



Représentation en logiques de description des alertes et des informations contextuelles

—
Délivrable n°01

Salem BENFERHAT
CRIL

Safa YAHI
CRIL



1 Introduction

Afin de pouvoir faire coopérer des systèmes de détection d'intrusion hétérogènes, l'utilisation d'un langage formel commun pour exprimer les informations nécessaires et raisonner la dessus est crucial.

Le langage choisi dans ce travail correspond aux logiques de description [1] qui sont une famille de langages de représentation de connaissances d'une manière formelle et structurée. En plus, elles ont de nombreuses applications notamment en Web sémantique. Le choix de telles logiques, et non pas le choix de la logique des prédicats du premier ordre (dont la plupart des DLs sont un fragment) par exemple, se justifie par les points suivants:

- La plupart des logiques de description sont décidables.
- Il s'agit d'un langage utilisé pour structurer des informations en utilisant la notion de concepts et de rôles.
- Actuellement, on dispose d'un nombre considérable de logiques de description qui varient en terme d'expressivité en utilisant certains constructeurs. De plus, des travaux ont consisté à déterminer la complexité correspondante et par conséquent, l'utilisateur est capable de choisir la logique qui répond à ses besoins en termes d'expressivité avec un moindre coût en termes de complexité.
- De nombreux raisonneurs DL ont été développés à l'instar de Fact ++ [4]. La plupart de ces derniers utilisent des techniques d'optimisation sophistiquées. Par conséquent, ces raisonneurs sont généralement efficaces en pratique notamment sur des problèmes réels.
- Enfin, de nombreux langages de description d'ontologies sont basés sur les DLs étant données les caractéristiques précédentes. A titre d'exemple, on peut citer DAML+OIL [5] et le fameux langage OWL^{1,2} [3].

2 Rappel sur les logiques de description

Une base de connaissances (KB pour Knowledge Représentation) basée sur la logique de description comprend deux composantes : la TBox (Terminologies) et la ABox (Assertions). La TBox introduit la terminologie, i.e, la définition du domaine d'une application, alors que la ABox contient des assertions quant à des individus nommés en termes de ce vocabulaire.

Le vocabulaire consiste en des concepts, qui dénotent des ensembles d'individus, et des rôles qui dénotent des relations binaires entre des individus. En plus des concepts atomiques et des rôles atomiques (les noms des concepts et des règles), tous les systèmes DL permettent de construire des descriptions de concepts et de rôles plus complexes. Le langage de construction de descriptions constitue une caractéristique du système en question.

¹<http://www.w3.org/2004/OWL/>

²<http://www.cs.manchester.ac.uk/horrocks/ISWC2003/Tutorial/>

2.1 Langages de description

Les descriptions élémentaires sont les concepts atomiques et les rôles atomiques à partir desquels des descriptions complexes peuvent être générées via les constructeurs de concepts et les constructeurs de rôles. Dans ce qui suit, on utilisera les lettres A et B pour désigner des concepts atomiques, la lettre R pour un rôle atomique et les lettres C et D pour les descriptions de concepts. Les langages de description sont distingués par rapport aux constructeurs qu'ils fournissent. Le langage \mathcal{AL} (pour *Attributive Language*) a été introduit dans [8] comme un langage minimal ayant un intérêt pratique. La forme la plus générale des axiomes d'une TBox sont les inclusions générales de concepts langages de la famille \mathcal{AL} en sont une extension.

2.1.1 Le langage de description basique \mathcal{AL}

Les descriptions de concepts dans le langage \mathcal{AL} sont générées selon la règle syntaxique suivante:

$$\begin{array}{ll}
 C, D \rightarrow & A \mid \quad (\text{concept atomique}) \\
 & \top \mid \quad (\text{concept universel}) \\
 & \perp \mid \quad (\text{concept bottom}) \\
 & \neg A \mid \quad (\text{négation atomique}) \\
 & C \sqcap D \mid \quad (\text{intersection}) \\
 & \forall R.C \mid \quad (\text{restriction de valeur}) \\
 & \exists R.\top \quad (\text{quantificateur existentiel limité})
 \end{array}$$

Il est à noter, que dans le langage \mathcal{AL} , la négation n'est appliquée qu'aux concepts atomiques et seulement le concept universel est permis à la portée du quantificateur existentiel. D'autre part, le sous langage de \mathcal{AL} obtenu en supprimant la négation atomique est appelé \mathcal{FL}^- alors que le sous langage de \mathcal{FL}^- obtenu en supprimant le quantificateur existentiel limité est appelé \mathcal{FL}_0 .

L'exemple suivant nous donne une idée sur ce qui peut être exprimé dans le langage \mathcal{AL} :

Exemple 1 *Soient **Person** et **Female** deux concepts atomiques. Donc $\text{Person} \sqcap \text{Female}$ et $\text{Person} \sqcap \neg \text{Female}$ sont des concepts décrivant intuitivement les personnes qui sont femelle et les personnes qui ne sont pas femelle. En outre, étant donné un rôle atomique **hasChild**, on peut construire les concepts $\text{Person} \sqcap \exists \text{hasChild}.\top$ et $\text{Person} \sqcap \forall \text{hasChild}.\text{Female}$ dénotant les personnes qui ont un enfant et les personnes dont tous les enfants sont une femelle. Enfin, en utilisant le concept bottom, on peut également décrire les personnes qui n'ont pas d'enfants par $\text{Person} \sqcap \forall \text{hasChild}.\perp$*

En vue de définir une sémantique formelle des concepts- \mathcal{AL} , on considère une interprétation \mathcal{I} définie par la donnée d'un ensemble non vide $\Delta^{\mathcal{I}}$ (le domaine d'interprétation) et d'une fonction d'interprétation, qui assigne à chaque concept atomique A un ensemble $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ et à chaque rôle atomique R une relation binaire $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Une fonction d'interprétation est étendue aux descriptions de concepts via la définition inductive suivante :

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\perp^{\mathcal{I}} &= \emptyset \\
(\neg A)^{\mathcal{I}} &= \Delta^{\mathcal{I}} - A^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall b, (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(\exists R.\top)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists b, (a,b) \in R^{\mathcal{I}}\}
\end{aligned}$$

Deux concepts C et D sont dits équivalents, $C \equiv D$, si et seulement si $C^{\mathcal{I}} = D^{\mathcal{I}}$ pour toute interprétation \mathcal{I} . Par exemple, on peut aisément vérifier que les concepts $\forall \text{hasChild.Female} \sqcap \forall \text{hasChild.Student}$ et $\forall \text{hasChild.}(\text{Female} \sqcap \text{Student})$ sont équivalents.

2.1.2 La famille des langages \mathcal{AL}

D'autres langages, plus expressifs, peuvent être définis en rajoutant d'autres constructeurs au langage \mathcal{AL} à savoir:

- L'union de concepts, désignée par la lettre \mathcal{U} , notée par $C \sqcup D$ et interprétée par:

$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}.$$

- La quantification existentielle complète, désignée par la lettre \mathcal{E} , notée par $\exists R.C$ et interprétée par:

$$(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \exists b, (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}.$$

- Les restrictions de nombres, désignées par la lettre \mathcal{N} et notées par $\geq nR$ (restriction au moins) et par $\leq nR$ (restriction au plus) où n représente un entier positif. Ces restrictions de valeurs sont interprétées respectivement comme suit:

$$(\geq nR)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} : |\{b \mid (a,b) \in R^{\mathcal{I}}\}| \geq n\},$$

et

$$(\leq nR)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} : |\{b \mid (a,b) \in R^{\mathcal{I}}\}| \leq n\},$$

où " $|\cdot|$ " dénote le cardinal d'un ensemble donné.

- La négation de concepts arbitraires, désignée par la lettre (C) , notée par $\neg C$ et interprétée de la façon suivante :

$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} - C^{\mathcal{I}}.$$

Exemple 2 Avec ces nouveaux constructeurs, on peut par exemple décrire les personnes ayant soit pas plus d'un enfant soit au moins trois enfants dont un est une femme comme suit:

$$\text{Person} \sqcap (\leq 1 \text{ hasChild} \sqcup (\geq 3 \text{ hasChild} \sqcap \exists \text{hasChild.Female})).$$

Étendre le langage \mathcal{AL} par n'importe quel sous ensemble des constructeurs précédents génère un langage \mathcal{AL} particulier désigné par une chaîne de caractères de la forme :

$$\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{N}][\mathcal{C}],$$

où une lettre dans l'appellation reflète la présence du constructeur correspondant. Par exemple, $\mathcal{AL}\mathcal{E}\mathcal{N}$ est l'extension de \mathcal{AL} par la quantification existentielle complète et les restrictions de nombres.

D'un point de vue sémantique, on a $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$ et $\exists R.C \equiv \neg \forall R.\neg C$. Par conséquent, l'union et la quantification universelle complète peuvent être exprimées via la négation et vice versa. Ainsi, on utilisera la lettre \mathcal{C} au lieu des lettres \mathcal{UE} dans les noms de langages. Par exemple, on écrira $\mathcal{AL}\mathcal{C}\mathcal{N}$ au lieu de $\mathcal{AL}\mathcal{UE}\mathcal{N}$.

2.2 Les terminologies TBox

On a vu comment former des descriptions de concepts complexes. Maintenant, on introduit les axiomes terminologiques. Après, on considère les définitions comme étant des axiomes spécifiques et on identifie les terminologies comme un ensemble de définitions par lequel on introduit des concepts atomiques comme des abréviations ou des noms pour des concepts complexes.

2.2.1 Axiomes terminologiques

En général, les axiomes terminologiques sont de la forme

$$C \sqsubseteq D (R \sqsubseteq S)$$

ou

$$C \equiv D (R \equiv S)$$

où C, D sont des concepts (et R, S son des rôles). Les axiomes du premier type sont appelés inclusions alors que ceux du second type sont dits égalités. Pour simplifier, nous traiterons par la suite uniquement les axiomes liés aux concepts.

Sémantiquement, une interprétation \mathcal{I} satisfait une inclusion $C \sqsubseteq D$ si et seulement si $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ et satisfait une égalité $C \equiv D$ si et seulement si $C^{\mathcal{I}} = D^{\mathcal{I}}$. Si Σ est un ensemble d'axiomes, alors \mathcal{I} satisfait Σ si et seulement si \mathcal{I} satisfait chaque élément de Σ . Si \mathcal{I} satisfait un axiome (resp. un ensemble d'axiomes), alors \mathcal{I} est dite modelé de cet axiome (resp. l'ensemble d'axiomes). Deux axiomes ou deux ensembles d'axiomes sont équivalents si et seulement s'ils ont les mêmes modelés.

2.2.2 Définitions

Une égalité dont le membre gauche est un concept atomique est une définition. Les définitions sont utilisées pour affecter des noms symboliques à des descriptions complexes. Par exemple, par l'axiome:

$$\text{Mother} \equiv \text{Woman} \sqcap \exists \text{hasChild}.\text{Person}$$

on associe à la description $\text{Woman} \sqcap \exists \text{hasChild}.\text{Person}$ le nom **Mother**. Les noms symboliques peuvent être utilisés comme des abréviations dans d'autres descriptions. Si, par exemple, on a défini le concept **Father** par analogie au concept **Mother**, on peut définir le concept **Parent** comme suit:

$$\text{Parent} \equiv \text{Mother} \sqcup \text{Father}.$$

On appelle un ensemble de définitions Σ une terminologie ou une TBox si aucun nom symbolique n'est défini plus d'une fois, i.e, pour chaque concept atomique A , il existe au plus un axiome dans Σ dont le membre gauche est A .

Exemple 3 *La terminologie suivante exprime des concepts liés à des relations familiales:*

$$\begin{aligned} \text{Woman} &\equiv \text{Person} \sqcap \text{Female} \\ \text{Man} &\equiv \text{Person} \sqcap \neg \text{Woman} \\ \text{Mother} &\equiv \text{Woman} \sqcap \exists \text{hasChild}.\text{Person} \\ \text{Father} &\equiv \text{Man} \sqcap \exists \text{hasChild}.\text{Person} \\ \text{Parent} &\equiv \text{Mother} \sqcup \text{Father} \\ \text{GrandMother} &\equiv \text{Mother} \sqcap \exists \text{hasChild}.\text{Parent} \\ \text{MotherWithoutDaughter} &\equiv \text{Mother} \sqcap \forall \text{hasChild}.\neg \text{Woman} \end{aligned}$$

2.2.3 Terminologies avec axiomes d'inclusion

Des fois, on se trouve incapables de définir certains concepts d'une manière complète. Dans ce cas, on peut établir des conditions nécessaires pour l'appartenance de concepts en utilisant l'inclusion. On appelle une inclusion où le membre gauche est un concept atomique une spécialisation. Par exemple

$$\text{Woman} \sqsubseteq \text{Person}.$$

Si on permet les spécialisations dans une terminologie, cette dernière perd son caractère définitorial même si elle est acyclique. Une telle terminologie est dite généralisée.

Une terminologie généralisée Σ peut être transformée en une terminologie régulière Σ' contenant uniquement des définitions. La terminologie Σ' est obtenue à partir de Σ en attribuant pour chaque spécialisation $A \sqsubseteq C$ un nouveau concept de base \overline{A} et en remplaçant la spécialisation $A \sqsubseteq C$ par la définition $A \equiv \overline{A} \sqcap C$. La terminologie Σ' est appelée la normalisation de Σ .

La normalisation de la terminologie généralisée

$$\text{Woman} \sqsubseteq \text{Person}$$

est la terminologie

$$\text{Woman} \equiv \overline{\text{Woman}} \sqcap \text{Person}.$$

Intuitivement, le nouveau symbole de base $\overline{\text{Woman}}$ correspond aux qualités qui différencient les femmes des autres personnes.

2.3 Descriptions du monde ABox

Outre la terminologie ou la TBox, le second composant d'une base de connaissance est la description du monde ou la ABox.

La ABox décrit un état spécifique du domaine d'application en termes de concepts et de rôles. En fait, dans une ABox, on introduit des individus (en leur donnant des noms) ainsi que leur propriétés. Les individus sont notés par a, b, c . Soit C un concept et R un rôle. On peut former des assertions comme suit: $C(a)$ et $R(b, c)$.

La première assertion est dite assertion de concept, elle signifie que a appartient à (l'interprétation de) C tandis que la deuxième, qui est appelée assertion de rôle, elle signifie que c est un remplisseur du rôle R pour b .

Par exemple, si PETER, PAUL et MARY sont des noms d'individus, alors **Father** (PETER) signifie que PETER est un père et **hasChild**(MARY, PAUL) signifie que PAUL est enfant de MARY. Une ABOX notée \mathcal{A} est un ensemble fini de telles assertions.

Une ABox peut être vue comme étant une instance d'une base de données relationnelle avec seulement des relations unaires et binaires. Toutefois, contrairement à la sémantique du monde clos des bases de données classiques, la sémantique des ABox est une sémantique monde ouvert étant donné que les systèmes de représentation de connaissances sont appliqués dans des situations où l'on peut supposer que l'information est incomplète.

La sémantique des ABox est définie par l'extension des interprétations aux noms d'individus. Donc une interprétation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ n'assigne pas uniquement des ensembles et des relations binaires aux concepts et aux rôles mais aussi assigne à chaque nom d'individu a un élément $a^{\mathcal{I}}$ de $\Delta^{\mathcal{I}}$. On suppose que des noms d'individus distincts dénotent des objets différents. Par conséquent, l'interprétation doit satisfaire l'assomption de nom unique UNA (pour unique name assumption), i.e, si a et b sont des noms différents alors $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

D'autre part, une interprétation \mathcal{I} satisfait l'assertion de concept $C(a)$ si $a^{\mathcal{I}} \in C^{\mathcal{I}}$. Elle satisfait l'assertion de rôle $R(a, b)$ si $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ et elle satisfait une ABox \mathcal{A} si elle satisfait chaque assertion dans \mathcal{A} . Dans ce cas, \mathcal{I} est dite modèle de l'assertion ou de la ABox. Enfin, une interprétation \mathcal{I} satisfait une assertion ou une ABox par rapport à une terminologie Σ si en plus d'être modèle de l'assertion ou de la base, elle est également modèle de la terminologie Σ .

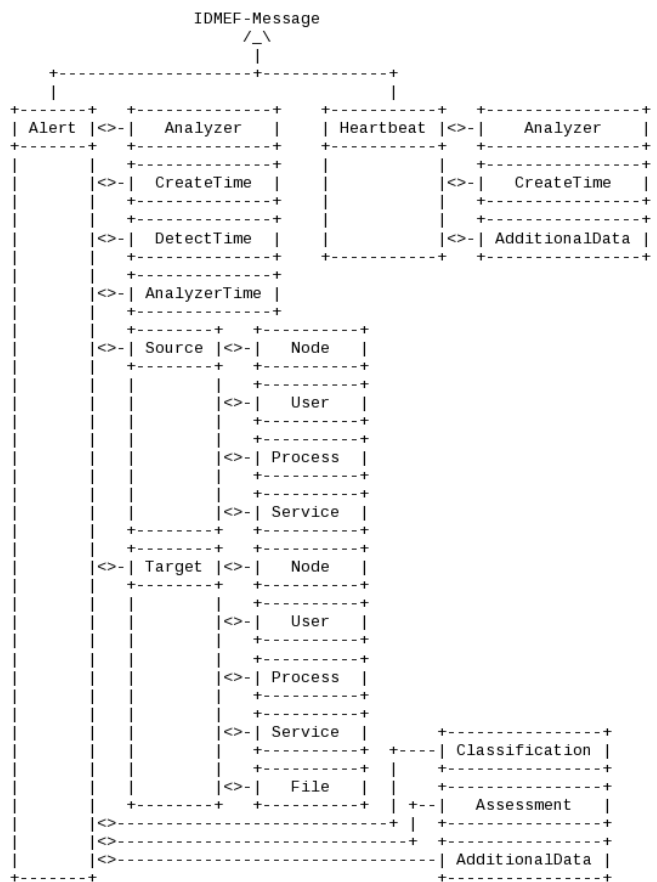
3 Rappel sur l'IDMEF

L'IDMEF (pour Intrusion Detection Message exchange Format) est un format d'alertes proposé pour le groupe IDWG (Intrusion Detection Working Group). Il s'agit d'un modèle orienté objet qui a été implémenté en XML.

Une description détaillée de ce format et sur laquelle on s'est basé est fournie par le RFC 4765 (<http://www.ietf.org/rfc/rfc4765.txt>).

Par exemple, une alerte en IDMEF admet les caractéristiques suivantes:

- Identifier : qui permet d'identifier l'alerte.
- CreateTime : le temps de la création de l'alerte
- DetectTime : le temps de la détection de l'attaque correspondante



- AnalyseTime
- Analyser : le système ayant généré l'alerte
- Source : la source de l'attaque
- Target : la cible de l'attaque
- Classification
- Assesement : peut refléter la sévérité de l'attaque.
- AdditionalData : ce champs peut contenir toute information additionnelle

Néanmoins, XML est limité à une représentation syntaxique. Donc, étant donné que cette représentation soit dénuée de sémantique, ce qui ne permet pas d'effectuer un raisonnement, il faut aller au delà de XML. En particulier, nous proposons de considérer le langage OWL qui est basé sur les logiques de description.

4 Contrepartie DL du la DTD du langage ID-MEF

Nous avons représenté le vocabulaire de l'IDMEF en logique de description. Pour ce faire, nous avons utilisé une TBox construite à partir de:

1. ≈ 20 concepts,
2. ≈ 60 rôles.

De plus, la TBox comprend

1. des axiomes de définitions,
2. des axiomes d'inclusions.

Commençons par le concept Alerte.

```
Alert  $\sqsubseteq$   $\forall$ messageId.String  $\sqcap = 1$  messageId  $\sqcap$ 
 $\forall$ hasCreateTime.Time  $\sqcap = 1$  hasCreateTime  $\sqcap$ 
 $\forall$ hasDetectTime.Time  $\sqcap \leq 1$  hasDetectTime  $\sqcap$ 
 $\forall$ hasAnalyserTime.Time  $\sqcap \leq 1$  hasAnalyserTime  $\sqcap$ 
 $\forall$ hasAnalyser.Analyser  $\sqcap = 1$  hasAnalyser  $\sqcap$ 
 $\forall$ hasSource.Source  $\sqcap$ 
 $\forall$ hasTarget.Target  $\sqcap$ 
 $\forall$ hasClassification.Classification  $\sqcap = 1$  hasClassification  $\sqcap$ 
 $\forall$ hasAssesement.Assessment  $\sqcap \leq 1$  hasAssessment  $\sqcap$ 
 $\forall$ hasAdditionalData.AdditionalData
```

Un tel axiome signifie le fait qu'une alerte admet un seul identificateur qui est de type String, un seul detecttime de type time, un seul createtime de type time, et au plus un seul analysertime de type time aussi. De plus, pour une alerte peut correspondre une source, voire même plusieurs. Il est de même, il

peut lui correspondre une cible, voire même plusieurs. En outre, une alerte admet une seule classification, au plus un élément assement qui peut décrire par exemple son degré de gravité et un champ additional data qui quant à lui, peut comprendre tout type d'information qui est en dehors des informations précédentes.

Pour le reste des concepts, nous avons proposé ces axiomes.

```

Analyser  ⊆  ∀analyzerId.String ⊎ = 1 analyzerId ⊎
           ∀manufacturer.String ⊎ = 1 manufacturer ⊎
           ∀model.String ⊎ = 1 model ⊎
           ∀name.String ⊎ = 1 name ⊎
           ∀version.String ⊎ = 1 version ⊎
           ∀class.String ⊎ = 1 class ⊎
           ∀ostype.String ⊎ = 1 ostype ⊎
           ∀osversion.String ⊎ = 1 osversion ⊎
           ∀hasNode.Node ⊎ ≤ 1 hasNode ⊎
           ∀hasProcess.Process ⊎ ≤ 1 hasProcess ⊎
           ∀hasAnalyser.Analyser ⊎ ≤ 1 hasAnalyser

Source    ⊆  ∀sourceId.String ⊎ ≤ 1 sourceId ⊎
           ∀spoofed.{yes,no,unknown} ⊎ ≤ 1 spoofed ⊎
           ∀interface.String ⊎ ≤ 1 interface ⊎
           ∀hasNode.Node ⊎ ≤ 1 hasNode ⊎
           ∀hasProcess.Process ⊎ ≤ 1 hasProcess ⊎
           ∀hasUser.User ⊎ ≤ 1 hasUser ⊎
           ∀hasService.Service ⊎ ≤ 1 hasService

Target    ⊆  ∀targetId.String ⊎ ≤ 1 targetId ⊎
           ∀decoy.{yes,no,unknown} ⊎ ≤ 1 decoy ⊎
           ∀interface.String ⊎ ≤ 1 interface ⊎
           ∀hasNode.Node ⊎ ≤ 1 hasNode ⊎
           ∀hasProcess.Process ⊎ ≤ 1 hasProcess ⊎
           ∀hasUser.User ⊎ ≤ 1 hasUser ⊎
           ∀hasService.Service ⊎ ≤ 1 hasService ⊎
           ∀hasFileList.File

Classification ⊆  ∀ident.String ⊎ ≤ 1 ident ⊎
                 ∀text.String ⊎ = 1 text ⊎
                 ∀hasReference.Reference

```

Assessment \sqsubseteq \forall hasImpact.Impact $\sqcap \leq 1$ hasImpact \sqcap
 \forall hasAction.Action \sqcap
 \forall hasConfidence.Confidence $\sqcap \leq 1$ hasConfidence

AdditionalData \sqsubseteq \forall type.String $\sqcap = 1$ type \sqcap
 \forall meaning.String $\sqcap \leq 1$ meaning

Node \sqsubseteq \forall ident.String $\sqcap \leq 1$ ident \sqcap
 \forall category.enumcat $\sqcap \leq 1$ category \sqcap
 \forall location.String $\sqcap \leq 1$ location \sqcap
 \forall name.String $\sqcap \leq 1$ name \sqcap
 \forall hasAddress.Address $\sqcap \forall$ name.hasAddress

Process \sqsubseteq \forall ident.String $\sqcap \leq 1$ ident \sqcap
 \forall name.String $\sqcap = 1$ name \sqcap
 \forall pid.Integer $\sqcap \leq 1$ pid \sqcap
 \forall path.String $\sqcap \leq 1$ path \sqcap
 \forall argv.String \sqcap
 \forall env.String

User \sqsubseteq \forall ident.String $\sqcap \leq 1$ ident \sqcap
 \forall category.{unknown, application, os-device} $\sqcap \leq 1$ category \sqcap
 \forall hasUserId.UserId $\sqcap \geq 1$ hasUserId

Service \sqsubseteq \forall ident.String $\sqcap \leq 1$ ident \sqcap
 \forall ip_version.Integer $\sqcap \leq 1$ ip_version \sqcap
 \forall iana_protocol_number.Integer $\sqcap \leq 1$ iana_protocol_number \sqcap
 \forall iana_protocol_name.String $\sqcap \leq 1$ iana_protocol_name \sqcap
 \forall name.String $\sqcap \leq 1$ name \sqcap
 \forall port.String $\sqcap \leq 1$ port \sqcap
 \forall portlist.PORTLIST $\sqcap \leq 1$ portlist \sqcap
 \forall protocol.String $\sqcap \leq 1$ protocol

WebService \sqsubseteq Service \sqcap
 \forall url.String $\sqcap = 1$ url \sqcap
 \forall cgi.String $\sqcap \leq 1$ cgi \sqcap
 \forall http-method.String $\sqcap \leq 1$ http-method \sqcap
 \forall arg.String

SNMPService \sqsubseteq Service \sqcap
 \forall oid.String $\sqcap \leq 1$ oid \sqcap
 \forall messageProcessingModel.Integer $\sqcap \leq 1$ messageProcessingModel \sqcap
 \forall securityModel.Integer $\sqcap \leq 1$ securityModel \sqcap
 \forall securityName.String $\sqcap \leq 1$ securityName \sqcap
 \forall securityLevel.Integer $\sqcap \leq 1$ securityLevel \sqcap
 \forall contextName.String $\sqcap \leq 1$ contextName \sqcap
 \forall contextEngineID.String $\sqcap \leq 1$ contextEngineID \sqcap
 \forall command.String $\sqcap \leq 1$ command

File \sqsubseteq $\forall \text{ident.String } \sqcap \leq 1 \text{ ident } \sqcap$
 $\forall \text{category.}\{\text{current,original}\} \sqcap = 1 \text{ category } \sqcap$
 $\forall \text{fstype.}\{\text{ufs,efs,nfs,afs,ntfs,fat16,fat32,pcfs,joliet,iso9660}\}$
 $\sqcap \leq 1 \text{ fstype } \sqcap$
 $\forall \text{file-type.String } \sqcap \leq \text{file-type } \sqcap$
 $\forall \text{name.String } \sqcap = 1 \text{ name } \sqcap$
 $\forall \text{path.String } \sqcap = 1 \text{ path } \sqcap$
 $\forall \text{create-time.DATETIME } \sqcap \leq 1 \text{ create-time } \sqcap$
 $\forall \text{modify-time.DATETIME } \sqcap \leq 1 \text{ modify-time } \sqcap$
 $\forall \text{access-time.DATETIME } \sqcap \leq 1 \text{ access-time } \sqcap$
 $\forall \text{date-size.Integer } \sqcap \leq 1 \text{ date-size } \sqcap$
 $\forall \text{disk-size.Integer } \sqcap \leq 1 \text{ disk-size } \sqcap$
 $\forall \text{hasFileAccess.FileAccess } \sqcap$
 $\forall \text{hasLinkage.Linkage } \sqcap$
 $\forall \text{hasInode.Inode } \sqcap \leq 1 \text{ inode } \sqcap$
 $\forall \text{hasChecksum.Checksum}$

Address \sqsubseteq $\forall \text{ident.String } \sqcap \leq 1 \text{ ident } \sqcap$
 $\forall \text{category.enumcat } \sqcap \leq 1 \text{ category } \sqcap$
 $\forall \text{vlan-name.String } \sqcap \leq 1 \text{ vlan-name } \sqcap$
 $\forall \text{vlan-num.Integer } \sqcap \leq 1 \text{ vlan-num } \sqcap$
 $\forall \text{address.String } \sqcap = 1 \text{ address } \sqcap$
 $\forall \text{netmask.String } \sqcap \leq 1 \text{ netmask}$

Impact \sqsubseteq $\forall \text{severity.}\{\text{info,low,medium,high}\} \sqcap \leq 1 \text{ severity } \sqcap$
 $\forall \text{completion.}\{\text{failed,succeeded}\} \sqcap \leq 1 \text{ completion } \sqcap$
 $\forall \text{type.}\{\text{admin,dos,file,recon,user,other}\} \sqcap \leq 1 \text{ type}$

Action \sqsubseteq $\forall \text{category.}\{\text{block-installed,notification-sent,taken-offine,other}\} \sqcap$
 $= 1 \text{ category}$

Confidence \sqsubseteq $\forall \text{rank.}\{\text{low, medium, high, numeric}\} \sqcap$
 $= 1 \text{ rank}$

Reference \sqsubseteq $\forall \text{name.String } \sqcap = 1 \text{ name } \sqcap$
 $\forall \text{url.String } \sqcap = 1 \text{ url } \sqcap$
 $\forall \text{origin.String } \sqcap = 1 \text{ origin } \sqcap$
 $\forall \text{meaning.String } \sqcap \leq 1 \text{ meaning}$

UserId \sqsubseteq $\forall \text{ident.String } \sqcap \leq 1 \text{ ident } \sqcap$
 $\forall \text{type.Enum } \sqcap \leq 1 \text{ type } \sqcap$
 $\forall \text{tty.String } \sqcap \leq 1 \text{ tty } \sqcap$
 $\forall \text{name.String } \sqcap \geq 1 \text{ tty } \sqcap$
 $\forall \text{number.Integer } \sqcap \geq 1 \text{ number}$

5 Illustration à travers des exemples

5.1 Exemple 1

```
<?xml version="1.0" encoding="UTF-8"?>
<idmef:IDMEF-Message xmlns:idmef="http://iana.org/idmef"
  version="1.0">
  <idmef:Alert messageid="abc123456789">
    <idmef:Analyzer analyzerid="hq-dmz-analyzer01">
      <idmef:Node category="dns">
        <idmef:location>Headquarters DMZ Network</idmef:location>
        <idmef:name>analyzer01.example.com</idmef:name>
      </idmef:Node>
    </idmef:Analyzer>
    <idmef:CreateTime ntpstamp="0xbc723b45.0xef449129">
      2000-03-09T10:01:25.93464-05:00
    </idmef:CreateTime>
    <idmef:Source ident="a1b2c3d4">
      <idmef:Node ident="a1b2c3d4-001" category="dns">
        <idmef:name>badguy.example.net</idmef:name>
        <idmef:Address ident="a1b2c3d4-002"
          category="ipv4-net-mask">
          <idmef:address>192.0.2.50</idmef:address>
          <idmef:netmask>255.255.255.255</idmef:netmask>
        </idmef:Address>
      </idmef:Node>
    </idmef:Source>
    <idmef:Target ident="d1c2b3a4">
      <idmef:Node ident="d1c2b3a4-001" category="dns">
        <idmef:Address category="ipv4-addr-hex">
          <idmef:address>0xde796f70</idmef:address>
        </idmef:Address>
      </idmef:Node>
    </idmef:Target>
    <idmef:Classification text="Teardrop detected">
      <idmef:Reference origin="bugtraqid">
        <idmef:name>124</idmef:name>
        <idmef:url>http://www.securityfocus.com/bid/124</idmef:url>
      </idmef:Reference>
    </idmef:Classification>
  </idmef:Alert>
</idmef:IDMEF-Message>
```

Figure 1: Exemple d'alerte

Considérons la figure 1 qui décrit une description IDMEF d'une alerte dans le format XML. Cet exemple est tiré du RFC de l'IDMEF. Le but est de traduire ce format en DL en utilisant les concepts et rôles définis dans la section précédente.

1. Différents concepts

- Alert(ALR1)
- Analyser(ANL1)
- Source(SRC1)
- Target(TRG1)
- Classification(CLS1)
- Node(NOD1)
- Node(NOD2)
- Node(NOD3)

- Reference(RFC1)
- Address(ADR1)
- Address(ADR2)

2. L'alerte

- messageId(ALR1, "abc123456789")
- hasCreateTime(ALR1, 2000-03-09T10:01:25.93464-05:00)
- hasAnalyzer(ALR1, ANL1)
- hasSource(ALR1, SRC1)
- hasTarget(ALR1, TRG1)
- hasClassification(ALR1, CLS1)

3. L'analyser

- analyzerId(ANL1, "hq-dmz-analyzer01")
- hasNode(ANL1, NOD1)

4. Le noeud N1

- category(NOD1, "dns")
- location(NOD1, "Headquarters DMZ Network")
- name(NOD1, analyzer01.example.com)

5. La source

- hasNode(SRC1, NOD2)
- hasIdent(SRC1, a1b2c3d4-002)

6. Le noeud N2

- hasIdent(NOD2, "a1b2c3d4-001")
- category(NOD2, "dns")
- name(NOD1, badguy.example.net)
- hasAddress(NOD2, ADR1)

7. L'adresse ADR1

- hasIdent(ADR1, "a1b2c3d4-002")
- category(ADR1, "ipv4-net-mask")
- address(ADR1, 192.0.2.50)
- netmask(ADR1, 255.255.255.255)

8. Le noeud N3

- hasIdent(NOD3, "d1c2b3a4")
- category(NOD3, "dns")
- hasAddress(NOD3, ADR2)

9. La cible

- `hasIdent(TRG1, "d1c2b3a4")`
- `hasNode(TRG1, NOD3)`

10. L'adresse ADR2

- `category(ADR2, "ipv4-addr-hex")`
- `address(ADR2, 0xde796f70)`

11. La classif

- `text(CLS1, "Teardrop detected")`
- `hasReference(CLS1, RFC1)`

12. La reference

- `origin(RFC1, "bugtraqid")`
- `name(RFC1, 124)`
- `url(RFC, http://www.securityfocus.com/bid/124)`

5.2 Exemple 2

Maintenant, considérons l'exemple suivant qui est pris d'une base d'alerte fournie par le projet PLACID.

1. `Address(ADR1)`
2. `category(ADR1, "ipv4-addr")`
3. `address(ADR1, 202.77.162.213)` •
4. `Address(ADR2)`
5. `category(ADR2, "ipv4-addr")`
6. `address(ADR1, 172.16.115.1)` •
7. `Node(NOD1)`
8. `name(NOD1, "localhost.localdomain")` •
9. `Node(NOD2)`
10. `address(NOD2, ADR1)` •
11. `Node(NOD3)`
12. `address(NOD2, ADR2)` •

```

<Alert ident="6135">
  <Analyzer analyzerid="snort_dmz" model="snort" version="2.3.2" ostype="Linux" osversion="2.6.9-5.EL">
    <Node>
      <name>localhost.localdomain</name>
    </Node>
  </Analyzer>
  <CreateTime ntpstamp="0xc6a2f0c5.0xcde68a0d">2005-08-09T09:07:17Z</CreateTime>
  <Source>
    <Node>
      <Address category="ipv4-addr">
        <address>202.77.162.213</address>
      </Address>
    </Node>
    <Service>
      <name>icmp</name>
      <protocol>icmp</protocol>
    </Service>
  </Source>
  <Target>
    <Node>
      <Address category="ipv4-addr">
        <address>172.16.115.1</address>
      </Address>
    </Node>
  </Target>
  <Classification origin="vendor-specific">
    <name>msg=ICMP PING</name>
    <url>none</url>
  </Classification>
  <Classification origin="vendor-specific">
    <name>sid=384</name>
    <url>http://www.snort.org/snort-db/sid.html?sid=384</url>
  </Classification>
  <Classification origin="vendor-specific">
    <name>class=misc-activity</name>
    <url>none</url>
  </Classification>
  <Classification origin="vendor-specific">
    <name>priority=3</name>
    <url>none</url>
  </Classification>
  <Assessment>
    <Impact severity="high"/>
  </Assessment>
  <AdditionalData meaning="sig_rev" type="string">5</AdditionalData>
</Alert>

```

Figure 2: Exemple d'alerte

13. Service(RV1)
14. name(SRV1, "icmp")
15. protocol(SRV1, "icmp")●
16. Impact(IMP1)
17. severity(IMP1,"high") ●
18. analyzerId(ANL1, "snort_dmz")
19. model(ANL1, "snort")
20. version(ANL1, "2.3.2")
21. ostype(ANL1, "Linux")

- 22. `osversion(ANL1, "2.6.9-5.EL")`
- 23. `hasNode(ANL1, NOD1)•`
- 24. `Source(SRC1)`
- 25. `hasNode(SRC1, N2)`
- 26. `hasService(SRC1, SRV)•`
- 27. `Target(TRG1)`
- 28. `hasNode(TRG1, N3)•`
- 29. `Assessment(ASS1)`
- 30. `hasImpact(IMP1) •`
- 31. `Alert(AL1)`
- 32. `messageId(ALR1, "6135")`
- 33. `hasCreateTime(ALR1, 2005-08-09T09:07:17Z)`
- 34. `hasAnalyzer(ALR1, ANL1)`
- 35. `hasSource(ALR1, SRC1)`
- 36. `hasTarget(ALR1, TRG1)`
- 37. `hasAssesement(ALR1, ASS1)`

6 Raisonner à partir de la traduction IDMEF en DL

La traduction DL de l'IDMEF que nous avons proposée peut être utilisée en vue d'effectuer du Clustering comme nous allons le voir tout au long de cette section.

Brièvement, le clustering revient à regrouper des alertes similaires. Parmi les méthodes de clustering, on peut citer

- les règles de Valdes et Skinner,
- les règles de Cuppens.

Les règles de Cuppens sont basées sur la comparaison d'attributs comme

- la classification,
- la source,
- la cible.

Considérons par exemple la similarité basée-classification qui décrit en général le type d'attaque en question. Étant donné que la classification peut différer d'un IDS à un autre, on aura besoin de règles de correspondance. de la forme:

$$R(\textit{standard_name}, \textit{IDS_name}, \textit{given_name}) \text{ où}$$

- *standard_name*: nom standard d'une attaque,
- *IDS_name*: un nom d'IDS,
- *given_name*: un nom donné par l'IDS *IDS_name* à l'attaque standard définie par *standard_name*.

Cette règle de classification stipule qu'une attaque A_X détectée par un IDS IDS_A est similaire à une A_Y détectée par un IDS IDS_B s'il existe un nom standard d'attaque A_S commun qui est détecté par l'IDS A sous le nom de A_X et par l'IDS B sous le nom A_Y .

Les logiques de description considèrent uniquement des relations unaires et binaires, la relation de correspondance étant ternaire, ceci implique qu'il va falloir la traduire en une relation binaire en utilisant le principe de réification qui pour réduire une relation R à n argument en une relation binaire consiste à:

- introduire un nouveau concept R' ,
- introduire n nouvelles relations binaires entre le nouveau concept et les n arguments de R .

Ceci revient à établir une correspondance un à un entre les individus du concept R' et les tuples de la relation R .

Soit le concept Correspondance défini ci-dessous le nouveau concept introduit par le principe de réification.

$$\begin{aligned} \text{Correspondance} \sqsubseteq & \exists \text{hasStandardName} . \top \sqcap \leq 1 \text{ hasStandardName} \sqcap \\ & \exists \text{hasIDSName} . \top \sqcap \leq 1 \text{ hasIDSName} \sqcap \\ & \exists \text{hasGivenName} . \top \sqcap \leq 1 \text{ hasGivenName}. \end{aligned}$$

Exemple 4 *Supposons que l'ont ait le tuple $R(SN_1, IDS_1, GN_1)$.*

Après réification, on doit introduire les assertions suivantes:

$$\begin{aligned} & \text{Correspondance}(R_a) \\ & \text{hasStandarName}(R_a, SN_1) \\ & \text{hasIDSName}(R_a, IDS_1) \\ & \text{hasGivenName}(R_a, GN_1) \end{aligned}$$

Maintenant, étant donnée une alerte Ax , déterminer les alertes qui lui sont similaires par rapport à la classification peut être obtenu en répondant à une requête conjonctive.

Rappelez qu'une **requête conjonctive** est une conjonction d'expressions de concepts de la forme $C(t)$ et d'expressions de rôle de la forme $r(t, t')$ où C est un concept, r un rôle et t et t' des termes c'est à dire des variables ou des noms d'individus. Par exemple

$$(x, y, z) \leftarrow \text{Father}(x) \wedge \text{Mother}(y) \wedge \text{hadChild}(x, z) \wedge \text{hasChild}(y, z).$$

permet de récupérer les couples père, mère et leurs fils.

Revenons objectif à savoir les alertes qui sont similaires à une larte A_X par rapport à la classification. Ceci peut être effectué via la requête conjonctive suivante:

$$\begin{aligned} A_y \leftarrow & \text{hasAnalyser}(A_x, I_a) \wedge \text{hasClassification}(A_x, N_a) \wedge \\ & \text{hasSName}(R, S) \wedge \text{hasIDS}(R, I_a) \wedge \text{hasGName}(R, N_a) \wedge \\ & \text{hasAnalyser}(A_y, I_b) \wedge \text{hasClassification}(A_y, N_b) \wedge \text{hasSName}(R', S) \wedge \\ & \text{hasIDS}(R', I_b) \wedge \text{hasGName}(R', N_b) \end{aligned}$$

7 Modèle M4D4

M4D4 est un modèle de données proposé dans le cadre de la détection d'intrusion afin de permettre à plusieurs analyseurs (IDS, Scanners de vulnérabilités, scanners réseaux, etc) de coopérer entre eux afin d'améliorer la détection d'intrusion. Le but de ce travail consiste à traduire ce modèle dans une logique de description combinée avec un ensemble de règles dites safe. La représentation proposée offre plusieurs avantages, en particulier le raisonnement s'effectue d'une manière décidable.

La logique de description ³ $\mathcal{SOIN}(D)$ est un fragment décidable de la logique du premier ordre. De ce fait, elle ne permet pas d'exprimer des axiomes arbitraires. En fait, les seuls axiomes qu'elle peut exprimer sont d'une certaine structure d'arbres. En revanche, les formalismes décidables basés-règles tels que les règles de Horn à fonctions libres ne partagent pas cette restriction mais manquent de la force expressive de $\mathcal{SHOIN}(D)$: elles sont restreintes à la quantification universelle et manquent de la négation.

Afin de pallier les limitations des ces deux approches, $\mathcal{SHOIN}(D)$ a été étendu avec le règles dans [6]. Cependant, il s'agit d'une extension indécidable.

Une combinaison décidable a été proposée dans [7] où la décidabilité est obtenue en restreignant les règles à des règles *DL-safe*. En fait, la restriction ne consiste pas à restreindre les composants des langages mais plutôt à restreindre l'interface entre eux.

Soient N_C un ensemble de noms de concept et N_R un ensemble de noms de rôles. Soit N_P un ensemble de prédicats tel que $N_C \cup N_R \subseteq N_P$. Un terme est soit une constante soit une variable. Un atome est un prédicat appliqué à des termes, c'est-à-dire de la forme $P(t_1, \dots, t_n)$, où P est un symbole de prédicat et les t_i 's sont des termes. Une règle est de la forme

$$H \leftarrow B_1, \dots, B_n$$

où H et B_i sont des atomes. H est appelé la tête de la règle alors que l'ensemble des de tous les B_i 's est appelé le corps de la règle. Un programme est un ensemble fini de règles.

Un atome-DL est un atome de la forme $A(s)$ où $A \in N_C$ ou de la forme $R(s, t)$ où $R \in N_R$.

Une règle est dite DL-safe si chaque variable de la règle apparaît dans son corps dans un atome qui n'est pas atome-DL.

³Pour une description plus complète voir le livrable intitulé "État de l'art dur les logiques de description".

Par exemple, si *Person*, *livesAt*, et *worksAt* sont des rôles et des concepts (donc atomes-DL), la règle suivante n'est pas DL-safe:

$$\text{Homeworker}(x) \leftarrow \text{Person}(x), \text{lives}(x, y), \text{works}(x, y)$$

La raison est que les deux variables x et y apparaissent dans des atomes-DL, mais n'apparaissent pas dans le corps de la règle dans un atome qui n'est pas DL.

Cette règle peut être transformée en une règle DL-safe en rajoutant des atomes non DL spéciaux $O(x)$ et $O(y)$ au corps de la règle, et en rajoutant un fait $O(a)$ pour chaque individu a comme suit:

$$\text{Homeworker}(x) \leftarrow \text{Person}(x), \text{lives}(x, y), \text{works}(x, y) \wedge O(x) \wedge O(y)$$

8 Formuler le modèle M4D4 en DLs et règles DL-safe

M4D4 est un modèle qui regroupe les concepts et les relations nécessaires afin de corréler des alertes. Les informations modélisées dans M4D4 peuvent être classées en 4 catégories:

- Information contextuelle (topologie et cartographie).
- Attaques et vulnérabilités.
- Analyseur (IDS, scanner de vulnérabilité et firewall).
- Évènements et alertes.

8.1 Informations contextuelles

8.1.1 Topologie

Décrire la topologie nous permet de déduire si un IDS est capable ou non de détecter une alerte. La topologie concerne les nœuds ainsi que leurs interconnexions.

On considère que le réseau surveillé est décomposé en sous réseaux admettant chacun une adresse.

$$\text{Network} \sqsubseteq \forall \text{netaddress}.\text{String} \sqcap = 1 \text{ netaddress}$$

Les nœuds représentent n'importe quelle machine connectée au réseau. Un nœud admet une adresse et appartient à un réseaux.

$$\text{Node} \sqsubseteq \forall \text{nodeaddress}.\text{String} \sqcap = 1 \text{ nodeaddress} \sqcap \\ \forall \text{hasNodeNet}.\text{Network}$$

L'appartenance d'un nœud à un réseaux peut être spécifiée comme un fait donné explicitement ou bien déduite à travers la règle:

$$\text{hasNodeNet}(H, N) \leftarrow \text{nodeaddress}(H, A_H) \wedge \text{netaddress}(N, A_N) \wedge \text{matches}(A_H, A_N)$$

Cette règle peut être transformée en une règles DL-safe comme suit:

$$hasNodeNet(H, N) \leftarrow nodeaddress(H, A_H) \wedge netaddress(N, A_N) \wedge matches(A_H, A_N) \wedge O(H) \wedge O(N)$$

En fait, le seul prédicat non-DL est *matches*. Par conséquent, les variables qui n'apparaissent pas dans un prédicat non DL sont *H* et *N* d'où le rajout de *O(H)* et *O(N)*.

Il est à noter que cette règles peut être exprimée via l'inclusion complexe de rôles (disponible par exemple dans les logiques de description \mathcal{RIQ} et \mathcal{SROIQ}) comme suit:

$$nodeaddress \circ matches \circ netaddress^- \sqsubseteq hasNodeNet$$

Les passerelles sont des nœuds particuliers dont l'objectif est de connecter des réseaux ensemble. Clairement, une passerelle appartient à plus d'un réseau.

$$\begin{aligned} Gateway &\sqsubseteq Node \sqcap > 1 hasNodeNet \sqcap hasNodeNet.\{N_{EXT}\} \\ Node \sqcap \neg Gateway &\sqsubseteq = 1 hasNodeNet \end{aligned}$$

Les passerelles directement accessibles par un nœud sont désignées par:

$$\begin{aligned} Node &\sqsubseteq \forall hasNodeGateway.Gateway \sqcap \\ &\quad \forall hasNodeSystemname.String \sqcap = 1 hasNodeSystemname \end{aligned}$$

De plus,

$$hasNodeGateway(H, H_G) \leftarrow hasNodeNet(H, N) \wedge hasNodeNet(H_G, N) \wedge Gateway(H_G)$$

Pour rendre cette règles DL-safe, on rajoute au corps de la règle:

$$O(H) \wedge O(N) \wedge O(H_G)$$

Ensuite, étant donné un nœud source H_S et un nœud destination H_D , obtenir les chemins possibles entre H_S et H_D consiste d'abord à définir la passerelle H_{G_S} la plus proche de H_S , et la passerelle H_{G_D} la plus proche de H_D , ensuite évoquer le prédicat *path*. Ceci donne cette règle:

$$\begin{aligned} route(H_S, H_D, L) &\leftarrow hasNodeGateway(H_S, H_{G_S}) \wedge \\ hasNodeGateway(H_D, H_{G_D}) &\wedge hasNodeNet(H_D, N) \wedge path(H_{G_S}, H_{G_D}, L, N) \end{aligned}$$

Enfin, un nœud peut être en plus caractérisé par son nom système:

$$Node \sqsubseteq \forall hasNodeSystemName.String \sqcap = 1 hasNodeSystemName$$

8.2 Cartographie

La cartographie dénote les relations entre nœuds et logiciels. Ceci est utile car les vulnérabilités affectent les logiciels.

Un produit est caractérisé par un seul nom, une seule version, un seul type et par une seule architecture:

$$\begin{aligned} \text{Software} \sqsubseteq & \forall \text{softwareName.String} \sqcap = 1 \text{ softwareName} \sqcap \\ & \forall \text{softwareVersion.String} \sqcap = 1 \text{ softwareVersion} \sqcap \\ & \forall \text{softwareType.String} \sqcap = 1 \text{ softwareType} \sqcap \\ & \forall \text{softwareArchitecture.String} \sqcap = 1 \text{ softwareArchitecture} \sqcap \end{aligned}$$

Un nœud héberge un logiciel:

$$\text{Node} \sqsubseteq \forall \text{hosts.Software}$$

Un processus est un produit étant exécuté par un utilisateur.

$$\begin{aligned} \text{Process} \sqsubseteq & \forall \text{hasSoftware.Software} \sqcap = 1 \text{ hasProduct} \sqcap \\ & \forall \text{hasUser.User} \sqcap = 1 \text{ hasUser} \end{aligned}$$

La règle suivante exprime le fait qu'un nœud exécutant un produit implique que ce même nœud l'héberge:

$$\text{host}(H, S) \leftarrow \text{exec}(H, P) \wedge \text{hasProduct}(P, S)$$

Cette règle peut être facilement exprimée via l'inclusion complexe de rôles.

Un service est un processus qui écoute sur un port:

$$\begin{aligned} \text{Service} \sqsubseteq & \forall \text{hasProcess.Process} \sqcap = 1 \text{ hasProcess} \sqcap \\ & \forall \text{port.Integer} \sqcap = 1 \text{ port} \end{aligned}$$

De plus, un service qui écoute sur un nœud est exécuté par ce même nœud.

$$\text{exec}(H, P) \leftarrow \text{listen}(H, S) \wedge \text{hasProcess}(S, P)$$

8.3 Vulnérabilités

En général, une vulnérabilité n'affecte pas un seul produit mais plutôt une combinaison de produits. Un tel ensemble est appelé configuration vulnérable (ou bien configuration tout court).

$$\text{Vulnerability} \sqsubseteq \forall \text{affects.Configuration} \sqcap > 0 \text{ affects}$$

L'appartenance d'un produit à une configuration est modélisée par:

$$\text{Software} \sqsubseteq \forall \text{takePartIn.Configuration}$$

De plus, une vulnérabilité est caractérisée par son degré de sévérité, le niveau d'accès requis afin de l'exploiter, ses conséquences et sa date de publication:

$$\begin{aligned} \text{Vulnerability} \sqsubseteq & \forall \text{severity.}\{high, medium, low\} \\ & \forall \text{requires.}\{remote, local, user\} \\ & \forall \text{losstype.}\{confidentiality, integrity, availability, privilege_escalation\} \\ & \forall \text{published.Date} \end{aligned}$$

8.4 Attaques

Une attaque exploite une vulnérabilité.

$$\text{Attack} \sqsubseteq \text{exploits.Vulnerability}$$

Les attaques peuvent être liées entre elles via des relations d'héritage. Identifier de telles relation nécessite une expertise humaine.

8.5 Analyseurs

On distingue trois types d'analyseurs: IDS, scanners de vulnérabilité et firewalls.

8.5.1 IDS

$$\text{IDS} \sqsubseteq \text{Analyseur}$$

Parmi les IDS, on distingue les host-based IDS, les application-based IDS e les networks-based IDS.

Les HIDS surveillent un nœud:

$$\text{HIDS} \sqsubseteq \text{IDS} \sqcap \forall \text{monitors.Node} \sqcap = 1\text{monitors}$$

Quant aux AIDS, ils surveillent un produit donné qui s'exécute sur un nœud:

$$\text{AIDS} \sqsubseteq \text{IDS} \sqcap \forall \text{monitors.Node} \sqcap = 1\text{monitors} \sqcap \forall \text{hasApplication.Software} \sqcap = 1\text{hasApplication}$$

Un IPS est un NIDS qui surveille une passerelle:

$$\text{IPS} \sqsubseteq \text{NIDS} \sqcap \forall \text{monitors.Gateway} \sqcap = 1\text{monitors}$$

Un IPS est capable d'analyser un paquet intrusif dont la source est H_S et la destination est H_D si la chemin suivi par le paquet inclut la passerelle surveillée par cet IPS:

$$\text{can_detect}(A, H_S, H_D) \leftarrow \text{IPS}(A) \wedge \text{route}(H_S, H_D, L) \wedge \text{monitors}(A, H_G) \wedge H_G \in H$$

Cette règle peut être transformée en une règle DL-safe en rajoutant $O(A) \wedge 0(H_G)$ car les variables H_S , H_D et L apparaissent déjà dans un prédicat non DL à savoir *route*.

Un NIDS utilise deux passerelles:

$$\text{NIDS} \sqcap \neg \text{IPS} \sqsubseteq \forall \text{monitors.Gateway} \sqcap = 2\text{monitors}$$

En ce qui concerne la visibilité topologique d'un NIDS, elle est donné par la règle suivante:

$$\text{can_detect}(A, H_S, H_D) \leftarrow \text{NIDS}(A) \wedge \text{route}(H_S, H_D, L) \wedge \text{monitors}(A, H_{G_i}) \wedge \text{monitors}(A, H_{G_j}) \wedge [H_{G_i}, H_{G_j}] \subset L$$

Un knowledge-based IDS est un IDS sur lequel des signatures peuvent être actives:

$$KIDS \sqsubseteq \forall active.Signature$$

Une signature détecte une attaque:

$$Signature \sqsubseteq \forall detect.Attack$$

Maintenant, la visibilité fonctionnelle ie la capacité d'un IDS à détecter une action intrusive selon sa méthode de détection est donnée comme suit:

$$KIDS(A) \wedge active(A, Sig) \wedge detect(Sig, K') \wedge AttackSubClass(K, K') \leftarrow func.vis(A, K)$$

Afin de mettre cette règle sous forme DL-safe, on rajoute $O(A) \wedge O(Sig) \wedge O(K') \wedge O(K)$

8.6 Alertes

Voir la traduction en logiques de description du format IDMEF.

8.7 Raisonner à partir du modèle M4D4

Les requêtes que l'on peut poser dans M4D4 sont par exemple les suivantes:

- Est-ce que le nœud est vulnérable ou non?
- Est-ce qu'il existe un autre analyseur capable de détecter la même attaque?
- Est-ce que cet analyseur a généré une alerte (en comparant l'intervalle de temps)?

9 Conclusion

Dans ce livrable, nous avons proposé une représentation du format d'alerte IDMEF en logique de description. Plus précisément, nous avons traduit l'intégralité de la RFC 4765. Nous avons également montré comment raisonner à partir d'une telle représentation en vue d'effectuer du clustering. Nous avons aussi représenté le modèle M4D4 en logique de description plus des règles DL-Safe. Cette combinaison est justifiée par le fait qu'elle soit décidable. Ainsi, nous avons proposé un langage commun en détection d'intrusions coopérative afin de permettre à plusieurs IDS et d'autres analyseurs de coopérer entre eux. Avoir un tel langage commun permet aussi de formuler différentes requêtes. Par exemple, on peut vouloir chercher les machines qui ne sont pas concernées par les alertes ou encore des machines potentiellement vulnérables, etc.

Par ailleurs, nous comptons appliquer les approches de gestion de l'incohérence que nous avons développées dans [2] et [9] afin d'effectuer la vérification d'alertes dans le cadre d'une détection d'intrusion coopérative.

References

- [1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [2] Salem Benferhat and Safa Yahı. Complexity and cautiousness results for reasoning from partially preordered belief bases. In *ECSQARU*, pages 817–828, 2009.
- [3] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, and Ulrike Sattler. Owl 2: The next step for owl. *J. Web Sem.*, 6(4):309–322, 2008.
- [4] Ian Horrocks. The fact system. In *TABLEAUX*, pages 307–312, 1998.
- [5] Ian Horrocks. Daml+oil: A reason-able web ontology language. In *EDBT*, pages 2–13, 2002.
- [6] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an OWL rules language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731. ACM, 2004.
- [7] Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, 2005.
- [8] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26, 1991.
- [9] Safa Yahı, Salem Benferhat, Sylvain Lagrue, Mariette Sérayet, and Odile Papini. A lexicographic inference for partially preordered belief bases. In *KR*, pages 507–517, 2008.