités
moyenne	règles de détection, nœuds, règles de filtrage
basse	observateurs
très basse	classes d'attaques

TABLE 3.16 – Volatilité des éléments de la base de connaissances.

La maintenance automatique de la base de connaissances suppose de pouvoir déduire les informations voulues de manière active via par exemple la mise en place d'agents spécifiques chargés de transmettre les informations sur les nouvelles versions de logiciels ou services mis en place, ou bien de manière passive à partir d'une analyse des flux réseau. De plus, certains faits sont plus difficiles à obtenir de manière automatique que d'autres. Par exemple, la traduction des règles de filtrage de pare-feu contenant plusieurs centaines ou milliers de règles complexes semble non trivial a priori.

La question des flux spécifiques nécessaires à la communication entre l'ensemble des agents de maintenance de chaque sous-réseau du système et la base de connaissances est également un élément à prendre en compte et qui peut se révéler délicat à mettre en place dans le cas de réseaux isolés.

3.4 Synthèse

Ce chapitre a proposé une base de connaissances inspirée de M4D4 et en proposant une évolution l'objectif est de servir de support à la génération de règles de corrélation en proposant une interface à tous les éléments contextuels importants (topologie, cartographie, observateurs, vulnérabilités, attaques et messages). Cette base de connaissances présente une structure minimale pour permettre à la fois de décrire les caractéristiques et spécificités d'un système donné et également de proposer les requêtes ou prédicats permettant d'obtenir les informations nécessaires à la création de règles de corrélation.

L'expression de la topologie et de la cartographie permet de sélectionner les nœuds du système répondant à des caractéristiques spécifiques. Les informations qui concernent la visibilité des observateurs permettent la sélection d'observateurs pertinents pour la détection d'un type d'action en un point du système. La modélisation des vulnérabilités permet d'éliminer les signatures qui correspondent à des exploitations de vulnérabilités non présentes sur certaines machines. La liste des attaques et des actions permet de définir les catégories d'attaques détectables par chaque IDS. Les messages spécifient les formats et contenus des messages générés par les observateurs.

L'objectif du chapitre suivant est de montrer que la combinaison de toutes ces informations permet, à partir d'un scénario d'attaque donné, de construire une règle de corrélation permettant la détection du scénario d'attaque sur le système ainsi modélisé.

Chapitre 4

Procédé de génération de règles de corrélation

Comme l'a montré le chapitre 2, il est possible d'exprimer des règles de corrélation sous diverses formes. Ces représentations des règles de corrélations partagent toutes des caractéristiques communes. Elles permettent ainsi de filtrer des messages en fonction des valeurs de leurs champs, d'exprimer des contraintes entre les valeurs de plusieurs champs, de définir un ordonnancement des messages attendus ou de spécifier des contraintes temporelles. Cependant, les règles sont obtenues manuellement ou bien sont dérivées de graphes d'attaques. Dans le premier cas, le travail est fastidieux et demande une analyse approfondie de la pertinence et la justesse des règles à chaque évolution du système. Dans le second cas, les règles sont automatiquement générées par le moteur de construction de graphes d'attaques. La structure du scénario est ainsi entièrement déterminée à partir des informations présentes sur les vulnérabilités connues du système, avec une redondance et une complexité élevée. De plus, ces graphes d'attaques ne prennent pas en compte les capacités réelles de détection des observateurs du système et sont constitués d'un enchaînement d'exploits. Notre contribution consiste à proposer un mécanisme permettant de générer des règles de corrélation spécifiquement adaptées au système en partant d'une spécification de scénarios d'attaque haut niveau et la plus générique possible.

4.1 Procédé de génération

La génération des règles de corrélation proposée ici a pour but de produire des règles de corrélation adaptées à un système donné. De plus, ce procédé doit permettre de regénérer simplement des règles de corrélation à partir d'un même scénario d'attaque lorsque le système évolue. Ce processus suppose de disposer au préalable d'un ensemble de scénarios d'attaque (génériques ou issus d'une étude de risques sur un système donné) et d'une base de connaissances décrivant le système surveillé. La structure et les éléments de cette base de connaissances sont exposés dans le chapitre 3. Le processus de génération des règles est décomposé en une succession d'étapes dont la première est manuelle tandis que toutes les autres sont automatiques.

Étape initiale (manuelle) Le point de départ met en jeu un arbre d'attaque décrivant un scénario d'attaque. Ce dernier peut être issu d'une étude de risque du système d'information cible. Un arbre d'attaque reste cependant une structure informelle faisant référence à des objectifs et sous-objectifs de l'attaquant. Le rôle de la première étape est d'enrichir cette structure en explicitant les informations sur la nature des actions associées aux sous-objectifs ainsi que la source et la cible de chaque action. Cet enrichissement doit permettre par la suite de réaliser un traitement automatique de la structure enrichie. Cet enrichissement est effectué à l'aide d'un langage d'actions dont la description est l'objet de la section 4.3. L'arbre d'attaque initial référence des sous-objectifs alors que l'arbre d'actions, qui est le résultat de cette étape initiale, décrit les actions élémentaires devant être réalisées pour atteindre ces objectifs.

Étapes de génération automatiques Dans une deuxième étape, les acteurs sont identifiés automatiquement. Ces acteurs correspondent à des machines, services, utilisateurs pouvant participer aux actions du scénario d'attaque spécifiées lors de l'étape précédente. Cette identification est conditionnée par les informations contenues dans la base de connaissances. Dans une troisième étape, les observateurs (sondes fonctionnelles, IDS) sont identifiés. Ces observateurs sont les entités du système disposant de la capacité de générer des événements ou des alertes lors de la réalisation d'une action. Les capacités de détection des différents observateurs sont décrites dans la base de connaissances. Une fois les observateurs identifiés, il est possible d'en déduire automatiquement une spécification des messages qui pourront être levés. Cette identification des messages constitue une nouvelle étape du processus. La structure obtenue à l'issue de l'étape d'identification des messages est appelée arbre de corrélation. Cet arbre de corrélation constitue une règle de corrélation générique et indépendante de tout langage de corrélation. Afin de rendre la règle de corrélation générée utilisable, il est nécessaire de réaliser la traduction de cette structure générique vers un langage de corrélation concret.

La figure 4.1 illustre l'enchaînement de ces différentes étapes en distinguant les étapes automatiques des étapes manuelles.

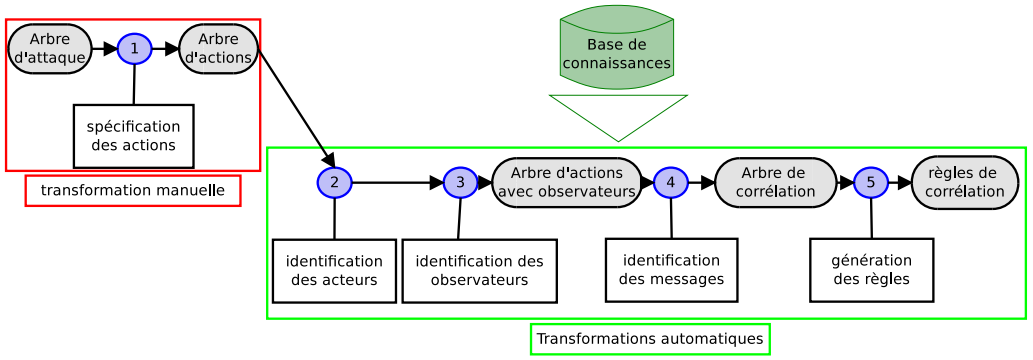


FIGURE 4.1 – Vue globale des différentes transformations permettant de générer des règles de corrélation à partir d'un arbre d'attaque représentant un scénario d'attaque.

4.2 Arbre d'attaque

La structure de départ consiste en un arbre d'attaque. Un arbre d'attaque permet de décrire un ou plusieurs scénarios d'attaque de manière informelle. Par exemple, deux nœuds distincts de l'arbre peuvent faire implicitement référence à une même machine si cette information peut être déduite intuitivement. En plus des opérateurs usuels *OR* et *AND*, il est possible d'utiliser l'opérateur séquentiel *SAND* pour spécifier un enchaînement de sous-objectifs dans un ordre déterminé [1].

La figure 4.2 donne un exemple d'un scénario d'attaque représenté sous la forme d'un arbre d'attaque. Dans le scénario exprimé, l'attaquant peut obtenir un accès sur un serveur en réalisant une attaque par injection sur une application vulnérable ou en réalisant une attaque par force brute sur l'interface d'administration d'une application web. Il réalise ensuite les deux actions suivantes dans un ordre quelconque : une mise en place d'une porte dérobée lui permettant d'interagir à distance avec la machine et une reconnaissance sur des machines internes depuis la machine compromise.

1. Les langages de corrélation introduisent souvent un opérateur de négation indiquant que certains messages ne doivent pas se retrouver dans un enchaînement pour valider un scénario. Généralement, cette négation sert à modéliser une éventuelle contre-mesure de la part d'un administrateur. Il peut être intéressant de laisser la possibilité d'introduire un opérateur de négation dans la structure de l'arbre d'attaque. Dans ce cas, cet arbre ne modélise plus uniquement la hiérarchie d'objectifs de l'attaquant, mais également les actions éventuelles des défenseurs. Cet ajout est possible, mais il n'est pas traité dans le cadre de cette thèse.

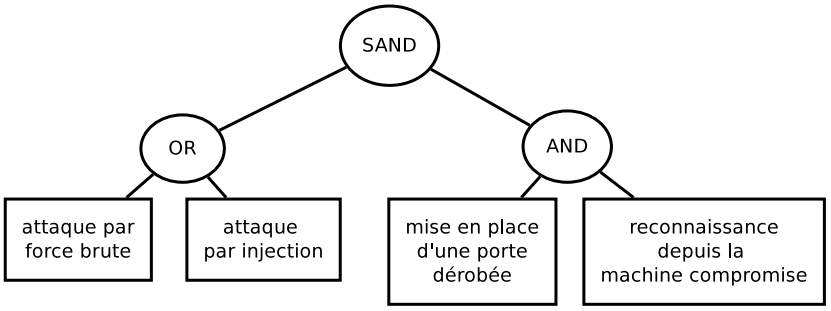


FIGURE 4.2 – Un arbre d'attaque présentant un scénario d'attaque générique.

4.3 Arbre d'actions

Un arbre d'actions est une structure qui résulte de l'enrichissement d'un arbre d'attaque permettant d'expliciter des informations implicites de cet arbre d'attaque. Cette section s'intéresse ainsi à la transformation manuelle de cette structure informelle d'arbre d'attaque en un arbre d'actions. Chaque sous-objectif de l'arbre d'attaque initial doit être traduit par une ou plusieurs actions. Si le sous-objectif requiert la réalisation de plusieurs actions, ces dernières sont organisées à l'aide des opérateurs *OR*, *AND* ou *SAND*. Les actions élémentaires sont spécifiées par l'intermédiaire d'un langage d'actions. En plus d'expliciter les éléments implicites, le langage doit permettre de déterminer si une action est observable dans le système (et comment elle est observable).

4.3.1 Langage d'actions

Cette sous-section présente une contribution liée à la définition d'un langage d'actions permettant d'enrichir un arbre d'attaque. La structure d'une action est définie dans cette sous-section. Une action consiste en un ensemble de couples champ-valeur organisés de manière hiérarchique. Une action est constituée de trois éléments principaux : son nom, sa source et sa cible. Le nom détermine le type d'action alors que la source et la cible peuvent donner à divers niveaux de détail des propriétés concernant la source de l'action et la cible de l'action. La grammaire BNF ci-dessous correspond à la description d'une action élémentaire de l'arbre d'action.

Listing 4.1 – grammaire BNF définissant la syntaxe d'une action.

```
<action-spec>           ::= <name-field> <source-field> <target-field>
<name-field>            ::= 'action:' <action-identifier>  <EOL>
<action-identifier>     ::= <Constant>
                          |  <Dynamic-variable>
<source-field>          ::= 'src.' <actor-fields>
<target-field>          ::= 'tgt.' <actor-fields>
<actor-fields>          ::= <node-id> <location-id>
                          | <node-id> <localisation-id> <additionnal-fields>
                          | <user-id> <localisation-id>
<node-id>               ::= 'id:' <VAL> <EOL>
<VAL>                   ::= <Constant>
                          | <Dynamic-variable>
                          | <Static-variable>
<location-id>           ::= 'loc:' <VAL> <EOL>
<user-id>               ::= 'username:' <VAL> <EOL>
<additionnal-fields>    ::= <service-id>
                          | <process-id>
                          | <file-id>
<service-id>            ::= 'service.' <type-id> <port-id>
                          | <port-id> <process-id>
<type-id>               ::= 'type:' <VAL> <EOL>
<port-id>               ::= 'port:' <VAL> <EOL>
<process-id>            ::= 'process.' <softype-id> <softname-id>
```

```
                          | <softype-id> <softname-id> <proc-user-id>
<softype-id>             ::= 'type:' <VAL> <EOL>
<softname-id>            ::= 'softname:' <VAL> <EOL>
<proc-user-id>           ::= 'user:' <VAL> <EOL>
<file-id>                ::= 'filename:' <VAL> <EOL>
<Dynamic-variable>       ::= '#'<identifier>'#'
<Static-variable>        ::= '<'<identifier>'>'
<Constant>               ::= <identifier>
```

Une vue condensée des différents champs qui constituent une action est proposée dans la figure 4.3.

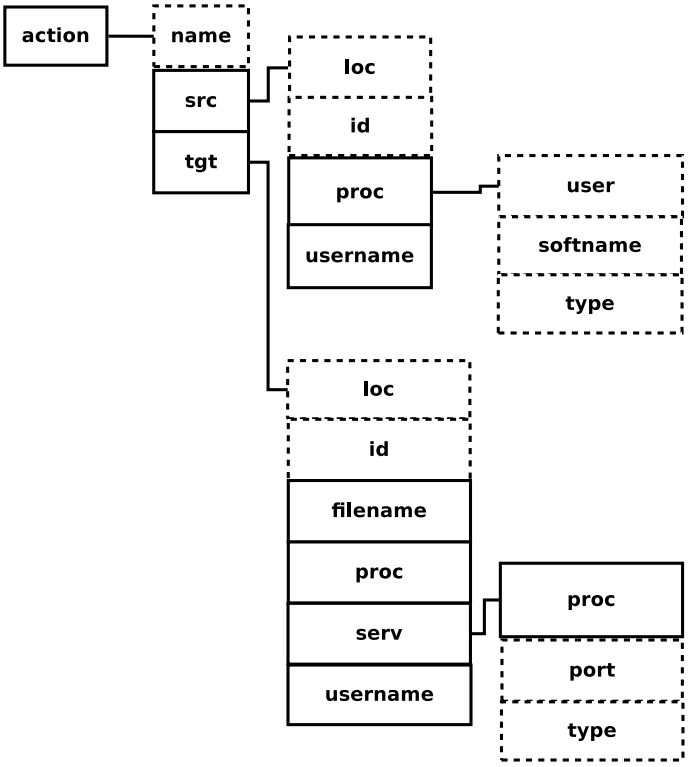


FIGURE 4.3 – Représentation condensée des champs définissant une action. Les champs terminaux sont représentés dans des cadres pointillés.

4.3.2 Sémantique

A Sémantique des différents champs du langage

action (name-field) Le nom associé à une action permet d'attribuer une catégorie spécifique à cette action. Chaque action peut entrer dans deux grandes catégories. Elle peut correspondre à une attaque, c'est-à-dire une action suffisamment grave pour justifier une levée d'alertes même hors du contexte du scénario d'attaque, par exemple une tentative d'injection SQL, ou bien elle peut correspondre à une action normale qui prend un caractère malveillant uniquement dans le cadre d'un scénario d'attaque. Par exemple, un échec de connexion, un flux de communication entre deux machines, la création d'un compte utilisateur. Le nom de l'action est issu d'une taxinomie (CAPEC pour les actions qui sont des attaques et CEE pour les actions normales). Lors de la génération des règles, ce nom est utilisé pour déterminer les observateurs capables de détecter la catégorie d'actions ou d'attaques correspondante.

id (node-id) : L'objectif de ce champ est de permettre l'identification non ambiguë d'un nœud du système. Cet identifiant peut consister en un identifiant classique (adresse IP) ou bien

un identifiant arbitraire issu de la base de connaissances. L'utilisation d'adresses IP ou mac ne sont pas conseillés pour les raisons suivantes : 1) un nœud peut disposer de plusieurs interfaces réseau, ce qui signifie qu'il y a plusieurs moyens d'identifier un même nœud ; 2) un système peut être réparti sur plusieurs sites disposant de plages d'adressage privées qui se chevauchent, ce qui a pour conséquence la potentielle non-unicité de l'association entre une adresse IP et un nœud. Pour les entités inconnues, cet identifiant est l'adresse IP. Cette problématique d'identification et de normalisation a été abordée en 3.3.1. Une action locale est caractérisée par l'affectation d'un même identifiant aux champs relatifs à l'identifiant de la source (*src.id*) et de la cible (*tgt.id*).

loc (location-id) Ce champ spécifie la zone d'appartenance d'une machine. Dans cette thèse, ce champ sert essentiellement à spécifier une localisation logique en termes de sous-réseaux, mais il est possible d'utiliser une autre organisation logique (physique par exemple). Une machine peut appartenir à plusieurs de ces zones, en particulier lorsqu'elle dispose de plusieurs interfaces réseau. Par exemple, le champ *loc : dmz* indique que l'entité concernée doit se trouver dans un sous-réseau nommé dmz. L'intérêt de ce champ et de permettre de sélectionner un ensemble de machines.

username (user-id) Le nom d'utilisateur permet d'exprimer des règles de corrélation liées à un utilisateur (par exemple une réception ou un envoi de mails, connexion sur un compte utilisateur spécifique).

process (process-id) Ce champ composé est utilisé pour mettre en jeu des informations sur un processus. Il est composé des sous-champs *type*, *softname* et *user*.

- *type (softype-id)* ce champ permet de spécifier un type de logiciel particulier. Les noms admissibles correspondent aux types des entrées de la base de connaissances définissant les logiciels : il s'agit de la variable *Type* du prédicat *software(Name,Version,Type,Arch)* (par exemple, *web* dans *software(apache,'2.2',web,x86)*).
- *softname (softname-id)* il s'agit d'un sous-champ de *process* identifiant le nom du logiciel dont le processus est l'instance. Ce nom correspond à la variable *Name* du prédicat *software(Name,Version,Type,Arch)* (*apache* de l'exemple qui précède).
- *user (proc-user-id)* Ce champ a pour but de permettre de spécifier le nom d'utilisateur dont les droits sont utilisés par un processus (ex : http-data, sshd). Il correspond à la variable *U* du prédicat *process(S,U)*.

service (service-id) Ce champ composé est utilisé pour spécifier la présence de services réseaux écoutant sur un port ou un ensemble de ports. Il est possible de spécifier les informations concernant le processus responsable du service.

Le sous-champ *type* est utilisé comme raccourci à la place de *serv.proc.type* lorsque la définition de champs sur le procesus n'est pas nécessaire mais que la spécification du type de logiciel l'est.

filename (filename-id) Ce champ peut être utilisé lorsqu'un fichier joue un rôle dans différentes actions. Par exemple, un fichier peut être écrit ou téléchargé par un attaquant pour être exécuté par la suite.

B Sémantique spécifique

Toutes les actions présentes dans CAPEC et CEE n'ont pas le même niveau de granularité et d'ambiguïté. Certaines actions sont par nature ambiguës dans le choix d'une source et d'une destination, comme l'usurpation d'identité par exemple.

Statut d'une action Lors de la création de scénarios d'attaque, il peut être souhaitable d'exprimer l'échec d'une action. Du point de vue des statuts des actions, il est important de noter que le standard CEE propose un champ statut (qui peut prendre pour valeur *cancel*, *error*, *failure*, *success*, *ongoing*, *unknown*). Ce statut d'action peut se modéliser d'au moins deux manières.

1. en ajoutant un attribut statut au langage d'action. Par défaut, la valeur est toujours *success*, sauf si un statut spécifique est requis.

2. en associant le statut à l'action en créant deux types de nom d'actions distincts (ex : connection-failed et connection(-success)). Ces deux noms viennent alors enrichir le dictionnaire des noms des actions possibles.

La première solution implique l'ajout d'un nouveau champ dans le langage. La seconde implique l'introduction de certains nouveaux noms d'action incluant un statut. Cette seconde solution est conservée dans notre approche.

4.3.3 Variables statiques et variables dynamiques

A Définition

Le langage a pour objectif de décrire une action en disposant d'un niveau de généricité ajustable selon les besoins et ainsi de permettre de privilégier la généricité ou la précision. Les attributs terminaux du langage peuvent presque tous [2] être décrits de trois manières : sous la forme d'une constante, d'une variable dynamique ou bien d'une variable statique. L'utilisation de constantes consiste à attribuer à un attribut une valeur fixe (par exemple en fixant un numéro de port). Les constantes sont particulièrement pertinentes pour certains attributs (localisation et nom d'une action), mais leur utilisation systématique pour les autres attributs limite la généricité et la maintenabilité de l'approche et demande une connaissance fine de certains détails du système.

Les variables statiques sont utilisées pour sélectionner des entités identifiables à partir de la base de connaissances. Leur utilisation permet de rester générique et de prendre en compte potentiellement plusieurs entités du système. En pratique, cela concerne les éléments fixes du système.

Les variables dynamiques modélisent des entités pour lesquelles l'identité ne sera connue qu'à l'exécution (lorsque le scénario d'attaque se réalise). Elles sont utiles pour modéliser des entités inconnues (par exemple pour identifier l'identité d'un attaquant externe ou modéliser un service malveillant qui sera mis en place par un logiciel malveillant) ou bien pour modéliser des entités potentiellement connues mais dont l'énumération explicite n'est pas souhaitée (par exemple pour modéliser un sous-réseau contenant uniquement des machines clientes). Une variable utilisée dans deux actions différentes (ou dans la même action) identifie la même entité. Ainsi, si la variable A modélise la source d'une action 1 et est utilisée dans la cible d'une action 2, cela signifie que l'objectif est d'identifier une machine m qui est à la fois la source de l'action 1 et la cible de l'action 2.

B Utilisation

Il est important de noter que contrairement à ce qu'indique la grammaire du listing 4.1, toutes les combinaisons possibles consistant à associer à un champ donné un type de variable ne sont pas valides. En particulier, l'utilisation d'une variable dynamique pour l'identifiant d'un nœud suivi de l'utilisation d'une variable statique pour identifier des composants d'un service, d'un utilisateur ou d'un processus sur ce même nœud n'a pas de sens. En effet, la variable dynamique indique une volonté de ne pas chercher à attribuer une valeur concrète aux identifiants de nœuds possibles. Or une résolution des autres informations renseignées avec des variables statiques implique une identification du nœud en question. Nous définissons un ordre de priorité dans les champs permettant d'évaluer la cohérence d'une combinaison. La relation *A est prioritaire sur B* signifie que lorsque le champ A est défini par une variable dynamique, le champ B ne peut être défini qu'avec une variable dynamique ou une constante. La localisation est prioritaire sur l'identifiant du nœud qui lui-même est prioritaire sur tous les champs caractérisant un nœud (sauf le champ localisation).

4.3.4 Construction de l'arbre d'actions

Le langage d'actions décrit précédemment peut maintenant servir à enrichir l'arbre d'attaque initial. Formellement, un arbre d'actions $at \in AT$ est décrit de manière récursive comme une action terminale a ou bien un ensemble constitué d'un opérateur $o \in \{SAND, AND, OR\}$ et d'une liste

2. Le nom d'une action ne peut pas être représenté par une variable statique. Pour représenter une action quelconque, une variable dynamique est utilisée

ordonnée d'arbres d'actions $at_i \in AT$:

$$at = \{o, [at_1, \ldots, at_n]\} \mid a$$

Une action a est modélisée par une fonction qui associe à chaque champ du langage d'actions une valeur dans l'ensemble $V = V_{stat} \cup V_{dyn} \cup Const$ construit par l'union de l'ensemble des variables statiques (V_{stat}), l'ensemble des variables dynamiques (V_{dyn}) et l'ensemble des constantes ($Const$).

A Application sur l'exemple

La figure 4.4 présente la création de l'arbre d'actions issu de l'arbre d'attaque de la figure 4.2. Chaque feuille de l'arbre d'attaque initial est transformée de manière à expliciter le nom de l'action, sa source et sa cible.

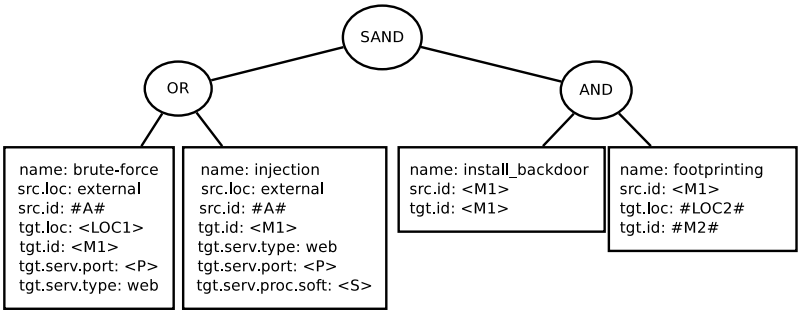


FIGURE 4.4 – Arbre d'actions construit à partir de l'arbre d'attaque 4.2. Les variables statiques sont représentées avec des symboles '< >' et les variables dynamiques avec les symboles '# #'

Les choix de modélisation des différentes actions sont justifiés ci-dessous :

Brute-force : Attaque par force brute sur une page d'administration Le nom de cette action est issu directement de CAPEC. Il est fait l'hypothèse que l'attaquant se situe sur un réseau extérieur et sur un nœud non identifié, donc représenté par une variable dynamique (#A#).

Pour spécifier que la cible de l'attaque peut être une machine quelconque du système accessible depuis l'extérieur, des variables statiques sont utilisées pour la localisation et l'identifiant de la machine cible. De plus, il est spécifié que l'attaque cible un service web écoutant sur un port quelconque.

Injection : réalisation d'une injection sur une application Web Cette action spécifie une attaque par injection depuis la machine de l'attaquant vers la machine cible. La source de cette exploitation provient également du réseau externe et de la même adresse que l'action précédente. L'action *injection* issue de CAPEC caractérise un envoi de données via une interface publique pouvant perturber le fonctionnement normal d'une application. Pour modéliser la possibilité pour l'attaquant de changer son adresse à l'origine de l'attaque, il suffit de dupliquer l'action d'injection et d'y utiliser une variable dynamique différente, par exemple *#A'#*.

Install_backdoor : Mise en place d'une porte dérobée Cette action porte dérobée n'existe pas de manière explicite dans CAPEC. Les classes d'attaque s'en rapprochant le plus sont :

— *Malicious logic inserted into product software* qui correspond à l'insertion d'une porte dérobée dans un produit mais l'idée développée par CAPEC et plutôt centrée autour de l'insertion malveillante de cette porte dérobée avant la distribution du logiciel. Or dans notre cas, nous nous intéressons à l'action d'un attaquant qui modifie à la volée un service existant pour y introduire sa porte dérobée.

— *execute code* correspond à une action d'installation et d'exécution de code malveillant sur une machine cible. Cette action correspond à la mise en place d'une porte dérobée mais elle reste cependant très générique.

Par conséquent, nous avons défini l'action *install_ backdoor* comme une classe d'attaque fille de la classe CAPEC *execute code*. Ce choix permet de reposer sur CAPEC tout en étant plus précis.

S'agissant d'une action locale, les identifiants de la source et de la cible sont identiques. Cette action décrit ainsi la mise en place d'un accès distant par l'intermédiaire d'une porte dérobée.

Footprinting : reconnaissance depuis le nœud infecté La différence avec la première action est que la source est la machine compromise et la cible est une machine quelconque. Une variable dynamique est utilisée pour spécifier à la fois la localisation et le nœud cible. Il est ainsi possible de modéliser un scan interne ou vers l'extérieur du système. Le nom de l'action, *footprinting*, est issu de CAPEC et regroupe les actions de découvertes dont les scans de port font partie.

4.3.5 Discussion

Cette section a présenté la méthode permettant de passer d'un arbre d'attaque à un arbre d'actions. La principale difficulté de cette étape réside dans son caractère manuel. Cette intervention manuelle pour transformer un arbre d'attaque en arbre d'actions est requise pour plusieurs raisons.

Une première raison vient de la nécessité de développer l'arbre d'attaque initial de manière à ce que ce dernier soit décomposé en sous-objectifs suffisamment précis pour être directement associables à des actions. Il est alors possible d'ajouter une contrainte incitant à utiliser uniquement des arbres d'attaques suffisamment détaillés comme point de départ. Mais la mise à disposition d'un arbre d'attaque suffisamment détaillé n'est pas l'unique problème.

Une seconde raison est liée à la nécessité de réaliser une interprétation pour identifier les différents champs du futur arbre d'actions à partir d'une feuille de l'arbre d'attaque qui est constitué simplement d'une description informelle. Cela suppose que cette description soir suffisamment bien structurée pour permettre l'extraction des informations nécessaires sans ambiguïtés.

La troisième raison concerne le choix d'utilisation de constantes, de variables dynamiques ou de variables statiques pour renseigner les champs du langage d'actions. L'attribution pourrait être automatisée selon des règles génériques liées aux champs et au type d'action, mais cela risquerait de réduire la souplesse d'utilisation.

Dans tous les cas, une automatisation de cette étape suppose que la structure d'arbre d'attaque de départ soit suffisamment enrichie et structurée (manuellement), ce qui ne fait que déplacer le problème.

4.4 Adaptation du scénario au système surveillé

4.4.1 Identification des acteurs

Cette étape a pour objectif d'expliciter les entités qui sont potentiellement impliquées par le scénario d'attaque spécifié. L'opération principale de cette étape consiste en l'instanciation des variables statiques à partir des informations contenues dans la base de connaissances. Chaque combinaison possible de valeurs de variables statiques donne lieu à la création d'un arbre dédié correspondant à un chemin d'attaque spécifique. Ces différents arbres sont ensuite fusionnés en un arbre unique à l'aide d'un opérateur *OR* servant de nouvelle racine. Le but de cet opération est de rassembler les différentes instanciations du scénario d'attaque initial. Cette opération se justifie car chacune des branches représente un cas possible de réalisation du scénario d'attaque initial sur le système cible.

A Définition de la transformation

Le processus d'identification des acteurs est décrit comme une transformation de la structure de l'arbre d'actions par des opérations élémentaires.

Instanciation partielle Soit $\sigma^{\Omega}[v_1 := \alpha_1, \ldots, v_n := \alpha_n] \in \Sigma^{\Omega}$ la substitution qui associe à chacune des variables statiques $v_i \in V_{stat}$ appartenant à l'ensemble des variables statiques V_{stat} une valeur $\alpha_i \in VAL(\Omega)$, où $VAL(\Omega)$ est l'ensemble des faits contenus dans la base de connaissances

Ω. Σ^{Ω} identifie l'ensemble des substitutions possibles avec des valeurs contenues dans la base de connaissances Ω. L'application de cette substitution sur une action a, notée $\sigma^{\Omega}(a)$, substitue les valeurs α_i aux variables statiques v_i de l'action a. Jusque-là, cette définition ne donne pas de restriction sur la cohérence des valeurs possibles. Ce sont les contraintes imposées par les actions qui restreignent ces valeurs (ces contraintes sont énumérées dans la sous-partie B). Ces actions constituent les feuilles de l'arbre d'actions at et sont notées $A(at)$

Ainsi, $\sigma^{\Omega}|_{A(at)}$ est une substitution qui vérifie toutes les contraintes induites par l'ensemble des actions $A(at)$ dans le contexte donné par la base de connaissances Ω.

Soit $\sigma^{\Omega}|_{A(at)}$ une telle substitution. La notation $InstPart_{\sigma}(at)$ correspond à l'instanciation partielle de l'arbre d'actions at, qui correspond à un arbre dont toutes les variables statiques sont substituées selon $\sigma^{\Omega}|_{A(at)}$. Cette fonction d'instanciation partielle est définie par :

(1) $\forall a \in A(at)$, $InstPart_{\sigma}(a) = \sigma^{\Omega}|_{A(at)}(a)$

(2) $InstPart_{\sigma}(\{o, [at_1, \ldots, at_n]\}) = \{o, [InstPart_{\sigma}(at_1), \ldots, InstPart_{\sigma}(at_n)]\}$.

L'instanciation partielle relative à une substitution σ d'un arbre réduit à une action revient à appliquer la substitution à cette action.

Instanciation totale On note $InstTot(at) = \{InstPart_{\sigma}(at), \forall \sigma \in \Sigma^{\Omega}|_{A(at)}\}$ l'ensemble des instanciations partielles de l'arbre d'actions at. Il s'agit de l'ensemble des instanciations partielles relatives aux substitutions issues de l'ensemble $\Sigma^{\Omega}|_{A(at)}$ des substitutions conformes aux contraintes de l'ensemble des actions $A(at)$ issues de l'arbre d'actions at.

Deux cas sont alors possibles.

— Si $Card(InstTot(at)) = 1$, alors l'unique arbre contenu dans l'ensemble $InstTot(at)$ constitue l'arbre d'actions instancié avec les acteurs, noté $at|_{Actor}$.

— Si $Card(InstTot(at)) > 1$, alors $at|_{Actor} = \{OR, [\ x \mid x \in InstTot(at)]\ \}$. Cette opération consiste à rassembler la forêt $InstTot(at)$ avec une nouvelle racine (OR) commune.

Détermination de l'ensemble des substitutions possibles Une étape reste en suspens dans la description précédente. Il s'agit de la détermination de l'ensemble des substitutions conformes aux contraintes de toutes les actions d'un arbre d'actions. Un arbre d'actions ne contenant pas d'opérateur *OR* est simple à traiter. En effet, le scénario suppose que l'attaquant est contraint de réaliser l'intégralité des actions, ce qui implique qu'une substitution est acceptable lorsqu'elle l'est pour l'ensemble des actions. Cependant, lorsqu'un opérateur *OR* est présent, l'attaquant n'a besoin de réaliser qu'un sous-ensemble des actions contenues dans l'arbre d'actions. Par exemple, prenons un opérateur *OR* qui lie une action modélisant l'exploitation d'un service Web sur un serveur et l'autre une attaque par force brute sur un service SSH sur le même serveur. Si aucun serveur ne dispose à la fois d'un service SSH et d'un service WEB, alors une substitution cohérente pour les deux actions est impossible sans pour autant que l'attaque réelle ne le soit [3]. Plusieurs choix sont possibles pour traiter ce problème :

(1) considérer que toutes les actions de l'arbre d'actions ont le même poids en termes de contraintes. Cela revient à dire qu'une substitution est acceptable uniquement s'il est possible de satisfaire simultanément l'ensemble des actions de l'arbre d'actions. Il s'agit de la méthode la plus simple à mettre en œuvre car il n'y a pas de distinctions à effectuer entre les différentes actions. La détermination des substitutions possibles revient à faire l'intersection des substitutions conformes pour chaque action. Ce choix a pour principale conséquence d'accorder la même importance à une action indispensable (dont le parent est un *SAND* ou un *AND*) et une action optionnelle (dont le parent est un *OR*). Ainsi, une contrainte d'accessibilité sur une action optionnelle peut restreindre l'ensemble des substitutions possibles pour l'ensemble des autres actions ;

(2) pour répondre aux défauts de la méthode précédente, la seconde option consiste à déterminer les nœuds optionnels et à procéder à un développement les faisant disparaître au profit d'une forêt d'arbres comprenant uniquement des séquences et des *ET*. Le point positif de cette approche est la prise en compte de plus de possibilités que précédemment. Par contre, la taille de la structure

3. Cet exemple est résolu simplement en utilisant deux variables statiques différentes pour différencier les deux nœuds.

est plus importante. De plus, cette approche n'est intéressante que si l'arbre d'actions dispose d'opérateurs *OR* dont au moins l'une des actions filles impose des contraintes nouvelles par rapport aux contraintes des actions obligatoires (celles qui ont un opérateur *SAND* ou *AND* comme père) ;

(3) une approche intermédiaire consiste à évaluer les contraintes qui concernent les sous-arbres des opérateurs *OR* et à ne réaliser le développement expliqué au point précédent que si ces contraintes sont plus fortes que les contraintes du reste de la structure.

L'option (3) répond de la manière la plus complète à cette problématique. Cependant, les bénéfices de cet algorithme sont faibles dans les cas où la configuration problématique n'est pas présente. Par construction, cette configuration problématique est contournable au moment de la création de l'arbre d'actions. C'est par conséquent l'option (1) traitant toutes les actions de manière identique qui est suivie. Même si ce choix semble restreindre la couverture de la règle de corrélation, il est possible de limiter les effets en tenant compte de ce mécanisme lors de la construction de l'arbre d'actions. Il est par exemple possible de distinguer les noms des variables statiques utilisées dans chacune des branches d'un opérateur *OR* lorsque des actions sont exclusives.

B Liens avec la base de connaissances

Ce processus d'identification des acteurs demande la mise à disposition d'informations sur le système cible. Les valeurs des variables statiques sont contraintes à la fois par les informations contenues dans la base de connaissances et également par les actions spécifiées dans l'arbre d'actions.

Types de contraintes Les contraintes s'appliquant sur chacune des variables sont déterminées par une requête à la base de connaissances permettant de sélectionner les valeurs possibles pour une variable donnée. Les contraintes liées au scénario sont résumées dans le tableau 4.1. Une contrainte de type restreint les valeurs d'une variable au type du champ concerné. Par exemple, un champ *action.id* ne peut être associé qu'à un nom d'action, alors que le champ *service.port* prend une valeur de numéro de port. Une contrainte fonctionnelle décrit les relations entre un nœud et les services ou logiciels qui y sont installés. De même, une contrainte peut restreindre le choix de deux nœuds par une condition sur leurs positionnements respectifs au sein d'un même sous-réseau. La contrainte sur les vulnérabilités est analogue à une contrainte fonctionnelle, mais concernant la présence de vulnérabilités. Les contraintes d'accessibilité sont liées aux actions qui demandent une communication entre deux nœuds. Ce type de contrainte restreint l'ensemble des nœuds impliqués.

Contrainte	Description
Type	Type de la variable (identifiant, port, réseau. . .)
Fonctionnelle	Caractéristiques d'un nœud (services présents, localisation)
Vulnérabilité	Lorsqu'une attaque exploite une vulnérabilité spécifique
Accessibilité	Existence d'une accessibilité entre deux nœuds lorsqu'une action le requiert

TABLE 4.1 – Contraintes de résolution des variables statiques.

La figure 4.5 présente la construction générique des requêtes permettant de sélectionner les acteurs. Pour chaque action, un ensemble de requêtes est envoyé. Ces requêtes dépendent des champs présents dans l'action. Par exemple, si une action ne met en jeu aucun service réseau, il est inutile d'exprimer une requête sur la présence de service.

Les requêtes présentées utilisent les variables statiques référencées dans la spécification de l'action et également des variables auxiliaires représentées avec le préfixe *?* pour permettre de les différencier [4]. Quelques explications sont nécessaires pour expliquer leur rôle dans ces requêtes.

4. Cette distinction est présentée ici pour faciliter la présentation. Les « _ » représentent des variables anonymes.

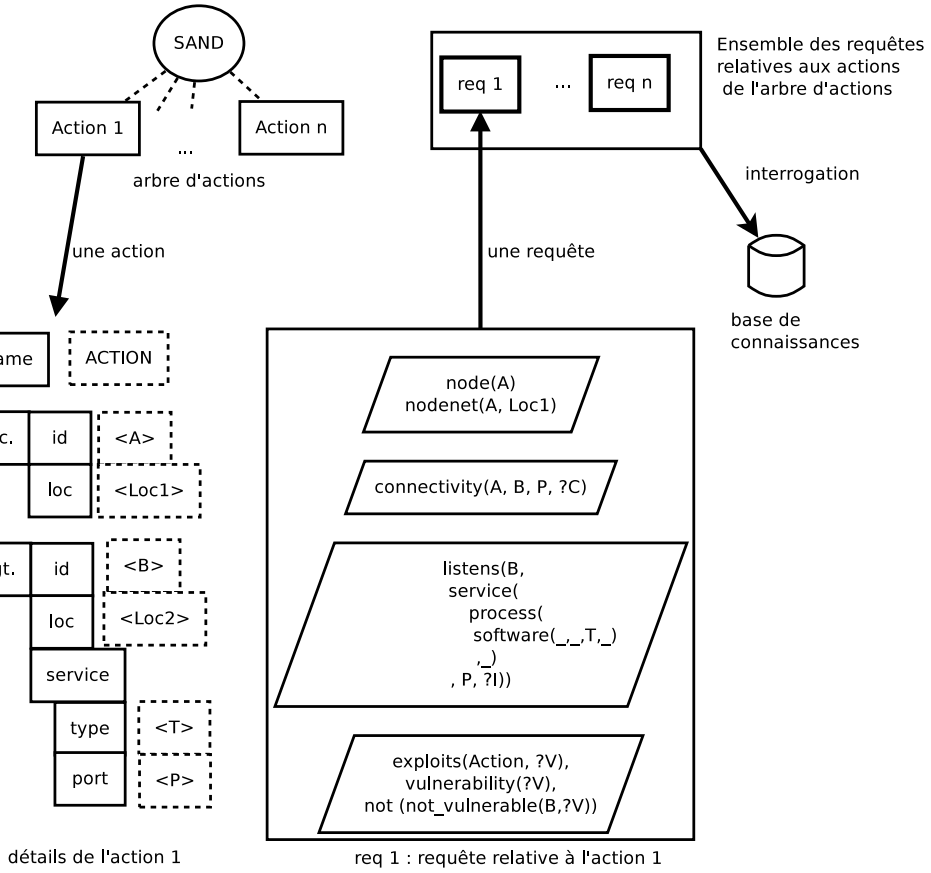


FIGURE 4.5 – Schéma générique de construction des contraintes permettant d'identifier les acteurs relativement à un arbre d'actions et à une base de connaissances donnée.

Le prédicat *listens* permet de spécifier différents paramètres d'un service réseau, en particulier l'interface d'écoute représentée par *?I*. Cette interface doit être accessible par la route permettant à la machine A de communiquer avec B sur le port P. Cette information est justement déductible à partir de la variable *?C* utilisée dans le prédicat *connectivity* car elle contient les informations sur les interfaces réseau utilisées pour réaliser la communication.

Lorsqu'une action fait référence à des vulnérabilités potentielles, la présence de ces vulnérabilités sur le nœud cible est vérifiée. Dans le cas où l'action n'est pas explicitement liée à une vulnérabilité, le nœud est sélectionné. En effet, si l'action est une *injection*, classe d'attaque très générique, il est difficile de prouver que le nœud n'est pas vulnérable à toutes les injections possibles [5]. Par contre, en définissant une action beaucoup plus précise comme *shellshock_ exploitation* [6], cette détermination de vulnérabilité du nœud a du sens. Cependant, l'utilisation d'un nom d'action aussi précis réduit la portée de la règle de corrélation finale.

C Application sur l'exemple

Dans notre exemple, le scénario contient quatre variables statiques. Elles correspondent respectivement à une localisation ($LOC1$), à l'identifiant d'un nœud ($M1$), à un numéro de port (P) et au nom du logiciel faisant tourner le service (S). Les contraintes fonctionnelles et les contraintes liées aux vulnérabilités que doit satisfaire $M1$ sont :

— un couplage avec la valeur de la variable $LOC1$ qui correspond à la localisation du nœud ;

5. Il peut cependant l'être à toutes les attaques par injection connues.
6. Cette action n'est pas dans CAPEC, il serait donc nécessaire de définir cette nouvelle classe d'attaque qui hériterait par exemple de la classe *arguments injection*.

— la présence [7] d'un service web vulnérable à une attaque par injection ;
— le nœud doit être vulnérable à une attaque de type reconnaissance (gather_information). De même, s'il n'est pas possible de prouver que le nœud n'est pas vulnérable à ce type d'attaque, alors la contrainte est automatiquement satisfaite.

Le nœud *M1* doit également satisfaire les contraintes liées à l'accessibilité qui sont spécifiées dans l'arbre d'actions :

— une accessibilité depuis la zone *external* ;
— une accessibilité vers au moins une autre machine (*LOC2* et *M2* étant des variables dynamiques, aucune information supplémentaire n'est disponible).

La figure 4.6 illustre cette transformation. L'opérateur *OR* situé à la racine permet de joindre deux chemins d'attaques réalisant le scénario générique. À droite, l'attaquant cible une machine identifiée par *id1*, à gauche, une machine identifiée par *id2*. Dans chacune de ces branches se retrouve le déroulement du scénario d'origine.

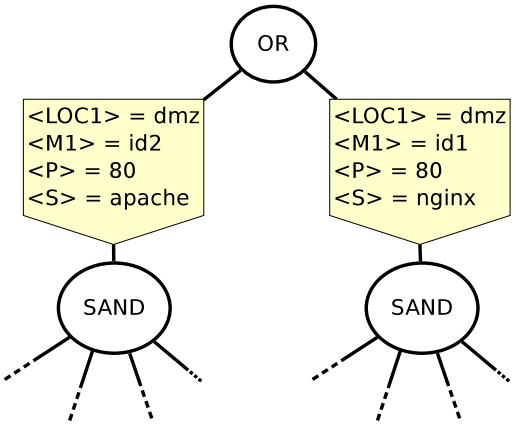


FIGURE 4.6 – Identification des acteurs pour l'arbre d'actions 4.4. Chaque branche correspond à l'arbre d'actions initial auquel les substitutions de valeurs ont été réalisées pour les variables statiques.

4.4.2 Identification des observateurs

L'étape précédente se focalise sur les acteurs du scénario d'attaque. L'étape courante vise à identifier les éléments du système capables d'observer les actions spécifiées dans l'arbre d'actions. Ces observateurs sont constitués de toutes les entités du système capables de détecter des actions (sonde, IDS).

La sélection des observateurs pertinents pour l'observation d'une action donnée repose sur deux propriétés. La première propriété, la visibilité topologique, est exploitée dans le but de sélectionner les observateurs qui sont positionnés à un emplacement adapté au sein du système d'information, c'est-à-dire qui sont capables de surveiller les acteurs (identifiés lors de l'étape précédente) impliqués dans l'action. La seconde propriété, la visibilité opérationnelle, permet de sélectionner les observateurs susceptibles de détecter l'action en fonction de son type (classe d'attaque spécifique ou action normale). Ces deux propriétés régissent donc globalement la sélection des observateurs. Cependant, l'exploitation de ces propriétés dépend fortement des caractéristiques des actions à traiter. Les détails des traitements réalisés dans chaque cas sont abordés dans la suite.

A Sélection selon la visibilité topologique

Deux cas sont identifiés pour réaliser la sélection d'un observateur relativement à sa visibilité topologique pour une action donnée. Dans le premier cas où l'action implique deux nœuds distincts, deux types d'observateurs sont potentiellement concernés : d'une part les observateurs locaux à un

7. En réalité, la sélection est réalisée s'il est impossible de prouver que la machine n'est pas vulnérable à l'attaque donnée. Voir Chapitre 3.

nœud et d'autre part les observateurs réseaux. En ce qui concerne les observateurs réseaux, leur capacité à voir l'action est conditionnée par leur présence sur le trajet du flux d'informations réalisé entre les deux machines impliquées dans l'action. Les observateurs locaux relatifs à chacun des nœuds de l'action (si ces nœuds sont connus au moment de l'instanciation, c'est-à-dire identifiés par des variables statiques) sont potentiellement concernés par la détection de l'action. Dans le second cas, un seul nœud est concerné, seuls les observateurs locaux sont sélectionnés.

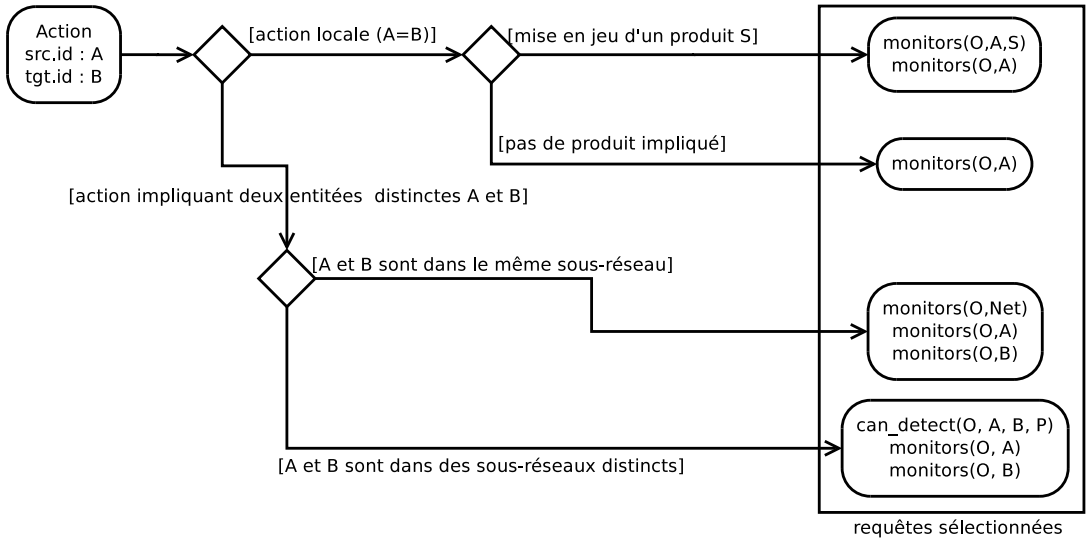


FIGURE 4.7 – Sélection des requêtes à réaliser pour la détermination des observateurs d'une action en fonction de la visibilité topologique.

La méthode de sélection des requêtes permettant de déterminer les observateurs d'une action en fonction de leur visibilité topologique est présentée dans le cas général à la figure 4.7. Dans tous les cas, les observateurs locaux sont recherchés indépendamment du type de l'action car une action impliquant deux nœuds distincts est potentiellement visible localement sur l'un des deux nœuds. Par exemple,une connexion d'une machine A vers une machine B est visible sur les deux nœuds si la liste des connexions actives de chaque machine est surveillée.

cas	src.loc	src.id	tgt.loc	tgt.id	sélection
(1)	dyn	dyn	dyn	dyn	aucun observateur ou tous les observateurs
(2)	dyn	dyn	const	dyn	observateur réseau dans tgt.loc
(3)	dyn	dyn	const	const	observateurs réseaux dans tgt.loc et locaux à tgt.id
(4)	const	dyn	dyn	dyn	symétrique du cas (2)
(5)	const	dyn	const	dyn	observateurs réseaux de src.loc et tgt.loc
(6)	const	dyn	const	const	observateurs réseaux de src. loc, tgt.loc et observateurs locaux de tgt.id
(7)	const	const	dyn	dyn	symétrique du cas (3)
(8)	const	const	const	dyn	symétrique du cas (6)
(9)	const	const	const	const	cas nominal

TABLE 4.2 – Résolution de la visibilité topologique en fonction de l'utilisation des variables dynamiques pour les quatre champs localisation et identifiants respectivement source et cible.

La figure 4.7 illustre un cas nominal dans lequel la source et la cible de l'action sont clairement identifiés, c'est-à-dire que les identifiants des nœuds sont des constantes (soit des constantes issues

directement de la spécification de l'arbre d'attaque, soit des constantes issues de la substitution de variables statiques lors de l'identification des acteurs). Le tableau 4.2 liste les différentes combinaisons possibles entre les affectations de variables dynamiques aux champs *src.id*, *src.loc*, *tgt.id* et *tgt.loc* et les conséquences en ce qui concerne la recherche d'observateurs en fonction de leur visibilité topologique. En résumé, lorsque l'identité d'un nœud n'est pas connue mais sa localisation l'est, les observateurs sélectionnés sont des observateurs dont la localisation coïncide avec celle du nœud en question. Le cas (9) correspond au cas nominal dans lequel aucune variable dynamique n'est utilisée dans un des quatre champs mentionnés précédemment. Le cas (1) constitue le plus pathologique car il modélise une action pour laquelle aucune information sur la source ou la cible n'est connue. À notre connaissance, il n'y a pas d'exemple concret et pertinent correspondant à cette situation.

Le tableau 4.2 présente les combinaisons possibles des types de variables et la manière sont les observateurs sont identifiés dans chaque cas. Les variables statiques sont résolues et substituées, ce qui explique que les champs ne puissent être associés qu'à des variables dynamiques (dyn) ou des constantes (const). Il est supposé que ces actions impliquent des nœuds différents, c'est-à-dire que les actions ne sont pas locales. Dans le cas d'une action locale, seuls les observateurs locaux sont conservés [8].

B Visibilité opérationnelle

Cette propriété permet de sélectionner des observateurs étant donné leurs capacités réelles à détecter l'occurrence d'une action selon leur configuration mise en œuvre. La sémantique de l'action est contenue dans son nom et appartient à une taxinomie. À partir de cette information, il est possible de sélectionner les observateurs pouvant détecter ce type d'action. Chaque action appartient à l'ensemble des attaques ou à l'ensemble des actions standard.

B.1 Classe d'attaque Dans le cas d'une action correspondant à une classe d'attaque, plusieurs cas généraux sont identifiables en fonction de la position relative de la classe d'attaque spécifiée dans l'action et les classes d'attaque définissant la visibilité opérationnelle d'un observateur. Ces cas sont les suivants :

(a) l'observateur dispose d'une visibilité opérationnelle qui couvre partiellement l'action spécifiée dans l'arbre d'actions. Ce cas est représenté en (a) dans la figure 4.8 ;

(b) la visibilité opérationnelle de l'observateur couvre entièrement les sous-classes d'attaques de la classe d'attaque définie dans l'arbre d'actions ;

(c) la visibilité opérationnelle de l'observateur inclut une classe d'attaque qui est une classe ancêtre de la classe d'attaque spécifiée dans l'arbre d'actions. Ce cas est illustré en (c) dans la figure 4.8 ;

(d) la classe d'attaque spécifiée dans l'arbre d'actions et celle définissant la visibilité opérationnelle de l'observateur sont dans deux branches différentes. (Cas (d) de la figure 4.8) ;

(e) cas potentiellement rencontré en cas d'héritage multiple. La classe d'attaque détectable par la règle n'est pas une sous-classe ni une classe ancêtre de la classe d'attaque spécifiée dans l'arbre d'action. Par contre, il existe un nœud fils ou petit-fils commun entre ces deux classes d'attaques ;

(0) cas le plus simple correspondant à l'égalité entre la classe d'attaque spécifiée dans l'arbre d'actions et la classe d'attaque définie explicitement comme visibilité opérationnelle de l'observateur *O*. Par exemple, lorsque l'action spécifiée dans l'arbre d'actions est *injection* et l'observateur *O* dispose d'une visibilité opérationnelle définie par *detects(O,injection)*.

Le cas (d) est le seul qui assure une incapacité de l'observateur à observer l'action. Dans tous les autres cas, la détection reste possible même si elle n'est pas garantie, en particulier pour le cas (a).

B.2 Actions normales Dans le cas d'une action standard, il n'y a pas de structure arborescente mise en jeu et par conséquent la détermination de la couverture en termes de visibilité fonctionnelle est immédiate. Par exemple, une action *login* est observable par un observateur *O*

8. Les cas qui ne sont pas représentés ici correspondent à des répartitions interdites.

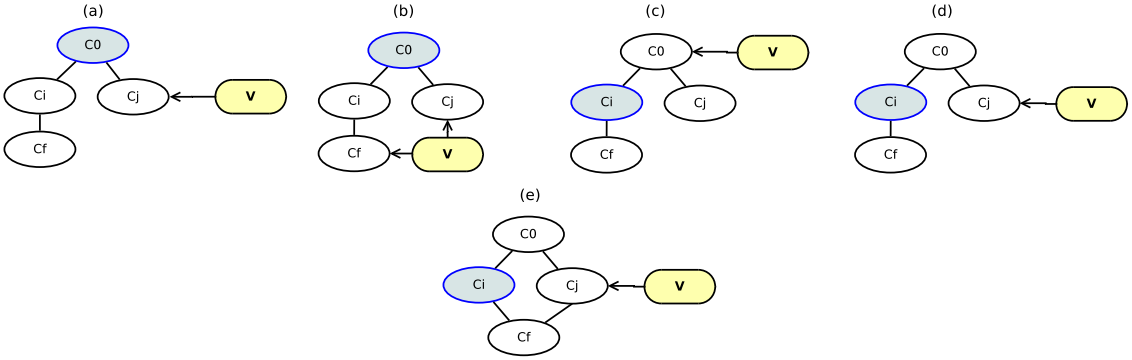


FIGURE 4.8 – Les nœuds C0, Ci, Cj et Cf correspondent à des classes d'attaque organisées sous la forme d'un arbre. Le nœud colorié (en bleu) appartenant à l'arborescence représente la classe d'attaque définie dans l'action pour laquelle les observateurs potentiels sont recherchés. La visibilité opérationnelle de l'observateur est représentée par la classe d'attaque pointée par le nœud V ainsi que l'ensemble des classes d'attaques filles de la classe pointée.

s'il existe une configuration associée à cet observateur (*detection_ conf(O,conf)*) permettant de détecter l'action (*detects(conf,login)*).

C Description formelle de l'étape

Cette étape d'identification des observateurs consiste à construire un arbre d'actions avec observateurs $at|_{obs}$. Soient a une action instanciée et $o \in OBS_\Omega$ un observateur appartenant à l'ensemble des observateurs possibles OBS_Ω . Cet observateur o vérifie $o \in f_{obs}(a)$ si et seulement si les visibilités topologiques et opérationnelles o lui permettent d'observer l'action a.

L'arbre d'actions instanciées avec observateurs est alors défini comme l'ensemble de l'arbre d'actions instanciée et de la fonction f_{obs} :

$at|_{obs} = \{at|_{actor}, f_{obs}\}$

D Exemple

L'application de cette étape à l'exemple se déroule de la manière qui suit : pour chaque action instanciée, les observateurs potentiels sont recherchés en fonction de leur visibilité topologique et opérationnelle.

Un NIDS réseau de type Snort est en écoute au niveau d'un switch sur le sous-réseau DMZ, ce qui se traduit par le fait *monitors(snort-1,dmz)*. La machine identifiée par *id2* est surveillée par un HIDS de type Ossec (fait *monitors(ossec-1,id2)*). La sélection des observateurs concernant le nœud *id2* est justifiée de la manière suivante :

- l'action de brute force sur un service de la machine *id2* fait intervenir deux nœuds appartenant à des zones distinctes et dont la source est représentée par une variable dynamique. La requête *can_ detect(O,external,id2,80)* permet de sélectionner le NIDS *snort-1* qui surveille le sous-réseau DMZ. Comme un service réseau est ciblé, un observateur local surveillant spécifiquement le produit associé est également recherché avec la requête *monitors(O,id2,apache)*. Cette requête permet de sélectionner l'HIDS *ossec-1* ;
- L'action d'injection sur la machine *id2* se résout de manière identique à la précédente ;
- L'action *install_ backdoor* est locale, pour cette raison, *ossec-1* est sélectionné car sa visibilité topologique couvre le nœud *id2* (*monitors(ossec-1,id2)*) ;
- L'action *footprinting* implique deux nœuds dont l'un est représenté par une variable dynamique. La seule information disponible est que la cible se trouve à la localisation DMZ, ce qui permet de sélectionner le NIDS *snort-1* qui surveille ce sous-réseau.

Au niveau de la visibilité opérationnelle pour ce même nœud *id2*, la sélection s'opère de la manière suivante :

— le NIDS *snort-1* dispose de règles permettant de détecter les attaques par force brute, des attaques par injections et des scans de port. Par exemple, cet IDS dispose de la règle '122 :19 :1' permettant de détecter des scans de ports TCP. Cette règle est associée à la classe d'attaque *port_ scanning* via *detects('122 :19 :1', port_ scanning)*. Comme cette classe d'attaque est une sous-classe de footprinting, (*attacksubclass(port_ scanning, footprinting)*), cela indique que *snort-1* est capable de détecter au moins une manifestation de l'action *footprinting* ;
— le HIDS ossec-1 dispose de règles permettant de détecter des attaques par force brute, des injections ainsi que l'installation de portes dérobées.

Pour la machine *id1*, le mécanisme est similaire mais la différence est que ce nœud ne dispose pas d'HIDS installé. Par conséquent, l'action de mise en place d'une porte dérobée n'a aucun observateur associé.

La figure 4.9 présente le résultat de l'identification des observateurs dans le cadre de l'exemple présenté précédemment.

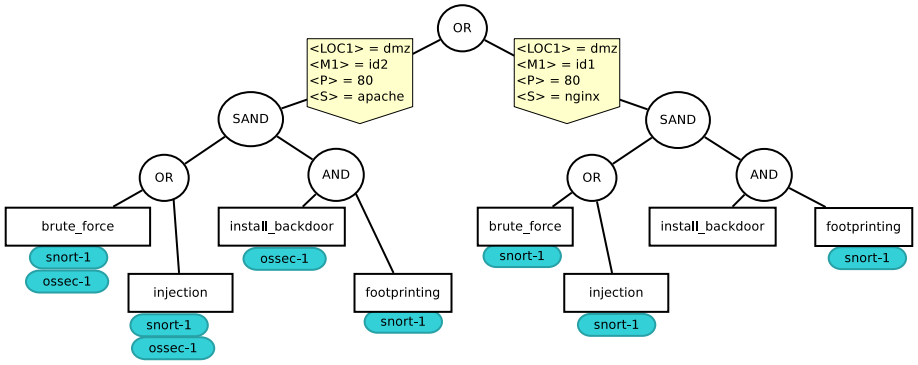


FIGURE 4.9 – Association des observateurs aux différentes actions de l'arbre d'action.

E Discussion

La principale discussion liée au déroulement de cette étape concerne la manière de prendre en compte les différents cas présentés à la figure 4.8, en particulier les cas illustrés en (a) et (c). La sélection de l'observateur dans le cas (a) a pour conséquence de laisser une possibilité de détection même si l'observateur ne couvre pas l'intégralité des sous-classes d'attaque possibles. Il est intéressant de noter que les conséquences de ce choix dépendent du type d'opérateur parent de l'action en question. S'il s'agit d'un opérateur *OR*, alors cela signifie que l'action n'est pas nécessairement réalisée et par conséquent la couverture partielle de cette possibilité a une conséquence différente du cas où l'opérateur parent de l'action est un opérateur autre que *OR*.

4.4.3 Identification des messages

Une fois l'association entre les observateurs et les actions réalisées, il est possible d'obtenir l'ensemble des spécifications des messages qui peuvent être levés à l'exécution, lors de la détection de chaque action. Chaque observateur pouvant générer des messages dans divers formats et contenant des informations spécifiques, une consultation de la base de connaissances est nécessaire pour obtenir d'une part le nom du format utilisé, d'autre part l'ensemble des champs présents dans le message et qui seront utilisés lors de la corrélation. En pratique, cette étape mène à la construction d'une structure appelée arbre de corrélation. Cette structure est obtenue à partir de l'arbre d'actions avec observateurs en remplaçant chaque feuille par un sous-arbre contenant les caractéristiques des messages possibles pour l'action courante.

A Structure d'une feuille représentant un message

Ce message est spécifié par le nom du format dans lequel il est créé par l'observateur, l'identifiant du message et l'ensemble des champs observables effectivement par l'observateur. L'identifiant

peut avoir différentes natures : dans le cas idéal, chaque message passe par une procédure de normalisation dans lequel un type d'action issu de l'une des taxinomies (CAPEC ou CEE) lui est attribué. Dans ce cas, l'identifiant du message correspond au nom de l'action. Dans le cas où cette normalisation n'est pas présente, cet identifiant référence la ou les noms des règles de détection de l'observateur permettant de lever l'événement (par exemple les identifiants de règles Snort ou de règles OSSEC).

B Cas de plusieurs messages ou observateurs possibles pour une action

Dans le cas où plusieurs observateurs peuvent détecter l'action ou bien lorsqu'un observateur donné peut détecter une action de différente manière, menant potentiellement à une levée de plusieurs messages de structure différente, un opérateur *OR* spécifique est ajouté avec autant de sous-branches qu'il y a d'observateurs.

Ce choix permet de valider la détection si au moins l'un des observateurs lève un message [9].

C Élimination des règles de détection inutiles

Cette sous-section concerne les actions d'attaques, les actions normales ne présentant pas la même ambiguïté. Il est possible qu'un observateur dispose de règles de détection permettant de détecter des classes d'attaques spécifiques pour lesquels le nœud surveillé n'est pas vulnérable. Afin de considérer uniquement les règles de détection pertinentes, une classification doit être réalisée. Une règle de détection *Rule* peut appartenir à l'une des catégories qui suit en fonction de la classe d'attaque *Att* à laquelle elle est associée dans la base de connaissances via le prédicat *detects(Rule, Att)* :

- (a) la règle est une règle générique qui n'est pas liée à une vulnérabilité mais peut détecter une classe d'attaque donnée. Par exemple, une règle comptabilisant le nombre de caractères spéciaux dans une requête SQL et levant une alerte si un seuil est dépassé permet de détecter des attaques appartenant à la classe *SQL injection* mais ne peut pas être associée à une vulnérabilité particulière. Un autre exemple est une attaque par force brute sur un service réseau ;
- (b) la règle est étroitement liée à la détection d'une vulnérabilité identifiée et ne peut pas permettre de détecter l'exploitation d'une autre vulnérabilité, même voisine. Par exemple, une règle Snort permettant de détecter une exploitation du CVE-2014-6271 (shellshock) est liée à une sous-classe de la classe d'attaque *injection* spécifique à l'exploitation de cette vulnérabilité ;
- (c) la règle permet de détecter une classe d'attaque qui peut exploiter un ensemble de vulnérabilités. Par exemple une règle permettant de détecter l'envoi d'un shellcode écrit pour une architecture de type Intel (exemple issu de l'article sur M4D4 [MMDD09]).

Les règles appartenant à la catégorie (a) doivent selon nous être conservées car elles sont suffisamment génériques pour permettre une détection. Les règles de type (b) et (c) sont candidates à une sélection en fonction de la vulnérabilité du nœud cible. Résoudre les cas (b) et (c) consiste à vérifier s'il existe une ou plusieurs vulnérabilités exploitées par la classe d'attaque *Att* et affectant le nœud ciblé. Dans ce cas, une règle activée *Sig* d'un observateur par rapport à la classe d'attaque *Att* ciblant le nœud *H* est pertinente dans le cas où une classe d'attaque exploite exclusivement une vulnérabilité *Vuln* à laquelle *H* est potentiellement vulnérable :

```
is_relevant(Sig,Att,H):-   detects(Sig, Att), exploits(Att, Vuln),
                             not(not_vulnerable(H, Vuln)).
```

9. Le mode de fonctionnement des plans de reconnaissance implique que par construction, un plan ne contiendra jamais deux éléments fils d'un *OR*. Si deux messages dont le parent direct est un *OR* sont réellement levés, ces derniers vont donc donner lieu à la création de deux alertes distinctes. Cela crée ainsi plusieurs alertes qui sont éligibles pour une agrégation (au titre d'une fusion ou d'une reconstruction d'attaque élémentaire). Pour éviter cela, une solution consiste à remplacer l'opérateur *OR* par un opérateur spécifique dont la traduction sous forme d'automate est expliquée en annexe A.1

D Représentation formelle de l'étape

Formellement, cette étape revient à construire une structure arborescente appelée arbre de corrélation CT à partir de l'arbre d'actions instancié et associés aux observateurs $at|_{obs}$. Pour cela, la fonction $f_{messages}$ qui associe à un observateur o et à une action a l'ensemble des messages générés par l'observateur lors de la détection de l'action est définie. Soit également $F_{messages}$ la fonction qui associe à un observateur et une action à un arbre qui est défini par :

(1) Si $Card(f_{messages}(o,a)) > 1$, alors $F_{messages}(o,a) = \{OR, [m \mid \forall m \in f_{messages}(o,a)]\}$

(2) Si $Card(f_{messages}(o,a)) = 1$, alors $F_{messages}(o,a) = f_{messages}(o,a)$

La fonction f_{CT} associe à une action a un arbre correspondant aux observations possibles de cette action.

(3) $f_{CT}(a) = \{OR, [\ F_{messages}(obs,a) \mid obs \in f_{obs}(a)]\}$

(4) Si $Card(f_{obs}(a)) = 1$ alors $f_{CT}(a) = F_{messages}(f_{obs}(a), a)$.

(5) Si $Card(f_{obs}(a)) = 0$ alors $f_{CT}(a) = \{UNOBSERVABLE\}$.

Le cas (3) correspond au cas général dans lequel un opérateur *OR* est utilisé pour lier les arbres d'observations de chaque observateur. Dans le cas où un seul observateur est présent, l'application du cas général mène à la construction d'un opérateur *OR* disposant d'une unique branche. Le cas (4) permet d'éviter cette construction. Le cas (5) permet d'expliciter l'absence d'observation possible pour une action par l'intermédiaire d'un indicateur nommé *UNOBSERVABLE*.

E Exemple

La figure 4.10 illustre la construction de l'arbre de corrélation dérivé de l'arbre présenté dans la figure 4.9. La structure de cet arbre est constituée d'un opérateur *OR* racine dont les branches correspondent à différents chemins d'attaques. Les opérateurs *OR* en pointillés sont ajoutés lors de cette étape et sont utilisés lorsqu'une action dispose de plusieurs observateurs. C'est par exemple le cas sur la branche de gauche, les deux premières actions pouvant être vues par deux observateurs. Le tableau 4.3 présentes les faits utilisés dans cet exemple.

Champs disponibles dans les messages de *snort-1*	*provides_field(snort-1,'tgt.port',injection)*
	provides_field(snort-1,'tgt.id',_)
	provides_field(snort-1,'src.id',_)
Champs disponibles dans les messages de *ossec-1*	*provides_field(ossec-1,'tgt.soft',injection)*
	provides_field(ossec-1,'tgt.port',injection)
Formats des messages	*message_format(snort-1, 'snort_alert_fast')*
	message_format(ossec-1,'ossec_syslog')

TABLE 4.3 – Faits utilisés pour la détermination des messages et des formats.

Dans chaque message, le champ *sensor* correspond à l'identifiant de l'observateur pouvant générer le message.

Le format dans lequel les messages sont attendus est également précisé. Pour décharger la figure, ces derniers sont présentés séparément. Les messages générés par l'IDS *snort-1* sont attendus dans le format *snort_fast*, alors que ceux issus d'Ossec sont générés en syslog.

F Discussion

Cette étape est une substitution directe des feuilles de l'arbre obtenu à l'étape précédente. Les deux étapes d'identification des observateurs et d'identification des messages sont séparées dans cette présentation, mais rien n'oblige à en faire deux étapes distinctes.

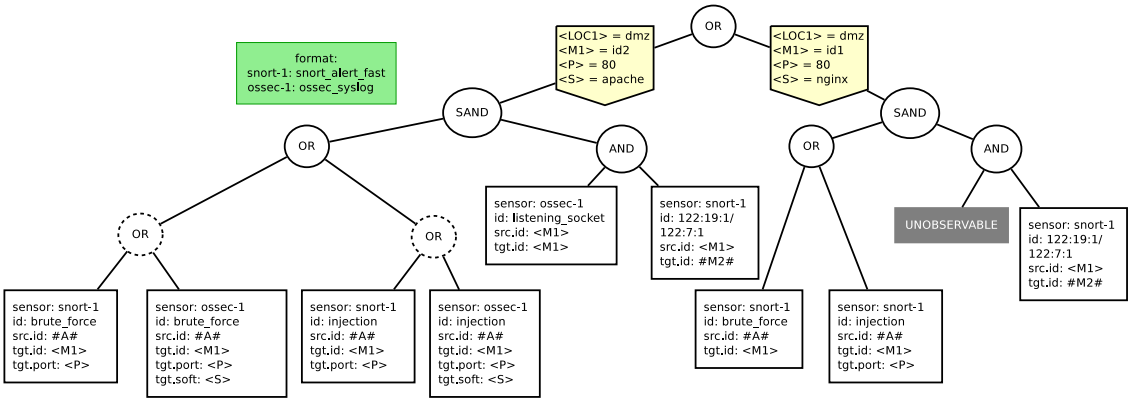


FIGURE 4.10 – Construction de l'arbre de corrélation. Chaque feuille de l'arbre est une représentation des observateurs aux différentes actions de l'arbre d'action. Deux identifiants de règles Snort sont rassemblés au sein d'un même message pour une représentation plus compacte

Un arbre de corrélation est constitué d'un grand nombre d'éléments redondants, en particulier lorsqu'un même observateur dispose de plusieurs règles pour détecter une action. Dans ce cas, l'unique différence entre les messages correspond aux identifiants de règles. S'il s'agit effectivement de l'unique différence, une factorisation est possible en regroupant les différents identifiants de règles de détection sous la forme d'une liste ou d'un ensemble, permettant ainsi de réduire la taille de l'arbre. Cette transformation a un intérêt si le langage de corrélation dans lequel cet arbre est traduit est capable de gérer la représentation de valeurs de champs sous forme d'ensemble de valeurs.

Il est également intéressant de noter les conséquences du choix de mettre de côté des règles de détection se référant à des vulnérabilités non présentes sur le système. Cette élimination permet de réduire la taille de l'arbre avec des feuilles inutiles et permet dans certains cas de minimiser le nombre de faux positifs. Cependant, cette élimination nous prive de la détection d'une tentative d'attaque exploitant une vulnérabilité non présente sur une machine. L'attaque échoue mais a bien eu lieu, ce qui est intéressant à prendre en compte selon le contexte. Un serveur directement accessible depuis l'extérieur subit des attaques automatisées et permanentes et dans ce cas l'élimination des règles non pertinentes est justifiée. Par contre, pour une machine interne protégée des attaques externes, cette détection de tentatives d'attaques peut être pertinente.

4.4.4 Traduction dans le langage de corrélation cible

L'objectif est de traduire la règle de corrélation abstraite exprimée au travers de l'arbre de corrélation dans un langage spécifique à un corrélateur. Dans notre cas, nous utilisons le langage ADeLe ([TVM04]). Plus précisément, la règle générée correspond à la partie *DETECT* d'une règle de corrélation ADeLe qui est composée de trois parties : *ENCHAIN*, *FILTER* et *CONSTRAINTS*.

A Construction des filtres

Chaque message issu de l'arbre de corrélation est transformé en filtre. Un filtre représente un type spécifique d'événement pouvant faire avancer la reconnaissance d'un scénario d'attaque. Il est constitué d'un identifiant, d'un nom de format et d'un ensemble de champs. Dans notre cas, un identifiant unique est généré pour le filtre, puis le nom du format est extrait du message. Les champs du filtre sont constitués des champs du message qui ont été instanciés. En ADeLe, un filtre représente un type particulier d'événement. Il est alors possible de définir une ou plusieurs instances d'événement correspondant à ce type d'événement. Dans le cas où plusieurs messages de l'arbre de corrélation sont constitués du même contenu, un unique filtre est créé et deux instances d'événements sont alors définies. Dans l'exemple 4.2, *TYPE1* représente un type et *E1* est une instance d'événement de type *TYPE1*.

Listing 4.2 – Définition d'un filtre.

```
<EVENTS>
TYPE1 : snort_fast {
        sensor == "snort-1"
        id ==  "brute_force"
        tgt.id == "id2"
        } E1

TYPE2 : ossec_syslog {
        sensor == "ossec-1"
        id ==  "brute_force"
        tgt.id == "id1"
        tgt.port == 80
        } E2
        ...
</EVENTS>
```

B Construction de l'enchaînement des événements

L'enchaînement des différentes instances d'événements s'exprime en ADeLe dans la section ENCHAIN. Pour réaliser cet enchaînement, un parcours en profondeur de l'arbre de corrélation est réalisé. Chaque opérateur de l'arbre de corrélation est traduit par un opérateur temporel en ADeLe. La correspondance entre les opérateurs est donnée par le tableau 4.4. Comme expliqué au paragraphe précédent, chaque message de l'arbre de corrélation est remplacé par une instance du filtre ADeLe correspondant. Par exemple, la figure 4.11 illustre la correspondance entre la définition de l'enchaînement des événements en ADeLe (2) et l'articulation des différentes feuilles de l'arbre de corrélation (1) correspondant. Dans cet exemple, les lettres sont utilisées pour identifier les instances d'événements.

Arbre de corrélation	ADeLe
SAND	Sequence
AND	NonOrdered
OR	OneAmong

Table 4.4 – Correspondance entre les opérateurs de l'arbre de corrélation et les opérateurs du langage de corrélation ADeLe.

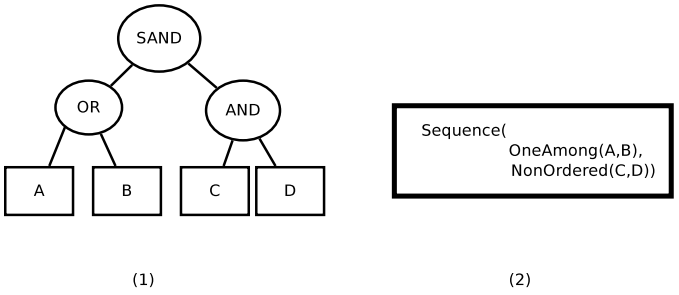


Figure 4.11 – Traduction de l'enchaînement des événements en ADeLe.

C Construction des contraintes entre les événements

La section CONSTRAINTS modélise les contraintes qui lient les instances de certains événements. Dans notre cas, une contrainte entre deux instances d'événement est générée lorsque les

messages correspondants dans l'arbre d'actions référencent des variables dynamiques communes. En effet, cela signifie qu'au moment de la détection, l'égalité entre les valeurs des deux champs respectifs des deux instances de messages est attendue. En pratique, cela se traduit en ADeLe par une contrainte du type m1.f1 == m2.f2, avec m1 et m2 les deux instances de messages et f1 et f2 les champs de ces messages respectifs.

Génération exhaustive ou sélective des contraintes L'utilisation d'une même variable dynamique à plus de deux reprises au sein d'un arbre d'actions rend nécessaire la traduction de la dépendance entre ces variables sous la forme de contraintes entre les champs d'un ou plusieurs messages. Plusieurs approches sont alors possibles pour générer ces contraintes. Dans une approche exhaustive, toutes les combinaisons possibles sont conservées(en supprimant les symétries du type a == b et b == a). La génération de l'intégralité des contraintes est possible si le nombre d'occurrences d'une variable dynamique n'est pas trop important. Au moment de la détection, si une contrainte ne peut être satisfaite à cause de l'absence d'un événement, alors cette contrainte n'est pas évaluée. Par conséquent, la construction de contraintes entre des messages qui s'excluent mutuellement ou qui peuvent ne pas être tous produits ne bloque pas la détection du scénario d'attaque.

Cas des contraintes qui impliquent des messages fils de OneAmong Un choix dans l'algorithme de détection utilisé par le corrélateur GNG rend inutile certaines contraintes entre champs d'événements.

Le principe de fonctionnement de cet opérateur a pour conséquence qu'une contrainte entre les champs des instances d'événements A et B au sein de l'enchaînement *OneAmong(A, B)* ne peut être satisfaite car A et B ne peuvent se retrouver dans un même plan simultanément. De fait, l'impossibilité de disposer d'une instance de A et de B dans le même plan ne mène pas à un blocage lié à la non-résolution de la contrainte.

Une contrainte a un impact uniquement lorsque l'opérateur parent commun entre deux feuilles est un opérateur *Sequence* ou *NonOrdered*.

D Traitement des actions non observables

L'action non observable prend la forme d'une feuille de type *UNOBSERVABLE* dans l'arbre de corrélation de notre exemple. Contrairement aux autres nœuds de l'arbre de corrélation, il ne s'agit pas d'un message qui peut être généré par un observateur. Cette action ne doit donc pas être traduite dans la règle de corrélation. Cependant, une suppression simple de cette action lors de l'étape de traduction dans le langage de corrélation peut mener à une règle de corrélation erronée. Cette section s'intéresse à identifier les cas problématiques liés aux actions non observables et à étudier les solutions applicables pour l'obtention de règles de corrélation correctes.

La figure 4.12 illustre les différents moyens de traduire l'arbre de corrélations contenant une action non observable en ADeLe. Chaque arbre est une représentation de l'arbre d'ordonnancement des événements en ADeLe. Dans les arbres (i) et (ii), le message fictif U correspond à une action non observable qui doit être transformée. Les arbres (1) à (3) correspondent à des transformations possibles de l'arbre (ii) et l'arbre (0) à une transformation possible de l'arbre (i).

Le tableau 4.5 résume les conséquences de chaque choix de traitement :

— (0) une suppression simple dans le cas où l'opérateur père est une *Sequence* (ou un *NonOrdered*) ;

— (1) une suppression simple de l'élément non observable peut bloquer la reconnaissance du scénario. Dans l'exemple donné, l'occurrence de la suite de message *A- ?-E*, où *?* correspond à une action de l'attaquant non détectée entraîne la non-reconnaissance du scénario ;

— (2) la suppression de l'opérateur *OneAmong* et de ses fils permet de résoudre le problème des faux négatifs issus de la transformation (1). Par contre, cette suppression peut mener à la détection de séquences d'actions commençant par A et finissant par E qui sont légitimes. De plus, lorsque les messages levés correspondent à la séquence *A-B-E* ou *A-D-E*, les messages B et D sont perdus et ne participent plus au scénario ;

— (3) pour mettre en commun les cas (1) et (2), une approche naïve consisterait à utiliser un opérateur *OneAmong* joignant les arbres (1) et (2). Cependant, cette construction a un

inconvénient car tous les scénarios reconnus par le sous-arbre (1) le sont également par (2). Afin de restreindre l'arbre (2) à la reconnaissance d'un enchaînement *A*-*E* sans la présence de *B* ou de *D* entre *A* et *E*, une nouvelle branche est introduite et comporte l'opérateur *Without* qui permet d'éliminer les cas de reconnaissance redondants Un cas de figure portant sur l'arbre (i) lorsque les nœuds *E* et la branche du *OneAmong* sont inversés ne peut pas subir une transformation du type (3) car l'opérateur *Without* doit s'appliquer sur une branche impliquant au moins une séquence de deux événements (ou à un *NonOrdered*).

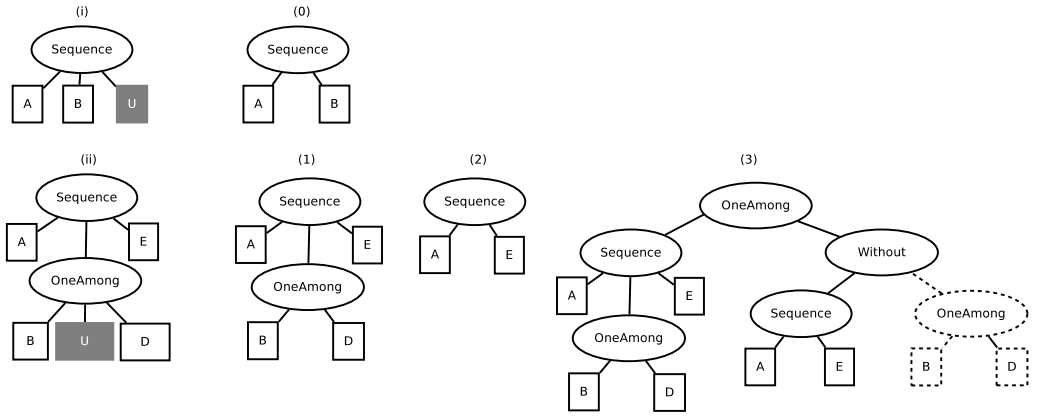


FIGURE 4.12 – Résolutions possibles des cas d'actions non-observables. Les nœuds en pointillé de l'arbre (3) correspondent à l'enchaînement qui ne doit pas être présent pendant la reconnaissance de la séquence $A - E$.

cas	traitement	conséquences
(1)	Suppression simple	Faux négatifs possibles
(2)	Suppression de l'opérateur père et de ses nœuds fils	Faux positifs possibles et élimination de messages potentiellement importants
(3)	Choix dynamique entre le cas (1) et (2)	Faux positifs possibles mais permet la détection d'un scénario complet
(4)	Suppression simple	Faux positifs possibles

TABLE 4.5 – Conséquences du choix de traitement des actions non observables.

E Exemple

La figure 4.13 est une représentation arborescente de l'enchainement des événements qui correspondent aux messages de l'arbre de corrélation de la figure 4.10. La transformation des opérateurs logiques en opérateurs temporels est effectuée. L'action non observable est éliminée ainsi que l'opérateur *AND/NonOrdered* devenu inutile puisqu'il n'a plus qu'un unique nœud fils (lien en gras sur la figure 4.13).

Même sur cet exemple simple, la règle de corrélation résultante est complexe. L'un des intérêts de la méthode de génération est qu'elle facilite la prise en compte de l'évolution du système dans la règle de corrélation. Prenons deux exemples d'évolutions.

(1) Un nouveau serveur est déployé au sein du même sous-réseau que les deux précédents et est également accessible depuis l'extérieur. Cette machine supplémentaire peut désormais constituer une nouvelle cible et ainsi conduire à une nouvelle possibilité d'instanciation du scénario d'attaque présenté dans la figure 4.4. Il suffit cependant de relancer la génération à partir de ce même arbre d'actions pour construire une règle de corrélation mise à jour. Elle correspond à un arbre de corrélation constitué d'un sous-arbre supplémentaire décrivant l'exploitation de cette nouvelle machine.

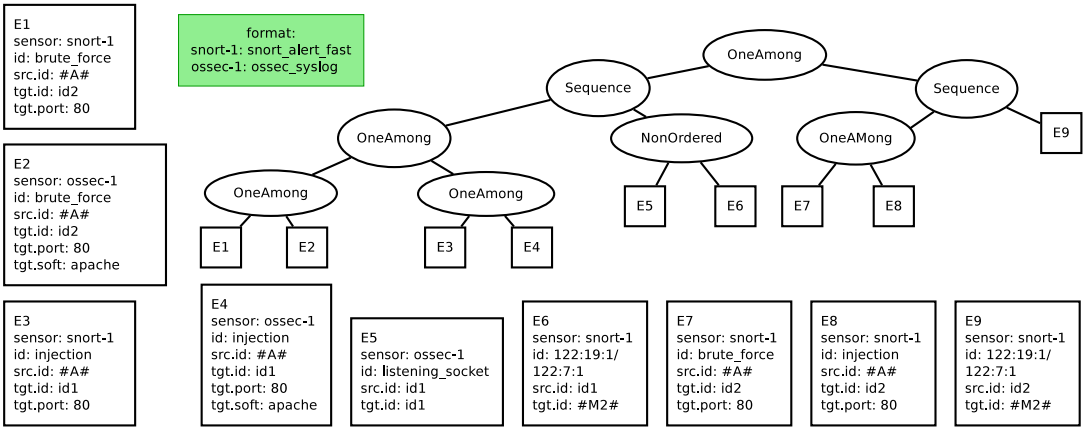


FIGURE 4.13 – Ordonnancement des messages et filtres correspondant à la règle de corrélation ADeLe.

(2) Un HIDS est finalement installé sur le second serveur. Dans ce cas, une nouvelle génération mènera à une règle de corrélation correspondant à un arbre de corrélation constitué de deux sous-arbres similaires (ils diffèrent seulement par les identifiants des nœuds et des HIDS).

Dans tous les cas cités, il est supposé que la base de connaissances est mise à jour (dans l'idéal de manière automatique) et contient bien les éléments qui correspondent au système déployé.

F Discussion

Lors de cette traduction en langage ADeLe, un sous-ensemble du langage est utilisé. Les opérateurs *Repeat* et *Without* ne sont ainsi pas utilisés. Le *Repeat* n'est qu'un moyen de factoriser une séquence. Quant au *Without*, il est difficile à traduire dans l'arbre d'actions car cet opérateur sert généralement à modéliser des contre-mesures possibles entravant la progression d'un scénario d'attaque. Les contraintes entre les champs des événements sont réduites à des égalités. Les contraintes de délais minimaux ou maximaux entre l'occurrence de deux événements ne sont pas définies car cette information n'est pas présente dans l'arbre d'actions initial. Ces contraintes temporelles permettent de réduire le nombre de plans possibles lors de la détection.

4.4.5 Gestion des classes d'équivalence

A Classes d'équivalence et identification des acteurs

Il est possible qu'un ensemble de machines appartenant à une même classe d'équivalence fasse partie des acteurs du scénario (voir section 3.2.2 du chapitre 3). Dans ce cas, au lieu d'une sélection de toutes les machines une à une, ce qui ferait exploser la taille de la règle de corrélation, le mécanisme permet de manipuler la classe d'équivalence à laquelle les machines appartiennent. Pour cela, un identifiant unique à la classe est généré et associé au champ identifiant du nœud de l'action. Un champ supplémentaire qui référence l'identifiant de la classe d'équivalence est associé à cet identifiant.

En reprenant l'exemple de la figure 4.4 et en supposant que le système contienne par exemple dix machines qui soient équivalentes à la machine identifiée initialement par *n1*, la figure 4.14 correspond alors au résultat de l'étape d'identification des acteurs lorsque les classes d'équivalences sont utilisées. Au lieu de générer un arbre dont la racine est reliée à une forêt d'une dizaine d'arbres, la structure est similaire à celle de la figure 4.6. La différence se situe au niveau de la valeur de la variable statique $< M1 >$ qui correspond à un identifiant arbitraire (ici *eq01*) associé à une classe d'équivalence spécifique, ici identifiée par *equiv-class1*.

L'identification des observateurs se fait de manière transparente car toutes les machines d'une même classe d'équivalence sont observées de la même manière (il s'agit d'une des conditions pour la construction de telles classes d'équivalence).

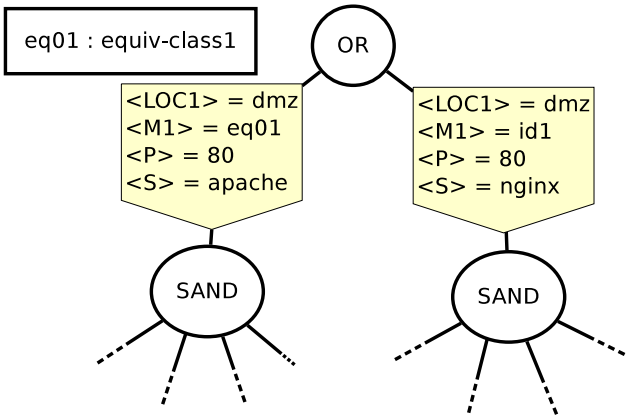


FIGURE 4.14 – Exemple d'instanciation mettant en jeu un élément d'une classe d'équivalence.

B Classes d'équivalence et génération des règles de corrélation

Au moment de la détection, un observateur n'a pas nécessairement connaissance de l'appartenance d'une machine à une classe d'équivalence. Par conséquent, le message généré fait bien référence à une machine et non à une classe d'équivalence. Pour permettre au corrélateur de réaliser cette association entre une machine et sa classe d'équivalence, plusieurs cas sont envisageables : dans le cas où le langage de corrélation supporte des opérations de vérification d'appartenance d'un élément à un ensemble, il suffit de créer la liste L des identifiants des machines équivalentes et de spécifier l'expression $M \in L$ comme contrainte pour la valeur du champ correspondant dans le message. Cette expression n'est pas possible en ADeLe. Une seconde solution consiste à utiliser une expression régulière pour énumérer tous les identifiants des machines de la classe d'équivalence. En reprenant l'exemple de la figure 4.14 et en notant $n_1, \ldots n_{10}$ les identifiants des dix nœuds sélectionnés et appartenant à la même classe d'équivalence, le listing 4.3 présente un extrait de la règle de corrélation ADeLe correspondante à cette seconde solution.

Listing 4.3 – Expression régulière pour spécifier un identifiant.

```
<EVENTS>
        T1 : sensor1 {
          src.id ~= "n1|n2|n3|n4|n5|n6|n7|n8|n9|n10"
          ...
          } E1
          T2 : sensor2 {
          src.id ~= "n1|n2|n3|n4|n5|n6|n7|n8|n9|n10"
          ...
          } E2
</EVENTS>
<CONTEXT>
        E1.src.id == E2.src.id
</CONTEXT>
```

4.4.6 Discussion

Cette section a présenté le fonctionnement des étapes de transformation automatique permettant de passer d'un arbre d'actions à une règle de corrélation. Chaque étape permet de spécialiser l'arbre initial en fonction des spécificités de l'environnement cible. Le succès de cette opération automatique de génération dépend de plusieurs facteurs. L'arbre d'actions doit être correctement spécifié et correspondre à un scénario réalisable sur le système pour que le processus produise une règle de corrélation pertinente. Il est également nécessaire que les éléments contenus dans la base de connaissances correspondent bien à la réalité, sous peine de générer des règles de corrélation incluant des combinaisons obsolètes ou bien au contraire des règles de corrélation qui ne contiennent qu'un sous-ensemble des scénarios d'attaque possibles.

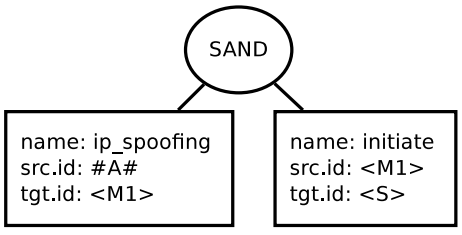
4.5 Exemples d'utilisation et limites

L'objectif de cette section est d'illustrer les possibilités offertes par le langage d'actions et de mettre en avant certaines limites de celui-ci.

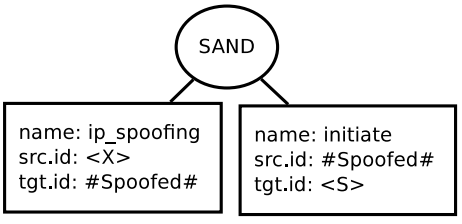
4.5.1 Modélisation de scénarios d'attaque

Cette sous-section a pour but d'illustrer la construction d'arbres d'actions modélisant des grandes catégories de scénario d'attaques ou d'actions d'attaque.

Usurpation d'identité La figure 4.15 illustre un cas difficile à modéliser car il met en jeu une usurpation d'identité. Ce type d'action implique généralement plus de deux acteurs : l'attaquant, l'entité usurpée et l'entité avec laquelle l'attaquant communique en utilisant l'identité usurpée. Dans le premier exemple (figure 4.15(a)), la première action est l'usurpation d'identité. La source de l'attaque est la machine attaquante et la cible est la machine dont l'adresse est usurpée. Dans la seconde action, nous cherchons à modéliser une connexion de l'attaquant à une machine S en utilisant l'identité de $M1$. Le défaut de cette modélisation est qu'elle ne permet de modéliser qu'une usurpation d'identité d'un nœud qui est présent dans le système. Il s'agit de la conséquence de l'emploi de la variable statique $< M1 >$. Dans l'exemple de la figure 4.15(b), cette victime est modélisée par une variable dynamique qui n'est résolue qu'au moment de la détection. Ce second exemple modélise également l'attaquant avec une variable statique, ce qui suppose qu'il s'agit d'une machine existante qui a été compromise.



(a) L'entité dont l'adresse est usurpée est représentée par une variable statique



(b) Modélisation de l'adresse usurpée par une variable dynamique

FIGURE 4.15 – Comparaison de deux modélisations possibles pour décrire une usurpation d'identité.

Scénario sans action d'attaque explicite Dans certain cas, le scénario d'attaque peut être constitué d'un ensemble d'actions légitimes lorsqu'elles sont considérées individuellement. L'exemple présenté ici montre que l'expressivité du langage d'actions permet de mettre en œuvre un tel scénario. La figure 4.16 illustre un cas de scénario d'attaque modélisé uniquement avec des actions légitimes. Il s'agit d'un attaquant ayant réussi à obtenir les identifiants administrateurs d'une machine (par des moyens non détectables ou une précédente attaque). Il se connecte depuis l'extérieur sur la machine $M1$, utilise ses droits pour créer un nouvel utilisateur (pour avoir une solution dans le cas où le mot de passe root changerait), puis il utilise les clefs SSH stockées sur la machine pour se connecter à une machine $X2$ située dans la même zone que $M1$. Il crée alors un fichier d'archive contenant des informations trouvées sur cette machine.

4.5.2 Limites

A Limites liées aux noms des actions

Techniques d'attaques, prérequis et conséquences CAPEC est organisé autour de classes qui représentent des techniques d'attaques. Lors de la spécification de certains scénarios d'attaque, la technique n'est pas nécessairement plus importante qu'un prérequis ou une conséquence de l'attaque. Par exemple, une action peut réussir lorsque l'attaquant exécute du code arbitraire sur une machine. Cette exécution de code arbitraire n'est pas une technique d'attaque mais une conséquence

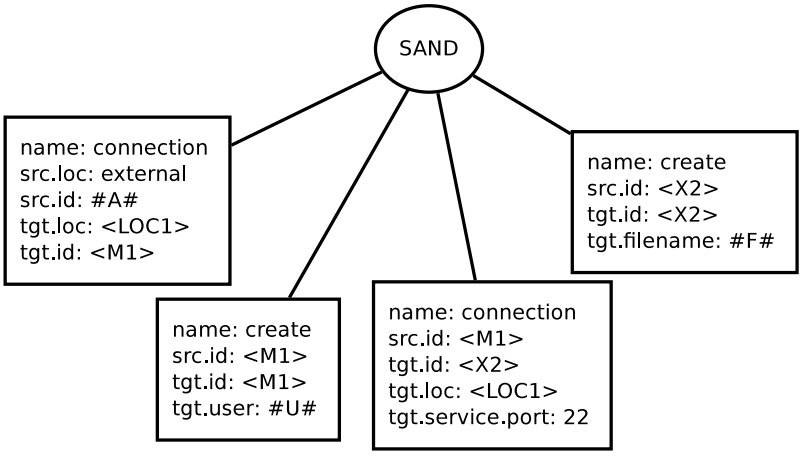


FIGURE 4.16 – Exemple de scénario faisant uniquement intervenir des actions légitimes.

d'une action d'attaque. L'utilisation de cette caractéristique demande la création d'une nouvelle classe d'attaque dont héritent toutes les attaques élémentaires qui sont réalisables à distance et qui permettent d'obtenir les privilèges d'un utilisateur.

Différence entre une tentative de connexion et une connexion réussie Du point de vue de la corrélation, il est intéressant de prendre en compte des connexions qui ont été rejetées par un pare-feu. Ces dernières peuvent correspondre à différentes tentatives d'un attaquant pour établir une reconnaissance (scan) ou tenter d'établir un canal de communication non autorisé. Une solution de modélisation consiste à définir une tentative de connexion comme un type d'action particulier qui est traité de telle sorte que les contraintes d'accessibilité ne soient pas bloquantes lors de l'étape d'identification des acteurs. Cependant, l'écriture d'un scénario d'attaque suppose de prendre le point de vue de l'attaquant. Or la définition d'une action spécifiant une tentative de connexion correspond plus au point de vue du défenseur qu'à celui de l'attaquant. Cette remarque peut être prise en compte en utilisant la règle suivante : lorsqu'une action spécifie une communication entre deux nœuds, qu'il existe des nœuds A et B qui ne sont pas sélectionnés par le processus d'identification des acteurs car il existe un pare-feu bloquant l'accès entre A et B, alors au lieu de rejeter entièrement cette combinaison, l'action est automatiquement remplacée par une action spécifiant une tentative de connexion entre A et B.

Cependant, l'introduction d'une telle règle pose deux nouveaux problèmes. Le premier problème est lié au risque de génération d'un grand nombre d'actions de tentatives de connexion. Le second concerne la réalisation compromise de la suite du scénario d'attaques si la connexion est rejetée. Dans une séquence de deux actions (A1 et A2) correspondant respectivement à la réalisation d'une injection SQL sur un serveur distant suivie d'une élévation de privilège sur ce serveur compromis, l'échec de connexion initial implique un échec de l'action A1, qui est un prérequis pour réaliser l'élévation de privilèges (A2). Si la règle de corrélation contient le message attendu lors de la réalisation de cette seconde action (A2), alors la reconnaissance est bloquée car A2 ne peut survenir que lorsque A1 réussit. La suppression de cette action A2 dans les cas où l'action A1 est remplacée par une action de tentative de connexion permet d'adapter la règle de corrélation à la détection des tentatives de réalisation du scénario d'attaque.

Choix entre plusieurs noms d'actions Une difficulté apparaît lorsque plusieurs noms peuvent être attribués à une action. Par exemple, la spécification d'une action consistant à modifier de manière malveillante un fichier partagé est modélisé directement avec un nom d'action correspondant au type d'attaque, c'est-à-dire *Modify Shared File* dans CAPEC. Cependant, rien n'interdit d'utiliser une action normale pour modéliser cette action, par exemple en utilisant le nom *modify* du dictionnaire CEE. Or ces deux actions ne sont pas équivalentes car l'action *modify* est plus générique et peut potentiellement causer des faux positifs. Pour limiter ce risque, il est donc souhaitable d'utiliser les noms d'actions les plus précis possibles lorsque le choix est possible.

B Limites liées au caractère statique de la base de connaissances

Lors de la génération des règles de corrélation, la base de connaissances est supposée figée. Cette hypothèse se répercute sur la difficulté de modélisation de certaines situations impliquant une modification de l'état du système au cours de l'attaque.

L'attaquant modifie l'adresse d'une machine compromise Ce cas peut mettre en défaut l'identification d'une machine. Cette action est considérée comme un cas particulier de l'action d'usurpation d'identité.

Création d'un nouveau nœud par l'attaquant L'attaquant peut chercher à créer une nouvelle machine (par exemple une machine virtuelle) pour lui permettre de continuer l'exploitation du système. Cette action peut également correspondre à un attaquant physiquement présent sur le site qui arrive à brancher un nœud sous son contrôle et à le cacher physiquement. Cette action peut se modéliser avec le nom d'action *new_node* et l'utilisation d'une variable dynamique pour identifier ce nœud. Ce nom d'action *new_node* n'existe pas dans le dictionnaire de CEE. Cela ne remet pas en question son utilisation car le dictionnaire d'actions de CEE est utilisé ici comme un point de départ et ne doit pas constituer une limitation.

Compromission d'un pare-feu et modification des flux autorisés Ce cas survient lorsqu'un pare-feu bloque les connexions entre un sous-réseau A et un sous-réseau B et que le scénario d'attaque implique la compromission du pare-feu par l'attaquant et la modification des règles de filtrage, ce qui lui permet d'accéder directement au sous-réseau B depuis le sous-réseau A. Une action impliquant une communication entre un nœud A1 contrôlé par l'attaquant et un nœud B2 situé dans le sous-réseau B est considérée comme impossible lors du processus d'identification des acteurs car dans la base de connaissance, le pare-feu bloque toujours cette connexion.

Une manière de résoudre ce problème est de considérer que cette action doit être l'action finale d'un scénario car c'est un événement suffisamment important si d'autres actions d'intrusion ont précédemment été détectées. Une autre approche visant à résoudre ce cas consiste à définir un type d'action spécifique, par exemple en préfixant l'action par *ignore_firewall_rule*. Ce type d'action donne alors une directive lors de l'identification des acteurs pour ne pas prendre en compte les contraintes de filtrage.

L'attaquant réalise un déni de service sur un observateur Le comportement souhaitable serait de générer une règle de corrélation référençant uniquement des messages qui peuvent être réellement émis. Un observateur mis hors service par une action de l'attaquant a pour conséquence de rendre impossible toute remontée d'information de la part de cet observateur sur la suite des activités de l'attaquant. Ainsi, si une telle action doit se produire, il est intéressant de considérer l'observateur comme actif sur un certain nombre d'actions puis inactif sur les actions qui suivent sa mise hors service. Le problème devient plus complexe lors de la prise en compte de toutes les combinaisons possibles avec les opérateurs logiques. Par exemple, lorsque l'action de désactivation est une action fille d'un opérateur *OR*, il n'y a aucune certitude sur la réalisation de l'action. Il faudrait alors définir un mécanisme permettant d'activer ou de désactiver des parties de la règle de corrélation à la volée en fonction de la réalisation de l'action problématique.

4.5.3 Discussion

Cette section a illustré par quelques exemples des cas spécifiques qui peuvent être modélisés avec le langage d'actions proposé. Certaines limites dans la modélisation ont également été mises en avant. En particulier, un défaut récurrent vient de l'incapacité du système à prendre en compte des évolutions dynamiques de l'environnement au cours de la réalisation d'un scénario d'attaque. D'un autre côté, permettre à la base de faits d'évoluer dynamiquement en fonction des actions de l'attaquant s'éloigne de l'idée de départ d'une base de faits qui décrit l'état du système tel qu'il est réellement pour se rapprocher d'un simulateur dynamique du système. Disposer d'un tel simulateur est intéressant mais s'éloigne de l'idée initiale d'une base de biens d'un système et demande un développement beaucoup plus poussé.

4.6 Conclusion

Ce chapitre a exposé les principes d'une méthode de génération de règles de corrélation à partir de la représentation d'un scénario d'attaque et d'une base de connaissances. Un langage d'attaque permet de spécifier les actions de l'attaquant au sein d'une structure reposant sur un arbre d'attaque. Cet arbre d'actions représentant un scénario d'attaque est croisé avec les informations pertinentes contenues dans la base de connaissances pour produire un arbre de corrélation. Cet arbre de corrélation est alors traduit dans un langage de corrélation spécifique (ici ADeLe). La maintenance des règles de corrélation est plus aisée du fait qu'une fois la base de connaissances modifiée (potentiellement par une mise à jour automatique), il suffit de générer de nouveau les règles à partir du même arbre d'actions qui demeure inchangé.

Une partie de la complexité de la construction des règles de corrélation est reportée dans la construction d'un arbre d'actions dérivé d'un arbre d'attaque. Plusieurs méthodes peuvent être adoptées pour spécifier de tels scénarios. (1) construire des scénarios d'attaque très génériques (reconnaissance suivie d'une compromission suivie d'une élévation de privilèges) ; (2) s'adapter plus spécifiquement aux particularités de certains scénarios d'attaque connus ; (3) utiliser les particularités du système surveillé et construire des scénarios issus d'une étude de risques de ce système. L'utilisation d'un outil de génération de graphes d'attaques peut donner une idée des chemins potentiellement empruntés par un attaquant. Une factorisation et généralisation de ce graphe permettrait d'obtenir un arbre d'actions pertinent pour un système donné.

Chapitre 5

Évaluation

Ce chapitre propose une évaluation du processus de génération de règles de corrélation. Dans un premier temps, nous nous intéressons aux différentes approches utilisées pour l'évaluation d'un processus de corrélation. Cette étude préliminaire est suivie d'une section consacrée à l'évaluation de l'approche dans un cas d'étude représentatif d'un système d'information d'entreprise. Cette première évaluation permet de tester l'applicabilité de la méthode et l'utilité des règles de corrélation générées sur un cas concret. Cette évaluation est réalisée dans un cas idéal dans lequel tous les composants du système sont connus et renseignés correctement dans la base de connaissances. C'est pourquoi l'évaluation menée ensuite porte cette fois-ci sur les conséquences potentielles des fautes pouvant affecter le bon déroulement de la génération des règles de corrélation. Une dernière partie étudie l'influence de la taille du système et de l'utilisation de classes d'équivalence sur la taille de l'arbre de corrélation.

5.1 Méthodes d'évaluations de la corrélation d'alertes

L'évaluation des systèmes de corrélation est une tâche complexe. Il est en effet nécessaire de mettre en œuvre ces systèmes dans des environnements représentatifs des réseaux d'entreprises classiques. Ces systèmes sont le siège d'activités normales. Cependant, des attaques suffisamment complexes pour nécessiter une étape d'identification doivent également se produire. Ces systèmes doivent générer une activité légitime par-dessus laquelle des scénarios d'attaque se produisent. Divers jeux de données ont été utilisés pour évaluer les processus de corrélation. Cette section propose de passer en revue leurs points forts et leurs faiblesses.

Un jeu de données idéal pour évaluer notre approche est caractérisé par : (1) une architecture connue et représentative d'un système d'information ; (2) la présence d'événements et d'alertes issus des observateurs placés dans le système ; (3) la présence d'au moins un scénario d'attaque non trivial dans les journaux ; (4) la présence de vie dans le réseau, vie qui peut générer du bruit et des fausses alarmes.

Ces caractéristiques se justifient respectivement par (1) la nécessité de la connaissance approfondie du système pour mettre en œuvre la méthode ; (2) la nécessité de disposer directement d'alertes ainsi que d'événements de plus bas niveau qui ne caractérisent pas nécessairement un comportement malveillant ; (3) un scénario impliquant un déplacement progressif de l'attaquant au sein du système est plus pertinent pour évaluer une technique de reconnaissance de scénario qu'un ensemble d'attaques frontales sur un unique serveur ; (4) la présence uniquement d'alertes pertinentes dans le jeu de données retire l'intérêt de la corrélation et ne rend pas le cas réaliste. Des alertes pertinentes doivent donc être noyées au sein d'alertes correspondant à des faux positifs ou des comportements occasionnellement déviants des utilisateurs du système.

5.1.1 Jeux de données DARPA

Les ensembles de données DARPA [1] 98, 99 et 2000 sont conçus initialement pour mettre à disposition un référentiel commun afin d'évaluer et de comparer les performances des IDS. Les données disponibles sont des captures réseaux prises à divers points du système ainsi que des fichiers d'audit de certains serveurs. Une partie identifiée des données ne contient pas de traces d'attaques, ce qui permet de mettre en œuvre des méthodes de détection qui reposent sur la connaissance du comportement normal du système.

Points forts Les données sont issues d'une architecture contenant un nombre important de machines. Un scénario d'attaque est réalisé sur ce système et de nombreux types d'attaques sont présents (reconnaissances, exploitation distante, élévations de privilèges).

Points faibles Initialement prévus pour tester les IDS, ces jeux de données proposent des journaux bruts. Ainsi, pour évaluer la corrélation d'alertes, il est nécessaire de configurer des IDS pour générer des alarmes [VS01]. L'évaluation porte alors non pas sur la corrélation seule, mais sur l'ensemble détection d'intrusion et corrélation d'alertes. Les auteurs de [KG13] insistent sur le fait qu'une comparaison équitable n'est pas possible sans deux environnements de tests identiques. Or dans ce cas, les résultats dépendent de la manière dont les IDS sont configurés.

Les données exploitables regroupent des fichiers de journalisation et des captures réseau. Les fichiers de journalisation locaux des machines sont donnés sans modifications. Cependant, il n'en est pas de même pour les captures réseau. Ces dernières sont construites autour d'un trafic de fond par-dessus lequel des traces d'attaques ont été ajoutées. Ce point constitue l'une des principales raisons invoquées par les critiques des jeux de données DARPA [McH00]. De plus, les attaques mises en œuvre dans le scénario utilisent des techniques utilisées dans les années 2000, mais ne reflètent plus les attaques actuelles qui sont centrées sur l'exploitation des services web ou des navigateurs web des clients.

5.1.2 Données issues des compétitions CTF (Capture the Flag)

Les deux jeux de données utilisés dans cette catégorie sont le CTF 9 de Defcon et la chasse au trésor (Treasure Hunt [Vig03]) de l'UCSB. Dans des compétitions CTF, plusieurs équipes s'affrontent pour exploiter des vulnérabilités sur le SI adverse afin de marquer des points. La chasse au trésor propose un scénario dans lequel deux groupes d'étudiants cherchent à compromettre un système pour réaliser des transactions bancaires frauduleuses. Plusieurs étapes sont nécessaires pour atteindre cet objectif et font intervenir en particulier des actions de reconnaissance, des exploitations de vulnérabilités WEB et des élévations de privilège.

Points forts : La chasse au trésor présente un scénario d'attaque complexe avec un objectif (réaliser des transactions bancaires). Les journaux d'événements n'ont pas subi de modifications (pas d'insertion de bruits de fond artificiel par exemple). Généralement, les jeux de donnés contiennent de nombreuses traces d'actions d'attaques.

Points faibles : Ces types de compétitions sont intéressants pour tester les capacités maximales des IDS ou corrélateur à absorber un trafic contenant un grand nombre d'attaques. En fait, la seule (ou principale) activité présente sur ces systèmes est celle des attaquants, ce qui limite leur caractère réaliste dans lequel un trafic légitime serait présent.

5.1.3 Pots de miel

Un pot de miel consiste en la mise en place de données ou de machines dont le but est de tromper un attaquant. Ce dernier réalise alors des actions sur ce pot de miel et, en fonction de l'interaction proposée par ce pot de miel, la démarche et les techniques de l'attaquant peuvent être analysées. Par exemple, un serveur web est mis en place sur Internet. Ce serveur est configuré

1. MIT Lincoln Laboratory, DARPA Intrusion Detection Evaluation, http ://www.ll.mit.edu/ideval/

pour journaliser un maximum d'actions et est surveillé par un NIDS. Le tout est placé derrière un pare-feu qui journalise également les connexions et tentatives de connexions effectuées. L'objectif est ensuite d'étudier les attaques qui y sont menées. Par exemple, les données issues du projet honeynet (en particulier le scan 17 [2]) sont utilisées.

Points forts : Les attaques sont réelles, plusieurs sources d'événements ou d'alertes sont disponibles (journaux du serveur web, du pare-feu et alertes levées Snort).

Points faibles : Tout d'abord, l'infrastructure est très limitée (un serveur, un pare-feu, un NIDS), ce qui limite l'apparition de scénarios d'attaque incluant des rebonds entre machines. Ensuite, la représentativité du trafic est limitée. Le trafic destiné au pot de miel est probablement généré en grande partie par des robots et des attaquants et il n'y a pas réellement de trafic légitime. Par conséquent, ce système n'est pas représentatif d'un système de production dans lequel un grand volume de trafic légitime est généré.

5.1.4 Discussion

Les différents jeux de données permettant de réaliser une évaluation des systèmes de corrélation présentent des faiblesses importantes. De plus, il est important de noter que l'évaluation des performances de détection seules ne permettent pas de rendre compte de l'intégralité du processus de génération de règles de corrélation. Les performances de détection constituent l'objectif final, mais notre principale contribution est centrée plus étroitement sur la manière d'obtenir des règles correctes à partir d'une spécification d'un scénario d'attaque et de la description du système à laquelle la règle est destinée.

La section suivante propose une application de notre approche à un système de manière à satisfaire au mieux les caractéristiques évoquées précédemment, c'est-à-dire la présence d'une architecture connue, la disponibilité d'alertes et d'événements, la présence d'un scénario d'attaque suffisamment complexe et la présence de trafic légitime en plus des attaques.

5.2 Évaluation de la qualité des règles dans un SI classique

Cette section présente l'évaluation de la méthode de génération des règles de corrélation sur un système d'information représentatif de l'infrastructure d'une entreprise de petite taille. Le but est de pouvoir évaluer si les règles générées permettent de détecter des attaques tout en évitant la génération de faux positifs.

5.2.1 Présentation de l'architecture

L'étude qui suit se base sur une architecture existante conçue spécialement pour la réalisation d'exercices d'attaque-défense. Les journaux générés pendant l'un de ces exercices ont pu être réutilisés pour ces travaux de thèse. Un aperçu des principaux éléments topologiques de ce système est donné par la figure 5.1.

Le système est divisé en plusieurs zones. La DMZ contient différentes machines accessibles de l'extérieur (deux serveurs web, un serveur VPN, un serveur DNS et un serveur de mail) ainsi qu'un serveur mandataire utilisé par les machines internes pour accéder à internet. Un serveur FTP est également présent dans cette zone. La zone serveurs contient un serveur mail, un serveur web (service intranet), un serveur d'annuaires Active Directory (ad) et un serveur hébergeant des données sensibles. Une zone est spécifiquement dédiée aux machines des clients. Certains clients externes peuvent accéder aux ressources situées dans la zone serveurs en passant par le VPN situé dans la DMZ. Des pare-feux sont positionnés entre l'extérieur et la DMZ ainsi qu'entre la DMZ et les zones serveurs et clients. Chaque machine située dans la zone serveurs et dans la DMZ dispose d'une interface réseau dédiée à l'administration.

2. http ://old.honeynet.org/scans/scan17/

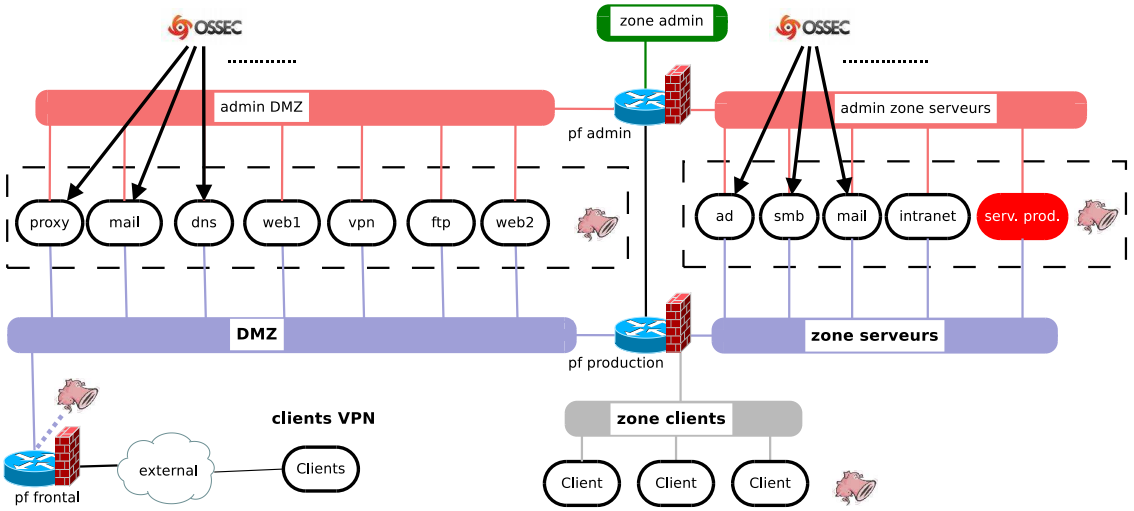


FIGURE 5.1 – Schéma de l'architecture de référence.

Dans ce système, l'objectif des attaquants est de prendre possession d'un fichier placé sur le serveur de production (serveur représenté à droite et mis en valeur sur la figure 5.1). Cette machine n'est pas accessible directement depuis l'extérieur. Par conséquent, il est possible que les attaquants réalisent un scénario d'attaque qui met en œuvre les étapes suivantes :

— attaque et compromission d'un des serveurs web de la DMZ accessible depuis l'extérieur ;
— recherche d'obtention des droits administrateur sur l'une des machines de la DMZ ;
— compromission d'un serveur de la zone serveurs depuis la DMZ ;
— accès au serveur de production contenant les informations à récupérer.

Chaque étape peut être précédée d'une phase de reconnaissance.

A Supervision

Des agents HIDS Ossec sont déployés sur chacune des machines du système. La disponibilité des services qui tournent sur les différents serveurs sont surveillés par l'intermédiaire de Nagios. Quatre NIDS Snorts sont déployés respectivement dans la DMZ, dans la zone serveurs, dans la zone clients ainsi qu'en frontal devant le pare-feu séparant la DMZ de l'extérieur. Pour limiter les faux positifs, les IDS Snort sont configurés uniquement avec les règles pertinentes pour chaque zone surveillée. Aux règles de base sont ajoutées (1) des règles spécifiques permettant de détecter l'exploitation de vulnérabilités connues présentes sur certains services (2) des règles traduisant la politique de sécurité du système. Ainsi, l'utilisation de certains protocoles est interdite dans certaines zones et certaines connexions entre machines sont interdites (par exemple, les deux serveurs Webs de la DMZ n'ont aucune raison de communiquer entre eux). Deux serveurs Ossec sont utilisés pour centraliser les données. Pour réaliser notre évaluation nous disposons donc :

— des données issues des IDS Snorts
— des données issues des serveurs Ossec qui comprennent :
  — les alertes issues des agents Ossec
  — des événements issus d'autres observateurs (Nagios, événements de login/logout du serveur Active Directory)

Nous disposons ainsi de ces données centralisées par les serveurs Ossec.

5.2.2 Scénarios d'attaque

A Construction de scénarios d'attaque probables

La méthodologie d'écriture des scénarios d'attaque consiste à définir dans un premier temps un ensemble de scénarios d'attaque probables sur le système. Ces attaques peuvent être génériques

(par exemple la compromission d'un client et mise en place d'un botnet) ou adaptées aux risques plus spécifiques en fonction des caractéristiques du système en question.

Identifiant	Structure
1	SAND(recExtDmz, expDMZ, OR(recDmzDmz, recDmzSer))
2	SAND(scanExtDmz, expDmzMail, expSerMail)
3	SAND(recExtDmz, expExtDmz, OR(SAND(recDmzExt, expDmzExt), spamDmzExt))
4	SAND(email, OR(scanClientClient, expClientSer, expClientClient))
5	SAND(recExtDmz, expExpDmz, OR(scanDmzDmz, expDmzDmz, connect))
6	SAND(scanExtDmz, expExtDmz, expDmzDmz, OR(usurpVPN, usurpClient), expSer)
7	SAND(scanSer, OR(servAttack, fbSer))
8	SAND(OR(leakExtDmz, webExtDmz), scanWebDmz, connectWebDmz)
9	SAND(expDmz, AND(backdoor, connect), privEsc, localDos)
10	SAND(OR(FTPinject, FTPfb), FTPbackdoor, FTPprivEsc, scan)
11	SAND(leakDmzExt, AND(backdoorDmz, toolsDownload))
12	SAND(expDmz, backdoorDmz, ScanDmzDmz, fbDmzDmz)
13	SAND(expDmz, expDmzDmz, clientSpoof, expSer)

TABLE 5.1 – Présentation des scénarios d'attaque.

Liste des arbres d'attaque Le tableau 5.1 liste les squelettes des scénarios d'attaque spécifiés. La description des actions élémentaires composant chaque scénario est décrite ci-dessous.

1 L'attaquant réalise une reconnaissance puis attaque une machine accessible depuis l'extérieur et s'en sert comme pivot pour identifier d'autres cibles potentielles. Cette identification peut être réalisée sur d'autres machines de la DMZ ou vers des machines de la zone serveurs. Les quatre actions mises en jeu sont :

recExtDmz : reconnaissance depuis extérieur vers une machine de la DMZ ;

expDMZ : attaque sur un service vulnérable ;

recDmzDmz : scan depuis la machine compromise vers une machine de la DMZ ;

recDmzSer : scan depuis la machine compromise vers une machine de la zone serveurs.

2 L'attaquant cible un serveur Mail accessible depuis l'extérieur, le compromet et tente d'attaquer le serveur mail interne. Ce scénario est construit à partir de la séquence des trois actions qui suivent :

scanExtDmz : scan depuis l'extérieur ;

expDmzMail : attaque d'une machine de la DMZ sur le port 25 depuis l'extérieur ;

expSerMail : attaque du service mail de la zone serveurs depuis la machine précédemment attaquée de la DMZ.

3 L'attaquant cherche à prendre le contrôle d'une machine pour ensuite réaliser des attaques vers l'extérieur à partir de la machine compromise. Après une action de reconnaissance suivie d'une action permettant de compromettre une machine, l'attaquant peut utiliser

cette machine compromise pour rebondir sur une autre machine de la même zone, ou bien se servir de la machine compromise pour envoyer du SPAM vers l'extérieur. Les actions mises en jeu sont alors :

recExtDmz : scan depuis l'extérieur vers une machine de la DMZ ;

expExtDmz : exploitation d'un vulnérabilité sur la machine scannée ;

recDmzExt : reconnaissance depuis la machine compromise de la DMZ vers l'extérieur ;

expDmzExt : exploitation d'un service réseau d'une machine externe à partir de la machine de la DMZ compromise ;

spamDmzExt : envoi de SPAM depuis la machine compromise vers l'extérieur.

4 Ce scénario décrit une attaque des postes utilisateurs par l'intermédiaire d'une pièce jointe piégée. Le poste client infecté peut alors servir à réaliser une reconnaissance au sein de la zone clients ou bien une exploitation de vulnérabilité sur un autre poste client ou sur une machine présente dans la zone serveurs. Les actions utilisées pour décrire ce scénario sont les suivantes :

email : envoi d'un email avec pièce jointe piégée à un utilisateur ;

scanClientClient : le poste de cet utilisateur effectue un scan du sous-réseau client ;

expClientSer : exploitation d'une vulnérabilité sur des serveurs internes ;

expClientClient : réalisation d'une attaque sur d'autre postes utilisateurs.

5 Dans ce scénario, l'attaquant réalise une action de reconnaissance sur une machine de la DMZ avant d'y exploiter une vulnérabilité. Cette première machine compromise sert alors de relais pour réaliser une reconnaissance de la DMZ, une exploitation d'une autre machine de la DMZ ou bien d'une connexion directe vers une machine (par exemple en SSH). Les actions mises en jeu sont alors :

recExtDmz : reconnaissance depuis l'extérieur ;

expExtDmz : exploitation d'un service réseau sur une machine de la DMZ ;

scanDmzDmz : scan de la DMZ depuis la machine compromise ;

expDmzDmz : exploitation d'une vulnérabilité d'un service réseau d'une machine de la DMZ à partir de la machine compromise ;

connect : tentative de connexion entre la machine compromise et une machine quelconque.

6 L'attaquant réalise une reconnaissance suivie d'une attaque permettant de prendre la main sur un serveur de la DMZ. Ce serveur compromis lui permet de réaliser un mouvement horizontal vers un autre serveur de la DMZ. Il utilise ensuite l'identité d'une autre machine pour tenter d'accéder à la zone serveurs dans le but d'y attaquer une nouvelle machine. Les actions impliquées dans ce scénario sont :

scanExtDmz : scan d'une première machine dans la DMZ ;

expExtDmz : exploitation d'une première machine dans la DMZ ;

expDmzDmz : attaque d'une seconde machine à partir de la première ;

usurpVPN : usurpation d'identité du serveur VPN ;

usurpClient : usurpation de l'identité d'un client utilisant le VPN ;

expSer : exploitation d'un serveur dans la zone serveurs à partir de l'adresse usurpée.

7 Ce scénario décrit une attaque de divers services de la zone serveurs. Un scan de ports sur une machine est suivi par une attaque d'un de ses services vulnérables ou bien d'une attaque par force brute. Les trois actions utilisées sont :

scanSer : scan de ports sur une machine de la zone serveurs ;

servAttack : attaque d'un service vulnérable ;

fbSer : Réalisation d'une attaque par force brute pour se connecter à un service.

8 L'attaquant récupère des informations sensibles qui sont stockées sur un serveur Web ou compromet directement ce serveur en ciblant le service Web. Des actions sont ensuite menées depuis la machine compromise pour scanner d'autres nœuds de la DMZ ou tenter d'établir des connexions sur d'autres machines. Les actions suivantes sont utilisées :

leakExtDmz : accès à des informations sensibles sur un serveur accessible depuis l'extérieur ;

webExtDmz : attaque d'un service Web ;

scanWebDmz : scan depuis la machine compromise vers des machines de la DMZ ;

connectWebDmz : tentative de connexion à une machine de la DMZ depuis la machine compromise.

9 Dans ce scénario, l'attaquant prend le contrôle complet d'une machine en exploitant une vulnérabilité d'un service réseau, puis il dépose une porte dérobée sur cette machine et tente une connexion vers une autre machine. Il continue son attaque en y réalisant une élévation de privilèges et en effectuant un déni de service sur un service local.

expDmz : exploitation d'une vulnérabilité sur un service d'une machine de la DMZ ;

backdoor : mise en place d'une porte dérobée sur la machine compromise ;

connect : tentative de connexion vers une autre machine ;

privEsc : élévation de privilèges sur la machine compromise ;

localDos : déni de service sur un service local.

10 Ce scénario se concentre sur l'attaque d'un serveur proposant un service FTP. L'attaquant utilise des attaques par injection ou tente une attaque par force brute pour obtenir un premier niveau de compromission du serveur. L'attaque se poursuit par la mise en place d'une porte dérobée suivie d'une élévation de privilège et par un scan du sous-réseau pour identifier de nouvelles cibles.

FTPinject : attaque par injection sur le serveur FTP ;

FTPfb : attaque par force brute sur ce même serveur depuis une machine de la DMZ ;

FTPbackdoor : mise en place d'une porte dérobée ;

FTPprivEsc : élévation de privilège ;

scan : scan depuis cette machine vers une machine quelconque.

11 L'attaquant accède à des informations de configuration sensibles sur une machine. Ces informations lui permettent de gagner un accès sur cette machine et d'y mettre en place une porte dérobée. L'attaquant cherche également à récupérer des outils lui permettant de continuer son exploitation. Les actions impliquées sont les suivantes :

leakDmzExt : Récupération d'informations sensibles sur une machine de la DMZ ;

backdoorDmz : mise en place d'une porte dérobée ;

toolsDownload : téléchargement d'outils sur la machine compromise.

12 L'attaquant prend le contrôle d'une machine de la DMZ en y exploitant une vulnérabilité. Après avoir mis en place une porte dérobée, il réalise une reconnaissance à partir de cette dernière machine avant de lancer une attaque par force brute sur une autre machine accessible de la DMZ.

expDmz : exploitation d'une vulnérabilité sur un serveur de la DMZ ;

backdoorDmz : mise en place d'une porte dérobée ;

scanDmzDmz : scan de la DMZ ;

fbDmzDmz : attaque par force brute de la machine compromise vers un service d'une machine de la DMZ.

13 Ce scénario représente la séquence des actions permettant d'accéder à un serveur de la zone serveurs depuis l'extérieur. L'attaque commence par la compromission d'un premier serveur situé dans la DMZ, puis d'un second serveur de la DMZ est exploité à partir de la première machine attaquée. Une usurpation d'identité est utilisée pour pouvoir accéder à la zone serveurs depuis cette seconde machine compromise. Les identifiants des différentes actions sont alors :

expDmz : exploitation d'une vulnérabilité sur un premier serveur ;

expDmzDmz : exploitation d'une vulnérabilité à partir du premier serveur vers un second ;

clientSpoof : une usurpation d'identité réalisée depuis le second serveur compromis en utilisant une adresse d'un client VPN ;

expSer : une attaque vers un serveur de la zone serveurs en utilisant l'adresse usurpée.

Des arbres d'actions sont construits pour modéliser chacun de ces 13 scénarios. Nous disposons ainsi d'un ensemble d'arbres d'actions qui représentent des scénarios d'attaques de longueurs et de complexité variées. Ces arbres d'actions seront ensuite transformés automatiquement en règle de corrélation par notre processus de génération.

Il reste à préparer la base de connaissances pour représenter l'environnement d'exécution et à s'assurer que les informations présentes dans les journaux soient suffisamment normalisées pour permettre de réaliser notre étape de corrélation.

5.2.3 Travaux préliminaires

Cette évaluation a été réalisée à partir d'un fichier d'export d'un journal rassemblant tous les messages obtenus durant la période de l'exercice attaque/défense. Cependant, les événements contenus dans les journaux ne sont pas normalisées. Les travaux préliminaires à l'évaluation consistent d'une part à effectuer une normalisation pour faciliter le traitement du journal et d'autre part à modéliser le système.

A Normalisation

Normalisation syntaxique Les journaux intègrent des événements qui sont issus de deux types sources : les serveurs Ossec, qui centralisent les messages issus des agents, et des NIDS Snort. Un format unique est utilisé pour représenter ces deux types d'alertes et réunit les champs essentiels au travail de corrélation. La sélection permet de représenter le champ date, l'identifiant du message, la source, la cible et éventuellement le port cible, le nom du logiciel et l'identifiant de l'utilisateur. Toutes ces informations sont toujours présentes dans les messages mais une étape de normalisation sémantique est nécessaire pour les rendre utilisables.

Normalisation sémantique Une tâche essentielle consiste à associer un identifiant unique à chaque nœud du système. Dans les différents messages, une machine, par exemple le serveur mail, peut être identifié par son adresse de production (*10.0.0.2*), par son adresse d'administration (*10.1.0.2*) ou par son adresse publique avant la translation d'adresse statique (*10.2.0.3*). L'une de ces trois adresses peut être choisie comme référence arbitraire pour identifier le nœud sans ambiguïté.

Une seconde tâche consiste à identifier la source et la cible concernées par chaque message. Ces informations sont présentes dans les messages mais pas systématiquement présentées de manière homogène. Par exemple, dans le format des messages Ossec, trois champs permettent d'identifier un nœud potentiel : *location*, *srcip* et *dstip*. Ces trois champs ne sont pas toujours présents simultanément. En fonction du type de message, *location* peut correspondre à l'entité à l'origine de la génération du message, à la source ou à la cible d'une action. Les champs *srcip* et *dstip* correspondent respectivement toujours à l'adresse source et à l'adresse destination.

B Base de connaissances

Topologie et Cartographie Ces éléments reprennent le schéma 5.1 de l'architecture. L'accès par VPN est modélisé de la manière suivante : les clients VPN appartiennent à un sous-réseau spécifique à partir duquel il est directement possible d'accéder au sous-réseau DMZ par l'intermédiaire du serveur VPN qui fait office de routeur. Avec cette représentation, le NIDS Snort placé au niveau du pare-feu frontal n'a pas de visibilité sur le contenu des flux joignant les clients VPN et le serveur VPN, ce qui est le cas en pratique étant donné que les flux sont chiffrés.

Observateurs La modélisation des NIDS Snort se fait sans ambiguïté. Ils sont accessibles depuis leurs réseaux d'administration respectifs et disposent d'une visibilité topologique englobant le réseau de production associé.

Plusieurs choix de modélisation des observateurs hôtes sont possibles. Il est possible de spécifier de manière exhaustive chaque source de donnée individuelle comme observateur (les services réseaux générant des journaux, les services d'audit système, les agents Ossec locaux) ou bien s'inspirer de l'architecture réelle dans lesquels les deux serveurs Ossec collectent et centralisent les données issues des différents agents. Le serveur Ossec responsable de la DMZ dispose ainsi d'une visibilité topologique sur l'ensemble des nœuds présents dans la DMZ. De la même manière, le serveur Ossec localisé dans la zone serveurs dispose d'une visibilité topologique sur l'ensemble des machines de la zone serveurs. La visibilité opérationnelle de chaque serveur correspond alors à l'union des visibilités des sources de données individuelles réparties sur les nœuds surveillés.

Visibilité opérationnelle Pour disposer d'une base de connaissances complète, les identifiants des messages doivent être associés à des noms d'actions. Cette opération n'est pas toujours facile pour plusieurs raisons. L'association d'une règle à une action demande une part d'interprétation et d'équilibre entre le risque de générer des faux positifs et de manquer une étape d'attaque. Par exemple, un message renvoyant des erreurs PHP peut être un effet de bord provoqué par une attaque d'injection de code ou bien l'effet d'une erreur sans conséquence de l'application web. C'est à la charge de l'administrateur de savoir quand une association peut être bénéfique ou non. Certaines catégories d'attaques ne sont pas référencées dans la taxinomie de base CAPEC et il est donc important de savoir quand il est nécessaire de l'étendre. Par exemple, certaines actions sont des violations d'une politique de sécurité qui dépend du système. Il est possible de toutes les rassembler comme action *violation de la politique de sécurité*, mais cela rassemble des actions hétérogènes qui comprennent à la fois des actions impliquant des flux réseaux prohibés et des actions locales sur des fichiers protégés. Une solution consiste à différencier ces catégories en introduisant autant de classes intermédiaires que de types de violations possibles.

Les journaux générés contiennent 142 identifiants de règles uniques (92 pour Ossec et 50 pour Snort). Les tableaux 5.2 et 5.3 explicitent les associations les plus importantes réalisées entre les identifiants des règles et les actions détectées par ces règles. Ces tableaux mettent en avant les choix d'interprétations qui ont été réalisés pour certaines associations. Par exemple, nous choisissons d'associer la règle intitulée *HTTP unknown method* à la classe d'attaque *injection* car cette règle renseigne d'une possible tentative d'injection de données arbitraires via le protocole HTTP. Dans un autre contexte, cette association peut ne plus être pertinente.

5.2.4 Résultats

Les résultats de la détection présentés dans le tableau 5.4 sont obtenus à partir des règles de corrélation issues des scénarios d'attaque spécifiés et présentés dans le tableau 5.1. Les chemins d'attaques explorés lors de l'exercice attaque défense se résument à :

- des actions de reconnaissance, d'attaques et d'accès à des informations sensibles sur le serveur Web1 ;
- un accès à la console d'administration du serveur Web2 puis la mise en place d'un Web-shell ;
- la reconnaissance de la DMZ depuis le serveur Web2 suivie de l'exploitation d'une vulnérabilité sur le serveur FTP ;
- une élévation de privilèges sur le serveur FTP et une désactivation de son agent Ossec local ;
- une reconnaissance à partir du serveur FTP et une écoute du trafic réseau ;
- une usurpation d'identité d'un client VPN permettant d'accéder à la zone serveurs ;
- un ensemble d'attaques ciblant le serveur Web de la zone serveurs ;
- un accès au serveur de données sensibles en utilisant les identifiants et mots de passes récupérés au préalable.

Certains scénarios spécifiés ne se sont effectivement pas produits et leur trace n'est donc pas présente dans les journaux. C'est par exemple le cas de la compromission d'un poste client pour réaliser des attaques vers l'extérieur (scénarios 3 et 4). Le scénario 2 ne lève aucune alerte car l'attaque a été abandonnée au milieu du scénario.

Intitulé de la règle	fait Prolog
Sudo to root	detects('101007',privilege_escalation)
Access database passwd	detects('102007', forbidden_file_read)
Access php file	detects('102022',data_excavation_attacks)
Access attempt to forbidden file/directory	detects('30105', forbidden_file_read)
Common web attack	detects('31104', directory_traversal)
Multiple 400 error from same IP	detects('31151', functionality_misuse)
PHP internal error	detects('31421', code_injection)
Multiple SSHD auth failure	detects('5720', brute_force)
Attempt to login using a non-existent user	detects('5710', brute_force)
Access attempt to database passwd	detects('102008', forbidden_file_read)
CVE-2013-2094	detects('102015', privilege_escalation)
PHP fatal error	detects('31420', injection)
Ossec agent disconnected	detects('101002', ressource_depletion)
CVE-2004-2687	detects('102019', code_injection)
Zone transfert denied	detects('12145', dns_zone_transfert)
Integrity checksum changed	detects('551', software_integrity_attacks)
Web attack success	detects('31106', injection)
Package installed	detects('2932', install)
Changed network interface	detects('7204', identity_spoofing)
Possible arpspoofing attempt	detects('7209', man_in_the_middle)

TABLE 5.2 – Association des identifiants des règles des messages aux classes d'attaques (Ossec).

Les scénarios ayant mené à la génération d'alertes permettent de mettre en évidence les étapes de compromission de différents nœuds du système. Par exemple, le scénario d'attaque 9 a mené à la construction d'une règle de corrélation permettant de détecter l'enchaînement d'une exploitation de vulnérabilité d'un service réseau sur le serveur FTP suivi d'une modification de la liste des ports en écoute sur le serveur puis de l'exploitation d'une vulnérabilité locale dans la version du noyau permettant d'obtenir les privilèges administrateur. La dernière action correspond à l'arrêt de l'observateur hôte (agent HIDS Ossec) sur cette machine par les attaquants.

Cette première évaluation nous permet de conclure que les règles générées permettent de détecter les scénarios d'attaque lorsque ces derniers sont réalisés. De plus, nous ne constatons pas de levées d'alertes intempestives (liée à une mauvaise reconnaissance d'un scénario) malgré la présence de faux positifs levés par les IDS. Nous constatons également que les scénarios non réalisés ne mènent pas à une levée d'alerte.

En conclusion, cette expérimentation permet de constater l'applicabilité de la méthode dans le cas d'un système maîtrisé.

5.2.5 Discussion

Les résultats de cette expérimentation peuvent être critiqués sur plusieurs points, en particulier en ce qui concerne les étapes manuelles à réaliser ainsi que les conséquences de la maîtrise du système d'information mis en jeu.

L'approche est applicable dans ce cas mais demande tout de même un travail important pour

Intitulé de la règle	fait prolog
Sensitive data transmitted	detects('1:100030:1',data_theft)
/etc/passwd via HTTP	detects('1:100034:8',data_theft)
/etc/passwd via HTTP	detects('1:110012:8',data_theft)
Access /page/pagedata.txt	detects('1:100019:0',data_theft)
Invalid FTP command	detects('125:2:1',command_injection)
UDP portsweep	detects('122:19:1',port_scanning)
HTTP unknown method	detects('119:31:1',injection)
SSH protocol on production network	detects('1:110004:1',forbidden_connection)
Multiple connexions attempt	detects('1:100001:0',brute_force)
TCP portscan	detects('122:5:1',port_scanning)
POST request with PHP code	detects('1:110008:1',code_injection)
Unauthorized traffic	detects('1:100027:0', forbidden_connection)
HTTP long Header	detects('1:110008', injection)

TABLE 5.3 – Association des identifiants des règles des messages aux classes d'attaques (Snort).

créer la base de faits et les scénarios d'attaques. En particulier, l'identification du lien entre les règles de détection et les classes d'attaques constitue une étape clef sur laquelle repose une grande partie des résultats.

La maîtrise du système a également facilité cette expérimentation. Les règles de détection mises en place au sein des IDS sont adaptées au système en définissant certaines propriétés liées à la politique de sécurité du système. Une autre partie des règles de détection des IDS est dédiée à la détection de points clefs dans l'avancement des attaquants, les vulnérabilités principales du système étant connues. Par conséquent, les IDS disposent d'une configuration adaptée au système et génèrent donc naturellement un nombre de faux positifs assez bas. Cependant, la présence de clients induit la création d'un bruit de fond ainsi que des faux positifs. Toutes ces caractéristiques facilitent grandement le travail de corrélation que nous avons réalisé.

Il est également important de noter l'impossibilité de détecter des scénarios d'attaque réalisés mais pour lesquels il n'existe pas de règle de corrélation. Cette limite est à attribuer à l'approche explicite de corrélation par scénarios sur laquelle la méthode proposée dans cette thèse repose.

5.2.6 Conclusion

Cette évaluation sur un système représentatif a consisté à appliquer la méthode de génération des règles de corrélation sur réseau d'une petite entreprise.

La capacité de description du système a ainsi pu être mise à l'épreuve pour un système de complexité moyenne. Les règles de corrélation produites permettent de détecter les scénarios lorsque ces derniers sont réalisés et ne lèvent pas d'alertes lorsque qu'ils ne le sont pas.

Par contre, cette évaluation est faite sur un système figé, par conséquent, l'un des points forts de notre approche en ce qui concerne la prise en compte de l'évolution du système ne peut pas être évaluée ici.

Cette évaluation ne couvre qu'une petite partie du processus de génération défini. C'est pourquoi une évaluation centrée non pas sur la détection mais sur la cohérence entre la base de biens et le système réel est également envisagée afin de couvrir un champ plus vaste.

Scénario d'attaque décrit	Levée d'alertes	Commentaires
1	oui	Détection d'erreur 500 et PHP suivi d'alertes TCP portscan de Snort à partir du serveur web2.
2	non	Le serveur mail a bien été attaqué, mais sans succès.
3	non	Ce scénario ne s'est pas produit.
4	non	Certaines pièces jointes ont été identifiées comme malveillantes par des antivirus, mais aucune attaque n'a suivie.
5	oui	Après la détection de la compromission du serveur Web2, une tentative d'accès au serveur de messagerie a été détectée.
6	oui	Détection de la compromission des serveurs web2 puis du serveur FTP, de l'usurpation d'un adresse de client VPN et d'une attaque ciblant le serveur intranet.
7	oui	Détection de tentatives de connexions manuelles sur l'annuaire à partir d'une adresse IP usurpée.
8	oui	Détection d'un accès à une page protégée et envoi d'une requête mal formée au serveur web2, puis scan tcp depuis le serveur web2.
9	oui	La détection porte sur l'exploitation d'une vulnérabilité distcc à distance et l'exploitation de la vulnérabilité perf_event permettant de réaliser une élévation de privilèges. La dernière action correspond à la désactivation d'un agent Ossec local.
10	oui	Détection de l'attaque dirigée contre le serveur FTP. Il s'agit d'essais successifs de nombreux mots de passe pour se connecter au service, de la détection d'une modification des sockets réseau et utilisation de perf_event et scan de port tcp sur la DMZ depuis le serveur FTP.
11	oui	Détection de la compromission du serveur web1. Il s'agit d'un accès à des informations sensibles contenues sur une page spécifique, d'une modification des sockets en écoute et de l'installation d'outils (scapy) via le gestionnaire de paquets
12	oui	détection de la compromission du serveur web2. En particulier, il s'agit de la détection de requêtes HTTP non standards, d'un scan des ports SSH de divers serveurs de la DMZ et d'une tentative de brute force SSH sur ces serveurs.
13	oui	Détection de la compromission successive des serveurs web2, FTP et intranet.

TABLE 5.4 – Scénarios d'attaque détectés.

5.3 Évaluation de la résistance aux fautes

L'évaluation précédente fait l'hypothèse que la base de connaissances représente correctement l'état du système au moment de la détection. Cependant, il est probable que cette propriété ne soit pas toujours vérifiée en cas d'évolution du système et de mise à jour partielle de la base de connaissances. Il est alors intéressant d'évaluer les conséquences de ce phénomène sur la qualité des règles de corrélation générées.

La première sous-section répertorie les fautes qui peuvent survenir au cours du processus. La seconde décrit une méthode d'évaluation de l'influence de certaines fautes sur le processus de

génération des règles de corrélation et les conséquences au niveau de la détection.

5.3.1 Classification des fautes

Dans un cas idéal, l'arbre d'action permet de produire, à l'aide de la base de connaissances du système, une règle de corrélation décrivant ce que le système permet de détecter. Au cours de la détection, les observateurs génèrent les messages attendus et ces derniers permettent de valider la reconnaissance du scénario d'attaque correspondant à la règle de corrélation.

Cependant, des perturbations peuvent altérer le fonctionnement nominal du processus. En particulier, la spécification cohérente de l'arbre d'actions initial, le fonctionnement nominal des observateurs lors de la détection ou encore la synchronisation de la base de connaissances avec le système réel constituent des hypothèses pour un fonctionnement nominal. Les deux premières hypothèses font intervenir des éléments externes à notre système de génération, alors que la troisième dépend plus fortement de l'adéquation de la base de connaissances avec le système réel.

A Sources de fautes externes au processus de génération

Le premier type de fautes est lié à une mauvaise spécification d'un arbre d'attaque. Dans ce cas, les règles générées ne permettent pas de détecter les scénarios attendus.

Le second type de faute survient dans le cas où certains observateurs ne produisent pas les messages souhaités lorsque les actions correspondantes à ce qui a été spécifié dans l'arbre d'actions sont effectuées par l'attaquant. La reconnaissance du scénario d'attaque est alors compromise. Si la cause de ce problème est une mauvaise spécification des capacités de détection des observateurs, alors il s'agit d'un problème lié à la base de connaissance, traité dans la sous-section précédente. Il peut s'agir d'une défaillance d'un observateur (ce dernier ne renvoie plus d'alertes par exemple) ou bien d'un attaquant ayant réussi à effectuer une action prévue de manière furtive. La règle de corrélation peut être mise hors de cause dans ces cas. Il faut par conséquent compter sur les capacités du corrélateur à tolérer des messages manquants pour faire progresser la reconnaissance du scénario d'attaque.

B Base de connaissances non représentative de l'environnement

Lorsque l'environnement évolue sans mise à jour de la base de connaissances, celle-ci ne représente alors plus le système courant. La base de connaissances peut représenter un sur-ensemble du système (présence d'éléments obsolètes), un sous-ensemble (certains éléments présents dans le système ne sont pas répertoriés) ou une combinaison de ces deux propriétés.

Il est important de différencier l'impact de chaque faute, sur l'arbre de corrélation d'une part, sur la détection d'autre part. Pour un scénario d'attaque donné, une faute peut ne pas avoir de conséquence sur l'arbre de corrélation. C'est le cas lorsque l'élément impacté dans la base de connaissances n'est pas requis pour la création de l'arbre de corrélation. La structure de l'arbre de corrélation est alors potentiellement modifiée par une faute, mais il est possible que cette modification n'impacte pas la qualité de la détection. La structure est par exemple impactée lorsqu'une nouvelle branche est ajoutée à un opérateur *OR* et référence des messages invalides qui ne peuvent pas être générés par des observateurs du système.

Quatre cas élémentaires principaux caractérisent les différences entre les éléments représentés par une base de connaissances B et une base de connaissances B'.

(1) $B = B'$

(2) $B \subsetneq B'$

(3) $B \supsetneq B'$

(4) $B \not\subset B' \wedge B \not\supset B' \wedge B \neq B'$

Le cas (1) représente l'identité entre les deux bases de connaissances (les mêmes éléments du système sont représentés dans le deux bases). Les cas (2) et (3) représentent respectivement une base de connaissances B' qui est un sur-ensemble strict de B (B' contient des entrées additionnelles obsolètes) et un sous-ensemble strict de B (Certaines entrées manquent dans B' par rapport à B). Le cas (4) est rencontré lorsque, partant de B, certains faits sont modifiés pour représenter des

éléments non présents dans le système. Ce cas peut se modéliser par un retrait de n faits suivi d'un ajout de n nouveaux faits qui ne sont pas déjà présents dans la base de connaissances. Par conséquent, la suite propose uniquement une analyse de deux types de fautes qui sont les entrées additionnelles obsolètes et les entrées manquantes dans la base de connaissances.

B.1 Entrées additionnelles obsolètes La présence dans la base de connaissances d'éléments non présents dans le système réel peut avoir plusieurs conséquences qui sont référencées dans le tableau 5.5. Des faux positifs sont possibles lorsque l'arbre de corrélation référence des messages non pertinents. L'occurrence de ces messages fait avancer la détection et mène à une reconnaissance erronée du scénario d'attaque. Cependant cela ne devrait pas se produire si aucun observateur n'est capable de générer de telles informations. Les faux négatifs liés par exemple à la présence de messages obsolètes ajoutés comme fils d'un opérateur *AND* ou *SAND* peuvent bloquer le processus de reconnaissance [3].

Influences sur l'arbre de corrélation	influence sur la détection
branche supplémentaire	faux positifs
feuilles supplémentaires	faux positifs ou faux négatifs

TABLE 5.5 – Influences des entrées additionnelles obsolètes sur l'arbre de corrélation et sur la détection.

La figure 5.2 présente un exemple de conséquence d'une entrée additionnelle obsolète correspondant à un ancien NIDS retiré du système mais toujours présent dans la base de connaissances. La conséquence sur l'arbre de corrélation est la présence des feuilles $N0\_1$ et $N0\_2$ qui correspondent à la spécification des messages potentiellement générés par cet observateur pour la première action du scénario d'attaque.

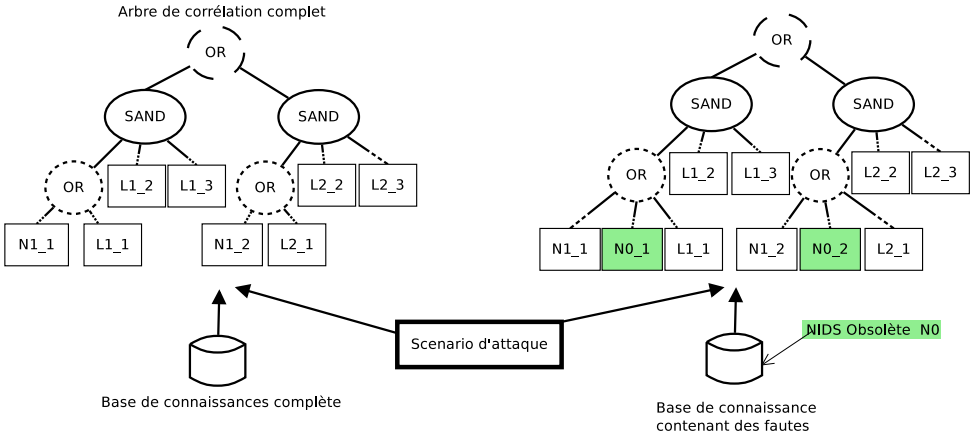


FIGURE 5.2 – Influence d'un élément additionnel obsolète sur un arbre de corrélation. L'arbre d'actions correspondant est constitué d'une séquence de trois actions dont la première est visible à la fois par un NIDS $N1$ et par un HIDS $L1$. L'opérateur $SAND$ présent initialement dans l'arbre d'action est représenté en traits pleins alors que les opérateurs OR introduits lors de la création de l'arbre de corrélation sont en pointillés.

B.2 Entrées manquantes L'absence de certains éléments présents dans le système mène à la génération de règles de corrélation incomplètes ou erronées. Le tableau 5.6 résume les effets possibles d'un tel manque sur l'arbre de corrélation et sur la détection.

3. Ce blocage dépend du mode de fonctionnement du corrélateur, en particulier si ce dernier a une tolérance lui permettant de continuer la reconnaissance d'un plan dans le cas où un message est manquant, mais le suivant est présent.

Influences sur l'arbre de corrélation	influence sur la détection
branche manquante	faux négatifs
feuilles manquantes	faux positifs ou faux négatifs

TABLE 5.6 – Influences des entrées manquantes sur l'arbre de corrélation et sur la détection.

Les faux négatifs sont liés à l'absence de certaines parties de l'arbre de corrélation par rapport à l'arbre de référence, comme dans l'exemple de la figure 5.3 pour laquelle une attaque sur le nœud manquant ne peut pas être détectée en utilisant l'arbre de corrélation. Les faux positifs sont liés à la réduction des règles de corrélation et à la suppression potentielle de certains messages indispensables pour différencier l'attaque d'un comportement normal. Ce cas intervient par exemple dans la séquence de trois actions suivantes : la connexion à une machine avec des droits non privilégiés ; l'exploitation d'une vulnérabilité permettant d'obtenir les privilèges administrateurs et la création d'un nouvel utilisateur privilégié. Lorsque le HIDS capable de détecter l'élévation de privilège n'est pas renseigné dans la base de connaissances, l'arbre de corrélation généré se réduit à une séquence de deux actions dont l'enchaînement est potentiellement légitime (connexion et création d'un nouveau compte administrateur), ce qui provoque des faux positifs dans le cas où cet enchaînement légitime survient.

Par exemple, la figure 5.3 illustre les conséquences sur l'arbre de corrélation de la suppression d'un nœud de la base de connaissances (nœud manquant). Une branche entière manque à l'arbre de corrélation résultant par rapport à l'arbre de corrélation de référence.

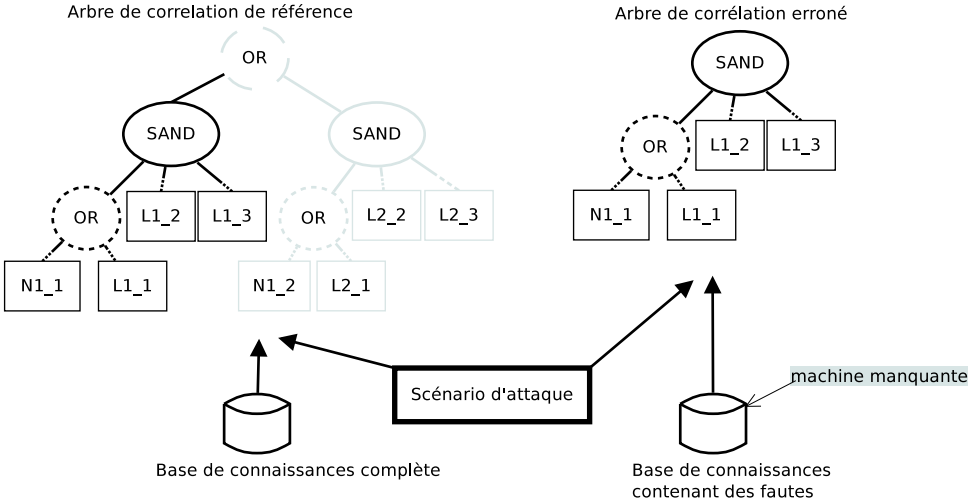


FIGURE 5.3 – Influence d'un élément manquant.

Les trois dimensions (mauvaise spécification, dysfonctionnement des observateurs, base de connaissances non représentative du système) pouvant altérer le processus de reconnaissance de scénarios d'attaque sont liées à des causes différentes. Une mauvaise spécification de l'arbre d'action initial ou un dysfonctionnement d'un observateur au moment de la détection sont des facteurs externes au processus de génération et difficilement contrôlable. Par conséquent, ils ne sont pas considérés dans l'évaluation. Par contre, l'adéquation du contenu de la base de connaissances avec le système réel est une composante essentielle de ce processus. Il est donc naturel de mettre l'accent sur l'évaluation des conséquences de l'utilisation d'une base de connaissances qui est désynchronisée de l'environnement réel, c'est-à-dire qui ne le modélise pas correctement.

5.3.2 Mise en œuvre d'un processus d'évaluation de l'influence des fautes sur la détection

Cette sous-section décrit la mise en œuvre d'une expérimentation pour évaluer l'influence individuelle de différents types de fautes sur le processus de génération de règles de corrélation. Cette évaluation est centrée sur deux types de fautes qui peuvent affecter la base de connaissances : les entrées additionnelles obsolètes et les entrées manquantes. L'évaluation consiste à injecter un faute spécifique parmi un ensemble déterminé dans une base de connaissances idéale (complète et à jour) d'un système donné. Les capacités de détection des règles de corrélation altérées et idéales sont ensuite comparées. Cette sous-section présente une analyse qualitative des fautes puis la méthode d'évaluation est présentée, suivie de scénarios d'attaque utilisés pour l'évaluation. Les résultats sont alors analysés.

A Analyse qualitative

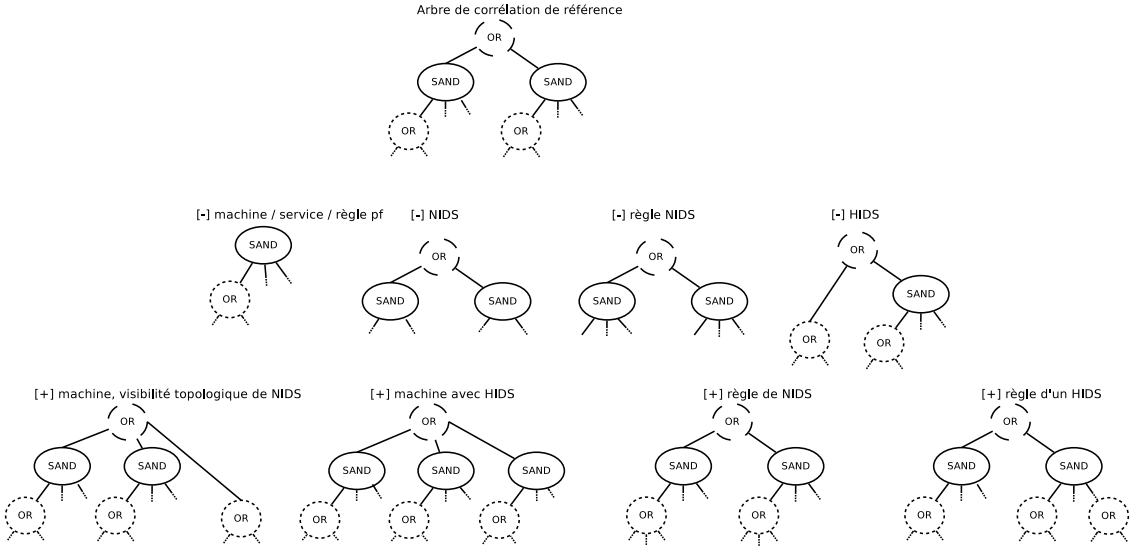


FIGURE 5.4 – Allure des arbres de corrélation en fonction de l'élément affecté par la faute. Les trois lignes correspondent de haut en bas à l'arbre de corrélation de référence, les arbres dans le cas d'éléments manquants et d'éléments additionnels obsolètes

Le but est d'estimer les effets théoriques de certaines fautes sur la détection. La figure 5.4 présente différentes allures d'arbres de corrélation générés à partir de bases de connaissances ayant subi des modifications par rapport à une base de connaissances de référence (représenté en haut de la figure 5.4). Les arbres de corrélation présentés sur la seconde ligne sont issus de la base de connaissances à laquelle certains faits ont été retirés. Cette situation représente une base de connaissances contenant des éléments manquants par rapport à la base de référence. Les quatre cas référencés correspondent respectivement à :

- le retrait d'une machine, d'un service ou bien d'une règle de pare-feu a pour conséquence d'éliminer une branche dont l'origine est la racine. Cette branche référençait les messages relatifs à la machine attaquée et qui a été affecté par la modification de la base de connaissances, que ce soit directement (retrait de la machine ou d'un service essentiel à la réalisation du scénario d'attaque) ou bien indirectement (retrait d'une règle de filtrage définissant l'accessibilité de cette machine) ;
- le retrait d'un NIDS a pour conséquence la disparition des messages qu'il peut générer de l'arbre de corrélation. Dans l'exemple de la figure 5.4, le NIDS retiré génère les messages fils des opérateurs *OR* situés à la profondeur 3 de l'arbre de référence ;
- dans le cas du retrait d'une unique règle de détection du même NIDS, seul l'un des messages générés par ce dernier disparaît ;

— contrairement au retrait d'un NIDS qui impacte potentiellement de nombreuses branches, le retrait d'un HIDS impacte uniquement les branches qui font référence au nœud sur lequel le HIDS réside. Dans l'exemple, seule la branche de gauche est affectée par le retrait du HIDS : les deux messages du NIDS sont les seuls qui subsistent.

Sur la troisième ligne, des faits ont été ajoutés (toujours par rapport à la base de référence), ce qui permet de modéliser la présence d'éléments additionnels obsolètes. Les variations observées dépendent de l'objet affecté par la faute. Les quatre cas présentés correspondent à :

— l'ajout d'une machine pouvant jouer un rôle dans le scénario d'attaque mène à la création d'une nouvelle branche. Dans ce cas, la nouvelle branche référence uniquement les deux messages du NIDS liés par un *OR* étant donné qu'aucun observateur local au nouveau nœud n'est défini pour observer les deux actions suivantes ;
— contrairement au cas précédent, l'ajout d'une machine et d'un HIDS associé mène à la création d'une branche similaire aux autres branches existantes ;
— une règle de détection supplémentaire associée au NIDS et qui est également pertinente pour la détection de la première action mène à l'ajout d'un troisième message fils des opérateurs *OR* liant les deux messages de base du NIDS ;
— contrairement au cas qui précède, l'ajout d'une règle de détection à l'un des deux HIDS affecte uniquement la branche de droite sur lequel l'HIDS concerné est positionné ;

Des effets similaires sont potentiellement causés par différentes combinaisons d'éléments manquants au sein de la base de connaissances. Par exemple, si un nœud manque par rapport à l'arbre de référence, l'arbre de corrélation résultant ne contient pas la ou les branches de l'arbre dans laquelle ce nœud intervient. Ce même effet est obtenu par le retrait de la règle de filtrage spécifiant que ce nœud est accessible depuis l'extérieur ou encore que le service exploité n'existe pas.

De plus, certaines catégories de fautes ont des impacts qui affectent l'arbre de corrélation de manière symétrique (NIDS ou règles de NIDS obsolètes ou manquants par exemple) alors que d'autres sont asymétriques (par exemple, l'absence d'un observateur local affecte uniquement les branches qui mettent en jeu un nœud surveillé par cet observateur).

B Mécanismes d'évaluation de l'influence des fautes sur la détection

La base de connaissances idéale sert de point de référence pour l'évaluation. L'injection d'une faute dans cette base de connaissances de référence mène à la création de la base de connaissances altérée KB'. Pour un scénario d'attaque donné, il est possible de générer une règle de corrélation à partir de chacune des bases de connaissances. Une fois les règles générées, elles sont transmises au corrélateur. Ce dernier doit alors traiter un ensemble de journaux (dont une partie contient des attaques correspondant au scénario d'attaque courant).

Modèle de fautes Il est nécessaire de caractériser et de justifier l'utilisation de fautes élémentaires pour altérer la base de connaissances. Potentiellement, de nombreuses fautes peuvent affecter cette base de connaissances. Ces fautes se caractérisent de la manière suivante. Par hypothèse, les fautes ne concernent que les faits qui constituent la base de connaissances et non les règles, car les règles sont statiques et indépendantes du système. Les fautes considérées se caractérisent par deux dimensions qui sont l'élément de la base de connaissances impacté par la faute (cible) et le type de faute qui détermine sa nature (élément additionnel obsolète ou élément manquant). La cible caractérise l'élément de la base de connaissances qui est affecté par la faute. D'après l'étude qualitative effectuée, il ressort qu'il est possible de se limiter à un sous-ensemble des éléments potentiellement affectés de la base de connaissances. Ces éléments sont :

— un nœud ;
— un observateur local ;
— un observateur réseau ;
— une règle d'observateur réseau.

Le type de faute est de la nature suivante :

— élément manquant : ce type de faute consiste à retirer un fait définissant l'élément dans la base de référence ;

— élément additionnel obsolète : ce type de faute consiste à ajouter un fait définissant l'élément (supposé absent dans la base de référence).

D'autres possibilités existent, par exemple un fait pourrait être erroné. Ce cas est l'association d'un retrait d'un fait et d'un ajout simultané : le fait erroné remplace le fait initial. Par conséquent, cela revient à considérer qu'il est absent de la base de connaissances, ce fait erroné correspondant à l'ajout d'un élément non présent ou redondant.

Les exemples développés au sein de la figure 5.4 illustrent le fait que certaines fautes peuvent mettre en jeu différents faits mais avoir des conséquences identiques au niveau de l'arbre de corrélation selon le scénario d'attaque choisi. Par exemple, lorsqu'une action implique l'exploitation à distance d'un service réseau sur une machine, les absences du fait caractérisant la machine et du fait caractérisant la présence du service hébergé par cette machine sont équivalentes.

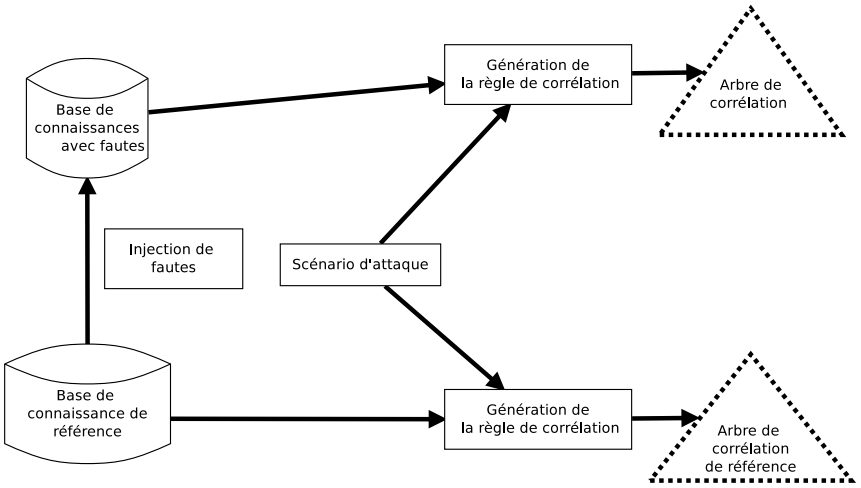


FIGURE 5.5 – Processus permettant de générer des règles de corrélation issues d'une base de connaissances contenant des fautes.

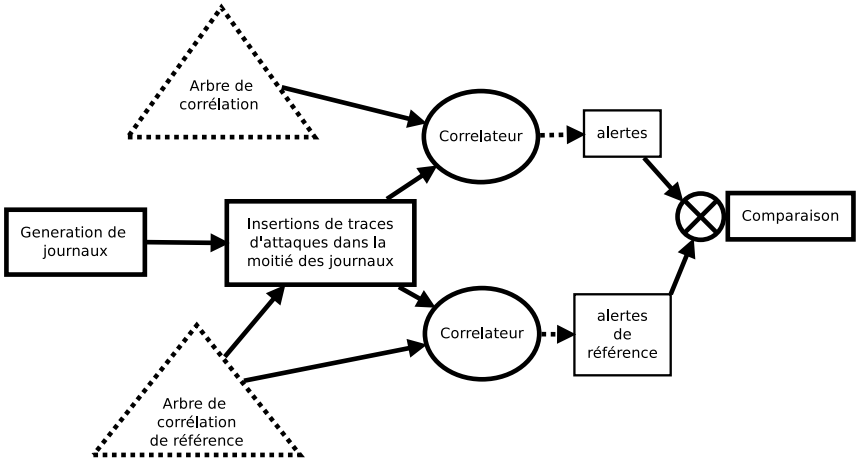


FIGURE 5.6 – Processus permettant de générer des règles de corrélation issues d'une base de connaissances contenant des fautes (2).

Les figures 5.5 et 5.6 illustrent le principe de cette évaluation. La partie basse des figures correspond au processus de génération des règles de corrélation à partir d'une base de connaissances supposée représentative du système surveillé. Une règle de corrélation est générée à partir d'un arbre d'actions. Pour tester cette règle de corrélation avec le corrélateur, des journaux sont générés. Une transaction correspondant à une trace d'attaque est introduite de manière contrôlée dans une partie des journaux. Une trace d'attaque est définie comme étant un ensemble ordonné de messages générés

lors de la réalisation d'un scénario d'attaque. Nous disposons ainsi de journaux ne contenant pas de traces d'attaques et de journaux contenant des traces d'attaques.

B.1 Faux positifs et faux négatifs Pour chaque fichier de journalisation contenant une attaque, l'absence de levée d'alertes est reportée comme un faux négatif. Pour chaque fichier de journalisation ne contenant pas d'attaque, la levée d'une ou plusieurs alarmes est reportée comme un faux positif.

B.2 Génération des journaux Les journaux utilisés pour l'évaluation sont générés de manière semi-aléatoire de telle sorte que chaque événement d'un journal correspond à un ensemble de paramètres choisis au hasard parmi une liste de valeurs possibles. Pour chaque scénario d'attaque, 200 fichiers de journalisation sont générés. Chaque fichier de journalisation contient 1000 entrées (qui correspondent à une ligne). Initialement, chaque journal est testé de manière à vérifier que son utilisation au sein du corrélateur ne mène pas à la levée d'alertes avec l'utilisation de la règle de corrélation correspondant à l'arbre de corrélation de référence. Une trace de l'attaque est ensuite insérée dans la moitié des journaux.

Extraction d'une trace d'attaque Les traces d'attaques sont incluses dans les journaux de la manière suivante. Une séquence de messages correspondant à un chemin d'attaque possible est extraite de l'arbre de corrélation. Pour obtenir la liste des messages permettant de construire une trace d'attaque, un parcours de l'arbre F est réalisé de la manière qui suit. L'arbre $F \in Tree$ est modélisé de manière récursive par $F = leaf|\{Op, [F1, \ldots, Fn]\}$, où $leaf$ est une feuille, $Op \in \{AND, SAND, OR\}$ est un opérateur et $F_i \in Tree$ sont des sous-arbres.

(1) $Select(\{OR_{racine}, [F1, \ldots, Fn]\}) = Select(F_i),\ i = rand(1, n)$

(2) $Select(\{OR, [F1, \ldots, Fn]\}) = Concat(Select(Fi_1, \ldots, Fi_k)),\ k = rand(1, n),\ i_j \in \{1 \ldots n\}$

(3) $Select(\{Op, [F1, \ldots, Fn]\}) = Concat(Select(F1), \ldots, Select(Fn)),\ Op \in \{SAND, AND\}$

Select correspond à la fonction de sélection d'un chemin d'attaque. Elle associe à un arbre de corrélation une liste de messages correspondant à une trace d'attaque possible.

Si la racine est un opérateur *OR*, une branche aléatoire est sélectionnée pour la suite de l'algorithme (cas (1)). Ce choix correspond à un chemin d'attaque possible.

Chaque fois qu'un opérateur *OR* est rencontré (excepté la racine de l'arbre), un nombre aléatoire de sous branches sont sélectionnées pour la suite du parcours et les listes de messages découlant de l'application de la fonction de sélection sont concaténées. L'origine du *OR* constitue l'élément discriminant pour effectuer les traitements adaptés. Trois cas sont possibles :

- l'opérateur *OR* provient directement de l'arbre d'action initial. Cela correspond à un ensemble d'options réalisables par l'attaquant. La sélection réalisée par l'algorithme permet alors de modéliser un attaquant qui teste un sous-ensemble de ces options ;
- si l'opérateur est issu de l'étape d'identification des observateurs et traduit la présence de plusieurs observateurs, alors la sélection revient à considérer que tous les observateurs potentiels ne génèrent pas nécessairement de messages pour une action donnée. Ceci correspond par exemple à une tentative d'évasion de la part de l'attaquant ;
- si l'opérateur est également issu de l'étape d'identification des observateurs mais est lié à plusieurs messages possibles pour une action et un observateur donné, alors on considère que les messages non sélectionnés correspondent à des critères de déclenchement d'une variante de l'action menée.

Insertion de la trace d'attaque dans les journaux L'algorithme précédent permet d'extraire une liste de messages correspondant à une trace d'attaque. Il reste à insérer cette trace dans un fichier de journalisation. Chaque message de la liste peut être traduit directement en une ligne du journal. Il reste alors à déterminer la manière dont ces dernières seront insérées parmi les entrées existantes. La propriété à respecter est le respect de l'ordre d'insertion des entrées correspondant à la liste des messages générés. Pour éviter d'insérer l'ensemble des lignes en un bloc unique qui correspondrait à la réalisation du scénario d'attaque sans aucune interruption d'événement externe, ces lignes sont réparties au sein du fichier de journalisation.

Les mécanismes généraux utilisés pour l'évaluation ont été présentés dans cette sous-section. La sous-section suivante présente et justifie les choix des scénarios d'attaque qui sont utilisés pour l'évaluation.

C Scénarios de test

C.1 Topologie et cartographie de référence Cette évaluation se déroule sur le système représenté à la figure 5.7. Il est composé d'un sous-réseau qui fait office de DMZ contenant deux machines.

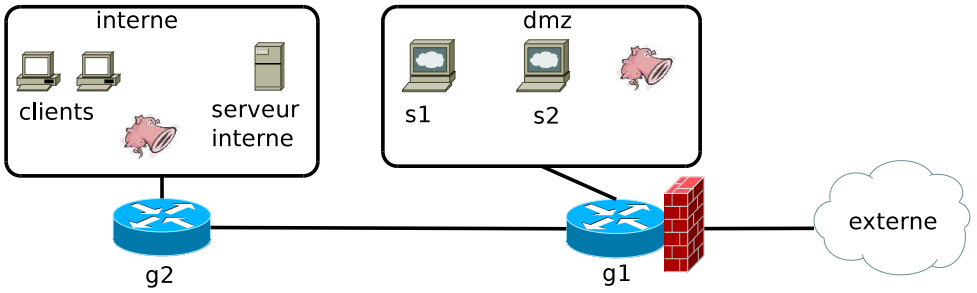


FIGURE 5.7 – Schéma de l'architecture de référence.

C.2 Scénarios de test Il est difficile de tester l'intégralité des scénarios d'attaque imaginables. Par conséquent, l'étude se limite à un sous-ensemble de scénario d'attaques choisis pour représenter des motifs représentatifs et variés. Une première étape consiste donc à caractériser cette variété.

Un arbre d'actions est une structure dont les nœuds sont des opérateurs (*OR*, *SAND* et *AND*) et les feuilles les actions. Les variations peuvent venir de la description spécifiée dans chaque action ou de l'enchaînement et le nombre des différents opérateurs.

Ce que l'on propose est de fournir un ensemble de scénarios qui jouent sur trois composantes :

(1) les types d'opérateurs utilisés ;

(2) le nombre d'actions (feuilles) de l'arbre d'action ;

(3) la localité des actions (action locale à un nœud ou impliquant une source et une cible distincte).

Ainsi, la première composante (1) permet de tester l'influence des différents types d'opérateurs. Par exemple, la suppression d'une action dont l'opérateur parent est un *OR* devrait avoir des conséquences moindres que celle dont le parent est un *SAND* ou un *AND*. La seconde composante (2) permet de tester l'influence de la taille (et indirectement de la complexité) de l'arbre d'actions. Intuitivement, un arbre de taille faible doit être plus sensible a des suppressions ou des ajouts qu'un arbre de grande taille. La troisième composante (3) permet de mesurer l'influence de la visibilité d'une action et ainsi de différencier l'impact des actions visibles localement ou au niveau de l'activité réseau.

L'évaluation est effectuée en utilisant 13 arbres d'attaque différents. La figure 5.8 en montre une partie. Ces scénarios sont construits à partir de six actions élémentaires identifiées A à F sur la figure. Chaque action est observable localement ou au niveau du réseau. Chaque scénario combine une partie de ces actions avec différents opérateurs.

Action	visibilité par un observateur local	visibilité par un observateur réseau
A	oui	oui
B	non	oui, avec deux messages différents
C, D	non	oui
E, F	oui	non

TABLE 5.7 – Visibilité des différentes actions.

Les propriétés de ces scénarios sont présentées dans le tableau 5.8

Identifiant	opérateur(s)	actions	Commentaires
S0	SAND	A,B	séquence de deux actions
S1	SAND	C,A,B	séquence de trois actions
S2	SAND	C, D, A, B	séquence de quatre actions
S3	AND	A, B	S0 en remplaçant SAND par AND
S4	AND	C, A, B	S1 en remplaçant SAND par AND
S5	AND	C, D, A, B	S3 en remplaçant SAND par AND
S6	SAND	B, E	séquence de deux actions
S7	SAND	B, D, E	séquence de trois actions
S8	SAND	B, F, D, E	séquence de quatre actions
S9	SAND	A,B E	SAND entre S0 et S6
S10	SAND, AND	A,B	SAND entre S0 et S3
S11	OR, SAND	A, B, E	OR entre S0 et S6
S12	OR, SAND, AND	A, B	OR entre S0 et S3

TABLE 5.8 – Propriétés des arbres d'actions testés.

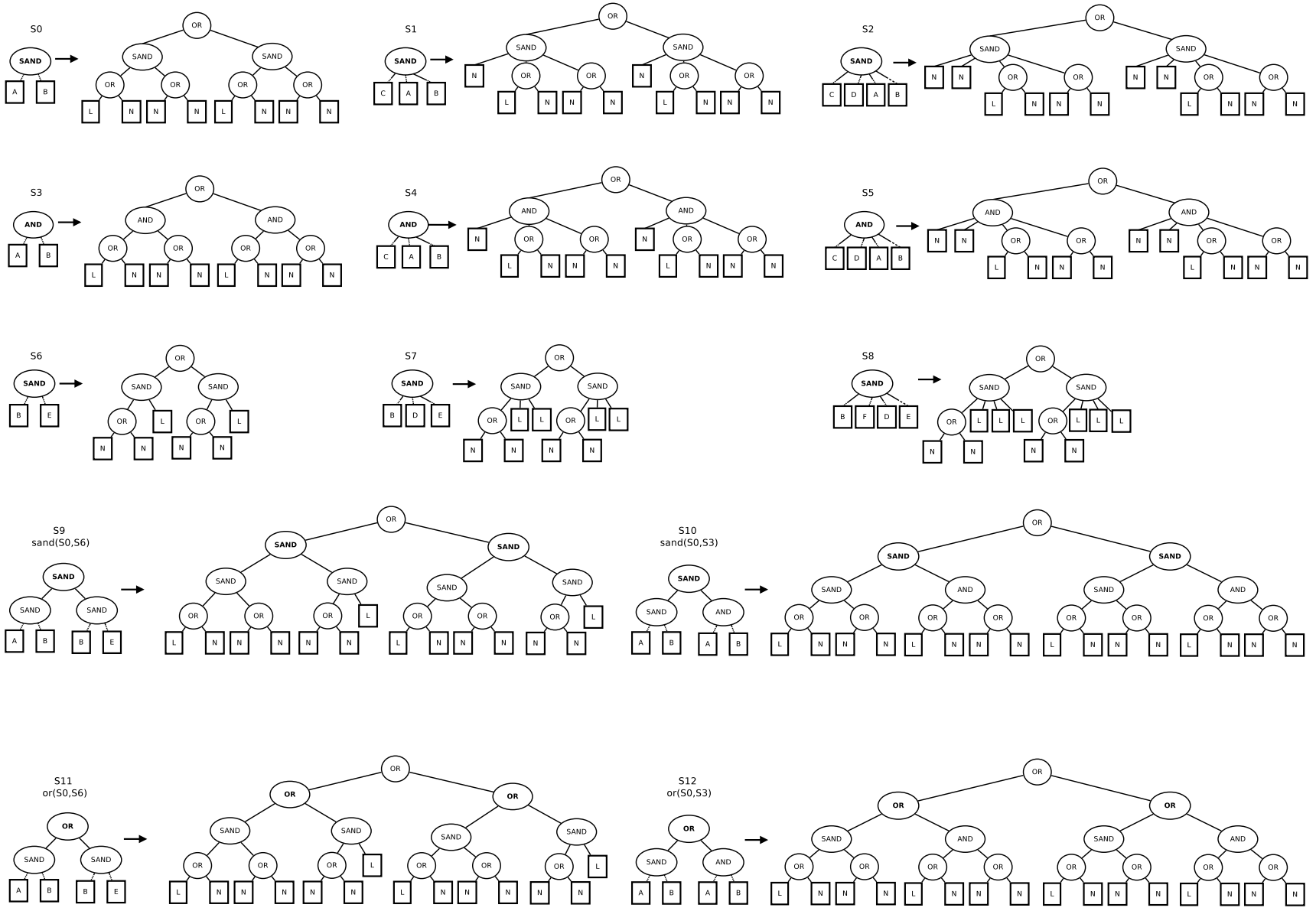


FIGURE 5.8 – Allure des arbres d'actions et des arbres de corrélation correspondants. A, B, C, D, E et F correspondent à des actions différentes. Les feuilles L et N des arbres de corrélation correspondent respectivement à des messages générés par un observateur local et un observateur réseau.

5.3.3 Résultats

Les résultats des mesures de l'influence des différentes fautes sur la qualité de la détection sont présentés dans les tableaux 5.10 et 5.11. Ces tableaux résument les effets des différentes fautes (colonnes) sur la détection en fonction des scénarios d'attaques utilisés (lignes). Les noms des colonnes sont ensuite construites en deux parties suivant la convention *Type de faute* suivi de *Élément affecté*. Un préfixe *O* caractérise un élément additionnel obsolète alors que le préfixe *M* identifie un élément manquant. Concrètement, dans le cadre de cette expérimentation, les fautes injectées modélisant des éléments additionnels obsolètes correspondent respectivement à :

O_node : Un nouveau nœud est ajouté dans la base de connaissances (dans la DMZ).
O_sig_net_sensor : Une nouvelle règle de détection est ajoutée à la configuration du NIDS qui surveille la DMZ.
O_node_loc_sensor : Un nouveau nœud surveillé par un HIDS est positionné dans la DMZ.
O_sig_log_sensor : Une nouvelle règle de détection est ajoutée à l'un des HIDS des serveurs de la DMZ.

Les fautes modélisant des éléments manquants correspondent aux cas qui suivent :

M_node : L'un des serveurs de la DMZ est retiré de la base de connaissances.
M_net_sensor : Le NIDS surveillant la DMZ est retiré.
M_local_sensor : L'un des HIDS des serveurs de la DMZ est retiré.
M_net_sig : L'une des règles de détection de NIDS surveillant la DMZ est retirée.

Faute	Influence sur la qualité de détection	Prévision de l'intensité de l'influence
éléments additionnels obsolètes	aucune	indépendante du scénario d'attaque
Nœud manquant	faux négatifs seulement	dépend du nombre de branches partant de la racine
Observateur réseau manquant	faux positifs et faux négatifs	très dépendante du scénario d'attaque
Observateur local manquant	faux positifs et faux négatifs	très dépendante du scénario d'attaque
Règle de détection manquante	faux négatifs	très dépendante du scénario d'attaque

TABLE 5.9 – Résumé de l'influence des différentes fautes sur la détection.

L'analyse des résultats de cette expérimentation montre que, dans les grandes lignes, les éléments additionnels obsolètes n'ont pas d'influence sur la qualité de la détection, alors que les éléments manquants altèrent la qualité de manière visible. De plus, l'influence de certaines fautes altèrent la qualité de la détection de manière homogène pour tous les types des scénarios testés (nœuds manquant), alors que l'influence des autres fautes dépend fortement des caractéristiques des scénarios d'attaques. Le tableau 5.9 présente les conclusions que l'on peut tirer à la fois de l'étude théorique sur l'influence des fautes sur la structure de l'arbre mais également de leur l'influence sur la détection rapportée par les deux tableaux précédents. L'influence de chaque faute est ainsi résumée (en termes de possibilités de faux positifs ou de faux négatifs) ainsi que la prévisibilité de l'intensité de ces influences.

Les spécificités des résultats induits par les différents types de fautes sont discutées dans la suite, puis les limites de cette évaluation sont exposés.

Fautes	O_node		O_sig_net_sensor		O_node_loc_sensor		O_sig_loc_sensor	
Scénarios	Rec.	Pre.	Rec.	Pre.	Rec.	Pre.	Rec.	Pre.
S0	100	100	100	100	100	100	100	100
S1	100	100	100	100	100	100	100	100
S2	100	100	100	100	100	100	100	100
S3	100	100	100	100	100	100	100	100
S4	100	100	100	100	100	100	100	100
S5	100	100	100	100	100	100	100	100
S6	100	100	100	100	100	100	100	100
S7	100	100	100	100	100	100	100	100
S8	100	100	100	100	100	100	100	100
S9	100	100	100	100	100	100	100	100
S10	100	100	100	100	100	100	100	100
S11	100	100	100	100	100	100	100	100
S12	100	100	100	100	100	100	100	100

TABLE 5.10 – Précision et rappel pour les éléments additionnels obsolètes.

A Analyse de l'impact des fautes

Entrées additionnelles obsolètes Les expérimentations n'ont pas mis en évidence une influence sur le nombre de faux positifs et de faux négatifs, comme le montrent les résultats rassemblés dans le tableau 5.10. Pour produire des faux positifs, les messages supplémentaires doivent permettre de faire avancer la reconnaissance jusqu'à l'état final. Pour cela, il est nécessaire que le message en question soit présent dans les journaux avec les contraintes satisfaisantes et que l'ordonnancement temporel soit respecté. L'absence de faux négatifs est une conséquence de la prise en compte d'un sur-ensemble des enchaînements de messages possibles dans l'arbre de corrélation.

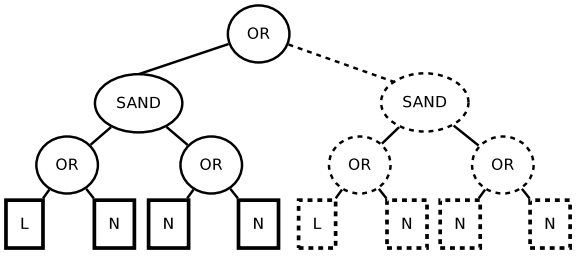


FIGURE 5.9 – Influence de l'absence d'un des deux nœuds intervenant dans le scénario d'attaque. L'une des branches partant de la racine est supprimée, ce qui mène à l'impossibilité de détecter le scénario impliquant les messages supprimés (en pointillés).

Nœud manquant Comme le montre la colonne *M_ node* du tableau 5.11, les résultats sont comparables pour tous les scénarios. La précision n'est pas affectée (pas de faux positifs). Cependant, le rappel compris entre 47% et 57% selon les scénarios, ce qui correspond approximativement à une attaque manquée sur deux. La racine de l'arbre de corrélation est un *OR* qui joint deux chemins d'attaque, chacun d'eux concernant l'un des deux serveurs de la DMZ. Par conséquent, la suppression de ce nœud mène à la disparition de la branche entière, comme indiqué dans l'exemple de la figure 5.9. Le processus d'insertion de traces d'attaque sélectionnant de manière aléatoire l'une des deux branches, cela explique que l'attaque ne soit pas détectée en moyenne une fois sur deux. Chaque levée d'alerte correspond alors à la réalisation d'une instance du scénario d'attaque

Fautes	M_node		M_net_sensor		M_local_sensor		M_net_sig	
Scénarios	Rec.	Pre.	Rec.	Pre.	Rec.	Pre.	Rec.	Pre.
S0	50	100	92	52	76	100	50	100
S1	57	100	90	52	80	100	61	100
S2	50	100	91	51	71	100	55	100
S3	47	100	90	51	81	100	57	100
S4	47	100	94	52	68	100	40	100
S5	49	100	91	52	69	100	49	100
S6	45	100	100	56	100	90	53	100
S7	49	100	100	97	100	92	47	100
S8	57	100	100	95	100	85	63	100
S9	51	100	82	64	71	100	31	100
S10	51	100	33	83	67	100	22	100
S11	49	100	99	50	79	100	68	100
S12	49	100	91	51	73	100	49	100

TABLE 5.11 – Précision (Pre) et rappel (Rec) pour les éléments manquants.

sur l'unique branche restante qui est elle inaltérée par rapport à l'arbre de référence, ce qui explique l'absence de faux positifs.

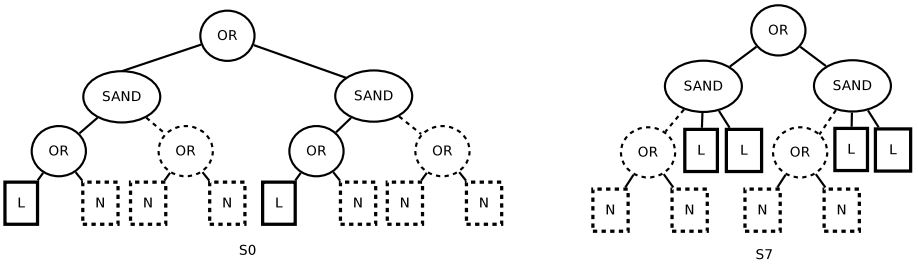


FIGURE 5.10 – Influence de l'absence de l'observateur réseau sur deux arbres de corrélation de structure différente.

Observateur réseau manquant Les mesures de l'influence de cette faute sur la détection sont données par la colonne *M_ net_ sensor* du tableau 5.11. Dans les cas étudiés et illustrés à la figure 5.8, tous les messages identifiés par N proviennent du même NIDS. Son absence entraîne donc leur disparition. La figure 5.10 illustre les différences de structure entre les arbres de corrélation qui correspondent respectivement aux scénarios S0 et S7.

Les scores de rappels et de précisions des scénarios S0, S1, S2, S3, S4, S5 et S12 sont très proches. Le point commun de ces sept scénarios est que leur règle de corrélation associée se limite à un *OR* entre deux messages (appartenant à des observateurs hôtes). D'une part, il suffit que l'un de ces deux messages survienne pour que le corrélateur lève une alerte, d'où le nombre élevé de faux positifs. D'autre part, les faux négatifs correspondent à des cas où la trace d'attaque injectée ne fait pas intervenir d'entrée correspondant à l'un des deux messages locaux.

Pour les scénarios S6, S7 et S8, le nombre de faux positifs diminue avec la taille de l'arbre de corrélation. L'observateur réseau joue un rôle uniquement pour la première action. Son absence a donc moins d'influence lorsque d'autres actions viennent consolider le scénario. L'absence de faux négatifs relatifs à ces trois scénarios est liée à la manière dont les traces d'attaques sont générées. Contrairement à l'action A qui, une fois la construction de l'arbre de corrélation effectuée, est

observable sous la forme d'un message réseau et d'un message local liés par un opérateur *OR*, l'action *B* mène à un arbre composé d'un opérateur *OR* qui lie deux messages réseau et l'action *E* est traduite par un unique message local. Ce message local est toujours sélectionné pour simuler une trace d'attaque dans le scénario *S*7 car ce dernier est le fils d'un opérateur *SAND* et non d'un *OR*. Or l'arbre de corrélation issu de la base de connaissances avec le NIDS manquant est réduit à un *OR* entre ces deux messages locaux [4].

Le rappel le plus faible répertorié est associé au scénario S10. Pour que la règle de détection altérée permettent de détecter l'attaque, il est nécessaire que cette attaque fasse intervenir successivement deux messages d'observateurs locaux, chacun de ceux-ci pouvant être sélectionné ou non par l'algorithme de création de trace d'attaque.

En conclusion, cette faute (observateur réseau manquant) a une influence sur la détection qui dépend fortement du scénario d'attaque.

Observateur local manquant Contrairement à l'observateur réseau, l'observateur local dispose d'une visibilité topologique portant sur un seul nœud. La colonne *M_local_sensor* du tableau 5.11 contient les résultats relatifs à cette faute.

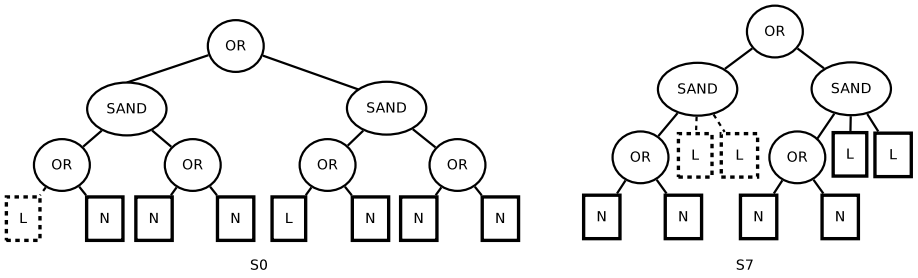


FIGURE 5.11 – Influence de l'absence d'un observateur local à l'un des nœuds sur deux arbres de corrélation de structure différente.

À la lecture de la colonne correspondante du tableau 5.11, la tendance générale qui se dégage est une présence de faux négatifs, mais pas de faux positifs. Cependant, cette tendance n'est pas vraie pour tous les scénarios testés. Les scénarios d'attaque S6, S7 et S8 se distinguent dans ce cas. En effet, contrairement à tous les autres scénarios, ils mènent à des règles de corrélation provoquant des faux positifs, mais pas de faux négatifs. L'absence de faux négatifs s'explique par la présence d'une unique action visible par un IDS réseau alors que les autres actions sont visibles uniquement par un observateur local. Par conséquent, une branche de l'arbre de corrélation ne contient qu'un opérateur *OR* entre deux messages de l'IDS réseau. La levée de l'un des deux messages suffit alors pour considérer le scénario comme détecté, mais va également provoquer des faux positifs.

Les scénarios S0 à S5 ne sont pas affectés en termes de faux positifs par cette faute car la structure de l'arbre fait que le message de l'observateur apparaît uniquement associé à un message réseau par un opérateur *OR*. Par contre, les faux négatifs sont liés à l'absence de reconnaissance des traces d'attaques qui incluent ce message local. Ces configurations issues des scénarios S0 et S7 sont illustrées dans la figure 5.11.

Encore une fois, l'influence de cette faute sur la détection dépend fortement du scénario d'attaque mis en jeu.

Règle de détection manquante La dernière colonne du tableau 5.11 référence les résultats relatifs à cette faute. L'analyse générale de l'influence de cette faute sur la détection révèle que la précision n'est pas impactée (pas de faux positifs) alors que le rappel l'est (présence de faux négatifs). La base de connaissances utilisée lors des expérimentations est construite de telle sorte que la détection de l'action B par l'observateur réseau présent puisse se faire en mettant en jeu deux messages possibles. Autrement dit, il existe deux règles de détection distinctes permettant à l'observateur réseau installé dans la DMZ de détecter l'action B. Par conséquent, les arbres de

4. Par contre, la substitution de l'action *B* par l'action *A* dans les arbres d'actions S6, S7 et S8 aurait causé des faux négatifs.

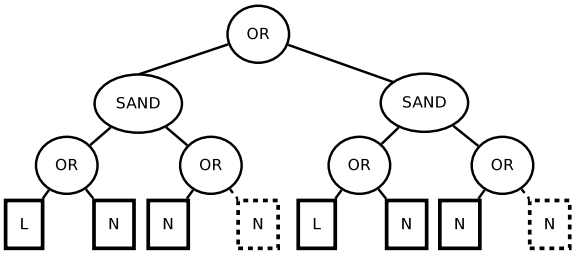


FIGURE 5.12 – Influence de l'absence d'une règle de détection sur l'arbre de corrélation issu du scénario S0.

corrélation générés ne contiennent plus que l'un des deux messages possibles. Lorsqu'une attaque mène uniquement à la génération du message manquant, ce dernier n'apparaît pas dans la règle de corrélation ne fait pas avancer la détection. Par contre, dans ce cas précis, cette absence ne provoque pas de faux positifs car toutes les règles de corrélation contiennent au moins une séquence de deux actions. L'influence de cette faute sur l'arbre de corrélation est illustrée par la figure 5.12.

B Résumé et Discussion

Cette évaluation a porté sur l'influence de la non synchronisation du contenu de la base de connaissances avec l'environnement réel. L'approche utilisée consiste en une comparaison des performances de détection d'une règle de corrélation issue d'une base de connaissances idéale avec les performances de règles de corrélation issues de bases de connaissances contenant des fautes. La mesure de l'influence des fautes sur la détection est classée dans trois catégories :

— Les fautes qui n'ont pas d'influence sur la détection (les éléments additionnels obsolètes) ;
— Les fautes qui ont une influence faiblement dépendante du scénario d'attaque (nœud manquant) ;
— Les fautes ayant une influence qui dépend fortement du scénario d'attaque (les éléments manquants liés aux observateurs)

De plus, les fautes testées ciblent des éléments représentatifs à la fois des acteurs et des observateurs. En reprenant les éléments ci-dessus, il peut être conclu que les fautes qui affectent les acteurs ont une influence sur le nombre de branches de l'arbre de corrélation, ce qui se mesure ensuite par une possibilité de faux négatifs. Les fautes qui affectent des observateurs ont par contre une influence variée sur différentes sous-branches ou feuilles de l'arbre de corrélation, et leur influence sur la détection varie fortement d'un scénario d'attaque à l'autre.

Même si cette approche permet de conclure sur l'influence des fautes à un premier niveau, elle n'est pas exempte de défaut.

Les fautes testées représentent un sous-ensemble des fautes possibles. Elles sont toutes mises sur un pied d'égalité de manière arbitraire. Or, en pratique, rien ne permet de conclure que toutes les catégories de fautes ont la même probabilité de survenir.

De plus, ces expérimentations considèrent uniquement des fautes élémentaires et pas des combinaisons de fautes. Par exemple, nous ne répondons pas à la question de savoir si ces résultats sont cumulatifs où s'il existe des couplages, par exemple dans un cas où la base de connaissances contient à la fois un élément obsolète et un élément manquant.

Les actions utilisées dans les scénarios testés ont toutes un poids équivalent. La mesure de l'impact et de l'influence de l'utilisation d'actions dont l'occurrence naturelle diffère de manière mesurable est un axe d'approfondissement de cette évaluation. Par exemple, en reprenant les notations du tableau 5.7, cela consisterait à mesurer l'influence d'une action A rare par rapport à une action B.

5.4 Passage à l'échelle

Cette partie s'intéresse à une étude permettant de mettre en avant les limites de tailles de règles de corrélation qu'il est possible de générer. Un scénario d'attaque trop générique appliqué à un système de taille importante peut mener à une explosion de la taille de la règle de corrélation. Outre le problème lié à la possibilité de générer effectivement des règles de corrélation de grande taille, une règle de taille trop importante risque de ne pas être utilisable par un corrélateur.

La taille de l'arbre de corrélation est liée à la taille du système. Un nombre important de nœuds pouvant jouer un rôle dans un scénario d'attaque mène à d'autant plus de branches dans l'arbre de corrélation. Cependant, la taille du système n'est pas le seul paramètre pouvant avoir une influence sur la taille de l'arbre de corrélation. Les types de variables (statiques ou dynamiques) jouent également un rôle important étant donné qu'une variable statique est substituée lors de la phase d'identification des acteurs par toutes les valeurs acceptables pour un système et un scénario donné.

Cette section propose une étude théorique sur les paramètres régissant la taille d'un arbre de corrélation puis une évaluation sur un exemple permettant de montrer une limite de notre approche.

5.4.1 Étude théorique

Cette section propose une estimation de la taille maximale d'un arbre de corrélation en fonction de la taille de l'arbre d'actions et de la taille du système. Le but de cette estimation n'est pas de donner une valeur exacte mais plutôt de proposer un ordre de grandeur et un outil de comparaison.

Soit AT un arbre d'actions contenant $N(AT)$ actions. Après identification des acteurs, le nombre de branches est de $N_{br}(AT)$. Sur le système, N_o observateurs peuvent observer une action et pour chacun de ces observateurs, N_m messages au maximum peuvent être associés à la détection d'une action. Le nombre de messages dans l'arbre de corrélation est par conséquent borné par le produit de ces quatre termes, comme indiqué dans l'équation 5.4.1.

$$N_{messages} \leq N(AT) \times N_{br}(AT) \times No \times N_m \tag{5.4.1}$$

Les termes les plus simples à évaluer sont $N(AT)$, No dont une borne supérieure peut être estimée à partir du nombre total d'observateurs ainsi que N_m pour lequel une estimation similaire peut être réalisée. Par exemple, l'arbre d'actions du chapitre 4 contient 4 actions, le nombre total d'observateurs dans le système est de 2 et le nombre maximum de message associé à une action est de 2.

Le terme intéressant qui ne peut être facilement estimé est N_{br}. Pour estimer le nombre de branches de l'arbre de corrélation dans le pire des cas, il faut raisonner à partir de l'ensemble des variables statiques $V_{stat}(AT) = \{ v_1, \dots v_n \}$ en sachant que chaque branche de l'arbre correspond à une substitution spécifique $\{ v_1 = a_1, \dots v_n = a_n \}$ de l'ensemble des variables statiques. Une estimation de ce nombre de substitutions permet donc de déterminer le nombre de branches partant de la racine dans l'arbre de corrélation. En notant nf_i le nombre de faits de la base de connaissances qui représentent l'ensemble des valeurs du même type que la variable statique v_i (par exemple si v_1 représente un identifiant d'un nœud, nf_1 représente le nombre d'identifiants de nœuds renseignés dans la base de connaissances), la borne supérieure de l'ensemble des substitutions possibles est $nf_1 \times nf_2 \times \cdots \times nf_n$. Il est possible de borner cette expression en notant NF le nombre total de faits renseignés dans la base de connaissances. Comme le nombre de faits d'un type donné est inférieur ou égal au nombre de faits (tout type confondu) renseignés dans la base de connaissances, on en déduit $\forall i \in \{1 \dots n\},\ nf_i \leq NF$. Le nombre de substitutions possibles est ainsi borné par NF^n, où n représente le nombre de variables statiques définies dans l'arbre d'action AT.

Nous cherchons désormais à exprimer la complexité en taille de l'arbre de corrélation en fonction du nombre n de variables statiques utilisées dans l'arbre d'actions et du nombre de faits NF dans la base de connaissances. Le terme $N(AT)$ peut être supposé proportionnel au nombre de variables statiques utilisées car au maximum, une action contient k variables statiques, d'où $N(AT) \leq k \times n$. Les termes No et N_m sont bornés par NF car ce sont des faits de la base de connaissances. Il vient que le nombre de messages dans l'arbre de corrélation $N_{messages} \leq n \times NF^{n+2}$. Cette expression modélise le pire des cas possibles et met en évidence le problème d'explosion des états.

5.4.2 Influence des caractéristiques du système et des choix de modélisation

Nous avons vu que la taille de l'arbre de corrélation explose en théorie très rapidement avec la taille de l'arbre d'actions et de la base de connaissances. Nous proposons de réaliser des mesures permettant de trouver les limites pratiques de l'implémentation de la solution ainsi que l'influence de l'introduction de classes d'équivalence sur la taille de l'arbre de corrélation. L'environnement de référence et les scénarios d'attaque utilisés dans le cadre de cette évaluation sont d'abord présentés, suivi de l'analyse des résultats puis d'une discussion des limites de l'approche.

A Environnement de test

La Figure 5.13 modélise l'architecture de référence pour la réalisation des mesures. Il s'agit d'une évolution du système présenté à la figure 5.7 consistant à y ajouter de nouveaux nœuds et des sous-réseaux.

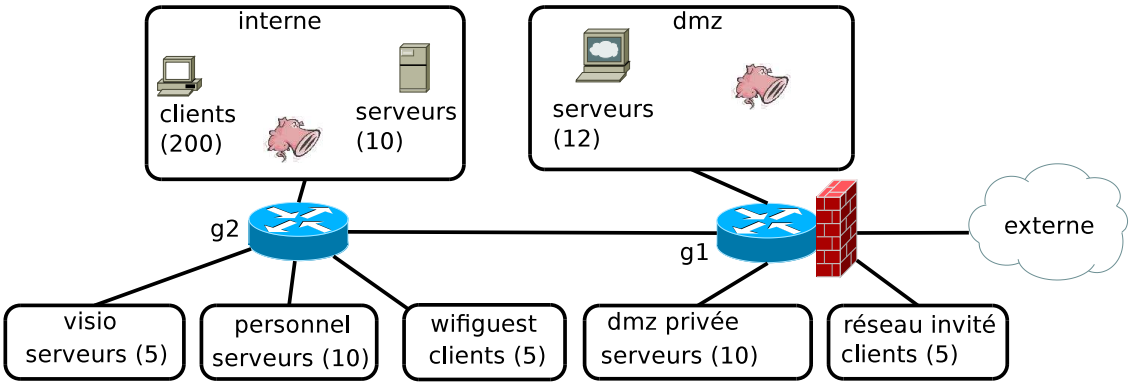


FIGURE 5.13 – Schéma de l'architecture de référence.

Le sous-réseau *internal* contient 200 machines clientes et 10 serveurs. Dix serveurs supplémentaires sont disposés dans la *dmz*, 50 dans la zone *personal* et 5 dans la zone *visio*, *wifiguest* et *guestnet*. Cette configuration de référence est notée *Ref*. Une configuration modifiée identifiée par *Mod* correspond à l'ajout d'un nouveau serveur dans la dmz, d'un poste utilisateur dans le réseau client interne (*internal*) ainsi que d'un NIDS dans le sous-réseau DMZ par rapport au système dans sa configuration initiale.

Dans un sous-réseau donné, tous les postes utilisateurs (respectivement les serveurs) disposent de configurations identiques et l'accessibilité est également identique pour chacune de ces machines. Par conséquent, des classes d'équivalence peuvent être avantageusement définies. Les cas Ref-NA, Ref-EC, Mod-NA et Mod-EC sont ainsi définis à partir des configurations *Ref* et *Mod*. *EC* correspond à l'utilisation de classes d'équivalence et *NA* à l'absence d'utilisation de telles classes.

Lorsque le parc de machines est uniforme, toutes les machines appartenant à une classe d'équivalence peuvent jouer des rôles équivalents au sein d'un même scénario d'attaque. L'arbre de corrélation correspondant est formé d'un ensemble de branches qui diffèrent seulement par les identifiants des nœuds concernés.

B Scénario d'attaque mis en jeu

Les scénarios d'attaque mis en jeu sont décrits dans le tableau 5.12. Il s'agit d'un scénario construit séquentiellement, la première partie comprend les deux premières actions de ce scénario, la seconde ces deux premières actions ainsi que les trois suivantes, ainsi de suite. La première partie correspond à l'attaque initiale d'un serveur, la seconde à un ensemble d'actions locales réalisées par l'attaquant sur ce serveur compromis, la troisième partie correspond aux actions d'un utilisateur qui récupère et exécute un fichier sur le serveur compromis et la dernière partie correspond aux

Actions	Variables statiques	Actions
2	4	a : injection sur un serveur de la DMZ depuis l'extérieur ; b : connexion entre le serveur compromis et la machine de l'attaquant
5	4	a ; b puis trois actions locales aux serveurs compromis correspondant respectivement à : (c) la création d'un fichier ; (d) une élévation de privilège ; (e) une attaque sur l'intégrité d'un fichier de la machine locale.
8	8	a ; b ; c ; d ; e puis f : connexion d'un client du réseau interne vers le serveur compromis ; g : récupération d'un fichier créé par l'attaquant ; h : exécution de ce fichier
9	12	a ; b ; c ; d ; e ; f ; g ; h puis i : connexion du client vers un serveur interne mettant à disposition un service de transfert de fichiers.

TABLE 5.12 – Différentes parties du scénario d'attaque.

Actions	Variables statiques	Ref-NA	Ref-EC	Mod-NA	Mod-EC
2	4	30	3	66	6
5	4	60	6	99	9
8	8	10.10^3	10	15.10^3	14
9	12	(110.10^3)	11	(166.10^3)	15

TABLE 5.13 – Évolution de la taille de l'arbre de corrélation.

actions réalisées par le poste client compromis pour accéder à un serveur interne. Cette construction séquentielle permet de mettre en évidence l'influence de différentes sous-parties de l'environnement.

C Résultats

Le tableau 5.13 présente le nombre de messages constituant l'arbre de corrélation correspondant à chaque scénario d'attaque. L'explosion de la taille correspondant à un arbre d'actions contenant les huit actions s'explique par la participation des postes utilisateurs du sous-réseau *internal* à l'une des actions. La pertinence de l'utilisation des classes d'équivalence est visible dans un cas comme celui-ci. En effet, sans leur utilisation, la taille de l'arbre explose tout en étant constitué d'un grand nombre de branches se différenciant uniquement par l'identifiant des machines référencées.

Le nombre de messages entre parenthèses à la dernière ligne du tableau 5.13 est une estimation de la taille de l'arbre de corrélation, la maquette ne permettant pas de gérer une telle quantité de données. Encore une fois, la configuration mettant en œuvre des classes d'équivalence permet de construire des arbres de corrélation de taille raisonnable sur cet exemple.

D Discussion

Le tableau 5.13 semble montrer que l'utilisation des classes d'équivalence permet de résoudre le problème d'explosion de l'arbre de corrélation. Cependant, il ne s'agit que d'un cas particulier. Imaginons un système similaire à celui présenté à la figure 5.13 mais dans lequel toutes les machines se différencient de telle sorte qu'aucune classe d'équivalence ne peut être avantageusement définie. La conséquence de cette configuration est un retour à la situation dans laquelle les classes d'équivalence ne sont pas utilisées.

5.5 Conclusion

Le processus de génération de règles de corrélation a été évalué selon différents critères. L'applicabilité de la solution à un système représentatif d'une architecture réelle a été testée et a permis de mettre en évidence à la fois la possibilité de modélisation offerte par le modèle ainsi que la capacité des règles de corrélation générées à permettre une détection des scénarios d'attaque correspondants.

Une seconde évaluation centrée sur les conséquences de fautes dans la base de connaissances a mis en évidence l'influence de ces fautes sur l'arbre de corrélation et sur la qualité de la détection. Les résultats permettent de conclure que parmi les fautes testées, seules celles simulant un élément manquant dans la base de connaissances ont une influence mesurable sur la détection.

L'étude de l'évolution de la taille de l'arbre de corrélation en fonction de la taille de l'arbre d'actions et du contenu de la base de connaissances a mis en évidence les défauts de l'approche dans le cas d'une modélisation naïve des nœuds présents sur dans le système d'information. L'arbre de corrélation est alors trop volumineux pour être généré. Cette limitation est repoussée par l'utilisation pertinente de classes d'équivalence entre machines qui permet d'obtenir des arbres de corrélation de taille réduite.

Chapitre 6

Conclusion et travaux futurs

Nous proposons de récapituler les contributions apportées par ces travaux de thèse puis de présenter quelques perspectives d'études issues de problématiques soulevées lors de la réalisation de ces travaux.

6.1 Contributions

Cette thèse s'est positionnée sur les problématiques de corrélation et plus particulièrement sur la reconnaissance explicite de scénarios d'attaque. L'étude sur l'état de l'art nous a amené à constater à notre connaissance l'absence de méthode permettant de créer et de maintenir un ensemble de règles de corrélation adaptées aux caractéristiques du système surveillé. Le constat de ce manque nous a amené à orienter nos travaux vers l'étude d'une méthode permettant de produire des règles de corrélation adaptées au système surveillé. Les contributions principales de cette thèse sont au nombre de trois et sont présentées dans les paragraphes qui suivent.

Le cœur de la thèse consiste en une proposition d'un processus de transformation permettant de construire des règles de corrélation à partir d'un arbre d'attaque. La division de ce processus en plusieurs étapes logiques permet de faire ressortir à chaque fois des fonctions élémentaires ainsi qu'une prise en compte progressive des caractéristiques du système d'information. De toutes ces étapes, seule la première consistant à passer d'un arbre d'attaque à un arbre d'action est manuelle. Ce processus produit un arbre de corrélation, une structure générique qu'il est possible de traduire dans un langage de corrélation adapté. Dans notre cas, le choix s'est porté vers le langage ADeLe.

La seconde contribution est une base de connaissances servant de support au processus de génération des règles de corrélation. Cette base de connaissances est une évolution de M4D4 qui permet de représenter à la fois les biens importants du système d'information mais aussi les éléments permettant de caractériser la détection d'actions d'un attaquant sur ce système. Son contenu est essentiel pour la création des règles de corrélation.

Les autres contributions concernent les évaluations de différents aspects de ce processus de génération ainsi que des règles de corrélation produites. Une première application sur un système représentatif permet de tester l'applicabilité de l'approche. L'évaluation des conséquences de fautes dans la base de connaissances permet de donner un aperçu des éléments ayant un impact sur la qualité des règles de corrélation générées. Un dernier aspect a consisté à évaluer les capacités de passage à l'échelle de la solution proposée ainsi qu'à effectuer une comparaison de la taille des règles de corrélation obtenues en fonction des choix de modélisation.

6.2 Perspectives

Les perspectives d'évolutions ou d'approfondissement des travaux réalisés dans cette thèse concernent plusieurs axes qui sont les règles de corrélation générées au cœur de ces travaux, la base de connaissances et l'évaluation.

Enrichissement des règles de corrélation L'un des axes d'amélioration est la mise en œuvre d'un mécanisme permettant de préciser des événements importants nécessitant une levée d'alertes lors d'une reconnaissance partielle d'un scénario d'attaque. Ceci est intéressant dans des cas d'arbres d'actions construits à partir d'une longue séquence d'actions.

Un axe lié au précédent concerne la possibilité d'utiliser un scénario d'attaque complet préalablement défini comme une simple action au sein d'un second scénario d'attaque. Cette fonctionnalité permettrait de réutiliser un ensemble de scénarios de base déjà définis au sein de nouveaux scénarios d'attaque plus complexes. Une réutilisation d'un scénario demande cependant d'étudier la manière dont les variables issues des différents scénarios interagissent entre elles.

Un grand manque dans les règles de corrélation actuellement générées est l'absence de contraintes temporelles statiquement définies, telles que des délais maximum ou minimum entre l'arrivée de deux messages. Cet élément n'est pas aisément déductible automatiquement car il dépend de l'enchaînement de deux types d'actions. Une solution pourrait consister à laisser la possibilité de définir ces contraintes entre actions manuellement, directement dans l'arbre d'actions.

Éléments structurels Les actions des scénarios d'attaque n'ont pas toutes la même importance et le même rôle dans le scénario d'attaque. Des actions sont plus importantes ou plus significatives que d'autres. Il devient alors intéressant de mettre en place des métriques permettant de faire ressortir ces propriétés au moment de la détection. D'une manière similaire, les messages des observateurs n'ont pas tous la même fiabilité. Cet élément est également intéressant à quantifier et à prendre en compte afin que la reconnaissance du scénario d'attaque puisse être associée à un indicateur de pertinence et de fiabilité. Par exemple, la fiabilité permettrait de fournir une aide pour l'interprétation des cas où plusieurs observateurs peuvent potentiellement observer une action mais que seulement une partie d'entre eux la détecte à l'exécution, menant à la génération d'un sous-ensemble des messages attendus.

L'un des inconvénients des règles de corrélation générées est l'absence de prise en compte de la fusion et de la reconstruction d'attaques élémentaires. Cette problématique survient lorsqu'une action donnée est visible par plusieurs observateurs. Nous avons choisi d'introduire un nouvel opérateur *OR* dans la structure pour lier les différentes observations possibles. Cependant, en considérant la sémantique du langage de corrélation ADeLe, ce choix a pour conséquence de mener à la création de plusieurs plans alors qu'idéalement, un seul contenant l'intégralité des observations relatives à l'action devrait être présent. Une solution consisterait à introduire un nouvel opérateur remplaçant le *OR* dans ce cas précis. Une ébauche de ce que pourrait être cet opérateur est présentée en annexe A.1.

Le choix d'une structure d'arbre d'attaque comme point initial a pour principal avantage la proximité avec la structure d'une règle de corrélation. Cependant, certaines opérations ne sont pas aisément représentables avec cette structure d'arbre, comme des boucles par exemple. La question qui se pose est alors de savoir si l'utilisation de boucles est réellement requis. Dans ce cas, il serait envisageable d'adapter la structure d'arbre d'actions à la forme plus générique d'un graphe d'actions.

Évolution de la base de connaissances Les travaux centrés sur la base de connaissances peuvent se diriger vers les questions du peuplement, de la synchronisation et des problématiques d'intégration d'une telle base au sein d'un système d'information complexe.

Les biens de l'environnement sont systématiquement renseignés manuellement dans nos expérimentations. Ceci ouvre un axe d'amélioration possible consistant à mettre en place un système de peuplement automatique de la base de connaissances pour gérer le grand nombre d'équipements et d'évolutions rencontrés au cours du fonctionnement d'un système d'information. Les problématiques qui se posent alors sont les moyens à mettre en œuvre pour obtenir de manière fiable et automatique l'ensemble des informations requises pour peupler cette base de connaissances.

La question de la dynamicité de la base de biens est également un axe à étudier. La base de biens doit-elle rester une représentation du système à un moment donné ou bien devrait-elle permettre de réaliser des simulations de l'état futur en fonction des actions attendues ? La mise en place d'un système de simulation va potentiellement demander un niveau de modélisation plus poussé (par exemple avec une modélisation des différentes couches des protocoles réseau).

Évaluation et amélioration des performances L'axe principale d'amélioration concerne les performances pour le passage à l'échelle. Même si les classes d'équivalences apportent une solution curative, elles ne permettent pas d'appliquer la méthode sur un environnement constitué de nombreux nœuds très hétérogènes. Il faudrait alors étudier une manière de générer un arbre de corrélation en factorisant les parties redondantes lorsque cela est possible. La figure 6.1 illustre une factorisation possible. Lors de la phase d'identification des acteurs, des nœuds redondants sont générés. L'arbre (1) correspond par exemple à l'exploitation d'un premier service réseau ($A1$) sur une machine suivie d'une élévation de privilège (B) et $A2$ correspond à l'exploitation d'une vulnérabilité d'un autre service réseau présent sur la même machine. Dans des scénarios plus longs, d'autres actions jouent un rôle similaire à l'action instanciée B. Sur l'arbre (2), la redondance a été supprimée. Si cette opération semble réalisable une fois l'arbre entièrement généré, il reste à étudier les moyens pour l'appliquer à la volée afin d'éviter la manipulation d'une structure dont la taille est trop importante.

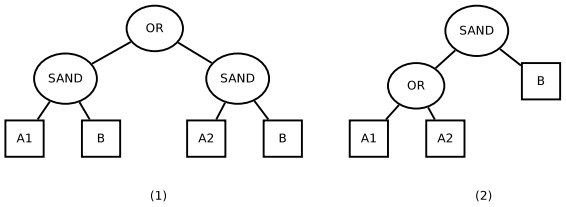


FIGURE 6.1 – Processus de factorisation. En (1), arbre non factorisé, en (2), arbre factorisé.

Concernant l'évaluation, un élargissement des cas étudiés dans le cadre de l'évaluation des conséquences de l'injection de fautes dans la base de connaissances serait intéressant. En particulier, il peut être envisagé de tester l'influence non pas de fautes élémentaires mais des combinaisons de plusieurs types de fautes. Nos expérimentations ont de plus été menées sur un unique système. Il serait intéressant de mettre en œuvre ce même type d'évaluation sur des systèmes dont l'architecture et la taille peuvent varier.

Un autre point intéressant à évaluer concerne l'impact des différences de probabilité d'occurrences de différentes actions au sein d'un même arbre d'actions. Par exemple, dans un arbre d'actions constitué d'une séquence de trois actions *SAND(A, B, C)*, les occurrences attendues de ces actions A, B et C sont sans doute différentes. Supposons que l'action A corresponde à une attaque rare alors que l'action B soit une action normale courante et C soit une action d'attaque plus courante que A. L'observabilité relative de chaque action influence alors la qualité de la détection. Par exemple, si l'action A n'est pas observable, alors il est possible que la règle de corrélation générée soit susceptible de lever un plus grand nombre de faux positifs que lorsque l'action B est la seule à ne pas être observable.

Annexe A

Annexes

A.1 Généralisation de l'opérateur OR pour agréger les observateurs

Lors de l'étape de création de l'arbre de corrélation, l'ajout d'un opérateur *OR* permet de regrouper les observations potentielles de plusieurs observateurs relativement à une action. Cependant, lors de la traduction de l'arbre de corrélation dans le langage ADeLe, cet opérateur est traduit par l'opérateur *OneAmong*. Les propriétés de cet opérateurs ont pour conséquence l'impossibilité de réaliser une instance de détection contenant l'intégralité des messages levées par tous les observateurs d'une même action. Par exemple, si pour une action donnée, trois observations a, b ou c sont possibles de manière non exclusives, alors aucun plan de reconnaissance ne contiendra à la fois des instances de a et de b (respectivement de a et c et de b et c).

La définition d'un nouvel opérateur dont la traduction sous la d'un forme d'automate est donnée à la figure A.1 permettrait de repousser cette limite. Dans cet automate, un timer permet de temporiser pour collecter les instances a, b et c. Lorsque le temps est écoulé, l'automate passe à l'état suivant si au moins un des événements est détecté. L'introduction de ce mécanisme demande cependant de définir la manière dont les contraintes entre les messages sont gérées, le comportement à adopter lorsque plusieurs instances d'un même événements sont reçues dans l'intervalle de temps.

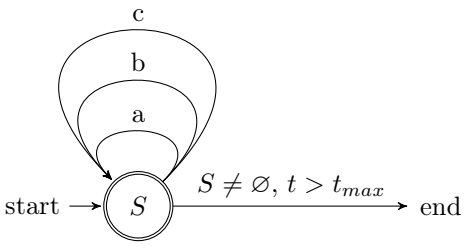


FIGURE A.1 – Exemple d'un automate permettant de réaliser une fusion des observations d'une action.

A.2 Problématique liée au chiffrement

La présence de chiffrement peut limiter la capacité de certaines sondes réseaux à détecter certaines actions. Une modélisation de ce mécanisme demande une prise en compte de plusieurs dimensions caractérisant les propriétés du flux chiffré et le fonctionnement des observateurs. Ces dimensions sont explicitées ici pour exposer le problème.

Types de flux chiffrés Concernant le statut du flux (chiffré ou non chiffré), différents cas peuvent être rencontrés :

(a) certains flux entre deux nœuds sont toujours chiffrés (ex : accès SSH) ;

(b) tous les flux entre deux nœuds sont chiffrés (VPN local) ;

(c) les flux sont chiffrés sur certaines portions du réseau et en clair sur d'autres (passerelle IPSEC, serveurs mandataires inverses) ;

(d) un flux peut subir plusieurs niveaux de chiffrement (SSL/TLS dans de l'IPSEC, emails chiffrés et échangés en SSL/TLS, etc.).

Cette caractérisation du flux chiffré ne suffit pas car les observateurs ont différents comportements face au chiffrement.

Comportement d'un observateur face au chiffrement Un observateur réseau peut :

(a) utiliser des signatures portant sur le contenu en clair (par exemple une expression rationnelle d'une signature Snort pour détecter un contenu malveillant) ;

(b) utiliser des caractéristiques d'un flux donné (interdiction de se connecter en SSH sur certaines machines) ;

(c) utiliser des statistiques sur les flux (ex : calcul rapport paquet émis/reçus) ;

(d) être placé en coupure et déchiffrer à la volée certaines connexions chiffrées.

Justification de l'absence de modélisation spécifique Une modélisation permettant de prendre en compte tous ces cas (et leurs combinaisons) demande d'avoir au minimum :

(a) une modélisation des protocoles et des couches TCP/IP pour définir les protocoles chiffrés et non chiffrés ;

(b) une indication du type de protocole utilisé pour chaque service réseau ;

(c) une modélisation des passerelles VPN ou mécanismes qui permettent de chiffrer ou déchiffrer un flux ;

(d) une indication sur la capacité d'une règle de détection à détecter une action donnée dans un flux chiffré.

(e) de la capacité d'une sonde à déchiffrer le trafic

L'ajout de tous ces éléments pour raffiner la visibilité de certaines sondes complexifie la modélisation de manière significative et se répercute sur les autres éléments du modèle, par exemple sur les règles de filtrage, la modélisation des protocoles et des services. De plus, la présence de sondes réseaux utilisant des règles nécessitant un accès au contenu des flux sans mise en œuvre de déchiffrement peut être considéré comme une erreur d'architecture ou de déploiement. Nous justifions donc l'absence de modélisation de cette composante dans le modèle. En contrepartie, le modèle considère certaines actions non détectables en réalité comme détectable s'il se trouve qu'une sonde réseau sans moyen de déchiffrement surveille un flux chiffré. La conséquence sur la détection est un risque de faux négatif lié à la non-détection des actions en question.

A.3 Liste des publications

Erwan Godefroy, Éric Totel, Frédéric Majorczyk, and Michel Hurfin. Génération automatique de règles de corrélation pour la détection d'attaques complexes. In *9eme conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR-SSI).*, 2014.

Erwan Godefroy, Éric Totel, Frédéric Majorczyk, Michel Hurfin, and Amine Maaroufi. Automatiser la construction de regles de corrélation : prérequis et processus. In *C&ESAR 2014-Détection et réaction face aux attaques informatiques.*, 2014.

Erwan Godefroy, Éric Totel, Michel Hurfin, and Frédéric Majorczyk. Automatic generation of correlation rules to detect complex attack scenarios. In *Information Assurance and Security (IAS),*

2014 10th International Conference on., pages 23-28, IEEE, 2014.

Erwan Godefroy, Éric Totel, Michel Hurfin, and Frédéric Majorczyk. Automatic generation of correlation rules to detect complex attack scenarios. *Journal of Information Assurance and Security*, 10(4), 2015.

Erwan Godefroy, Éric Totel, Michel Hurfin, and Frédéric Majorczyk. Assessment of an Automatic Correlation Rules Generator. In *Eleventh International Conference on Information Systems Security (ICISS 2015).*, 2015.

Poster :
Erwan Godefroy, Éric Totel, Michel Hurfin, and Frédéric Majorczyk. Generation and assessment of correlation rules to detect complex attack scenarios. In *Communications and Network Security (CNS), 2015 IEEE Conference on.*, pages 707-708, IEEE, 2015.

Bibliographie

[AAA+10] Faeiz Alserhani, Monis Akhlaq, Irfan U Awan, Andrea J Cullen, and Pravin Mirchandani. Mars : multi-stage attack recognition system. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 753–759. IEEE, 2010.

[AASB08] K. Alsubhi, E. Al-Shaer, and R. Boutaba. Alert prioritization in intrusion detection systems. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 33 –40, april 2008.

[AD99] D. Alessandri and M. Dacier. Vulda : A vulnerability database. Technical Report RZ3111, IBM Zurich Research Laboratory, March 1999.

[AJA11] Seyed Hossein Ahmadinejad, Saeed Jalili, and Mahdi Abadi. A hybrid model for correlating alerts of known and unknown attack scenarios and updating attack graphs. *Computer Networks*, 55(9) :2221–2240, 2011.

[AJN12] Massimiliano Albanese, Sushil Jajodia, and Steven Noel. Time-efficient and cost-effective network hardening using attack graphs. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–12. IEEE, 2012.

[AJPS11] Massimiliano Albanese, Sushil Jajodia, Andrea Pugliese, and VS Subrahmanian. Scalable analysis of attack scenarios. In *Computer Security–ESORICS 2011*, pages 416–433. Springer, 2011.

[Ale04] Dominique Alessandri. *Attack-class-based analysis of intrusion detection systems*. PhD thesis, University of Newcastle upon Tyne School of Computing Science, 2004.

[AMZ09] Safaa O Al-Mamory and Hongli Zhang. Ids alerts correlation using grammar-based approach. *Journal in computer virology*, 5(4) :271–282, 2009.

[APRS05] Paul Ammann, Joseph Pamula, Ronald Ritchey, and J des Street. A host-based approach to network attack chaining analysis. In *Computer Security Applications Conference, 21st Annual*. IEEE, 2005.

[Art02] Michael Lyle Artz. *Netspa : A network security planning architecture*. PhD thesis, Massachusetts Institute of Technology, 2002.

[AWK02] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Conference on Computer and Communications Security (CCS)*, 2002.

[BB13] Mehdi Bateni and Ahmad Baraani. Time window management for alert correlation using context information and classification. *International Journal of Computer Network and Information Security (IJCNIS)*, 5(11) :9, 2013.

[BBG13] Mehdi Bateni, Ahmad Baraani, and Ali Akbar Ghorbani. Using artificial immune system and fuzzy logic for alert correlation. *Int. J. Netw. Secur*, 15(1) :160–174, 2013.

[BBGR13] Mehdi Bateni, Ahmad Baraani, Ali Ghorbani, and Abbas Rezaei. An ais-inspired architecture for alert correlation. *International Journal of innovative Computing, Information & Control*, 9(1) :231–255, 2013.

[BCTG13] Fabrizio Baiardi, Fabio Coro, Federico Tonelli, and Luca Guidi. Gvscan : Scanning networks for global vulnerabilities. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pages 670–677. IEEE, 2013.

[BCTS14] Fabrizio Baiardi, Fabio Coro, Federico Tonelli, and Daniele Sgandurra. A scenario method to automatically assess ict risk. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 544–551. IEEE, 2014.

[BKM08] Salem Benferhat, Tayeb Kenaza, and Aicha Mokhtari. Réseaux bayésiens naifs pour la détection des attaques coordonnées. In *Journées Francophone sur les Réseaux Bayésiens*, 2008.

[BLP+06] Ahto Buldas, Peeter Laud, Jaan Priisalu, Märt Saarepera, and Jan Willemson. Rational choice of security measures via multi-parameter attack trees. In *Critical Information Infrastructures Security*, volume 4347 of *Lecture Notes in Computer Science*, pages 235–248. Springer Berlin Heidelberg, lopez, javier edition, 2006.

[BP03] P. J. Brooke and R. F Paige. Fault trees for security system design and analysis. *Computers & Security*, 22(3) :256–264, 2003.

[CLF03] Steven Cheung, Ulf Lindqvist, and Martin W Fong. Modeling multistep cyber attacks for scenario recognition. In *DARPA information survivability conference and exposition, 2003. Proceedings*, volume 1, pages 284–292. IEEE, 2003.

[CM02] Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 202–215, 2002.

[CNTBB04] Tobias Chyssler, Simin Nadjm-Tehrani, Stefan Burschka, and Kalle Burbeck. Alarm reduction and correlation in defence of ip networks. In *Proceedings of the 13th IEEE International Workshops on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE'04)*, pages 229–234. IEEE Computer Society, 2004.

[CO00] Frédéric Cuppens and Rodolphe Ortalo. LAMBDA : A Language to Model a Database for Detection of Attacks. In H. Debar, L. Mé, and S. F. Wu, editors, *Proceedings of the Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, number 1907 in LNCS, pages 197–216, October 2000.

[cY07] Seyit Ahmet Çamtepe and Bülent Yener. Modeling and detection of complex attacks. In *Proceedings of the 3rd International Conference on Security and Privacy in Communications Networks*, Nice, France, September 2007.

[CYP10] Jun Chang, Jiang Yu, and Yijian Pei. Ms2ifs : A multiple source-based security information fusion system. In *Communications and Intelligence Information Security (ICCIIS), 2010 International Conference on*, pages 215–219. IEEE, 2010.

[DC01] Oliver M. Dain and Robert K. Cunningham. Building scenarios from a heterogeneous alert stream. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, 2001.

[DH04] Jerald Dawkins and John Hale. A systematic approach to multi-stage network attack analysis. In *Second IEEE International Information Assurance Workshop (IWIA'04)*, page 48, 2004.

[DW01] Hervé Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In W. Lee, L. Mé, and A. Wespi, editors, *Proceedings of the Fourth International Symposium on the Recent Advances in Intrusion Detection (RAID'2001)*, number 2212 in LNCS, pages 85–103, October 2001.

[EVK00] Steven T. Eckmann, Giovanni Vigna, and Richard A. Kemmerer. Statl : An attack language for state-based intrusion detection. In *Proceedings of the ACM Workshop on Intrusion Detection*, November 2000.

[FAK15] Hamid Farhadi, Maryam AmirHaeri, and Mohammad Khansari. Alert correlation and prediction using data mining and hmm. *The ISC International Journal of Information Security*, 3(2), 2015.

[Fra09] Virginia Nunes Leal Franqueira. *Finding multi-step attacks in computer networks using heuristic search and mobile ambients*. PhD thesis, University of Twente, 2009.

[FVEWL09] Virginia NL Franqueira, Pascal Van Eck, Roel Wieringa, and Raul HC Lopes. A mobile ambients-based approach for network attack modelling and simulation. In *Availability, Reliability and Security, 2009. ARES'09. International Conference on*, pages 546–553. IEEE, 2009.

[FWSJ08] Marcel Frigault, Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring network security using dynamic bayesian network. In *Proceedings of the 4th ACM workshop on Quality of protection*, pages 23–30. ACM, 2008.

[GGB13] Mohammad GhasemiGol and Abbas Ghaemi-Bafghi. A new alert correlation framework based on entropy. In *Computer and Knowledge Engineering (ICCKE), 2013 3th International eConference on*, pages 184–189. IEEE, 2013.

[GHH+01] Robert P Goldman, Walter Heimerdinger, Steven Harp, Christopher W Geib, Vicraj Thomas, Robert L Carter, et al. Information modeling for intrusion report aggregation. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, volume 1, pages 329–342. IEEE, 2001.

[GKAD07] Mohammed S Gadelrab, El Kalam, Anas Abou, and Yves Deswarte. Defining categories to select representative attack test-cases. In *Proceedings of the 2007 ACM workshop on Quality of protection*, pages 40–42. ACM, 2007.

[GLO08] Jean Goubault-Larrecq and Julien Olivain. A smell of orchids. In *Runtime Verification*, volume 5289 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008.

[GMHD12] Gustavo Gonzalez Granadillo, Yosra Ben Mustapha, Nabil Hachem, and Herve Debar. An ontology-based model for siem environments. In Springer, editor, *ICGS3 '11 : 7th International Conference in Global Security, Safety and Sustainability*, volume 99, pages 148–155, Thessalonik, Greece, 2012.

[GW07] Suvajit Gupta and Joel Winstead. Using attack graphs to design systems. *Security & Privacy, IEEE*, 5(4) :80–83, 2007.

[HG07] Jorge Herrerias and Roberto Gomez. A log correlation model to support the evidence search process in a forensic investigation. In *Proceedings of the Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'07)*, 2007.

[How97] John D. Howard. *An Analysis Of Security Incidents On The Internet 1989 - 1995*. PhD thesis, CARNEGIE MELLON UNIVERSITY, April 1997.

[HS14] Neminath Hubballi and Vinoth Suryanarayanan. False alarm minimization techniques in signature-based intrusion detection systems : A survey. *Computer Communications*, 49 :1–17, 2014.

[ICL+09] Kyle Ingols, Matthew Chu, Richard Lippmann, Seth Webster, and Stephen Boyer. Modeling modern network attacks and countermeasures using attack graphs. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pages 117–126. IEEE, 2009.

[ILP06] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 121–130. IEEE, 2006.

[IW08] V Igure and R Williams. Taxonomies of attacks and vulnerabilities in computer systems. *Communications Surveys & Tutorials, IEEE*, 10(1) :6–19, 2008.

[JN07] Sushil Jajodia and Steven Noel. Topological vulnerability analysis : A powerful new approach for network attack prevention, detection, and response. *Indian Statistical Institute Monograph Series*, 2007.

[JN10a] Sushil Jajodia and Steven Noel. Advanced cyber attack modeling analysis and visualization. Technical report, DTIC Document, 2010.

[JN10b] Sushil Jajodia and Steven Noel. Topological vulnerability analysis. In *Cyber Situational Awareness*. Springer, 2010.

[JNO05] Sushil Jajodia, Steven Noel, and Brian O'Berry. Topological analysis of network attack vulnerability. In *Managing Cyber Threats*, volume 5 of *Massive Computing*. Springer US, 2005.

[JSW02] Somesh Jha, Oleg Sheyner, and Jeannette M. Wing. Minimization and reliability analyses of attack graphs. Technical Report CMU-CS-02-109, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2002.

[Jul03] Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security*, 6(4), 2003.

[JWCY09] Shuyuan Jin, Yong Wang, Xiang Cui, and Xiaochun Yun. A review of classification methods for network vulnerability. In *Proceedings of the 2009 IEEE international conference on Systems, Man and Cybernetics*, pages 1171–1175. IEEE Press, 2009.

[KA13] Goran Kap and Dana Ali. Statistical analysis of computer network security. Master's thesis, KTH Royal Institute of Technology, Sweden, October 2013.

[KBN05] Dongyoung Kim, HyoChan Bang, and Jung-Chan Na. Intrusion alert normalization method using awk scripts and attack name database. In *Advanced Communication Technology, 2005, ICACT 2005. The 7th International Conference on*, volume 1, pages 608 – 611 Vol. 1, feb. 2005.

[KDP+12] Wael Kanoun, Samuel Dubus, Serge Papillon, Nora Cuppens-Boulahia, and Frédéric Cuppens. Towards dynamic risk management : Success likelihood of ongoing attacks. *Bell Labs Technical Journal*, 17(3) :61–78, 2012.

[Ken99] Kristopher Kendall. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1999.

[KG13] Robert Koch and Mario Golling. Architecture for evaluating and correlating nids in real-world networks. In *Cyber Conflict (CyCon), 2013 5th International Conference on*, pages 1–20. IEEE, 2013.

[Kha09] Parvaiz Ahmed Khand. System level security modeling using attack trees. In *In Proceedings of the 2nd International Conference on Computer, Control and Communication (IC4)*, pages 1–6, 2009.

[KMRS11] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. Foundations of attack-defense trees. In Springer, editor, *Lecture Notes in Computer Science*, volume 6561, pages 80–95, 2011.

[KMRS12] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. Attack–defense trees. *Journal of Logic and Computation*, 2012.

[KMT04] Kevin S Killourhy, Roy A Maxion, and Kymie MC Tan. A defense-centric taxonomy based on attack manifestations. In *Dependable Systems and Networks, 2004 International Conference on*, pages 102–111. IEEE, 2004.

[KPCS13] Barbara Kordy, Ludovic Pietre-Cambacedes, and Patrick Schweitzer. Dag-based attack and defense modeling : Don't miss the forest for the attack trees. *CoRR*, abs/1303.7397, 2013.

[KRV04] Christopher Kruegel, William Robertson, and Giovanni Vigna. Using alert verification to identify successful intrusion attempts. *Practice in Information Processing and Communication (PIK)*, 27(4), October 2004.

[Lag10] Philippe Lagadec. Visualisation et Analyse de Risque Dynamique pour la Cyber-Défense. In *Proceedings of SSTIC'2010*, pages 3–31, 2010.

[LFXX10] Xuejiao Liu, Chengfang Fang, Debao Xiao, and Hui Xu. A goal-oriented approach for modeling and analyzing attack graph. In *Information Science and Applications (ICISA), 2010 International Conference on*, pages 1–8. IEEE, 2010.

[LI05] Richard Paul Lippmann and Kyle William Ingols. An annotated review of past papers on attack graphs. Technical report, DTIC Document, 2005.

[LIS+05] Richard P Lippmann, Kyle W Ingols, Chris Scott, Keith Piwowarski, Kendra J Kratkiewicz, Michael Artz, and RK Cunningham. Evaluating and strengthening enterprise network security using attack graphs. Technical report, Lincoln Laboratory Massachusetts Institute of Technology, 2005.

[LJ97] Ulf Lindqvist and Erland Jonsson. How to systematically classify computer security intrusions. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, 1997.

[LLWL07] Zhi-tang Li, Jie Lei, Li Wang, and Dong Li. A data mining approach to generating network attack graph for intrusion prediction. In *Fuzzy Systems and Knowledge Discovery, 2007. FSKD 2007. Fourth International Conference on*, volume 4, pages 307–311. IEEE, 2007.

[LM01] Richard C. Linger and Andrew P. Moore. Foundations for survivable system development : Service traces, intrusion traces, and evaluation models, 2001.

[Lou01] Daniel Lowry Lough. *A Taxonomy of Computer Attacks with Applications to Wireless Networks*. PhD thesis, Virginia Polytechnic Institute and Sate University, 2001.

[LT09] Wan Li and Shengfeng Tian. Preprocessor of intrusion alerts correlation based on ontology. In *Communications and Mobile Computing, 2009. CMC'09. WRI International Conference on*, volume 3, pages 460–464. IEEE, 2009.

[LWC08] Zhijie Liu, Chongjun Wang, and Shifu Chen. Correlating multi-step attack and constructing attack scenarios based on attack pattern modeling. In *International conference on Information Security and Assurance ISA 2008*, pages 214–219, 2008.

[LXP08] Xuejiao Liu, Debao Xiao, and Xi Peng. Towards a collaborative and systematic approach to alert verification. *Journal of Software*, 3(9) :77–84, 2008.

[MBFB06] Miles A. McQueen, Wayne F. Boyer, Mark A. Flynn, and George A. Beitel. Quantitative cyber risk reduction estimation methodology for a small scada control system. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, volume 9, Hawaii, USA, January 2006.

[McH00] John McHugh. Testing intrusion detection systems : a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transaction on Information and System Security*, 3(4) :262–294, 2000.

[MIS12] Ashara Banu Mohamed, Norbik Bashah Idris, and Bharanidharan Shanmugum. Alert correlation framework using a novel clustering approach. In *Computer & Information Science (ICCIS), 2012 International Conference on*, volume 1, pages 403–408. IEEE, 2012.

[MLB08] Frederic Massicotte, Yvan Labiche, and Lionel C. Briand. Toward automatic generation of intrusion detection verification rules. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC'2008)*, 2008.

[MMC11] Fabio Manganiello, Mirco Marchetti, and Michele Colajanni. Multistep attack detection and alert correlation in intrusion detection systems. In *Information Security and Assurance*. Springer, 2011.

[MMDD02] Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducassé. M2D2 : A Formal Data Model for IDS Alert Correlation. In Andreas Wespi, Giovanni Vigna, and Luca Deri, editors, *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID'2002)*, volume 2516 of *Lecture Notes in Computer Science*, pages 115–127. Springer, 2002.

[MMDD09] Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Duccassé. M4d4 : a logical framework to support alert correlation in intrusion detection. *Information Fusion*, 10(4) :285–299, October 2009.

[Mor04] Benjamin Morin. *Corrélation d'alertes issues d'outils de détection d'intrusions avec prise en compte d'informations sur le système surveillé*. PhD thesis, INSA de Rennes, 2004.

[NCR02] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *In Proceedings of the 9th ACM conference on Computer and communications security*, pages 245–254, 2002.

[ND07] Sanjeeb Nanda and Narsingh Deo. A highly scalable model for network attack identification and path prediction. In *SoutheastCon, 2007. Proceedings. IEEE*, pages 663–668, 2007.

[NEJ+09] Steven Noel, Matthew Elder, Sushil Jajodia, Pramod Kalapa, Scott O'Hare, and Kenneth Prole. Advances in topological vulnerability analysis. In *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*, pages 124–129. IEEE, 2009.

[NJ08] Steven Noel and Sushil Jajodia. Optimal ids sensor placement and alert prioritization using attack graphs. *Journal of Network and Systems Management*, 16(3) :259–275, 2008.

[NRJ04] S. Noel, E. Robertson, and S. Jajodia. Correlating intrusion events and building attack scenarios through attack graph distances. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 350–359, 2004.

[NX04] Peng Ning and Dingbang Xu. Hypothesizing and reasoning about attacks missed by intrusion detection systems. *ACM Trans. Inf. Syst. Secur.*, 7(4) :591–627, nov 2004.

[NXHA04] Peng Ning, Dingbang Xu, Christopher G. Healey, and Robert St. Amant. Building attack scenarios through integration of complementary alert correlation methods. In *11th Annual Network and Distributed System Security Symposium*, 2004.

[OBM06] Xinming Ou, Wayne F Boyer, and Miles A McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345. ACM, 2006.

[OMSH03] Dirk Ourston, Sara Matzner, William Stump, and Bryan Hopkins. Application of hidden markov models to detecting multi-stage network attacks. In *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)*, 2003.

[OSR13] Jorge Lucangeli Obes, Carlos Sarraute, and Gerardo Richarte. Attack planning in the real world. *arXiv preprint arXiv :1306.4044*, 2013.

[PC10] Ludovic Piètre-Cambacédès. *Des relations entre sûreté et sécurité*. PhD thesis, Telecom ParisTech, november 2010.

[PD01] Jean-Philippe Pouzol and Mireille Ducassé. From Declarative Signatures to Misuse IDS. In W. Lee, L. Mé, and A. Wespi, editors, *Proceedings of the Fourth International Symposium on the Recent Advances in Intrusion Detection (RAID'2001)*, number 2212 in LNCS, pages 1–21, October 2001.

[PD02] Jean-Philippe Pouzol and Mireille Ducassé. Formal Specification of Intrusion Signatures and Detection Rules. In *Proceedings of the 15th IEEE Computer Security Foudations Workshop (CSFW'02)*, pages 64–76. IEEE Computer Society, June 2002.

[PDR12] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. Dynamic security risk management using bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing*, 9(1) :61–74, Jan-Feb 2012.

[PFV02] Phillip A. Porras, Martin W. Fong, and Alfonso Valdes. A mission-impact-based approach to infosec alarm correlation. In Andreas Wespi, Giovanni Vigna, and Luca Deri, editors, *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID'2002)*, volume 2516 of *Lecture Notes in Computer Science*, pages 95–114. Springer, 2002.

[PTC+14] Manuel Gil Pérez, Juan E Tapiador, John A Clark, Gregorio Martínez Pérez, and Antonio F Skarmeta Gómez. Trustworthy placements : Improving quality and resilience in collaborative attack detection. *Computer Networks*, 58 :70–86, 2014.

[QL04a] Xinzhou Qin and Wenke Lee. Attack plan recognition and prediction using causal networks. In *Computer Security Applications Conference*, pages 370 – 379, 2004.

[QL04b] Xinzhou Qin and Wenke Lee. Discovering novel attack strategies from infosec alerts. In *In Proceedings of the 9th European Symposium on Research in Computer Security, Sophia Antipolis*, pages 439–456, 2004.

[RA00] Ronald W. Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, May 2000.

[RC07] Jonathan Rouzaud-Cornabas. Corrélation d'événements et d'alarmes pour la détection d'intrusions dans les systèmes répartis. Master's thesis, ENSI-Bourges, 2007.

[RCM10] Sebastian Roschke, Feng Cheng, and Christoph Meinel. A flexible and efficient alert correlation platform for distributed ids. In *Network and System Security (NSS), 2010 4th international conference on*, pages 24–31. IEEE, 2010.

[RCM11] Sebastian Roschke, Feng Cheng, and Christoph Meinel. A new alert correlation algorithm based on attack graph. In *Computational Intelligence in Security for Information Systems*. Springer, 2011.

[RD11] Elias Raftopoulos and Xenofontas Dimitropoulos. Detecting, validating and characterizing computer infections from ids alerts. Technical report, ETH Zurich, 2011.

[RP05] Indrajit Ray and Nayot Poolsapassit. Using attack trees to identify malicious attacks from authorized insiders. In *Computer Security–ESORICS 2005*. Springer, 2005.

[RSG10] Hanli Ren, Natalia Stakhanova, and Ali A Ghorbani. An online adaptive approach to alert correlation. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2010.

[SBD+91] Steven R. Snapp, James Brentano, Gihan V. Dias, Terrance L. Goan, L. Todd Heberlein, Che lin Ho, Karl N. Levitt, Biswanath Mukherjee, Stephen E. Smaha, Tim Grance, Daniel M. Teal, and Doug Mansur. Dids (distributed intrusion detection system) - motivation, architecture, and an early prototype. In *In Proceedings of the 14th National Computer Security Conference*, pages 167–176, 1991.

[Sch99] B. Schneier. Attack trees : Modeling security threats. *Dr. Dobb's Journal*, 24(12) :21–29, 1999.

[SCS11] Xinming Ou Sathya Chandran Sundaramurthy, Loai Zomlot. Practical ids alert correlation in the face of dynamic threats. In *The 2011 International Conference on Security and Management*, 2011.

[SEH13] Teodor Sommestad, Mathias Ekstedt, and Hannes Holm. The cyber security modeling language : A tool for assessing the vulnerability of enterprise system architectures. *Systems Journal, IEEE*, 7(3) :363–373, 2013.

[SG06] Reza Sadoddin and Ali Ghorbani. Alert correlation survey : framework and techniques. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust : Bridge the Gap Between PST Technologies and Business Services*, page 37. ACM, 2006.

[SG09] R. Sadoddin and A. A. Ghorbani. An incremental frequent structure mining framework for real-time alert correlation. *Computers & Security*, 28(3-4) :153–173, 2009.

[SH04] Ray Hunt Simon Hansman. A taxonomy of network and computer attacks. *Computers & Security*, 2004.

[SHJ+02] Oled Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Symposium on Security and Privacy*, 2002.

[SJ01] Helen Svensson and Audun Jøsang. Correlation of intrusion alarms with subjective logic. In *Proceedings of the sixth Nordic Workshop on Secure IT systems (NordSec2001), Copenhagen, Denmark*, 2001.

[SJDM08] Reuben Smith, Nathalie Japkowicz, Maxwell Dondo, and Peter Mason. Using unsupervised learning for network alert correlation. In *Canadian AI'08 Proceedings of the Canadian Society for computational studies of intelligence, 21st conference on Advances in artificial intelligence*, pages 308–319, 2008.

[SMC09] Robert Schuppenies, Christoph Meinel, and Feng Cheng. Automatic extraction of vulnerability information for attack graphs. *Hasso-Plattner-Institute for IT Systems Engineering, University of Potsdam*, 2009.

[SMFDV13] Saeed Salah, Gabriel Maciá-Fernández, and Jesús E Díaz-Verdejo. A model-based survey of alert correlation techniques. *Computer Networks*, 2013.

[SO08] Reginald E. Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In *Proceedings of the 13th European Symposium on Research in Computer Security : Computer Security*, ESORICS '08, pages 18–34. Springer-Verlag, 2008.

[SP03] Umesh Shankar and Vern Paxson. Active mapping : Resisting nids evasion without altering traffic. In *SP '03 : Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 44, Washington, DC, USA, 2003. IEEE Computer Society.

[SPB12] Raheel Hassan Syed, Jasmina Pazardzievska, and Julien Bourgeois. Fast attack detection using correlation and summarizing of security alerts in grid computing networks. *The Journal of Supercomputing*, 62(2) :804–827, 2012.

[SPEC01] Laura P Swiler, Cynthia Phillips, David Ellis, and Stefan Chakerian. Computer-attack graph generation tool. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, volume 2, pages 307–321. IEEE, 2001.

[STPRS15] Guillermo Suarez-Tangil, Esther Palomar, Arturo Ribagorda, and Ivan Sanz. Providing siem systems with self-adaptation. *Information Fusion*, 21 :145–158, 2015.

[SZFL13] Alireza Sadighian, Saman Taghavi Zargar, José M Fernandez, and Antoine Lemay. Semantic-based context-aware alert fusion for distributed intrusion detection systems. In *Risks and Security of Internet and Systems (CRiSIS), 2013 International Conference on*, pages 1–6. IEEE, 2013.

[TBLM11] Karim Tabia, Salem Benferhat, Pierre Leray, and Ludovic Mé. Alert correlation in intrusion detection : Combining ai-based approaches for exploiting security operators knowledge and preferences. In *Security and Artificial Intelligence (SecArt)*, 2011.

[TL00] Steven J. Templeton and Karl Levitt. A requires/provides model for computer attacks. In *Proceedings of the 2000 New Security Paradigms Workshop (NSPW'00)*, pages 31–38, September 2000.

[TL07] Karim Tabia and Philippe Leray. Approches basées sur les réseaux bayésiens pour la prédiction d'attaques sévères. In *5èmes Journées Francophones sur les Réseaux Bayésiens (JFRB2010)*, 2007.

[TVM04] Eric Totel, Bernard Vivinis, and Ludovic Mé. A Language Driven Intrusion Detection System for Event and Alert Correlation. In *Proceedings ot the 19th IFIP International Information Security Conference*, pages 209–224, Toulouse, August 2004. Kluwer Academic.

[Vaa09] Risto Vaarandi. Real-time classification of ids alerts with data mining techniques. In *IEEE MILCOM Conference*, 2009.

[Val06] Fredrik Valeur. *Real-Time Intrusion Detection Alert Correlation.* PhD thesis, UNIVERSITY OF CALIFORNIA, 2006.

[VDM+09] J. Viinikka, H. Debar, L. Mé, A. Lehikoinen, and M. Tarvainen. Processing intrusion detection alert aggregates with time series modeling. *Information Fusion*, 10(4) :312–324, 2009.

[Vig03] Giovanni Vigna. Teaching hands-on network security : Testbeds and live exercises. *Journal of Information Warfare*, 3(2) :8–25, 2003.

[VJ03] Stilianos Vidalis and Andy Jones. Using vulnerability trees for decision making in threat assessment. Technical Report CS-03-02, School of Computing, University of Glamorgan, Pontypridd, Wales, UK, 2003.

[VS01] Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In W. Lee, L. Mé, and A. Wespi, editors, *Proceedings of the Fourth International Symposium on the Recent Advances in Intrusion Detection (RAID'2001)*, number 2212 in LNCS, pages 54–68, October 2001.

[VTM05] Bernard Vivinis, Eric Totel, and Ludovic Mé. Sémantique du langage de description d'attaques adele. Technical report, Supelec, avril 2005.

[WLD06] Rayford B. Vaughn Wei Li and Yoginder S. Dandass. An approach to model network exploitations using exploitation graphs. *Simulation*, 82(6) :523–541, 2006.

[WLJ05] Lingyu Wang, Anyi Liu, and Sushil Jajodia. An efficient and unified approach to correlating, hypothesizing, and predicting intrusion alerts. In *ESORICS*, volume 3679 of *Lecture Notes in Computer Science*. Springer, 2005.

[WLJ06] Lingyu Wang, Anyi Liu, and Sushil Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer communications*, 29(15) :2917–2933, 2006.

[WOL11] Zheng Wu, Yang Ou, and Yujun Liu. A taxonomy of network and computer attacks based on responses. In *Information Technology, Computer Engineering and Management Sciences (ICM), 2011 International Conference on*, volume 1, pages 26–29. IEEE, 2011.

[WpWm11] Lv Wen-ping and Li Wei-min. Space based information system security risk evaluation based on improved attack trees. In *Third International Conference on Multimedia Information Networking and Security (MINES)*, page 480–483, November 2011.

[WPWP10] Jie Wang, Raphael C.-W. Phan, John N. Whitley, and David J. Parish. Quality of detectability (qod) and qod-aware aat-based attack detection. In *Proceedings of the 2010 International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 1–6, London, UK, November 2010.

[XCT+09] Anming Xie, Zhuhua Cai, Cong Tang, Jianbin Hu, and Zhong Chen. Evaluating network security with two-layer attack graphs. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pages 127–136. IEEE, 2009.

[Yag06] Ronald R. Yager. Owa trees and their role in security modeling using attack trees. *Information Sciences*, 176(20), 2006.

[YCM+06] Lihua Yuan, Hao Chen, Jianning Mai, Chen-Nee Chuah, Zhendong Su, and Prasant Mohapatra. Fireman : A toolkit for firewall modeling and analysis. In *Security and Privacy, 2006 IEEE Symposium on*. IEEE, 2006.

[YF07] Dong Yu and Deborah Frincke. Improving the quality of alerts and predicting intruder next goal with hidden colored petri-net. *Computer Networks*, 51(3) :632–654, 2007.

[YRS+04] Jinqiao Yu, YV Ramana Reddy, Sentil Selliah, Srinivas Kankanahalli, Sumitra Reddy, and Vijayanand Bharadwa. Trinetr : an intrusion detection alert management systems. In *Enabling Technologies : Infrastructure for Collaborative Enterprises, 2004. WET ICE 2004. 13th IEEE International Workshops on*, pages 235–240. IEEE, 2004.

[Zar14] Saman T Zargar. Ontids : A highly flexible context-aware and ontology-based alert correlation framework. In *Foundations and Practice of Security : 6th International Symposium, FPS 2013, La Rochelle, France, October 21-22, 2013, Revised Selected Papers*, volume 8352, page 161. Springer, 2014.

[ZG05] Bin Zhu and Ali A. Ghorbani. Alert correlation for extracting attack strategies. *International Journal of Network Security*, 3(3) :244–258, 2005.

[ZGHS09] Xin Zan, Feng Gao, Jiuqiang Han, and Yu Sun. A hidden markov model based framework for tracking and predicting of attack intention. In *Multimedia Information Networking and Security, 2009. MINES'09. International Conference on*, volume 2, pages 498–501. IEEE, 2009.

[ZNIR04] Y. Zhai, P. Ning, P. Iyer, and D.S. Reeves. Reasoning about complementary intrusion evidence. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 39 – 48, dec. 2004.

[ZOH11] Su Zhang, Xinming Ou, and John Homer. Effective network vulnerability assessment through model abstraction. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 17–34. Springer, 2011.

[ZU04] Urko Zurutuza and Roberto Uribeetxeberria. Intrusion detection alarm correlation : a survey. In *Proceedings of the IADAT International Conference on Telecommunications and computer Networks*, pages 1–3, 2004.

[ZY10] Shi Zhicai and Xia Yongxiang. A novel hidden markov model for detecting complicate network attacks. In *Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on*, pages 312–315. IEEE, 2010.

[ZYL09] Shangqin Zhong, Danfeng Yan, and Chen Liu. Automatic generation of host-based network attack graph. In *Computer Science and Information Engineering, 2009 WRI World Congress on*, volume 1, pages 93–98. IEEE, 2009.

[ZYZ08] Tianning Zang, Xiaochun Yun, and Yongzheng Zhang. A survey of alert fusion techniques for security incident. In *Web-Age Information Management, 2008. WAIM'08. The Ninth International Conference on*, pages 475–481. IEEE, 2008.