UNIVERSITÉ DE MONTRÉAL

DÉTECTION D'INTRUSION À L'AIDE D'UN SYSTÈME EXPERT BASÉ SUR L'ONTOLOGIE

ÉTIENNE DUCHARME
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2017

© Étienne Ducharme, 2017.

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

DÉTECTION D'INTRUSION À L'AIDE D'UN SYSTÈME EXPERT BASÉ SUR L'ONTOLOGIE

présenté par : DUCHARME Étienne
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. GAGNON Michel, Ph. D., président
M. FERNANDEZ José M., Ph. D., membre et directeur de recherche
M. GINGRAS Éric, Ph. D., membre et codirecteur de recherche
M. BARRERA David, Ph. D., membre

DÉDICACE

À quelque chose malheur est bon.

REMERCIEMENTS

Je remercie mon directeur de recherche José M. Fernandez pour m'avoir donné l'opportunité de participer à cette recherche. Son soutien et sa compréhension ont été précieux. Je remercie également mon codirecteur de recherche Éric Gingras. Par ses encouragements et le suivi qu'il m'a offert, il a grandement contribué à l'aboutissement de cette recherche. À cause de mes présences constantes au laboratoire de recherche, je remercie mes collègues qui m'ont apporté leur soutien.

Je tiens à remercier mon collègue François Labrèche et mon ami Sébastien Lorrain. Ils m'ont aidé à entretenir la passion que je possède pour le domaine de la sécurité informatique.

Ce projet n'aurait pas eu lieu sans le financement offert par MITACS, que je tiens à remercier et qui m'a permis de travailler à l'entreprise Groupe Access. Je remercie mes collègues de cette entreprise, et plus particulièrement Yves Lépine dont la générosité m'a aidé à traverser des moments ardus.

Je remercie mon acolyte de recherche Simon Malenfant-Corriveau pour sa contribution au projet. À l'aide de nos forces respectives, nous avons réussi à toucher terre dans l'océan brumeux dans lequel nous tentions de naviguer.

Je tiens à remercier mes amis qui m'ont aidé à y voir plus clair. Leurs conseils judicieux et leur écoute ont eu un impact positif sur mon développement durant ce projet. Je dirige des remerciements spéciaux à Francis Deslauriers qui m'a appuyé à plusieurs moments au cours de mes années de recherche.

J'adresse des remerciements à ma mère Céline, mon père Claude-André, ma belle-mère Ginette, ma soeur Catherine et mon beau-frère Martin. Le soutien de diverses natures qu'ils m'ont offert durant ces années d'études a été grandement apprécié. Leur implication m'a touché et a renforcé mes valeurs familiales. Je tiens à remercier ma belle-famille Paquette-Lafontaine pour leur compréhension et leur support.

Enfin, mes plus grands remerciements vont à l'endroit de ma conjointe Laurence. C'est un euphémisme de dire que cette recherche n'aurait pas vu son aboutissement sans elle. J'ai bénéficié de son appui maintes fois pour traverser les embuches que j'ai rencontrées sur mon parcours. Je ne pourrai jamais oublier la patience et la persévérance dont elle a fait preuve à mon égard.

RÉSUMÉ

Les attaques informatiques sont une réalité importante d'aujourd'hui. Leur omniprésence s'explique par le fait que les attaquants peuvent tirer parti de la complexification de l'environnement des systèmes d'information. Ceci est causé à la fois par une informatisation massive des activités, tel le stockage de données personnelles, et par l'intégration de nouvelles technologies comme les technologies sans-fil. L'appât du gain est en augmentation et les surfaces d'attaque sont plus grandes que jamais.

Les mécanismes de défense traditionnels peinent à s'adapter à cet environnement qualifié d'hétérogène à cause du large spectre de types d'information qui le composent. C'est à travers des règles que les systèmes de défense procèdent à la détection d'intrusion. Malheureusement, les langages dans lesquels sont écrites ces règles de détection possèdent plusieurs faiblesses. D'une part, leur écriture demande une grande expertise. D'autre part, les langages permettent difficilement de faire interagir des concepts de nature différente. L'humain a été impliqué dans le processus de détection d'intrusion pour pallier cette dernière faiblesse. Cependant, la détection d'intrusion est sujette aux faiblesses de l'humain, telles que la fiabilité et la performance.

Nous proposons d'informatiser le processus de détection d'intrusion avec l'utilisation d'un système expert. Ce type d'outil est un système qui reproduit le raisonnement d'un expert humain. Le système expert que nous proposons est DIOSE (Détection d'Intrusion avec l'Ontologie par un Système Expert). Il sera basé sur les ontologies, qui sont des méthodes de représentation de la connaissance qui permettent d'expliciter un concept afin qu'il soit compréhensible par une machine. La représentation de connaissance par l'ontologie permet de définir un concept pour l'utiliser dans un raisonnement. L'utilisation de base de données ontologique offre une flexibilité qui permet de passer d'une détection basée uniquement sur les événements à une détection basée sur les événements, leur contexte ainsi que les vulnérabilités. Ceci a pour but d'améliorer la détection d'attaques informatiques en faisant une corrélation des différentes informations collectées. Une représentation des connaissances par l'ontologie va également permettre d'utiliser l'abstraction pour approcher le langage des règles de détection d'intrusion vers le langage de l'expert du domaine.

Nous présenterons l'architecture de DIOSE à travers ses deux composantes. D'une part, nous décrirons la modélisation par ontologie qui permet de stocker les données collectées. Grâce au raisonnement logique, l'ontologie corrélera les informations dans le but de détecter des étapes de scénarios. Par exemple, une corrélation pourrait être faite pour inférer qu'une requête

HTTP (HyperText Transfer Protocol) a été effectuée par un administrateur réseau. Cette corrélation serait une correspondance entre l'adresse IP (Internet Protocol) de la machine qui fait la requête HTTP et l'adresse IP de la machine où l'utilisateur-administrateur est connecté. D'autre part, nous décrirons la composante qu'est la machine à état. Celle-ci permet de détecter des scénarios complets en reliant ses étapes. Dans un tel cas, les sorties d'une étape seront les entrées d'une étape subséquente.

La conception et le développement de DIOSE se basent sur la détection de trois scénarios d'attaque informatique que nous avons développés avec une expertise du domaine. Ces scénarios ont été choisis pour leur réalisme et leur complexité. Nous concevrons une solution qui permet de détecter ces trois scénarios en les modélisant dans DIOSE. Lors de cette modélisation, nous étudierons les efforts déployés pour créer une solution qui détecte ces trois scénarios. Nous nous intéresserons également à la viabilité de notre système expert dans un environnement réel d'entreprise.

Nos résultats théoriques démontrent que DIOSE permet de stocker les différents types d'information dans le but de les inclure dans les règles de corrélation. Ceci nous permet de détecter les trois scénarios d'attaque. De plus, nous pouvons abaisser l'expertise requise pour l'utilisation de notre solution en approchant le langage de règle vers le langage humain. Notre analyse des efforts à fournir pour détecter chaque scénario nous indique que ces efforts diminuent avec l'augmentation de nombre de scénarios modélisés. Finalement, nous décrivons les différentes contraintes nécessaires à l'utilisation de notre système dans un contexte réel.

ABSTRACT

Computer attacks are an important reality today. Their ubiquity is explained by the fact that attackers can take advantage of the growing complexity of the information systems environment. This is due to a massive computerization of activities, such as the storage of personal data, and the integration of new technologies, such as wireless technologies. The lure of gain is increasing and the attack surfaces are bigger than ever.

Traditional defence mechanisms are struggling to adapt to this heterogeneous environment due to the broad spectrum of information it contains. It is through detection rules that defence systems perform intrusion detection. Unfortunately, the languages in which these detection rules are written have several weaknesses. On the one hand, their writings require great expertise. On the other hand, these languages make it difficult to make concepts of different natures interact. Humans have been involved in the intrusion detection process to address this weakness. Thereby, intrusion detection is subject to human weaknesses, such as reliability and performance.

We propose to computerize the intrusion detection process with the use of an expert system. This type of tool is a system that replicates the reasoning of a human expert. The expert system that we propose is DIOSE (Détection d'Intrusion avec l'Ontologie par un Système Expert). It will be based on ontologies, which are methods of representation of knowledge that make it possible to explain a concept so that it can be understood by a machine. The uses of ontological database provide flexibility that makes it possible to move from a detection based only on events, to a detection based on both events, event contexts and vulnerabilities. This is to improve the detection of computer attacks by correlating different information collected. A representation of knowledge with the ontology will also allow to use the abstraction to bring the language of the rules of detection of intrusion closer to the language of the expert.

When describing DIOSE, we will present its two components. On the one hand, we will present an ontology model that stores the collected data. Through logical reasoning, the ontology will correlate the information for the purpose of detecting scenario steps. For example, a correlation could be made to infer that a HTTP request was made by a network administrator. This correlation would be a match between the IP address of the machine making the HTTP request and the IP address of the machine where an administrator user is logged on. On the other hand, we will describe the component that is the state machine. This allows to detect complete scenarios by linking its steps. In such a case, the outputs of

a step will be the inputs of a subsequent step.

The design and development of DIOSE is based on the detection of three attack scenarios that we have developed with our domain expertise. These scenarios have been chosen for their realism and complexity. We will design a solution that can detect those three scenarios by modelling them in DIOSE. During this modelling, we will study the efforts made to create a solution that detects these three scenarios. We will also focus on the viability of our expert system in a real business environment.

Our theoretical results show that our expert system based on the ontology developed makes it possible to store the different types of information in order to include them in the correlation rules. This allows us to detect the three attack scenarios. In addition, we can lower the expertise required to use our solution by approaching the rules language to human language. Our analysis of the efforts required to detect each scenario indicates that these efforts are decreasing as the number of modelled scenarios increases. Finally, we describe the different constraints necessary to use our system in a real context.

TABLE DES MATIÈRES

DÉDICACE iii

REMERCIEMENTS iv

RÉSUMÉ v

ABSTRACT vii

TABLE DES MATIÈRES ix

LISTE DES TABLEAUX xii

LISTE DES FIGURES xiii

LISTE DES SIGLES ET ABRÉVIATIONS xiv

LISTE DES ANNEXES xvi

CHAPITRE 1 INTRODUCTION 1
- 1.1 Solution possible 6
- 1.2 Objectif de recherche 7
- 1.3 Plan du mémoire 8

CHAPITRE 2 DÉTECTION D'INTRUSION ET ONTOLOGIES 9
- 2.1 Détection d'intrusion 9
  - 2.1.1 Solution manuelle 10
  - 2.1.2 Solution SIEM rigide 13
  - 2.1.3 Solution SIEM flexible 15
  - 2.1.4 Produits existants 16
  - 2.1.5 Rappel des limites 20
- 2.2 Les ontologies et les technologies sémantiques 20
  - 2.2.1 L'ontologie 20
  - 2.2.2 Les règles de raisonnement 23
  - 2.2.3 Les requêtes 25
  - 2.2.4 Modélisation d'une ontologie 26
- 2.3 Ontologies et corrélations en sécurité informatique 32

2.3.1 Classification . . . 32
2.3.2 Corrélation . . . 34
2.3.3 Fondements de notre recherche . . . 42

CHAPITRE 3 ARCHITECTURE DE DIOSE . . . 46
3.1 Données concrètes en entrée . . . 47
3.1.1 L'événement . . . 47
3.1.2 Le contexte . . . 48
3.1.3 La vulnérabilité . . . 49
3.1.4 Les pilotes . . . 49
3.2 Modèle ontologique . . . 52
3.2.1 Méthodologie utilisée . . . 52
3.2.2 Branche supérieure . . . 54
3.2.3 Branche inférieure . . . 58
3.2.4 Jonctions des branches . . . 60
3.2.5 L'inconsistance logique . . . 61
3.3 Machine à état . . . 63
3.3.1 Requêtes prédéfinies . . . 64
3.3.2 Machine à état déterministe . . . 65
3.4 Résumé de l'architecture . . . 70

CHAPITRE 4 EXEMPLES D'APPLICATION DE DIOSE . . . 72
4.1 Description des scénarios d'attaque . . . 72
4.1.1 Choix des scénarios . . . 73
4.1.2 Scénario n° 1 - Infection par média amovible malveillant . . . 73
4.1.3 Scénario n° 2 - Vol de données par vecteur d'entrée Web . . . 77
4.1.4 Scénario n° 3 - Rançongiciel par exploitation Heartbleed . . . 81
4.2 Application à travers les scénarios . . . 85
4.2.1 Pilotes liés aux senseurs et configurateurs . . . 85
4.2.2 Ontologie et règles . . . 86
4.2.3 Machine à état . . . 106

CHAPITRE 5 ÉVALUATION DE VIABILITÉ . . . 119
5.1 Suppositions . . . 119
5.1.1 Mode de fonctionnement des senseurs . . . 119
5.1.2 Les faux négatifs . . . 120
5.2 Flexibilité . . . 120

5.2.1 Effort de développement 121
5.2.2 Maintenabilité 123
5.3 Les patrons d'attaques 124
5.4 Processus d'affaires 126
5.4.1 Première utilisation : le fabricant 127
5.4.2 Deuxième utilisation : le client 128
5.4.3 Troisième utilisation : en production 128
5.4.4 Quatrième utilisation : les besoins de maintenance 129

CHAPITRE 6 DISCUSSION ET CONCLUSION 132
6.1 Discussion 132
6.1.1 Hypothèse n° 1 - L'expressivité des règles 132
6.1.2 Hypothèse n° 2 - La diminution de l'expertise requise 134
6.1.3 Hypothèse n° 3 - L'extensibilité du modèle 135
6.1.4 Hypothèse n° 4 - La viabilité en environnement de production 136
6.2 Conclusion 138
6.2.1 Limitations de DIOSE 139
6.2.2 Améliorations futures 140

RÉFÉRENCES 142

ANNEXES 148

LISTE DES TABLEAUX

Tableau 2.1 Niveaux d'abstraction d'une injection SQL 16
Tableau 2.2 Caractéristiques des propriétés pour R(x,y) 22
Tableau 2.3 Caractéristiques des propriétés de l'exemple de conception 30
Tableau 2.4 Règles en logique descriptive de l'exemple de conception 30
Tableau 2.5 Sortie de la requête SPARQL de l'exemple de conception 31
Tableau 3.1 Montée en abstraction - Configurateur et senseur 58
Tableau 3.2 Montée en abstraction - Règles 59
Tableau 3.3 Exemple d'inconsistance 62
Tableau 3.4 État de la requête R037 66
Tableau 3.5 Résultats de la requête R050 67
Tableau 3.6 Résultats de la requête R053 67
Tableau 3.7 Requête R053 avec et sans contrainte 69
Tableau 3.8 Montée en abstraction - Requêtes 71
Tableau 4.1 Échantillon des pilotes conçus 86
Tableau 4.2 Volumétrie de l'ontologie développée 87
Tableau 5.1 Nombre d'axiomes dans l'ontologie à la fin de chaque scénario 121
Tableau A.1 Propriétés d'objets 151
Tableau A.2 Propriétés de données 152
Tableau A.3 Disjonctions des classes 153
Tableau A.4 Règles 156
Tableau B.1 Requêtes SPARQL 160
Tableau B.2 Pilotes liés aux senseurs 177
Tableau B.3 Pilotes liés aux configurateurs 179

LISTE DES FIGURES

Figure 1.1 Logique d'un senseur avec traitement 3
Figure 2.1 Chemin des données à travers le processus de détection d'intrusion . . 10
Figure 2.2 Évaluations des SIEM selon les quadrants magiques 18
Figure 2.3 Exemple d'ontologie 24
Figure 2.4 Hiérarchie de classe 29
Figure 2.5 Relations entre les concepts fondamentaux d'attaques informatiques . 43
Figure 3.1 Montée en abstraction par DIOSE 46
Figure 3.2 Exemple de tâche d'un pilote avec un minimum de logique 50
Figure 3.3 Exemple de tâche d'un senseur comportant une logique 51
Figure 3.4 Diagramme de montée en abstraction 53
Figure 3.5 Branche supérieure du diagramme 55
Figure 3.6 Ontologie Heartbleed avant C001 dans la branche supérieure 55
Figure 3.7 Ontologie Heartbleed après C001 dans la branche supérieure 56
Figure 3.8 Ontologie Heartbleed après T043 dans la branche supérieure 57
Figure 3.9 Ontologie Heartbleed après T044 dans la branche supérieure 57
Figure 3.10 Branche inférieure du diagramme 58
Figure 3.11 Ontologie Heartbleed avant C001 et S006 dans la branche inférieure . 58
Figure 3.12 Ontologie Heartbleed après C001 et S006 dans la branche inférieure . 60
Figure 3.13 Ontologie Heartbleed après T060 dans la branche inférieure 61
Figure 3.14 Ontologie Heartbleed après T013 dans la branche inférieure 62
Figure 3.15 Machine à état générale avec R050 et R053 66
Figure 3.16 Machine à état spécifique avec R050 et R053 68
Figure 4.1 Diagramme du déroulement du scénario n° 1 74
Figure 4.2 Diagramme du déroulement du scénario n° 2 79
Figure 4.3 Diagramme du déroulement du scénario n° 3 83
Figure 4.4 Machine à état modélisant le scénario n° 1 110
Figure 4.5 Machine à état modélisant le scénario n° 2 114
Figure 4.6 Machine à état modélisant le scénario n° 3 117
Figure 5.1 Nombre d'axiomes introduits en fonction du scénario 122
Figure 6.1 Diagramme de montée en abstraction de l'exploitation Mimikatz . . . 135
Figure A.1 Hiérarchie des classes reliées au concept de vulnérabilité 148
Figure A.2 Hiérarchie des classes reliées au concept d'événement 149
Figure A.3 Hiérarchie des classes reliées au concept de contexte 150

LISTE DES SIGLES ET ABRÉVIATIONS

ATOM	Abstractions Translation Ontology Method
APT	Advanced Persistent Thread
CAML	Correlated Attack Modeling Language
CEE	Common Event Expression
CMS	Configuration Management System
CVE	Common Vulnerabilities and Exposures
DHCP	Dynamic Host Configuration Protocol
DIOSE	Détection d'Intrusion avec l'Ontologie par un Système Expert
DMZ	Demilitarized Zone - *Zone démilitarisée*
DNS	Domain Name System
DPI	Deep Packet Inspection
DoS	Denial of Service
FTP	File Transfer Protocol
HIDS	Host Intrusion Detection System
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IDMEF	Intrusion Detection Message Exchange Format
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPS	Intrusion Prevention System
IRC	Internet Relay Chat
LAN	Local Area Network
LD	Logique Descriptive
MAC	Media Access Control
NIDS	Network Intrusion detection system
NVD	National Vulnerability Database
OS	Operating System
OSI	Open Systems Interconnection
OSVDB	Open Source Vulnerability Database
OWL	Web Ontology Language
PHP	PHP : Hypertext Preprocessor
RDF	Resource Description Framework

RDFS	Resource Description Framework Schema
RDP	Remote Desktop Protocol
RFID	Radio-Frequency Identification
SCADA	Système d'acquisition et de contrôle de données
SGBDR	Systèmes de Gestion de Bases de Données Relationnelles
SIEM	Security Information and Event Management
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SQWRL	Semantic Query-Enhanced Web Rule Language
SSH	Secure Shell
SSL	Secure Sockets Layer
SWRL	Semantic Web Rule Language
TCP	Transmission Control Protocol
TI	Technologies de l'information
TLS	Transport Layer Security
UDP	User Datagram Protocol
UML	Unified Modeling Language
USB	Universal Serial Bus
USM	Unified Security Management
VBA	Visual Basic for Applications
VoIP	Voice over IP - *Voix sur IP*
VPN	Virtual Private Network - *Réseau Virtual Privé*
WAF	Web Application Firewall
WAN	Wide Area Network
XML	Extensible Markup Language
XSS	Cross-Site Scripting

LISTE DES ANNEXES

Annexe A Présentation des éléments de l'ontologie 148
Annexe B Présentation des composantes de DIOSE 160

CHAPITRE 1 INTRODUCTION

L'informatique est de plus en plus au coeur de notre société. Une grande partie des activités qui se faisaient traditionnellement « hors-ligne » est maintenant réalisée sur un appareil informatique. On pense aux transactions bancaires, à la sauvegarde d'informations privées et au réseautage social, pour ne nommer que ceux-ci. Ce changement complexifie les environnements informatiques puisque ces derniers sont de plus en plus utilisés.

Qui plus est, la constante émergence de nouvelles technologies contribue également à la complexité des environnements informatiques. Aujourd'hui, il y a l'infonuagique et l'informatique mobile, et demain sera l'ère de l'Internet des objets connectés. Ces technologies sont déployées dans la grande majorité des systèmes informatiques et contribuent à l'hétérogénéité des éléments dans ces systèmes.

Nous définissons le terme « système informatique » comme étant tout environnement qui traite l'information à l'aide d'ordinateurs. Les systèmes informatiques que nous traiterons dans cette recherche sont les systèmes informatiques conventionnels, communément appelés les systèmes des technologies de l'information (TI). Ces systèmes informatiques sont opposés aux systèmes opérationnels, par exemple les systèmes d'acquisition et de contrôle de données (SCADA) et les systèmes de contrôle du trafic aérien.

Pour illustrer l'hétérogénéité d'un environnement, prenons l'exemple du réseau d'une entreprise moyenne. Il y a plusieurs années, les machines connectées à un tel réseau étaient composées uniquement de postes de travail et de serveurs. Du fait que cet environnement était relativement homogène et statique, il était possible de déterminer le périmètre d'un environnement pour le défendre. De nos jours, cette entreprise moyenne possèderait en plus un point d'accès pour les clients sans-fil, des téléphones VoIP (Voice over IP) et un service Web de courriel hébergé sur Internet. On qualifierait cet environnement d'hétérogène à cause des éléments très différents qu'il possède. Ces changements ont eu des répercussions sur le périmètre d'environnement au point qu'il est très difficile de le définir, et par le fait même de le défendre.

Les attaques contre les systèmes informatiques ont su tirer parti de leur croissante complexité. En effet, les vecteurs d'entrée sont plus nombreux qu'antérieurement et les chemins pour arriver à un dessein malveillant se sont multipliés. Alors que l'efficacité des mesures de défense diminue lorsque le périmètre d'un environnement augmente, la surface d'attaque augmente au même rythme que ce périmètre.

Selon le rapport annuel de l'entreprise Symantec (2016), le nombre d'intrusions (*breaches*) est en constante augmentation d'année en année. Cette croissance est due entre autres à la complexité croissante des systèmes informatiques. Celle-ci facilite la tâche des attaquants, qui profitent d'une plus grande surface d'attaque que les administrateurs de réseau ont peine à défendre (Covington et Carskadden, 2013). Pour utiliser les trois paramètres d'un crime (Muehrcke et al., 2010), soit la capacité, la motivation et l'opportunité, nous décrivons cette situation par une croissance des opportunités pour les attaquants. De plus, à cause de la migration des activités vers le domaine de l'informatique, l'attaquant a de plus en plus de raisons pour procéder à ses desseins criminels puisque l'appât du gain augmente. Par exemple, de plus en plus de données personnelles, telles que les numéros d'assurance sociale ou les numéros de carte de crédit, sont stockées dans le monde virtuel ; ce qui était moins le cas il y a une dizaine d'années. On traduit donc ceci par une croissance de la motivation d'un attaquant. Ainsi, à cause des augmentations de la surface d'attaque et de l'appât du gain, l'opportunité et la motivation des cybercriminels ont augmenté les activités de ceux-ci (Sood et Enbody, 2013).

Dans un tel environnement, il est nécessaire d'avoir des technologies en place qui permettent la détection des attaques informatiques, appelées aussi intrusions informatiques. Nous définissons une intrusion informatique par tout accès non autorisé ou imprévu à un bien, que ce soit une donnée, une machine physique ou un réseau.

La détection d'intrusion dans les environnements informatiques est un domaine qui intéresse l'industrie et le domaine académique depuis que les systèmes informatiques existent.

Les premiers efforts déployés pour la détection d'intrusion ont été consacrés à la création de senseurs, qui sont des points de collecte d'événements. Un événement est quelque chose qui se produit, qui arrive (Larousse). Qu'ils soient malveillants ou non, ils sont collectés par les senseurs. Typiquement, le point de collecte d'un senseur est limité à un seul domaine d'application, c'est-à-dire que chaque senseur est concerné par un seul type d'information. Puisqu'il existe un très grand nombre de senseurs différents, nous en présentons que quelques-uns :

— Network Intrusion detection system (NIDS)
— Host Intrusion detection system (HIDS)
— Logiciel antivirus
— Pare-feu
— Journalisations de serveurs applicatifs (serveur Web, base de données, etc.)
— Journalisations de système d'exploitation
— Filtre antipourriel

Il existe deux types de senseurs : ceux sans traitement et ceux avec traitement. Dans le premier cas, il s'agit d'une catégorie de senseurs qui prend l'événement collecté tel quel et le rapporte. Par exemple, un serveur de courriel annonce la réception d'un courriel pour un certain utilisateur ; ou alors un logiciel gestionnaire de base de données rapporte qu'une donnée a été insérée dans une table spécifique. Ce type de senseur est généralement intégré dans l'application qui génère les événements.

Dans le deuxième cas, soit un senseur avec traitement, l'événement collecté est analysé par le senseur avant d'être rapporté. Ce type de senseur est généralement un module complémentaire plutôt qu'un module intégré directement au système. Par exemple, un logiciel antivirus effectue un certain traitement sur les fichiers qu'il analyse pour détecter s'ils sont malveillants ou non ; ou encore, un filtre antipourriel effectue des opérations de recherche de chaînes de caractères sur les courriels pour détecter leur malveillance. Lorsqu'un traitement effectué par un senseur avec traitement est positif, on dit que le senseur génère une alerte. Ce terme est fréquemment utilisé dans la littérature, mais dans notre cas nous ne ferons pas de différence entre la notion d'alerte et d'événement, puisque les alertes sont à la base des événements.

Les senseurs avec traitement sont caractérisés par leur méthode de détection : une détection basée sur les signatures ou une détection basée sur le comportement anormal (Axelsson, 2000). Dans le premier cas, des signatures sous forme de règles sont écrites pour détecter une spécificité. Un événement, appelé aussi alerte, est généré quand une règle est vérifiée, c'est-à-dire que la condition de la règle a été validée. La figure 1.1 démontre cette logique qui est utilisée par un grand nombre de senseurs. Par exemple, si ce type de senseur observe un téléchargement de fichier et la signature (ou empreinte) de ce fichier a déjà été identifiée malveillante, le senseur va signaler le fichier en générant une alerte.

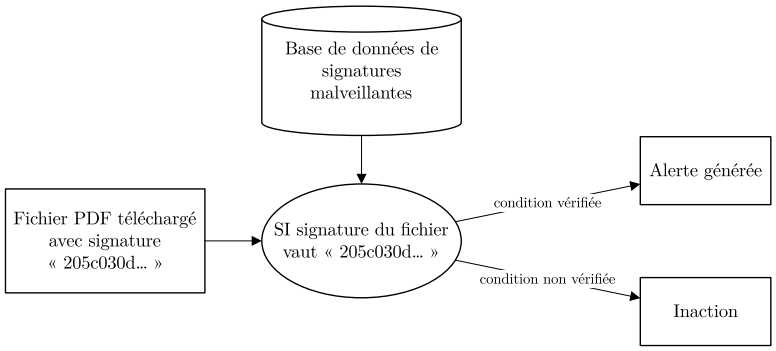

Figure 1.1 Logique d'un senseur avec traitement

Dans le deuxième cas, soit une détection basée sur le comportement anormal, un modèle de comportement normal est construit. Par la suite, la détection se fera en comparant une

activité avec ce modèle normal pour détecter à quel point les deux diffèrent. Dans le cas d'une discordance, cette activité sera classifiée malveillante puisqu'elle déroge au modèle normal.

Le déploiement de senseurs variés dans un environnement est une solution pour réduire la problématique des faux négatifs, qui sont des intrusions non détectées. Par contre, les limites de cette solution sont rapidement rencontrées lorsqu'il est nécessaire de faire des liens entre les informations collectées. On parle ici de corrélation avec les événements collectés et d'autres types d'information, tels que les données contextuelles ou les vulnérabilités informatiques. Cette corrélation permet de répondre à deux besoins :

1. La validation des événements dans le but de réduire le taux de faux positifs, qui sont des événements non malveillants. Un faux positif est essentiellement une alerte qui n'aurait pas dû être générée. Comme recensé dans la littérature (Mokarian et al., 2013), le taux élevé de faux positifs est un problème important des senseurs avec traitement. Lorsque ce taux est élevé, les vrais positifs sont noyés par les faux positifs et l'analyse des alertes est impraticable manuellement.
2. La corrélation des événements pour permettre de détecter une menace globale et plus complexe. Ainsi, il serait possible de détecter une attaque distribuée à plusieurs étapes en reliant les événements que constituent les étapes de cette attaque (Elshoush et Osman, 2011). On pourrait également faire des liens entre un événement et une information d'environnement pour enrichir l'événement et par le fait même sa détection.

Pour pouvoir effectuer une corrélation sur des informations collectées par des senseurs, il faut avoir les données centralisées et uniformisées dans un format donné. Il s'agit de l'étape d'agrégation, qui regroupe et effectue une normalisation sur les données collectées. Ces étapes forment le processus de détection d'intrusion tel qu'on l'entendra dans ce mémoire. Le processus débute avec l'étape de collecte dans laquelle des informations sont collectées. Par la suite, ces informations sont centralisées dans un format uniforme. Ceci constitue l'étape d'agrégation, également appelée l'étape de stockage. Finalement, l'étape de corrélation effectue des liens entre ces informations stockées dans le but de tirer des conclusions.

Au début des efforts de détection d'intrusion, les étapes d'agrégation et de corrélation étaient déployées par l'effort manuel d'un analyste expert. En effet, un expert observe les informations collectées par les senseurs et effectue intuitivement une agrégation de ces informations en les stockant mentalement. Par la suite, il traite ces informations pour détecter une tentative d'intrusion selon ses connaissances et son niveau d'expertise. Ceci constitue l'étape de la corrélation. La faiblesse de cette solution est liée à la présence d'un expert dans le processus. Le principal problème est à l'étape d'agrégation, où le stockage manuel fait par l'expert

atteint des limites par rapport à la quantité d'informations à stocker, puisque le volume est généralement très important. Un problème supplémentaire de l'humain dans ce processus est le manque de fiabilité du raisonnement que l'expert effectue.

Pour pallier ce problème, on a tenté d'informatiser le processus en créant des produits appelés Security Information and Event Management (SIEM). Un SIEM se définit comme étant une solution pour la gestion des informations de sécurité d'un système informatique. Les SIEM sont des logiciels qui offrent plusieurs fonctionnalités, dont la collecte, le stockage et la corrélation. Appliqués à la problématique de détection d'intrusion dans un système informatique, ils effectuent la collecte avec des senseurs, puis centralisent ces informations dans un modèle de stockage uniforme, soit généralement une base de données relationnelle. De plus, ils offrent un module de corrélation permettant d'exécuter des règles. Ils résolvent ainsi le problème de la solution précédente en ayant un stockage permanent, peu importe le volume de données à traiter.

Bien que ces SIEM prétendent faire de la corrélation, la réalité est qu'une corrélation efficace, c'est-à-dire qui arrive à des conclusions intéressantes pour des fins de détection d'intrusion, est difficilement réalisable. Les raisons de ceci sont la rigidité des règles de corrélation qu'il est possible d'écrire et l'effort de maintien de ces règles pour détecter de nouvelles menaces.

Dans la grande majorité des cas, le langage pour l'écriture de ces règles est éloigné du langage de l'expert qui l'utilise et conçoit ces règles. Il s'agit plutôt d'un langage peu flexible et proche des données brutes à corréler. Cette faiblesse s'explique par le modèle de stockage des données, qui est très statique et qui convient difficilement aux données hétérogènes de l'environnement.

De plus, une limite survient lorsqu'on doit stocker d'autres types d'information que des événements. En effet, le stockage d'informations lorsque celles-ci sont uniquement des événements ne rencontre pas de problème puisqu'il s'agit d'informations sommairement semblables, soit homogènes. Par contre, une meilleure détection doit se baser sur d'autres éléments, soit l'environnement et les vulnérabilités. Avec l'ajout de ces deux éléments, la nature des données devient encore plus fortement hétérogène et la solution devient inadéquate. En raison de la faible expressivité de ces règles de corrélation, l'humain expert est encore une fois forcé d'être impliqué dans le processus pour effectuer du travail de corrélation entre les événements, les données d'environnement et les vulnérabilités. Cette réalité est un problème de la solution des SIEM puisque l'utilisation d'un humain apporte des faiblesses de performance et de fiabilité.

1.1 Solution possible

À la lumière des problèmes précédemment exposés, nous cherchons une solution qui possède les ressources nécessaires pour stocker un grand volume de données, avec la capacité de stockage de données fortement hétérogènes et la fiabilité déterministe qu'un traitement effectué par une machine peut apporter. Cette solution devra permettre la corrélation des données à travers une interface par laquelle un utilisateur peut aisément intégrer son expertise de détection d'intrusion.

Une solution possible serait d'utiliser un modèle de représentation de la connaissance pour représenter les informations générées par les senseurs. L'ontologie est un de ces modèles qui pourrait structurer de façon plus formelle ces informations. Une ontologie est une représentation formelle d'une connaissance, utilisée dans le but qu'une machine puisse raisonner avec la connaissance modélisée. Cette solution correspondrait à la définition d'un système expert, qui est un système qui permet de reproduire les capacités de raisonnement d'un expert. La conception d'un tel système basé sur une ontologie apporterait plusieurs avantages envers les problèmes de détection d'intrusion. Un de ceux-ci est de tirer parti de la flexibilité des ontologies. En effet, ce modèle offre des avantages en étant extensible et en permettant facilement la réutilisabilité (Spyns et al., 2002). Une ontologie est un outil idéal pour effectuer une classification d'éléments hétérogènes. Il s'agit du cas du domaine actuellement à l'étude puisque les informations sont des événements à différents niveaux d'abstraction ainsi que des informations sur l'environnement qui est très varié.

Cette flexibilité de l'ontologie permettrait de passer d'une détection d'intrusion basée uniquement sur les événements à une détection basée sur un ensemble de connaissances, constitué des événements, de leur contexte et des vulnérabilités. Ces concepts composent les attaques informatiques et sont primordiaux pour leur détection. L'ontologie qui pourrait être développée modélisera ces trois concepts et permettra de les utiliser à l'étape de corrélation du processus de détection d'intrusion. Ainsi, la détection sera basée à la fois sur l'événement, le contexte et la vulnérabilité.

Également, l'étape de corrélation du processus de détection d'intrusion pourra bénéficier du concept de l'abstraction dont tire parti l'ontologie. En effet, par la classification faite avec une ontologie, il sera possible de regrouper des concepts ayant des caractéristiques communes pour les représenter à un plus haut niveau comparé au langage expert. Ainsi, on passerait d'un langage de corrélation proche des données à un langage de corrélation proche du langage utilisé par l'expert.

L'utilisation d'une ontologie pour représenter les informations utilisées pour la détection

d'intrusion dans des systèmes informatiques serait une solution qui répondrait aux lacunes des précédentes solutions décrites.

1.2 Objectif de recherche

L'objectif de cette recherche sera de vérifier l'applicabilité des technologies ontologiques pour aider au travail de détection d'intrusion. En construisant un système expert basé sur un modèle ontologique pour le domaine des TI, il sera possible de l'utiliser pour inclure des éléments tels que le contexte, les événements et les vulnérabilités dans le processus d'analyse qui a pour but la détection d'intrusion. Également, cette utilisation permettra de bénéficier de l'abstraction pour approcher les informations utilisées pour la détection vers un niveau plus abstrait, soit le langage expert. L'objectif d'appliquer ce modèle dans le domaine de la détection d'intrusion est de transférer les tâches typiquement faites par des experts à des tâches réalisables par la machine avec le modèle en question. Ceci simplifierait le processus de détection d'intrusion relatif à la corrélation entre différents éléments dans un système informatique.

Les concepts-clés modélisés dans l'ontologie sont ceux d'événements, de contexte et de vulnérabilités. Par événement, nous définirons tout ce qui se produit, par exemple la réception d'un courriel, l'envoi d'une requête HTTP ou l'insertion d'une clé USB (Universal Serial Bus). Le contexte est défini par les éléments de l'environnement, tels qu'une machine, un réseau ou un utilisateur. Finalement, une vulnérabilité désigne une faille répertoriée qui permet de nuire à la sécurité d'un système.

De cet objectif général découlent quatre hypothèses de recherche.

1. L'utilisation d'un système expert basé sur une ontologie permet d'apporter une grande expressivité aux règles de corrélation utilisées pour ladite détection. Cette expressivité permettra à l'engin de règles d'inclure dans les règles des éléments essentiels pour la détection d'intrusion informatique tels que l'événement, le contexte et la vulnérabilité. Le travail de détection de l'utilisation d'un tel système bénéficierait des avantages de l'inclusion de ces trois éléments.
2. L'utilisation d'un tel système expert dans le domaine de la détection d'intrusion permet aux règles de corrélation de passer d'un langage proche des données à un langage plus abstrait, soit près du langage expert. Cette montée en abstraction sera effectuée avec le raisonneur de l'ontologie et permettra de diminuer l'expertise requise pour procéder à la détection d'intrusion avec notre système expert.
3. Un système expert basé sur une ontologie est une solution de détection d'intrusion

qui est extensible, c'est-à-dire que ses agrandissements se font sans trop d'efforts. Ces changements peuvent être des modifications sur le modèle de données, ou des ajouts de scénarios d'attaque dans le but d'améliorer la détection.

4. Une solution basée sur une ontologie qui permet la détection d'intrusion est viable dans un environnement réel de production.

1.3 Plan du mémoire

La suite du mémoire est divisée en cinq chapitres. Le deuxième chapitre établit les notions de base pour comprendre le domaine de la détection d'intrusion dans les systèmes informatiques et le domaine des technologies ontologiques. Ce chapitre poursuivra avec la revue de littérature de certaines approches de corrélation d'informations étudiées dans le monde de la recherche.

Par la suite, nous présenterons la solution que nous proposons, soit un système expert basé sur une ontologie. Ce troisième chapitre présentera la montée en abstraction offerte par notre système en analysant les parties dont est composé notre système expert permettant la détection d'intrusion.

Dans le quatrième chapitre, nous présenterons trois scénarios d'attaque informatique que nous avons conçus. Ces scénarios sont des suites d'étapes effectuées par un acteur malveillant. Le but de notre recherche est de proposer une solution qui permettra de détecter ces scénarios, ainsi qu'éventuellement d'autres futurs scénarios. Les trois scénarios ne sont pas utilisés à des fins de validation, mais servent plutôt de cadre pour la conception de l'ontologie. Également dans ce chapitre, nous présenterons la détection de ces trois scénarios avec notre système expert. La flexibilité de l'approche sera évaluée pour appuyer le fait que notre solution est flexible pour la détection d'éventuels scénarios à modéliser.

Ensuite, le cinquième cinq étudiera le modèle proposé en tant qu'une solution complète de détection d'intrusion en tant que SIEM complet. Finalement, dans le chapitre six, nous discuterons des hypothèses de recherche et nous formulerons une conclusion.

CHAPITRE 2 DÉTECTION D'INTRUSION ET ONTOLOGIES

Dans ce chapitre, nous établirons les bases du domaine dans lequel se situe notre recherche. La première partie présentera le domaine de détection d'intrusion selon un aspect évolutif. Nous mettrons l'accent sur la détection d'intrusion telle qu'utilisée dans l'industrie. Dans la deuxième partie, nous établirons les bases des technologies reliées à l'ontologie. Nous présenterons des exemples de conception d'ontologies. Finalement, la troisième partie présentera l'état de l'art du domaine théorique de l'utilisation des ontologies, ainsi que de la détection d'intrusion avec et sans technologie ontologique.

2.1 Détection d'intrusion

La détection d'intrusion à l'échelle d'une entreprise est une problématique complexe et préoccupante depuis quelques dizaines d'années. Les intrusions informatiques (ou attaques informatiques) sont de plus en plus nombreuses à cause de l'agrandissement de l'appât du gain et de la surface d'attaque. La complexification des systèmes informatiques permet aux attaques d'être elles-mêmes plus complexes. Par exemple, elles possèdent de plus en plus d'étapes, et voient leurs portes d'entrée augmenter.

À travers les efforts déployés au fil du temps, des solutions se sont développées d'une façon itérative pour tenter de régler cette problématique. Par ces avancées technologiques, un processus du traitement des informations utilisées à des fins de détection d'intrusion a été développé. Ce processus n'ayant pas été formellement défini dans la littérature, chaque solution de détection d'intrusion possède son propre fonctionnement et diffère par ses étapes. Malgré leurs différences, il existe un tronc commun constitué de trois étapes essentielles qui sont la collecte, l'agrégation et la corrélation (Cheng et al., 2013; Di Sarno et al., 2013).

Les solutions qui effectuent ces étapes sont nommées des SIEM (Security Information and Event Management). Un SIEM est une solution centralisée pour gérer des informations pertinentes envers la sécurité d'un système (Inns, 2014). Essentiellement, il s'agit d'un outil qui regroupe une grande quantité d'informations variées utilisées à des fins de détection d'intrusion. À la base, ils ont été conçus pour regrouper les informations collectées par les senseurs positionnés sur le périmètre de sécurité d'un réseau. À cause du changement de la réalité, ce périmètre est de moins en moins délimité, ce qui complexifie la tâche des senseurs et des solutions centralisées.

Les SIEM diffèrent par leur fonctionnalité, mais ils effectuent les mêmes tâches principales,

soit la collecte, l'agrégation et la corrélation. Nous présenterons un aperçu du processus formé par ces trois étapes. Lors de l'étape de collecte, différentes informations, telles que les événements, sont recensées par des agents liés au SIEM. Par la suite, l'étape d'agrégation rassemble et structure les informations dans un modèle de stockage, tel qu'une base de données relationnelle. Une fois les informations stockées, elles peuvent être utilisées dans des règles pour la phase de corrélation. Cette phase permet de tirer des conclusions comme la détection d'une séquence déterminée d'événements (Kostrecová et Bínová, 2015). La figure 2.1 démontre le chemin des données pour un événement d'injection de requêtes SQL (Structured Query Language).

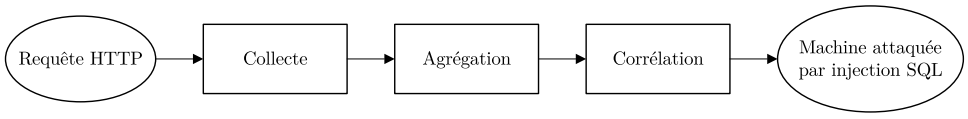

Figure 2.1 Chemin des données à travers le processus de détection d'intrusion

C'est à travers ce processus à trois étapes, collecte, agrégation et corrélation, que nous allons étudier l'histoire des solutions de détection d'intrusion.

2.1.1 Solution manuelle

Étant donné qu'il n'existait aucune infrastructure de détection d'intrusion, les experts de l'époque se sont intéressés à la première étape du processus de détection d'intrusion, c'est-à-dire la collecte d'informations. À cause de la nature d'un scénario d'attaque qui est basé sur les événements, ce type d'informations a été le seul type utilisé dans le processus de détection d'intrusion de cette première solution. Ainsi, la première avancée dans le domaine a été la création de senseurs.

Les senseurs

Ils se définissent comme des points de collecte pour des événements. La collecte faite par un senseur est généralement limitée à un domaine d'application spécifique pour sa collecte, tel que le trafic réseau, le système d'exploitation ou le navigateur.

Bien que les senseurs ne soient pas catégorisés formellement, il est important de comprendre qu'il y a deux types de senseurs : les senseurs sans traitement et les senseurs avec traitement. Dans le premier cas, ce type de senseur est le type le plus simple : il recense les événements sans aucun traitement supplémentaire. Il ne possède pas de logique ; il signale tout ce dont il est programmé pour voir. Ce type de senseur n'est généralement pas un produit en soi.

Il est plutôt lié à l'application de son domaine. Voici quelques exemples de senseurs sans traitement :

— Journalisation de serveur de base de données MySQL
— L'observateur d'événements Windows
— Journalisation du pare-feu

Dans le deuxième cas, ce sont des senseurs avec traitement qui possèdent une logique de traitement pour les types d'événements de leur domaine d'application. Les événements signalés seront ceux qui ont été analysés selon une certaine logique. Voici quelques exemples de senseurs avec traitement :

— Antivirus
— Filtre antipourriel
— Intrusion detection system (IDS)

Il existe deux logiques de traitement pour ce type de senseurs : la détection par règle ou par anomalie.

Détection par règle Les senseurs qui font la détection avec une approche par règle se basent sur une liste d'activités à signaler. Les éléments de cette liste sont typiquement appelés des signatures, et permettent d'identifier les éléments à détecter. Par exemple, dans le cas d'un antivirus qui détecte la présence de fichiers malveillants, les signatures sont des hachages cryptographiques du contenu des fichiers. S'il y a une correspondance entre la signature d'un fichier téléchargé et une signature connue, le fichier est détecté comme étant malveillant. Les signatures peuvent prendre d'autres formes, par exemple une séquence d'octets définie. Cette approche de détection est également appelée « détection par *misuse* » puisqu'elle consiste à définir les cas d'utilisation à ne pas rencontrer (Depren et al., 2005). Une telle approche se base sur la connaissance d'un expert pour définir les comportements à signaler. À cause de son approche de définition des comportements à détecter, la détection par règle fait face à un problème de volume. En effet, les bases de données de signatures de comportements malveillants ne cessent d'augmenter en volume. Malgré cette contrainte, il s'agit d'un bon compromis entre le taux de détection, le taux de faux positifs et la capacité de traitement.

Détection par anomalie Contrairement à l'approche de détection par règle, les détections par anomalie ne se basent pas directement sur la connaissance de l'expert. Cette approche de détection se déroule en deux parties. En premier lieu, un modèle de comportement normal

est construit à partir de données d'environnement lorsque tout est normal. La construction de ce modèle se fait sur des paramètres qui décrivent le comportement attendu. Par la suite, les données à analyser sont comparées au modèle construit pour déterminer si elles y correspondent. Si ce n'est pas le cas, les données sont classifiées anormales et une alerte est générée. Ce type d'approche, également appelé « détection basée sur le comportement », est moins populaire que la détection par règle puisqu'il est sujet à différentes limitations. Une principale de celles-ci survient lorsqu'il est présumé que les comportements sont normaux pendant la construction du modèle (Gadelrab, 2008). Une telle présomption est très difficile à vérifier puisqu'il peut y avoir des comportements malveillants inconnus à ce moment. Un mauvais modèle normal détruit l'efficacité du système puisqu'il s'agit de la base de ce dernier.

La création et le déploiement de différents senseurs étaient une avancée essentielle pour procéder à la détection d'intrusion à l'échelle d'une entreprise. L'étape de collecte étant effectuée par des programmes, les deux étapes subséquentes, soit l'agrégation et la corrélation, étaient effectuées par des experts du domaine. Pour l'étape d'agrégation, un expert ouvrait manuellement les sorties de différents senseurs et stockait mentalement ces informations. Par la suite, la corrélation effectuée était faite sur ces informations humainement stockées. La performance de la corrélation dépendait des connaissances de l'expert puisqu'il traversait mentalement ses règles de corrélation, pour savoir entre autres si le système avait subi une attaque informatique d'un certain type.

Pour imager la solution, l'utilisateur de celle-ci est un expert-analyste posté devant des écrans qui affichent les informations collectées par les senseurs, majoritairement sous forme de fichiers de journalisation. L'expert parcourt continuellement les écrans de données collectées pour les mémoriser. Par le fait même, les données qu'il a stockées mentalement seront analysées selon les règles de corrélation en fonction de son expertise dans le but de conclure à la détection d'une intrusion.

Désavantages

Cette solution comporte plusieurs désavantages. Le principal est d'impliquer l'humain dans le processus. La science de l'informatique essaie continuellement de maximiser l'automatisation pour diminuer le plus possible le facteur humain (Haight et Kecojevic, 2005). Dans notre cas, les désavantages de ce choix sont rencontrés avec le volume du traitement fait par un humain expert. En effet, effectuer l'étape d'agrégation avec un humain oblige ce dernier à se souvenir de toutes les informations collectées dans le but de pouvoir faire des corrélations avec celles-ci. L'expert ne pourra pas se souvenir d'un volume important d'informations. Dans le domaine de la détection d'intrusion, la quantité d'informations à traiter est très grande, ce qui est en

fait un problème majeur.

Un autre désavantage est la fiabilité d'un expert, qui n'est pas assurée. Contrairement à une machine qui a un comportement déterministe, un expert peut oublier des informations ou des règles de corrélation, ce qui rend son travail imprécis.

Par contre, il est important de noter que malgré ces désavantages fondamentaux sur le choix entre un humain et une machine pour effectuer la même tâche, l'humain a une certaine force pour le stockage de concepts hétérogènes. En effet, le cerveau excelle dans cette tâche, et c'est ce qui fait que cette solution a pu être viable.

2.1.2 Solution SIEM rigide

Face aux inconvénients de la solution précédente, l'industrie a développé les SIEM. Le besoin de développement pour ces nouveaux types de produits était de maximiser l'informatisation du processus de détection. Si pour une même tâche, un humain et une machine produisent le même résultat, il va sans dire que l'utilisation de la machine pour réaliser cette tâche est un meilleur choix (Drury et Sinclair, 1983).

Les premières versions des SIEM concernaient principalement l'étape d'agrégation du processus. En effet, le défi d'informatisation du stockage est une problématique connue en informatique et elle a été réglée avec l'implantation des bases de données. Cette technologie de stockage est mature et son utilisation permet d'informatiser la tâche d'agrégation. Ainsi, après la phase de collecte par les senseurs, les informations sont rapatriées dans la solution SIEM centralisée. Toutes ces informations sont stockées dans un modèle, qui est typiquement une base de données relationnelle.

Les SIEM se sont également attaqués au problème de la corrélation. L'étape de corrélation prend des informations en entrée, les analyse et fait des liens entre certaines de ces informations pour en créer de nouvelles. L'informatisation de cette tâche s'est concrétisée par l'implémentation d'un engin de règles. Ces règles permettaient des corrélations assez simples. Par exemple, lier deux événements par un attribut spécifique, tel que deux événements réseau qui ont la même adresse IP de destination. Un autre exemple est de lier des événements du même type qui arrivent dans une fenêtre de temps définie, tel qu'une attaque de force brute par mot de passe pour accéder à un service réseau. Ces règles étaient préenregistrées dans l'outil SIEM et elles étaient écrites dans un langage spécifique au SIEM. Chacun des outils SIEM avait sa propre façon d'implémenter ces règles, mais ils avaient tous une caractéristique commune : les règles étaient compliquées à être implémentées. Par exemple, un certain SIEM avait ses règles implémentées directement dans son code source Java. Cette complexité

apporte le problème qu'il est fort difficile pour les utilisateurs d'écrire leurs propres règles. Ils ne peuvent travailler avec ce langage de règles peu flexible et très complexe. Pour être en mesure d'écrire ces règles, il faut une très grande expertise qui est relative à chaque SIEM.

Désavantages

Ces types de SIEM ont été déployés dans l'industrie pour tenter d'améliorer la performance et de réduire le coût de maintenance du processus en diminuant l'implication de l'humain. Globalement, ces avancées ont amélioré la situation de l'agrégation, mais elles n'étaient que partielles, surtout en ce qui concerne la corrélation.

L'étape d'agrégation a été améliorée. Les problèmes de persistance et de fiabilité rencontrés dans la solution humaine précédente ont été réglés, puisque l'agrégation faite en stockant les informations dans une base de données relationnelle ne comportait pas ces limites. Par contre, il y a un avantage qu'une agrégation humaine possède et qui est une limite dans le cas d'une agrégation par base de données relationnelle : la flexibilité. Le modèle d'agrégation informatique est un type de stockage structuré, mais très rigide. Il s'agit d'une technologie intéressante pour stocker des informations d'un même type, mais lorsque les informations sont fortement hétérogènes, le stockage est difficile à réaliser.

C'est le cas du domaine de la détection d'intrusion puisqu'il y a plusieurs types variés d'information à stocker. Les relations entre les informations des événements, telles que les relations de sous-classes ou de causalité, sont difficilement implémentables. Ceci empêche de tirer parti de certains concepts de classification comme l'abstraction. Ainsi, il est très difficile de modéliser des éléments d'environnement ainsi que des vulnérabilités. Ces deux concepts, en plus du concept d'événements, sont l'ensemble des informations qui devraient être utilisées pour détecter des intrusions informatiques. Par contre, à cause des faiblesses de stockage à l'étape d'agrégation, il était très difficile de stocker ces différentes données. Seules les informations principales, soit les événements, ont été stockées pour uniformiser le stockage, mais ce choix a nui à la performance de la détection d'intrusion.

L'étape de corrélation a pu se rapprocher des besoins d'automatisation avec des règles préécrites, mais celles-ci sont écrites dans un langage complexe qui nécessite l'apport d'un expert. Qui plus est, les règles de corrélation étant liées au stockage, elles se retrouvent avec un manque de flexibilité, ce qui en fait des règles rigides et peu utilisables. Ceci amène à la situation où un expert-analyste peut exécuter ces règles, mais il doit également procéder à une corrélation manuelle en analysant les données stockées. Les règles ne sont qu'une mince couche supplémentaire pour détecter quelques cas précis.

Malgré ces faiblesses, le processus s'est amélioré en ce qui concerne l'automatisation. Une grande partie peut être effectuée sans un expert, ce qui est une avancée.

2.1.3 Solution SIEM flexible

La dernière avancée des SIEM pour améliorer le processus de détection d'intrusion concernait l'étape de corrélation. Étant donné que le manque de flexibilité de cette étape était un grand problème, des efforts ont été mis en place et une nouvelle génération de SIEM a été mise sur le marché. Le principal avantage de ceux-ci était d'offrir une interface accessible pour l'écriture de règles. Cette interface diffère d'un SIEM à un autre. Dans plusieurs cas, les interfaces pour gérer les règles de corrélation sont des langages que nous qualifions de « bas niveau », c'est-à-dire près des données brutes. Un exemple d'interface de ces SIEM est d'offrir l'utilisation du langage SQL pour interroger le modèle de stockage des données. Utiliser un langage de requêtes déjà éprouvé, tel que le SQL, est une amélioration puisque celui-ci est connu et qu'il abaisse l'expertise nécessaire pour utiliser l'interface. Dans d'autres cas, il peut également y avoir des langages créés uniquement pour le SIEM, mais avec des bases existantes telles que le Extensible Markup Language (XML). Également, pour d'autres cas, la méthode pour interagir avec les règles peut être graphique, c'est-à-dire qu'elle met à la disposition de l'utilisateur une interface graphique qui lui permet de faire des liens entre les informations. Cette approche apporte une flexibilité à l'étape de corrélation et poursuit la démarche d'informatisation du processus de détection d'intrusion.

Ainsi, ce processus démarre avec une collecte effectuée par les senseurs. Par la suite, les données sont acheminées dans une base de données relationnelle, puis sont analysées par des règles écrites par l'expert du système. Il interroge le système lui-même, ce qui permet une meilleure performance des règles puisqu'elles sont adaptées à son environnement.

Désavantages

Cette solution possède encore des faiblesses. Bien qu'une interface pour gérer les règles de corrélation soit offerte, celle-ci va toujours être liée à la méthode d'agrégation, donc partager ses problèmes. Ceci apporte une faiblesse d'abstraction du langage des règles de corrélation par rapport au langage de l'expert. Le langage des règles est proche des données. La raison de cette faiblesse est que les technologies de stockage utilisées, soit les bases de données relationnelles, ont de la difficulté à effectuer l'abstraction de concepts. L'abstraction permettrait de rapprocher le langage des règles de corrélation vers un langage proche du langage expert, plutôt que proche des données.

Cette abstraction par rapport au langage s'explique bien par un exemple. Pour celui-ci, nous représenterons une information isolée. Il ne s'agira pas d'une corrélation puisque la donnée est seule, mais l'exemple est ainsi simplifié. Comme présenté au tableau 2.1, cette information possède deux représentations. La première est proche des données, soit de l'information brute. Dans ce cas-ci, la requête concerne les détails de cette information et exige une grande compréhension du concept à détecter. La requête se formule comme suit : « Y a-t-il cette séquence exacte dans l'information brute ? ». La deuxième représentation s'éloigne du niveau des données pour se rapprocher du langage expert. La requête devient « Y a-t-il une injection SQL ? ». Les deux représentations concernent la même information, soit la requête HTTP, mais dans la deuxième représentation, la requête est beaucoup plus simple. Cette représentation près du langage expert est qualifiée d'abstraite. Cette montée en abstraction de la première à la deuxième représentation nécessite un traitement qui est difficilement faisable dans les SIEM actuels à cause des faiblesses des bases de données relationnelles qu'ils utilisent.

Tableau 2.1 Niveaux d'abstraction d'une injection SQL

Langage proche des données	Langage proche du langage expert
Y a-t-il une requête HTTP dont les données contiennent la séquence « `1' OR '1'='1` » ?	Y a-t-il une injection SQL ?

2.1.4 Produits existants

Cette section présentera les solutions existantes dans l'industrie ainsi que leurs caractéristiques. Un SIEM étant un produit d'entreprise, il s'agit d'un type de produit généralement très couteux. Nous baserons notre étude de marché sur un rapport existant fait par l'entreprise Gartner. Il s'agit d'une entreprise de recherche qui effectue des études sur différents produits dans l'industrie de la technologie. Nous utiliserons leur rapport annuel sur les SIEM les plus populaires.

Gartner et les quadrants magiques

Le rapport Gartner (2016) évalue une grande quantité de produits différents dans tous les domaines de la technologie. Malgré ces produits très variés, la comparaison est toujours effectuée selon la même méthodologie. Celle-ci consiste à les évaluer selon deux critères. Premièrement, le critère « Completeness of vision » évalue l'efficacité d'un produit à répondre

aux besoins des différents clients de l'industrie et évalue sa stratégie d'innovation. Un produit est donc évalué par ses fonctionnalités actuelles et futures, et par l'utilité de celles-ci pour les joueurs de l'industrie. Deuxièmement, le critère « Ability to Execute » évalue la performance des fonctionnalités offertes par un SIEM, par exemple en évaluant l'expérience du consommateur, la stabilité de la solution, et le plan de vente.

Une fois ces deux critères déterminés, ceux-ci sont placés sur un graphique où les axes sont les critères d'évaluation. Le graphique est scindé en quatre selon un expert et chaque quart est nommé quadrant magique. Les produits sont ainsi caractérisés en fonction du quadrant où ils se trouvent. Les noms des quatre quadrants sont : « Niche Players », qui sont les solutions qui répondent aux besoins d'un petit nombre de joueurs et qui offrent des fonctionnalités peu innovantes ; « Visionaries », qui sont les solutions qui offrent des fonctionnalités innovantes, mais qui ne possèdent pas un grand éventail de fonctionnalités pour rejoindre un grand nombre de joueurs ; « Challengers », qui sont les solutions qui possèdent des fonctionnalités de base pour aujourd'hui, mais qui n'ont pas un bon portrait des requis de l'industrie ; et « Leaders », qui sont des solutions bien établies pour le futur avec les fonctionnalités innovantes et qui ont une bonne vision du marché.

Nous présenterons quatre SIEM évalués dans le rapport Gartner (2016). Bien que ce rapport évalue plus d'une dizaine de SIEM, nous en avons choisi un dans chaque quadrant magique pour bien couvrir l'éventail des possibilités. Le graphique de ces quadrants magiques est présenté à la figure 2.2 Gartner (2016).

AlienVault

La compagnie AlienVault offre le produit AlienVault Unified Security Management (USM) (AlienVault, Inc.). Ce produit offre les fonctionnalités habituelles d'un SIEM en plus d'inclure les senseurs. De plus, il offre un module d'analyse de vulnérabilités. La grande majorité de ses composantes innovantes sont des solutions à code source ouvert. Celles-ci ont placé ce SIEM dans le quadrant « Visionnaries ». AlienVault USM centralise ces solutions et offre une méthode simplifiée pour les utiliser. Ce produit est vendu dans une solution clés en main, ce qui évite une grande charge de travail aux administrateurs du système qui achètent le produit.

Le principal désavantage de ce produit est l'intégration de sources de données non incluses. Leur ajout dans le stockage et dans les règles de corrélation est laborieux et compliqué. Il y a un manque de flexibilité pour certains types de données. Également, il est difficile d'utiliser des éléments contextuels, tels que l'utilisateur, dans des règles de corrélation.

Figure 2.2 Évaluations des SIEM selon les quadrants magiques

BlackStratus

La compagnie BlackStratus offre un outil de type SIEM qui se décline en deux composantes : LOGStorm est un module pour interagir avec les fichiers de journalisation et pour les stocker ; SIEMStorm est un module qui permet la corrélation sur les événements de sécurité (Blackstratus, Inc.). Ce SIEM en deux modules permet la gestion de réponse aux incidents lorsqu'une situation indésirable se produit. Le petit nombre de ses composantes ainsi que sa faible présence sur le marché ont placé le produit de BlackStratus dans le quadrant « Niche Players ».

Ses faiblesses sont par rapport au petit nombre de types de données supportés. À cause d'une faible flexibilité, il est difficile pour un acheteur de modifier le système pour y inclure des types de données au-delà de ceux compris initialement. Ceci limite les méthodes d'analyse telles que l'investigation du trafic réseau avec ou sans Deep packet inspection (DPI). Finalement, le langage de requête pour corrélation est très limitant puisqu'il ne permet pas d'utiliser des éléments de logique booléenne, tels que la conjonction ou la disjonction.

RSA

La compagnie de sécurité RSA Security offre le SIEM NetWitness Suite (RSA Security, Inc.). Il offre des fonctions de surveillance (*monitoring*) en temps réel, d'analyse et de gestion des alertes et il permet la gestion de réponse aux incidents. Il possède un grand nombre de fonctionnalités qui répondent bien à l'industrie d'aujourd'hui, mais ce SIEM est en retard pour les fonctionnalités dernier cri. Pour cette raison, il est classé dans le quadrant des « Challengers ». Selon le rapport Gartner, il s'agit d'un produit complet qui offre un grand ensemble de fonctionnalités. Cela fait de ce SIEM un produit très compliqué et difficilement modifiable pour l'ajuster à ses besoins. Il faut être un expert de ce produit pour pouvoir modifier ses modules, tels que les règles de corrélation.

Splunk

Le produit Splunk (Splunk, Inc.) est un SIEM qui offre des fonctions pour traiter des événements de sécurité. Il offre également son propre langage de règles de corrélation qui est assez extensible pour relier divers événements ensemble. Son grand nombre de composantes et leur intégration pour les besoins du futur ont placé ce produit dans le quadrant des « Leaders ». Selon le rapport Gartner 2016, il est un des chefs de file de l'industrie.

Sa plus grande faiblesse apparaît lorsqu'il faut inclure des éléments contextuels dans les règles de corrélation. Inclure des éléments, tels que le contexte de l'utilisateur pour un certain

événement, peut être ardu avec son langage des règles de corrélation.

2.1.5 Rappel des limites

Nous avons présenté l'évolution de l'informatisation du processus de détection d'intrusion. La dernière itération de ce processus a permis de développer des produits SIEM qui utilisaient des senseurs pour procéder à la collecte d'événements. Ceux-ci sont alors stockés dans un modèle qui consiste dans la majeure partie des cas en une base de données relationnelle. Un tel type de support de stockage ne permet pas de contenir des éléments hétérogènes comme des informations contextuelles ou de vulnérabilités informatiques. Il s'agit d'une faiblesse de l'étape d'agrégation telle qu'implémentée par ces SIEM. Par la suite, l'étape de corrélation utilise les informations précédemment stockées pour faire des liens entre celles-ci à l'aide de règles. Le langage offert par cette étape est très concret, c'est-à-dire proche des données. Il s'agit d'une faiblesse si on le compare à un langage abstrait, soit proche du langage expert. La cause de cette faiblesse est que le système de stockage inhérent à l'engin de règles, soit les bases de données relationnelles et le langage SQL, ne permet pas de bien implémenter le concept d'abstraction.

2.2 Les ontologies et les technologies sémantiques

Pour répondre aux problèmes des solutions antérieures, nous tenterons d'explorer une solution qui utilise les ontologies. Mais qu'est-ce qu'une ontologie et quelles sont les technologies qui l'utilisent ? Nous répondrons à ces questions, puis des exemples d'utilisation seront présentés.

2.2.1 L'ontologie

Une ontologie se situe dans le domaine de la représentation de la connaissance. L'ontologie est une science à laquelle s'intéressent les philosophes depuis l'antiquité. Elle se définit par la science de l'existence ou par la science qui étudie les êtres (Cimiano, 2006).

L'intérêt que l'intelligence artificielle a porté à la science de l'ontologie a fait naître un nouveau concept : l'ontologie en tant qu'objet informatique. Sa première définition a été établie par Gruber et al. (1993) : « An explicit specification of a conceptualization ». Entre d'autres mots, une ontologie est un ensemble de représentations de concepts explicitement définis. Ces représentations de concepts se traduisent par des classifications liées par des relations. La définition de l'ontologie a évolué pour ajouter les notions de consensus et de formalité : « An ontology is a formal, explicit specification of a shared conceptualization » (Studer et al., 1998). En ajoutant la notion de formalisme, la définition explicite qu'une ontologie

est formelle et sans ambiguïté. De plus, le concept de partage (*« shared »*) définit que la représentation du concept par l'ontologie doit être consensuelle, c'est-à-dire acceptée par des pairs (Guarino et al., 2009).

L'intelligence artificielle s'intéresse à l'ontologie puisque son utilisation permet aux machines et aux humains de partager la connaissance (Shotton et al., 2010). Pour transmettre la connaissance d'un concept d'un humain à un autre, nous le conceptualisons en langue naturelle, par exemple le français ou l'anglais. La difficulté est de transmettre la connaissance de ce concept à une machine. Selon l'état actuel de la technologie, les machines n'ont pas d'interpréteur pour les grammaires de langues naturelles, alors il est impossible qu'ils comprennent ce concept avec une définition sous ce format. Il est difficile pour les machines de comprendre la langue naturelle puisque ces dernières ne sont pas formelles, et leurs ambiguïtés complexifient énormément le traitement. L'ontologie est une forme de représentation de la connaissance à la fois interprétable par une machine et par un humain. Noy et al. (2001) décrivent cette caractéristique de l'ontologie : « It includes machine-interpretable definitions of basic concepts in the domain and relations among them. ».

Une ontologie est un ensemble d'entités, soient des classes et leurs instances, liées par des propriétés, appelées aussi des relations. Il est fréquent de représenter une ontologie sous forme de graphe orienté dont les noeuds sont les entités et les arêtes sont les propriétés. Un des formats de représentation d'une ontologie est RDF (Resource Description Framework) (Raimond et Schreiber, 2014). Ce format représente l'ontologie en tant qu'ensemble de triplets, qui sont une association entre deux entités par une propriété. Les éléments triplets RDF sont nommés selon ce formalisme : `(<sujet>, <prédicat>, <objet>)`. Par exemple, pour représenter le concept d'un ordinateur qui possède une adresse IP spécifique, nous aurions le triplet RDF suivant : `(machineX, possèdeAdresseIP, « 192.168.1.100 »)`.

Nous expliquerons la définition des concepts fondamentaux de l'ontologie, puis nous donnerons un exemple de développement d'une ontologie.

Les propriétés

Les liens entre les différentes entités dans une ontologie sont faits par les propriétés, également appelées relations. Il existe deux types de propriétés : les propriétés d'objets et les propriétés de types de données. Ces propriétés sont différenciées par le type d'entité qu'elles mettent en relation. Dans le premier cas, celui des objets, la propriété relie deux entités. Dans le deuxième cas, celui des types de données, la propriété relie une entité à un littéral, par exemple une chaîne de caractères, un entier ou un booléen. Une propriété possède également un domaine et une portée, qui permettent d'inférer la classe d'une instance reliée par la propriété. Le

domaine d'une propriété est la classe des sujets qui sont impliqués dans le triplet avec une propriété. La portée possède la même définition, mais par rapport à l'objet d'un triplet. Également, il est possible de définir des caractéristiques aux propriétés. Le tableau 2.2 décrit six de ces caractéristiques qui permettent d'enrichir les propriétés.

Tableau 2.2 Caractéristiques des propriétés pour R(x,y)

Nom	Description (Sánchez et al., 2012)
fonctionnelle	une seule valeur y pour chaque x
fonctionnelle inverse	une seule valeur x pour chaque y
symétrique	si x est relié à y, y est relié à x
asymétrique	si x est relié à y, y n'est pas relié à x
réflexive	ce qui est relié par R est relié à soi-même
irréflexive	ce qui est relié par R n'est jamais relié à soi-même

Lors du développement d'une ontologie, il est possible de créer des propriétés, mais généralement une partie de ces propriétés existe déjà et est stockée dans des librairies de langages existantes. RDF, RDFS (Resource Description Framework Schema) et OWL (Web Ontology Language) en sont des populaires. La première offre des relations de type pour faire des instanciations de classe, entre autres à l'aide de la propriété « rdf:type ». Les deux autres offrent des propriétés plus riches telles que « rdfs:subClassOf » pour spécifier une relation de sous-classe, ou « owl:disjointWith » pour spécifier la disjonction entre deux classes. Si les relations à représenter ne sont pas offertes par ces langages existants, il est possible de les créer. Par exemple, si notre ontologie modélise le concept d'un être humain, nous créerons la propriété « hasAge » pour stocker l'âge d'une personne.

Les entités

Les entités comprennent les instances de classe, appelées aussi les individus, et les classes.

Une classe est un ensemble d'éléments qui ont des propriétés similaires. Toutes les classes présentes dans une ontologie sont souvent reliées à au moins une autre classe par la relation de sous-classe. Chaque classe possède un ensemble de propriétés propres à elle ou obtenues par ses classes parentes. Ces propriétés la caractérisent par rapport aux autres classes. Par exemple, si une propriété définit que la classe A est équivalente à la classe B, alors les individus de la classe A sont également individus de la classe B.

Les instances sont des instanciations de classes. Elles peuvent appartenir à une ou plusieurs classes. Pour créer une instance, on peut créer un individu qui est en relation avec sa classe par la propriété « rdf:type ».

Il existe différents types de classe, mais seulement deux sont pertinents dans cette recherche : les classes nommées et les restrictions (McGuinness, 2004).

Les classes nommées Elles sont le type de classe le plus commun. On leur donne le nom du concept qu'elles représentent, par exemple la classe nommée « Ordinateur » pour classifier les instances qui appartiendront à cette classe.

Les restrictions Elles sont un type de classe plus complexe que les classes nommées. Elles possèdent une propriété qui établit une équivalence envers un ensemble d'entités. Cette relation d'équivalence permet de définir strictement la classe. Par exemple, la classe « Humain » aurait la propriété d'équivalence envers l'ensemble des individus qui ont un nom et qui parlent au moins une langue. Ainsi, ces propriétés définissent les restrictions de la classe.

Exemple de base de connaissances

Nous pouvons représenter une ontologie par un ensemble de relations représentées par des triplets sous le format suivant : `<sujet>, <prédicat>, <objet>`. Ils constituent un formalisme des ontologies qui est associé à la spécification RDF. Par exemple, nous représentons en 2.1 un serveur qui offre un service FTP (File Transfer Protocol).

$$\begin{gathered} \textit{ClasseServeur rdfs:subClassOf ClassOrdinateur} \\ \textit{serveur rdf:type ClasseServeur} \\ \textit{serviceFTP rdf:type ClasseService} \\ \textit{serveur :offreService serviceFTP} \end{gathered} \tag{2.1}$$

La représentation en graphe orienté de la figure 2.3 illustre cet ensemble de triplets.

2.2.2 Les règles de raisonnement

Nous avons décrit les composantes principales de l'ontologie, soient les classes, les propriétés et les instances. La représentation des connaissances dans ce modèle est très flexible, mais assez statique. De ce fait, les ontologies s'accompagnent de mécanismes d'inférence. Une inférence est une opération logique qui, en fonction d'un raisonnement sur des connaissances initiales, permet d'arriver à une conclusion. En parlant d'une ontologie, on inclut généralement le modèle ontologique, soit les entités et leurs propriétés, ainsi que ses règles d'inférence. Ces dernières sont interprétées par un interpréteur de règles, appelé un raisonneur, et permettent

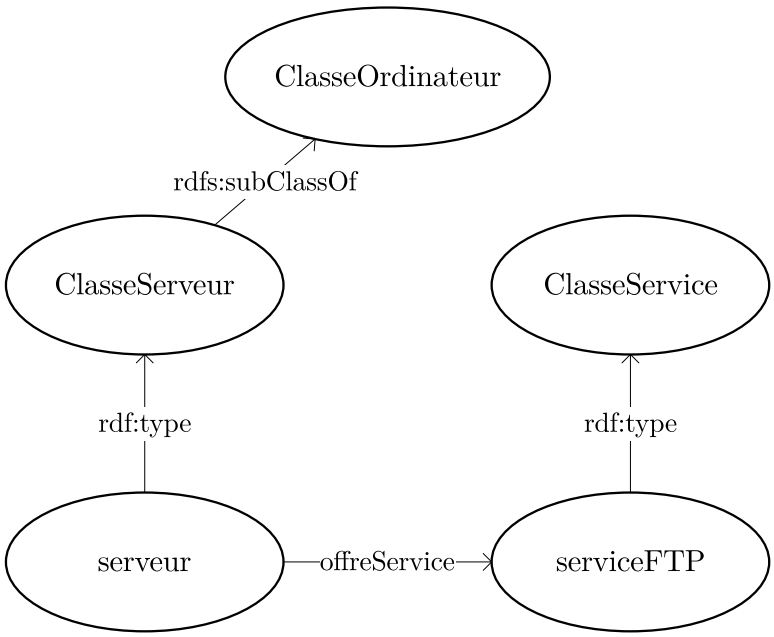

Figure 2.3 Exemple d'ontologie

d'ajouter de la connaissance au modèle en instanciant de nouvelles propriétés. Ces règles d'inférence pour enrichir le modèle peuvent être données par le modèle lui-même, ou par des règles d'inférence externes. Nous présenterons ces deux sources d'inférences.

Les inférences en logique descriptive

Ce type de source d'inférences a été mentionné dans la description des classes de type restriction. Ces inférences sont réalisées par les propriétés des entités de l'ontologie.

L'exemple le plus simple survient lors de relations de sous-classes à plusieurs niveaux. Par exemple, nous avons une hiérarchie de classes qui décrit une classe « requête HTTP ». Celle-ci est sous-classe de la classe « requête réseau », qui est elle-même sous-classe de la classe « événement ». En tant qu'humain, nous inférons naturellement qu'une requête réseau est un événement, puisqu'une requête HTTP est une requête réseau qui est elle-même est un événement. Dans l'ontologie, cette inférence est effectuée par le raisonneur. En fonction de la chaîne des relations de sous-classe, il pourra conclure que tous les individus qui appartiennent à la classe « requête HTTP » appartiennent aussi à la classe « événement ».

Un autre exemple d'inférence couramment utilisé est celui qui permet d'inférer le type d'un objet en se basant sur les relations qu'il a avec d'autres objets. Par exemple, nous pouvons spécifier dans l'ontologie que toutes les instances de la classe Machine qui offrent une instance de la classe Service font partie de la classe Serveur. Nous avons spécifié la classe Serveur par

une relation d'équivalence et enrichi l'ontologie avec une règle en logique descriptive (LD).

Les inférences par les règles SWRL

Ces inférences s'ajoutent à celles permises par la LD. Un langage différent est utilisé et il permet d'exprimer des règles plus expressives et plus complexes. Le langage que nous avons utilisé dans cette recherche est SWRL (Semantic Web Rule Language). Ces règles s'appliquent globalement sur l'ontologie et permettent de créer de nouvelles inférences lorsque leurs prédicats logiques sont vérifiés. Une règle SWRL possède un antécédent, qui est une condition, et une conséquence, qui est une inférence créée lorsque l'antécédent est validé. Par exemple, une règle logique exprimerait que si une machine offre un service et que ce service possède une vulnérabilité logicielle, alors la machine est également vulnérable. La règle SWRL traduisant cette inférence se décrit comme suit :

$$\begin{aligned} &Machine(?machine) \wedge offers(?machine, ?service) \wedge \\ &hasVulnerability(?service, ?vulnerability) \rightarrow \\ &hasVulnerability(?machine, ?vulnerability) \end{aligned} \tag{2.2}$$

2.2.3 Les requêtes

Dans la pile de technologies sémantiques reliées à l'ontologie, on retrouve également des langages de requête qui permettent d'effectuer des opérations de lecture sur les ontologies. Un de ces langages est SPARQL (SPARQL Protocol and RDF Query Language) et sera utilisé dans cette recherche. Le langage SPARQL fait partie des technologies liées au domaine du Web sémantique. Il a été standardisé par l'organisme reconnu World Wide Web Consortium (Clark et al., 2008). Une requête SPARQL interroge le modèle en récupérant des ensembles de triplets en se basant sur les correspondances entre ceux-ci. SPARQL est aux ontologies ce que SQL est aux systèmes de gestion de bases de données relationnelles (SGBDR).

```
SELECT ?user WHERE
{
        ?event rdf:type :Authentication .
        ?event :happenedOn ?machine .                    (2.3)
        ?event :hasUser ?user .
        ?user :hasAdminRightsOn ?machine
}
```

Le langage d'une requête SPARQL permet de contraindre l'ensemble de données selon les triplets demandés. Prenons l'exemple de la requête 2.3. Après les sélecteurs, l'ensemble de retour sera contraint pour ne contenir que les événements de type « Authentication ». La deuxième ligne réduira cet ensemble pour conserver dans cet ensemble seulement les individus qui sont liés par la propriété « happenedOn ». S'il est défini dans l'ontologie que la propriété « happenedOn » relie obligatoirement une instance de type Événement à une instance de type Machine, nous n'avons pas à typer l'objet « ?machine » puisque son type a été inféré par le raisonneur. Les deux dernières lignes continuent de contraindre l'ensemble de données, puis celui-ci est retourné en fonction de la sélection.

2.2.4 Modélisation d'une ontologie

La conception d'une ontologie peut s'effectuer avec le logiciel libre Protégé. Il s'agit d'un des logiciels les plus populaires pour faire la modélisation des ontologies. Pour pouvoir utiliser une ontologie en tant qu'implémentation logicielle dans un environnement de production, il faut avoir recours à d'autres logiciels, tels que le cadriciel Jena (Apache Software Foundation, 2011).

Dans le cadre de cette recherche, nous avons utilisé Protégé pour prototyper notre ontologie. Cet outil permet de concevoir une ontologie, d'y insérer des règles logiques et d'y exécuter des requêtes.

Nous présenterons un exemple de conception pour aider à la compréhension du concept d'ontologie. Pour encadrer cette modélisation d'une ontologie, nous utiliserons une méthode. Il existe plusieurs méthodes dans la littérature et chacune possède ses avantages et inconvénients. Celle que nous avons choisie pour cet exemple est une des premières méthodologies proposées et a été développée par Uschold et King (1995). Nous ne soutenons pas qu'il

s'agisse de la meilleure méthodologie de développement d'ontologies. Par contre, c'est une méthodologie assez simple et elle convient amplement pour notre démonstration.

Exemple de conception

Nous développerons une ontologie pour aider à la gestion de ressources informatiques prêtées à des employés. Les employés de cette entreprise peuvent emprunter des ordinateurs et des téléphones cellulaires selon leur statut d'employé. Nous voulons améliorer la gestion de cet inventaire pour savoir quelle ressource informatique est prêtée et à qui. Cette problématique sera traitée à travers les quatre étapes de la méthodologie de Uschold et King (1995).

Étape n^o 1 - Identification des concepts dans le domaine d'intérêt Cette étape extrait les entités et les relations de la situation à représenter. Dans notre exemple, nous ferons la liste suivante :

- employé
- poste d'un employé : développeur, directeur ou stagiaire
- nom d'un employé
- matricule d'un employé
- ressource informatique
- type de ressource
- téléphone cellulaire
- ordinateur
- adresse MAC (Media Access Control)
- posséder une ressource
- avoir une adresse MAC
- avoir un nom
- avoir un matricule
- être prêtée

Dans cette étape, il est important de garder en tête le domaine d'intérêt de notre ontologie. Celui-ci nous permettra de limiter les cas que nous aurons à modéliser. En effet, une erreur fréquente est de vouloir développer une ontologie universelle applicable dans toutes les situations (Poli, 1996). Cela complexifierait fortement le développement du modèle, et il est recommandé généralement de limiter l'explicitation des concepts selon leur domaine d'intérêt. Dans notre exemple de ressources informatiques, le domaine d'intérêt futur est une entreprise qui utilise le modèle pour recenser ses biens et garder leur trace.

Par exemple, selon ce domaine, nous établissons qu'un employé a toujours un matricule et que quelque chose qui a un matricule est nécessairement un employé. Dans le cadre de notre ontologie, cette règle est valide. Par contre, dans un autre domaine d'application, soit la gestion des étudiants et du personnel dans une école, cette règle ne serait plus valide puisque des étudiants peuvent également avoir des matricules. Ce cas est en dehors des limites du domaine que nous modélisons.

Étape n° 2 - Définition et désambiguïsation des concepts Dans cette étape, chacun des concepts trouvés à l'étape précédente sera défini. Il est important que les définitions des concepts soient le moins ambiguës possible pour éviter les erreurs. En effet, en développant l'ontologie que nous présenterons dans les prochains chapitres, nous avons fait cette erreur à plusieurs reprises. Une faible explicitation des concepts introduit des malentendus entre les développeurs d'une ontologie, puisque chacun ne réfère pas au même concept.

- employé : personne travaillant dans l'entreprise
- poste d'un employé : rôle d'un employé (directeur, développeur ou stagiaire)
- matricule d'un employé : numéro unique identifiant un employé
- ressource informatique : appareil informatique possédé par l'entreprise et qui peut être prêté
- téléphone cellulaire : appareil informatique et mobile qui permet d'effectuer des communications sans-fil
- ordinateur : machine automatique de traitement de l'information, obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques (Larousse)
- type de ressource : catégorie d'une ressource informatique, soit un téléphone cellulaire, soit un ordinateur
- adresse MAC : chaîne hexadécimale unique d'une longueur de 6 octets et qui identifie une ressource informatique
- avoir emprunté une ressource : se dit d'un employé qui a emprunté une ressource informatique
- avoir une adresse MAC : se dit d'une ressource informatique qui possède une adresse MAC
- avoir un matricule : se dit d'un employé qui possède un matricule
- être prêtée : se dit d'une ressource qui a été empruntée par un employé

Étape n° 3 - Identification des termes qui réfèrent aux concepts À travers les définitions présentées à l'étape précédente, nous avons recensé des informations inhérentes aux concepts. Ces informations seront utilisées pour mettre les classes en relation et ainsi

créer une hiérarchie de classes. Par la suite, nous définirons nos propriétés ainsi que leurs caractéristiques. Ces tâches peuvent être effectuées avec le logiciel Protégé.

Nous identifions deux concepts distincts, soit un employé et une ressource. Chacun aura sa classe et elles seront reliées par la propriété d'exclusion mutuelle. Pour la classe « employé », nous identifions trois types d'employés, que nous classifions en sous-classes : les classes « développeur », « directeur » et « stagiaire ». Pour la classe « ressource », nous identifions deux types de ressources, que nous classifions en sous-classes : les classes « ordinateur » et « téléphone cellulaire ». Ceci constitue notre hiérarchie de classes telle que présentée à la figure 2.4

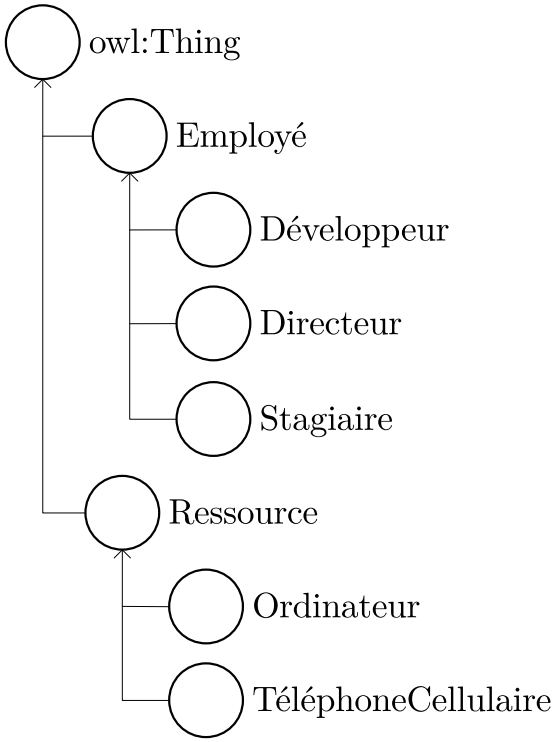

Figure 2.4 Hiérarchie de classe

Après la hiérarchie de classes, nous créons les propriétés en fonction des relations recensées aux étapes précédentes. Nous avons identifié quatre relations que nous caractériserons. D'une part, ceci s'effectue en définissant leur domaine et leur portée. D'autre part, nous définissons les caractéristiques logiques des propriétés. Les résultats de l'exemple se retrouvent au tableau 2.3. Ensuite, nous enrichissons la définition des classes en les mettant en relation avec les propriétés d'ensemble d'équivalence (*equivalent to*) et de sous-classe (*subClass of*). Nous relions ces propriétés à des expressions en LD lorsqu'il est nécessaire, par exemple pour exprimer une cardinalité. Le tableau 2.4 présente ces expressions logiques.

Tableau 2.3 Caractéristiques des propriétés de l'exemple de conception

Nom	Domaine	Portée	Caractéristiques
aEmprunté	Employé	Ressource	fonctionnelle inverse, asymétrique, irréflexive
estPrêtée	Ressource	booléen	fonctionnelle
aMatricule	Employé	entier	fonctionnelle
aAdresseMAC	Ressource	*string*	fonctionnelle

Tableau 2.4 Règles en logique descriptive de l'exemple de conception

n°	Classe	Equivalent to	SubClass of
1	Employé	`aMatricule some xsd:integer`	-
2	Directeur	-	`aEmprunté exactly 1 TéléphoneCellulaire`
3	Développeur	-	`(aEmprunté exactly 0 TéléphoneCellulaire) and (aEmprunté max 1 Ordinateur)`
4	Stagiaire	-	`aEmprunté exactly 0 Ressource`
5	Ressource	`aAdresseMAC some xsd:string`	-

Ces règles en LD permettent d'inférer des relations de type et de détecter des inconsistances logiques. Par exemple, si une instance de n'importe quel type est reliée à un entier (`xsd:integer`) par la propriété « aMatricule », par la règle en LD n° 1 nous inférerons que cette instance est de type Employé. Un autre exemple est une instance de type Directeur qui a deux fois la propriété « aEmprunté » envers deux instances de TéléphoneCellulaire. À cause de la règle n° 2, le raisonneur annoncera une inconsistance puisque toute instance de la classe Directeur est également sous-classe d'une classe définie par la restriction suivante : l'instance est en relation avec une instance de type TéléphoneCellulaire par la propriété « aEmprunté » exactement une fois.

Pour conclure l'étape de développement, nous souhaitons définir la propriété « estPrêtée ». Celle-ci relie une ressource à un booléen qui a la valeur « vrai » lorsque la ressource est reliée par la propriété « aEmprunté » à une instance de la classe Employé. À cause de la simplicité de l'exemple courant, nous aurions pu faire cette règle en LD. Faisons l'exercice de faire cette inférence en SWRL, puisque les règles SWRL ont leurs utilités dans les modélisations plus complexes. Ainsi, nous pourrions avoir de la difficulté à exprimer une règle en LD puisque ce langage ne permet pas de régler certains problèmes de modélisation. Il nous est alors possible d'exprimer la règle dans le langage SWRL et de la faire appliquer à toute l'ontologie. Cette règle est la règle 2.4.

$$aEmprunté(?employé, ?ressource) \rightarrow estPrêtée(?ressource, true) \tag{2.4}$$

Notons que nous n'avons pas à typer les variables « ?employé » et « ?ressource », puisqu'en définissant le domaine et la portée de la propriété « aEmprunté », nous avons spécifié qu'elle peut lier uniquement des instances de type « Employé » à des instances de type « Ressource ».

Étape nº 4 - Consensus Cette étape n'est qu'une formalité dans notre exemple. Une ontologie non consensuelle n'est pas une bonne ontologie, il faut donc qu'elle soit validée par un pair.

Notre ontologie est développée et nous pouvons utiliser des requêtes SPARQL pour l'interroger. Pour l'exemple actuel, nous avons peuplé notre ontologie avec des instances d'employés et de ressources reliées entre elles. Nous désirons connaître les ressources prêtées ainsi que la personne qui les a empruntées, donc nous exécutons la requête 2.5. Cette requête retourne une sortie présentée dans le tableau 2.5. La sortie est composée de trois couples d'instances, chacun constitué d'une ressource empruntée et de son emprunteur.

```
SELECT ?ressource ?emprunteur WHERE
{
        ?ressource :estPrêtée true .
        ?emprunteur :aEmprunté ?ressource
}
```
(2.5)

Tableau 2.5 Sortie de la requête SPARQL de l'exemple de conception

?ressource	?emprunteur
:macbook	:patron
:iphone	:patron
:android	:développeurWeb

2.3 Ontologies et corrélations en sécurité informatique

La représentation sous forme d'ontologie a été utilisée à multiples reprises dans le domaine de la sécurité pour des buts différents. Nous présenterons un état de l'art de ces utilisations, en commençant par les recherches qui ont présenté l'ontologie en tant qu'outil de taxonomie. Par la suite, nous présenterons les articles qui ont pour sujet la corrélation d'informations à des buts de détection d'intrusion. Ces articles n'utiliseront pas tous une ontologie pour effectuer leurs corrélations. Finalement, nous présenterons des travaux dont le but et l'approche sont très semblables aux nôtres. Parmi ceux-ci, nous exposerons le travail qui a été la base de notre recherche. Nous expliquerons pourquoi celui-ci était une bonne base, ainsi que les raisons pourquoi il était nécessaire de continuer la recherche dans ce domaine.

2.3.1 Classification

Plusieurs travaux ont été proposés pour utiliser une représentation d'information en sécurité. Une grande partie de ceux-ci effectue une classification avec une taxonomie, qui est une structure de stockage de la connaissance similaire à une ontologie. Elle est moins riche que l'ontologie puisqu'elle n'offre que certaines relations avec ses classes. Ces relations sont en majeure partie des relations de sous-classes. De plus, une taxonomie n'offre pas toutes les possibilités d'inférences logiques telles que celles permises par la logique descriptive. Nous pouvons résumer la définition d'une taxonomie par une hiérarchie de classes servant à classifier l'information.

Landwehr et al. (1994) ont étés parmi les premiers à présenter une classification reliée au domaine de la sécurité informatique. Celle-ci est basée sur les failles informatiques, et a été effectuée en recensant un grand nombre de failles existantes. La classification proposée sous forme de taxonomie classifie les failles selon trois composantes : la nature (ou genèse) de la faille, son moment d'entrée ainsi que son emplacement. Une telle classification a permis de faire les premiers pas pour le consensus d'un vocabulaire commun dans le domaine de la sécurité informatique.

Les ouvrages de Simmonds et al. (2004) ont présenté une classification d'une attaque réseau qui est effectuée en agrégeant différentes taxonomies existantes. Cette classification se fait avec les concepts de base de la sécurité informatique, soit la confidentialité, l'intégrité et la disponibilité. Elle inclut également les concepts généraux reliés aux attaques, par exemple l'acteur, la motivation et le gain. Une telle classification permet de représenter un grand nombre d'attaques. Plutôt que d'utiliser une taxonomie pour faire la classification, les auteurs ont choisi d'utiliser une ontologie. Malheureusement, la description de celle-ci est très faible,

puisqu'ils ne font qu'énumérer les différentes classes et propriétés.

Avec leurs travaux, Hansman (2005) ont proposé une taxonomie des attaques informatiques sur les réseaux et les ordinateurs. Ils prétendent que leur taxonomie est une amélioration des précédentes structures de classification présentées dans la littérature. En effet, ces chercheurs avancent que les autres taxonomies sont trop abstraites et qu'elles ne permettent pas de classifier concrètement une attaque simple. Celles-ci représentent les attaques avec des concepts abstraits, tels que l'objectif, le résultat ou les outils d'une attaque. La classification proposée prend une approche plus pragmatique pour classifier une attaque simple, telle qu'un débordement de tampon. La classification proposée possède quatre dimensions, toutes relatives à l'attaque : le vecteur d'entrée, la cible, les vulnérabilités impliquées ainsi que la charge utile, soit la finalité. Cette classification concrète est présentée du point de vue de l'attaquant et elle permet une représentation différente des autres taxonomies.

Une taxonomie ayant comme considération principale la protection contre une attaque informatique a été présentée dans les travaux de Simmons et al. (2014). Cette classification des attaques informatiques est basée sur cinq composantes d'une attaque : le vecteur d'attaque, le résultat immédiat, le moyen de remédiation, le type de cible et la conséquence du succès de cette attaque. Les auteurs ont proposé une taxonomie axée sur la défense d'un système en misant sur les aspects de mitigation et de remédiation d'attaques informatiques. En utilisant le modèle proposé, il est énoncé qu'il sera plus facile de minimiser les impacts de ces attaques.

Les ouvrages de van Heerden et al. (2012) sont allés dans une direction semblable en proposant une classification par ontologie des attaques réseau. À l'inverse des travaux qui présentent une classification d'attaque du point de vue de l'attaquant ou de la défense, cette ontologie proposée inclut les deux points de vue. Les concepts principaux de cette structure de représentation de l'information sont abstraits et permettent de représenter divers scénarios d'attaque élaborée. Voici quelques-uns des concepts modélisés : attaquant, but de l'attaque, niveau d'automatisation, conséquences, motivation, portée et cible. Cet article illustre bien l'utilisation de l'ontologie proposée puisqu'il présente les résultats sommaires des modélisations de différents scénarios d'attaque.

L'utilisation de taxonomies et d'ontologies de classification simple avait pour but d'améliorer la compréhension du domaine des attaques informatiques. Tel que nous l'avons présenté par l'état de l'art de la classification, il y a un fort couplage des concepts entre les différentes ontologies présentées. Par exemple, le concept d'attaque revient dans la majorité des classifications. Cependant, la définition de ce concept peut varier puisque chaque concepteur d'une classification peut proposer une définition différente. D'autres concepts présentent aussi cette confusion dans leur définition : actif (*gain*), acteur, finalité d'une attaque, etc. Ainsi, en rai-

son des multiples définitions pour un même concept, il y a un problème de formalisation du vocabulaire autour du concept d'une attaque informatique. Tel que l'explique Donner (2003) : « Today, far too much security terminology is vaguely defined. We find ourselves confused when we communicate with our colleagues and, worse yet, we confuse the people we're trying to serve. ». De ce fait, des travaux ont été faits pour tenter d'utiliser l'ontologie pour formaliser la représentation de ces concepts.

Fenz et Ekelhart (2009) se sont attaqués à ce problème. Ils se sont basés sur des taxonomies existantes dans le but de les uniformiser. En plus de cette uniformisation, l'ontologie qu'ils ont proposée modélise les concepts gravitant autour du concept principal de l'attaque, soit l'infrastructure d'une entreprise. La modélisation résultante s'est faite autour de trois concepts généraux : sécurité, entreprise et emplacement. Malgré les efforts de formalisation des concepts du domaine, la critique principale du modèle proposé est qu'il modélise des concepts peu applicables et très génériques (Souag et al., 2012). Par exemple, le concept d'ordinateur n'est pas représenté directement, bien que le concept d'actif le représente. Cette critique de représentation trop abstraite va dans les deux sens puisqu'une ontologie trop concrète serait critiquée comme n'étant pas assez abstraite. Les travaux proposés comportent un aspect pragmatique puisqu'ils démontrent l'utilisation de l'ontologie, par exemple avec l'exécution de requête SPARQL pour extraire de l'information concrète.

Les ouvrages de Herzog et al. (2007) ont été dans la même lignée : ils ont proposé une approche pragmatique de l'utilisation d'un modèle. L'ontologie qu'ils ont proposée se base sur les concepts de haut niveau suivants : actif, menace et contre-mesure. Malgré une base de représentation de ces concepts abstraits, la classification proposée va vers le concret pour que l'ontologie soit la plus utilisable possible. Par exemple, elle modélise les concepts suivants : carte à puce, fonction de hachage cryptographique, témoin (*cookie*), fichier de configuration, pollution de cache DNS (Domain Name System), injection de code PHP (PHP : Hypertext Preprocessor). À cause des efforts de modélisation, cette ontologie offre une représentation de concepts très abstraits et très concrets. Pour ajouter à l'aspect concret de cette ontologie, les auteurs décrivent ses utilisations avec des requêtes SPARQL. Ainsi, avec ceux-ci, il est possible d'obtenir des informations à différents niveaux de granularité.

2.3.2 Corrélation

Au-delà de tous ces travaux permettant de classifier les composantes d'une attaque, certains ont tenté d'aller plus loin par rapport à l'utilisation d'une structure de représentation de la connaissance en sécurité informatique. La taxonomie est une des structures qui permettent une classification. Les travaux que nous avons présentés ont exposé que l'utilisation

des taxonomies a été abondante puisque plusieurs concepts relatifs à la détection d'attaque ont été modélisés. Par contre, l'utilisation des ontologies n'était pas maximisée puisque cette structure offre plus de fonctionnalités que de permettre seulement la classification. Ainsi, des travaux se sont intéressés aux corrélations réalisables avec une ontologie. Nous divisons cette section sur les ouvrages relatifs à la corrélation en trois parties. Premièrement, nous décrirons des articles qui traitent du problème de normalisation des informations. Deuxièmement, les articles qui utilisent la corrélation pour réduire les alertes non valides, c'est-à-dire les faux positifs, seront présentés. Finalement, nous terminerons avec la présentation des travaux qui proposent une détection de scénarios d'attaque, principalement avec une corrélation d'événements en étapes.

Normalisation

La représentation de connaissances par l'ontologie a permis de tirer parti des capacités de corrélation de cette technologie. Cependant, pour pouvoir effectuer une corrélation, il est important d'uniformiser l'information dans un même format. Il s'agit d'une normalisation de l'information utile à la détection d'intrusion.

L'avancée la plus populaire en normalisation de ces informations a été de créer le format Intrusion Detection Message Exchange Format (IDMEF). Ce format a été créé par l'Internet Engineering Task Force (IETF) dans le but de standardiser l'interopérabilité entre les produits de détection d'intrusion (Debar et al., 2007). Le standard IDMEF offre une structure XML associée à une terminologie pour uniformiser les concepts du domaine. Bien que l'utilisation de ce standard ait aidé l'échange d'informations, il subit encore plusieurs critiques. La principale est que le vocabulaire proposé dans la structure de IDMEF est uniquement représenté d'une façon syntaxique. Ceci apporte le désavantage que des acteurs différents peuvent utiliser le même vocabulaire, mais ne pas référer au même concept (Kemmerer et Vigna, 2002). Une représentation syntaxique ne fait qu'apposer une étiquette, soit un nom, sur un concept sans le définir. Pour régler ce problème, il faut passer d'une représentation syntaxique à une représentation sémantique, dans laquelle un concept est formellement défini. Le standard d'échange IDMEF ne permet pas de représenter sémantiquement un concept.

Malgré ce standard qui est encore utilisé, d'autres efforts de normalisation d'information ont été proposés. Les travaux de Azodi et al. (2013) ont proposé un format de normalisation d'événements pour corréler ceux-ci en détection d'attaques informatiques. Les auteurs ont pris une approche concrète en partant des événements bruts, qui proviennent de fichiers de journalisation, pour arriver à une détection de scénarios d'attaque. Les événements sont représentés dans une version personnalisée du standard Common Event Expression (CEE). À

l'aide d'expressions régulières, les informations des événements sont extraites dans un format uniforme connu par le système centralisé de détection d'intrusion, soit un SIEM. L'article propose d'utiliser le langage associé au système de stockage, soit SQL, pour effectuer la corrélation afin de détecter des scénarios. Ces travaux proposent un format de représentation syntaxique des événements, ce qui ne se situe toujours pas dans le domaine sémantique. Bien qu'ils semblent pallier certaines faiblesses du format IDMEF, ces efforts laissent à désirer. Qui plus est, la corrélation avec le langage SQL est un problème que nous abordons dans notre recherche. Il est possible de bénéficier des avantages d'autres techniques de stockage pour avoir une représentation des données flexible et plus intuitive.

De Vergara et al. (2010) se sont intéressés au problème de normalisation d'événements sous un autre angle. Malgré les lacunes de IDMEF, il s'agit d'un format intéressant puisque son utilisation a démontré qu'elle améliore l'interopérabilité. Ainsi, ces auteurs ont choisi de bâtir sur une base existante en réglant les lacunes de IDMEF. Pour ce faire, ils ont créé une ontologie du standard IDMEF pour régler les ambiguïtés des concepts. Ils ont proposé une telle solution pour aider au problème de partage d'informations entre des SIEM différents. L'article prend une approche concrète en implémentant une solution de détection qui corrèle trivialement les événements avec des requêtes SPARQL. Quelques cas de corrélation menant à des détections d'attaques connues ont été présentés. Les limites de ces travaux sont reliées aux limites du format IDMEF. Ce format ne permet pas de décrire les informations riches reliées à l'alerte représentée. Ainsi, en basant l'approche sur ce format existant, leur solution proposée ne peut utiliser les informations qui ont étés perdues en utilisant le format IDMEF. Néanmois, leur approche de détection améliore l'utilisation du standard IDMEF.

Réduction d'alertes invalides

Les prochains travaux se sont intéressés à un problème important du domaine de la détection d'intrusion : la réduction de faux positifs. Comme mentionné précédemment, un faux positif est une alerte faussement générée. Leur présence apporte un coût d'opération puisqu'une corrélation est nécessaire pour comprendre qu'il s'agit d'une fausse alerte (Meyer, 2008). Il s'agit généralement d'une corrélation entre un événement, un contexte et une vulnérabilité. Il s'agit d'un problème récurrent dans le domaine de la détection d'intrusion puisqu'une grande partie des senseurs intelligents ont un taux très élevé de faux positifs, ce qui noie les vraies alertes. Ainsi, à la suite des ouvrages de classification et de normalisations d'informations relatives à la détection, nous présenterons les ouvrages de corrélation ayant pour but de réduire les faux positifs.

Massicotte et al. (2005) ont travaillé sur ce problème en proposant une corrélation entre les événements et les vulnérabilités logicielles. Celles-ci sont préalablement répertoriées dans des bases de données de vulnérabilités telles que la base de données Bugtraq ou celle de la référence Common Vulnerabilities and Exposures (CVE). Les auteurs proposent de relier l'environnement d'un événement à la vulnérabilité qu'il exploite. Ainsi, une corrélation entre ces deux éléments peut être faite pour empêcher la création d'une fausse alerte. Par exemple, une exploitation de débordement de tampon se produit sur un serveur qui possède le système d'exploitation Windows XP. Cette exploitation utilise une vulnérabilité uniquement présente dans les systèmes Windows NT ou Windows 2000. En reliant le contexte de l'événement (Windows XP) avec les conditions de la vulnérabilité exploitée (Windows NT ou Windows 2000), une fausse alerte liée à cet événement sera détectée puisque l'exploitation ne peut avoir lieu. Le cadre de corrélation proposé est modélisé en se basant sur les bases du langage UML (Unified Modeling Language). Il serait possible de faire usage de la modélisation avec ontologie, puisque leur modélisation ressemble déjà à une ontologie. Ceci permettrait d'utiliser les avantages d'une ontologie, soit la représentation d'éléments très variés, ainsi que l'utilisation de règles d'inférence.

Eschelbeck et Krieger (2003) ont proposé une méthode de validation d'alertes dans la même lignée. Pour valider une alerte, ils proposent une méthode active qui utilise un outil qui permet de détecter des vulnérabilités dans un contexte, soit un balayeur de vulnérabilités. Leur solution utilise l'IDS Snort ainsi que le balayeur de vulnérabilités QualysGuard. Les sorties de ces deux outils sont corrélées pour inférer si le contexte de l'alerte de Snort est vulnérable selon QualyGuard. Le cas échéant, il s'agit d'une alerte valide, donc un vrai positif. Ces ouvrages présentent de bons résultats pour réduire les faux positifs, mais il faut considérer que le balayeur de vulnérabilités génère également de faux positifs. Un tel cas réduit de beaucoup l'efficacité de cette solution. Cependant, leurs travaux ont contribué à démontrer l'importance de considérer le contexte d'un événement pour l'enrichir en procédant à une corrélation.

Une technique de validation d'alerte a aussi été proposée par Kruegel et Robertson (2004). Leur technique permet de déterminer si une alerte est un faux positif ou une alerte non pertinente. Ils proposent d'utiliser conjointement un balayeur de vulnérabilités, soit Nessus, dans un IDS, soit Snort. Ainsi, ce dernier peut faire une validation active d'une alerte avant de la générer. Cette validation active permet de déterminer si le contexte de la cible est un contexte où la vulnérabilité exploitée pourrait exister. Avec cette technique, ils réussissent à identifier bon nombre d'alertes en tant que faux positifs ou alertes non pertinentes. Il s'agit d'une autre démonstration que la corrélation entre des informations d'événements, de contexte et de vulnérabilités est inévitable pour enrichir la détection d'un événement.

La problématique des alertes a également été approchée par l'utilisation d'ontologies. Les travaux de Li et Tian (2009) se sont concentrés sur les alertes redondantes, ce qui contribue au problème du volume des alertes non pertinentes. Ils ont proposé l'utilisation combinée d'une ontologie et du format IDMEF pour représenter les événements. En transformant successivement les événements dans ces deux formats de normalisation, ils peuvent détecter des équivalences entre deux alertes. Le stockage de ces événements dans une ontologie d'attaques informatiques permet également de représenter le concept d'attaque. Ces représentations dans l'ontologie pourraient être utilisées dans une étape suivante pour aider à la détection. Cependant, cette proposition n'a pas été faite dans l'article et ils ne décrivent pas assez l'utilisation de l'ontologie qu'ils proposent. Bien qu'ils affirment qu'il ne s'agit pas d'une taxonomie, nous ne pouvons arriver à cette conclusion nous-mêmes puisque seul le diagramme de classes est présenté.

Corrélation d'étapes

Plusieurs recherches se sont intéressées à la corrélation des étapes d'une attaque. Certains ont réinventé la roue en développant leurs propres langages de description d'attaques, plutôt que d'utiliser des technologies existantes. Nous présenterons des articles qui ont pris ce choix, puis nous poursuivrons avec les corrélations d'étapes qui utilisent la technologie des ontologies.

Un ouvrage important dans ce domaine est celui de Cheung et al. (2003). Ces derniers ont créé un langage de corrélation d'attaques qu'ils ont nommé Correlated Attack Modeling Language (CAML). En l'utilisant, ils réussissent à détecter des attaques à plusieurs étapes. Le langage a été développé en maximisant la modularité pour faciliter la réutilisation. Ainsi, des modules représentant des étapes d'attaque sont créés. Ces modules possèdent un corps, une condition d'entrée et une sortie. À l'aide d'inférences, les modules sont reliés entre eux pour représenter une attaque. Ces travaux ont été une base importante pour notre recherche. Le cadre d'utilisation du langage CAML partage plusieurs concepts avec la solution que nous proposons : les modules réutilisables, les préconditions, et les sorties des modules.

Cuppens et Miege (2002) ont proposé d'utiliser un langage de description d'attaque créé par les auteurs. Ce langage est nommé Lambda et permet de décrire une attaque à plusieurs étapes en fonction de ses paramètres. Ceux-ci sont une précondition, c'est-à-dire une condition d'entrée, une post-condition, c'est-à-dire une sortie résultante d'une attaque, ou la combinaison d'événements qui représentent une attaque. Les auteurs illustrent qu'il est possible de procéder à une corrélation d'alertes en utilisant un cadre de corrélation lié au langage Lambda. Cette corrélation correspond à un ensemble de règles qui infèrent qu'une suite d'événements s'est produite ou qu'une alerte est un faux positif. Les corrélations sont

effectuées sur les préconditions et les post-conditions des attaques. En effet, pour deux attaques successives, la post-condition d'une étape devient la précondition de l'étape suivante. La description d'attaques à l'aide du langage Lambda est faite d'une façon modulaire, ce qui permet de représenter facilement des patrons d'attaques connues.

Ces notions d'entrée et de sortie d'une attaque ou d'un événement sont des concepts récurrents dans les ouvrages de détection d'intrusion. Ning et al. (2002) les ont nommés prérequis et conséquences d'une attaque et ils en ont fait le coeur de leurs travaux. Ainsi, ils proposent d'effectuer des corrélations entre alertes lorsque les prérequis d'une alerte correspondent aux conséquences d'une autre. Le prérequis d'une alerte est un contexte pouvant être relié à une vulnérabilité. Les corrélations sont représentées sous forme de graphe acyclique dans lequel les arêtes représentent une corrélation et les noeuds représentent des alertes. Les auteurs ont pris une approche très concrète dans la présentation de leurs travaux. Ils présentent de quelle façon la corrélation est utilisée pour détecter des attaques informatiques précises. De plus, ils ont procédé à une implémentation de leur solution avec une base de données relationnelle. Une critique que nous faisons pour ce choix d'implémentation est qu'il serait plus naturel de représenter les concepts dans une ontologie. En effet, cette structure de représentation de la connaissance se modélise trivialement en tant que graphe, à l'instar de la modélisation de leur solution proposée. Une base de données relationnelle se modélise en table, ce qui est moins naturel pour stocker une connaissance déjà sous forme de graphe.

Les approches que nous avons présentées jusqu'à présent sont déterministes, à l'opposé des approches probabilistes. L'utilisation de celles-ci a également été explorée. En utilisant des techniques probabilistes, Qin (2005) a proposé une solution pour effectuer des corrélations sur des événements. Il utilise un réseau bayésien pour modéliser les relations entre les alertes. Ces relations sont des causalités, soit des prérequis ou des conséquences d'une attaque. En catégorisant les alertes, qui sont en fait les événements, dans des fenêtres de temps précises, il propose une solution pour trouver la causalité statistique entre chacune des alertes. On appelle ce traitement une corrélation statistique. Ce modèle construit peut être utilisé pour détecter de futures attaques qui sont composées d'alertes préalablement analysées.

À l'instar de cette recherche, les travaux de Ren et al. (2010) proposent également de faire une corrélation statistique pour la détection d'attaques. Leur modèle d'apprentissage est aussi basé sur un réseau bayésien de causalité d'événements. Leur approche est concrète puisqu'ils proposent de faire cette corrélation en ligne, pour détecter les attaques lorsqu'elles se produisent. Excepté le fait qu'ils ont mis l'accent sur la détection en ligne, leur approche est très semblable à la recherche de Qin (2005), puisque les deux utilisent la corrélation statistique de causalité d'événements.

Ces corrélations statistiques présentent des avantages par rapport aux approches déterministes. Le principal avantage est qu'il n'est pas requis de préalablement connaître les scénarios d'attaque à détecter. Ceux-ci seront classés en fonction de la causalité des événements. La conséquence de cet avantage est que la corrélation statistique de causalité permet de détecter des attaques inconnues. Il s'agit d'un avantage très pertinent qui a toujours intéressé le domaine de détection d'attaques.

Le problème de corrélation d'informations à des fins de détection d'attaques a également été étudié par des approches du domaine de la théorie des graphes. Liu et al. (2008) ont proposé des corrélations basées sur des graphes d'attaque. Pour arriver à leurs fins, ils proposent une modélisation par graphe avec deux types de noeuds : une exploitation, qui est constituée de la machine source, de la machine ciblée et de la vulnérabilité exploitée ; et une condition de sécurité, par exemple la présence d'une vulnérabilité spécifique sur une machine. Les arêtes modélisent une dépendance ou une exigence, en fonction du sens de la relation. En représentant un scénario d'attaque sous cette forme, il est possible de représenter la séquentialité d'une attaque à plusieurs étapes. Un tel graphe, qui contient différentes modélisations d'attaques à plusieurs étapes, est massif. Pour cette raison, les auteurs ont mis des efforts sur la performance associée à la complexité algorithmique du parcours d'un tel graphe. Ainsi, ils conservent seulement les alertes qui correspondent au graphe d'attaque.

D'autres travaux de corrélation par correspondance de graphe ont été effectués par Liu et al. (2008). Ils proposent de préalablement modéliser les attaques sous forme de graphe pour tenter d'appliquer cette forme aux alertes à corréler. Ils ont utilisé le principe d'abstraction en étiquetant chaque alerte dans une de ces classes : « probe », « scan », « intrusion », « goal ». Ainsi, ils peuvent inférer si une suite connue d'étapes s'est déroulée.

Une modélisation de connaissance avec l'ontologie se rapproche d'une forme de graphe. La technologie de requête sur un tel modèle, soit le langage SPARQL, fait essentiellement de la reconnaissance de graphe (Pérez et al., 2006). De ce fait, nous présenterons des ouvrages qui utilisent la représentation par graphe à l'aide d'ontologies.

Li et al. (2008) ont proposé d'utiliser une ontologie pour effectuer une représentation d'attaque pouvant être utile pour une détection. La classification qu'ils ont proposée effectue la corrélation d'alertes pour procéder à la détection d'intrusion. Ils ont modélisé le concept d'attaques avec le langage OWL et proposent d'effectuer la corrélation avec des règles SWRL. Bien que leurs travaux permettent une représentation d'attaques informatiques, ils n'aident pas concrètement à la détection d'attaques. Cet aspect pragmatique permettrait d'utiliser l'ontologie comme outil de détection, et non comme outil de représentation. Ainsi, bien que leur ontologie proposée ne permette qu'une classification, ils ont proposé un concept intéres-

sant, qui est la corrélation d'événements dans le but de détecter des attaques informatiques.

Plusieurs autres recherches ont été réalisées pour utiliser l'ontologie à des fins de détection d'attaques informatiques. Abdoli et Kahani (2009) proposent une architecture distribuée pour améliorer l'effort de détection d'attaques par déni de service. La solution est composée d'agents distants, qui sont les IDS, et d'un agent maître qui centralise les informations des agents distants. Cet agent central est composé d'une ontologie d'attaque qui stocke les alertes des agents distants. Leurs travaux se sont arrêtés à cette étape, ce qui en fait une solution partielle. Nous formulons plusieurs critiques envers cet article. D'une part, la solution proposée n'est en rien distribuée : ce n'est qu'une solution client-serveur comme plusieurs autres l'ont proposée. D'autre part, l'utilisation de l'ontologie est très faible puisqu'ils l'utilisent dans une taxonomie. En n'utilisant pas les règles logiques pour décrire les concepts de l'ontologie, la solution est dans le domaine syntaxique. Ainsi, le manque de richesse de l'ontologie proposée fait que la solution n'est pas dans le domaine sémantique. Néanmoins, la problématique qu'ils tentaient d'adresser était semblable à la nôtre et le principe d'agents qui communiquent avec une ontologie centrale est un fondement intéressant.

Razzaq et al. (2014) adressent certaines faiblesses du précédent article. En effet, ils proposent d'utiliser une ontologie pour définir formellement les concepts reliés à des attaques Web. En représentant ces concepts, tels que l'entête d'une requête HTTP et le verbe HTTP utilisé, ils implémentent des règles pour détecter des irrégularités qui correspondent à des attaques. Ils ont donné des exemples concrets de scénarios d'attaque Web, tels qu'un empoisonnement de cache Web. L'article démontre concrètement de quelle façon leur solution détecte ces attaques à l'aide de règles logiques et de règles SWRL. Le fait de limiter la solution à un domaine spécifique et structuré, soit celui des requêtes HTTP, aide à la conception de la solution. Ceci permet d'obtenir une modélisation riche sans trop d'effort. Les travaux de Razzaq et al. (2014) sont parmi les premiers que nous avons rencontrés qui font une modélisation sémantique du domaine. En effet, l'utilisation d'une ontologie ne garantit pas qu'il s'agisse d'une modélisation sémantique. Pour ceci, il faut définir formellement les concepts plutôt que de leur apposer une simple étiquette. L'approche pragmatique des auteurs a permis d'expliquer concrètement comment leur solution pouvait être utilisée. La critique que nous leur adressons est par rapport au domaine de modélisation, qui est les attaques Web. Il serait intéressant de voir si le modèle est extensible dans un domaine plus large et moins bien structuré, par exemple les environnements des TI.

Une autre recherche qui présente une utilisation intéressante de l'ontologie est celle de Saad et Traore (2010). Ces auteurs ont présenté une solution qui utilise l'ontologie dans le domaine de l'investigation numérique. Bien que la détection d'intrusion et l'investigation numérique

ne semblent pas être des domaines liés, c'est bien le cas. Dans le domaine de l'investigation numérique, tel que le suggèrent Saad et Traore (2010), on cherche généralement des intrusions, ce qui revient à développer une solution de détection d'intrusion. Les deux domaines diffèrent envers l'écart de temps entre la collecte des données et leur analyse. Dans le cas de la détection d'intrusion, la différence est presque nulle puisque l'inspection se fait en direct, contrairement à l'investigation numérique. Saad et Traore (2010) proposent une ontologie d'attaques sur les réseaux informatiques. Cette modélisation comprend entre autres les concepts d'attaquant, d'exploitation, d'emplacement et de séquence d'étapes. Ces concepts sont à la fois abstraits et concrets, ce qui est un avantage pour l'utilisation de l'ontologie proposée. Les auteurs du texte énoncent souvent que leur solution de détection utilise le raisonnement pour créer des inférences. Bien qu'il soit énoncé que leur modèle utilise le raisonnement, l'utilisation de celui-ci reste à démontrer. Comme présenté, il ne s'agit que d'une classification syntaxique. Malgré cette faiblesse, nous trouvons le concept général intéressant. Les auteurs ont utilisé concrètement l'ontologie en tant que graphe orienté pour détecter des attaques informatiques.

2.3.3 Fondements de notre recherche

Nous présenterons maintenant les travaux de Sadighian (2015). Ceux-ci ont été importants pour notre recherche puisqu'ils sont nos fondements. En effet, nous avons effectué nos travaux dans le même laboratoire de recherche que celui de Sadighian (2015), mais à des périodes différentes. Nos recherches respectives ont tenté de résoudre la même problématique. Le fait que nous avons pu construire sur une base existante a aidé à l'avancement de nos travaux. Cependant, nous avons plusieurs critiques de ces travaux antérieurs. Nous présenterons la recherche sur laquelle nous nous sommes basés, puis nous énoncerons ses faiblesses dans le but d'y remédier.

Présentation des fondements

Les travaux de Sadighian (2015) présentent une solution de détection d'intrusion basée sur les ontologies. Ce choix de conception aide à faire de la corrélation d'informations pour permettre une meilleure détection des attaques et pour diminuer le taux de faux positifs. Les concepts fondamentaux de ce travail ainsi que leurs relations sont présentés à la figure 2.5.

Cette modélisation de l'interaction entre ces concepts fondamentaux est une base pour démarrer une modélisation. De celle-ci sont nées quatre ontologies, soit une pour chaque concept. Ces ontologies utilisaient abondamment les relations de sous-classe pour hiérarchiser les concepts. Voici quelques exemples de concepts présents dans chaque ontologie.

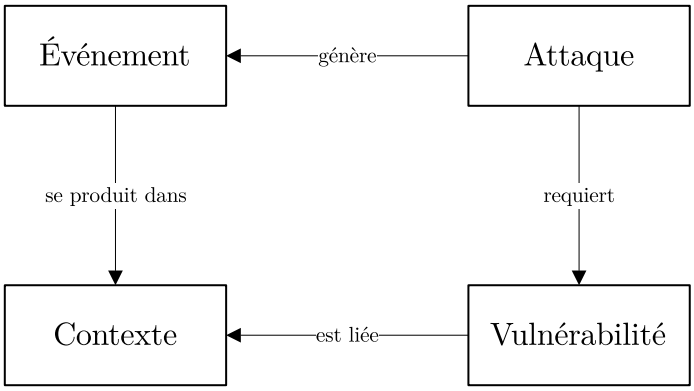

Figure 2.5 Relations entre les concepts fondamentaux d'attaques informatiques

— Événement : AntiVirusEvent, WebServer, NetEvent, WebServerEvent
— Contexte : User, Operating System (OS), Router, SubNet
— Vulnérabilité : CVE
— Attaque : Illegal Access, Social Engineering, Spyware, Denial of Service (DoS)

Utilisant les classes de ces ontologies, l'auteur propose de faire de la corrélation avec le langage SQWRL (Semantic Query-Enhanced Web Rule Language) en interrogeant l'ontologie. La thèse présente de quelle façon la solution détecte différents scénarios d'attaque. La requête 2.6 est un exemple de requête pour corréler des événements avec leurs contextes, dans le but de détecter une attaque.

$$OS(?os1) \wedge hasName(?os1, "WinXP") \wedge hasVersion(?os1, "SP1") \wedge OS(?os2) \wedge hasName(?os2, "WindowsServer") \wedge hasVersion(?os2, "2003") \wedge Application(?ap1) \wedge hasName(?ap1, "AdobeAcrobat") \wedge hasVersion(?ap1, 8) \wedge Application(?ap2) \wedge Host(?h3) \wedge hasName(?ap2, "MSSQLServer") \wedge Host(?h1) \wedge hasApp(?h1, ?ap1) \wedge Host(?h2) \wedge hasOS(?h2, ?os1) \wedge hasOS(?h3, ?os2) \wedge hasApp(?h3, ?ap2) \wedge Event(?e1) \wedge hasDest(?e1, ?h2) \wedge hasSource(?e1, ?h1) \wedge Event(?e2) \wedge hasSource(?e2, ?h2) \wedge hasDest(?e2, ?h3) \rightarrow sqwrl : select(?e1) \wedge sqwrl : select(?e2) \wedge sqwrl : count(?e1) \wedge sqwrl : count(?e2) \quad (2.6)$$

Ainsi, un retour de la requête 2.6 indique que l'étape est détectée, et la requête qui correspond à une étape subséquente dans le scénario peut être exécutée.

Améliorations possibles

Nous notons des critiques des travaux sur lesquels nous devions construire. Notre première critique est qu'il y a un vide important entre les modèles développés et leur utilisation. En effet, la thèse présente les ontologies que nous avons décrites ci-haut. Ces ontologies possèdent bon nombre de classes qui ne sont jamais utilisées et dont l'utilisation demeure un mystère. Par exemple, la classe « Objective » qui fait partie de l'ontologie d'attaque n'est jamais utilisée. Nous pouvons déduire qu'elle est utilisée pour représenter ses sous-classes, mais il n'est jamais décrit comment et surtout dans quelle fin elle est utilisée. Entre la présentation de la classification et l'utilisation concrète d'une ontologie, il manque une étape. L'absence de description de celle-ci est une faiblesse qui est présente dans un grand nombre de travaux de ce domaine. Il est facile d'utiliser l'ontologie d'une façon simpliste qui ne bénéficie pas de ses principales forces, soit la définition formelle des concepts et le raisonnement logique sur ceux-ci.

La deuxième critique que nous tirons des travaux de Sadighian (2015) est liée à l'utilisation de l'ontologie. Entre une utilisation syntaxique et une utilisation sémantique, nous affirmons que l'utilisation présentée dans cette recherche est syntaxique. Une telle utilisation a été faite par plusieurs auteurs que nous avons cités précédemment. Il s'agit d'utiliser l'ontologie purement en tant que graphe orienté et de ne pas bénéficier des capacités de définition de concept. Une approche syntaxique est utilisée en apposant une étiquette arbitraire sur un concept. À l'opposé, une approche sémantique définit un concept au-delà de son étiquette. Il peut être défini formellement par un ensemble de clauses logiques qui relient les concepts entre eux, par exemple avec des relations de sous-classe. L'abstraction est un exemple concret d'une représentation sémantique d'un concept. Tel qu'on peut le voir dans la requête 2.6, elle ne bénéficie pas du concept d'abstraction. Les différentes variables de la requête (hôte, application, événement, etc.) ont été stockées dans une ontologie puis elles ont été récupérées sans qu'elles ne soient transformées d'aucune façon. Il s'agit surtout d'une taxonomie dans laquelle les classes abstraites ont à peine été utilisées. L'abstraction a tout de même été mentionnée à plusieurs reprises et quelques exemples de requêtes qui en bénéficient ont été présentés. Cependant, pour démontrer l'utilité d'une ontologie, il aurait été pertinent de bénéficier abondamment de l'abstraction et de faire cette présentation. Sans une représentation sémantique, l'ontologie n'est qu'un graphe orienté qui stocke des données.

La troisième critique que nous adressons concerne les scénarios présentés pour démontrer l'utilisation de ce graphe. Au meilleur de notre expertise, nous évaluons que ces scénarios sont simplistes, voire bancals. Les attaques modernes ont évolué au-delà du scénario classique d'exploitation de vulnérabilités pour avoir les privilèges sur une machine. Le contexte actuel

est différent : les attaques sont plus sophistiquées et elles bénéficient de portes d'entrée très variées. Un des scénarios modélisés dans les travaux de Sadighian (2015) a été recensé dans le jeu de données « DARPA 2000 » (Zissman, 2000). Dans un domaine aussi changeant que l'informatique, il est naturel qu'un jeu de données vieux de 17 ans ne soit plus d'actualité.

La quatrième critique est envers l'utilisation concrète de la solution. Peu d'explications sont données sur l'instanciation des informations stockées dans l'ontologie. Il est décrit que les senseurs, par exemple les IDS, sont interfacés avec l'ontologie par un pilote (*driver*). Cependant, une fois l'instance d'un événement créée, l'événement doit être lié à son contexte. Par exemple, une partie du contexte d'une requête HTTP est le site Web qu'elle destine et la machine source qui l'a générée. Nous remarquons que l'événement est éventuellement lié à son contexte, mais rien n'est expliqué sur la façon de le faire, excepté manuellement. Une solution de détection d'intrusion basée sur une ontologie doit améliorer toutes les étapes du processus et nous trouvons que cette étape est manquante.

Nous avons recensé ces critiques pour mieux construire notre solution. Les faiblesses de la solution dirigeront notre recherche, puisque nous tenterons de pallier certaines de celles-ci.

CHAPITRE 3 ARCHITECTURE DE DIOSE

Nous proposons DIOSE (Détection d'Intrusion avec l'Ontologie par un Système Expert), qui est un système expert basé sur une ontologie en tant que méthode de stockage. L'utilisation de DIOSE, qui est également un SIEM, permet de corréler l'information brute pour la transformer dans un format plus près du langage humain. Ce système expert est utilisé pour détecter des scénarios d'attaque informatique qui représentent des intrusions.

Ce chapitre présente l'architecture de la solution que nous proposons. Nous procéderons à la description en utilisant le chemin des informations qui transigent dans DIOSE.

La figure 3.1 représente le parcours des informations à travers notre solution proposée. Tout d'abord, nous avons les données brutes en aval de la solution, suivi du traitement proposé par DIOSE. Ce traitement se décompose en deux parties, soit l'ontologie et la machine à état. Par la suite, nous avons la sortie résultante de notre solution constituée de concepts plus près du langage de l'expert.

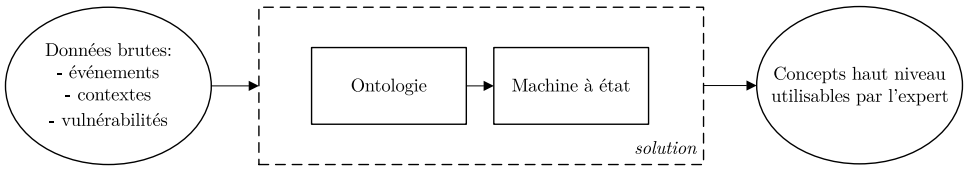

Figure 3.1 Montée en abstraction par DIOSE

Le système expert proposé permet une montée d'abstraction en ayant en entrée des informations concrètes ou brutes, et en sortie des informations abstraites plus faciles à utiliser pour un utilisateur.

Prenons la peine de dissiper les confusions autour du concept d'abstraction. L'abstraction de différentes informations n'est pas évaluable sans un point de comparaison. Dans notre cas, il s'agit du langage de l'expert. Notre but est de traduire, ou d'abstraire, une information loin du langage expert vers un langage plus près de celui qu'utilise l'humain. Une telle abstraction est courante en informatique. Un exemple est celui des langages de programmation. Au début de l'ère de l'informatique moderne, le langage assembleur est né. Ce langage était quelque peu cryptique et une poignée d'experts pouvait l'utiliser. Par contre, il était une amélioration si on le compare à une pseudo-programmation à l'aide de codes d'opération (*opcode*) compréhensibles par la machine. Cette amélioration est une abstraction vers le langage de l'expert, soit un développeur. La popularité de l'informatique a permis à d'autres

langages de plus en plus abstraits d'être développés, tel que le C++ qui est plus abstrait que l'assembleur. Finalement furent créés des langages, tels que Python et C#, encore plus abstraits qui empruntent un bon nombre de mots au langage qu'utilise l'expert, soit la langue naturelle. On dit que ces langages sont plus abstraits que l'assembleur par rapport à un utilisateur-expert, qui est un développeur dans cet exemple.

Ainsi, nous avons des informations en entrée représentées à la première étape. Celles-ci sont de différents types, que nous présenterons. Ces données entrent dans DIOSE et passent par la première étape qui est l'ontologie. Par la classification et l'engin de règle qu'offre ce modèle de stockage, nous pouvons effectuer des traitements qui permettent une montée en abstraction. Par la suite, les résultats de ces traitements sont traités par un module constitué de machines à état déterministes. Le but de cette étape est de détecter des scénarios d'attaque. Chaque étape d'un scénario représente un état qui possède des entrées et des sorties. Les données modélisées par les états de la machine à état sont utilisables par l'expert-utilisateur du système.

3.1 Données concrètes en entrée

Notre solution permet de stocker toutes les informations pertinentes à la détection d'intrusion. Nous avons séparé ces informations en trois concepts généraux, qui représentent les types de données nécessaires pour effectuer de la corrélation. Il s'agit des concepts d'événement, de contexte et de vulnérabilité. Les définitions de chacun de ces concepts seront présentées.

3.1.1 L'événement

Dans le cadre de notre recherche, nous définissons ce concept comme étant quelque chose généré par un senseur et qui possède une estampille temporelle. Comme exemple d'événement, nous pouvons penser à des actions humaines telles que l'insertion d'une clé USB ou la génération d'une requête HTTP servant à effectuer une injection SQL. Il faut également inclure les actions initiées par les machines, telles que l'attribution d'une adresse IP à une nouvelle machine, ou l'ouverture d'un port réseau par le pare-feu. En raison de notre besoin, soit la détection de scénarios, il va de soi que les événements sont au coeur de la solution.

À cause de la nature hétérogène de l'environnement des senseurs, certaines informations collectées ne représentent pas le même niveau d'abstraction par rapport au langage de l'expert. Prenons par exemple le cas d'un événement qui se fait inspecter par deux senseurs différents. Cet événement est une requête HTTP dont le corps contient une charge utile pour l'exploitation d'une faille d'injection SQL. Dans un premier cas, la requête est collectée par

le senseur Web Filter qui journalise les requêtes HTTP. Ce senseur ne s'intéresse qu'au fait qu'il s'agisse d'une requête HTTP. Il n'analysera pas le contenu de celle-ci. Dans un second cas, la requête HTTP est collectée par Snort, le NIDS du réseau interne. En plus de la classifier en tant que requête HTTP, ce senseur fera un traitement, soit du DPI, pour analyser le corps de la requête. À la suite de celui-ci, il la classifiera en tant qu'injection SQL. Ainsi, le même événement n'a pas été classifié de la même façon par deux senseurs différents. Il est représenté selon deux modélisations qui ne se situent pas au même niveau d'abstraction. La classification de la requête en tant qu'injection SQL fait d'elle un élément plus près du langage humain, à l'inverse de la requête HTTP dont le contenu n'a pas été inspecté.

3.1.2 Le contexte

Ce concept, tel qu'utilisé à des fins de détection d'intrusion, est défini comme un ensemble de propriétés d'un événement. Selon le dictionnaire Larousse, la définition du contexte relié à l'événement est : « Ensemble des circonstances dans lesquelles se produit un événement, se situe une action ». C'est de cette définition dont nous nous sommes inspirés pour établir notre définition. Ainsi, un sous-ensemble de relations, dans lequel un événement est, forme son contexte. Bien que cette définition mette le concept d'événement au coeur de la définition du contexte, il s'agit d'un cas attendu puisque les deux concepts sont indissociables ; le contexte ne peut aller sans l'événement. Contrairement aux concepts d'événements et de vulnérabilités, le contexte n'est pas modélisé en tant que classe dans l'ontologie. Il représente plutôt un ensemble de relations dans lequel est impliqué l'événement qui s'y rattache.

Nous introduisons également le concept de configurateurs qui sont les capteurs d'informations pour les données du contexte. Le configurateur est au contexte ce que le senseur est à l'événement. Nous prenons le soin de différencier les logiciels requis pour collecter les données en fonction de leur type, soit les événements ou les contextes. La raison de ceci est que les informations d'événements et de contexte n'ont pas la même dynamique. En effet, les senseurs sont très dynamiques, c'est-à-dire qu'ils collectent sans cesse des informations changeantes. Dans le cas des données d'environnement, elles sont plutôt statiques, c'est-à-dire qu'elles changent moins. Par exemple, le configurateur responsable d'instancier les utilisateurs d'une entreprise va s'exécuter une seule fois. Il peut devoir s'exécuter à nouveau s'il y a un nouvel employé, mais il va s'agir d'une exécution ponctuelle. Pour poursuivre le parallèle, on dira que les senseurs sont des exécutions continues. À des fins de recherche dans la littérature, il est important de noter que nous avons inventé le terme « configurateurs » ainsi que sa délimitation avec les senseurs. La raison de ceci est que nous proposons une approche innovante qui comprend un autre type d'informations que celui des événements. Certains environnements

d'entreprise utilisent un Configuration Management System (CMS) pour stocker des éléments contextuels. Nous ne considérons pas ces cas puisque leur utilisation est moins répandue et nous pouvons réimplémenter leurs fonctionnalités sans trop d'effort.

3.1.3 La vulnérabilité

Une vulnérabilité se définit comme étant une faiblesse dans un système dont un attaquant peut tirer parti pour nuire à la confidentialité, l'intégrité ou la disponibilité de ce système. Cette définition ne se limite pas aux vulnérabilités logicielles. Elle inclut par exemple les vulnérabilités humaines, telles que l'ingénierie sociale, et les vulnérabilités électroniques, telles que celles liées à la technologie RFID (Radio-Frequency Identification). Ce sont uniquement ces derniers types de vulnérabilité qui seront modélisés dans l'ontologie. La totalité des vulnérabilités utilisées dans cette ontologie sera prise dans des bases de données existantes telles que la National Vulnerability Database (NVD) ou l'Open Source Vulnerability Database (OSVDB).

Nous n'avons pas eu le besoin de préciser un logiciel collecteur propre aux vulnérabilités telles que le sont les senseurs pour les événements et les configurateurs pour le contexte. La raison de ceci est qu'une vulnérabilité représente un contexte spécifique. Prenons l'exemple de la vulnérabilité avec la référence CVE « CVE-2016-0034 ». L'exploitation de cette vulnérabilité a permis l'existence de plusieurs campagnes de logiciels malveillants en 2016. Elle concerne une faille dans le logiciel Microsoft Silverlight utilisé abondamment par le fureteur Internet Explorer. Tout logiciel Silverlight dont la version est entre 5.0 et 5.1.41212.0 est vulnérable à cette vulnérabilité. Ainsi, pour la représenter, nous n'avons besoin que d'informations contextuelles, telles que le logiciel installé et sa version.

3.1.4 Les pilotes

Dans la couche des données brutes, nous retrouvons également des logiciels de type pilote (ou *driver*). Ceux-ci ont la tâche de traduire l'information collectée par les senseurs et les configurateurs dans un format compréhensible par l'ontologie.

Par exemple, un senseur NIDS génère un événement de type « balayage de port » provenant de l'adresse IP 10.1.1.20 vers l'adresse 10.1.1.30. Le pilote s'interfacera avec ce senseur et il traduira l'information selon la structure de l'ontologie dans laquelle l'information va être acheminée. Dans cet exemple, le pilote crée une instance de la classe « balayage de port » avec les relations « adresse IP source » et « adresse IP destination » ayant respectivement les valeurs « 10.1.1.20 » et « 10.1.1.30 ».

Le pilote exposé dans l'exemple précédent est spécifique au senseur dans l'exemple, soit un NIDS. Cela démontre l'envergure du travail de l'étape de collecte de DIOSE. En effet, il y a un grand nombre de domaines d'application différents des senseurs, et un grand nombre de senseurs dans chacun de ces domaines d'applications.

Durant la conception de DIOSE, nous avons pris un choix qui a permis de minimiser la présence de logique dans les pilotes. Ce choix a été fait dans une perspective de déploiement dans un environnement de production dans lequel les concepteurs de pilotes n'ont pas à connaître les fonctionnements précis de l'ontologie. Le plus de logique possible a été intégrée dans l'ontologie pour minimiser les efforts de développement des pilotes. Nous pouvons reprendre l'exemple précédent et présenter deux cas dans lesquels la présence de logique dans le pilote varie. Dans un premier cas, le pilote possède un minimum de logique, comme dans l'exemple précédent qui consistait en un balayage de ports sur une machine. Dans un tel cas, le pilote possède peu de connaissances sur la modélisation. Il est nécessaire que le pilote connaisse la hiérarchie de classes pour spécifier que l'instance qu'il crée, représentant un balayage de ports, est de type « balayage de ports ». Également, il doit également connaître le nom des propriétés qui représentent des relations d'adresses IP source et destination. La figure 3.2 illustre l'instance créée par ce pilote.

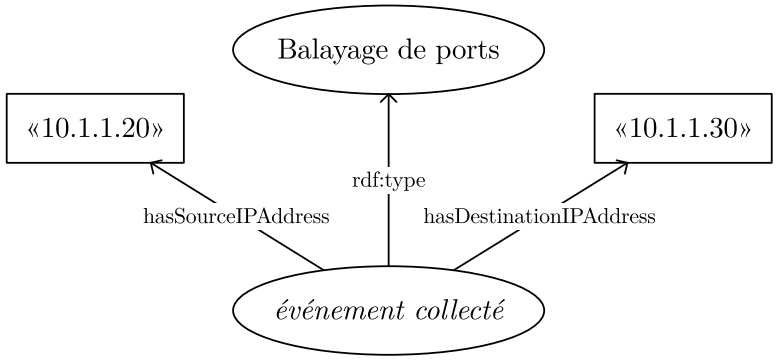

Figure 3.2 Exemple de tâche d'un pilote avec un minimum de logique

Dans un deuxième cas où la logique présente dans le pilote n'est pas minimisée, celui-ci doit connaître plus d'informations sur la modélisation. Il doit connaître les mêmes éléments que le cas présenté précédemment. Il doit également obtenir les individus de classes qui sont liés à l'événement qu'il génère. Dans le cas de l'exemple, ces individus sont la machine source et la machine cible. La connaissance de ces instances est un travail compliqué pour un pilote et elle empêche la conception modulaire souvent prônée en génie logiciel. La figure 3.3 illustre l'instance créée par le pilote.

Il est important de noter que dans les deux cas, l'information inférée par l'ontologie à la fin

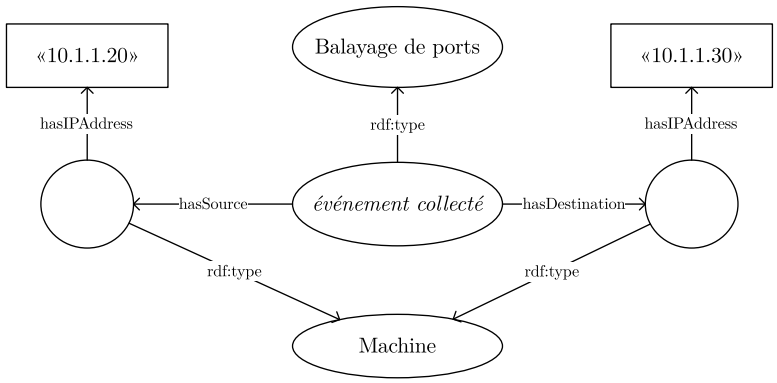

Figure 3.3 Exemple de tâche d'un senseur comportant une logique

de son raisonnement sera la même. La question est plutôt par rapport à implémenter cette logique de raisonnement dans le pilote ou dans l'ontologie.

Cette minimisation de logique a été effectuée principalement pour les pilotes reliés aux senseurs. Dans le cas des pilotes liés aux configurateurs, nous n'avons pas pu offrir un même effort de minimisation des connaissances de l'ontologie dans le pilote. La raison de ceci est que les données d'environnement sont fortement reliées entre eux et qu'une connaissance de l'ontologie est nécessaire pour éviter les problèmes, tel que d'avoir à créer des instances en double. Nous sommes à l'aise avec ce choix de conception puisque les senseurs et les configurateurs n'ont pas les mêmes concepteurs. Dans le premier cas, il s'agit d'un fabricant indépendant qui n'est pas lié du tout à DIOSE. Les senseurs de détection existent déjà et sont souvent très complexes. Il n'y a aucune raison de les réimplémenter pour y ajouter notre contrainte de connaissance du modèle. Dans le deuxième cas, les configurateurs pour les données d'environnement n'existent à peu près pas. Également, leurs logiques de collecte sont assez simples, ce qui fait que nous pouvons nous permettre de les créer nous-mêmes et y ajouter les contraintes de connaissance du modèle.

Au-delà de cette problématique de logique dans le pilote, il est important de noter une importante contrainte des données brutes. À cause du différent type des senseurs, soit avec et sans traitement, les événements ne sont pas au même niveau d'abstraction. En d'autres mots, des senseurs avec traitement ont classifié au préalable certains événements, tandis que d'autres événements représentent directement la donnée brute. Le fait d'avoir des événements à différents niveaux d'abstraction n'est pas en soit un problème puisque le modèle que nous proposons d'utiliser, soit l'ontologie, est une bonne solution pour stocker des informations de différents niveaux d'abstraction. L'hétérogénéité des données, que ce soit par rapport à leur domaine d'application ou leur niveau d'abstraction, n'est pas un problème en utilisant l'ontologie.

3.2 Modèle ontologique

Cette composante est l'ontologie et elle représente une de nos contributions les plus importantes. Une fois que les informations sont transmises par un pilote à partir d'un senseur ou d'un configurateur, les données sont stockées sous forme d'instances de classe dans une ontologie préalablement développée. Par la suite, le travail du raisonneur s'exécute et si les conditions des règles logiques présentes dans l'ontologie sont validées, des inférences sont effectuées. Essentiellement, ces inférences ajoutent des relations dans l'ontologie pour que les informations représentées soient plus riches. Par notre définition du contexte, les inférences faites par les règles logiques enrichissent le contexte.

Pour illustrer nos propos, nous poursuivons l'exemple de l'étape précédente. Dans celui-ci, l'événement de type « balayage de port » possède la propriété « adresse IP destination » valant « 10.1.1.30 ». Le modèle possèderait une règle logique qui équivaudrait à : si un événement a une propriété « adresse IP destination » qui vaut X et qu'une machine a une propriété « adresse IP » qui vaut X, l'événement est relié à la machine par la relation « destination ».

Nous proposons d'utiliser la logique de traitement qui crée des inférences pour effectuer une montée des concepts d'un niveau d'abstraction à un autre plus élevé. Cette logique de traitement sera faite soit par des règles en LD, soit par des règles SWRL ou soit par un algorithme dans un langage de programmation. Le choix de la technologie pour effectuer le traitement est lié aux restrictions des technologies. Par exemple, certains types de traitements arithmétiques sont difficilement réalisables en LD, alors il vaut mieux utiliser un langage de programmation pour les implémenter.

3.2.1 Méthodologie utilisée

Durant notre recherche, nous avons développé une méthodologie pour faire la conception d'une ontologie. La méthodologie se nomme ATOM (Abstractions Translation Ontology Method) et elle a été développée en collaboration avec Simon Malenfant-Corriveau. Elle ne sera pas détaillée en profondeur puisqu'il ne s'agit pas du sujet principal de cette recherche. Nous présenterons tout de même les artéfacts que son utilisation a produits. Un de ceux-ci est un diagramme d'abstraction de concepts. Le cas de figure représenté à la figure 3.4 sera présenté.

Nous avons créé notre terminologie des différents éléments dans le diagramme présenté. Il se lit selon le sens des flèches du diagramme, soit de droite à gauche. Ce sens désigne celui de la montée en abstraction. Ainsi, plus les concepts dans les boîtes sont vers la droite, plus ils sont près des données brutes. Plus ils sont vers la gauche, plus ils sont abstraits par rapport au langage d'un expert. Nous dénombrons trois types de concepts dans le diagramme : les boîtes

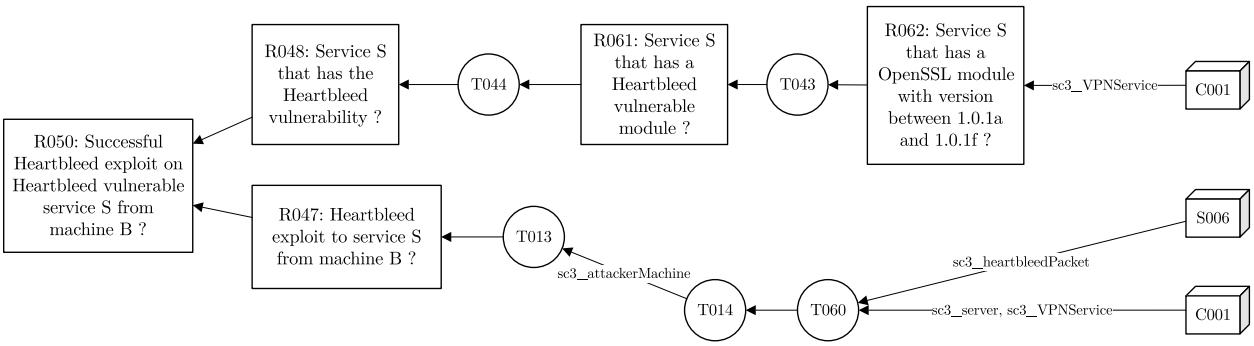

Figure 3.4 Diagramme de montée en abstraction

à trois dimensions, les cercles et les boîtes à deux dimensions. Chacun de ces concepts possède un index qui réfère à sa description dans la documentation. Nous décrirons ces concepts ainsi que leurs utilités.

Premièrement, nous avons les boîtes à trois dimensions (tout à droite), qui représentent la couche des données brutes. Il s'agit des pilotes liés aux senseurs et configurateurs. Notre diagramme contient le senseur S006 et le configurateur C001 (à deux reprises). Deuxièmement, nous avons les cercles qui sont les règles de transformation pour effectuer des inférences. Celles-ci peuvent être une règle en LD, une règle SWRL ou un algorithme en langage de programmation. Cette énumération de format de règles n'est pas exhaustive, puisque les règles peuvent être sous n'importe quel format. Par contre, les formats énumérés sont ceux que nous avons utilisés pour ce projet. Les règles de transformation dans le diagramme de l'exemple sont T044, T043, T060, T013 et T014. Troisièmement, nous avons les boîtes à deux dimensions qui représentent des requêtes SPARQL. Dans le diagramme, ce concept nous aide à son développement puisqu'il documente l'état où est rendue la donnée dans son abstraction. Le nombre de ces boîtes est arbitraire puisqu'il ne s'agit pas d'un concept directement lié à l'ontologie. Nous l'utilisons, entre autres, comme aide au développement. Dans notre cas, nous en dénombrons cinq : R050, R048, R047, R062 et R061. Ce choix est arbitraire et nous avons suivi notre intuition pour les créer lors du développement. En plus d'être un outil d'aide au développement, les requêtes sont les résultats utilisables. Elles ont une importance majeure lors de la réutilisabilité de DIOSE. Ceci sera présenté en détail dans la présentation de la deuxième partie de notre solution, soit la machine à état.

Notre diagramme possède une orientation par les flèches qu'il contient. En plus de représenter deux niveaux d'abstraction, il démontre aussi une notion de dépendances entre les concepts. En effet, pour utiliser une requête dénotée dans le diagramme, tous les concepts

(représentés par les formes) qui sont liés par une flèche à cette requête et dont la flèche part de ceux-ci doivent être présents dans l'ontologie. Par concepts, nous faisons référence aux règles de transformation, aux senseurs et aux configurateurs. Du fait que les requêtes ne sont rien de concret dans l'ontologie, celles-ci ne sont pas considérées comme un élément de dépendance. Par exemple, dans le diagramme à la figure 3.4, pour pouvoir utiliser la requête référencée R061, il est nécessaire que T043 soit dans l'ontologie. Celle-ci possède à son tour une dépendance envers le configurateur C001, ce qui conclut la hiérarchie de dépendances pour R061. Ainsi, cette requête nécessite la présence dans l'ontologie de la règle T043 et du configurateur C001. Cette description de dépendances par graphe orienté est très utile à des fins de modularité puisque nous savons quelle partie de la solution est utilisée à quel endroit. Il est important de noter que lorsque deux flèches aboutissent au même concept, il s'agit d'une opération logique de conjonction. En d'autres mots, il s'agit d'un « ET » entre les deux flèches qui représentent les dépendances.

La dernière partie du diagramme à expliquer est la présence des noms sur les flèches. Ces étiquettes représentent des instances créées dans l'ontologie. Le nom qu'ils possèdent est lié à une instance du même nom créé dans l'ontologie utilisé pendant le développement. En les écrivant sur les flèches, cela nous permet de documenter les types des instances créés par la forme du diagramme. Par exemple, « sc3_vpnService » est une instance de type VPNService créée par le configurateur C001. Notons que le préfixe du nom des instances, tel que « sc3 » dans cet exemple, est le numéro du scénario dans lequel l'instance est utilisée dans la modélisation que nous présenterons au prochain chapitre.

Nous analyserons en détail le diagramme et les concepts qu'il contient pour démontrer la montée en abstraction. Pour structurer la présentation, nous décomposons le diagramme en trois parties : la branche supérieure (R048 et ses dépendances), la branche inférieure (R047 et ses dépendances) et la jonction des deux branches (R050). Il est important de noter que les senseurs, configurateurs, requêtes et règles utilisés dans cet exemple sont présentés à la fin de celui-ci : les configurateurs et senseurs sont dans le tableau 3.1, les règles dans le tableau 3.2 et les requêtes dans le tableau 3.8.

3.2.2 Branche supérieure

Pour la branche supérieure, dont le diagramme est illustré à la figure 3.5, nous présenterons l'état de l'ontologie après chaque transformation. Ceci représente quatre états : avant le configurateur C001, après ce configurateur, après la règle T043 et après la règle T044.

Premièrement, nous présentons à la figure 3.6 l'ontologie préalablement définie pour cette branche. Dans ce chapitre, toutes les représentations d'ontologies sous forme de graphe uti-

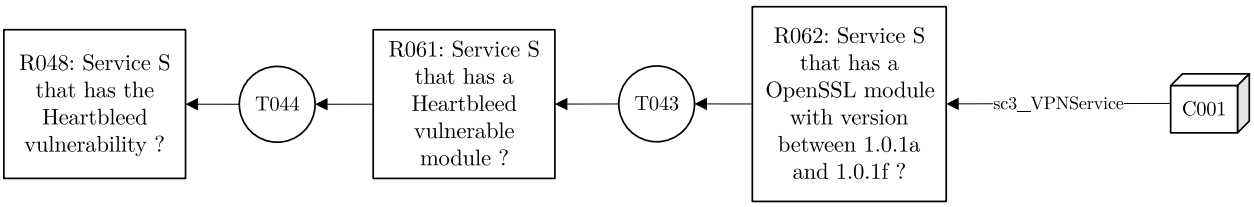

Figure 3.5 Branche supérieure du diagramme

lisent le formalisme suivant :

— trait standard : ressources instanciées par les senseurs
— trait pointillé : ressources préalablement dans l'ontologie
— trait gras : ressources inférées

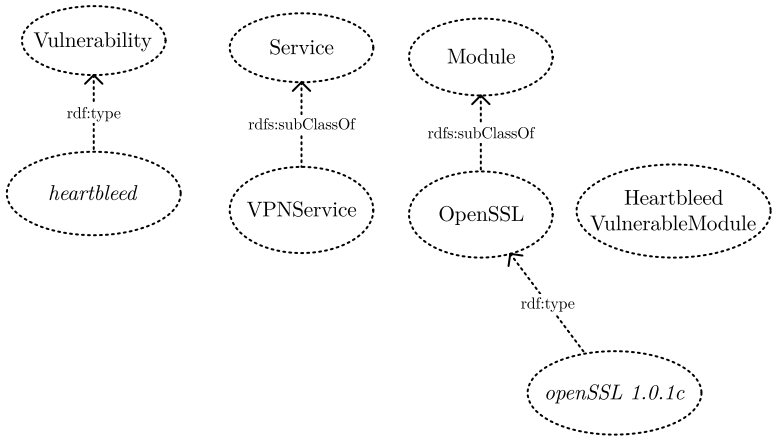

Figure 3.6 Ontologie Heartbleed avant C001 dans la branche supérieure

Ainsi, comme présentée à la figure 3.6, l'ontologie dans cet état contient deux instances, six classes et quatre propriétés.

Remarquons que dans l'état actuel, la dernière requête de la branche, soit la requête R048, retourne une réponse négative puisque le modèle de l'ontologie est vide. Nous poursuivrons la montée en abstraction pour que l'exécution de cette requête soit positive.

Deuxièmement, la branche supérieure commence par le configurateur C001 qui est décrit au tableau 3.1. Il crée une instance de la classe « VPNService ». Cette instance et ses propriétés rattachées sont illustrées sous forme de graphe orienté à la figure 3.7. Le configurateur ajoute des propriétés à l'instance « sc3_vpnService » : la propriété d'objet « hasModule » vers l'instance du module « OpenSSL 1.0.1c », et la propriété de données « hasPort ».

Par la suite, selon le flot de notre diagramme, l'information est documentée par une requête.

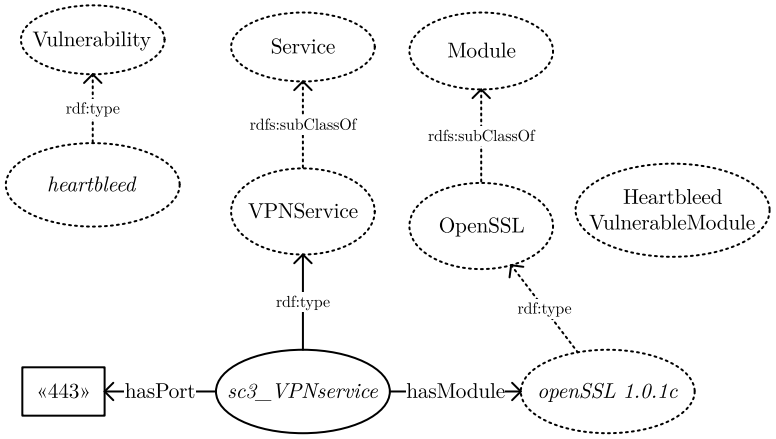

Figure 3.7 Ontologie Heartbleed après C001 dans la branche supérieure

Toutes les requêtes SPARQL de l'exemple sont décrites dans le tableau 3.8. Rappelons que les requêtes ne sont que des boîtes réutilisables par l'utilisateur et que leur emplacement dans le diagramme résulte d'une décision arbitraire des utilisateurs de la méthodologie. Leur présence dans le processus nous aide à comprendre où est rendue la transformation de la donnée. Ainsi, nous pouvons répondre à la requête R062 résumée comme suit : « Service qui a un module OpenSSL avec une version entre 1.0.1a et 1.0.1f ». Il a été possible d'obtenir une réponse à la requête à cause de l'instance qui vient d'être créée. Au premier état de cette branche, il n'y aurait pas eu de retour de la requête R062 puisqu'il n'y avait pas de service instancié.

Troisièmement, la donnée du service VPN (Virtual Private Network) se fait transformer par la règle T043. Cette règle ainsi que l'ensemble des règles dans cet exemple sont présentées dans le tableau 3.2. La transformation par cette règle permet à l'information de s'abstraire à un plus haut niveau par rapport à un langage expert. La règle T043 dit que les modules OpenSSL dont la version est entre 1.0.1a et 1.0.1f sont également du type HeartbleedVulnerableModule. L'application de cette règle sur notre ontologie est illustrée à la figure 3.8. Toujours selon le flot du diagramme, l'application de cette règle permet à la requête R061 de retourner une réponse positive. Cette requête demande les services qui ont un module vulnérable.

Quatrièmement, la règle T044 s'exécute sur le modèle ontologique. Cette règle est en langage SWRL et nous la décortiquerons pour aider à la compréhension. En langue naturelle, cette règle se lit comme suit : SI quelque chose de type Service ET cette chose est en relation avec quelque chose de type Module par la relation « hasModule » ET cette chose de type Module est aussi de type VulnerableHeartbleedModule DONC la chose de type Service est liée à l'instance de la vulnérabilité Heartbleed par la propriété « hasVulnerability ». L'exécution

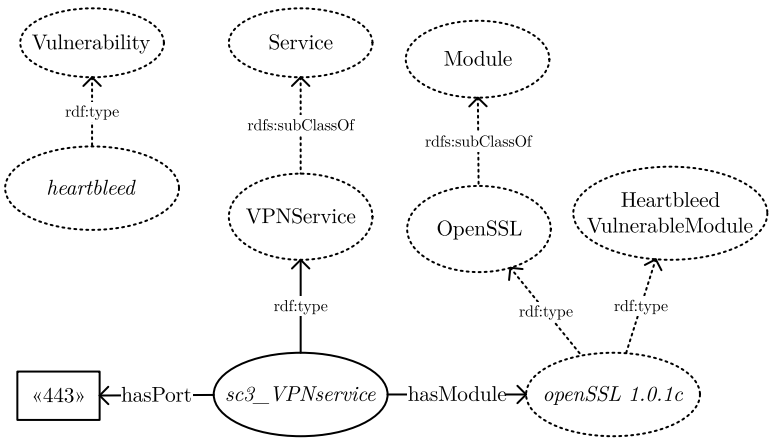

Figure 3.8 Ontologie Heartbleed après T043 dans la branche supérieure

de cette règle permet d'associer le service à une vulnérabilité.

À la suite de l'exécution de cette règle, nous avons la requête R048 qui est résumée par « service qui possède la vulnérabilité Heartbleed ». Sa représentation graphique correspond à la figure 3.9. Dans cette figure, notons la présence du lien en gras qui a été créé par une inférence. C'est grâce à lui qu'une réponse peut être donnée à la requête R048. Également, remarquons que de R062 à R048, aucune nouvelle information brute n'a été ajoutée, mais la requête n'est pas la même. La requête R062 est plus abstraite, soit plus proche du langage expert, comparé à la requête R048. La requête R061 est plus abstraite que R062, mais plus concrète que R048. Ce cas de figure représente le coeur de DIOSE : une montée en abstraction de R062 à R048 par des règles inhérentes à l'ontologie.

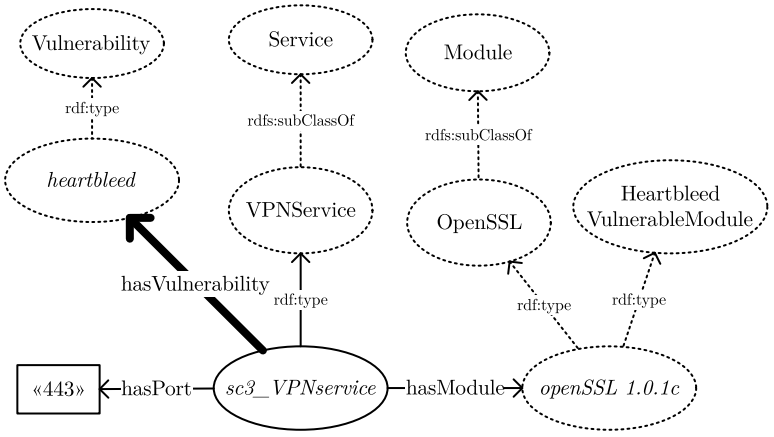

Figure 3.9 Ontologie Heartbleed après T044 dans la branche supérieure

Tableau 3.1 Montée en abstraction - Configurateur et senseur

index	description du configurateur ou du senseur
C001	• Création d'instances de :Machine basées sur les machines locales. Elles peuvent avoir les propriétés `:hasIPAddress some xsd:string` et `:offers some :Service` • Création d'instances de :VPNService avec les propriétés `:hasPort some xsd:integer` et `:hasModule some :Module`
S006	• Création d'instances de :HeartbleedExploit avec les propriétés `hasDestinationPort some xsd:integer`, `hasDestinationIPAddress some xsd:string` et `hasSourceIPAddress some xsd:string`

3.2.3 Branche inférieure

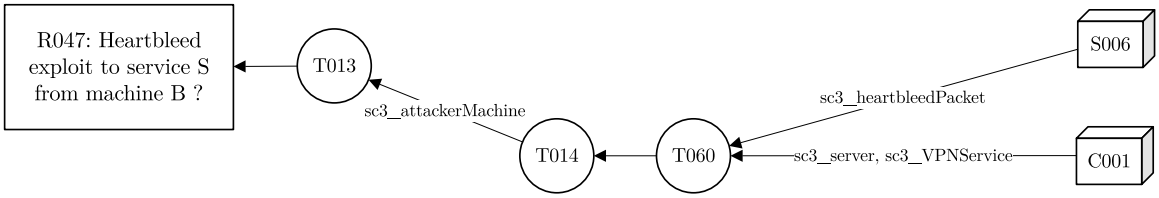

Figure 3.10 Branche inférieure du diagramme

Examinons le cas de la branche inférieure, tel que présenté à la figure 3.10. Dans celle-ci, nous souhaitons pouvoir répondre à la requête R047. Selon le flot du diagramme, nous présentons qu'il est nécessaire de procéder au parcours de la branche de droite à gauche pour pouvoir avoir une réponse à la requête R047. Ce parcours est composé de T060, suivi de T014 et suivi de T013. Initialement, nous avons une certaine structure dans l'ontologie, telle que représentée à la figure 3.11.

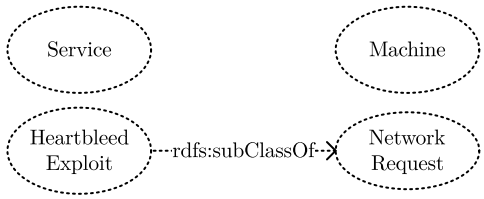

Figure 3.11 Ontologie Heartbleed avant C001 et S006 dans la branche inférieure

Premièrement, le configurateur C001 et le senseur S006 créeront des instances. Une instance de type « HeartbleedExploit » avec les propriétés de données « hasDestinationIPAddress », « hasDestinationPort » et « hasSourceIPAddress » sera créée par le senseur S006. Le configurateur C001 créera deux instances : la première de type Machine avec la propriété de données

Tableau 3.2 Montée en abstraction - Règles

réf.	type	règle
T043	LD	Les instances de :openSSL1.0.1a, :openSSL1.0.1b, :openSSL1.0.1c, :openSSL1.0.1d, :openSSL1.0.1e et :openSSL1.0.1f sont classifiées comme :HeartbleedVulnerableModule
T044	SWRL	Service(?service) ∧ hasModule(?service, ?module) ∧ HeartbleedVulnerableModule(?module) → hasVulnerability(?service, heartbleed)
T060	SWRL	NetworkRequest(?request) ∧ hasDestinationPort(?request, ?port) ∧ hasDestinationIPAddress(?request, ?dstip) ∧ hasIPAddress(?machine, ?dstip) ∧ offers(?machine, ?service) ∧ hasPort(?service, ?port) → hasDestination(?request, ?service)
T014	Algorithme	Requête toutes les instances de :NetworkEvent qui possèdent :hasDestinationIPAddress X ou :hasSourceIPAddress X. S'il n'y a pas d'instance de :Machine avec l'adresse IP X, crée (?newMachine rdf:type :ExternalMachine ; ?newMachine :hasIPAddress X) où :ExternalMachine est sous-classe de :Machine
T013	SWRL	Event(?event) ∧ hasSourceIPAddress(?event, ?x) ∧ hasIPAddress(?machine, ?x) → hasSource(?event, ?machine)

« hasIPAddress », et la deuxième de type Service avec la propriété de données « hasPort ». L'instance de type Machine sera reliée à l'instance de type Service par la propriété d'objet « offers ». À la suite de ces créations d'instances, nous obtenons l'ontologie qui est représentée sous forme de graphe à la figure 3.12.

Deuxièmement, un traitement sera effectué par la règle T060 sur les données. Celle-ci lie un événement réseau, soit notre exploitation Heartbleed, au service qu'il cible par les adresses IP et le port réseau. La règle est présentée au tableau 3.2 sous sa forme SWRL. Son exécution aboutit à l'ontologie suivante présentée à la figure 3.13.

Troisièmement, la règle T014 sera appliquée. Celle-ci est une règle quelque peu différente de celles que nous avons présentées jusqu'à présent. Contrairement aux autres règles SWRL et en LD, la règle T014 peut créer des instances. Elle est nécessaire pour que des machines distantes, dont les événements réseau qu'elles ont générés, soient modélisées dans l'ontologie. Pour éviter d'avoir initialement à modéliser l'ensemble des machines externes (qui représente une quasi-infinité), nous avons pris le choix de ne modéliser que celles qui sont liées aux événements de notre modélisation. Ainsi, la règle T014 parcourt les propriétés « hasDestinationIPAddress » et « hasSourceIPAddress » de tous les événements réseau. Pour la valeur de chacune de ces

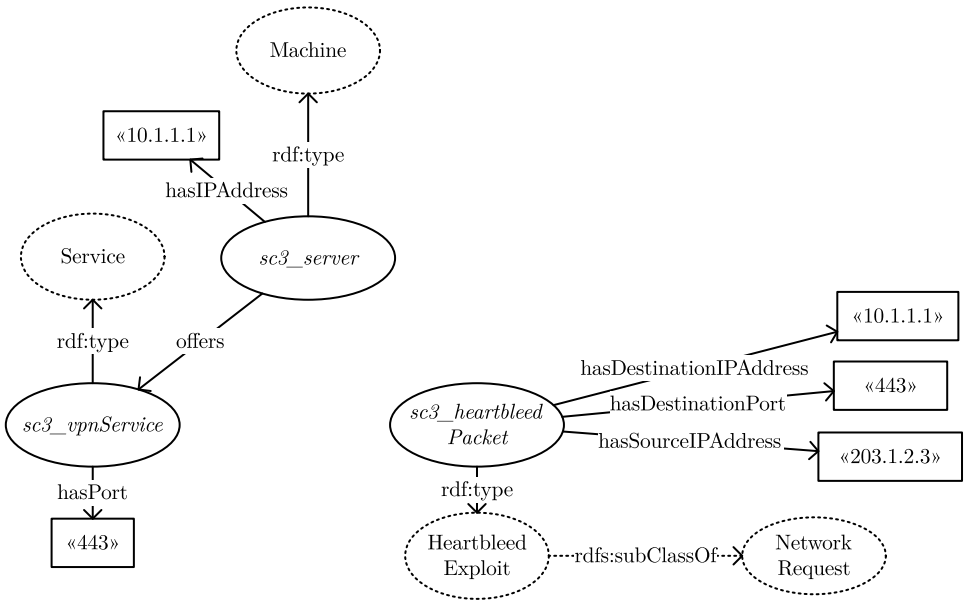

Figure 3.12 Ontologie Heartbleed après C001 et S006 dans la branche inférieure

propriétés, elle vérifie si une instance de type Machine existe avec cette adresse IP. Si ce n'est pas le cas, elle crée une instance de type ExternalMachine avec la bonne adresse IP. Puisque la règle T014 crée des instances, elle ne pouvait être en LD ou en SWRL.

Pour notre modélisation, l'algorithme trouve les adresses IP « 10.1.1.1 » et « 203.1.2.3 » qui sont reliées à l'événement réseau. Puisqu'aucune machine modélisée ne possède l'adresse IP « 203.1.2.3 », l'algorithme en créera une et y assignera cette adresse IP.

Quatrièmement, la règle T013 s'exécute à la suite de cette instanciation. Cette règle SWRL relie un événement réseau à la machine qui le génère. Cette relation est basée sur les adresses IP de la machine et l'adresse IP source de l'événement réseau. La figure 3.14 présente l'ontologie à la suite de cette transformation.

Pour terminer le flot de la branche inférieure, nous avons la requête R047 qui demande tous les paquets Heartbleed qui destinent un service et qui ont une machine source. Comme présenté à la figure 3.14, les traitements par des règles ont permis d'inférer les deux liens en gras qui permettent d'obtenir une réponse à la requête R047.

3.2.4 Jonctions des branches

Cette partie est constituée de la jonction des branches inférieures et supérieures. Celles-ci sont jointes dans une requête, soit la requête R050, pour permettre l'exécution de cette requête. Cette union des deux requêtes se fait en concaténant les triplets des deux requêtes.

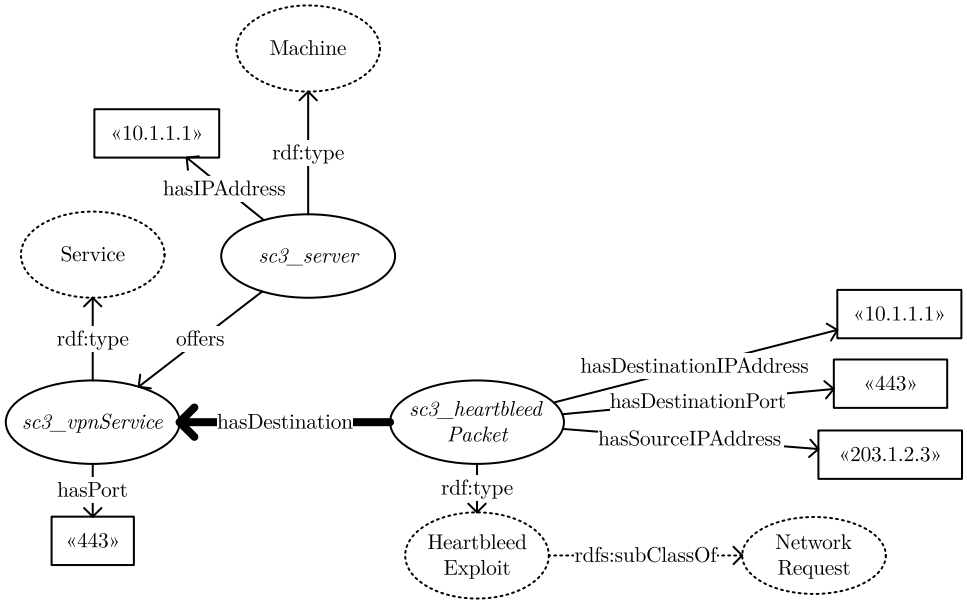

Figure 3.13 Ontologie Heartbleed après T060 dans la branche inférieure

Ainsi, nous avons R048 qui demande un service qui possède la vulnérabilité Heartbleed, puis nous avons R047 qui demande une exploitation Heartbleed sur un service provenant d'une machine. R050 se résume donc ainsi : une exploitation Heartbleed provenant d'une machine et destinant un service qui possède la vulnérabilité Heartbleed.

3.2.5 L'inconsistance logique

En plus des inférences créées, le stockage des informations dans l'ontologie apporte un autre avantage : la validation de consistance logique. En effet, lors d'une exécution du raisonneur, il va traverser les règles de l'ontologie (en LD et en SWRL) pour trouver des conditions valides et créer la propriété spécifiée dans la règle. Il peut être possible pour le raisonneur d'arriver dans un état indésirable lorsqu'il rencontre une déclaration qui est en contradiction avec une autre.

Par exemple, nous avons la propriété « offers » qui relie une machine à un service. Nous avons défini qu'elle est « fonctionnelle inverse », c'est-à-dire que pour « offers(x,y) », un y ne peut avoir qu'un seul x. Nous créons trois individus : deux machines, nommées « machine1 » et « machine2 », et un service nommé « service ». Le service « service » est offert à la fois par « machine1 » et « machine2 ». Finalement, nous explicitons que la « machine1 » et la « machine2 » sont deux individus distincts. Notons que nous comprenons qu'il s'agit d'une inconsistance puisqu'il est impossible pour un service réseau d'être offert par deux machines différentes. Puisque nous arrivons à cette conclusion avec notre logique, regardons comment

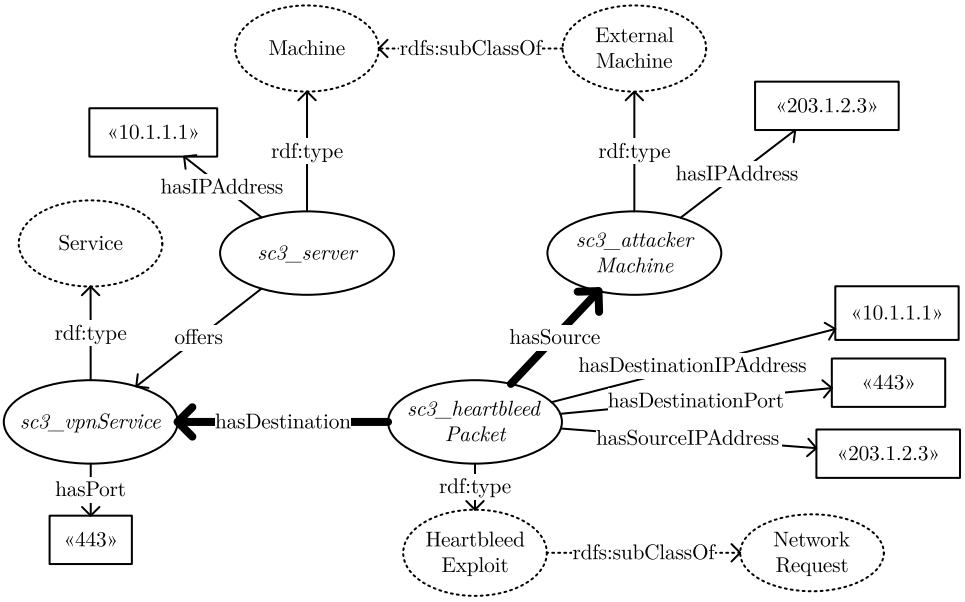

Figure 3.14 Ontologie Heartbleed après T013 dans la branche inférieure

le raisonneur arrive également à cette conclusion. La suite de triplets RDF représentés au tableau 3.3 exprime cet exemple :

Tableau 3.3 Exemple d'inconsistance

nº	triplet RDF
1	`machine1 rdf:type :Machine`
2	`machine2 rdf:type :Machine`
3	`service rdf:type :Service`
4	`machine1 :offers service`
5	`machine2 :offers service`
6	`machine1 owl:differentFrom machine2`

Le raisonneur qui parcourt cette suite de triplets RDF ne détectera qu'une inconsistance à la ligne nº 6. En effet, à la ligne nº 5, il a inféré que les individus « machine1 » et « machine2 » sont le même individu puisque la propriété « offers » est fonctionnelle inverse et qu'il ne peut exister qu'un seul x pour un y. Donc, selon son raisonnement et pour les termes liés par une propriété fonctionnelle, sa déduction serait comme suit :

si (x1, y) et (x2, y) donc x1 = x2

Ce n'est pas un cas anormal dans les ontologies puisque deux constantes peuvent désigner une même entité. Par contre, à la ligne n° 6, il est explicitement décrit que les deux individus ne sont pas les mêmes, c'est-à-dire que *x1 != x2*. À cause de cette inconsistance, le raisonnement ne peut se produire et, soit le modèle doit être adapté puisqu'il ne représente pas bien le cas à modéliser, soit les données sont invalides.

Ainsi, avec un modèle bien détaillé, il est possible de détecter les cas de figure non souhaités. Ceci assure une validation des données entrées dans l'ontologie.

Il est important de noter que l'utilisation d'une ontologie pour détecter des inconsistances logiques n'est pas un effet recherché, mais est plutôt une conséquence positive de l'utilisation de cette technologie. Celle-ci n'a pas été créée pour ce but, et son utilisation en tant que détecteur d'inconsistance n'a pas été éprouvée.

3.3 Machine à état

L'information étant disponible dans l'ontologie pour une corrélation selon l'expertise de l'expert, nous avons travaillé à améliorer la facilité d'utilisation et l'accessibilité de cette étape. En effet, la finalité désirée de notre projet est d'améliorer l'effort de détection d'intrusion en utilisant les ontologies. L'ontologie proposée aide à cet objectif, mais nous pouvons faire mieux en travaillant sur la facilité d'utilisation de DIOSE.

Nous avons réfléchi au concept d'une couche supplémentaire qui interagirait avec le module de corrélation, soit l'engin de requêtes SPARQL. Cette couche représente l'utilisation de l'ontologie développée à la couche précédente. Cette utilisation permet d'arriver, à l'aide de corrélation, à des conclusions intéressantes pour un expert. Ces corrélations s'utilisent pour détecter des étapes isolées des scénarios. À l'aide des fonctionnalités de DIOSE, il est possible de corréler ces étapes en un scénario pour permettre la détection de celui-ci. Ce chapitre sera présenté selon cet ordre, avec en premier la détection d'étapes. Ensuite, nous présenterons la détection de scénario en effectuant la corrélation des étapes.

Il y a différentes façons pour détecter une étape d'une attaque informatique. La plus simple est qu'un utilisateur choisit d'exécuter directement une requête SPARQL. Par exemple, il peut vouloir exécuter la requête 3.1 pour obtenir l'ensemble des machines qui possèdent une vulnérabilité et qui ont été la cible d'un balayage de port.

```
SELECT ?machine WHERE
{
        ?machine rdf:type :Machine .
        ?machine :hasVulnerability ?vulnerability .
        ?event rdf:type :PortScan .
        ?event :hasDestination ?machine
}
```
(3.1)

Ce mode d'utilisation exige une connaissance du langage de requête SPARQL. L'accessibilité de ce langage est assez aisée, puisqu'il s'agit d'un langage expressif et facilement lisible. L'expert se familiarise avec ce langage et une fois acquis, il peut effectuer toutes les requêtes possibles sur la base de connaissances pour y extraire les informations inférées dans l'ontologie. Ce mode d'utilisation offre un avantage de flexibilité. Par contre, dû à l'apprentissage d'un nouveau langage, il y a place à amélioration envers la facilité d'utilisation de DIOSE. Nous avons eu l'idée de proposer le concept de requêtes prédéfinies qui peuvent être réutilisées.

3.3.1 Requêtes prédéfinies

Les requêtes prédéfinies sont des requêtes SPARQL écrites à l'avance par un expert et elles peuvent être utilisées par un utilisateur sans une compréhension profonde de leur fonctionnement. Ces requêtes prédéfinies peuvent être incluses dans DIOSE.

Prenons un exemple de requête prédéfinie nommé R037. Elle permet d'obtenir toutes les machines sources, les pages Web et les sites Web impliqués dans une exploitation de vol de témoin (*cookie stealing*) par Cross-Site Scripting (XSS). Un expert du domaine créera cette requête, ainsi que toutes les classes, propriétés et règles qu'elle nécessite. L'expert a ainsi encodé son savoir dans le système expert et cela permet ainsi à un utilisateur moins expérimenté d'utiliser la requête sans connaître son fonctionnement intrinsèque. Il n'a pas besoin de savoir le déroulement précis d'une attaque de vol de témoin par XSS, mais il peut vouloir la détecter puisqu'elle fait partie des attaques Web les plus populaires.

Ce cas de figure démontre la force de DIOSE puisqu'il permet d'encoder le savoir d'un expert et d'utiliser ce savoir par un utilisateur moins expert.

Les requêtes associées à DIOSE se situent à différents niveaux d'abstraction de l'information : certaines sont près des données brutes et certaines sont près du langage expert. Au niveau d'abstraction le plus élevé, nos requêtes représentent les étapes des scénarios. Ainsi, un utilisateur de DIOSE peut utiliser ces requêtes prédéfinies pour détecter indépendamment chacune des étapes d'un scénario. On peut comparer ces requêtes prédéfinies à des blocs de construction où un utilisateur peut choisir d'utiliser un bloc pour la première étape d'un scénario, suivi d'un autre bloc pour la deuxième étape de ce scénario. Nous les appelons des blocs puisqu'ils sont construits de telle sorte qu'ils soient modulaires. Ils sont exécutés de façon indépendante et c'est l'expert qui effectue par lui-même la corrélation des entrées et sorties entre les deux blocs. Par exemple, une corrélation pourrait être nécessaire pour définir qu'il s'agit de la même machine source entre deux étapes.

Ceci n'est qu'une solution partielle puisque notre objectif était la modélisation à des fins de détection de scénarios complets et non d'étapes isolées. En vue d'améliorer cette solution, nous avons approfondi l'idée d'une couche supérieure à l'ontologie.

3.3.2 Machine à état déterministe

Cette couche implémente le comportement d'une machine à état finie pour modéliser un scénario. Chaque état de celle-ci est une étape, soit une requête SPARQL réutilisable, et elle a des entrées qui dépendent des étapes précédentes en fonction du scénario. Nous avons choisi d'utiliser une machine à état pour modéliser un scénario puisqu'il s'agit d'un modèle qui épouse parfaitement la forme du concept à modéliser. Une machine à état est composée de différents états avec des conditions d'entrée et de sortie. En modélisant chaque étape d'un scénario en tant qu'état et en déterminant les sorties et les conditions d'entrées, nous pouvons représenter les scénarios et utiliser les sorties de l'ontologie comme élément de notre machine à état. Également, l'aspect séquentiel de notre scénario peut être respecté lors d'une modélisation en machine à état.

Ainsi, l'utilisation d'une machine à état permet de représenter un scénario. Puisque les requêtes prédéfinies représentent déjà les étapes, notre machine à état sera une succession de requêtes SPARQL avec des conditions de passage d'une étape à l'autre. Ceci définit un autre mode d'utilisation des requêtes prédéfinies : le cas où une ou plusieurs variables de la requête sont spécifiées, ou contraintes. Dans le cas présenté précédemment, soit la requête R037, nous demandions tous les sites Web, toutes les pages Web et toutes les machines sources qui ont été impliqués dans une exploitation XSS. Il s'agit d'une utilisation générale de ce bloc de construction puisque nous ne contraignons aucune variable. Par contre, nous pourrions vouloir exécuter la requête en contraignant la variable du site Web pour la spécifier et désigner

un site Web précis. Le retour d'une telle requête serait plus contraint que la même requête sans contrainte, mais elle nous permet de l'utiliser dans un scénario où la contrainte sur le site Web aurait été la sortie d'une étape précédente. Ainsi, l'état représenté par la requête R037 se définit par ses sorties, ses possibles entrées et son corps, soit la requête SPARQL. Le tableau 3.4 décrit ces caractéristiques. Notons que pour notre machine à état, nous définissons que toutes les variables d'une requête sont de potentielles sorties. Il en va selon la nature de SPARQL qui retourne les ensembles correspondant aux triplets de la requête, puis qui ne fait que sélectionner ceux à afficher à la suite de l'exécution de la requête. À des fins de simplicité, nous ne présenterons pas les événements en tant que sortie puisque nous n'effectuons jamais de corrélation entre ceux-ci. Étant des variables dans une requête SPARQL, ils sont toutefois des sorties potentielles d'une requête.

Tableau 3.4 État de la requête R037

entrée	?website
sorties	?event1 ?event2 ?machine ?website ?webpage ?time1 ?time2

requête	SELECT * WHERE {
	?event1 rdf:type :XSSPayload ;
	:hasDestination ?webpage .
	?event2 rdf:type :WebpageRequestByAdmin ;
	:hasDestination ?webpage .
	...
	?event1 :hasTimestamp ?time1 .
	?event2 :hasTimestamp ?time2 .
	FILTER(?time1 < ?time2)
	}

Nous présenterons un exemple de corrélation entre deux étapes à l'aide d'une machine à état avec l'exemple des requêtes R050 et R053. Nous souhaitons détecter un scénario dans lequel premièrement, un utilisateur exploite un service réseau qui possède la vulnérabilité Heartbleed à partir d'une machine et deuxièmement, il s'authentifie à ce service vulnérable à partir de la même machine source que celle de l'étape précédente. Nous représentons ces deux requêtes ainsi que leurs variables dans la figure 3.15.

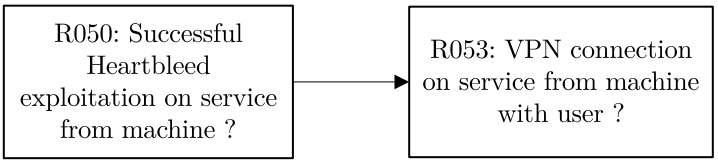

Figure 3.15 Machine à état générale avec R050 et R053

Sans l'implémentation d'une machine à état, un utilisateur exécuterait la première requête (R050) sans contrainte sur les variables. L'exécution de la requête résulterait en : « pour toutes les exploitations Heartbleed, sortir les machines sources et les services ciblés ». Cette requête sur une ontologie populée pour l'exemple retournerait les résultats présentés au tableau 3.5.

Tableau 3.5 Résultats de la requête R050

résultat	?machine	?service
n° 1	bot49	printerInterface
n° 2	bot23	fileTransferService
n° 3	attackerMachine	VPNservice
n° 4	bot34	VPNservice

Par la suite, la deuxième requête serait exécutée : « pour toutes les authentifications réseau sur un service VPN, sortir le service, la machine source de l'événement ainsi que l'utilisateur lié à cette authentification ». Les résultats de cette requête sont au nombre de trois et ils sont présentés au tableau 3.6.

Tableau 3.6 Résultats de la requête R053

résultat	?machine	?service	?user
n° 1	aliceLaptop	VPNservice	Alice
n° 2	attackerMachine	VPNservice	Bob
n° 3	bobAndroid	VPNservice	Bob

L'enchaînement de ces deux requêtes nécessite un travail de corrélation pour déduire que le couple d'individus « attackerMachine » et « VPNService » est le même dans les deux requêtes. Cela permet de déduire que le scénario se produit avec ces deux individus, et qu'il ne s'agit pas de différentes instances qui sont présentes dans les deux étapes.

Pour éviter que cette corrélation de liaison entre deux étapes soit faite manuellement, nous proposons une machine à état en langage de programmation qui contraindra les requêtes pour spécifier la valeur d'une variable. Nous présenterons le scénario R050-R053 par la figure 3.16 qui suit un formalisme que nous avons défini.

La figure 3.16 présente deux états, R050 et R053, qui possèdent des variables présentées en dessous de la boîte supérieure de l'état. Ces variables sont celles utilisées dans les requêtes SPARQL et sont des sorties de la requête. Selon les besoins du scénario, ces variables peuvent être contraintes avec une valeur précise. Ce mécanisme représente l'entrée d'un état. Ainsi, l'état R050 ne possède pas d'entrée et possède deux sorties (M et S). L'état R053 possède deux entrées (M et S) qui sont les individus M et S de l'étape précédente, et elle possède

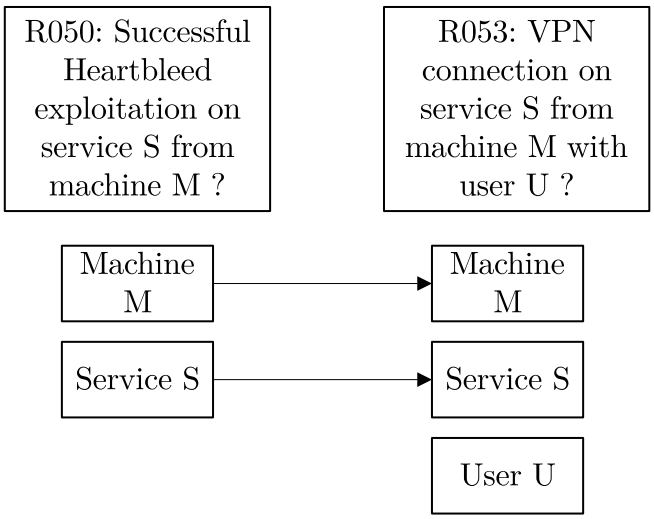

Figure 3.16 Machine à état spécifique avec R050 et R053

trois sorties (M, S et U). Ce formalisme permet de modéliser un scénario et de représenter le lien entre les différentes étapes d'un scénario.

Ainsi, nous proposons la modélisation d'un scénario sous cette forme. Chaque étape sera composée d'une requête SPARQL qui possède ses variables. Celles-ci peuvent être contraintes en tant qu'entrée, et elles sont également des sorties de l'étape.

La modélisation de notre scénario doit également posséder une composante essentielle : la temporalité. Cette composante se divise en deux aspects que nous traiterons séparément. Premièrement, un scénario doit être séquentiel. En raison de sa nature, il est constitué d'événements qui se déroulent dans un ordre prédéfini pour que cela corresponde au scénario. Par exemple, le scénario présenté à la figure 3.16 exprime implicitement une séquentialité entre les deux étapes : l'étape R050 s'effectue avant l'étape R053. Cette séquentialité n'est pas explicitement exprimée, mais elle est bien présente. Dans ce cas, le système expert qui serait responsable de la machine à état validerait que le temps associé à l'événement de l'étape R053 est supérieur au temps de l'événement de l'étape R050. Une telle implémentation préserve la séquentialité. Notre diagramme exprime la séquentialité par l'orientation des flèches entre les états, c'est-à-dire de gauche à droite. Dans les cas où il y a plusieurs événements dans une même étape, on considère l'estampille temporelle du premier événement. Ainsi, nous intégrons l'aspect séquentiel des étapes. Que ce soit entre deux étapes ou deux événements, nous pouvons extraire les temps et faire une comparaison par valeur pour nous assurer de leur ordre. Ces comparaisons s'effectuent dans les requêtes SPARQL ou directement avec DIOSE.

Deuxièmement, nous devons considérer que les étapes d'un scénario se déroulent dans une

fenêtre de temps. Pour donner un exemple de la nécessité de cette fonctionnalité, considérons le scénario présenté à la figure 3.16. En se basant uniquement sur la séquentialité, le scénario pourrait être détecté si l'étape n° 1 se produit le 1er janvier 2016 et l'étape n° 2 se produit le 20 août 2017. La séquentialité est respectée, mais la différence de temps entre les deux étapes est beaucoup trop importante pour que le scénario soit plausible. Ainsi, en plus de la séquentialité, une validation sur la différence des temps de deux étapes est effectuée par DIOSE. Le seuil de cette différence dépend de la nature des étapes. De plus, il s'agit d'une valeur très arbitraire et qui peut être largement débattue. Il est important de noter que malgré la description du concept de fenêtre de temps, DIOSE ne l'utilise pas. Nous avons pris ce choix pour contraindre notre recherche et ne pas affronter toutes les problématiques en même temps.

Nous reprenons maintenant l'exemple présenté à la figure 3.15 en lui appliquant un modèle de machine à état. Le scénario débute avec la requête R050 qui ne possède pas d'entrée. Les résultats de cette sortie, qui sont inchangés de l'exemple précédent, sont présentés au tableau 3.5. L'étape suivante est la requête R053. Contrairement au cas général présenté au tableau 3.6, il y a des contraintes d'entrée sur les variables ?machine et ?service. Au tableau 3.7, nous présentons la comparaison entre les deux formats de la requête R053 : la requête générale sans contrainte et la requête spécifique avec contraintes.

Tableau 3.7 Requête R053 avec et sans contrainte

Requête générale	Requête spécifique

SELECT ?service ?machine ?user WHERE
{
?event rdf:type :VPNAuthentication .
?event :hasSource ?machine .
?event :hasDestination ?service .
?service rdf:type :Service .
?machine rdf:type :Machine .
?event :hasUser ?user
}
	SELECT ?service ?machine ?user WHERE
	{
	?event rdf:type :VPNAuthentication .
	?event :hasSource **:attackerMachine** .
	?event :hasDestination **:VPNService** .
	:VPNService rdf:type :Service .
	:attackerMachine rdf:type :Machine .
	?event :hasUser ?user
	}

Ainsi, en utilisant la requête spécifique, nous pouvons fixer les variables ce qui permet de modéliser nos conditions d'entrée d'un état à l'autre, c'est-à-dire d'une étape de scénario à l'autre. Les requêtes représentant les étapes seront exécutées avec les entrées le cas échéant, puis si la requête de la dernière étape retourne une réponse, le scénario en entier a été détecté.

3.4 Résumé de l'architecture

Nous avons présenté DIOSE, qui est le système expert que nous proposons. En entrée, nous avons des informations brutes. Il peut s'agir d'informations relatives à un événement, par exemple l'authentification d'un utilisateur par HTTP, ou la détection d'un balayage de ports. Les informations brutes peuvent également être des données d'environnement, telles qu'un utilisateur de l'entreprise, un serveur Web ou une politique de groupe utilisée pour le groupe des administrateurs réseau.

Ces entrées sont collectées par des senseurs ou configurateurs selon leurs types. Cette collecte constitue l'entrée des données brutes dans DIOSE. Elles sont stockées dans une ontologie qui a été antérieurement conçue. Cette ontologie possède une logique de règles de classification, dont est responsable le raisonneur. Celui-ci enrichira les données brutes avec des traitements prédéfinis sous forme de règles de divers formats. À la suite du stockage et d'une partie de la corrélation, les informations brutes ont été abstraites en étapes de scénarios d'attaque informatique.

La prochaine phase du système expert est d'abstraire ces étapes en scénario complet. Pour ce faire, les étapes représentées dans l'ontologie sont modélisées en états d'une machine à état déterministe. Cette représentation permet l'ajout de concepts d'entrée et de sorties à un état, ce qui épouse parfaitement la forme d'une étape d'un scénario.

Ainsi, un scénario d'attaque informatique est modélisé en machine à état. Donc, ces états sont des concepts dans l'ontologie qui représente eux-mêmes des données collectés. Ceci représente une transformation de l'information, d'une représentation concrète à une représentation abstraite.

Nous proposons que l'abstraction offerte par DIOSE soit utilisée comme appui à un expert. Ce dernier pourra stocker son expertise dans le système expert. L'ontologie est une bonne technologie pour entreposer de l'information variée, ce qui est le cas de l'expertise de détection d'intrusion. Ainsi, l'expert pourra continuer d'effectuer la détection d'intrusion, mais sa performance sera augmentée puisqu'il pourra avoir DIOSE comme appui.

Tableau 3.8 Montée en abstraction - Requêtes

Index	Requête

R062	SELECT * WHERE {
	{ ?service :hasModule :openSSL1.0.1a } UNION
	{ ?service :hasModule :openSSL1.0.1b } UNION
	{ ?service :hasModule :openSSL1.0.1c } UNION
	{ ?service :hasModule :openSSL1.0.1d } UNION
	{ ?service :hasModule :openSSL1.0.1e } UNION
	{ ?service :hasModule :openSSL1.0.1f } .
	?service rdf:type :Service
	}

R061	SELECT * WHERE {
	?service :hasModule ?module .
	?module rdf:type :HeartbleedVulnerableModule .
	?service rdf:type :Service
	}

R048	SELECT * WHERE {
	?service rdf:type :Service .
	?service :hasVulnerability :heartbleed
	}

R047	SELECT * WHERE {
	?heartbleedExploit rdf:type :HeartbleedExploit .
	?heartbleedExploit :hasSource ?attackerMachine .
	?heartbleedExploit :hasDestination ?service .
	?attackerMachine rdf:type :Machine .
	?service rdf:type :Service
	}

R050	SELECT * WHERE {
	?event rdf:type :HeartbleedExploit .
	?event :hasSource ?attackerMachine .
	?event :hasDestination ?service .
	?service :hasVulnerability :heartbleed .
	?service rdf:type :Service .
	?machine :offers ?service
	}

Note: chacune des requêtes est précédée par les lignes suivantes: PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#>	

CHAPITRE 4 EXEMPLES D'APPLICATION DE DIOSE

À la suite de la description de DIOSE, nous présenterons son application à l'aide de scénarios d'attaque informatique que nous avons conçus. Ce chapitre est composé de deux parties. Premièrement, nous présenterons trois scénarios d'attaque. Deuxièmement, nous décrirons la détection de chacun de ces scénarios à l'aide de DIOSE. Ceci constitue le coeur de notre travail.

4.1 Description des scénarios d'attaque

Notre solution proposée aidera à la détection d'attaques informatiques qui sont décrites sous forme de scénarios d'attaque. Nous définissons le concept d'attaque informatique par une suite d'événements qui utilise des éléments de l'environnement et optionnellement des vulnérabilités dans le but de nuire à la sécurité d'un système. Un scénario est composé d'étapes, qui sont des regroupements d'événements, d'éléments d'environnement et de vulnérabilités fondées sur les connaissances d'un expert. Une étape d'un scénario tel qu'il est décrit dans cette recherche contient au minimum un événement.

La conception d'un modèle ontologique est dirigée par les questions auxquelles il doit répondre. Une pratique à éviter est de modéliser un domaine sans savoir de quelle façon ce modèle sera interrogé et ce que nous souhaitons en tirer (Gómez-Pérez et Suárez-Figueroa, 2009). Une telle approche n'est pas viable étant donné que la modélisation complète d'un domaine est extrêmement complexe, voire impossible. En établissant initialement les questions auxquelles l'ontologie doit répondre, nous définissons la limite de notre développement de l'ontologie. Ainsi, nous avons créé selon notre expertise trois scénarios d'attaque informatique. L'ontologie proposée doit aider à la détection de ces scénarios qui sont constitués de plusieurs étapes.

Notre validation à travers une poignée de scénarios choisis arbitrairement ne prouve pas que la solution proposée est efficace contre tous les scénarios. Par contre, si DIOSE fonctionne bien à travers ces trois scénarios, elle sera assez flexible pour détecter un quatrième et un cinquième scénario hypothétiques. Ainsi, dans une section subséquente nous soutiendrons que le modèle construit pour la détection de ces trois scénarios peut aisément être étendu pour détecter d'éventuels futurs scénarios. Nous avancerons l'hypothèse que le nombre de scénarios qui représente la majorité des attaques informatiques n'est pas infini et qu'il est dans les limites du monde à modéliser. En d'autres mots, la modélisation d'un nombre raisonnable

de scénarios précis permettra de détecter la majorité des attaques informatiques réelles en raison du nombre limité d'étapes possibles.

4.1.1 Choix des scénarios

Les trois scénarios qui seront présentés dans ce chapitre ont été choisis arbitrairement. Pour s'assurer de leur réalisme, qui est une caractéristique très importante pour ceux-ci, des éléments ont été pris de diverses sources (Verizon, 2017; Mandiant, 2017; Cisco, 2017; SonicWall, 2017; Verizon, 2016; Symantec, 2016; Lindqvist et al., 2003). Certaines de ces sources sont des rapports annuels d'entreprise qui font état de scénarios d'attaque rencontrés et qui les documentent pour aider les lecteurs à se prévenir contre ceux-ci. En incluant des éléments décrits dans ces rapports, nous nous assurons que nos scénarios ont un niveau de réalisme acceptable. Également, nous avons fait valider ces scénarios par des experts du domaine. Ceux-ci ont pu confirmer que les scénarios étaient bien réalistes et d'actualité.

4.1.2 Scénario n^{o} 1 - Infection par média amovible malveillant

Le premier scénario débute par le vecteur d'infection initiale d'une clé USB malveillante. Ce type de vecteur d'entrée est prisé par les attaquants puisque l'utilisateur n'a pas une grande méfiance envers l'action d'insérer une clé USB inconnue. Le scénario sera décrit en détail par ses paramètres initiaux puis par ses étapes. Ensuite, la règle de détection pour la corrélation sera présentée.

Paramètres du scénario

Le scénario prend place dans une entreprise qui possède un département de recherche et développement. Un prototype d'un nouveau produit est développé dans ce département, et cette propriété intellectuelle constitue un bien que l'attaquant souhaite obtenir. Les postes de travail de l'entreprise fonctionnent sous une version entreprise de Windows, et le service centralisé Active Directory est présent dans l'entreprise. Ce service offre une gestion des privilèges des utilisateurs qui est implémentée adéquatement dans l'entreprise.

Individus impliqués

- une partition réseau qui est identifiée par son chemin valant « M: »
- un fichier protégé nommé « blueprints.dwg » qui contient des plans secrets. Le fichier est sur la partition réseau « M: » dont les droits en lecture sont restreints aux administrateurs du réseau et au personnel approprié

— un groupe Active Directory qui contient les administrateurs du réseau et qui possède une politique de groupe qui décrit que les membres du groupe régis par la politique ont des droits administrateurs sur l'ensemble des machines
— un administrateur réseau nommé Bob qui est dans le groupe des administrateurs
— une machine qui se fera infecter, dont l'adresse IP est « 10.1.2.3 » et dont le nom d'hôte (*hostname*) est « WINDOWS-PC ». Cette machine fonctionne sous le système d'exploitation Windows 7.
— une machine externe dont l'adresse IP est « 96.20.1.2 » et qui est contrôlée par l'acteur malveillant
— une utilisatrice prénommée Alice qui possède la machine infectée et qui sera responsable de l'infection à ses dépens

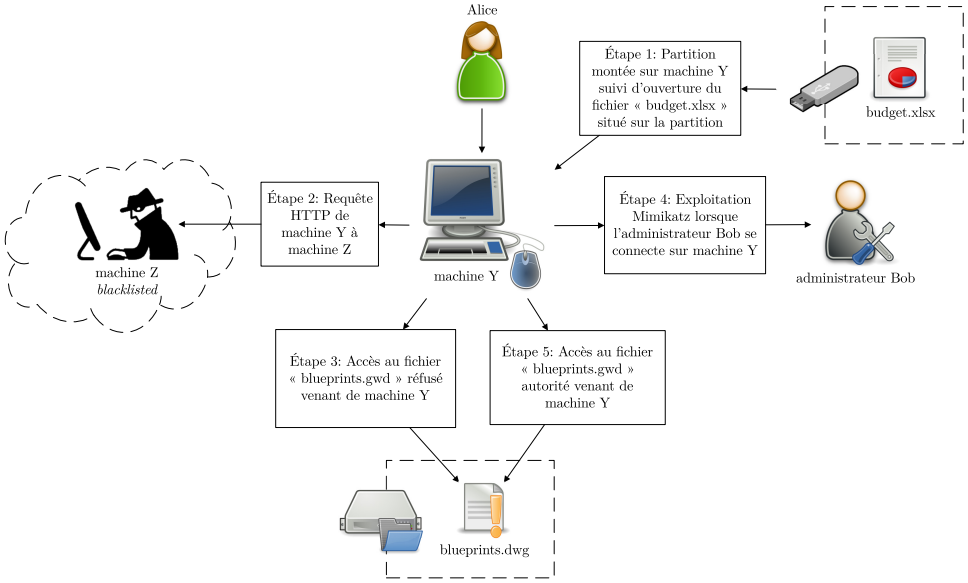

Figure 4.1 Diagramme du déroulement du scénario n° 1

Étapes du scénario

Première étape En arrivant au travail, l'employé du département de finances Alice trouve une clé USB dans le stationnement de l'entreprise. La clé USB possède la mention « Rapports financiers », ce qui pousse Alice à vouloir inspecter son contenu. Une fois arrivée à son poste de travail, elle insère la clé USB dans son ordinateur et y trouve des chiffriers Excel, nommés selon des exercices financiers annuels. Curieux de voir si les chiffriers sur la clé USB sont valides et pourquoi ces fichiers confidentiels sont dans un support amovible, l'employé ouvre un des chiffriers Excel. Le ruban jaune de sécurité de Microsoft l'avertit que les macros ont été désactivées par mesure de sécurité, mais le contenu de la feuille Excel avertit l'utilisatrice

que les macros doivent être activées pour bien voir le document. Ainsi, elle active les macros ce qui permet au code malveillant de s'exécuter.

Ce code malveillant est écrit sous le langage Visual Basic for Applications (VBA) et permet d'exécuter des fonctions système, telles que modifier, exécuter ou télécharger des fichiers.

- *Résultat* – Apporter le vecteur d'infection sur une machine interne
- *Trace observable n° 1* – Partition « Y: » d'un support amovible montée. Concrètement, il s'agit d'un événement Windows avec l'identifiant numérique 98. La trace est observée par la journalisation d'événements Windows
- *Trace observable n° 2* – Fichier sur partition « Y: » ouvert. Concrètement, il s'agit d'un événement Windows avec l'identifiant numérique 4663. La trace est observée par la journalisation des événements Windows

Deuxième étape La macro en VBA initie une connexion externe vers la machine de l'attaquant et elle attend des instructions. Ceci permet à un attaquant d'exécuter un programme qui écoute (*listener*) sur son poste distant et d'attendre que les machines infectées s'y connectent.

La machine infectée fera une connexion HTTP sur le port 80 d'une machine externe au réseau local. Le choix de ce protocole est commun pour déguiser le canal de communication en trafic normal qui éveillera moins les soupçons. Une fois la connexion entre les deux machines effectuée, la machine infectée attend les commandes système de la machine de l'attaquant et elle les exécute. Ceci offre un terminal de commande à l'attaquant, il peut ainsi effectuer des commandes en ligne de console comme s'il était physiquement sur la machine. La machine externe possède une adresse IP qui a été classifiée comme malveillante par un service externe tel que Google Safe Browsing.

- *Résultat de l'étape* – L'attaquant possède un terminal en ligne de commande sur la machine interne
- *Trace observable n° 1* – Connexion réseau sortante de la machine d'Alice vers la machine de l'attaquant avec un protocole identifié comme douteux. La trace est observée par le pare-feu
- *Trace observable n° 2* – Machine externe est désignée comme malveillante à cause de son adresse IP. La trace est observée par une liste noire externe, telle que Google Safe Browsing

Troisième étape L'attaquant possède une console sur la machine infectée pour exécuter des commandes. Il tente de conclure sa mission en accédant au fichier confidentiel sur le

disque réseau « M: ». Ainsi, à travers l'invite de commande sur la machine infectée, l'attaquant tente d'effectuer une lecture sur le fichier « blueprints.dwg ». Cette action provient de l'utilisateur connecté sur la machine infectée. Elle échoue puisque le système de privilèges est bien implémenté et l'utilisateur n'a pas les droits pour accéder à ce fichier.

- *Résultat de l'étape* – Échec de lecture d'un fichier protégé
- *Trace observable* – Accès refusé à une ressource distante faite par la machine infectée. Concrètement, il s'agit d'un événement Windows avec l'identifiant numérique 5140 et le statut d'échec. La trace est observée par la journalisation d'événements Windows

Quatrième étape L'attaquant détient les privilèges de l'utilisatrice Alice sur la machine. Il est donc administrateur local de sa machine, mais ne possède pas les droits pour lire la ressource distante confidentielle qui l'intéresse. Il effectue une exploitation de type Mimikatz qui se décompose en trois parties.

La première est la présence d'un système d'exploitation vulnérable. La portée de cette vulnérabilité est assez large puisqu'elle ne nécessite que le système d'exploitation soit l'un des suivants : Windows XP, Windows Vista, Windows 7 ou Windows 8.

La deuxième partie est une authentification distante de l'utilisateur Bob qui possède les droits d'administrateur sur la machine, et dont les identifiants de connexion seront conservés par le processus « Lsass.exe ».

La troisième partie est la présence d'un événement de lecture de la mémoire du processus « Lsass.exe ». Cette exploitation est associée à l'outil du même nom qui permet d'obtenir les mots de passe des utilisateurs connectés sur la machine. Ceci est possible à cause d'une fonctionnalité Windows par laquelle les mots de passe des utilisateurs connectés sont conservés en clair par le processus d'authentification « Lsass.exe ».

- *Résultat de l'étape* – Obtention des identifiants d'un utilisateur qui a accès au fichier confidentiel de propriétés intellectuelles
- *Trace observable n° 1* – Connexion distante d'un administrateur sur la machine. Concrètement, il s'agit d'un événement Windows avec l'identifiant numérique 4624 et le champ « logon type » à 10. La trace est observée par la journalisation d'événements Windows
- *Trace observable n° 2* – Lecture de mémoire du processus « Lsass.exe ». Concrètement, il s'agit d'un événement Windows avec l'identifiant numérique 3065. La trace est observée par la journalisation d'événements Windows

Cinquième étape Comme il l'a fait à la troisième étape de ce scénario, l'attaquant va tenter d'accéder à la ressource protégée « blueprints.dwg ». Le compte qu'il a compromis est administrateur du réseau, et celui-ci possède des droits de lecture et d'écriture sur toutes les partitions réseau, incluant celle qui contient le prototype du département de recherche et développement. Le fichier confidentiel que l'attaquant obtient est exfiltré sur une machine externe à l'entreprise. Cette exfiltration n'est pas un événement du scénario puisqu'elle se fait par le canal utilisé par l'attaquant depuis qu'il a compromis la machine.

- *Résultat de l'étape* – Lecture réussie du fichier protégé
- *Trace observable* – Accès réussi à une ressource distante fait par la machine infectée. Concrètement, il s'agit d'un événement Windows avec l'identifiant numérique 5140 et le statut de succès. La trace est observée par la journalisation d'événements Windows

4.1.3 Scénario n° 2 - Vol de données par vecteur d'entrée Web

Le deuxième scénario se déroule autour d'une entreprise qui possède une plateforme de commerce électronique pour vendre ses services. Ce type de vecteur d'entrée est populaire dans les attaques informatiques puisque ces plateformes sont généralement vulnérables à diverses attaques. Plusieurs raisons expliquent ceci. D'une part, elles sont faciles à attaquer puisqu'elles utilisent un protocole standardisé. D'autre part, leur développement continu se fait souvent en négligeant la phase de validation, ce qui introduit des failles de sécurité. Nous décrirons le scénario avec ses paramètres initiaux, puis les étapes qui le constituent.

Paramètres du scénario

Environnement L'entreprise de commerce électronique possède un serveur Web accessible depuis l'Internet et qui agit également comme serveur de base de données. Ce serveur fonctionne sous le système d'exploitation Linux avec la pile populaire de technologies Web : PHP comme langage des tâches d'arrière-plan, Apache comme serveur Web, et MySQL comme SGBDR.

Différents senseurs sont en place en tant que mécanismes de défense, tels que l'IDS Snort pour analyser le trafic réseau en entrée et en sortie. De plus, ModSecurity, une solution de type Web Application Firewall (WAF), est en place sur le serveur Web. Ce senseur est un pare-feu applicatif qui permet de filtrer et d'analyser le trafic HTTP entrant sur la machine pour y détecter des attaques Web. Également, un filtreur Web, SquidGuard, est en place pour contrôler et analyser ce qui se passe sur la couche HTTP. SquidGuard est un serveur mandataire (*proxy*) HTTP qui permet de contrôler l'accès à certaines ressources distantes

potentiellement malveillantes. Étant un intermédiaire pour toutes les connexions HTTP, il permet de garder leurs traces en les journalisant pour une analyse future.

Ce scénario comprend un attaquant qui souhaite obtenir des numéros de carte de crédit pour les monétiser. Il ne mise pas sur sa furtivité puisque son attaque s'effectue rapidement et il n'a aucun effet négatif à se faire détecter, excepté de ne pas pouvoir réaliser son objectif.

Individus impliqués

- une politique de groupe Active Directory qui explicite que les membres du groupe régies par la politique sont administrateurs du site Web de l'entreprise
- un groupe Active Directory qui contient les administrateurs du site Web et qui possède la politique de groupe décrite ci-haut
- un utilisateur nommé Bob qui a les droits d'administrateur sur le service Web qu'est le site Web
- une machine externe qui a l'adresse IP 132.207.1.2
- une machine distante à partir de laquelle l'attaquant fait la majeure partie de l'attaque. Elle possède l'adresse IP 96.1.2.3
- une machine qui offre un site Web qui se fait attaquer. Cette machine possède l'adresse IP interne 10.2.1.100 et contient des numéros de carte de crédit dans sa base de données
- une machine interne utilisée par un administrateur réseau de l'entreprise. Son nom d'hôte est « BOB-PC » et son adresse IP est « 10.2.1.101 »
- un site Web qui possède le nom de domaine « site.lan »
- une plateforme Web de commerce électronique installée sur ce site Web. La plateforme est WooCommerce et sa version est 2.3.2
- deux pages Web sur le site Web et qui ont respectivement comme chemin « /about » et « /forum »
- une page Web d'authentification qui fait partie de la section administrative du site Web et qui possède le chemin « /admin »
- une page Web qui fait partie de la section administrative du site Web et qui possède le chemin « /admin_plugins »

Étapes du scénario

Première étape Le scénario débute par un balayage de vulnérabilité Web effectué par l'attaquant. À l'aide d'une de ses machines, l'attaquant exécute sans arrêt son balayeur de vulnérabilités Web « W3af » sur tous les sites Web qu'il peut trouver. Dans le cas du site Web du scénario, des failles potentielles, telles que des champs utilisateurs mal filtrés ou des

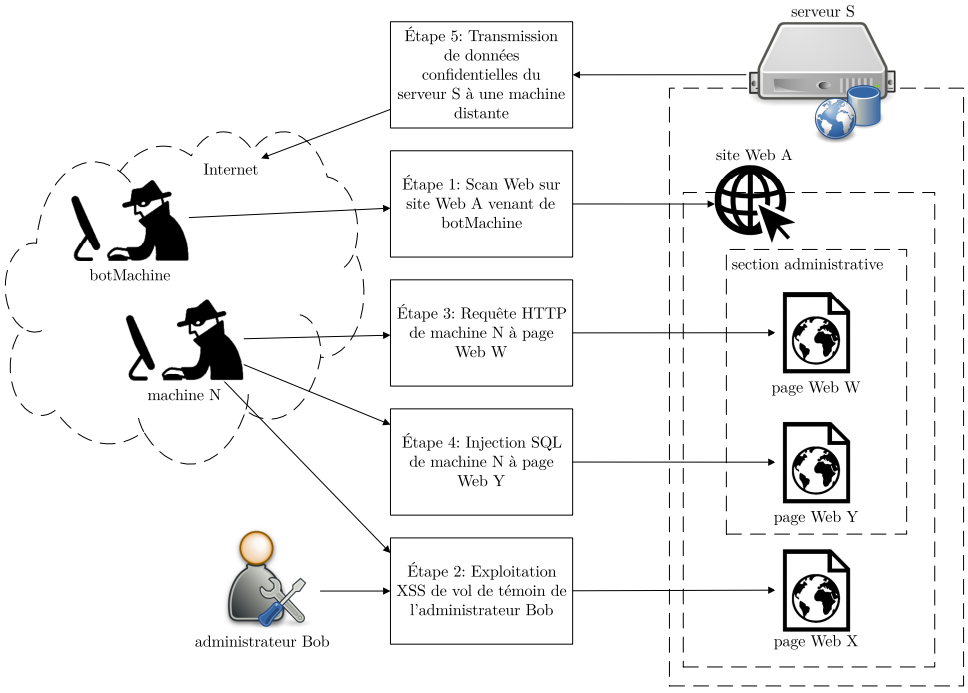

Figure 4.2 Diagramme du déroulement du scénario n° 2

dossiers lisibles sans autorisation, sont recensées. Le serveur Web n'étant pas sécurisé au plus haut point, l'outil de balayage permet de recenser quelques informations de reconnaissance utiles à l'attaquant. Un de ces résultats est la découverte d'une faille XSS. Il décide ainsi de poursuivre manuellement son attaque en ciblant le site Web.

- *Résultat de l'étape* – Découverte d'une faille XSS sur le site Web
- *Trace observable* – Balayage de vulnérabilités Web. La trace est observée par le NIDS Snort

Deuxième étape Une section du site Web agit en tant que forum, ce qui suscite l'intérêt de l'attaquant. Les textes envoyés sur le forum sont stockés dans la base de données pour être affichés sur le site Web.

À cause d'une mauvaise filtration des entrées du côté serveur, cette section est vulnérable aux exploitations XSS de type permanent. Ceci permet à du code client, typiquement des instructions JavaScript, d'être exécuté dans le navigateur d'une victime. C'est ce qu'effectue l'attaquant du scénario. Il inclut une instruction JavaScript qui effectue une requête vers une ressource distante, par exemple une image hébergée sur une autre machine. La valeur d'un des paramètres de cette requête sera le contenu des témoins (*cookies*) de la victime. Cette attaque s'appelle un vol de témoin (*cookie stealing*) et elle est une résultante prisée lors de

l'exploitation de vulnérabilités XSS. Ainsi, tous les utilisateurs qui naviguent sur la page Web où est affiché ce message tenteront de charger la fausse image, et enverront à leur insu la valeur de leurs témoins sur le site Web externe, qui est contrôlé par l'attaquant. Ceci permet à l'attaquant d'obtenir plusieurs identifiants d'authentification, dont celui de l'administrateur du site Web.

C'est spécifiquement ce témoin qui intéresse l'attaquant, puisqu'avec celui de l'administrateur du site Web, il va pouvoir se connecter au portail administrateur. Comme prévu, l'administrateur du site Web visite la page infectée et ses identifiants de connexion sont compromis à son insu.

- *Résultat de l'étape* – L'exploitation de la faille XSS a réussi : obtention du témoin de session d'un administrateur du site Web
- *Trace observable nº 1* – Requête HTTP comprenant l'exploitation de la faille XSS et qui a pour destination une page Web qui sera infectée. La trace est observée par ModSecurity, soit le WAF en place
- *Trace observable nº 2* – Requête HTTP ayant pour destination la page Web infectée et provenant d'une machine sur laquelle Bob l'administrateur du site Web est connecté. Trace détectée par la journalisation des connexions HTTP du serveur Web et par la journalisation des événements Windows

Troisième étape L'attaquant possède le témoin de l'administrateur du site Web et il peut l'utiliser comme le sien. En naviguant sur le portail du site Web, le mécanisme d'authentification reconnait le témoin de l'administrateur et permet à l'attaquant d'accéder à la section administrative du site Web.

- *Résultat de l'étape* – L'attaquant se connecte au portail administrateur du site Web
- *Trace observable* – Connexion à une page Web située dans la section administrative du site Web. La trace est observée par la journalisation des connexions HTTP du serveur Web

Quatrième étape Bien que l'attaquant ait accès à l'interface utilisateur, il n'a pas accès au contenu de la base de données, qui comprend l'ensemble des numéros de carte de crédit. Pour arriver à cette fin, il exploite une vulnérabilité Web qui lui donnerait accès à l'ensemble de la base de données. En fonction de la version des modules installés, il fait une recherche de vulnérabilités existantes. Pour la version de WooCommerce 2.3.2, il trouve la vulnérabilité référencée WPVDB-7846 qui permet une injection SQL. Cette vulnérabilité consiste en une mauvaise filtration d'un champ d'entrée dans l'interface administrative du module WooCom-

merce. En l'exploitant, l'attaquant peut extraire n'importe quelles données d'une table dans la base de données. Il s'intéresse à la table qui contient les numéros de carte de crédit et il forge son exploitation pour que le contenu de cette table soit retourné.

- *Résultat de l'étape* – L'attaquant a accès aux données confidentielles qu'il souhaite voler, c'est-à-dire les numéros de carte de crédit
- *Trace observable* – Requête d'injection SQL. La trace est observée par ModSecurity, soit le WAF en place

Cinquième étape L'attaquant procède à un téléchargement de la page Web qui contient tous les numéros de carte de crédit. Ces données transigent du serveur Web à la machine externe de l'attaquant, ce qui complète son attaque.

- *Résultat de l'étape* – L'attaquant extrait l'entièreté des numéros de carte de crédit vers une machine externe.
- *Trace observable* – Transmission de données personnelles sur le réseau. La trace est observée par le NIDS Snort

4.1.4 Scénario nº 3 - Rançongiciel par exploitation Heartbleed

Ce scénario est celui qui débute par l'exploitation réseau d'une vulnérabilité de type « 1-day ». Pour introduire cette appellation, nous expliquerons le type plus connu, soit les vulnérabilités « 0-day ». Celles-ci sont des vulnérabilités informatiques qui ne sont pas publiées. Leurs existences sont tenues secrètes par ceux qui les ont découvertes ou qui les ont achetées. Puisque son existence est inconnue, la vulnérabilité ne sera pas corrigée. On les nomme des vulnérabilités « 0-day » puisque les mesures pour corriger ce type de vulnérabilité sont au jour 0, soit avant le premier jour. Dans le même ordre d'idée, les vulnérabilités « 1-day » sont des vulnérabilités qui ont été publiées, mais uniquement depuis un court laps de temps. Par exemple, un blogue réputé décrit une nouvelle vulnérabilité pour accéder à l'interface de configurateur d'un pare-feu sans identifiant de connexion. Durant les vingt-quatre heures qui suivent cette publication, les développeurs du produit vulnérable travaillent pour concevoir et déployer un correctif, tandis que les attaquants malintentionnés abusent de la vulnérabilité publiée.

Paramètres du scénario

Environnement Dans le cas de ce scénario, ce sera la vulnérabilité Heartbleed qui sera exploitée. Il s'agit d'une vulnérabilité qui concerne certaines versions de la librairie OpenSSL, qui implémente les protocoles Secure Sockets Layer (SSL) et Transport Layer Security (TLS)

dans les systèmes d'exploitation Unix, tels que Mac OS et Linux. En envoyant une certaine requête du protocole Heartbeat, il était possible d'obtenir une partie du contenu de la mémoire de l'application utilisant la librairie OpenSSL, telle qu'un serveur VPN ou un serveur Web qui utilise HTTPS (HyperText Transfer Protocol Secure). Le jour de la publication de la vulnérabilité, plusieurs attaques ont eu lieu, puisque le correctif envers Heartbleed était laborieux. Ce correctif était constitué d'une mise à jour de la librairie OpenSSL et d'obtention de nouveaux certificats et de clés de chiffrement.

Le scénario prend place dans une entreprise qui possède un environnement Active Directory. Pour permettre le télétravail de ses employés, l'entreprise possède un serveur VPN acceptant les connexions entrantes du Wide Area Network (WAN). Bien que l'environnement de l'entreprise soit composé majoritairement de machines Windows, le serveur VPN est un appareil tout-en-un, à la même manière qu'un routeur. Il fonctionne sous une version du système d'exploitation Linux qui utilise la version 1.0.1c de la librairie OpenSSL, soit une version vulnérable. Comme mécanisme de défense, un NIDS est en place entre le *local area network* (LAN) et le réseau externe.

Individus impliqués

- Un service VPN qui possède la vulnérabilité Heartbleed
- Une librairie OpenSSL version 1.0.1c qui est utilisée par le service VPN et dont la version rend le service vulnérable à Heartbleed
- Un serveur qui offre le service VPN et qui est identifié par son adresse IP « 10.5.1.200 »
- La machine distante avec laquelle l'acteur malveillant procède à son attaque. Elle a l'adresse IP « 201.1.2.3 »
- Une machine interne qui se fera infecter à la dernière étape. Il s'agit d'un poste client qui a comme adresse IP « 10.5.1.201 » et comme nom d'hôte « BOB-PC ».
- Le compte utilisateur de l'utilisatrice Alice, dont les identifiants vont être compromis par l'exploitation de la vulnérabilité sur le service VPN
- Des processus qui sont exécutés sur la machine qui se fera infecter par un logiciel de rançongiciel. À des fins de sécurité, les processus qui effectuent beaucoup d'opérations d'écriture sont étiquetés dans une liste blanche

Étapes du scénario

Première étape L'attaquant exécute un balayage de port sur une plage d'adresse IP pour trouver des hôtes vulnérables à la vulnérabilité Heartbleed. En utilisant le programme Nmap (Lyon, 2009), il peut voir quels ports sont fermés et quels ports sont ouverts. Ainsi, il cible les

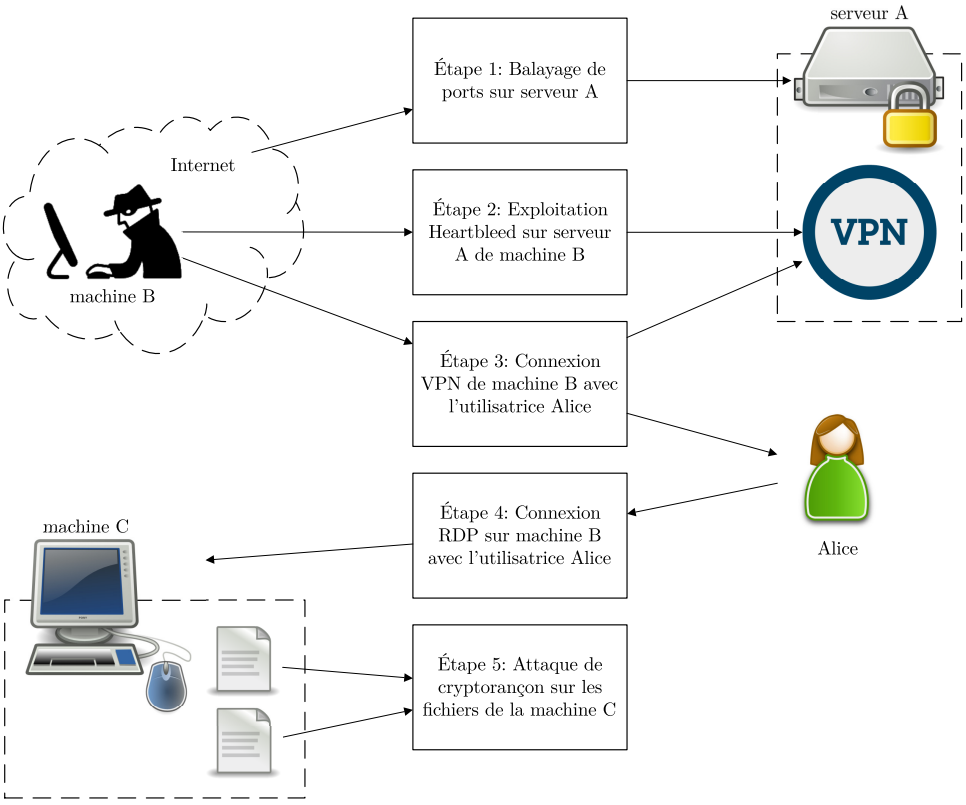

Figure 4.3 Diagramme du déroulement du scénario n° 3

ports des services affectés par la vulnérabilité Heartbleed, soit typiquement 1194 sous UDP (User Datagram Protocol) pour le service VPN ou 443 sous TCP (Transmission Control Protocol) pour le protocole HTTP. Le serveur VPN de l'entreprise est ciblé par le balayage de port.

- *Résultat de l'étape* – L'attaquant découvre un service potentiellement vulnérable sur le serveur VPN
- *Trace observable* – Balayage de port. La trace est observée par le NIDS Snort

Deuxième étape Pour certaines vulnérabilités, il est possible d'effectuer une vérification pour savoir si la vulnérabilité est bien présente. Dans le cas de Heartbleed, il n'existe pas de telle vérification. C'est donc directement l'exploitation de la vulnérabilité qui confirmera si l'attaque est un succès.

La vulnérabilité Heartbleed exploite une erreur de programmation dans une fonctionnalité « heartbeat » du protocole SSL. Il s'agit d'un mécanisme pour garder une connexion ouverte lorsque des données ne sont pas échangées. Ainsi, le serveur qui reçoit la connexion envoie un « heartbeat » au client lui demandant de lui répondre avec des données arbitraires et la

taille de ces données. La vulnérabilité est autour de ces paramètres. Si la taille des données annoncée est plus grande que la taille réelle des données, une partie de la mémoire aussi grande que 64 kilo-octets sera retournée. Cette partie de la mémoire de l'application exploitée peut contenir des informations confidentielles, telles que des identifiants de connexion ou des clés de chiffrements.

Les attaquants procèdent à l'exploitation de la vulnérabilité sur l'application du service VPN. Après quelques exploitations, ils obtiennent des identifiants de l'utilisatrice Alice.

- *Résultat de l'étape* – Obtention des identifiants de connexion VPN d'un utilisateur
- *Trace observable* – Requête d'exploitation de la vulnérabilité Heartbleed. La trace est observée par le NIDS Snort

Troisième étape Étant en possession d'une paire d'identifiants constituée d'un nom d'utilisateur et d'un mot de passe, l'attaquant procède à l'authentification. Il se fait attribuer une adresse IP locale à l'entreprise dans un sous-réseau dédié aux clients VPN.

- *Résultat de l'étape* – En utilisant le compte compromis, l'attaquant effectue une connexion vers le réseau interne de l'entreprise
- *Trace observable* – Connexion VPN avec l'utilisatrice Alice. La trace est observée par la journalisation des connexions du serveur VPN.

Quatrième étape Il ouvre son analyseur de trafic réseau préféré, Wireshark, pour tenter de faire de la reconnaissance passive de son environnement. À l'aide de celui-ci, il observe différents paquets du protocole RDP (Remote Desktop Protocol) qui destinent une même adresse IP. L'attaquant en déduit que cette adresse IP est celle du serveur RDP, et il s'y connecte avec les identifiants usurpés de l'utilisatrice Alice. La connexion est un succès et il a un accès graphique à un poste de travail interne à l'entreprise.

- *Résultat de l'étape* – L'attaquant se connecte sur une machine du réseau interne
- *Trace observable* – Authentification RDP avec l'utilisateur compromis sur une machine interne. Concrètement, il s'agit d'un événement Windows avec l'identifiant numérique et le champ « logon type » à 10. La trace est observée par la journalisation des événements Windows

Cinquième étape L'attaquant choisit un fichier exécutable malveillant de type rançongiciel pour infecter la machine sur laquelle il est connecté. Un rançongiciel (*ransomware*) est un logiciel malveillant qui chiffre les données sur un poste de travail. Le déchiffrement de ces fichiers est pratiquement impossible sans la clé de déchiffrement qui est vendu par celui qui

a malhonnêtement chiffré les données. Les fichiers personnels de l'utilisateur sont donc tenus en otage contre une rançon pécuniaire, d'où le nom « rançongiciel ».

L'attaquant télécharge l'exécutable qui va procéder à l'infection à travers le canal VPN déjà ouvert puis l'exécute. Tous les fichiers excepté ceux nécessaires au fonctionnement du poste de travail voient leurs contenus chiffrés et leurs noms changés.

- *Résultat de l'étape* – Infecter la machine et conclure la mission
- *Trace observable* – Actions d'écriture sur des fichiers faits à une fréquence anormalement élevée. Concrètement, il s'agit d'événements Windows avec l'identifiant numérique 4663. La trace est observée par la journalisation des événements Windows.

4.2 Application à travers les scénarios

À la suite de la description des trois scénarios d'attaque informatique, nous présenterons leurs détections à travers DIOSE, dont nous avons décrit l'architecture au chapitre précédent. Dans cette section, la présentation se fera selon le processus de détection d'intrusion, soit les étapes de collecte, d'agrégation et de corrélation. Nous ne proposons aucune amélioration pour l'étape de collecte puisque les inconvénients de celle-ci ont déjà été résolus à travers les itérations des produits existants. Néanmois, nous présenterons les pilotes liés aux senseurs et configurateurs que nous avons conçus pour faire le lien entre les étapes de collecte et d'agrégation. Ensuite, nous présenterons le coeur de DIOSE, soit l'ontologie qui constitue l'étape de stockage. Ensuite, nous présenterons l'utilisation de l'ontologie à des fins de corrélation. Cette dernière partie du processus de détection présentera la détection des étapes de chacun des trois scénarios.

4.2.1 Pilotes liés aux senseurs et configurateurs

Lors du développement de DIOSE pour détecter les scénarios, nous avons réalisé la conception de différents pilotes. Ceux-ci sont liés soit à des configurateurs, soit à des senseurs. Les pilotes sont responsables de faire le lien entre la collecte et la représentation de l'information par l'ontologie. Concrètement, ils analysent l'information collectée et créent des instances pour bien représenter l'information. Les instances créées sont également enrichies par leurs propriétés pertinentes.

Pour effectuer la détection des trois scénarios précédemment présentés, nous avons fait la conception de 9 pilotes pour les senseurs et de 5 pilotes pour les configurateurs.

À l'instar des composantes que nous avons créées, soit les règles, les requêtes, les pilotes et

les détails de la modélisation avec l'ontologie, nous ne les présenterons pas tous dans cette section. Nous prenons ce choix pour aider à la fluidité de la lecture. Les composantes reliées à l'ontologie sont présentées à l'annexe A, tandis que les composantes reliées à DIOSE, soit les senseurs, les configurateurs et les requêtes, sont présentées à l'annexe B.

Nous présentons les pilotes que nous avons conçus selon leur nature : ceux liés aux senseurs sont présentés au tableau B.2 et ceux liés aux configurateurs sont présentés au tableau B.3.

Tableau 4.1 Échantillon des pilotes conçus

Index	Description
S004	S'interface avec la journalisation du pare-feu et crée une instance de :NetworkRequest avec ces propriétés, lorsqu'applicable: • :hasSourceIPAddress (p. ex. « 132.207.4.47 ») • :hasDestinationPort (p. ex. 22) • :hasSourcePort (p. ex. 12635) • :hasDestinationIPAddress (p. ex. « 10.5.1.15 ») • :hasProtocol (p. ex. :TCP)
C004	Crée des instances de :Group en fonction des groupes Active Directory. Elles ont ces propriétés: • :hasName (p. ex. « dev ») • :hasMachineMemberName (p. ex. « BOB-PC ») • :hasUserMemberName (p. ex. « Bob ») • :hasGroupPolicy :GroupPolicy Crée des instances de :GroupPolicy en fonction des politiques de groupes Active Directory. Elles ont ces propriétés: • :hasName (p. ex. « dev-policy ») • :hasAdminRightsOnGroupName (p. ex. « dev ») • :hasAdminRightsOnName (p. ex. « cluster1.dev.lan »)

Nous présentons un pilote des deux types existants au tableau 4.1. Le pilote S004 présenté s'interface avec le senseur du pare-feu. Il collecte les événements des paquets aux couches 2, 3 et 4 du modèle Open Systems Interconnection (OSI). Le pilote C004 présenté s'interface avec le gestionnaire de groupe et de politiques de groupe Active Directory.

4.2.2 Ontologie et règles

Nous avons développé une ontologie qui permet de stocker les informations nécessaires à la détection des trois scénarios. L'ontologie est une modélisation réalisée autour de trois concepts généraux que sont l'événement, la vulnérabilité et l'environnement. Nous présenterons les classes, les propriétés et les règles que nous avons développées. Le tableau 4.2 décrit

quantitativement les éléments que nous avons créés dans notre ontologie.

Tableau 4.2 Volumétrie de l'ontologie développée

concepts	dénombrement
classes	67
propriétés d'objet	25
propriétés de données	25
règles SWRL	28

Notons que nous avons choisi l'usage de l'anglais pour les noms des individus, des classes et des propriétés. Malgré que nous prônons l'usage du français dans tous les domaines, nous avons misé sur l'interopérabilité de notre modélisation. Ainsi, elle peut être réutilisée dans une autre solution existante plus facilement due à la prédominance de l'anglais dans le domaine de l'informatique.

Les classes

Chacune des classes créées est associée à la modélisation d'un des trois concepts centraux de l'ontologie (événement, vulnérabilité et contexte). Ces trois ensembles de classes auraient pu être dans des ontologies séparées, mais nous avons choisi de les réunir dans une seule ontologie. Pour structurer la présentation des composantes de l'ontologie utilisée à des fins de détection, la présentation se fera par concept central modélisé. Pour chaque classe développée, nous décrirons la classe, nous expliquerons les choix de modélisation que nous avons pris et finalement nous décrirons dans quelle étape d'un scénario est utilisé ce concept.

Premièrement, nous présenterons notre hiérarchie de classes autour du concept d'événement. Celle-ci est représentée sous forme graphique à la figure A.2 où chaque lien est une propriété de sous-classe. Un total de 37 classes gravitent autour du concept d'événements, et nous les présenterons sommairement.

CompletionStatus Ce concept est utilisé pour modéliser l'état d'achèvement d'une opération. Nous avons modélisé uniquement les cas d'un succès et d'un échec. Par exemple, une lecture d'un fichier a échoué puisque l'utilisateur impliqué n'avait pas les bonnes permissions. Cet événement a un statut d'échec. Cette classe est une énumération qui ne possède que ces deux instances nommées : « failure » et « success ». La classe « CompletitionStatus » n'a pas d'autre classe mère à l'exception de « owl:Thing ». Elle est donc au plus haut niveau de la hiérarchie de classes. Elle possède l'axiome logique d'équivalence qui est défini comme suit : `{failure, success}`. Ces deux instances sont vues comme des singletons, c'est-à-dire

qu'elles sont les seules membres de la classe. Nous avons utilisé les instances de cette classe dans le scénario n° 1 avec les requêtes R011, R012, R025 et R026.

Event Cette classe représente un des quatre concepts fondamentaux de notre ontologie : l'événement. Elle est directement sous-classe de « owl:Thing » donc se situe au plus haut niveau de la hiérarchie. Cette classe possède l'axiome logique d'équivalence qui dit que tout ce qui possède une ou plusieurs estampilles temporelles est de type « Event ». L'expression de cet axiome en LD se fait comme suit : `hasTimestamp some xsd:dateTime`. Cette classe ne possède pas d'individu puisqu'elle est très générale, mais elle est utilisée dans les six règles de transformation suivantes : T004, T005, T018, T021, T052 et T053. À cause de son importance, cette classe est utilisée dans les trois scénarios.

ContinuousEvent Cette classe est sous-classe de la classe Event et représente les événements qui possède une estampille temporelle de début et une de fin. Cette définition est représentée avec son axiome logique d'équivalence en LD : `(hasEndTimestamp exactly 1 xsd:dateTime) and (hasStartTimestamp exactly 1 xsd:dateTime)`. Les instances de cette classe possèdent un événement ponctuel de début et un de fin. Ceci est traduit en tant que propriété de sous-classe définie avec l'expression logique suivante : `(hasEndingEvent some PunctualEvent) and (hasStartingEvent some PunctualEvent)`. Il s'agit d'une des rares classes qui n'est pas utilisée dans notre modélisation, c'est-à-dire qu'elle ne se retrouve dans aucune règle et aucune requête. Nous l'avons modélisée pour aller selon notre intuition d'expert, qui nous dit qu'un événement peut être de deux types en fonction du nombre d'estampilles temporelles qui lui est rattaché.

Session Ce concept est celui d'une session créée par une connexion à proprement parler. Nous voulons représenter la continuité débutant par un événement de connexion et se terminant par un événement de déconnexion. Ainsi, une session est reliée à un événement d'authentification, un événement de désauthentification et un utilisateur. Sa relation de sous-classe est avec la classe ContinuousEvent. L'axiome logique décrivant ces caractéristiques de cette classe est le suivant : `(hasEndingEvent some Deauthentication) and (hasStartingEvent some Authentication) and (hasUser some User)`. Cette classe est instanciée par la règle T033 dont nous proposons une implémentation algorithmique. Elle est utilisée au scénario n° 2 avec les règles T033 et T036 puis avec la requête R033.

PunctualEvent Cette classe est sous-classe de la classe Event. Elle est utilisée pour abstraire les événements ponctuels, soit ceux qui possèdent une seule estampille temporelle.

Ceci se traduit dans l'ontologie par l'axiome d'équivalence suivant : `hasTimestamp exactly 1 xsd:dateTime`. Au même titre que sa classe ContinuousEvent du même niveau, elle n'est jamais utilisée directement puisqu'elle est très générale. Sa raison d'être dans l'ontologie est selon notre expertise. En effet, nous trouvons naturel de représenter les événements selon le nombre d'estampilles temporelles qui y sont associés.

Authentication Cette classe représente une authentification, soit l'action de s'identifier pour accéder à un système, tel qu'un site Web ou une machine. Elle est sous-classe de la classe PunctualEvent et également sous-classe de l'ensemble décrit par l'axiome logique `hasUser some User`. Par ces propriétés, nous définissons qu'une instance de la classe Authentication possède une estampille temporelle et est liée à un utilisateur. Cette classe ne sera pas instanciée directement. Les instanciations se feront sur ses classes enfants, tel qu'il sera présenté plus tard. Elle est utilisée aux scénarios n^{os} 1 et 2 par les requêtes R024, R032 et par la règle T033.

LocalAuthentication Cette classe représente les authentifications effectuées par un utilisateur physiquement sur un poste de travail Windows. Elle est sous-classe de la classe Authentication. Les instances de cette classe sont toujours liées à une machine, sur laquelle se déroule l'authentification. La classe est donc sous-classe de l'ensemble représenté comme suit : `happenedOn some Machine`. Les postes de travail étant sur Windows, nous pouvons définir la classe par tous les événements Windows qui ont l'identifiant 4624 et une propriété « logon type » valant 2. Ceci est exprimé par l'axiome logique d'équivalence suivant : `WindowsEvent and (hasIdentifier value 4624) and (hasLogonType value 2)`. On retrouve cette classe dans le scénario n^{o} 2 où une instance de WindowsEvent devient type de cette classe.

RemoteAuthentication Cette classe est sous-classe de la classe Authentication et de la classe NetworkRequest, qui sera présentée postérieurement. Elle représente les authentifications effectuées à l'aide d'une requête réseau, telle qu'une connexion d'un utilisateur à un service réseau ou à une machine distante. Lors de l'utilisation de l'ontologie pour la détection des trois scénarios, trois instances dans les scénarios n^{os} 1 et 3 sont du type de cette classe. Les requêtes R015, R021 et R056 utilisent la classe qui représente ce concept.

RDPAuthentication Cette classe représente les authentifications distantes effectuées à l'aide du protocole RDP. Elle est sous-classe de la classe NetworkAuthentication. Nous avons représenté son équivalence par l'ensemble des événements du gestionnaire d'événements Win-

dows qui possède l'identification 4624 et qui possède la propriété « logon type » valant 10. Cette relation d'équivalence se traduit par l'axiome logique suivant : `WindowsEvent and (hasIdentifier value 4624) and (hasLogonType value 10)`. Les scénarios n^{os} 1 et 3 utilisent cette classe et cela est traduit avec la requête R055.

SSHAuthentication Cette classe représente les authentifications réseaux effectuées avec le protocole SSH (Secure Shell). Elle est sous-classe de NetworkAuthentication. Cette classe se fait instancier par le senseur S010. Pour notre détection de scénarios, cela se produit lors du scénario n^{o} 3 à travers la requête R057.

VPNAuthentication Cette classe modélise les authentifications effectuées à travers le protocole VPN. Elle est sous-classe de NetworkAuthentication. Le senseur S009 crée les instances de cette classe puis les requêtes R051, R052 et R053 l'utilisent dans le scénario n^{o} 3.

Deauthentication Cette classe modélise les événements de désauthentification qui sont liés à une session. Puisque les instances de cette classe possèdent toujours un utilisateur, elle est sous-classe de `hasUser some User`. Elle est également sous-classe de PunctualEvent. Cette classe est utilisée dans le scénario n^{o} 2 avec la requête R032 et les règles de transformation T033.

LocalDeauthentication Cette classe est l'opposé de la classe LocalAuthentication. Elle est sous-classe de Deauthentication. Elle possède une relation d'équivalence envers la classe représentant les instances de WindowsEvent qui possède un identifiant valant 4634 ainsi qu'une propriété « logon type » valant 2. Celle-ci s'exprime comme suit : `WindowsEvent and (hasIdentifier value 4634) and (hasLogonType value 2)`. Les instances du type de cette classe sont toujours inférées, c'est-à-dire qu'ils ne sont pas directement créés par un senseur. Pour notre détection, ces instances sont utilisées au scénario n^{o} 2.

RemoteDeauthentication Cette classe est sous-classe de Deauthentication et également sous-classe de NetworkRequest. Elle représente l'inverse d'un NetworkAuthentication, soit une déconnexion liée à une session. Elle regroupe plusieurs concepts différents, tels qu'une désauthentification d'une session RDP ou d'une session SSH. Nous n'avons pas modélisé ces classes enfants puisque nous ne les utilisons pas, mais cela aurait été possible. Dans notre détection des trois scénarios, nous n'avons pas utilisé cette classe. Par contre, à cause de la nature d'une session qui est composée d'une authentification et d'une désauthentification, il était naturel de créer cette classe.

FileOperation Cette classe modélise tous les événements qui sont des opérations sur un fichier. Elle est sous-classe de la classe PunctualEvent. Les instances de cette classe ont toujours ces propriétés : possèdent un état d'achèvement de l'opération, sont effectuées sur un ou plusieurs fichiers et représentent un type d'opération de fichier. Par la nécessité que les instances de cette classe possèdent ces relations, cette classe est également sous-classe de l'ensemble représenté par l'axiome logique suivant : `(hasCompletionStatus some CompletionStatus) and (hasFile some File) and (hasOperationType some OperationType)`. Les instances de cette classe peuvent être à la fois inférées et affirmées. À travers notre modélisation des trois scénarios, nous avons utilisé cette classe aux scénarios n^{os} 1 et 3. Elle est également utilisée dans le senseur S003 et dans les requêtes R003, R058 et R059.

FileRead Cette classe est une sous-classe de la classe FileOperation et elle représente toutes les opérations sur des fichiers dont le type d'opération est une opération de lecture. Par sa définition, nous pouvons établir une relation nécessaire et suffisante, soit d'équivalence, traduite par la règle logique suivante : `FileOperation and (hasOperationType value read)`. Les instances de cette classe ne sont pas affirmées : elles sont plutôt inférées dues aux relations de sous-classe. Cette classe est présente dans les requêtes R005 et dans la règle T006.

ExternalDriveFileRead Cette classe est une sous-classe de la classe FileRead. Elle regroupe les opérations de lecture de fichier dont le fichier est sur un support amovible. Ses instances sont toujours de type inféré par une règle. Cette classe est présente dans le scénario n^{o} 1 avec la règle T006 et la requête R006.

RemoteFileAccess Cette classe est sous-classe à la fois de FileOperation et de NetworkRequest. Elle représente les opérations de fichier effectué sur un fichier distant à la machine source. Elle ne possède pas plus de propriétés que ceux offerts par ses ancêtres. Les instances de cette classe ne sont pas affirmées (*asserted*), c'est-à-dire qu'elles ne sont pas créées par un senseur ou configurateur ; elles sont plutôt inférées. Cette classe est utilisée au scénario n^{o} 1 avec les requêtes R012 et R026.

WindowsRemoteFileAccess Cette classe est sous-classe de la classe RemoteFileAccess. Elle contraint l'ensemble de sa classe parente en spécifiant que les accès à des fichiers distants se font sur le système d'exploitation Windows. La classe possède une propriété d'équivalence qui se traduit comme suit : `WindowsEvent and (hasIdentifier value 5140)`. Ainsi, elle est également sous-classe de la classe WindowsEvent. Les instances de cette classe sont inférées et sont présentes au scénario n^{o} 1.

WindowsFileOperation Cette classe représente les opérations sur des fichiers qui se font sur des machines Windows. Elle possède une propriété d'équivalence qui exprime que tous les événements Windows avec un identifiant 4663 font partie de cette classe. L'axiome logique traduisant cette équivalence est le suivant : `WindowsEvent and (hasIdentifier value 4663)`. Les instances de cette classe sont inférées et sont au nombre de trois pour notre modélisation des trois scénarios. Son utilisation se retrouve aux scénarios n^{os} 1 et 3 mais elle n'est jamais instanciée directement puisque nous utilisons ses classes parentes plus abstraites.

MountedPartition Cette classe est sous-classe de PunctualEvent et représente les actions de monter une partition. Les instances de cette classe doivent être reliées à la partition qui est montée et l'événement doit se dérouler sur une machine. Nous traduisons ces nécessités par une relation de sous-classe formulée comme suit : `(happenedOn some Machine) and (hasPartition some Partition)`. Le senseur S002 créé des instances affirmées du type de cette classe. Ce concept est utilisé au scénario n^{o} 1 avec une instance, par la règle T006 et par la requête R004.

WindowsMountedPartition Cette classe est sous-classe de MountedPartition et contraint les événements de sa classe mère en spécifiant qu'ils se produisent dans un contexte de système d'exploitation Windows. Nous pouvons définir cette classe par l'ensemble des événements Windows dont l'identification vaut 98. Cette relation nécessaire et suffisante permet d'établir une relation d'équivalence que nous traduisons comme suit : `WindowsEvent and (hasIdentifier value 98)`. Les instances de la classe WindowsMountedPartition seront toujours inférées et non affirmées comme c'est le cas pour sa classe parente.

NetworkRequest Cette classe est sous-classe de PunctualEvent et elle représente une requête réseau. Ce concept est général puisqu'il regroupe les unités de données de protocole de différentes couches du modèle OSI. Les unités regroupées sont le segment TCP (couche 4), le datagramme UDP (couche 4) et la requête de données (couche 7). Cette classe se définit par tout ce qui possède un protocole, deux adresses IP (une de source et une de destination) et deux ports (un de source et un de destination). Cette équivalence se traduit ainsi : `(hasProtocol some Protocol) and (hasDestinationIPAddress some xsd:string) and (hasSourceIPAddress some xsd:string) and (hasDestinationPort some xsd:integer) and (hasSourcePort some xsd:integer)`. Les instances de la classe sont à la fois inférées et affirmées. C'est le senseur S004 qui créera les instances. La classe représentant ce concept fondamental est utilisée dans les trois scénarios par les requêtes R007 et R043, puis par les règles T008, T009, T010, T011, T015, T050 et T060.

HeartbleedExploit Cette classe représente les exploitations de la vulnérabilité Heartbleed. Cette dernière est également modélisée dans l'ontologie et sera décrite en détail. L'exploitation de cette vulnérabilité est détectable avec une règle d'un NIDS tel Snort. La classe HeartbleedExploit est sous-classe des classes NetworkRequest et SuspiciousEvent, et elle ne possède pas de relation d'équivalence. Elle possède une faible description dans l'ontologie puisque la donnée brute est classifiée au préalable par un senseur intelligent, soit un NIDS. Le pilote lié à ce senseur est S006 et il est utilisé au scénario n° 3 par les requêtes R047 et R050.

HTTPRequest Cette classe modélise une requête réseau utilisant le protocole HTTP. Elle est sous-classe de NetworkRequest. La classe possède une relation d'équivalence avec l'ensemble représente par l'axiome suivant : `NetworkRequest and hasProtocol value HTTP`. Cette relation définit que toutes les instances de NetworkRequest qui ont le protocole HTTP sont également des instances de HTTPRequest. Les instances de cette classe sont reliées à une page Web, possèdent un nom de domaine et possèdent un chemin (*path*). De ce fait, cette classe possède l'axiome de sous-classe suivant : `(hasDestination some Webpage) and (hasDomainName some xsd:string) and (hasPath some xsd:string)`. Les instances du type de cette classe peuvent être affirmées par le senseur S008 ou inférées à l'aide de règles. C'est dans les scénarios n^{os} 1 et 2 que nous utilisons ce concept et plus spécifiquement dans les requêtes R033, R034, R035 et R038, puis dans les règles T030, T031, T032 et T036.

SQLInjectionPayload Cette classe représente une requête HTTP qui a pour but d'exploiter une vulnérabilité d'injection SQL. Elle ne possède pas de caractéristiques différentes qu'une requête HTTP non-malveillante, excepté qu'elle a déjà été classifiée au préalable par un senseur avec traitement, tel qu'un NIDS. Sa classification au préalable est un processus complexe que nous n'avons pas à reproduire dans l'ontologie. Nous n'avons qu'à classifier le résultat de cette classification au préalable. La classe est sous-classe de HTTPRequest et de SuspiciousEvent dû à sa nature malveillante. Le senseur S006 crée les instances du type de cette classe. Elle est utilisée dans le scénario n° 2 par les requêtes R039 et R041.

WebpageRequestByAdmin Cette classe représente les requêtes HTTP effectuées par un utilisateur sur un site Web dont il possède les droits d'administrateur. Les instances de cette classe ne sont jamais instanciées directement, elles sont plutôt inférées. Elle est sous-classe de HTTPRequest et de l'ensemble des choses qui possèdent une source qui est un administrateur. Cette relation de sous-classe s'exprime comme suit : `hasSource some Admin`. Nous avons utilisé cette classe au scénario n° 2, dans la règle T032 et les requêtes R036 et R037.

WebScan Cette classe représente les requêtes HTTP qui sont utilisées pour effectuer un balayage dans le but de trouver une vulnérabilité ou une erreur de configuration sur un site Web. Elles sont sous-classe de HTTPRequest et de SuspiciousEvent. Les instances de cette classe sont également classifiées au préalable par un senseur avec traitement. C'est celui-ci, soit S006, qui instancie la classe WebScan. Cette classe est utilisée dans le scénario n° 2 avec les requêtes R027 et R028.

XSSPayload Cette classe représente les requêtes HTTP qui ont pour but d'exploiter une faille XSS. La modélisation de cette classe est très similaire à la classe SQLInjectionPayload puisque ce sont deux concepts qui sont classifiés au préalable par un senseur avec traitement et ils exploitent une faille d'un site Web. Cette classe est sous-classe de HTTPRequest et de SuspiciousEvent. Le senseur S007 crée des instances du type de cette classe lors du scénario n° 2. Les requêtes R029 et R037 utilisent également cette classe.

MalwareTraffic Cette classe représente du trafic réseau utilisé dans une situation de logiciel malveillant (*malware*). Nous définissons cette classe par toutes les requêtes HTTP qui ont pour destination une machine bannie *blacklisted* et qui utilisent un protocole que nous identifions comme potentiellement malveillant. Ces conditions sont suffisantes et nécessaires pour classifier un individu dans cette classe, ce qui se traduit par l'axiome d'équivalence suivant : `NetworkRequest and (hasDestination some BlacklistedMachine) and (hasProtocol some PotentiallyMaliciousProtocol)`. La classe est également sous-classe de SuspicousEvent puisqu'elle est de nature malveillante. De plus, ses instances sont toujours inférées. Nous l'avons utilisé au scénario n° 1 par la requête R010 et la règle T015.

PortScan Cette classe est sous-classe de NetworkRequest et SuspiciousEvent. Elle représente les balayages de ports, qui sont une technique d'intrusion fréquemment utilisée lors d'une reconnaissance de cibles. Il s'agit d'une classe dont les instances sont classifiées au préalable par un senseur intelligent et qui l'instancie par le senseur S006. Cette classe a été utilisée au scénario n° 3 avec les requêtes R045 et R046.

SensitiveDataAlert Cette classe représente une requête réseau dont le corps contient des informations confidentielles, telles que des numéros d'assurance sociale ou des numéros de carte de crédit. Ce traitement est effectué par un senseur NIDS et est déjà effectué avant le stockage dans l'ontologie. Cette classe est sous-classe de NetworkRequest et de SuspiciousEvent. Le senseur qui crée les instances de cette classe est S006. La classe est utilisée dans la requête R042, soit dans le scénario n° 2.

SuspiciousEvent Cette classe est sous-classe de PunctualEvent et elle regroupe tous les événements qui sont impliqués dans le contexte d'une intrusion. Elle n'agit que de regroupement pour permettre à un utilisateur de l'ontologie d'accéder facilement à ses classes enfants. Si on oublie le problème des faux positifs, les classes enfants de la classe SuspiciousEvent témoignent de la présence d'une intrusion quelconque. Cette classe n'est pas utilisée dans les scénarios, et a été conçue selon l'intuition de l'expert. En effet, notre expertise nous a guidés pour créer cette classe, qui pourrait être utile pour un utilisateur de DIOSE.

WindowsEvent Cette classe représente tous les événements Windows, qui sont la plus petite granularité d'une action ou d'un événement sur une machine Windows. Ceux-ci sont structurés selon une documentation et sont regroupés dans le gestionnaire d'événements Windows. Ainsi, pour les instancier dans l'ontologie, nous n'avons qu'à nous interfacer avec le gestionnaire d'événements Windows. Il est nécessaire qu'une instance de WindowsEvent possède un identifiant numérique. Ceci nous donne la relation de sous-classe représentée selon l'axiome logique suivant : `hasIdentifier some xsd:integer`. Elle est également sous-classe de PunctualEvent et est instanciée par le senseur S001. Pour l'ensemble de nos scénarios, ce senseur a créé 11 instances de WindowsEvent qui se sont fait classifier dans l'ontologie selon leurs propriétés. Les trois scénarios ont utilisé la classe WindowsEvent à travers les requêtes R001, R002, R011, R022, R025 et R054.

LsassMemoryRead Cette classe représente les événements Windows qui effectuent une lecture du processus Lsass. Ce dernier est un processus Windows utilisé pour l'authentification transparente dans les environnements Active Directory. Nous pouvons définir cette classe par l'ensemble des événements Windows qui possède l'identifiant 3065. Ceci nous donne la classe d'équivalence suivante : `WindowsEvent and (hasIdentifier value 3065)`. Elle est également sous-classe de WindowsEvent et de l'ensemble représenté par la règle `happenedOn some Machine`. Cette classe n'est pas instanciée directement, et elle est utilisée au scénario n^{o} 1 par les requêtes R023 et R024.

OperationType Ce concept est utilisé pour représenter le concept du type d'une opération sur un fichier. Nous avons modélisé les opérations de suppression, de lecture et d'écriture sur un fichier. Pour ceci, nous avons créé trois instances statiques nommées « delete », « read » et « write ». Cette classe est une énumération de ces trois instances, telle que représentée par son axiome d'équivalence : `{delete, read, write}`. Étant sous-classe de « owl :Thing », elle est située au plus haut niveau de la hiérarchie de classes. Cette classe est utilisée dans les scénarios n^{os} 1 et 3 par les senseurs S001 et S003, par les requêtes R003, R058 et R059.

Whitelist Cette classe représente un conteneur non ordonné d'instances. Elle est sous-classe « rdf:Bag ». Son instanciation nous permet d'utiliser le concept d'un conteneur pour vérifier si un élément en fait partie. Nous l'avons utilisée lors de la dernière étape du scénario n° 3 pour vérifier si un processus fait partie de la liste des processus sur une certaine liste blanche. À cet effet, nous avons instancié la classe Whitelist pour créer une liste blanche qui peut être remplie d'instances par le configurateur C005. Nous avons utilisé cette classe dans les requêtes R058, R059 et R060.

Deuxièmement, nous ferons la présentation des classes reliées au contexte. Ce concept est quelque peu « fourre-tout » puisque le contexte, tel qu'il est défini dans cette recherche, est indissociable de l'événement. Les classes distinctes reliées au contexte sont au nombre de 25 et leur diagramme de classe sous forme de graphe est présenté en annexe à la figure A.2.

File Avec cette classe, nous représentons le concept d'un fichier. La classe est directement sous-classe de « owl :Thing », donc elle se situe au plus haut niveau de la hiérarchie de classes. Notre modélisation considère qu'un fichier possède toujours un nom et qu'il est toujours sur une partition d'une machine. Ainsi, ceci amène à une relation de sous-classe représentée par l'axiome suivant : `(hasPartition some Partition) and (hasName some xsd:string)`. Les instances du type de la classe File sont créées par le configurateur C002. Il lie les instances créées à la partition sur laquelle les fichiers sont stockés et à leurs noms sous forme de chaîne de caractères. Dans la détection de nos trois scénarios, nous avons créé une seule instance de cette classe et c'était au scénario n° 1. La règle T018 et les règles R012 et R026 utilisent la classe File.

Group Cette classe représente le concept d'un groupe dans un environnement Windows. Cela consiste en un regroupement arbitraire de membres, qui sont des ordinateurs ou des utilisateurs. Par cette définition, nous pouvons définir une relation d'équivalence pour la classe Group. Il en va comme suit : `hasMember some (Machine or User)`. Du fait qu'un groupe possède également un nom, nous établissons une relation de sous-classe avec l'axiome suivant : `hasName some xsd:string`. Cette classe se retrouve au plus haut niveau de la hiérarchie de classes. Les groupes Windows sont créés sporadiquement par le configurateur C004 qui crée les instances. Pour la détection des trois scénarios, nous avons utilisé deux instances de Group : un au premier scénario et un au deuxième. À travers l'ensemble de nos règles et requêtes, cette classe est utilisée par les requêtes R017, R018 et R020, puis par les règles T023, T024, T025 et T027.

GroupPolicy Cette classe représente le concept qu'est une politique de groupe. Ce concept est relié au groupe et contient des privilèges que possède un groupe si ce groupe possède la politique de groupe. Dans le cas de notre recherche, nous n'avons modélisé que les droits d'administrateur comme privilèges décrits dans une politique de groupe. Une instance de GroupPolicy possède toujours des droits administrateurs sur un groupe ou une ressource. Par ce fait, nous établissons la relation nécessaire de sous-classe décrite comme suit en LD : `hasAdminRightsOn some (Group or ManageableResource)`. Cette classe étant une sous-classe directe de « owl :Thing », elle se situe au plus haut niveau de la hiérarchie de classes. À l'instar des instances de la classe Group, ceux de la classe GroupPolicy sont créés par le configurateur C004. Pour nos trois scénarios d'attaque, nous avons créé deux instances de GroupPolicy : une au premier scénario et une au deuxième. Cette classe se retrouve également dans les règles T025 et T027, puis dans les requêtes R016, R018 et R020.

ManageableResource Cette classe représente les ressources matérielles ou logicielles qui sont administrables. Par ceci, nous définissons qu'elles peuvent être administrées par une politique de groupe, ou par un utilisateur. Elle possède donc la relation de sous-classe suivante : `isAdministratedBy some (GroupPolicy or User)`. La classe est au niveau de hiérarchie de classes le plus élevé. À travers nos trois scénarios, cette classe ne s'est pas fait instancier directement. La classe a été plutôt utilisée dans les règles T022 et T025, puis dans les requêtes R016, R018 et R019.

Machine Cette classe représente un appareil informatique branché sur un réseau. Une telle définition peut aussi bien regrouper un ordinateur qu'un téléphone intelligent. Elle est sous-classe de ManageableResource. Les instances de cette classe doivent posséder un système d'exploitation, un nom d'hôte (*hostname*) et une adresse IP. De ce fait, elle a aussi la relation de sous-classe suivante : `(hasOperatingSystem some OperatingSystem) and (hasHostname some xsd:string) and (hasIPAddress some xsd:string)`. Représentant un concept fondamental, cette classe a eu un rôle majeur dans nos scénarios. En effet, elle a été instanciée trois fois par scénarios, pour un total de neuf instanciations. Le configurateur qui crée les instances de ce type est le configurateur C001. Cette classe est utilisée dans les requêtes R006, R010, R012, R013, R014, R015, R017, R020, R021, R023, R024, R026, R030, R037, R043, R045, R051, R053, R055, R056, R057, R058 et R059. Elle a également été utilisée dans les règles T004, T005, T012, T013, T020, T024, T027, T036, T039, T045 et T050.

ExternalMachine Cette classe représente une machine externe au réseau interne de l'entreprise, donc qui est sur le WAN. Elle est sous-classe de Machine. Cette classe se fait instan-

cier par l'algorithme représenté par la règle de transformation T039. Nous procédons de cette façon pour éviter d'avoir à instancier statiquement toutes les machines à l'avance. En utilisant une règle de transformation, la modélisation est dynamique puisque seules les machines du scénario d'attaque sont modélisées. Nous avons utilisé ce concept aux scénarios n^{os} 1 et 2 avec chacune une instance de la classe ExternalMachine. Également, ce concept se retrouvait dans la requête R042 et la règle T014.

BlacklistedMachine Cette classe représente le concept d'une machine sur une liste noire. Une telle machine aurait été identifiée par une entité experte en tant que malveillante. Un exemple d'une telle base de données de machines malveillantes est Google Safe Browsing. Ainsi, en nous fiant à une entité reconnue et externe, nous pouvons utiliser cette information dans l'ontologie de détection. Du fait que ces machines sont toujours sur le réseau étendu, cette classe est sous-classe de ExternalMachine. Elle ne se fait jamais instancier directement, et une instance de ce type a été inférée au scénario n^{o} 1. Cette classe est présente dans la requête R009 puis dans les règles T015 et T039.

Server Cette classe est sous-classe de Machine et elle représente un serveur informatique. Nous la définissons comme étant une machine qui offre un service réseau. De cette définition, nous lui attachons l'axiome d'équivalence suivant : `Machine and (offers some Service)`. Cette classe ne se fait pas instancier par un configurateur. Ce sont plutôt les règles de transformation qui associent des instances existantes à cette classe. Pour la détection de nos trois scénarios, la classe a été instanciée à deux reprises : une au deuxième et une au troisième scénario. La classe Server a également été présente dans les requêtes R044 et R046.

Service Cette classe est sous-classe de ManageableResource et elle représente un service réseau. Comme exemple de Service, on peut penser à un service VPN, un site Web et un service de courriel. Les instances de cette classe possèdent toujours des modules installés, sont associées à un port réseau et sont offertes par une instance de type Serveur. Ces relations nécessaires à la classification se traduit par l'axiome de sous-classe suivant : `(hasModule some Module) and (hasPort some xsd:integer) and (isOfferedBy some Server)`. Puisque cette classe est très générique, elle n'est jamais instanciée directement par un configurateur. Ce sont plutôt ses classes enfants qui sont instanciées, tel que nous le verrons. Dans notre ontologie de détection pour les trois scénarios, nous avons utilisé le concept de Service dans tous les scénarios. Les requêtes qui utilisaient ce concept sont les requêtes R017, R043, R047, R048, R049, R050, R053, R061 et R062. Les règles qui l'utilisent sont les règles T044, T045 et T060.

VPNService Cette classe représente un service réseau qui permet l'utilisation du protocole VPN. Elle est sous-classe de Service, et ne possède pas de propriété supplémentaire. Cette classe se fait instancier directement par le configurateur C001, qui associe les bonnes propriétés aux instances créées. Une instance a été créée lors de notre modélisation des trois scénarios, et ce lors du scénario n° 3. Nous avons utilisé cette classe dans la requête R053.

Website Cette classe modélise le concept d'un site Web. Elle est sous-classe de Service, et est définie comme quelque chose qui possède des pages Web et un nom de domaine. De ce fait, nous établissons l'axiome d'équivalence suivant : `(hasWebpage some Webpage) and (hasDomainName some xsd:string)`. Une instance de Website se fait instancier directement par le configurateur C001, et c'est ce qui a été nécessaire pour notre détection du scénario n° 2. L'utilisation de ce concept se retrouve dans les règles T030, T031, T032 et T038, puis dans les requêtes R027, R031, R035, R037, R038, R039, R040, R041 et R043.

Module Cette classe représente un objet installé sur une machine et utilisé par un service. Par module, nous englobons les librairies système, les plugiciels spécifiques aux services et les logiciels que pourrait vouloir utiliser un service. Ce regroupement très large nous permet de modéliser ces concepts sans avoir à développer une modélisation trop complexe. Cette classe est une sous-classe directe de « owl :Thing ». Puisqu'elle est très générique, cette classe ne se fait jamais instancier directement. Nous ne l'avons pas utilisée dans les requêtes et règles puisque c'est toujours ses classes enfants plus concrètes qui sont utilisées. La classe Module existe uniquement puisqu'il est naturel de regrouper des concepts similaires ensemble.

OpenSSL Cette classe représente la librairie de cryptographie OpenSSL. Étant sous-classe de Module, nous y référerons en tant que module et non en tant que librairie Les versions de cette librairie sont instanciées à une seule reprise et les instances qui l'utilisent réfèrent à cette instance qui agit comme un singleton. Les instances reliées aux versions sont peuplées au préalable dans l'ontologie (par exemple comme ceux des classes CompletionStatus et OperationType). Nous avons créé six instances de ce type soit : openSSL1.0.1a, openSSL1.0.1b, openSSL1.0.1c, openSSL1.0.1d, openSSL1.0.1e et openSSL1.0.1f. Le configurateur C001 vérifie si une instance de Service utilise un module et le cas échéant, il crée une relation pour les lier. Nous avons utilisé la classe OpenSSL au scénario n° 3 avec les requêtes R049 et R062.

WooCommerce Cette classe représente le module complémentaire WooCommerce qui s'installe sur une plateforme Web Wordpress. Il permet d'ajouter à un site Web des fonctionnalités de site Web transactionnel. Cette classe est sous-classe de Module. À la manière de

OpenSSL, nous instancions à une seule reprise chaque version du module complémentaire que nous souhaitons modéliser. Pour notre modélisation, nous avons les instances suivantes : WooCommerce_2.3, WooCommerce_2.3.1, WooCommerce_2.3.2, WooCommerce_2.3.3, WooCommerce_2.3.4 et WooCommerce_2.3.5. À l'instar des autres sous-classes de Module, le configurateur C001 crée le lien entre un module et un service lorsque nécessaire. Nous avons utilisé cette classe au scénario n° 2 pour modéliser un site Web vulnérable.

OperatingSystem Cette classe présente le concept d'un système d'exploitation. Étant une sous-classe directe de « owl :Thing », elle est au plus niveau de la hiérarchie de classes. Nous n'instancions pas directement cette classe, puisque ses classes enfants sont plus spécifiques. La classe OperatingSystem est très générale et elle existe pour regrouper des classes similaires ensemble. Cette classe parente n'est pas utilisée dans les règles et requêtes contrairement à ses enfants.

Linux Cette classe représente le système d'exploitation Linux. Elle est sous-classe de OperatingSystem. Nous avons choisi de modéliser ses instances en tant que singleton, c'est-à-dire qu'il y a une instance par type de système d'exploitation. Les instances peuplées au préalable dans l'ontologie sont nommées « debian » et « ubuntu ». Lors de la création d'une instance Machine, le configurateur C001 associe la machine à la bonne instance de la classe Linux le cas échéant.

Windows Cette classe représente le système d'exploitation Windows et elle est sous-classe de la classe OperatingSystem. À l'instar de la classe Linux, les instances de la classe Windows sont des singletons des différentes versions de l'OS. Pour notre modélisation, nous retrouvons les instances « windows10 », « windows7 », « windows8 », « windowsVista » et « windowsXP ». Le configurateur C001 relie une machine avec l'instance de la classe Windows à sa création lorsque nécessaire. Nous retrouvons cette classe dans la requête R013 au scénario n° 1.

Partition Cette classe représente le concept d'une partition de disque dur. C'est une classe au plus haut niveau de la hiérarchie de classes. Selon notre modélisation, une instance de Partition est toujours située sur une machine et possède toujours un nom. De ce fait, nous établissons l'axiome logique de sous-classe suivant : `(isPartitionOf some Machine) and (hasName some xsd:string)`. Les instances de la classe Partition sont créés par le configurateur C001, qui les associe également aux bonnes instances de Machine. Pour notre détection des trois scénarios, nous avons instancié à deux reprises cette classe au premier scénario. Les

requêtes et règles qui utilisent ce concept sont les suivantes : R004, R005, R006, T004 et T006.

Process Cette classe représente un processus, soit l'exécution d'un programme informatique sur une machine. Elle possède l'axiome de sous-classe `hasName some xsd:string` donc une instance de cette classe possède toujours un nom. Nous avons choisi de les modéliser en singleton, c'est-à-dire qu'une instance d'un processus ne représente pas l'exécution de celui-ci, mais bien le processus lui-même. Ainsi, deux processus ne devraient pas avoir le même nom tant qu'il n'y a pas deux concepts similaires. Les instances de Process peuvent être créées de deux façons. D'une part, le configurateur C005 permet à un utilisateur du système de créer certains processus à la main. Ceux-ci sont des processus qui peuvent être admis dans une liste blanche (classe Whitelist), qui est élaborée à la dernière étape du scénario n° 3. D'autre part, la règle de traduction T052 crée des instances en fonction des propriétés des événements dans l'ontologie. Cette procédure d'instanciation quelque peu incongrue nous permet de ne pas avoir à instancier l'entièreté des processus sur toutes les machines dans l'environnement modélisé, ce qui serait loin d'une solution performante. Pour notre modélisation, nous avons instancié deux processus : « wannaCry » et « windowsDefender ». Cette classe est utilisée par les règles et requêtes R058, R059, R060, T052 et T053.

Protocol Cette classe représente un protocole réseau. Elle peut regrouper les protocoles de toutes les couches du modèle OSI, mais dans notre modélisation il ne s'agit que des couches 4, 5, 6 et 7. N'ayant aucune classe parente que nous avons créée, elle est au plus haut niveau de la hiérarchie de classes. Les instances de cette classe sont des instances statiques qui représentent des protocoles. Dans notre cas, il s'agit de « TCP » et « UDP ». Elle possède plus d'instances, mais celles-ci seront décrites lorsque sa classe enfant sera présentée. Cette classe est utilisée par le senseur S004, par les requêtes R007 et R008 et finalement par les règles T008, T009, T010, T011 et T015.

PotentiallyMaliciousProtocol Cette classe est sous-classe de Protocol et elle représente un protocole réseau qu'un expert a identifié comme potentiellement malveillant. Cette identification ne garantit aucunement qu'une communication réseau effectuée par un protocole de cette classe soit malveillante. Cependant, il requiert une classification particulière due à sa popularité pour des activités de logiciels malveillants. Cette classe est modélisée de la même façon que la classe Protocol. Pour notre modélisation, nous avons instancié les individus suivants : « DNS », « HTTP », « IRC » et « SSL ». Nous utilisons ce concept au scénario n° 1 à l'aide de la règle T015.

User Cette classe représente un utilisateur du système. À cause du fort couplage entre notre modélisation et l'environnement Active Directory, nous établissons que les utilisateurs représentés par cette classe sont des utilisateurs Active Directory. Du fait que ceux-ci possèdent toujours un nom, nous associons l'axiome de sous-classe suivant à la classe User : `hasName some xsd:string`. Les instances de cette classe sont créées par le configurateur C003. Pour la modélisation des scénarios, nous avons utilisé ce concept à tous les scénarios à travers quatre instances. À cause de sa forte présence dans nos scénarios, cette classe se retrouve dans plusieurs requêtes et règles. Les requêtes qui l'utilisent sont les suivantes : R015, R017, R018, R019, R020, R021, R024, R032, R033, R035, R052, R053, R055, R056 et R057. Les règles qui utilisent cette classe sont les suivantes : T021, T023, T025, T032, T033 et T036.

Admin Cette classe est sous-classe de User et représente un utilisateur qui possède des droits d'administrateur. Nous pouvons traduire cette définition par l'axiome logique d'équivalence suivant : `User and (hasAdminRightsOn some ManageableResource)`. Cette classe n'est pas instanciée directement, mais elle possède tout de même des instances créées par inférences. Cette classe est utilisée au scénario n° 2 à travers les requêtes R030 et R031.

Webpage Cette classe représente le concept d'une page Web. Ce concept possède un chemin (*path*) et est contenu par un site Web. De ce fait, l'axiome de sous-classe de cette classe est le suivant : `(isWebpageOf some Website) and (hasPath some xsd:string)`. Cette classe est une sous-classe directe de « owl :Thing » et est instanciée par le configurateur C001 qui lie les instances créées au site Web correspondant. Notre modélisation pour la détection des scénarios a nécessité deux instances, toutes deux pour le deuxième scénario. De plus, les règles et requêtes suivantes ont utilisé cette classe : R027, R028, R029, R035, R036, R037, R038, R039, R041, T030 et T031.

AdminWebpage Cette classe est sous-classe de Webpage et représente une page Web que seuls les administrateurs du site Web ont droit d'accès. Il s'agit généralement des pages Web constituant la section administrative d'un site Web. Comme pour la classe Webpage, le senseur C001 crée les instances de cette classe. Pour la détection du scénario n° 2, nous avons créé deux instances de la classe AdminWebpage. Les requêtes R038, R039 et R041 utilisent cette classe.

Troisièmement, nous présenterons les classes reliées au concept de vulnérabilité informatique. Cette partie de l'ontologie est composée de cinq classes. Le faible nombre de classes s'explique par le fait que nous n'avons pas eu à modéliser beaucoup de vulnérabilité. Ces scénarios d'attaque utilisent chacun une seule vulnérabilité, ce qui explique que la modélisation de

concept est moins étoffée que les concepts d'événements et de contexte. Nous présenterons le graphe de la hiérarchie de classes en annexe à la figure A.1 .

Vulnerability Cette classe représente le concept général d'une vulnérabilité. Elle est au niveau le plus haut de la hiérarchie de classes. Les vulnérabilités modélisées dans l'ontologie sont des instances de cette classe ou de ses classes enfants. Ces instances sont créées au préalable dans l'ontologie. Pour notre modélisation, elle a été instanciée à deux reprises avec des instances nommées « heartbleed » et « mimikatz ». Cette classe est utilisée par les règles T020,T044 et T045 puis par les requêtes R048, R050, R014 et R024.

SQLInjectionVulnerability Cette classe représente le concept de vulnérabilité logicielle permettant une exploitation de type injection SQL. Elle est sous-classe de Vulnerability. Cette classe a été instanciée à une reprise lors de la modélisation du deuxième scénario. Cette instance se nomme « WPVDB-7846 » et conceptualise la vulnérabilité référencée par son nom. La classe SQLInjectionVulnerability est utilisée par les requêtes R040 et R041.

WPVDB-7846VulnerableModule Cette classe modélise le concept d'un module vulnérable à la vulnérabilité dont l'identifiant unique est « WPVDB-7846 ». Elle est sous-classe de WooCommerce, elle-même sous-classe de Module. Elle est modélisée en tant qu'énumération qui décrit quelle version de ce module représente la vulnérabilité. Ainsi, nous établissons que cette classe possède la relation d'équivalence avec l'ensemble suivant : `{WooCommerce_2.3, WooCommerce_2.3.1, WooCommerce_2.3.2, WooCommerce_2.3.3, WooCommerce_2.3.4, WooCommerce_2.3.5}`. La règle T038 utilise cette classe au deuxième scénario.

HeartbleedVulnerableModule Cette classe représente le concept d'un module vulnérable à la vulnérabilité Heartbleed. Cette classe est sous-classe de la classe OpenSSL, elle-même sous-classe de Module. Nous avons modélisé cette classe de la même façon que la classe WPVDB-7846VulnerableModule, c'est-à-dire en tant qu'énumération. Ainsi, la classe représente les versions du module qui sont vulnérables à la vulnérabilité Heartbleed. La classe HeartbleedVulnerableModule est équivalente à l'ensemble suivant : `{openSSL1.0.1a, openSSL1.0.1b, openSSL1.0.1c, openSSL1.0.1d, openSSL1.0.1e, openSSL1.0.1f}`. Elle est utilisée dans le scénario n° 3 par la requête R061 et par la règle T044.

MimikatzVulnerableOS Cette classe est sous-classe de OperatingSystem et elle représente le concept d'un système d'exploitation vulnérable à l'exploitation Mimikatz. À l'instar

des autres classes représentant des composantes vulnérables, cette classe est une énumération. Elle est équivalente à l'ensemble des instances suivantes : `{windows7, windows8, windowsVista, windowsXP}`. Par cette équivalence, il est inféré que cette classe est également sous-classe de la classe Windows. La classe MimikatzVulnerableOS est utilisée dans le premier scénario par les règles de transformation T020.

Pour alléger la présentation des classes, nous avons omis de parler des règles d'exclusion mutuelle des classes. Lorsque nous affirmons qu'une classe A est disjointe d'une classe B, nous énonçons qu'une instance de la classe A ne peut être également instance de la classe B. Cette propriété est particulièrement importante dans notre ontologie puisque nous avons des classes qui sont uniquement décrites par leurs relations d'exclusion mutuelle. Sans celles-ci, la classe est vide de propriété, ce qui est une mauvaise pratique. Le tableau A.3 en annexe présente les disjonctions de chaque classe.

Les propriétés

Pour notre modélisation, nous avons créé plusieurs propriétés d'objets et de données. Nous en présenterons quelques-unes à titre d'exemple. Elles se passent de description puisque leur nom les explicite bien. Pour structurer la présentation, nous faisons généralement la distinction envers le type des propriétés : les propriétés d'objet, qui relient deux individus, et les propriétés de données, qui relient un individu à un littéral.

Les propriétés d'objets

- hasAdminRightsOn
- hasCompletionStatus
- hasDestination
- hasEndingEvent
- hasGroup
- hasOperatingSystem
- isOfferedBy

Les propriétés de données

- happenedOnHostname
- hasSourceIPAddress
- hasHostname
- hasIdentifier

— hasProcessName
— hasDestinationPort
— hasTimestamp
— hasUsername

Nous présentons en annexe l'ensemble des propriétés que nous avons créées. Elles sont présentées avec leurs sous-propriétés, leur domaine, leur portée et leurs caractéristiques logiques. Le tableau A.1 présente les propriétés d'objets, tandis que le tableau A.2 présente les propriétés de données.

Les règles SWRL

Nous avons développé 28 règles SWRL associées à notre modélisation. Elles sont utilisées pour enrichir l'ontologie en inférant une propriété supplémentaire. L'ensemble des règles SWRL que nous avons créées est présenté en annexe au tableau A.4. Nous n'en présenterons que quelques-unes dans la section courante.

La règle 4.1, également nommée « T013 », est utilisée pour relier un événement réseau à la machine qui l'a généré. Ce lien est inféré en effectuant une correspondance entre l'adresse IP de destination de l'événement et l'adresse IP de la machine source.

$$\begin{aligned} &NetworkEvent(?event) \wedge hasSourceIPAddress(?event, ?x) \wedge \\ &hasIPAddress(?machine, ?x) \rightarrow hasSource(?event, ?machine) \end{aligned} \tag{4.1}$$

La règle 4.2, également nommée « T020 », permet de créer l'inférence qu'une machine est vulnérable à une exploitation Mimikatz. Cette règle vérifie si une machine possède un système d'exploitation qui est vulnérable à cette exploitation. Le cas échéant, le lien est créé.

$$\begin{aligned} &Machine(?machine) \wedge hasOperatingSystem(?machine, ?os) \wedge \\ &MimikatzVulnerableOS(?os) \rightarrow hasVulnerability(?machine, mimikatz) \end{aligned} \tag{4.2}$$

La règle 4.3, également nommée « T025 », permet de modéliser un utilisateur en fonction de ses groupes. Le lien inféré est d'un utilisateur à une ressource administrable, soit une machine ou un service. L'inférence est faite lorsque le groupe, duquel fait partie l'utilisateur, possède une politique de groupe qui possède des droits privilégiés sur la ressource administrable.

$$hasMember(?group, ?user) \wedge User(?user) \wedge$$
$$hasGroupPolicy(?group, ?policy) \wedge hasAdminRightsOn(?policy, ?target) \wedge \quad (4.3)$$
$$ManageableResource(?target) \rightarrow hasAdminRightsOn(?user, ?target)$$

La règle 4.4, également nommée « T060 », relie un événement réseau au service qu'il cible. Une correspondance est faite entre le port de destination de la requête et le port sur lequel écoute le service. Si cette correspondance s'avère et il y a une correspondance sur l'adresse IP, l'inférence est effectuée.

$$NetworkRequest(?request) \wedge hasDestinationPort(?request, ?port) \wedge$$
$$hasDestinationIPAddress(?request, ?dstip) \wedge hasIPAddress(?machine, ?dstip) \wedge$$
$$offers(?machine, ?service) \wedge hasPort(?service, ?port) \rightarrow hasDestination(?request, ?service) \quad (4.4)$$

4.2.3 Machine à état

Après avoir présenté la modélisation ontologique conçue, nous présenterons la deuxième partie de DIOSE, soit la machine à état. Celle-ci utilise la modélisation et permet à un utilisateur de s'en servir. Notons que, comme exposé à la description de DIOSE au chapitre précédent, il est possible d'effectuer directement des requêtes SPARQL sur l'ontologie peuplée. Ce mode d'utilisation n'est pas mauvais, mais nous proposons une utilisation plus proche du langage de l'expert.

Pour alléger la présentation des étapes modélisées en requêtes SPARQL, nous effectuerons un raccourci. Nous omettons les deux premières lignes des requêtes. Ces lignes associent les mots-clés à l'emplacement des espaces de noms et sont constantes à travers toutes les requêtes. Elles définissent l'emplacement des ressources que nous utilisons dans nos requêtes. La première étant celle comprenant la syntaxe de RDF et la deuxième étant notre ontologie développée. Ces deux lignes sont les suivantes :

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://www.polymtl.ca/ontologies/siem#>

Détection du scénario n° 1

Première étape Cette étape est constituée de deux événements : une machine monte une partition puis une lecture de fichier est effectuée. Ce fichier se trouve sur la partition montée. Nous détectons cette étape avec la requête SPARQL R006. L'état représentant cette requête ne possède aucune entrée et possède trois sorties (?machine, ?event et ?partition). Le corps de cette requête qui permet la détection de la première étape du scénario n° 1 est présenté à la requête 4.5.

```
SELECT * WHERE {
        ?event rdf:type :ExternalDriveFileRead ;
                :happenedOn ?machine ;
                :hasPartition ?partition
}
```
(4.5)

Deuxième étape Cette étape est une requête réseau effectuée à partir de la machine de l'étape n° 1 et qui cible une machine dont l'adresse IP est sur une liste noire. Ce trafic réseau est identifié comme potentiellement malveillant à cause du protocole qu'il utilise. Nous détectons cette étape avec la requête SPARQL R010. L'état représentant cette requête possède une entrée sur la variable ?machine1 et trois sorties (?machine1, ?machine2 et ?event). Le corps de cette requête qui permet la détection de la deuxième étape du scénario n° 1 est présenté à la requête 4.6.

```
SELECT * WHERE {
        ?event rdf:type :MalwareTraffic ;
                :hasSource ?machine1 ;
                :hasDestination ?machine2 .
        ?machine1 rdf:type :Machine .
        ?machine2 rdf:type :Machine
}
```
(4.6)

Troisième étape Cette étape est une tentative de lecture de fichier sur une partition distante. La lecture échoue à cause de problèmes de privilèges. L'événement provient de la machine de l'étape n° 2. Nous détectons cette étape avec la requête SPARQL R012. L'état représentant cette requête possède une entrée sur la variable ?machine et trois sorties (?machine, ?file et ?event). Le corps de cette requête qui permet la détection de la troisième étape du scénario n° 1 est présenté à la requête 4.7.

```
SELECT * WHERE {
        ?event rdf:type :RemoteFileAccess ;
                :hasFile ?file ;
                :hasCompletionStatus :failure ;
                :hasSource ?machine .
        ?machine rdf:type :Machine
}
```
(4.7)

Quatrième étape Cette étape représente une exploitation Mimikatz qui est composée de deux événements dans un contexte précis. D'une part, un utilisateur-administrateur se connecte à la machine de l'étape n° 3. D'autre part, une lecture de la mémoire du processus « Lsass.exe » est effectuée sur cette même machine. Qui plus est, la machine est vulnérable à la vulnérabilité associée à l'exploitation Mimikatz. Nous détectons cette étape avec la requête SPARQL R024. L'état représentant cette requête possède une entrée sur la variable ?machine et quatre sorties (?machine, ?event1, ?event2, ?user). Le corps de cette requête qui permet la détection de la quatrième étape du scénario n° 1 est présenté à la requête 4.8.

```
SELECT * WHERE {
    ?machine :hasVulnerability :mimikatz ;
    ?event1 rdf:type :Authentication ;
            :happenedOn ?machine ;
            :hasUser ?user ;
            :hasTimestamp ?time1 .
    ?user :hasAdminRightsOn ?machine .
    ?event2 rdf:type :LsassMemoryRead ;
            :happenedOn ?machine ;
            :hasTimestamp ?time2
    FILTER (?time1 < ?time2)
}
```
(4.8)

Cinquième étape Cette étape est identique à celle de la troisième étape à l'exception que la lecture a réussie plutôt qu'échouée. Le fichier destiné par l'événement de lecture ainsi que la machine source de l'événement sont les mêmes. Nous détectons cette étape avec la requête SPARQL R026. L'état représentant cette requête possède deux entrées sur les variables ?machine et ?file et trois sorties (?file, ?machine et ?event). Le corps de cette requête qui permet la détection de la cinquième et dernière étape du scénario n° 1 est présenté à la requête 4.9.

```
SELECT * WHERE {
    ?event rdf:type :RemoteFileAccess ;
           :hasFile ?file ;
           :hasCompletionStatus :success ;
           :hasSource ?machine .
    ?machine rdf:type :Machine
}
```
(4.9)

Machine à état du scénario nº 1 À la suite de la présentation des cinq étapes qui forment le scénario, nous pouvons présenter la schématisation de la machine à état. Celle-ci est représentée à la figure 4.4.

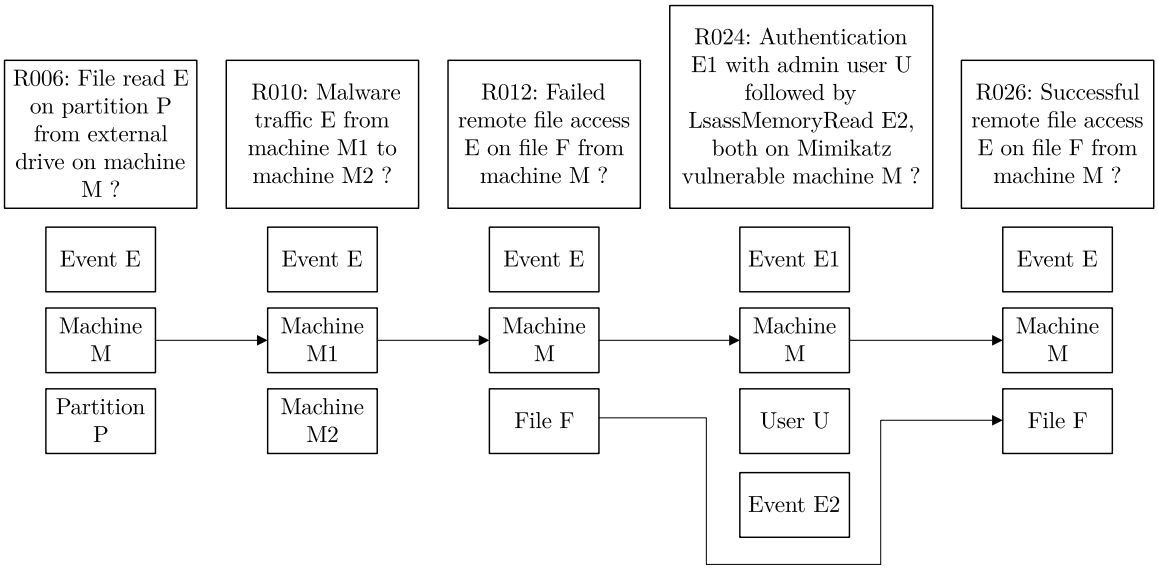

Figure 4.4 Machine à état modélisant le scénario nº 1

À la figure 4.4, nous pouvons voir les cinq états composés d'une requête SPARQL, des sorties et des entrées s'il y en a. Nous pouvons observer qu'une machine est commune à toutes les étapes de ce scénario. Il en va de soi, puisque ce scénario est constitué de l'infection d'une machine. À l'exception de l'étape nº 1, toutes les étapes de ce scénario nécessitent une corrélation. La raison de ceci est que toutes ces étapes possèdent au moins une entrée.

Détection du scénario nº 2

Première étape Cette étape est un événement de balayage de vulnérabilités Web envers un site Web. Nous détectons cette étape avec la requête SPARQL R027. L'état représentant cette requête ne possède aucune entrée et possède quatre sorties (?website, ?machine, ?event et ?webpage). Le corps de cette requête qui permet la détection de la première du scénario nº 2 est présenté à la requête 4.10.

```
SELECT * WHERE {
        ?event rdf:type :WebScan ;
                :hasDestination ?webpage ;
                :hasSource ?machine .
        ?machine rdf:type :Machine .
        ?webpage rdf:type :Webpage ;
                :isWebpageOf ?website
}
```
(4.10)

Deuxième étape Cette étape est une attaque de vol de témoins *cookies* par l'exploitation d'une faille XSS sur le site Web de l'étape n° 1. Elle se décompose en deux parties. Premièrement, une infection XSS d'une page Web est détectée, puis deuxièmement un administrateur du site Web où est hébergée la page Web accède à la page Web infectée. Nous détectons cette étape avec la requête SPARQL R037. L'état représentant cette requête possède une entrée sur la variable ?website et cinq sorties (?event1, ?event2, ?webpage, ?website et ?machine). Le corps de cette requête qui permet la détection de la deuxième étape du scénario n° 2 est présenté à la requête 4.11.

```
SELECT * WHERE {
        ?event1 rdf:type :XSSPayload ;
                :hasDestination ?webpage ;
                :hasTimestamp ?time1 ;
                :hasSource ?machine .
        ?event2 rdf:type :WebpageRequestByAdmin ;
                :hasDestination ?webpage ;
                :hasTimestamp ?time2 .
        ?webpage rdf:type :Webpage .
        ?website :hasWebpage ?webpage
        FILTER (?time1 < ?time2)
}
```
(4.11)

Troisième étape Cette étape est constituée d'une requête HTTP effectuée par la machine de l'étape n° 2 sur une page Web de la section administrative du site Web de l'étape n° 2. Nous détectons cette étape avec la requête SPARQL R038. L'état représentant cette requête possède deux entrées sur les variables ?machine et ?website et quatre sorties (?website, ?webpage, ?machine et ?event). Le corps de cette requête qui permet la détection de la troisième étape du scénario n° 2 est présenté à la requête 4.12.

```
SELECT * WHERE {
        ?event rdf:type :HTTPRequest;
                :hasSource ?machine;
                :hasDestination ?webpage .
        ?website :hasWebpage ?webpage .
        ?webpage rdf:type :AdminWebpage
}
```
(4.12)

Quatrième étape Cette étape est constituée de deux parties. D'une part, un vecteur d'injection SQL est détecté sur une page Web de la section administrative. D'autre part, cette page Web est sur le site Web de l'étape n° 3 qui possède une vulnérabilité permettant l'injection SQL. Nous détectons cette étape avec la requête SPARQL R041. L'état représentant cette requête possède une entrée sur la variable ?website et cinq sorties (?event, ?website, ?machine, ?webpage et ?vuln). Le corps de cette requête qui permet la détection de la quatrième étape du scénario n° 2 est présenté à la requête 4.13.

```
SELECT * WHERE {
        ?event rdf:type :SQLInjectionPayload ;
                :hasSource ?machine ;
                :hasDestination ?webpage .
        ?webpage rdf:type :AdminWebpage .
        ?website :hasWebpage ?webpage ;
                 :hasVulnerability ?vuln .
        ?vuln rdf:type :SQLInjectionVulnerability
}
```
(4.13)

Cinquième étape Cette étape est un événement de détection de transmission de données confidentielles sur le réseau. Il destine une machine externe et a comme source la machine de l'étape n^{o} 4. Nous détectons cette étape avec la requête SPARQL R042. L'état représentant cette requête possède une entrée sur la variable ?machine2 et possède trois sorties (?machine1, ?machine2 et ?event). Le corps de cette requête qui permet la détection de la cinquième et dernière étape du scénario n^{o} 2 est présenté à la requête 4.14.

```
SELECT * WHERE {
        ?event rdf:type :SensitiveDataAlert ;
                :hasDestination ?machine1 ;
                :hasSource ?machine2 .
        ?machine1 rdf:type :ExternalMachine
}
```
(4.14)

Machine à état du scénario n^{o} 2 À la suite de la présentation des cinq étapes qui forment le scénario, nous pouvons présenter la schématisation de la machine à état. Celle-ci est représentée à la figure 4.5.

À la figure 4.5, nous pouvons voir les cinq états représentés par une requête SPARQL, les sorties et les entrées s'il y en a. Nous pouvons observer qu'un site Web est commun à quatre étapes sur cinq. C'est bien normal puisque ce site Web est ciblé par l'acteur malveillant

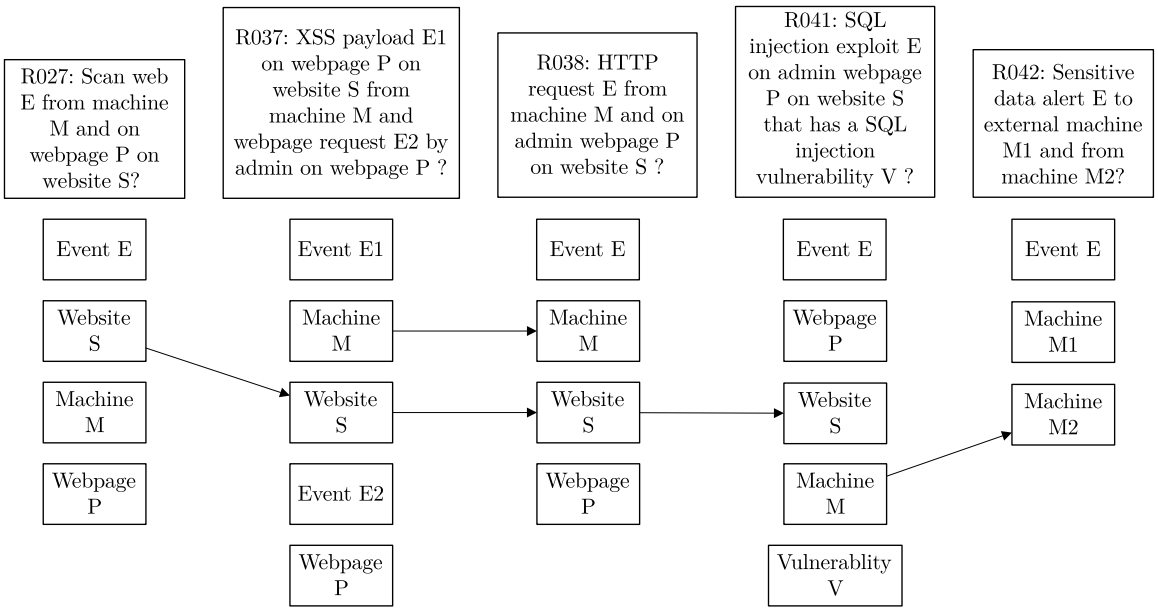

Figure 4.5 Machine à état modélisant le scénario n° 2

du scénario. À l'exception de l'étape n° 1, toutes les étapes de ce scénario nécessitent une corrélation. La raison de ceci est que toutes ces étapes possèdent au moins une entrée.

Détection du scénario n° 3

Première étape Cette étape est un balayage de port destinant une machine de type serveur. Nous détectons cette étape avec la requête SPARQL R048. L'état représentant cette requête ne possède aucune entrée et possède deux sorties (?machine et ?event). Le corps de cette requête qui permet la détection de la première étape du scénario n° 3 est présenté à la requête 4.15.

```
SELECT * WHERE {
        ?event rdf:type :PortScan ;
                :hasDestination ?machine .
        ?machine rdf:type :Server
}
```
(4.15)

Deuxième étape Cette étape est constituée d'une exploitation Heartbleed qui cible un service vulnérable à la vulnérabilité du même nom. Ce service est offert par la machine de l'étape n° 1. Nous détectons cette étape avec la requête SPARQL R050. L'état représentant cette requête possède une entrée sur la variable ?attackerMachine et quatre sorties (?attackerMachine, ?event, ?service, ?machine). Le corps de cette requête qui permet la détection de la deuxième étape du scénario n° 3 est présenté à la requête 4.16.

```
SELECT * WHERE {
        ?event rdf:type :HeartbleedExploit ;
                :hasSource ?attackerMachine ;
                :hasDestination ?service .
        ?service :hasVulnerability :heartbleed ;
                  rdf:type :Service .
        ?machine :offers ?service
}
```
(4.16)

Troisième étape Cette étape est une authentification VPN d'un utilisateur compromis sur le service vulnérable qui a été la cible dans l'étape n° 2. Nous détectons cette étape avec la requête SPARQL R053. L'état représentant cette requête possède deux entrées sur les variables ?machine et ?service et quatres sorties (?user, ?service, ?machine et ?event). Le corps de cette requête qui permet la détection de la troisième étape du scénario n° 3 est présenté à la requête 4.17.

```
SELECT * WHERE {
        ?event rdf:type :VPNAuthentication ;
                :hasUser ?user ;
                :hasSource ?machine ;
                :hasDestination ?service .
        ?service rdf:type :VPNService .
        ?machine rdf:type :Machine
}
```
(4.17)

Quatrième étape Cette étape représente une authentification distante sur une machine effectuée par l'utilisateur de l'étape nº 3. Nous détectons cette étape avec la requête SPARQL R056. L'état représentant cette requête possède une entrée sur la variable ?user et trois sorties (?user, ?machine, ?event). Le corps de cette requête qui permet la détection de la quatrième étape du scénario nº 3 est présenté à la requête 4.18.

```
SELECT * WHERE {
        ?event rdf:type :RemoteAuthentication ;
                :hasUser ?user ;
                :hasDestination ?machine .
        ?machine rdf:type :Machine
}
```
(4.18)

Cinquième étape Cette étape est une infection par rançongiciel (Moore, 2016). Elle est constituée d'une succession d'opérations d'écriture sur la machine à laquelle un utilisateur s'est connecté à l'étape nº 4. Ces événements sont générés par un processus inhabituel. Nous pouvons détecter cette infection à cause d'un nombre élevé d'opérations d'écriture. Ce nombre est un choix arbitraire et nous l'avons fixé à 100 (Olszewski, 2015). Le nombre d'événements à détecter a été implémenté avec les mots-clés « GROUP BY » et « HAVING » du langage SPARQL. Nous détectons cette étape avec la requête SPARQL R059. L'état représentant cette requête possède une entrée sur la variable ?machine et possède une sortie (?machine). Cette requête pour la cinquième et dernière étape du scénario nº 3 est présentée à la requête 4.19.

```
SELECT ?machine WHERE {
        ?event rdf:type :FileOperation ;
                :hasOperationType :write ;
                :happenedOn ?machine ;
                :hasProcess ?process .
        ?machine rdf:type :Machine .
        ?process rdf:type :Process .
        :whitelistedWriteProcesses rdf:li ?process2
        FILTER (?process != ?process2)
}
GROUP BY ?machine
HAVING (COUNT(?event) > 100)
```
(4.19)

Machine à état du scénario nº 3 À la suite de la présentation des cinq étapes qui forment le scénario, nous pouvons présenter la schématisation de la machine à état. Celle-ci est représentée à la figure 4.6.

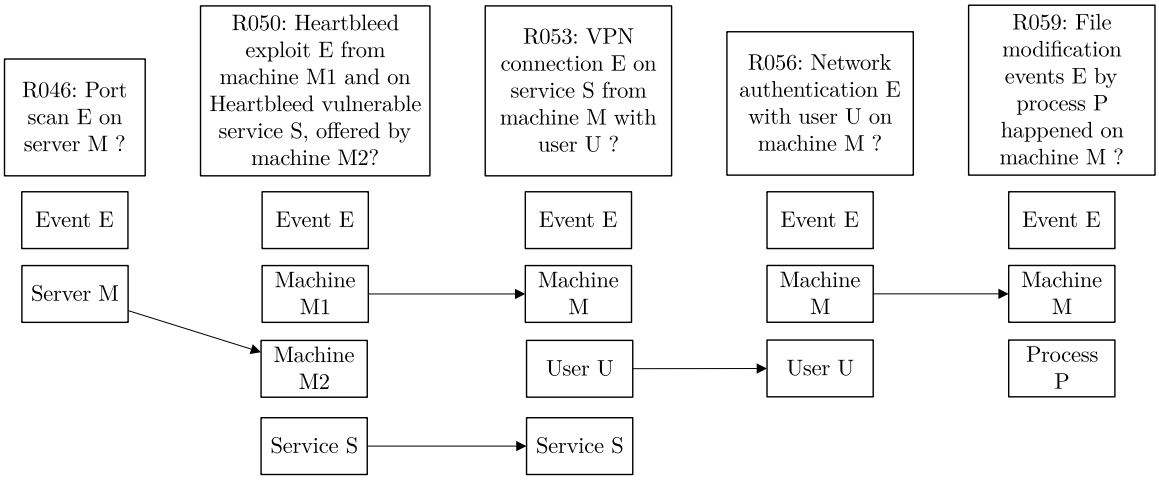

Figure 4.6 Machine à état modélisant le scénario nº 3

À la figure 4.6, nous pouvons voir les cinq états représentés par une requête SPARQL, les sorties et les entrées s'il y en a. Nous pouvons observer qu'aucune entité n'est commune à

travers la majorité des étapes. Cela fait de ce scénario un bon exemple de corrélation puisque la variable qui lie deux étapes change souvent. À l'instar des deux autres scénarios, toutes les étapes, à l'exception de la première, possèdent au moins une entrée.

Requêtes supplémentaires

Lors de la création de l'ontologie pour la détection des scénarios, nous avons créé plusieurs autres requêtes SPARQL. Celles-ci n'ont pas été présentées puisqu'elles sont généralement plus proches des données, ce qui n'aide pas à détecter les scénarios. Néanmois, elles constituent des résultats importants puisqu'un utilisateur peut souhaiter les utiliser. Prenons l'exemple de la requête R021 présentée en 4.20. Elle permet d'obtenir les authentifications d'un utilisateur sur une machine sur laquelle l'utilisateur a les droits d'administrateur. Cette requête n'a pas été présentée pour la détection des scénarios puisqu'elle a été utilisée pour construire la requête R024 au scénario n° 1. Malgré qu'elle soit assez concrète, elle n'en demeure pas moins utilisable. Notre expertise nous a guidés pour sa création puisque nous pensons qu'elle pourrait être utile à un utilisateur de DIOSE. L'ensemble des requêtes que nous avons créées est présenté en annexe au tableau B.1.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://www.polymtl.ca/ontologies/siem#>

SELECT * WHERE {
    ?event rdf:type :RemoteAuthentication ;
        :happenedOn ?machine ;
        :hasUser ?user .
    ?machine rdf:type :Machine .
    ?user :hasAdminRightsOn ?machine
}
```
(4.20)

CHAPITRE 5 ÉVALUATION DE VIABILITÉ

À la suite de la description du modèle proposé, nous étudierons la viabilité de notre SIEM. Initialement, des prémisses doivent être énoncées. Celles-ci sont des conditions qui doivent être validées pour que notre solution proposée fonctionne. Elles restreignent l'utilité de DIOSE, mais nous prétendrons que malgré celles-ci, la solution demeure utilisable. Par la suite, nous discuterons de la flexibilité de notre approche à l'aide de données récoltées lors du développement de DIOSE. Nous poursuivrons ce chapitre en décrivant une utilisation concrète de notre solution d'un point de vue de sécurité informatique. Cette section concernera les patrons d'attaques connues. Finalement, nous conclurons avec une section qui décrira des cas d'utilisation en fonction de différents acteurs potentiels de DIOSE. Cette dernière partie agit en tant que guide pratique d'utilisation et permet d'exposer la simplicité des différents acteurs ayant à utiliser DIOSE.

5.1 Suppositions

Bien que nous ayons proposé une solution purement théorique, nous établissons des contraintes liées à son utilisation. Ces contraintes sont des prémisses qui simplifient le travail de développement du DIOSE.

5.1.1 Mode de fonctionnement des senseurs

La première contrainte est reliée au mode de fonctionnement des senseurs avec traitement. En effet, ce type de senseurs a généralement deux modes de fonctionnement : un mode prévention et un mode détection. Dans le premier cas, lorsqu'une alarme est générée, la donnée qu'elle concerne est bloquée. Par exemple, un paquet réseau analysé comme malveillant est bloqué, ou un courriel analysé comme du pourriel est jeté à la corbeille. Une action réactive est faite après l'analyse pour tenter de nuire à la propagation de la menace. C'est ce qui le distingue du deuxième type, soit le mode détection, dans lequel il n'y a pas d'action qui est effectuée après l'analyse. Par exemple, un IDS détecte une possible exploitation réseau et génère une alerte, mais laisse passer le paquet.

Le mode d'opération des senseurs est une caractéristique principale de ceux-ci. En fonction de ce mode, les senseurs peuvent changer de nom. Par exemple, en parlant d'IDS, on dit que l'outil est en mode détection, et en parlant d'Intrusion Prevention System (IPS), il est en mode prévention.

Notre première prémisse sera que les senseurs sont tous utilisés en mode détection, et non en mode prévention. Nous pouvons faire ce postulat sans contraindre la portée de notre solution puisque sur le terrain, la plupart des senseurs sont en mode détection. Le mode prévention semble offrir une meilleure performance de sécurité en éradiquant immédiatement une menace. Cependant, il est une moins bonne solution à cause du volume trop élevé de faux positifs générés. En effet, un faux positif généré par un senseur en mode détection va utiliser des ressources pour l'analyste de sécurité qui doit traiter inutilement cette alerte. Dans le cas d'un faux positif généré par un senseur en mode prévention, l'information légitime sera bloquée ce qui va impacter grandement le système. Ainsi, l'impact est moins important lorsque le senseur est en mode détection et qu'il ne bloque pas l'événement générant une alerte de type faux positif.

5.1.2 Les faux négatifs

Notre deuxième contrainte concerne la détection d'attaques aux événements multiples. Nous supposons que toutes les informations ont été collectées. DIOSE permet une corrélation sur des données brutes, telles que des événements collectés par des senseurs. Si ce type d'informations n'est pas collecté par les senseurs pour une raison quelconque, nous ne pouvons l'utiliser pour la corrélation. Dans ce cas, il s'agit d'un problème relié au senseur et il sort du spectre de cette recherche. Par exemple, un balayage de port pour trouver un service vulnérable sur une machine peut passer sous le radar d'un IDS en ajoutant des délais lors du balayage. Également, il est possible qu'une défaillance survienne dans le système de collection des données et que certaines données soient perdues ou non accessibles. Nous ne couvrons pas ces cas et nous postulons que tout est disponible, dans la mesure de ce qui est raisonnable.

5.2 Flexibilité

Lors du développement du système expert que nous proposons pour aider le processus de détection d'intrusion, nous nous sommes intéressés à la flexibilité de cette approche. Nous avons démontré que DIOSE pouvait détecter les trois scénarios décrits, mais la réalité n'est pas composée seulement de ces trois scénarios : il en existe une multitude. Pour détecter ces scénarios, il est nécessaire de les modéliser dans DIOSE de la même façon que nous avons modélisé les trois scénarios exposés. À l'aide de métriques récoltées lors du développement de notre ontologie de détection, nous tenterons d'extrapoler l'effort de modélisation de futurs scénarios. Par la suite, nous discuterons de la flexibilité de notre approche par rapport à la maintenabilité de DIOSE.

5.2.1 Effort de développement

Nous voulions évaluer l'effort relié au développement d'une ontologie pour la détection d'un scénario. Pour cette tâche, nous nous sommes intéressés à la métrique du nombre d'axiomes d'une ontologie. Dans un tel système de stockage de la connaissance, tout ce qui est stocké est représenté en tant qu'axiome. Nous présentons l'énumération suivante qui représente une liste non exhaustive des opérations qui affecte cette métrique. Notez bien que pour alléger la lecture, nous considérons que toutes ces opérations sont des ajouts ou des suppressions des différents concepts listés.

— classe
— relation de classe d'équivalence et de sous-classe
— règle SWRL
— instance de classe
— relation de disjonction
— propriété de données ou d'objets
— domaine et portée d'une propriété
— caractéristique d'une propriété

Ainsi, toutes les modifications que nous effectuons sur une ontologie affectent le nombre d'axiomes. Pour cette raison, nous avons choisi cette métrique pour représenter l'effort de développement. Nous sommes conscients que ce dernier n'est pas représenté parfaitement par le nombre d'axiomes. En effet, l'ajout d'une règle SWRL ne représente pas nécessairement le même effort que l'ajout d'une caractéristique à une propriété. Il s'agit néanmoins de la meilleure métrique que nous avons identifiée. De plus, puisque nous avançons que nos scénarios sont similaires en ce qui concerne les métriques, nous amenuisons cette faiblesse.

Au cours de notre développement, nous avons noté le nombre d'axiomes à la fin de chaque modélisation d'un scénario. Cela nous a donné trois valeurs présentées au tableau 5.1.

Tableau 5.1 Nombre d'axiomes dans l'ontologie à la fin de chaque scénario

Scénario	Nombre total d'axiomes
n° 1	327
n° 2	504
n° 3	663

En effectuant la différence entre ces valeurs et leurs précédentes, nous pouvons découvrir des relations entre le nombre d'axiomes et l'effort de développement. Nous présentons le graphique à la figure 5.1 qui exprime le nombre d'axiomes ajouté à chaque scénario.

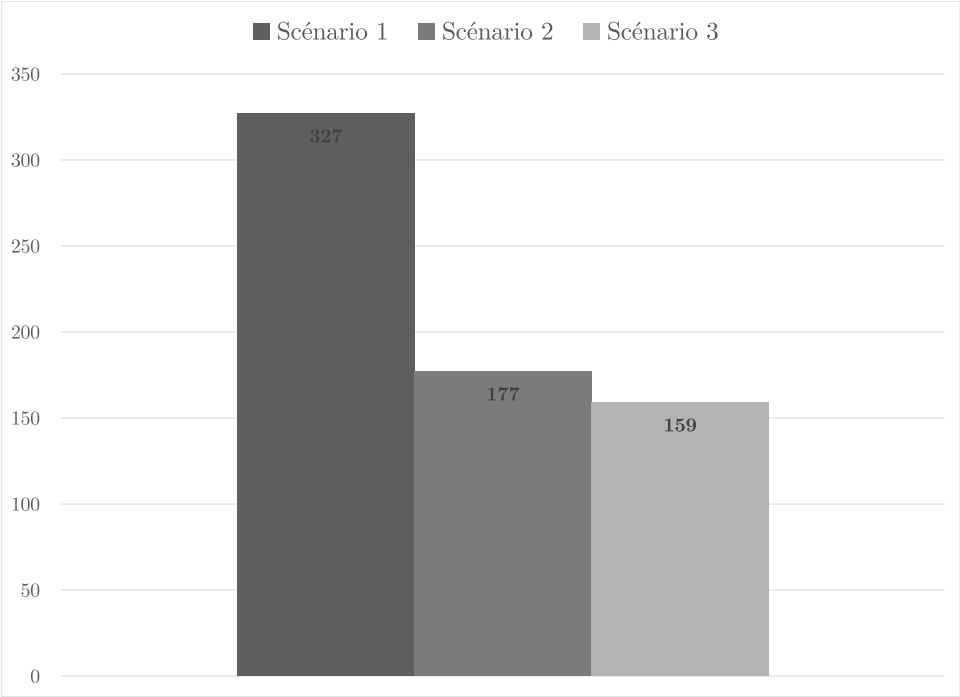

Figure 5.1 Nombre d'axiomes introduits en fonction du scénario

À la lumière du graphique de la figure 5.1, nous pouvons tirer deux conclusions. Premièrement, le nombre d'axiomes introduits à chaque nouveau scénario décroît. Deuxièmement, selon la tendance que nous percevons, un plancher sera éventuellement atteint. Ainsi, nous prévoyons que la modélisation d'un nouveau scénario nécessitera toujours un certain nombre d'axiomes. Cependant, ce nombre devrait être moins grand que le nombre d'axiomes ajoutés lors de la modélisation du premier scénario, et ainsi de suite.

Le nombre d'axiomes introduits étant relié à la quantité d'effort à déployer, nous affirmons qu'il sera plus facile de modéliser le quatrième scénario, ainsi que le cinquième. La raison de cette flexibilité est la réutilisation des concepts déjà modélisés dans l'ontologie. Par exemple, un quatrième scénario hypothétique pourrait être constitué d'une machine qui se fait infecter par un courriel frauduleux. La partie de ce scénario qui concerne l'ouverture du courriel comporte de nouveaux concepts qui ne sont pas présents dans l'ontologie créée en modélisant les scénarios n^{os} 1, 2 et 3. Par contre, la partie de l'infection de la machine par un logiciel malveillant a déjà été modélisée dans le premier scénario. Les concepts seront réutilisés en grande partie et le développement s'en verra facilité.

En démontrant que le nombre d'axiomes introduits, représentant l'effort de détection, est moins grand d'un scénario à l'autre, nous appuyons le fait que DIOSE est flexible d'un scénario à l'autre. En effet, il y a une forte réutilisabilité de concepts entre les scénarios. Ceci

s'explique par le fait que certains éléments, tels qu'une machine ou une requête HTTP, sont utilisés dans plusieurs scénarios, et que la modélisation faite dans un précédent scénario peut être réutilisée.

Il est important de noter que le nombre d'axiomes dans l'ontologie à ce jour est plus élevé que la valeur à la fin de la modélisation du scénario n° 3. Il s'agit du fruit d'une étape d'enrichissement de l'ontologie que nous avons effectuée par souci du détail et pour aider les travaux futurs. La valeur actuelle ne mérite pas d'être dans le graphique puisque l'enrichissement n'améliore pas la détection en tant que telle. Il permet plutôt d'augmenter d'autres critères de l'ontologie tels que l'interopérabilité et la réutilisabilité.

5.2.2 Maintenabilité

Étant basé sur une ontologie, DIOSE bénéficie des avantages d'utilisation de cette technologie. En plus de structurer la connaissance en la désambiguïsant, l'ontologie est un modèle extensible. Les forces d'utiliser un modèle extensible sont démontrées lorsqu'on tente de modifier ledit modèle. Nous présenterons trois exemples de modification de notre modèle ontologique pour exposer que ceux-ci ne nécessitent pas une grande quantité d'effort.

Ajout d'un concept

Le besoin est d'ajouter un nouveau concept qui n'est pas modélisé dans l'ontologie. Une classe sera créée pour avoir une meilleure conceptualisation de cette nouvelle réalité. Pour l'exemple, supposons que la classe « HTTPAuthentication » doit être ajoutée à l'ontologie. Ce modèle étant facile à modifier, le développeur de l'ontologie n'aura qu'à créer une nouvelle classe et déterminer ses relations d'équivalence de sous-classe. Il peut choisir de définir que cette classe sera équivalente à `HTTPRequest and RemoteAuthentication`. Un autre choix serait de catégoriser la nouvelle classe comme sous-classe de NetworkRequest et de RemoteAuthentication. Le choix est au concepteur, mais dans les deux cas, les modifications sont simples. Les requêtes SPARQL ainsi que les règles en LD ou en SWRL n'auront pas à être modifiées si elles concernent les classes parentes de la nouvelle classe puisque cette dernière sera considérée. En raison de sa nature modulaire, une modélisation par ontologie permet de minimiser les changements à effectuer.

Modification d'une classe

Le besoin est de modifier un concept déjà présent dans l'ontologie. Pour l'exemple, il s'agit du concept de PotentiallyMaliciousProtocol. Nous l'avons modélisé par l'ensemble des proto-

coles HTTP, DNS, IRC (Internet Relay Chat) et SSL. Cette classe est assez faible puisque ces protocoles ne garantissent en rien qu'une requête réseau est malveillante. Un administrateur de système pourrait vouloir changer cette définition de classe puisqu'elle ne correspond pas à sa réalité. Pour l'exemple, l'entreprise pourrait être une entreprise de développement d'applications Web. L'inclusion de HTTP, et par le fait même SSL, dans cette définition noie leur système de fausse alerte, soit des faux positifs. Pour les modifications qu'il souhaite apporter, il pourra aisément supprimer des individus de la classe PotentiallyMaliciousProtocol et en ajouter de nouveaux. À la suite de ses modifications, cette classe aura comme individus les protocoles DNS et IRC. Le fonctionnement par classe d'une ontologie fait en sorte que les ressources, soient les requêtes et règles qui utilisent cette classe, n'auront pas besoin d'être modifiées.

Ajout d'un senseur

Le besoin est d'ajouter un senseur à DIOSE. Pour l'exemple, prenons la première étape du scénario n^{o} 1 qui effectue une corrélation entre deux étapes avec le même contexte. Cette étape détecte l'ouverture d'un fichier présent sur un média amovible en détectant une partition montée puis une lecture de fichier présent sur cette partition. Dans notre exemple, une entreprise externe propose un senseur qui effectue exactement cette détection. En utilisant ce produit, des alertes sont générées lorsqu'un fichier est lu à partir d'un média amovible. La logique de ce senseur est probablement meilleure que la nôtre puisqu'il s'agit du principal champ d'expertise de ce produit. Ainsi, nous choisissons d'ajouter ce senseur à la solution. À cause de la conception modulaire de notre système expert, il est très facile d'ajouter ce senseur. Pour faire le pont entre ce senseur et l'ontologie, un pilote sera développé. Celui-ci transformera les alertes du senseur en instance de type ExternalDriveFileRead, qui est la classe que nous utilisons dans l'ontologie pour modéliser ce concept. Selon le choix de l'utilisateur, il peut laisser nos règles existantes puisqu'elles ne cohabitent pas mal avec ce nouveau senseur. Par contre, il faut tenir compte que certains événements seront possiblement des doublons. Cela peut être un choix puisqu'il vaut mieux avoir une alerte deux fois plutôt qu'une.

5.3 Les patrons d'attaques

Nous avons démontré que nous pouvons détecter trois scénarios précis et que l'effort d'en détecter d'éventuels va en diminuant. Par contre, si la réalité est composée d'un nombre indéfinissable de scénarios, DIOSE ne sera pas performant. En effet, il faudra modéliser ce très grand nombre de scénarios pour pouvoir les détecter, ce qui est impossible.

Nous résolvons partiellement cette problématique avec DIOSE puisqu'il bénéficie du principe d'abstraction. En effet, en représentant des étapes d'attaques par des groupes abstraits, nous pouvons détecter des attaques similaires avec une même règle qui utilise ces groupes abstraits. Prenons l'exemple de la quatrième étape du deuxième scénario. Dans cette étape, l'attaquant effectue une injection SQL sur une page Web située sur un site Web qui possède la vulnérabilité WPVDB-7846. Observons la requête 5.1, qui est une version simplifiée de la règle de détection de cette étape.

```
SELECT * WHERE
{
        ?event rdf:type :SQLInjectionPayload .
        ?event :hasDestination ?webpage .
        ?website :hasWebpage ?webpage .
        ?website :hasVulnerability ?vuln .
        ?vuln rdf:type :SQLInjectionVulnerability
}
```
(5.1)

Les deux dernières lignes de cette requête utilisent la classe abstraite qui représente les vulnérabilités qui permettent l'injection SQL, et non spécifiquement la vulnérabilité du scénario. Ainsi, nous avons utilisé l'abstraction pour permettre à notre règle d'être moins spécifique. Reprenons l'exemple en remplaçant la vulnérabilité WPVDB-7846 par la vulnérabilité CVE-2016-9864, qui permet également une injection SQL. Sans expliciter ce nouveau scénario, la règle de l'étape quatre du scénario n^{o} 2 permet de détecter cette attaque puisque nous avons utilisé une classe d'abstraction. Ainsi, l'abstraction permet de réduire le nombre de scénarios à développer puisque nous les définissons d'une façon plus générale. Ceci contribue à résoudre le problème du grand nombre de scénarios à modéliser. Cependant, il s'agit toujours d'un problème si ce nombre tend vers l'infini puisque ce n'est pas modélisable.

Nous avons poursuivi notre travail pour traiter la problématique du nombre de scénarios différents. À force de parcourir la littérature à la recherche de scénarios d'attaque informatique, nous nous sommes aperçus que la majorité des étapes des attaques informatiques sont similaires. À tout de moins, le bassin d'étapes possibles n'est pas infini et il serait possible d'expliciter une grande partie de ceux-ci. Du travail a été fait pour discerner des patrons (*patterns*) dans ces scénarios d'attaque informatique. De plus, tout porte à croire que pour la majorité des attaques informatiques, le nombre de ces patrons est fini, et qu'il est dans les li-

mites de ce qu'il est possible de modéliser. Ces patrons peuvent être modélisés et représentent une utilisation de l'abstraction, qui est une des forces de l'ontologie.

Ainsi, il existerait un nombre fini de scénarios décrits par une série de patrons, et ces scénarios peuvent être modélisés. L'utilisation DIOSE serait possible et elle n'est pas gênée par le grand nombre de scénarios. Qui plus est, notre concept d'étapes représenté par des états de machines à états correspond au concept de patrons. En effet, les deux représentent des abstractions d'étapes de scénario.

Par exemple, sans toucher au modèle de notre solution, celle-ci pourrait détecter le quatrième scénario hypothétique suivant : un attaquant exploite une vulnérabilité Heartbleed sur un serveur VPN puis se connecte sur une machine à l'aide des identifiants compromis d'un utilisateur. Par la suite, il procède à une exploitation Mimikatz pour effectuer un vol d'informations confidentielles de l'entreprise. Ce scénario est composé des quatre premières étapes du scénario n^{o} 3, suivi des deux dernières étapes du scénario n^{o} 1. En concevant les étapes d'un scénario en états de machine à état, nous avons créé des patrons réutilisables d'attaques. Ce que nous appelons couramment les blocs de construction, puisqu'ils peuvent être interchangeables dans la modélisation d'un scénario.

Nous sommes conscients que ce que nous avançons par rapport au nombre limité de patrons d'attaques s'applique uniquement aux attaques communes et connues. Une attaque ciblée de type Advanced Persistent Threat (APT) sera en mesure de suivre un scénario d'attaque qui déroge aux patrons d'attaques populaires. L'attaquant utilisera des vulnérabilités « 0-day », prendra des chemins non conventionnels et mettra un accent particulier sur sa furtivité (Virvilis et Gritzalis, 2013). Il s'agit de la limite de la flexibilité que nous proposons avec l'utilisation de DIOSE.

5.4 Processus d'affaires

Notre solution proposée possède différents contextes d'utilisation que nous présenterons pour démontrer son utilisation. Nous présenterons ces contextes d'utilisation tels qu'ils seraient rencontrés dans un processus d'affaires autour de DIOSE. Nous commencerons par une explication de l'utilisation par le fabricant de DIOSE. Ensuite, nous analyserons l'implantation effectuée par un client qui souhaite déployer la solution. Par la suite, nous exposerons comment nous voyons l'utilisation en temps réel de notre SIEM. Finalement, nous présenterons des cas d'utilisation de la maintenance DIOSE par ce client lorsque des modifications sont à effectuer.

5.4.1 Première utilisation : le fabricant

Le fabricant de DIOSE développera une solution générique qui fonctionne pour une majorité de scénarios et qui propose une modélisation d'un environnement TI générique. Pour la détection des scénarios, il souhaitera détecter les cas d'attaques connues les plus populaires. Comme vu dans l'architecture de DIOSE, le modèle qu'il développe sera autour des concepts d'événement, de vulnérabilités et d'éléments contextuels. En fonction du processus de développement ontologique choisi, il concevra une modélisation ontologique qui permettra de détecter ces scénarios d'attaque. Concrètement, il créera la hiérarchie de classes, donnera une définition à ces classes et aux propriétés, puis écrira des règles de transformations. Ce contenu dans le modèle ontologique sera effectué pour que la détection fonctionne dans l'environnement d'une entreprise typique. À cause de la nature de ce développement, il est nécessaire que ceux qui effectuent ce travail soient à la fois des experts en développement ontologique et en sécurité informatique.

Pour la modélisation d'un environnement générique, les éléments de contexte qu'il créera feront partie des bases d'un environnement en TI. Par exemple, le concept d'une machine est présent dans tous les environnements, ainsi que les règles qui relient un événement réseau à une machine par correspondance d'adresse IP. Ces éléments peuvent être incorporés dans une solution générique puisqu'il se retrouve dans tous les environnements. Effectuer une telle modélisation permet de proposer une solution complète à un client, malgré le fait que les besoins et l'environnement ne sont pas connus en détail.

Essentiellement, il s'agit du travail que nous avons effectué en développant un modèle qui permet de détecter trois scénarios d'attaque. Une grande partie du travail que nous avons négligé est le développement de pilotes pour interfacer les senseurs et configurateurs avec l'ontologie. La difficulté de la création de pilotes réside dans le fait que le nombre de pilotes différents à créer est énorme. Comme présenté dans la problématique, les données nécessaires pour détecter des scénarios sont très hétérogènes. Pour chaque type de données, il existe différents senseurs et configurateurs. Chacun de ceux-ci nécessite un pilote différent. Il ne s'agit pas d'un travail complexe, mais il est laborieux. Ainsi, le fabricant de DIOSE doit développer les différents pilotes des produits qu'ils souhaitent supporter. Ces tâches peuvent être effectuées par une communauté selon la licence de droits de DIOSE. Lors de notre travail, nous avons également modélisé des concepts hors des scénarios. Ceci faisait partie de la modélisation d'un environnement générique et avait pour but de faciliter les futures modifications.

Pour offrir une solution complète à ses clients, le concepteur de DIOSE doit s'assurer que le système expert possède la modélisation d'un ensemble de vulnérabilités. Le fait de modéliser

l'ensemble des vulnérabilités existantes représente une tâche laborieuse. Ainsi, les concepteurs pourraient modéliser seulement les vulnérabilités qui retiennent l'attention des experts de la communauté de sécurité informatique. Ces vulnérabilités concernent généralement des logiciels dont les mises à jour sont souvent négligées, par exemple Windows, Internet Explorer, Samba ou Apache. Ce choix arbitraire est discutable, mais il est un pas dans la bonne direction pour détecter la majorité des attaques puisque celles-ci utilisent généralement un petit ensemble de vulnérabilité.

5.4.2 Deuxième utilisation : le client

DIOSE sera déployé chez le client. Celui-ci recevra un système expert dont le modèle représente une solution générique. À l'aide d'expert en modélisation ontologique, le client va vouloir adapter DIOSE à ses besoins. La hiérarchie de classes évoluera pour représenter ses besoins. Par exemple, le client créera une classe InternalWebsite s'il propose des sites Web à l'interne et veut les distinguer de ses sites Web publics. Également, il devra créer les classes d'événements qui représentent les senseurs présents dans son environnement.

À la suite de la modification de la structure de la modélisation, il procédera à la population de DIOSE. Cette opération concerne différents types de données, et peut se faire soit manuellement ou programmatiquement. Les informations qui seront représentées par des instances sont toutes celles liées à un configurateur. On peut penser à une machine, un service réseau, un site Web, un module utilisé par un service, un utilisateur, etc. Si le configurateur ainsi que son pilote associé ont été créés, leur exécution permettrait de peupler aisément le modèle de l'ontologie.

Les administrateurs responsables de l'implantation du DIOSE auraient avantage à tirer parti des informations disponibles dans un précédent système de gestion de l'information, par exemple un SIEM. En effet, si l'entreprise possède un tel système, il pourrait tirer parti des attaques dont l'entreprise a été la cible et des vulnérabilités couramment exploitées dans leur environnement.

Ensuite, si les événements détectés par ses senseurs ne peuvent être instanciés dans l'ontologie, il doit développer un pilote pour faire ce pont. Ceci permettra de ne rien modifier envers ses mécanismes de défense et permettra l'utilisation des événements collectés par ceux-ci.

5.4.3 Troisième utilisation : en production

À la suite du déploiement de DIOSE, celui-ci peut être exécuté pour effectuer une détection. En reprenant le processus de détection d'intrusion exposé à la figure 2.1, la collecte est

effectuée par les senseurs et les configurateurs. Les senseurs sont exécutés en continu pour être toujours à l'écoute d'événements malveillants. Pour des soucis d'optimisation de ressources, les configurateurs peuvent être exécutés périodiquement puisque l'environnement ne tend pas à changer constamment. Par exemple, le configurateur C001 qui crée les machines, les services, les modules et les pages Web n'a pas besoin d'être exécuté en continu puisque ces concepts sont très statiques. Ce configurateur pourrait être exécuté quotidiennement.

L'étape d'agrégation est effectuée par les pilotes, qui s'interface avec les senseurs et configurateurs pour envoyer leurs données collectées dans l'ontologie. Une fois dans l'ontologie, l'étape de corrélation s'effectue par le raisonneur et les règles de raisonnement. Celles-ci sont en divers formats (SWRL, LD, algorithmique, machine à état) et permettent ultimement d'approcher les données vers un langage utilisé par l'expert. Le but ultime est de détecter des intrusions et d'aider au travail d'un expert effectuant cette tâche.

5.4.4 Quatrième utilisation : les besoins de maintenance

Comme tout système de détection, DIOSE doit procéder à des maintenances régulières pour s'assurer que sa détection soit à jour. En effet, le domaine de la sécurité informatique est très dynamique puisqu'il y a constamment de nouvelles attaques exploitant de nouvelles vulnérabilités. De plus, l'environnement de l'entreprise est appelé à changer éventuellement. Ces éléments font que la modélisation doit être mise à jour pour que celle-ci représente bien la réalité et que le taux de détection d'intrusion soit maximisé.

Ajout d'un concept d'environnement

Nous présenterons l'implantation du besoin d'ajouter le concept d'emplacement réseau d'une machine dans l'ontologie. La finalité de cette modification sera de pouvoir utiliser ce concept dans les requêtes pour décrire des scénarios d'attaque. Nous décrirons les étapes effectuées par un utilisateur du système qui procède à la modification du modèle ontologique dans le but d'intégrer le concept d'emplacement réseau. Notons qu'il y a plusieurs façons différentes de faire la modélisation et que nous n'en présenterons qu'une seule.

Premièrement, le responsable de la maintenance crée la classe NetworkLocation qui est au premier niveau de la hiérarchie de classes. Il peut également enrichir cette hiérarchie en créant des sous-classes à la classe créée. Par exemple, il peut représenter la zone démilitarisée (DMZ) en créant une classe du même nom. Par la suite, il crée les propriétés relatives à ce concept : la propriété d'objet « hasNetworkLocation » qui relie une machine à un emplacement réseau et la propriété de données « hasSubnet » qui sera associée à une machine.

Deuxièmement, il crée un configurateur qui peuplera la classe NetworkLocation et ses sous-classes. Par exemple, ce configurateur créera les instances suivantes sous-classe de NetworkLocation : « localNetwork », « serverNetwork » et « wan ». Il créera également les instances suivantes, qui sont sous-classe de DMZ (elle-même sous-classe de NetworkLocation) : « dmz1 » et « dmz2 ». Ces instances doivent posséder la propriété « hasSubnet » pour les décrire. Par exemple, nous aurions les deux triplets suivants : (serverNetwork, hasSubnet, « 192.168.100.0/24 ») et (dmz1, hasSubnet, « 10.1.2.0/8 »).

Troisièmement, le responsable de la maintenance modifie le configurateur existant qui est responsable de la création des machines. Lors de cette création, les instances doivent avoir la propriété « hasSubnet ». Un exemple d'instance créée ainsi serait le triplet suivant : (pc, hasSubnet, « 192.168.1.0/24 »).

Quatrièment, une règle d'inférence doit être écrite pour relier les machines aux instances de NetworkLocation. Nous avons choisi de faire la règle en SWRL telle que représentée à la règle 5.2.

$$\begin{gathered} Machine(?m) \wedge hasSubnet(?m, ?subnet) \wedge NetworkLocation(?location) \wedge \\ hasSubnet(?location, ?subnet) \rightarrow hasNetworkLocation(?m, ?location) \end{gathered} \tag{5.2}$$

Finalement, il peut terminer son ajout du concept en enrichissant des classes existantes. D'une part, il peut enrichir la classe Machine en ajoutant cet axiome à sa propriété de sous-classe : `and hasNetworkLocation some Machine`. D'autre part, il peut poursuivre l'enrichissement en définissant une propriété d'équivalence pour la classe ExternalMachine. Celle-ci serait décrite comme suit : `Machine and hasNetworkLocation value wan`.

Ajout d'une nouvelle vulnérabilité

Il est fortement recommandé de régulièrement mettre à jour la base de connaissances de vulnérabilités. L'ajout d'une vulnérabilité dans l'ontologie du système expert est une tâche qui requiert peu de modélisation. C'est une tâche qui est faite régulièrement puisque la plupart des entreprises sont abonnées à des flux de mises à jour pour être au courant des dernières vulnérabilités connues.

Nous exposerons le cas d'un administrateur de système qui souhaite prévenir son environnement contre une nouvelle vulnérabilité. Pour l'exemple, celle-ci sera la vulnérabilité CVE-2016-4117 qui concerne le module Adobe Flash Player. Si la version de cette dernière est inférieure à la version 21.0.0.226, il s'agit d'une libraire vulnérable à une exécution distante

de code. Il s'agit d'une vulnérabilité critique et son implémentation dans le DIOSE est une priorité une fois qu'elle est découverte.

Premièrement, l'administrateur créera une sous-classe de Module qu'il nommera AdobeFlashPlayer. Celle-ci possèdera les instances représentant les différentes versions du module. Pour notre exemple, il y aura les instances « adobeflash-2100226 » et « adobeflash-2101000 ». Ces instances seront créées par le configurateur lié aux machines et modules. Celui-ci devra être modifié pour qu'il recense les différentes versions d'Adobe Flash Player installées. Les instances créées auront également une propriété « hasVersion » qui les associe à la valeur représentant leur version.

Deuxièmement, la vulnérabilité en tant que telle doit être modélisée. D'une part, la classe CVE-2016-4117VulnerableModule sera créée. Celle-ci sera sous-classe de la classe AdobeFlashPlayer. D'autre part, nous créerons une règle SWRL qui implémente la logique représentant la vulnérabilité. Cette règle est présentée à la règle 5.3. Notons que l'antécédent de cette règle utilise un mot-clé que nous n'avons pas introduit. En utilisant la propriété « swrlb:lessThanOrEqual », nous pouvons faire une comparaison numérique pour spécifier la version de la librairie.

$$\begin{aligned} &AdobeFlashPlayer(?m) \wedge hasVersion(?m, ?version) \wedge \\ &swrlb{:}lessThanOrEqual(?version, 2100226) \rightarrow CVE20164117VulnerableModule(?m) \end{aligned} \tag{5.3}$$

En ayant créé les classes, les instances et la règle, les modules dont la version les rend vulnérables font maintenant partie de la classe CVE-2016-4117VulnerableModule. Ainsi, avec une simple requête SPARQL, l'utilisateur peut obtenir l'ensemble des machines qui possèdent le module vulnérable et peut prendre les mesures nécessaires pour mitiger le risque d'exploitation.

CHAPITRE 6 DISCUSSION ET CONCLUSION

À la suite de la présentation de nos résultats, nous discuterons de ceux-ci en regard de notre objectif de recherche. Nous terminerons le mémoire avec une conclusion pour décrire les limitations de DIOSE en plus des travaux futurs.

6.1 Discussion

Nous avons tiré quatre hypothèses de notre objectif de recherche. Dans la discussion, nous exposerons le travail que nous avons fait dans le but de confirmer ou infirmer ces hypothèses.

6.1.1 Hypothèse n^o 1 - L'expressivité des règles

Notre première hypothèse de recherche était relative à l'expressivité des règles de corrélation de DIOSE. Cette hypothèse stipule que l'utilisation de notre système expert avec un tel module de corrélation peut allier les éléments essentiels à la détection d'intrusion. Ces éléments sont les événements, les vulnérabilités et les éléments contextuels.

Au chapitre 4, nous avons présenté trois scénarios d'attaque. Ces scénarios étaient dotés de réalisme puisqu'une grande partie de leurs éléments découlait de rapports externes reconnus. Nous avons présenté de quelle façon nous prévoyons effectuer la détection des étapes constituant ces attaques à l'aide de requêtes SPARQL. Elles utilisent toutes les concepts d'événements et de contexte. Le concept de vulnérabilité est seulement utilisé dans quelques étapes. En utilisant les ontologies comme modèle de données, nous avons pu aisément encoder les informations reliées aux trois concepts généraux utilisés pour la détection d'intrusion. Comme expliqué au chapitre 2, nous nous démarquons de certaines solutions antérieures dans lesquelles seuls les événements étaient utilisés dans les règles de détection.

Chacune de ces requêtes SPARQL est en fait une règle de corrélation. En utilisant le langage de requêtes SPARQL, nous pouvons aisément créer des requêtes qui combinent les trois concepts fondamentaux. Nous soulignons la grande accessibilité du langage SPARQL puisque celui-ci est standardisé et populaire. L'utilisation de ce langage place DIOSE à l'opposé de bons nombres d'autres solutions qui inventaient leur propre langage de corrélation. Le fait d'utiliser un langage connu permet de tirer parti de sa robustesse et d'une grande quantité de documentation.

En plus de l'expressivité d'écriture des règles SPARQL, nous affirmons qu'une solution qui propose une détection basée uniquement sur les événements est moins riche qu'une détection

basée sur les événements, le contexte et les vulnérabilités. Ainsi, l'expressivité offerte par SPARQL, en plus d'une modélisation ontologique comme celle que nous avons proposée, permet d'avoir une meilleure compréhension de la réalité de la détection d'intrusion.

Pour illustrer nos propos, prenons l'exemple de l'étape n° 2 du deuxième scénario, décrit à la section 4.1.3. Cette étape représente une attaque complète XSS : une infection initiale d'une page Web suivie par la visite de cette page Web par un administrateur du site Web qui contient la page Web. Dans le cas d'une détection basée uniquement sur les événements, elle serait constituée de deux requêtes : la requête R029 qui obtient les requêtes HTTP qui exploitent une faille XSS, puis la requête qui obtient simplement les requêtes HTTP. La conclusion d'une telle détection est reliée au fait que l'événement a eu lieu ou non. Pour arriver à une conclusion que l'événement a eu lieu et qu'un administrateur s'est fait infecter par cet événement, une corrélation avec des éléments contextuels est nécessaire. Toujours dans le cas d'une solution qui offre une détection basée uniquement sur les événements, la corrélation devra se faire manuellement avec un expert. Celui-ci corrélera que sur la même page Web destinée par l'exploitation XSS, une machine sur laquelle l'administrateur du site Web est connecté a généré une requête destinée à la page Web.

Dans le cas d'une détection basée sur les événements, le contexte et les vulnérabilités, il s'agit de la requête R037. Grâce à l'expressivité du langage de corrélation, il nous est possible d'exprimer la corrélation décrite précédemment. L'utilisation d'éléments contextuels permet à DIOSE d'effectuer les corrélations présentées à l'énumération suivante :

- La requête HTTP qui contient l'exploitation XSS destine une page Web P
- Une requête HTTP est effectuée après l'exploitation XSS. Elle destine la page Web P et provient d'une machine M
- Un utilisateur U est connecté sur la machine M
- L'utilisateur U fait partie d'un groupe possédant une politique de sécurité qui possède des droits administrateurs sur un site Web S
- Le site Web S contient la page Web P

L'expressivité permet de transférer le raisonnement autrefois fait par l'expert à DIOSE. Notons que l'exemple que nous avons utilisé pour illustrer nos propos n'utilise pas le concept de vulnérabilités. Bien que l'attaque que nous avons décrite exploite une vulnérabilité XSS, notre expertise nous a guidés pour prendre le choix de ne pas la modéliser. Des vulnérabilités de ce type sont rarement répertoriées et connues à cause du comportement très dynamique des applications Web. Selon l'expertise impliquée pendant la modélisation, il aurait été possible d'inclure une telle vulnérabilité dans la règle de corrélation.

6.1.2 Hypothèse n^{o} 2 - La diminution de l'expertise requise

Notre deuxième hypothèse de recherche stipulait que l'utilisation d'un système expert tel que nous l'avons proposé permet de réduire l'expertise d'utilisation dudit système expert. Nous défendons que l'hypothèse est confirmée puisque la modélisation de connaissance avec une ontologie est une bonne solution pour encoder le savoir expert. Avec DIOSE, l'encodage d'informations s'effectue avec une montée d'abstraction de l'information. Cela permet aux données de passer d'une représentation concrète à une représentation abstraite proche de celle utilisée par l'expert. Nous présenterons en quoi la montée d'abstraction permet de diminuer l'expertise requise pour confirmer l'hypothèse de recherche.

Lors du développement de DIOSE, nous avons encodé un savoir expert pour détecter des scénarios d'attaque. L'exemple que nous analyserons est la quatrième étape du premier scénario, tel que décrit à la section 4.1.2. Cette étape est celle d'une exploitation Mimikatz, et elle est détectée par la requête R024. Elle est composée d'un contexte vulnérable, d'un événement d'authentification par un utilisateur-administrateur de la machine ciblée ainsi que d'un événement de lecture de mémoire d'un processus spécifique. Lors de la modélisation utilisée pour détecter cette étape, nous avons encodé le savoir expert en utilisant la montée en abstraction des concepts. Nous avons déterminé quelles transformations de l'information étaient nécessaires pour faire passer les données brutes vers un format proche du langage expert.

La figure 6.1 présente le diagramme de montée d'abstraction utilisée pour détecter cette étape. En d'autres mots, les règles impliquées dans cet exemple permettent de simplifier la détection de cette étape. Nous présenterons cette figure pour appuyer le fait qu'il y a plusieurs règles sous-jacentes à la composition de la requête R024 qui permet de détecter l'étape. Cependant, une fois que les éléments de ce diagramme sont créés, la requête R024 peut être utilisée par un utilisateur du système. Ce dernier n'a pas besoin de comprendre le fonctionnement interne de l'exploitation Mimikatz. Il peut procéder à cette détection en utilisant la requête R024 précédemment créée par un expert qui a défini la requête de détection. Dans ce cas, l'expert a défini qu'une exploitation Mimikatz est constituée de deux événements qui se déroulent sur une machine dont le système d'exploitation est vulnérable à cette exploitation. Le premier événement est une authentification distante par un utilisateur qui possède des droits administrateurs sur cette machine. Le deuxième événement est une lecture de mémoire du processus « lsass.exe » qui se produit sur cette même machine. Cette mécanique interne est abstraite par le concept de requêtes prédéfinies. Ceci permet à un analyste de sécurité de bénéficier du savoir encodé dans le modèle de données. Il pourra procéder à la détection d'exploitation Mimikatz sans avoir une compréhension détaillée des éléments qui la composent.

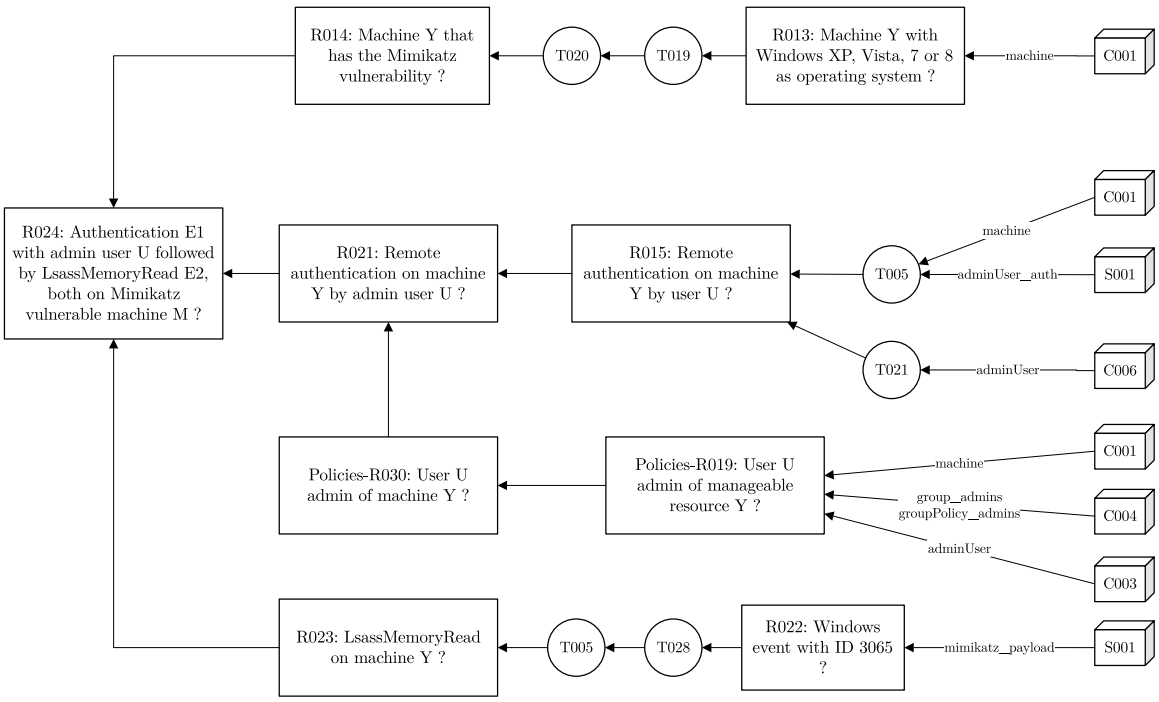

Figure 6.1 Diagramme de montée en abstraction de l'exploitation Mimikatz

6.1.3 Hypothèse n^{o} 3 - L'extensibilité du modèle

Notre troisième hypothèse de recherche était relative à l'extensibilité de notre solution. L'extensibilité se dit d'une solution flexible qui peut bien croître, c'est-à-dire que les modifications ne sont pas trop laborieuses. Dans la section 5.2, nous avons tenté de confirmer cette hypothèse en deux parties. Premièrement, la difficulté de modélisation de scénarios décroit en fonction du nombre de scénarios précédemment modélisés. Ceci s'explique principalement par la réutilisation de concepts inhérents au domaine de la sécurité informatique, ainsi que par l'adaptation des développeurs envers la modélisation de scénarios. Au début de la modélisation de chacun des trois scénarios, nous avons noté la volumétrie de notre modélisation utilisant une ontologie. Ainsi, nous avons pu analyser ces trois valeurs, qui représentent le nombre total d'axiomes dans l'ontologie. Le graphique 5.1 démontre que le nombre d'axiomes introduits à chaque scénario décroit. Puisque nous établissons que la quantité étudiée est en relation directe avec l'effort de développement, nous pouvons conclure que l'effort diminue plus le nombre de scénarios modélisés augmente.

Deuxièmement, nous avons proposé trois cas de figure de modifications à une modélisation dans la sous-section 5.2.2. À travers ces trois cas de maintenance, nous voulions démontrer d'une manière très qualitative que les changements à la modélisation liés à DIOSE se font aisément. Les trois cas de figure ont été choisis arbitrairement et ils représentent des opérations

populaires pour des systèmes de ce type.

Par la présentation de la tendance de diminution d'efforts ainsi que la facilité de modifier le modèle de données, nous confirmons l'hypothèse que DIOSE est extensible.

Critiques

Nous adressons quelques critiques envers l'extensibilité de notre solution. Premièrement, notre analyse d'effort de développement avec le nombre d'axiomes introduits à chaque scénario a été faite sur un petit échantillon. Il est assez faible d'établir une tendance avec seulement trois points. Si nous avions développé plus de scénarios, notre analyse aurait également pu présenter une valeur plateau du nombre d'axiomes introduits. De plus, il aurait été intéressant de noter quels concepts étaient réutilisés d'un scénario à l'autre.

Deuxièmement, nous adressons une critique à la présentation des cas de figure de maintenance. Pour pouvoir modifier l'ontologie, il faut avoir une compréhension des technologies qui y réfèrent. Les utilisateurs qui souhaitent effectuer des modifications ont une courbe d'apprentissage qui n'est pas négligeable. Ainsi, pour des néophytes du domaine de la modélisation ontologique, il peut être ardu d'effectuer une modification.

6.1.4 Hypothèse n^o 4 - La viabilité en environnement de production

Notre quatrième et dernière hypothèse de recherche était relative à la viabilité de DIOSE dans un environnement réel de production. Pour valider cette hypothèse, nous avons analysé DIOSE sous trois aspects : les contraintes nécessaires à son utilisation, l'utilité de notre solution pour la détection d'intrusion, ainsi que le processus d'affaires pour la déployer dans un contexte d'entreprise.

Premièrement, nous avons présenté deux contraintes liées à l'utilisation de DIOSE. L'une prétend que les senseurs doivent fonctionner en mode détection plutôt qu'en mode prévention, et l'autre prétend qu'il n'y a pas de faux négatif pour les scénarios à détecter. L'ajout de contraintes restreint l'utilisation de DIOSE, mais nous défendons qu'il s'agisse de contraintes raisonnables pour que notre solution demeure utilisable. En effet, l'utilisation de senseurs en mode prévention est une solution qui semble intéressante. Cependant, à cause du nombre élevé de faux positifs, ce mode est difficilement utilisable puisqu'il bloque des événements légitimes. Le mode d'opération des senseurs qui est le meilleur compromis est la détection. Ainsi, nous pouvons supposer que tous les senseurs sont en mode détection sans trop restreindre la portée de l'utilisation de DIOSE.

Notre deuxième contrainte est relative à l'absence de faux négatifs pour les scénarios à dé-

tecter. Par exemple, nous définissons un scénario spécifique comme étant l'enchainement des étapes hypothétiques A, B et C. L'étape B est constituée d'une requête HTTP qui comprend une injection XSS. Si l'attaquant met un souci particulier à la furtivité de son attaque pour être sous le radar, nous ne pourrons détecter cette étape. Dans ce cas précis, il pourrait utiliser une technique d'évasion en construisant une exploitation XSS spécifique pour passer sous le nez du module de détection par règle du pare-feu applicatif. Ainsi, les étapes A et C sont détectées, mais pas l'étape B. Puisque nous avons défini que ce scénario est catégoriquement constitué des trois étapes, DIOSE conclura que le scénario n'a pas été détecté. Nous n'avons implémenté aucun mécanisme de flexibilité si une information est manquante dans la détection d'étapes ou de scénarios. Cette contrainte par rapport aux événements non détectés ne rend pas DIOSE inutilisable. En effet, une grande majorité des attaques ne mise pas sur la furtivité pour arriver à leurs fins. Les APT sont un type d'attaques informatiques qui prennent ces précautions, mais ils ne sont qu'une minorité des attaques informatiques. Donc, DIOSE reste applicable même lorsque nous considérons que nous ne prenons pas en compte l'existence des faux négatifs.

Deuxièmement, nous avons présenté l'utilité de notre SIEM face aux menaces en sécurité informatique. Nous prétendons qu'un nombre raisonnable de patrons d'attaques existent. Ces patrons peuvent être constitués d'une étape ou d'une succession d'étapes. Par exemple, nous avons présenté la requête 5.1 à la section 5.3 qui représente une injection SQL exploitant une vulnérabilité du même type. Il s'agit d'un patron d'attaque existant qui peut être utilisé pour détecter divers types d'injection SQL qui exploitent diverses vulnérabilités du même type. Ainsi, nous défendons que pour un environnement typique du domaine des TI, le nombre de patrons d'attaques est fini et qu'il est possible de modéliser ces patrons d'attaques dans DIOSE. À la suite d'un tel travail, il serait possible de détecter une grande partie des attaques informatiques. Évidemment, nous excluons les attaques ciblées complexes qui utilisent des chemins incongrus pour accomplir leurs desseins. En règle générale, nous ne pouvons détecter ce que nous ne connaissons pas.

Troisièmement, nous avons présenté l'utilisation de notre solution à travers son processus d'affaires. Nous avons décrit celui-ci à travers les acteurs qui y sont présents à différentes étapes. En présentant ce processus d'affaires, nous tentons de démontrer l'implantation possible de DIOSE. Il était nécessaire de faire cette preuve qualitative puisque nous n'avons pas effectué concrètement ce processus. La dernière étape du processus d'affaires présente des exemples de modifications que souhaiterait effectuer une entreprise. Cette section se veut également un guide d'utilisation pour les futures modifications. En proposant de tels cas de figure, nous appuyons le fait que DIOSE possède sa place sur le terrain pour effectuer la détection d'intrusion.

6.2 Conclusion

L'environnement toujours changeant des systèmes informatiques a contribué à augmenter à la fois la surface d'attaque et la motivation des acteurs malintentionnés. Dans ce contexte, les mécanismes de défense ont des difficultés face aux diverses menaces. Les SIEM sont responsables de cette défense en agrégeant des données collectées. Avec celles-ci, ils effectuent des corrélations sur l'information pour tirer des conclusions de détection d'intrusion. Les SIEM peinent à implémenter ce processus de détection d'intrusion pour diverses raisons : langage de corrélation peu expressif, langage de corrélation complexe et loin du langage expert et système de stockage rigide. Envers ces problématiques, nous proposons DIOSE, un système expert basé sur une représentation de connaissance par l'ontologie. Cette solution permettrait d'améliorer l'effort de détection d'intrusion envers le langage de corrélation et la flexibilité.

Notre parcours de la littérature était axé sur l'utilisation des ontologies et des technologies similaires dans le domaine de la détection d'intrusion. Nous avons recensé peu d'ouvrages qui prévoyaient d'utiliser la modélisation par ontologie dans le contexte d'un système expert. Plusieurs ouvrages proposaient des classifications avec ou sans ontologie à des buts de documentation, ce qui n'était pas notre but. Certains travaux s'adressaient à un problème précis, tels que la réduction de faux positifs ou la normalisation des informations. Les travaux qui présentaient un système expert pour effectuer la corrélation d'informations ont été retenus pour s'assurer que nous construisions sur des bases existantes.

Nous avons proposé la conception de DIOSE, un système expert basé sur une modélisation ontologique. Celui-ci permet d'agréger les différentes données collectées, soit les événements, les informations du contexte et les vulnérabilités. La corrélation sur ces informations permet de détecter des étapes d'attaques informatiques. Par la suite, grâce au concept de machines à état, DIOSE permet d'enchaîner ces étapes en fonction de leur sortie. Ceci est effectué dans le but de détecter un scénario à plusieurs étapes. À l'aide de trois scénarios d'attaque informatique, nous avons validé que notre système expert pouvait effectuer leur détection. Lors du développement de la modélisation pour détecter ces scénarios, nous avons noté le nombre de concepts introduits à chaque scénario dans le but d'étudier la tendance de l'effort de développement. À la suite de la description de l'application de DIOSE, nous avons expliqué comment nous voyons l'implémentation de celui-ci dans un environnement réel. Ce contexte apporte des contraintes différentes, ainsi des besoins, tels que la facilité de maintenance.

À la suite de nos expérimentations, nous confirmons par nos résultats que la modélisation qui utilise l'ontologie est applicable à la détection d'intrusion. Le langage de corrélation est amé-

lioré en le rendant plus expressif envers les types d'information. Il se rapproche également du langage expert et amoindrit l'expertise nécessaire pour utiliser la solution. Par notre analyse des efforts requis pour modéliser les scénarios, nous confirmons que l'effort de développement s'amoindrit à mesure que le nombre de scénarios modélisés augmente. Ceci vise à augmenter l'extensibilité du système expert. Finalement, nous avons expliqué en quoi DIOSE pourrait théoriquement fonctionner dans un environnement réel de production.

6.2.1 Limitations de DIOSE

Notre système expert basé sur l'ontologie possède des limitations. Nous en avons recensé quatre principales, et nous les décrirons.

Premièrement, nous avons une réserve envers l'utilisation de la modélisation ontologique que nous proposons. Durant notre recherche, nous avons compris qu'une ontologie peut être utilisée de deux façons. La première est associée au domaine de l'intelligence artificielle et utilise le principe du monde ouvert. Ce principe est une hypothèse qui dit que ce qui n'est pas connu est inconnu. L'hypothèse du monde ouvert va en contradiction avec l'hypothèse du monde fermé où il est établi que tout ce qui est inconnu est faux. Les bases de données relationnelles, entre autres, fonctionnent sous cette logique. On pose une question et si la réponse n'est pas positive, on la considère comme négative. Dans le monde ouvert, si la réponse n'est pas positive, on la considère comme inconnue, ce qui n'est pas nécessairement une réponse négative. La première façon d'utiliser une ontologie utilise l'hypothèse du monde ouvert. Pour cette façon, les concepts doivent tous être formellement définis, ce qui est difficile à réaliser puisque la réalité est très complexe à définir.

La deuxième façon d'utiliser l'ontologie est en tant que graphe orienté qui offre un bon système de règles et requêtes. Dans ce mode, nous pouvons fonctionner selon l'hypothèse du monde fermé, à l'instar des autres systèmes de stockage de données. C'est ce mode que nous avons utilisé pour DIOSE. Ceci est démontré lorsque nous établissons le fonctionnement de notre solution. En effet, nous définissons que la détection est négative si l'exécution d'une requête, qui représente une étape, ne retourne rien. Cette affirmation ne suit pas l'hypothèse du monde ouvert puisque selon celle-ci, nous affirmerions que la détection de l'étape est inconnue. Il serait difficile d'effectuer un système de détection fonctionnant selon l'hypothèse du monde ouvert.

Deuxièmement, notre solution pourrait être améliorée par rapport à la classification des vulnérabilités. Il s'agit d'une partie fondamentale pour procéder à la détection d'intrusion avec DIOSE, et nous n'en avons modélisé que trois. Pour avoir un système qui offre une bonne détection, un grand nombre d'entre elles devra être modélisé et pour ceci, un travail

de classification devra être effectué. Nous n'avons rien proposé à ce propos. Par contre, notons que du travail de classification de vulnérabilités existe déjà et une voie pour pallier cette faiblesse serait de relier le travail existant à notre ontologie.

Troisièmement, nous avons utilisé des raccourcis lors de notre modélisation. Nous avons fait ces choix pour épargner du temps et puisque ces raccourcis n'impactaient pas la modélisation du petit nombre de scénarios. Par contre, cela pourrait se complexifier lorsque plusieurs scénarios différents sont modélisés. Un de ces raccourcis est d'utiliser une adresse IP comme identifiant d'une machine. La réalité est plus complexe, en autre à cause de l'adressage dynamique du protocole DHCP (Dynamic Host Configuration Protocol) très couramment utilisé dans les environnements TI. Ainsi, une machine pourrait obtenir une nouvelle adresse IP en renouvelant son bail DHCP, et DIOSE considérerait qu'elle n'est plus la même.

Finalement, DIOSE possède une faiblesse en égard de l'aspect temporel des scénarios. Nous avons discuté de l'utilisation de la séquentialité des événements et des étapes. Cependant, le principe de fenêtre de temps mérite d'être plus abordé. Par exemple, celui-ci n'a pas été implémenté dans nos requêtes. À titre d'exemple, observons la règle T006 et la requête R024. Celles-ci utilisent le principe de séquentialité en comparant la valeur des temps associés à deux événements. Cependant, il n'y a rien concernant le principe de fenêtre de temps. Pour régler cette faiblesse, il peut s'agir d'une simple modification aux règles et requêtes sous la forme d'une différence entre deux temps. Nous n'avons pas exploré une telle avenue et présentons nos règles et requêtes sans cette fonctionnalité.

6.2.2 Améliorations futures

Les travaux futurs relatifs à notre solution sont de diverses natures. D'abord, nous analysons que des travaux pourraient être faits pour peaufiner DIOSE. Il y a une interrogation par rapport à la performance d'exécution de notre solution. En effet, un système qui s'exécute en continu se doit de pouvoir effectuer ses traitements dans un temps raisonnable. Si l'exécution faite par le raisonneur et l'engin de règles SPARQL n'est pas assez performante, le système ne sera pas utilisable. Nous prévoyons qu'une analyse détaillée de la performance d'exécution pourrait être une avenue de travaux subséquents. La finalité de ces travaux pourrait valider si DIOSE est applicable à un volume de données réaliste. Les travaux pourraient également offrir des recommandations concrètes pour améliorer la performance, tels que des choix de modélisations.

À la suite d'une confirmation positive de ces travaux de performance, DIOSE pourrait être à sa première version finale. Néanmois, nous prévoyons que l'utilisation de ce système expert pourrait être étendue à d'autres buts. L'un de ceux que nous n'avons pas directement explorés

est celui de la réduction des faux positifs. Il s'agit d'un des principaux chevaux de bataille de l'industrie de détection d'intrusion. Nous l'avons réglé indirectement en incluant des éléments contextuels pour valider les alertes sur les événements. Cependant, une étude plus approfondie pourrait être effectuée pour quantifier l'amélioration.

Également, nous prévoyons que des efforts supplémentaires pourraient être effectués pour ajouter une fonctionnalité importante à DIOSE : un mode d'opération *a posteriori*. En effet, nous n'avons pas exploré cette avenue de recherche puisque nous nous sommes concentrés sur un mode où le système s'exécute en continu. Une solution qui peut analyser des données historiques place cet outil dans le domaine de l'investigation numérique. Les travaux pour ajouter ce mode d'opération à DIOSE s'intéresseraient principalement aux notions de temps. La séquentialité et le concept de fenêtre de temps devront être étudiés pour comprendre comment les intégrer dans le nouveau contexte de DIOSE.

À plus long terme, nous espérons voir une implémentation concrète de nos travaux. Bien que nous souhaitons avoir mis la table pour que cette future implémentation se fasse avec le moins d'embûches possible, nous ne pouvons prévoir le futur. De plus, notre expérience nous a enseigné qu'il s'agit d'une aberration d'affirmer que tous travaux futurs ne rencontreront pas de problèmes inattendus.

RÉFÉRENCES

F. Abdoli et M. Kahani, “Ontology-based distributed intrusion detection system”, dans *Computer Conference, 2009. CSICC 2009. 14th International CSI.* IEEE, 2009, pp. 65–70.

AlienVault, Inc., “Alienvault”. En ligne: `https://www.alienvault.com/products`

Apache Software Foundation, “Jena”, 2011. En ligne: `https://jena.apache.org/`

S. Axelsson, “Intrusion detection systems: A survey and taxonomy”, Technical report, Rapp. tech., 2000.

A. Azodi, D. Jaeger, F. Cheng, et C. Meinel, “Pushing the limits in event normalisation to improve attack detection in ids/siem systems”, dans *Advanced Cloud and Big Data (CBD), 2013 International Conference on.* IEEE, 2013, pp. 69–76.

Blackstratus, Inc., “Blackstratus”. En ligne: `https://www.blackstratus.com`

F. Cheng, A. Azodi, D. Jaeger, et C. Meinel, “Security event correlation supported by multi-core architecture”, dans *IT Convergence and Security (ICITCS), 2013 International Conference on.* IEEE, 2013, pp. 1–5.

S. Cheung, U. Lindqvist, et M. W. Fong, “Modeling multistep cyber attacks for scenario recognition”, dans *DARPA information survivability conference and exposition, 2003. Proceedings*, vol. 1. IEEE, 2003, pp. 284–292.

P. Cimiano, *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications.* Springer, 2006.

Cisco, “2017 annual cybersecurity report”, Rapp. tech., 2017.

K. Clark, L. Feigenbaum, et E. Torres, “SPARQL protocol for RDF”, W3C, W3C Recommendation, Jan. 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-protocol-20080115/.

M. J. Covington et R. Carskadden, “Threat implications of the internet of things”, dans *Cyber Conflict (CyCon), 2013 5th International Conference on.* IEEE, 2013, pp. 1–12.

F. Cuppens et A. Miege, “Alert correlation in a cooperative intrusion detection framework”, dans *Security and privacy, 2002. proceedings. 2002 IEEE symposium on.* IEEE, 2002, pp. 202–215.

J. E. L. De Vergara, V. A. Villagrá, P. Holgado, E. De Frutos, et I. Sanz, "A semantic web approach to share alerts among security information management systems", dans *Web Application Security*. Springer, 2010, pp. 27–38.

H. Debar, D. Curry, et B. Feinstein, "The intrusion detection message exchange format (idmef)", Internet Requests for Comments, RFC Editor, RFC 4765, March 2007, `http://www.rfc-editor.org/rfc/rfc4765.txt`. En ligne: `http://www.rfc-editor.org/rfc/rfc4765.txt`

O. Depren, M. Topallar, E. Anarim, et M. K. Ciliz, "An intelligent intrusion detection system (ids) for anomaly and misuse detection in computer networks", *Expert systems with Applications*, vol. 29, no. 4, pp. 713–722, 2005.

C. Di Sarno, V. Formicola, M. Sicuranza, et G. Paragliola, "Addressing security issues of electronic health record systems through enhanced siem technology", dans *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*. IEEE, 2013, pp. 646–653.

M. Donner, "Toward a security ontology", *IEEE Security & Privacy*, no. 3, pp. 6–7, 2003.

C. G. Drury et M. A. Sinclair, "Human and machine performance in an inspection task", *Human Factors*, vol. 25, no. 4, pp. 391–399, 1983.

H. T. Elshoush et I. M. Osman, "Alert correlation in collaborative intelligent intrusion detection systems—a survey", *Applied Soft Computing*, vol. 11, no. 7, pp. 4349–4365, 2011.

G. Eschelbeck et M. Krieger, "Eliminating noise from intrusion detection systems", *Information Security Technical Report*, vol. 8, no. 4, pp. 26–33, 2003.

S. Fenz et A. Ekelhart, "Formalizing information security knowledge", dans *Proceedings of the 4th international Symposium on information, Computer, and Communications Security*. ACM, 2009, pp. 183–194.

M. E.-S. Gadelrab, "Évaluation des systèmes de détection d'intrusion", Thèse de doctorat, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2008.

Gartner, "Magic quadrant for security information and event management", *Gartner Research*, 2016.

A. Gómez-Pérez et M. C. Suárez-Figueroa, "Scenarios for building ontology networks within the neon methodology", dans *Proceedings of the fifth international conference on Knowledge*

capture. ACM, 2009, pp. 183–184.

T. R. Gruber *et al.*, “A translation approach to portable ontology specifications”, *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

N. Guarino, D. Oberle, et S. Staab, “What is an ontology ?” dans *Handbook on ontologies*, 2e éd., S. Staab et R. Studer, éds. Springer, 2009, pp. 1–17.

J. M. Haight et V. Kecojevic, “Automation vs. human intervention: What is the best fit for the best performance ?” *Process Safety Progress*, vol. 24, no. 1, pp. 45–51, 2005.

R. Hansman, Simon et Hunt, “A taxonomy of network and computer attacks”, *Computers & Security*, vol. 24, no. 1, pp. 31–43, 2005.

A. Herzog, N. Shahmehri, et C. Duma, “An ontology of information security”, *International Journal of Information Security and Privacy (IJISP)*, vol. 1, no. 4, pp. 1–23, 2007.

J. Inns, “The evolution and application of siem systems”, *Network Security*, vol. 2014, no. 5, pp. 16–17, 2014.

R. A. Kemmerer et G. Vigna, “Intrusion detection: a brief history and overview”, *Computer*, vol. 35, no. 4, pp. supl27–supl30, 2002.

E. Kostrecová et H. Bínová, “Research paper security information and event management”, *Management*, vol. 4, no. 2, 2015.

C. Kruegel et W. K. Robertson, “Alert verification determining the success of intrusion attempts.” dans *DIMVA*, vol. 4, 2004, pp. 1–14.

C. E. Landwehr, A. R. Bull, J. P. McDermott, et W. S. Choi, “A taxonomy of computer program security flaws”, *ACM Computing Surveys (CSUR)*, vol. 26, no. 3, pp. 211–254, 1994.

Larousse, éd., *Le Petit Larousse illustré 2004*.

W. Li et S. Tian, “Preprocessor of intrusion alerts correlation based on ontology”, dans *Communications and Mobile Computing, 2009. CMC’09. WRI International Conference on*, vol. 3. IEEE, 2009, pp. 460–464.

W. Li, Y. Zhu, et S. Tian, “Intrusion alerts correlation model based on xswrl ontology”, dans *Intelligent Information Technology Application, 2008. IITA’08. Second International Symposium on*, vol. 1. IEEE, 2008, pp. 894–898.

U. Lindqvist, S. Cheung, et R. Valdez, “Correlated attack modeling (cam)”, SRI International, Menlo Park, CA, Rapp. tech., 2003.

Z. Liu, C. Wang, et S. Chen, “Correlating multi-step attack and constructing attack scenarios based on attack pattern modeling”, dans *2008 International Conference on Information Security and Assurance (ISA)*. IEEE, 2008, pp. 214–219.

G. F. Lyon, *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

Mandiant, “M-trends 2017”, Mandiant, Rapp. tech., 2017.

F. Massicotte, M. Couture, Y. Labiche, et L. Briand, “Context-based intrusion detection using snort, nessus and bugtraq databases.” dans *PST*, 2005.

C. McGuinness, Deborah et Welty, “OWL web ontology language guide”, W3C, W3C Recommendation, Fév. 2004, http://www.w3.org/TR/2004/REC-owl-guide-20040210/.

R. Meyer, “Challenges of managing an intrusion detection system (ids) in the enterprise”, *Tech. Rep., SANS Institute InfoSec Reading Room*, 2008.

A. Mokarian, A. Faraahi, et A. G. Delavar, “False positives reduction techniques in intrusion detection systems-a review”, *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 13, no. 10, p. 128, 2013.

C. Moore, “Detecting ransomware with honeypot techniques”, dans *Cybersecurity and Cyberforensics Conference (CCC), 2016*. IEEE, 2016, pp. 77–81.

C. Muehrcke, E. Ruitenbeek, K. Keefe, et W. Sanders, “Characterizing the behavior of cyber adversaries: The means, motive, and opportunity of cyberattacks”, dans *2010 International Conference on Dependable Systems and Networks Supplemental, IEEE/IFIP International Conference on Dependable Systems and Networks*, 2010.

P. Ning, Y. Cui, et D. S. Reeves, “Constructing attack scenarios through correlation of intrusion alerts”, dans *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, 2002, pp. 245–254.

N. F. Noy, D. L. McGuinness *et al.*, “Ontology development 101: A guide to creating your first ontology”, 2001.

L. Olszewski. (2015, Mai) Detecting ransomware with windows event log. En ligne: https://keepitsafe.pl/detecting-ransomware-with-windows-event-log/

J. Pérez, M. Arenas, et C. Gutierrez, "Semantics and complexity of sparql", dans *International semantic web conference*. Springer, 2006, pp. 30–43.

R. Poli, "Ontology for knowledge organization", *Advances in Knowledge Organization*, vol. 5, pp. 313–319, 1996.

X. Qin, "A probabilistic-based framework for infosec alert correlation", Thèse de doctorat, Georgia Institute of Technology, 2005.

Y. Raimond et G. Schreiber, "RDF 1.1 primer", W3C, W3C Note, Juin 2014, http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

A. Razzaq, K. Latif, H. F. Ahmad, A. Hur, Z. Anwar, et P. C. Bloodsworth, "Semantic security against web application attacks", *Information Sciences*, vol. 254, pp. 19–38, 2014.

H. Ren, N. Stakhanova, et A. A. Ghorbani, "An online adaptive approach to alert correlation." dans *DIMVA*, vol. 10. Springer, 2010, pp. 153–172.

RSA Security, Inc., "Rsa netwitness suite". En ligne: https://www.rsa.com/en-us/products/threat-detection-and-response/siem-and-beyond

S. Saad et I. Traore, "Method ontology for intelligent network forensics analysis", dans *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on*. IEEE, 2010, pp. 7–14.

A. Sadighian, "Intrusion detection from heterogenous sensors", Thèse de doctorat, École Polytechnique de Montréal, 2015.

D. Sánchez, A. Moreno, et L. Del Vasto-Terrientes, "Learning relation axioms from text: An automatic web-based approach", *Expert Systems with Applications*, vol. 39, no. 5, pp. 5792–5805, 2012.

D. Shotton, C. Catton, et G. Klyne, "Ontologies for sharing, ontologies for use", 2010. En ligne: http://ontogenesis.knowledgeblog.org/312

A. Simmonds, P. Sandilands, et L. Van Ekert, "An ontology for network security attacks", dans *Asian Applied Computing Conference*. Springer, 2004, pp. 317–323.

C. Simmons, C. Ellis, S. Shiva, D. Dasgupta, et Q. Wu, “Avoidit: A cyber attack taxonomy”, dans *9th Annual Symposium on Information Assurance (ASIA ’14)*, 2014, p. 2.

SonicWall, “2017 annual threat report”, SonicWall, Rapp. tech., 2017.

A. K. Sood et R. J. Enbody, “Targeted cyberattacks: a superset of advanced persistent threats”, *IEEE security & privacy*, vol. 11, no. 1, pp. 54–61, 2013.

A. Souag, C. Salinesi, et I. Comyn-Wattiau, “Ontologies for security requirements: A literature survey and classification”, dans *Advanced Information Systems Engineering Workshops*. Springer, 2012, pp. 61–69.

Splunk, Inc., “Splunk”. En ligne: `https://www.splunk.com`

P. Spyns, R. Meersman, et M. Jarrar, “Data modelling versus ontology engineering”, *ACM SIGMod Record*, vol. 31, no. 4, pp. 12–17, 2002.

R. Studer, V. R. Benjamins, et D. Fensel, “Knowledge engineering: principles and methods”, *Data & knowledge engineering*, vol. 25, no. 1-2, pp. 161–197, 1998.

Symantec, “Internet security threat report”, Symantec Corporation, Rapp. tech., Avr. 2016.

M. Uschold et M. King, *Towards a methodology for building ontologies*. Artificial Intelligence Applications Institute, University of Edinburgh Edinburgh, 1995.

R. P. van Heerden, B. Irwin, et I. Burke, “Classifying network attack scenarios using an ontology”, dans *Proceedings of the 7th International Conference on Information-Warfare & Security (ICIW 2012)*, 2012, pp. 311–324.

Verizon, “Data breach digest”, Verizon, Rapp. tech., 2016.

——, “Data breach digest”, Verizon, Rapp. tech., 2017.

N. Virvilis et D. Gritzalis, “The big four-what we did wrong in advanced persistent threat detection ?” dans *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*. IEEE, 2013, pp. 248–254.

M. Zissman, “Darpa intrusion detection scenario specific data sets”, 2000.

ANNEXE A Présentation des éléments de l'ontologie

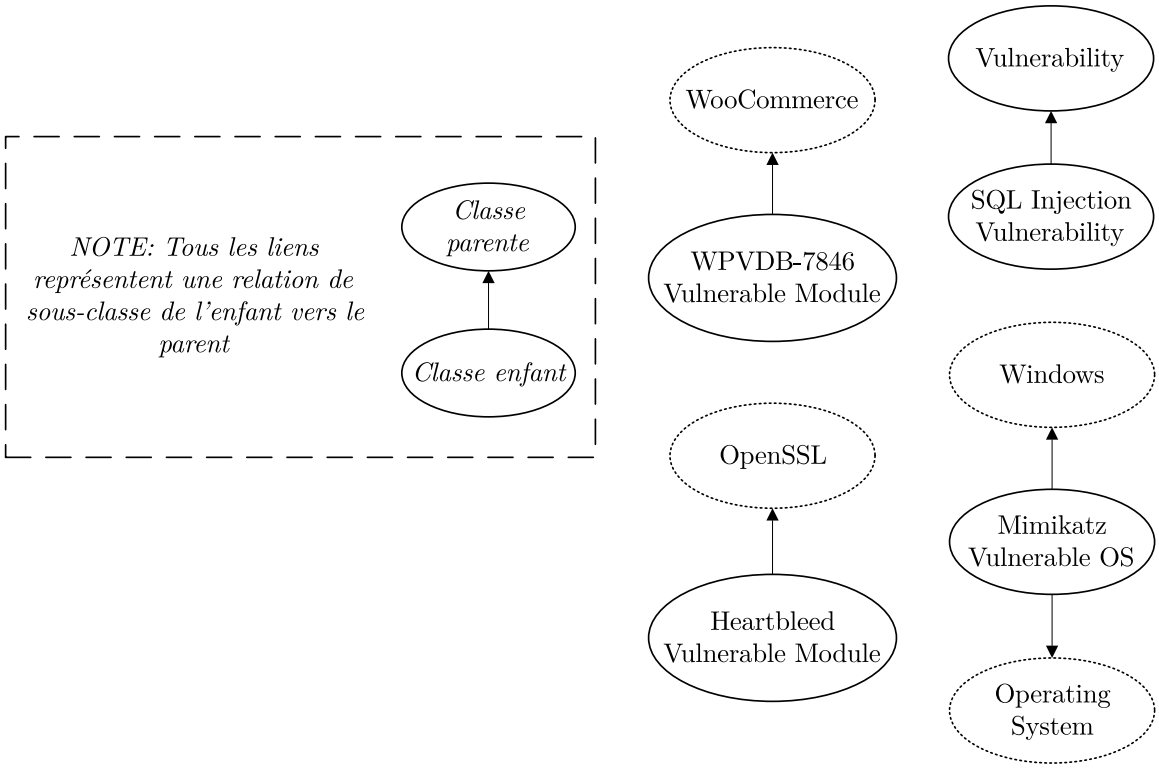

Figure A.1 Hiérarchie des classes reliées au concept de vulnérabilité

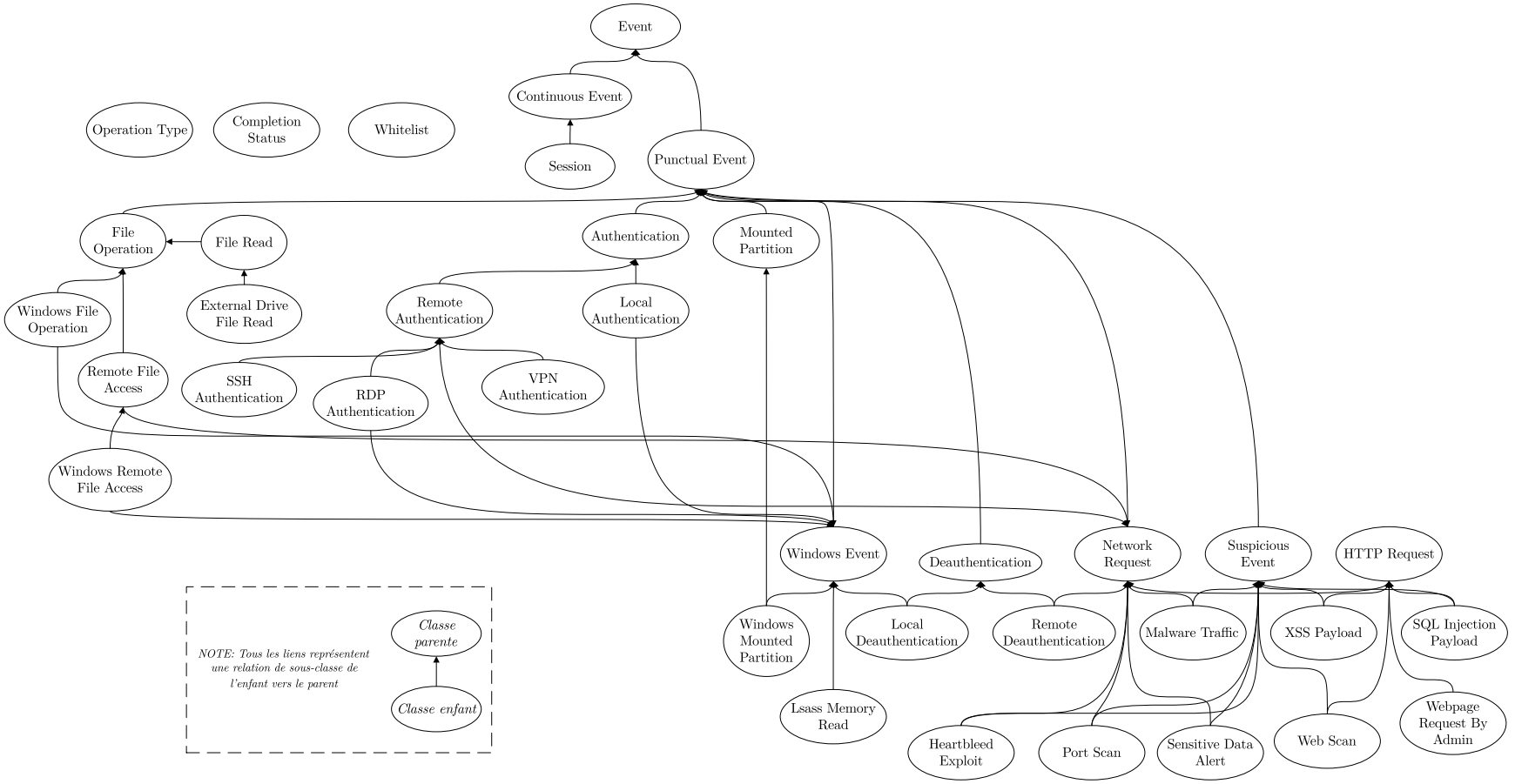

Figure A.2 Hiérarchie des classes reliées au concept d'événement

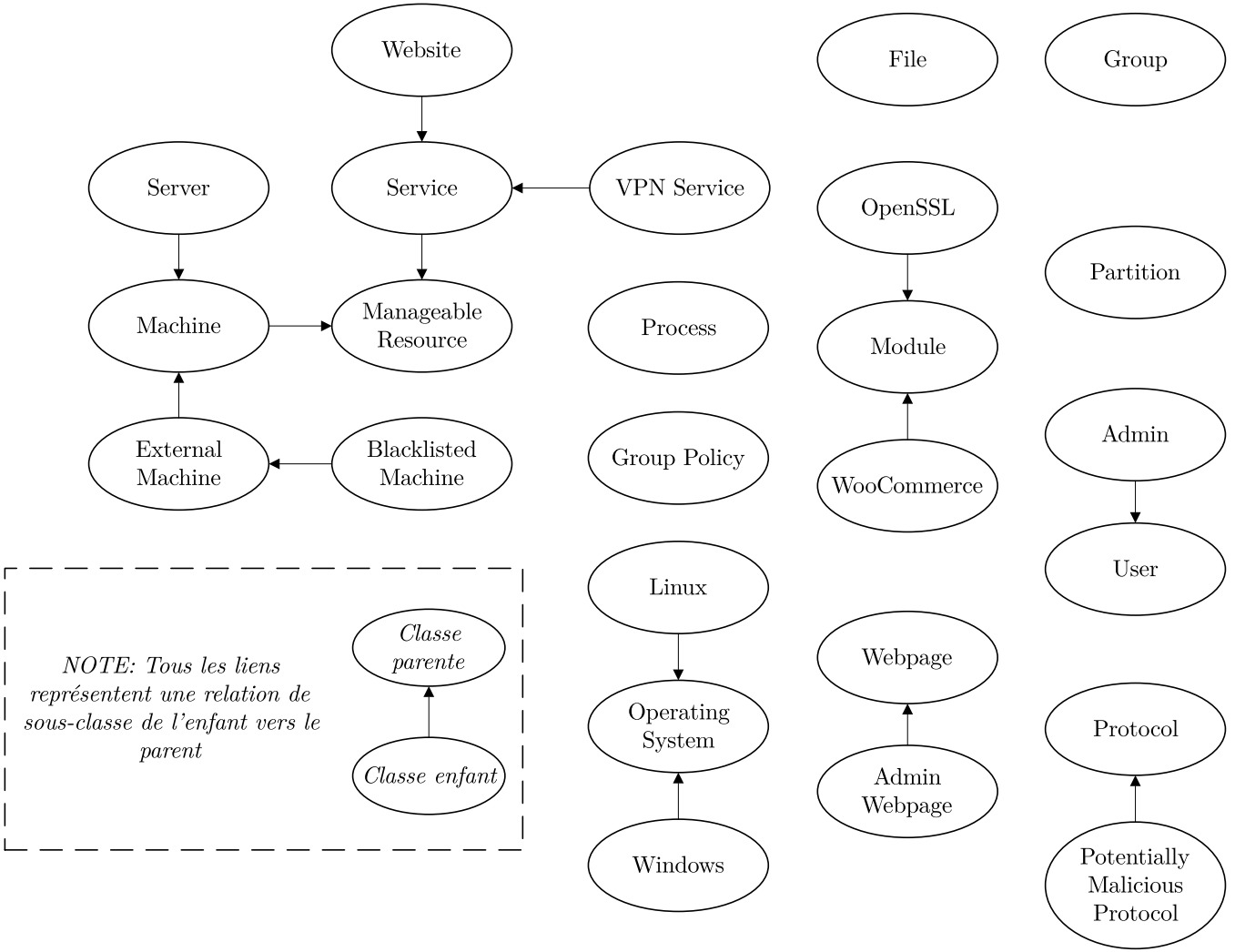

Figure A.3 Hiérarchie des classes reliées au concept de contexte

Tableau A.1 Propriétés d'objets

Propriété	Domaine	Portée	*Inverse of*	Fonc.	Fonc. inv.	Asym.	Irrefl.
happenedOn	Event	Machine		✓		✓	✓
hasAdminRightsOn	GroupPolicy or User		isAdministratedBy			✓	✓
hasCompletionStatus	Event	CompletionStatus		✓		✓	✓
hasDestination	Event					✓	✓
hasEndingEvent	ContinuousEvent	PunctualEvent		✓		✓	✓
hasFile	Event	File		✓		✓	✓
hasGroup	Machine or User	Group	hasMember			✓	✓
hasGroupPolicy	Group	GroupPolicy				✓	✓
hasMember	Group	Machine or User	hasGroup			✓	✓
hasModule	Service	Module				✓	✓
hasOperatingSystem	Machine	OperatingSystem				✓	✓
hasOperationType	FileOperation	OperationType		✓		✓	✓
hasPartition		Partition	isPartitionOf			✓	✓
hasProcess	Event	Process		✓		✓	✓
hasProtocol	NetworkRequest	Protocol				✓	✓
hasSource	Event	Machine or User				✓	✓
hasStartingEvent	ContinuousEvent	PunctualEvent		✓		✓	✓
hasUser	Event	User		✓		✓	✓
hasVulnerability	Machine or Service	Vulnerability				✓	✓
hasWebpage	Website	Webpage	isWebpageOf		✓	✓	✓
isAdministratedBy			hasAdminRightsOn			✓	✓
isOfferedBy			offers	✓		✓	✓
isPartitionOf			hasPartition			✓	✓
isWebpageOf			hasWebpage	✓		✓	✓
offers	Machine	Service	isOfferedBy		✓	✓	✓

Tableau A.2 Propriétés de données

Propriété	Domaine	Portée	Propriété parente	Fonc.
happenedOnHostname	Event	xsd:string		✓
hasAdminRightsOnGroupName	GroupPolicy	xsd:string		
hasAdminRightsOnName	GroupPolicy	xsd:string		
hasDestinationHostname	Event	xsd:string		✓
hasDestinationIPAddress	NetworkRequest	xsd:string		✓
hasDestinationPort	NetworkRequest	xsd:integer		✓
hasFileName	Event	xsd:string		✓
hasIdentifier		xsd:string		
hasDomainName	HTTPRequest or Website	xsd:string	hasIdentifier	✓
hasHostname	Machine	xsd:string	hasIdentifier	✓
hasUsername	Event	xsd:string	hasIdentifier	✓
hasIPAddress	Machine	xsd:string		
hasLogonType	WindowsEvent	xsd:integer		✓
hasMachineMemberName	Group	xsd:string		
hasName		xsd:string		✓
hasPartitionName	Event	xsd:string		✓
hasPath	Event of File or Webpage	xsd:string		
hasPort	Service	xsd:integer		
hasProcessName	Event	xsd:string		✓
hasSourceIPAddress	NetworkRequest	xsd:string		✓
hasSourcePort	NetworkRequest	xsd:integer		✓
hasTimestamp	Event	xsd:dateTime		
hasEndTimestamp	Event	xsd:dateTime	hasTimestamp	✓
hasStartTimestamp	Event	xsd:dateTime	hasTimestamp	✓
hasUserMemberName	Group	xsd:string		

Tableau A.3 – Disjonctions des classes

Classe	owl:disjointWith
CompletionStatus	Event, File, Group, GroupPolicy, ManageableResource, Module, OperatingSystem, OperationType, Partition, Process, Protocol, User, Vulnerability, Webpage, Whitelist
Event	CompletionStatus, File, Group, GroupPolicy, ManageableResource, Module, OperatingSystem, OperationType, Partition, Process, Protocol, User, Vulnerability, Webpage, Whitelist
ContinuousEvent	PunctualEvent
PunctualEvent	ContinuousEvent
Authentication	Deauthentication, FileOperation, MountedPartition, SuspiciousEvent, LsassMemoryRead
LocalAuthentication	RemoteAuthentication, NetworkRequest
RemoteAuthentication	LocalAuthentication
RDPAuthentication	SSHAuthentication, VPNAuthentication
SSHAuthentication	RDPAuthentication, VPNAuthentication
VPNAuthentication	SSHAuthentication, RDPAuthentication
Deauthentication	Authentication, FileOperation, MountedPartition, SuspiciousEvent, LsassMemoryRead
LocalDeauthentication	RemoteDeauthentication, NetworkRequest
RemoteDeauthentication	LocalDeauthentication
FileOperation	Authentication, Deauthentication, MountedPartition, HTTPRequest, SuspiciousEvent, LsassMemoryRead
MountedPartition	Authentication, Deauthentication, FileOperation, NetworkRequest, SuspiciousEvent, LsassMemoryRead
NetworkRequest	LocalAuthentication, LocalDeauthentication, MountedPartition, LsassMemoryRead
HeartbleedExploit	HTTPRequest, PortScan, RemoteAuthentication, RemoteDeauthentication, RemoteFileAccess
HTTPRequest	PortScan, RDPAuthentication, SSHAuthentication, VPNAuthentication, HeartbleedExploit

SQLInjectionPayload	WebpageRequestByAdmin, XSSPayload, HeartbleedExploit, RemoteFileAccess
WebpageRequestByAdmin	XSSPayload, SQLInjectionPayload, WebScan
WebScan	WebpageRequestByAdmin, HeartbleedExploit, SQLInjectionPayload, RemoteAuthentication, RemoteDeauthentication, RemoteFileAccess
XSSPayload	WebpageRequestByAdmin, SQLInjectionPayload, HeartbleedExploit, RemoteAuthentication, RemoteDeauthentication, RemoteFileAccess
PortScan	HeartbleedExploit, HTTPRequest, RemoteAuthentication, RemoteDeauthentication, RemoteFileAccess
SensitiveDataAlert	RemoteAuthentication, RemoteDeauthentication, XSSPayload, WebScan, WebpageRequestByAdmin, RemoteFileAccess
SuspiciousEvent	Authentication, Deauthentication, FileOperation, MountedPartition, LsassMemoryRead
WindowsEvent	SuspiciousEvent, HTTPRequest
LsassMemoryRead	Authentication, Deauthentication, FileOperation, MountedPartition, NetworkRequest, SuspiciousEvent
OperationType	CompletionStatus, Event, File, Group, GroupPolicy, ManageableResource, Module, OperatingSystem, Partition, Process, Protocol, User, Vulnerability, Webpage, Whitelist
Whitelist	CompletionStatus, Event, File, Group, GroupPolicy, ManageableResource, Module, OperatingSystem, OperationType, Partition, Process, Protocol, User, Vulnerability, Webpage
File	CompletionStatus, Event, Group, GroupPolicy, ManageableResource, Module, OperatingSystem, OperationType, Partition, Process, Protocol, User, Vulnerability, Webpage, Whitelist, Whitelist
Group	CompletionStatus, Event, File, GroupPolicy, ManageableResource, Module, OperatingSystem, OperationType, Partition, Process, Protocol, User, Vulnerability, Webpage, Whitelist
GroupPolicy	CompletionStatus, Event, File, Group, ManageableResource, Module, OperatingSystem, OperationType, Partition, Process, Protocol, User, Vulnerability, Webpage, Whitelist

ManageableResource	CompletionStatus, Event, File, Group, GroupPolicy, Module, OperatingSystem, OperationType, Partition, Process, Protocol, User, Vulnerability, Webpage, Whitelist
Machine	Service
Service	Machine
VPNService	Website
Website	VPNService
Module	CompletionStatus, Event, File, Group, GroupPolicy, ManageableResource, OperatingSystem, OperationType, Partition, Process, Protocol, User, Vulnerability, Webpage, Whitelist
OpenSSL	WooCommerce
WooCommerce	OpenSSL
OperatingSystem	CompletionStatus, Event, File, Group, GroupPolicy, ManageableResource, Module, OperationType, Partition, Process, Protocol, User, Vulnerability, Webpage, Whitelist
Linux	Windows
Windows	Linux
Partition	CompletionStatus, Event, File, Group, GroupPolicy, ManageableResource, Module, OperatingSystem, OperationType, Process, Protocol, User, Vulnerability, Webpage, Whitelist
Process	CompletionStatus, Event, File, Group, GroupPolicy, ManageableResource, Module, OperatingSystem, OperationType, Partition, Protocol, User, Vulnerability, Webpage, Whitelist
Protocol	CompletionStatus, Event, File, Group, GroupPolicy, ManageableResource, Module, OperatingSystem, OperationType, Partition, Process, User, Vulnerability, Webpage, Whitelist
User	CompletionStatus, Event, File, Group, GroupPolicy, ManageableResource, Module, Webpage, OperatingSystem, OperationType, Partition, Process, Protocol, Vulnerability, Whitelist
Webpage	CompletionStatus, Event, File, Group, GroupPolicy, ManageableResource, Module, OperatingSystem, OperationType, Partition, Process, Protocol, User, Vulnerability, Whitelist

Vulnerability	CompletionStatus, Event, File, Group, GroupPolicy, ManageableResource, Module, Operating-System, OperationType, Partition, Process, Protocol, User, Webpage, Whitelist

Tableau A.4 – Règles

Index	Type	Règle
T004	SWRL	Event(?event) ∧ hasPartitionName(?event, ?X) ∧ happenedOnHostname(?event, ?Y) ∧ Machine(?machine) ∧ hasName(?partition, ?X) ∧ hasPartition(?machine, ?partition) ∧ hasHostname(?machine, ?Y) → hasPartition(?event, ?partition)
T005	SWRL	Event(?event) ∧ happenedOnHostname(?event, ?X) ∧ hasHostname(?machine, ?X) ∧ Machine(?machine) → happenedOn(?event, ?machine)
T006	SWRL	MountedPartition(?event1) ∧ FileRead(?event2) ∧ hasPartition(?event1, ?partition) ∧ hasPartition(?event2, ?partition) ∧ hasTimestamp(?event1, ?time1) ∧ hasTimestamp(?event2, ?time2) ∧ swrlb:lessThan(?time1, ?time2) → ExternalDriveFileRead(?event2)
T008	SWRL	NetworkRequest(?event) ∧ hasDestinationPort(?event, 80) ∧ hasProtocol(?event, TCP) → hasProtocol(?event, HTTP)
T009	SWRL	NetworkRequest(?event) ∧ hasDestinationPort(?event, 443) ∧ hasProtocol(?event, TCP) → hasProtocol(?event, SSL)
T010	SWRL	NetworkRequest(?event) ∧ hasDestinationPort(?event, 6667) ∧ hasProtocol(?event, TCP) → hasProtocol(?event, IRC)
T011	SWRL	NetworkRequest(?event) ∧ hasDestinationPort(?event, 53) ∧ hasProtocol(?event, UDP) → hasProtocol(?event, DNS)
T012	SWRL	NetworkEvent(?event) ∧ hasDestinationIPAddress(?event, ?x) ∧ hasIPAddress(?machine, ?x) → hasDestination(?event, ?machine)

T013	SWRL	NetworkEvent(?event) ∧ hasSourceIPAddress(?event, ?x) ∧ hasIPAddress(?machine, ?x) → hasSource(?event, ?machine)
T014	Algorithme	Requête toutes les instances de :NetworkEvent qui possèdent :hasDestinationIPAddress X ou :hasSourceIPAddress X. S'il n'y a pas d'instance de :Machine avec l'adresse IP X, crée (?newMachine rdf:type :ExternalMachine ; ?newMachine :hasIPAddress X) où :ExternalMachine est sous-classe de :Machine
T015	LD	MalwareTraffic equivalentTo `NetworkRequest and (hasDestination some BlacklistedMachine) and (hasProtocol some PotentiallyMaliciousProtocol)`
T018	SWRL	Event(?event) ∧ hasPath(?event, ?path) ∧ hasFileName(?event, ?name) ∧ File(?file) ∧ hasPath(?file, ?path) ∧ hasName(?file, ?name) → hasFile(?event, ?file)
T020	SWRL	Machine(?machine) ∧ hasOperatingSystem(?machine, ?os) ∧ MimikatzVulnerableOS(?os) → hasVulnerability(?machine, mimikatz)
T021	SWRL	Event(?event) ∧ hasUsername(?event, ?username) ∧ User(?user) ∧ hasName(?user, ?username) → hasUser(?event, ?user)
T022	SWRL	hasAdminRightsOnName(?policy, ?identifier) ∧ hasIdentifier(?target, ?identifier) ∧ ManageableResource(?target) → hasAdminRightsOn(?policy, ?target)
T023	SWRL	Group(?group) ∧ hasUserMemberName(?group, ?memberName) ∧ User(?user) ∧ hasName(?user, ?memberName) → hasMember(?group, ?user)
T024	SWRL	Group(?group) ∧ hasMachineMemberName(?group, ?hostname) ∧ Machine(?machine) ∧ hasHostname(?machine, ?hostname) → hasMember(?group, ?machine)
T025	SWRL	hasMember(?group, ?user) ∧ hasAdminRightsOn(?policy, ?target) ∧ User(?user) ∧ hasGroupPolicy(?group, ?policy) ∧ ManageableResource(?target) → hasAdminRightsOn(?user, ?target)

T026	SWRL	hasAdminRightsOnGroupName(?policy, ?name) ∧ hasName(?group, ?name) → hasAdminRightsOn(?policy, ?group)
T027	SWRL	hasGroupPolicy(?groupG, ?policy) ∧ hasAdminRightsOn(?policy, ?groupH) ∧ Group(?groupH) ∧ hasMember(?groupH, ?machine) ∧ Machine(?machine) → hasAdminRightsOn(?policy, ?machine)
T030	SWRL	HTTPRequest(?event) ∧ hasDomainName(?event, ?domainName) ∧ Website(?website) ∧ hasDomainName(?website, ?domainName) ∧ hasPath(?event, ?path) ∧ hasWebpage(?website, ?webpage) ∧ hasPath(?webpage, ?path) → hasDestination(?event, ?webpage)
T031	SWRL	HTTPRequest(?request) ∧ hasDestination(?request, ?webpage) ∧ Webpage(?webpage) ∧ Website(?website) ∧ hasWebpage(?website, ?webpage) → hasDestination(?request, ?website)
T032	SWRL	hasAdminRightsOn(?user, ?website) ∧ hasDestination(?event, ?website) ∧ hasSource(?event, ?user) ∧ User(?user) ∧ HTTPRequest(?event) ∧ Website(?website) → WebpageRequestByAdmin(?event)
T033	Algorithme	Pour chaque paire d'instances de {:Deauthentication,:Authentication} avec les mêmes objets pour :hasUser et :happenedOn et qui suivent une séquentialité temporelle, crée une instance de :Session avec: • :hasStartTimestamp some xsd:dateTime • :hasEndTimestamp some xsd:dateTime • :hasUser some :User • :happenedOn some :Machine
T036	SWRL	HTTPRequest(?event) ∧ hasSource(?event, ?machine) ∧ Machine(?machine) ∧ hasTimestamp(?event, ?requestTime) ∧ Session(?session) ∧ hasUser(?session, ?user) ∧ happenedOn(?session, ?machine) ∧ hasStartTimestamp(?session, ?loginTime) ∧ hasEndTimestamp(?session, ?logoutTime) ∧ swrlb:lessThan(?loginTime, ?requestTime) ∧ swrlb:lessThan(?requestTime, ?logoutTime) → hasSource(?event, ?user)
T038	SWRL	hasModule(?website, ?module) ∧ WPVDB-7846VulnerableModule(?module)

		→ hasVulnerability(?website, WPVDB-7846)
T039	Algorithme	Requête toutes les instances de :ExternalMachine et si leurs adresses IP se retrouvent dans la liste noire « Google Safe Browsing », elles sont sous-classes de :BlacklistedMachine
T044	SWRL	Service(?service) ∧ hasModule(?service, ?module) ∧ HeartbleedVulnerableModule(?module) → hasVulnerability(?service, heartbleed)
T045	SWRL	Machine(?machine) ∧ offers(?machine, ?service) ∧ hasVulnerability(?service, ?vulnerability) → hasVulnerability(?machine, ?vulnerability)
T050	SWRL	NetworkRequest(?event) ∧ hasDestinationHostname(?event, ?X) ∧ hasHostname(?machine, ?X) ∧ Machine(?machine) → hasDestination(?event, ?machine)
T052	Algorithme	Requête toutes les instances de :Event avec :hasProcessName X et s'il n'y a pas d'instances de :Process qui possède :hasName X, créé l'instance de :Process: (?newProcess rdf:type :Process ; ?newProcess :hasName X)
T053	SWRL	Event(?event) ∧ hasProcessName(?event, ?processName) ∧ Process(?process) ∧ hasName(?process, ?processName) → hasProcess(?event, ?process)
T060	SWRL	NetworkRequest(?request) ∧ hasDestinationPort(?request, ?port) ∧ hasDestinationIPAddress(?request, ?dstip) ∧ hasIPAddress(?machine, ?dstip) ∧ offers(?machine, ?service) ∧ hasPort(?service, ?port) → hasDestination(?request, ?service)

ANNEXE B Présentation des composantes de DIOSE

Tableau B.1 – Requêtes SPARQL

Index	Requête

R001	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :WindowsEvent .
	?event :hasIdentifier 98
	}

R002	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :WindowsEvent .
	?event :hasIdentifier 4663
	}

R003	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :FileOperation .
	?event :hasOperationType :read
	}

R004	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :MountedPartition .
	?event :hasPartition ?partition
	}

R005	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :FileRead .
	?event :hasPartition ?partition
	}

R006	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :ExternalDriveFileRead .
	?event :happenedOn ?machine .
	?event :hasPartition ?partition
	}

R007	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :NetworkRequest
	{
	?event :hasDestinationPort 6667 .
	?event :hasProtocol :TCP
	} UNION {
	?event :hasDestinationPort 80 .
	?event :hasProtocol :TCP
	} UNION {
	?event :hasDestinationPort 443 .
	?event :hasProtocol :TCP
	} UNION {
	?event :hasDestinationPort 53 .
	?event :hasProtocol :UDP
	}
	}

R008	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#>

	SELECT * WHERE {
	?event :hasDestination ?machineZ .
	?event :hasSource ?machineY
	{ ?event :hasProtocol :HTTP } UNION
	{ ?event :hasProtocol :SSL } UNION
	{ ?event :hasProtocol :DNS } UNION
	{ ?event :hasProtocol :IRC }
	}

R009	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?machine rdf:type :BlacklistedMachine
	}

R010	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :MalwareTraffic .
	?event :hasSource ?machine1 .
	?machine1 rdf:type :Machine .
	?event :hasDestination ?machine2 .
	?machine2 rdf:type :Machine
	}

R011	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :WindowsEvent .
	?event :hasIdentifier 5140 .
	?event :hasCompletionStatus :failure
	}

R012	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#> SELECT * WHERE {

	?event rdf:type :RemoteFileAccess . ?event :hasFile ?file . ?event :hasCompletionStatus :failure . ?event :hasSource ?machine . ?machine rdf:type :Machine }

R013	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?machine rdf:type :Machine
	{ ?machine :hasOperatingSystem :windows7 } UNION
	{ ?machine :hasOperatingSystem :windowsXP } UNION
	{ ?machine :hasOperatingSystem :windowsVista } UNION
	{ ?machine :hasOperatingSystem :windows8 }
	}

R014	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?machine rdf:type :Machine .
	?machine :hasVulnerability :mimikatz
	}

R015	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :RemoteAuthentication .
	?event :happenedOn ?machine .
	?event :hasUser ?user
	}

R016	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#> SELECT * WHERE { ?policy rdf:type :GroupPolicy .

	?policy :hasAdminRightsOn ?target . ?target rdf:type :ManageableResource }

R017	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?group :hasMember ?member
	{ ?member rdf:type :Machine } UNION
	{ ?member rdf:type :User } UNION
	{ ?member rdf:type :Service }
	}

R018	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?user rdf:type :User .
	?group :hasMember ?user .
	?group :hasGroupPolicy ?policy .
	?policy :hasAdminRightsOn ?target .
	?target rdf:type :ManageableResource
	}

R019	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?user :hasAdminRightsOn ?target .
	?user rdf:type :User .
	?target rdf:type :ManageableResource
	}

R020	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#> SELECT * WHERE { ?user rdf:type :User . ?groupG :hasMember ?user .

	?groupG :hasGroupPolicy ?policy . ?groupH rdf:type :Group . ?policy :hasAdminRightsOn ?groupH . ?groupH :hasMember ?machine . ?machine rdf:type :Machine FILTER (?groupG != ?groupH) }

R021	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :RemoteAuthentication .
	?event :happenedOn ?machine .
	?event :hasUser ?user .
	?user :hasAdminRightsOn ?machine
	}

R022	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :WindowsEvent .
	?event :hasIdentifier 3065
	}

R023	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :LsassMemoryRead .
	?event :happenedOn ?machine
	}

R024	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#> SELECT * WHERE { ?machine :hasVulnerability :mimikatz . ?event1 rdf:type :Authentication .

	?event1 :happenedOn ?machine . ?event1 :hasUser ?user . ?user :hasAdminRightsOn ?machine . ?event2 rdf:type :LsassMemoryRead . ?event2 :happenedOn ?machine . ?event1 :hasTimestamp ?time1 . ?event2 :hasTimestamp ?time2 FILTER (?time1 < ?time2) }

R025	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :WindowsEvent .
	?event :hasIdentifier 5140 .
	?event :hasCompletionStatus :success
	}

R026	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :RemoteFileAccess .
	?event :hasFile ?file .
	?event :hasCompletionStatus :success .
	?event :hasSource ?machine .
	?machine rdf:type :Machine
	}

R027	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#> SELECT * WHERE { ?event rdf:type :WebScan . ?event :hasDestination ?webpage . ?webpage rdf:type :Webpage . ?webpage :isWebpageOf ?website . ?event :hasSource ?machine .

	?machine rdf:type :Machine }

R028	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :WebScan .
	?event :hasDestination ?webpage .
	?webpage rdf:type :Webpage
	}

R029	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :XSSPayload .
	?event :hasDestination ?webpage .
	?webpage rdf:type :Webpage
	}

R030	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?user :hasAdminRightsOn ?target .
	?user rdf:type :Admin .
	?target rdf:type :Machine
	}

R031	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?user :hasAdminRightsOn ?target .
	?user rdf:type :Admin .
	?target rdf:type :Website
	}

R032	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#>

	SELECT * WHERE {
	{ ?event rdf:type :Authentication } UNION
	{ ?event rdf:type :Deauthentication }
	?event :happenedOn ?machine .
	?event :hasUser ?user
	}

R033	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :HTTPRequest .
	?event :hasSource ?machine .
	?event :hasTimestamp ?requestTime .
	?session rdf:type :Session .
	?session :hasUser ?user .
	?session :happenedOn ?machine .
	?session :hasEndTimestamp ?logoutTime .
	?session :hasStartTimestamp ?loginTime
	FILTER (?loginTime < ?requestTime)
	FILTER (?requestTime < ?logoutTime)
	}

R034	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :HTTPRequest .
	?event :hasSource ?machine .
	?event :hasTimestamp ?time
	}

R035	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#> SELECT * WHERE { ?event rdf:type :HTTPRequest . ?event :hasDestination ?webpage .

	?webpage rdf:type :Webpage . ?event :hasSource ?user . ?user rdf:type :User . ?website :hasWebpage ?webpage }

R036	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :WebpageRequestByAdmin .
	?event :hasDestination ?webpage .
	?webpage rdf:type :Webpage
	}

R037	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event1 rdf:type :XSSPayload .
	?event1 :hasDestination ?webpage .
	?event2 rdf:type :WebpageRequestByAdmin .
	?event2 :hasDestination ?webpage .
	?webpage rdf:type :Webpage .
	?event1 :hasSource ?machine .
	?website :hasWebpage ?webpage .
	?event1 :hasTimestamp ?time1 .
	?event2 :hasTimestamp ?time2
	FILTER (?time1 < ?time2)
	}

R038	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#> SELECT * WHERE { ?event rdf:type :HTTPRequest . ?event :hasSource ?machine . ?event :hasDestination ?webpage . ?website :hasWebpage ?webpage .

	?webpage rdf:type :AdminWebpage }

R039	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :SQLInjectionPayload .
	?event :hasSource ?machine .
	?event :hasDestination ?page .
	?website :hasWebpage ?page .
	?page rdf:type :AdminWebpage
	}

R040	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?website :hasVulnerability ?vuln .
	?vuln rdf:type :SQLInjectionVulnerability .
	?website rdf:type :Website
	}

R041	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :SQLInjectionPayload .
	?event :hasSource ?machine .
	?event :hasDestination ?webpage .
	?website :hasWebpage ?webpage .
	?webpage rdf:type :AdminWebpage .
	?website :hasVulnerability ?vuln .
	?vuln rdf:type :SQLInjectionVulnerability
	}

R042	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#> SELECT * WHERE {

	?event rdf:type :SensitiveDataAlert . ?event :hasDestination ?machine1 . ?machine1 rdf:type :ExternalMachine . ?event :hasSource ?machine2 . ?machine2 rdf:type :Machine }

R043	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :NetworkRequest .
	?event :hasSource ?machine .
	?machine :offers ?website .
	?website rdf:type :Website
	}

R044	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?machine rdf:type :Server
	}

R045	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :PortScan .
	?event :hasDestination ?machine .
	?machine rdf:type :Machine
	}

R046	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#> SELECT * WHERE { ?event rdf:type :PortScan . ?event :hasDestination ?machine . ?machine rdf:type :Server

	}

R047	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?heartbleedExploit rdf:type :HeartbleedExploit .
	?heartbleedExploit :hasSource ?attackerMachine .
	?heartbleedExploit :hasDestination ?service .
	?attackerMachine rdf:type :Machine .
	?service rdf:type :Service
	}

R048	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?service rdf:type :Service .
	?service :hasVulnerability :heartbleed
	}

R049	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?service rdf:type :Service
	{ ?service :hasModule :openSSL1.0.1a } UNION
	{ ?service :hasModule :openSSL1.0.1b } UNION
	{ ?service :hasModule :openSSL1.0.1c } UNION
	{ ?service :hasModule :openSSL1.0.1d } UNION
	{ ?service :hasModule :openSSL1.0.1e } UNION
	{ ?service :hasModule :openSSL1.0.1f }
	}

R050	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#> SELECT * WHERE { ?event rdf:type :HeartbleedExploit . ?event :hasSource ?attackerMachine .

	?event :hasDestination ?service . ?service :hasVulnerability :heartbleed . ?service rdf:type :Service . ?machine :offers ?service }

R051	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :VPNAuthentication .
	?event :hasSource ?machine .
	?machine rdf:type :Machine
	}

R052	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :VPNAuthentication .
	?event :hasUser ?user
	}

R053	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :VPNAuthentication .
	?event :hasUser ?user .
	?event :hasSource ?machine .
	?event :hasDestination ?service .
	?service rdf:type :VPNService .
	?machine rdf:type :Machine
	}

R054	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#> SELECT * WHERE { ?event rdf:type :WindowsEvent .

	?event :hasIdentifier 4624 }

R055	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :RDPAuthentication .
	?event :hasUser ?user .
	?event :hasDestination ?machine .
	?machine rdf:type :Machine
	}

R056	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :RemoteAuthentication .
	?event :hasUser ?user .
	?event :hasDestination ?machine .
	?machine rdf:type :Machine
	}

R057	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?event rdf:type :SSHAuthentication .
	?event :hasUser ?user .
	?event :hasDestination ?machine .
	?machine rdf:type :Machine
	}

R058	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#> SELECT * WHERE { ?event rdf:type :FileOperation . ?event :hasOperationType :write . ?event :happenedOn ?machine .

	?event :hasProcess ?process . ?process rdf:type :Process . :whitelistedWriteProcesses rdf:li ?process2 FILTER (?process != ?process2) }

R059	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT ?machine WHERE {
	?event rdf:type :FileOperation .
	?event :hasOperationType :write .
	?event :happenedOn ?machine .
	?machine rdf:type :Machine .
	?event :hasProcess ?process .
	?process rdf:type :Process .
	:whitelistedWriteProcesses rdf:li ?process2
	FILTER (?process != ?process2)
	}
	GROUP BY ?machine
	HAVING (COUNT(?event) > 100)

R060	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	:whitelistedWriteProcesses rdf:li ?process
	}

R061	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX : <http://www.polymtl.ca/ontologies/siem#>
	
	SELECT * WHERE {
	?service :hasModule ?module .
	?module rdf:type :HeartbleedVulnerableModule .
	?service rdf:type :Service
	}

R062	PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX : <http://www.polymtl.ca/ontologies/siem#>

```
SELECT * WHERE {
        { ?service :hasModule :openSSL1.0.1a } UNION
        { ?service :hasModule :openSSL1.0.1b } UNION
        { ?service :hasModule :openSSL1.0.1c } UNION
        { ?service :hasModule :openSSL1.0.1d } UNION
        { ?service :hasModule :openSSL1.0.1e } UNION
        { ?service :hasModule :openSSL1.0.1f }
        ?service rdf:type :Service
}
```

Tableau B.2 – Pilotes liés aux senseurs

Index	Description du pilote lié au senseur
S001	S'interface avec l'observateur d'événements Windows et crée une instance de :WindowsEvent pour chaque événement avec ces propriétés, lorsqu'applicables : • :hasIdentifier (p. ex. 4663) • :happenedOnHostname (p. ex. « ALICE-PC ») • :hasPartitionName (p. ex. « F: ») • :hasOperationType :OperationType (p. ex. :read) • :hasFileName (p. ex. « fichier.txt ») • :hasUsername (p. ex. « Alice ») • :hasCompletionStatus :CompletionStatus (p. ex. :success) • :hasLogonType (p. ex. 2)
S002	S'interface avec la journalisation du système d'explotation Linux et crée une instance de :MountedPartition avec ces propriétés, lorsqu'applicables : • :happenedOnHostname (p. ex. « ALICE-PC ») • :hasPartitionName (p. ex. « sdc ») • :hasTimestamp xsd:dateTime
S003	S'interface avec la journalisation « auditd » et crée une instance de :FileOperation avec ces propriétés, lorsqu'applicables : • :hasOperationType :OperationType (p. ex. :read) • :hasFileName (p. ex. « fichier.txt ») • :hasPartitionName (p. ex. « sdc ») • :hasUID (p. ex. 103) • :happenedOnHostname (p. ex. « ALICE-PC ») • :hasTimestamp xsd:dateTime
S004	S'interface avec la journalisation du pare-feu et crée une instance de :NetworkRequest avec ces propriétés, lorsqu'applicables : • :hasSourceIPAddress (p. ex. « 10.7.11.8 ») • :hasDestinationPort (p. ex. 80) • :hasSourcePort (p. ex. 40947) • :hasDestinationIPAddress (p. ex. « 96.20.12.34 ») • :hasProtocol :Protocol (p. ex. :TCP)
S006	S'interface avec la journalisation de l'IDS Snort et crée une instance de :WebScan, :SQLInjectionPayload, :SensitiveDataAlert, :PortScan ou :HeartbleedExploit avec ces propriétés, lorsqu'applicables :

	• :hasDomainName (p. ex. « www.site.lan ») • :hasPath (p. ex. « /index ») • :hasDestinationPort (p. ex. 80) • :hasSourceIPAddress (p. ex. « 10.7.11.8 ») • :hasDestinationIPAddress (p. ex. « 96.20.12.34 »)
S007	S'interface avec la journalisation de ModSecurtity et crée une instance de :XSSPayload avec ces propriétés, lorsqu'applicables : • :hasPath (p. ex. « /home ») • :hasDomainName (p. ex. « www.site.lan ») • :hasSourceIPAddress (p. ex. « 10.7.11.8 ») • :hasAccuracy (p. ex. 8)
S008	S'interface avec la journalisation du serveur Web et crée une instance de :HTTPRequest avec ces propriétés, lorsqu'applicables : • :hasDomainName (p. ex. « www.site.lan ») • :hasPath (p. ex. « /admin ») • :hasSourceIPAddress (p. ex. « 10.7.11.8 »)
S009	S'interface avec la journalisation du serveur VPN et crée une instance de :VPNAuthentication avec ces propriétés, lorsqu'applicables : • :hasDestinationIPAddress (p. ex. « 96.20.12.34 ») • :hasSourceIPAddress (p. ex. « 10.7.11.8 ») • :hasUserName (p. ex. « Alice »)
S010	S'interface avec la journalisation du serveur SSH et crée une instance de :SSHAuthentication avec ces propriétés, lorsqu'applicables : • :hasDestinationIPAddress (p. ex. « 96.20.12.34 ») • :hasSourceIPAddress (p. ex. « 10.7.11.8 ») • :hasUserName (p. ex. « Alice »)

Tableau B.3 – Pilotes liés aux configurateurs

ID	Description du pilote lié au configurateur
C001	Crée des instances de :Machine en fonction des machines locales. Elles ont ces propriétés : • :hasHostname (p. ex. « WINDOWS-PC ») • :hasPartition :Partition • :hasIPAddress (p. ex. « 192.168.3.1 ») • :offers :Service • :hasOperatingSystem :OperatingSystem Crée des instances de :Partition avec cette propriété : • :hasName (p. ex. « M: ») Crée des instances sous-classes de :Service avec ces propriétés : • :hasPort (p. ex. 80) • :hasModule :Module (p. ex. :WooCommerce_2.3) Le type des instances est en fonction du service: • :Website • :hasWebpage :Webpage • :hasDomainName (p. ex. « www.site.lan ») • :VPNService Pour un site Web, crée toutes les instances de :Webpage avec cette propriété : • :hasPath (« /login ») Pour une page Web exclusivement accessible par les administrateurs du site Web, elle fait partie de la classe :AdminWebpage
C002	Crée des instances de :File en fonction des fichiers distants. Elles ont ces propriétés : • :hasName (p. ex. « fichier.txt ») • :hasPath (p. ex. « M:\ »)
C003	Crée des instances de :User en fonction des utilisateurs Active Directory. Elles ont cette propriété : • :hasName (p. ex. « Alice »)
C004	Crée des instances de :Group en fonction des groupes Active Directory. Elles ont ces propriétés : • :hasName (p. ex. « dev »)

	• :hasMachineMemberName (p. ex. « BOB-PC ») • :hasUserMemberName (p. ex. « Bob ») • :hasGroupPolicy :GroupPolicy Crée des instances de :GroupPolicy en fonction des politiques de groupes Active Directory. Elles ont ces propriétés : • :hasName (p. ex. « dev-policy ») • :hasAdminRightsOnGroupName (p. ex. « dev ») • :hasAdminRightsOnName (p. ex. « cluster1.dev.lan »)
C005	Crée des instances de :Process et les ajoute à la liste blanche d'opérations d'écriture :WhitelistedProcesses. Elles ont la propriété suivante: • :hasName (p. ex. « calc.exe »)