The Knowledge Engineering Review, Vol. 18:3, 221–241.
DOI: 10.1017/S0269888904000049 Printed in the United Kingdom

Using DAML + OIL to classify intrusive behaviours

JEFFREY UNDERCOFFER, ANUPAM JOSHI, TIM FININ and JOHN PINKSTON

Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD-21250, USA; email: undercoffer@umbc.edu, joshi@umbc.edu, finin@umbc.edu, pinkston@umbc.edu

Abstract

We have produced an ontology specifying a model of computer attack. Our ontology is based upon an analysis of over 4000 classes of computer intrusions and their corresponding attack strategies and is categorised according to system component targeted, means of attack, consequence of attack and location of attacker. We argue that any taxonomic characteristics used to define a computer attack be limited in scope to those features that are observable and measurable at the target of the attack. We present our model as a target-centric ontology that is to be refined and expanded over time. We state the benefits of forgoing dependence upon taxonomies in favour of ontologies for the classification of computer attacks and intrusions. We have specified our ontology using the DARPA Agent Markup Language+Ontology Inference Layer and have prototyped it using DAMLJessKB. We present our model as a target-centric ontology and illustrate the benefits of utilising an ontology in lieu of a taxonomy, by presenting a use-case scenario of a distributed intrusion detection system.

1 Introduction

Intrusion Detection Systems (IDSs) are categorised according to model – signature or anomaly, and scope – network-based or host-based. Anomaly detectors attempt to detect usage outside the bounds of pre-established statistical norms. To effectively do this, anomaly detectors must create and dynamically maintain usage profiles reflecting normative behaviour for all users and processes of the system. In contrast, misuse detectors filter usage against a static set of attack signatures. Essentially, anomaly detectors compare events to what is deemed to be acceptable while misuse detectors compare events to what is deemed to be unacceptable. The key differentiator between host-based and network-based IDSs is that a network-based IDS, although run on a single host, is responsible for an entire network, or some network segment, while a host-based IDS is only responsible for the host on which it resides.

Research in the field of intrusion detection (ID) has been ongoing for approximately 20 years. One of the earliest papers in the field is James Anderson's 1980 paper ''Computer security, threat monitoring, and surveillance''. Seven years later, Dorothy Denning wrote ''An intrusion detection model'' (1987), providing a framework for IDSs. Denning held that evidence of malicious activity would be reflected in the audit records of the affected host.

The overarching philosophy driving intrusion detection is that, due to their static nature, intrusion prevention techniques such as firewalls and routing filter policies fail to stop many types of attack. Therefore, no matter how secure you try to make your system, intrusions still happen and therefore must be detected. Consequently, IDSs form a second line of defence that identifies and reports attacks in near real time in order to mitigate their damage.

This philosophy has recently become the object of dispute. In June 2003 Gartner, Inc., a research and advisory firm, published a report (Gartner, Inc., 2003) unequivocally stating that they believe that IDSs are a market failure – having failed to provide value relative to their costs. The report recommends that enterprises invest in firewall technologies as an alternative and it predicts that IDSs will be obsolete by the year 2005.

In congressional testimony, William Wulf, the President of the National Academy of Engineering (Wulf, 2001), equated today's computer security paradigm to that of the Maginot Line.[1] Dr Wulf further states that today's software systems are inherently insecure and their reliance upon network-level security mechanisms, comprising packet-filtering IDSs and firewalls placed at network access points, only serve to further exacerbate cyber vulnerabilities Essentially, we agree with Dr Wulf and only partially agree with Gartner, Inc. It is far better to prevent an intrusion then to detect it after it has happened; however, in highly secure environments searching for evidence of an intrusion is a necessary operational requirement.

Given the performance of today's IDSs, the motivation for Gartner's recommendation is understandable. However, to rely solely upon barrier technologies will only further increase reliance on Dr Wulf's ''virtual Maginot Line''. Consequently, we do not believe that IDSs will become obsolete and we are committed to improving their performance. During our research we have concluded that many IDSs operate at too high of a level of abstraction and should be functioning at more atomic levels. Moreover, the means for classifying, communicating and reasoning about the atoms that make up computer operations are in need of improvement.

Because IDSs are either adjacent to or colocated with the target of an attack it is imperative that any classification scheme used to represent an attack be *target-centric*, where each taxonomic characteristic comprises properties and features that are observable by the target of the attack. Our ontology, therefore, only defines properties and attributes that are observable and measurable by the target of an attack. As a basis for establishing our a posteriori target-centric attack ontology, we evaluated and analysed over 4000 computer vulnerabilities and the corresponding attack strategies employed to exploit them.

A central component of an IDS is the taxonomy employed to characterise and classify the attack or intrusion, and a language that describes instances of that taxonomy. The language is paramount to the effectiveness of the IDS because information regarding an attack or intrusion needs to be intelligibly conveyed, especially in distributed environments, and acted upon. Several taxonomies have been proposed by the research community. Some include a descriptive language; however, most do not. Likewise, several attack languages have been proposed, but most are not grounded in any particular taxonomy, hence their associated classification schemes are ad hoc and localised. The inherent problem with this approach is threefold:

i. In order to operate over instances of the data model characterised by a particular taxonomy, the data model must be encoded within a software system. Any changes or updates to the data model necessitate a change to the software system.
ii. Taxonomies only provide schemata for classification. They may not contain the necessary and sufficient constructs needed to enable a software system to reason over an instance of the taxonomy, which is representative of the domain under observation.
iii. Most attack and signature languages are particular to specific domains, environments and systems, consequently they are not extensible and are not communicable between non-homogeneous systems, and their semantics are often vague and lack grounding in any formal logic.

To mitigate the effects of these problems, we suggest transitioning from taxonomies to ontologies. We have constructed a data model that characterises the domain of computer attacks and

[1] The Maginot Line, named after Andre Maginot, the French Minister of War 1928–1932, was a series of defensive fortifications built by France along her border with Germany and Italy. It failed however; in 1940 the German army bypassed the Maginot Line, entering France through a ''neutral'' third country and swiftly defeated her.

intrusions as an ontology and implement that data model with an ontology representation language. Ontologies, unlike taxonomies, provide powerful constructs that include machine interpretable definitions of the concepts within a domain and the relations between them. Ontologies, therefore, provide software systems with the ability to share a common understanding of the information at issue, in turn empowering software systems with a greater ability to reason over and analyse this information. Gruber (1993) defines an ontology as an explicit specification of a conceptualisation. The term, which is borrowed from philosophy, is used to provide a formal specification of the concepts and relationships that can exist between entities within a domain. Accordingly, ontologies are designed for the purpose of enabling knowledge sharing and reuse between the entities within a domain. In our case, those entities are Intrusion Detection Systems (IDSs) and IDS sensors.

Semantic languages differ from syntactic languages. Both have tags that define a grammar; however, semantic languages have additional tags to support the language's semantic properties. Ontology representation languages are a type of semantic language and may be mapped into first-order relational sentences and a set of first-order logic axioms. This mapping restricts the allowable interpretations of the non-logical symbols (i.e. relations, functions and constants) (Fikes & McGuinness, 2001), enabling instances of the ontology to be operated over using sound and complete theorem provers. In short, the ontology, combined with some logic system, constitutes *knowledge representation.*

Commenting on the Internet Engineering Task Force's emerging standard – the Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition (IDMEF) (Curry and Debar, 2003) – and its ability to enable interoperability between heterogeneous IDS sensors, Kemmerer and Vigna (2002) state that the IDMEF is a first step and that additional effort is needed to provide a common ontology that lets IDS sensors agree on what they observe. We address Kemmerer and Vigna's comment by illustrating the benefits of using ontologies by presenting an implementation of one being utilised by a distributed intrusion detection system. We have constructed our ontology using the DARPA Agent Markup Language+Ontology Inference Layer (DAML+OIL) (Hendler, 2001) and have prototyped its logic using DAMLJessKB (Kopena, 2001) an extension to the Java Expert System Shell (Friedman-Hill, 1977).

The remainder of this paper is organised as follows: Section 2 presents an overview of our intrusion detection framework. Section 3 presents related work in the area of attack taxonomies, attack languages and ontologies for intrusion detection. Section 4 details the motivation for transitioning from taxonomies to ontologies. Our ontology is presented in Section 5. Section 6 details our implementation and Section 7 provides a use-case scenario illustrating the utility of using an ontology in detecting instances of denial of service, Mitnick and buffer overflow attacks. We conclude with Section 8.

2 Two-phase intrusion detection framework

Although our IDS model is not the focus of this paper, we describe it in order to provide context to the reader. Similar to Denning's approach, we hold that evidence of malicious activity is reflected in system-level data; however, by analysing data at a much lower level than audit data, we can profile the system's normative state and perform anomaly detection instead of signature detection. The underlying premise of our intrusion detection model is to describe attacks as instances of an ontology using a semantically rich language like DAML+OIL. This ontology captures information about attacks such as the system component it affects, the consequences of the attack, the means of the attack, the location of the attacker and so on. However, before anomalous system behaviour can be specified as an instance of the ontology, it first needs to be detected. Hence our intrusion detection model consists of two phases. The initial phase uses data mining techniques to analyse data streams that capture process, system and network states and detect anomalous behaviour. The

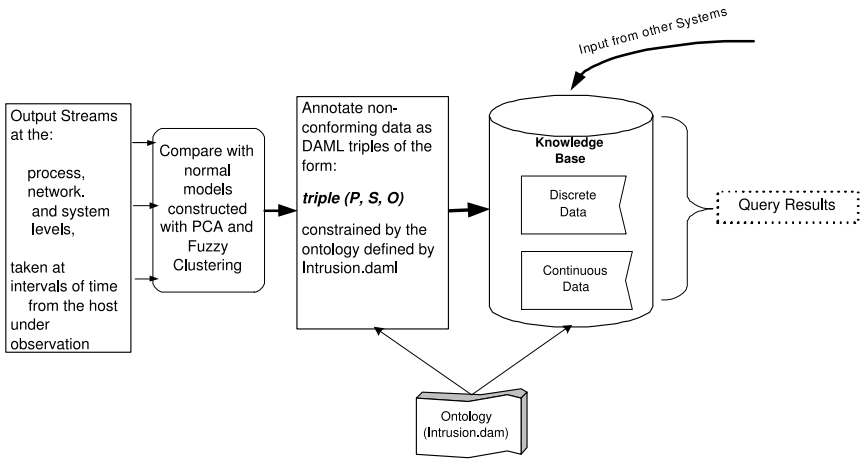

Figure 1 Single node of the distributed IDS framework

second, or high-level phase, reasons over anomalous data that is defined as an instance of an ontology. Figure 1 illustrates our IDS framework.

We have modelled the quiescent state of a system by instrumenting the Linux kernel and collect 190 system-level attributes in three separate data streams. Some work has already been done on using system call data and TCP packet information for building anomaly detection models (Hofmeyr *et al.*, 1998; Lee & Stolfo, 1998). Our model not only includes streams of system call and network data but also includes low-level kernel data such as memory maps, shared libraries, address offsets, calling addresses, return addresses, page faults, MIB data, interrupts, connections, users, CPU usage and so on. These data streams are provided by the operating system for the processes present in the system. Thus one can collect data for any process and build models that represent the quiescent state of the process and the system. We are exploring the use of fuzzy data mining and concepts introduced by the semantic web to operate in synergy to perform *distributed intrusion detection.*

We used Principal Component Analysis (PCA) (Golub & Loan, 1989) to reduce the dimensionality of the data set, the Mahalanobis metric (Mahalanobis, 1930) to measure distances between the data points and then use fuzzy clustering (Krishnapuram, 2001) on the reduced data set in order to obtain clusters that model the quiescent state of the system. (We are also experimenting with clustering based upon PDDP, Principal Direction Divisive Partitioning (Boley, 1998).) Once the baseline has been established, we use the Mahalanobis metric, in the case of fuzzy clustering, or Euclidean distance in the case of PDDP, as a dissimilarity measure in order to determine if subsequent data samples fall outside the bounds of the normative state. The second phase of our IDS *reasons* over the non-conforming sample from the data streams, which possibly represents anomalous behaviour. The sample, constrained by the ontology, is asserted into a *knowledge base* which is continually queried for evidence of an intrusion or an attack. Figure 1 illustrates a single component of our distributed system.

3 Related work

There is little, if any, published research formally defining ontologies for use in intrusion detection. Raskin *et al.* (2001) introduce and advocate the use of ontologies for information security. In stating the case for using ontologies, they claim that an ontology organises and systematises all of the phenomena (intrusive behaviour) at any level of detail, consequently reducing a large diversity of items to a smaller list of properties.

The preponderance of existing research in the area of the classification of computer attacks is limited to taxonomies and the taxonomies that are implicit in attack languages. The following subsections address taxonomies and attack languages.

3.1 Related work: taxonomies

There are numerous attack taxonomies proposed for use in intrusion detection research.

Landwehr *et al.* (1994) present a taxonomy categorised according to genesis (how), time of introduction (when) and location (where). They include sub-categories of *validation errors*, *boundary condition errors* and *serialisation errors*, as a means of effecting an intrusion. We have incorporated these sub-categories into our ontology.

Weber, during the 1998 and 1999 DARPA Off Line Intrusion Detection System Evaluations (Haines *et al.*, 2001; Kendall, 1999; Lippmann *et al.*, 2000), provided a taxonomy that defined the category *Consequence.* This includes the sub-categories of *Denial of Service*, *Remote to Local*, *User to Root* and *Probe.* We have incorporated these classifications into our work.

In defining their taxonomy, Lindqvist and Jonsson (1997) state that they ''focus on the external observations of attacks and breaches which the system owner can make''. Our effort is consistent with their focus because we hold that, since IDSs are either adjacent to or colocated with the target of an attack, it is imperative that any classification scheme used to represent an attack be *target-centric*, where each taxonomic character comprises properties and features that are observable by the target of the attack.

Ning *et al.* (2001) propose a hierarchical model for attack specification and event abstraction using three concepts essential to their approach: *System View*, *Misuse Signature* and *View Definition.* Their model is based upon a thorough examination of attack characteristics and attributes and is encoded within the logic of their proposed system. We include a global system view in our ontology.

As detailed by Allen *et al.* (2000) and McHugh (2000), the taxonomic characterisation of intrusive behaviour has typically been from the attacker's point of view, each suggesting that alternative taxonomies need to be developed. Allen *et al.* state that intrusion detection is an immature discipline and has yet to establish a commonly accepted framework. McHugh suggests classifying attacks according to protocol layer or, as an alternative, whether or not a completed protocol handshake is required. Likewise, Guha (1997) suggests an analysis of each layer of the TCP/IP protocol stack to serve as the foundation for an attack taxonomy. Consequently we have endeavoured to make our ontology as *target centric* as possible.

Aslam *et al.* (1996) observe that many potential faults and vulnerabilities are intrinsic to the software development process. Their observations are consistent with our own. Our ontology defines the class *Means of Attack* and comprises many of the attributes identified by Aslam *et al.*

Our intent is to not criticise the use of taxonomies. On the contrary, they have served their purpose well, particularly in identifying and classifying the characteristics of computer attacks and intrusions. We do, however, advocate leveraging their work by building upon existing taxonomies and transitioning to ontologies. We feel that this is necessary and warranted because, according to Staab and Maedche (2000), whereas taxonomies do not contain the necessary meta-knowledge required to convey modelling primitives such as concepts, relations and axioms that are required to make sense of and operate on specific objects, ontologies do. It should be pointed out that a complete and well-formed ontology subsumes a taxonomy.

3.2 Related work: attack languages

There are several *attack languages* proposed in the literature. These languages are often categorised as event, response, reporting, correlation, and recognition languages (Doyle *et al.*, 2001; Eckmann *et al.*, 2002). We concentrate on correlation, reporting and recognition languages because an ontology representation language is able to simultaneously provide the functionality of all three.

The P-BEST (Production-Based Expert System) Toolset (Lindqvist & Porras, 1999) is a correlation language from which users may specify the inference formula for reasoning and acting upon facts asserted into its fact base and from facts derived from external events. According to

Doyle *et al.* (2001), the P-BEST language lacks concepts that are specific to event recognition and consists solely of a formalism for expressing probabilistic and linguistic rules.

STATL (Eckmann *et al.*, 2002) is an extensible state/transition-based attack detection language designed to support intrusion detection. STATL allows one to describe computer penetrations as sequences of actions that an attacker performs in order to compromise a computer system. STATL lacks constructs for combining sub-events into larger events.

LogWeaver (Goubault-Larrecq, 2001) is a log auditing tool that takes a system log as input and processes it according to a signature (rule) file. The signature file defines the types of event that are to be monitored and reported on. LogWeaver is able to match regular expressions and make correlations between events, provided that they are executed by the same user. LogWeaver employs logic that is based upon *model checking* (Roger & Goubault-Larrecq, 2001). Essentially, LogWeaver is a specification for a detection language, which defines a syntax and grammar for the end-user to use when writing signatures.

The Internet Engineering Task Force's (IETF) proposed Intrusion Detection Message Exchange Format Data Model (IDEMF) and Extensible Markup Language (XML) Document Type Definition (Curry & Debar, 2003) is a profound effort to establish an industry-wide data model which defines computer intrusions. It defines a data model that is representative of data exported by an IDS. It also defines data formats and exchange procedures for inter/intra-IDS exchanges. The data model is defined in an XML *document type definition* and implemented in the Extensible Markup Language (XML) (W3C, 2003). The IDMEF assumes a hierarchal configuration of three IDS components: *sensors*, *analysers* and *managers.* Sensors are located at the bottom-most level of the hierarchy. Sensors output data to analysers, which in turn report up to a manager, located at the top-most level of the hierarchy.

Because the IDMEF data model, encoded in XML, is an emerging standard, we compare and contrast it to the notion of using ontologies to represent the data model and the subsequent encoding of the data model in an ontology representation language. The IDMEF's principal shortcoming is its use of XML, which is limited to a syntactic representation of the data model. This limitation requires that each individual IDS interpret and implement the data model programmaticaly. This shortcoming may be mitigated by using an ontology representation language such as DAML+OIL. This is not an indictment of XML, which serves its designer's intentions – a syntactic markup language.

For IDS purposes the ontology specification language DAML+OIL, which is a descriptive logic language and is grounded in both model-theoretic[2] and axiomatic semantics,[3] provides a viable alternative to XML. Replacing the IDMEF with a data model specified by an ontology representation language will result in:

i. A model that includes the attributes and characteristics of the specific domain.
ii. A decoupling of the data model from the underlying system of computational logic.
iii. The ability to report the existence of an instance of the domain (model) in a manner that is ''comprehensible'' by any entity that possess the specific ontology.
iv. Aggregate specific instances of the domain in a knowledge base and enable the conclusion that some larger, or more comprehensive, instance of the ontology exists.

4 From taxonomies to ontologies: the case for ontologies

An ontology subsumes a taxonomy; therefore, before explaining ontologies, a clear understanding of the definition, purpose and objective, and shortcomings of a taxonomy is in order.

A *taxonomy* is a *classification* system where the classification scheme conforms to a systematic arrangement into groups or categories according to established criteria (Websters, Inc., 1993). Glass

[2] Model-theoretic semantics is the process of constructing mathematical models of logical consequence and establishing when the model satisfies a formula.
[3] Axiomatic semantics is the process of defining a language using axioms and proof rules.

and Vessey (1995) contend that taxonomies provide a set of unifying constructs so that the area of interest can be systematically described and aspects of relevance may be interpreted. The overarching goal of any taxonomy, therefore, is to supply some predictive value during the analysis of an unknown specimen, while the classifications within the taxonomy offer an explanatory value.

According to Simpson (1961), classifications may be created either a priori or a posteriori. An a priori classification is created non-empirically whereas an a posteriori classification is created by empirical evidence derived from some data set. Simpson defines a taxonomic character as a feature, attribute or characteristic that is divisible into at least two contrasting states and used for constructing classifications. He further states that taxonomic characters should be observable from the object in question.

Amoroso (1994), Lindqvist and Jonsson (1997), Krusl (1998) and others have identified what they believe to be the requisite properties of a sufficient and acceptable taxonomy for computer security.

Most taxonomies used in IDS research imply a *crisp* border between the classes and their attributes within a domain. Unfortunately, most domains, including computer security, are not mutually exclusive and are often full of ambiguities. For instance, some computer attacks may be viewed as a class that is combination of two or more anomalous conditions. Specifically, they constitute a class that is derived through multiple inheritance. In a *rule-based* system the computational rules required to adequately deal with events spanning multiple classes quickly become overly complex. Fortunately, ontologies mitigate some of these complexities.

In applying ontologies to the problem of intrusion detection, the power and utility of the ontology is not realised by the simple representation of the attributes of the attack. Instead, the power and utility of the ontology is realised by the fact that we can express the relationships between collected data and use those attributes and their constructs, relationships and saliency to deduce that the particular data represents an attack of a particular type. Moreover, specifying an ontological representation decouples the data model defining an intrusion from the logic of the intrusion detection system. The decoupling of the data model from the IDS logic enables heterogeneous IDSs to share data without a prior agreement as to the semantics of the data. To effect this sharing, an instance of the ontology is shared between IDSs in the form of a set of DAML+OIL (or RDF) statements. If the recipient does not understand some aspect of the data, it obtains the ontology in order to interpret and use the data as intended by its originator.

Ontologies, therefore, unlike taxonomies, provide powerful constructs that include machine interpretable definitions of the concepts within a specific domain and the relations between them. In our case the domain is that of a particular computer or a software system acting on the computer's behalf in order to detect attacks and intrusions. Ontologies may be utilised to not only provide an IDS with the ability to share a common understanding of the information at issue but also further enable the IDS with improved capacity to reason over and analyse instances of data representing an intrusion. Moreover, within an ontology, characteristics such as cardinality, range and exclusion may be specified and the notions of inheritance and multiple inheritance are supported.

The following exemplifies the differences between an ontology and a taxonomy. The class ''family'' (abbreviated for our purposes) is specified as a taxonomy using an XML Document Type Definition (DTD) and as an ontology using DAML+OIL. The DTD is depicted in Figure 2, Figure 3 contains an instance of the DTD and Figure 4 contains the DAML specification of the class ''family''.

Given the DTD and its XML instance, a rule-based expert system would be hard pressed to answer the query ''list all females and their female children''. In order to correctly answer (the answer is Bea and her children Jane and June) domain knowledge regarding the nature of familial relationships that is not and cannot be encoded in a DTD is required.

Alternatively, the DAML-specified ontology imparts the requisite domain knowledge. Specifically, that daughters and mothers are restricted to female persons and that there is a parent–child relationship between mothers and daughters. Given the ontology of Figure 4 and the

```
<?xml version="1.0"?>
<!ELEMENT name (#PCDATA)>
<!ELEMENT son (#PCDATA)>
<!ELEMENT daughter (#PCDATA)>
<!ELEMENT sex (#PCDATA)>
<!ELEMENT mother (#PCDATA)>
<!ELEMENT person (name, son*, daughter*, sex?, mother?)>
<!ELEMENT family (person*)>
```

Figure 2 DTD specification of a family

```
<family>
  <person>
     <name>Bob</name>
      <son>Jake</son>
      <daughter>Jane</daughter>
      <daughter>June</daughter>
      <sex>male</sex>
    </person>
  <person>
     <name>Bea</name>
  </person>
  <person>
     <name>Jake</name>
     <mother>Bea</mother>
  </person>
  <person>
       <name>Jane</name>
       <mother>Bea</mother>
  </person>
  <person>
        <name>June</name>
        <mother>Bea</mother>
  </person>
</family>
```

Figure 3 Instance of the DTD specified Family

data in Figure 3 it follows that since Jane and June are Bob's daughters they are female, and that Jane and June's mother is Bea; because a mother is a female person, we may infer that Bea is a female with female children Jane and June.

5 Target-centric ontology: attributes of the class *intrusion*

In constructing our ontology, we conducted an empirical analysis (Undercoffer & Pinkston, 2002) of the features and attributes, and their interrelationships, of over 4000 classes of computer attack and intrusion that are contained in the CERT/CC Advisories and the ''Internet Catalog of Assailable Technologies'' (ICAT) maintained by NIST. Our analysis indicates that the overwhelming majority of attacks are the result of malformed input exploiting a software vulnerability of a network-attached process. According to CERT, root access is the most common consequence, while according to ICAT, a denial of service is the most common consequence. During our analysis we observed that an intrusion or attack consisted of:

- Input directed to a *system component*
  - network protocol stack
  - a running process
  - the system
- The input served as an agent of change, causing an *aberrant condition* consisting of
  - input validation errors
  - logic exploits
  - configuration errors

```
<daml:Class rdf:about="Family#Family" rdfs:label="Family">
</daml:Class>

<daml:Class rdf:about="Family#Person" rdfs:label="person">
</daml:Class>

<daml:Class rdf:about="Family#mother" rdfs:label="mother">
     <rdfs:subClassOf>
        <daml:Class rdf:about="Family#person">
     </rdfs:subClassOf>
     <rdfs:subClassOf>
        <daml:Restriction daml:sex="Family#female"/>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
        <daml:Class rdf:about="Family#Family">
     </rdfs:subClassOf>
</daml:Class>

<daml:ObjectProperty rdf:about="Family#mother_of" rdfs:label="mother_of">
     <rdfs:domain rdfs:about="Family#mother">
     <rdfs:range rdf:"resource&daughter"/>
</daml: ObjectProperty>

<daml:ObjectProperty rdf:about="Family#name rdfs:label=name>
     <rdfs:domain rdfs:about="Family#person >
     <rdfs:range rdf:resource&Literal"/>
</daml: ObjectProperty>

<daml:ObjectProperty rdf:about="Family#sex rdfs:label=sex>
     <rdfs:domain rdfs:about="Family#person >
     <rdfs:range rdf:resource&Family#male"/>
     <rdfs:range rdf:resource&Family#female"/>
</daml: ObjectProperty>

<daml:Class rdf:about="Family#Daughter" rdfs:label="daughter">
     <rdfs:subClassOf>
     <daml:Class rdf:about="Family#child">
     </rdfs:subClassOf>
     <rdfs:subClassOf>
         <daml:Restriction daml:sex="Family#female"/>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
         <daml:Class rdf:about="Family#Family">
     </rdfs:subClassOf>
</daml:Class>
```

Figure 4 DAML+OIL Specified Family

- The aberrant condition resulted in some unintended *consequence*
  - probe
  - denial of service
  - remote to local
  - user to root

Figure 5 presents a high-level view of our ontology. The lower ontology (leaf attributes of each class and subclass) are not depicted because it would make the illustration unwieldy. As stated in Section 1, we have instrumented the Linux kernel, using it to gather 190 distinct attributes at the system, process and network levels. Consequently, our ontology, and the taxonomy that it subsumes, is defined solely in terms of the causal relationships of the observables and measurables at the target of the attack.

It should be noted that an RDF graph does not depict flow. In an RDF graph, ellipses are used to denote a class, which may have several properties. When two vertices (classes) are connected by a directed edge, the edge represents a property whose domain is denoted by the start of the edge, and whose range is denoted by the end of the edge. An undirected edge between two vertices (classes) indicates that one class is an instance of another class.

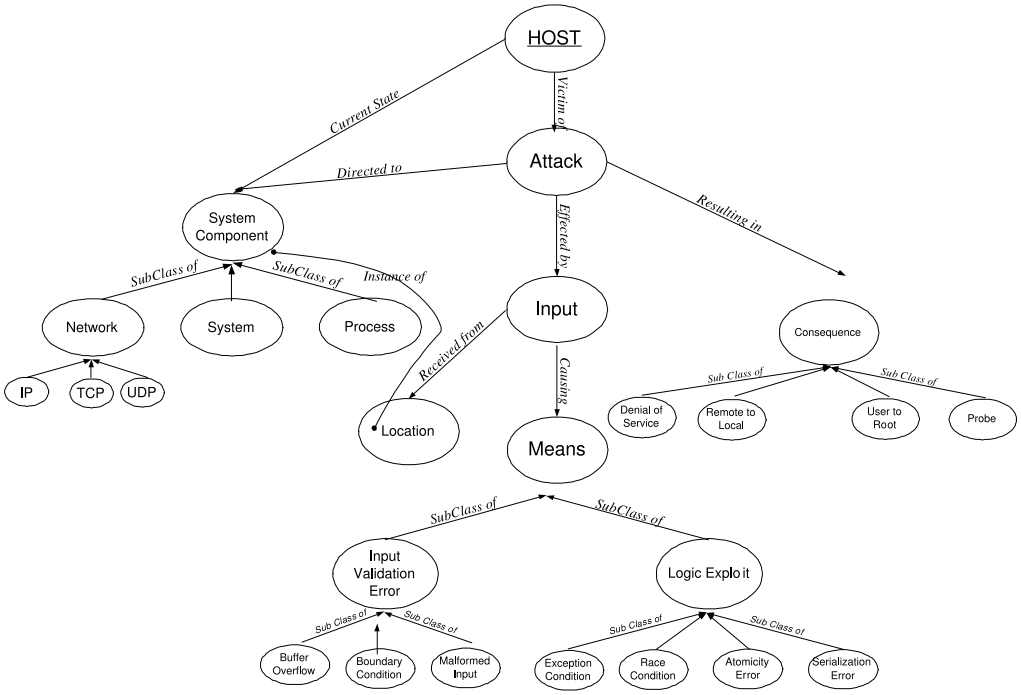

Figure 5 Target-centric ontology

At the top-most level of Figure 5 we define the class *Host. Host* has the predicates *Current State* and *Victim of*. *Current State* ranges over *System Component* and *Victim of* ranges over the class *Attack*. As earlier stated, the predicate defines the relationship between a subject and an object.

The System Component class comprises the following subclasses:

i. **Network** This class is inclusive of the network layers of the protocol stack. We have focused on TCP/IP, therefore we only consider the IP, TCP and UDP subclasses. For example, and as will be later demonstrated, the TCP subclass includes the properties *TCP_MAX*, *WAIT_STATE*, *THRESHOLD* and *EXCEED_T*. *TCP_MAX* defines the maximum number of TCP connections. *WAIT_STATE* defines the number of connections waiting on the final *ack* of the three-way handshake to establish a TCP connection. *THRESHOLD* specifies the allowable ratio between maximum connections and partially established connections. *EXCEED_T* is a boolean value indicating that the allowable ratio has been exceeded. It should be noted that these are only four of several network properties.

ii. **System** This includes attributes representing the operating system of the host. It includes attributes representing overall memory usage (*MEM_TOTAL*, *MEM_FREE*, *MEM_SWAP*) and CPU usage (*LOAD_AVG*). The class also contains attributes reflective of the number of current users, disk usage, the number of installed kernel modules and change in state of the interrupt descriptor and system call tables.

iii. **Process** This class contains attributes representing particular processes that are to be monitored. These attributes include the current value of the instruction pointer (*INS_P*), the current top of the stack (*T_STACK*), a scalar value computed from the stream of system calls (*CALL_V*) and the number of child processes (*N_CHILD*).

The class *Attack* has the properties *Directed to*, *Effected by* and *Resulting in.* This construction is predicated upon the notion that an attack consists of some input which is directed to some system component and results in some consequence. Accordingly, the classes *System Component*, *Input* and *Consequence* are the corresponding objects. The class *Consequence* comprises several subclasses which include:

i. **Denial of Service** The attack results in a denial of service to the users of the system. The denial of service may be because the system was placed into an unstable state or all of the system resources may be consumed by meaningless functions.
ii. **User Access** The attack results in the attacker having access to services on the target system at an unprivileged level.
iii. **Root Access** The attack results in the attacker being granted privileged access to the system, consequently having complete control of the system.
iv. **Probe** This type of an attack is the result of scanning or other activity wherein a profile of the system is disclosed.

Finally, the class *Input* has the predicates *Received from* and *Causing*, where *Causing* defines the relationship between the *Means* of attack and some input and *Received from* defines the relationship between *Input* and *Location.* The class *Location* is an instance of *System Component* and is restricted to instances of the Network and Process classes.

We define the following subclasses for *Means* of attack:

i. **Input validation error** An input validation error exists if some malformed input is received by a hardware or software component and is not properly bounded or checked. This class is further sub-classed as:
   (a) **Buffer overflow** The classic buffer overflow results from an overflow of a static-sized data structure.
   (b) **Boundary condition error** A process attempts to read or write beyond a valid address boundary or a system resource is exhausted.
   (c) **Malformed input** A process accepts syntactically incorrect input or extraneous input fields, or the process lacks the ability to handle field-value correlation errors.
ii. **Logic exploits** Logic exploits are exploited software and hardware vulnerabilities such as race conditions or undefined states that lead to performance degradation and/or system compromise. Logic exploits are further subclasssed as follows:
   (a) **Exception condition** An error resulting from the failure to handle an exception condition generated by a functional module or device.
   (b) **Race condition** An error occurring during a timing window between two operations.
   (c) **Serialisation error** An error that results from the improper serialisation of operations.
   (d) **Atomicity error** An error occurring when a partially modified data structure is used by another process; An error occurring because some process terminated with partially modified data where the modification should have been atomic.

6 Prototype implementation

There are several *reasoning systems* that are compatible with DAML+OIL (Frank *et al.*, 2003; Haarslev & Moller, 2001; Kopena, 2001; Horrocks *et al.*, 2000), some using descriptive logics and others predicate calculi (first order logic). Additionally, they are also classified according to their functionality, which is either backward-chaining or forward-chaining. Backward-chaining reasoners process queries and return proofs for the answers they provide. Forward-chaining reasoners process assertions substantiated by proofs, and draw conclusions.

We have prototyped the logic portion of our system using the DAMLJessKB (Kopena, 2001) reasoning system. We found DAMLJessKB to be easy to use and to integrate with other components of our system. However, the version used does not provide complete reasoning over DAML+OIL, requiring us to augment the system with some hand-crafted JESS rules. For example, the current JESS reasoning system is not able to compute all of the subsumption relationships which apply to new classes and instances. To overcome this limitation, we added specific JESS rules to ensure that the subsumption relationships key to our domain are computed.

Future versions of our system will move from DAML+OIL to OWL and we will replace the JESS-based reasoner with one that is complete, such as JTP, that will draw all of the inferences

```
<rdfs:Class rdf:about=
      "&IntrOnt;Attack"
   rdfs:label="Consequence">
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>

<rdf:Property rdf:about="&IntrOnt;Directed_To"
    rdfs:label="Directed_To">
   <rdfs:domain rdf:resource="&IntrOnt;Attack"/>
   <rdfs:range rdf:resource="&IntrOnt;SysComp"/>
</rdf:Property>

<rdf:Property rdf:about="&IntrOnt;Resulting_In"
    rdfs:label="Resutling_In">
   <rdfs:domain rdf:resource="&IntrOnt;Attack"/>
   <rdfs:range rdf:resource="&IntrOnt;Conseq"/>
</rdf:Property>

<rdf:Property rdf:about="&IntrOnt;Effected_By"
    rdfs:label="Effected_By">
   <rdfs:domain rdf:resource="&IntrOnt;Attack"/>
   <rdfs:range rdf:resource="&IntrOnt;Input"/>
</rdf:Property>
```

Figure 6 DAML+OIL statements defining the class attack and its properties: directed to, resulting in and effected by

licensed by the ontologies and instance data. This will simplify the architecture as well as enhance the process of debugging, extending and maintaining the system.

Accordingly, we reason over instances data that do not conform to our data model and are considered to be suspicious. These suspicious instances are constrained according to our ontology and asserted into the knowledge base.

Upon initialisation of DAMLJessKB, we parse the DAML+OIL statements representing the ontology, convert them into *N-Triples* (W3C, 2001), and assert them into a knowledge base as rules. The assertions are of the form

```
(assert
(PropertyValue (predicate) (subject) (object)))
```

Once asserted, DAMLJessKB generates additional rules which include the chains of implication derived from the ontology.

The following series of figures illustrate the DAML+OIL encoding of selected classes, subclasses and their respective properties, of our ontology.

Figure 6 lists the DAML+OIL statements defining the class *Attack* and its properties *Directed To*, *Resulting In* and *Effected By*. These properties correspond to the edges between the node labelled Target and the nodes labelled *System Component*, *Input* and *Consequence* respectively in Figure 5.

Figure 7 presents the DAML+OIL notation for the class *System Component*, its subclass *Network* and *Network*'s subclass *TCP*. Figure 8 lists the DAML+OIL notation for some of the attributes of the class *TCP*.

Figure 9 details the specification of the class *Consequence* while Figures 10 and 11 show similar details for the specification of the classes *Denial of Service* and *Syn Flood*. The *Syn Flood* class, which is not shown in Figure 5 illustrating our ontology, is a subclass of both *Denial of Service* and *TCP* and, as stated in the DAML+OIL notation, will only be instantiated when the threshold of pending TCP connections is exceeded.

6.1 Querying the knowledge base

Once the ontology is asserted into the knowledge base and all of the derived rules resulting from the chains of implication are generated, the knowledge base is ready to receive instances of the

```
<daml:Class rdf:about="&IntrOnt;SysComp"
    rdfs:label="State">
   <rdfs:subClassOf rdf:resource="&rdfs;
         Resource"/>
</daml:Class>

<daml:Class rdf:about="&IntrOnt;
      Network"
   rdfs:label="Network">
  <rdfs:subClassOf rdf:resource="&IntrOnt;
        SysComp"/>
</daml:Class>

<daml:Class rdf:about="&IntrOnt;TCP"
   rdfs:label="Network">
  <rdfs:subClassOf rdf:resource="&IntrOnt;
        Network"/>
</daml:Class>
```

Figure 7 DAML+OIL statements specifying the class *System Component* and its subclasses *Network* and *TCP*

```
rdf:Property rdf:about="&IntrOnt;TCP_Max"
   rdfs:label="TCP_Max">
  <rdfs:domain rdf:resource="&IntrOnt;Network"/>
  <rdfs:range rdf:resource="&rdfs;
        nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:about="&IntrOnt;Wait_State"
   rdfs:label="Wait_State">
  <rdfs:domain rdf:resource="&IntrOnt;Network"/>
  <rdfs:range rdf:resource=
        "&rdfs;nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:about="&IntrOnt;Threshold"
   rdfs:label="Threshold">
  <rdfs:domain rdf:resource="&IntrOnt;Network"/>
  <rdfs:range rdf:resource=
        "&rdfs;nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:about="&IntrOnt;Exceed_T"
 rdfs:label="Exceed_T">
<rdfs:domain rdf:resource="&IntrOnt;Network"/>
<rdfs:range rdf:resource="&IntrOnt;BooleanValue"/>
</rdf:Property>
```

Figure 8 DAML+OIL notation specifying attributes of the *TCP* subclass

```
<rdfs:Class rdf:about="&IntrOnt;Conseq"
   rdfs:label="Conseq">
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
```

Figure 9 DAML+OIL specification of the class *Consequence*

ontology. Instances are asserted and retracted into/from the knowledge base as temporal events dictate. The query language is of the form *((predicate) (subject) (object))* where at least one of the three elements of the triple must be contain a value. The other one or two elements may be left uninstantiated (signified by prefacing them with a ''?''). If there are any triples in the knowledge base that match the query either as the result of an assertion of a fact or derived rules resulting from the chain of implication, the value of those triples will be returned.

To query the knowledge base for the existence of an attack or intrusion, the query could be so precise that it requests an attack of a specific type, such as a Syn Flood:

```
<rdfs:Class rdf:about="&IntrOnt;DoS"
   rdfs:label="DoS">
  <rdfs:subClassOf rdf:resource="&IntrOnt;Conseq"/>
</rdfs:Class>
```

Figure 10 DAML+OIL statements specifying the *Denial of Service* subclass

```
<daml:Class rdf:about="&IntrOnt;Syn_Flood"
   rdfs:label="Syn_Flood">
  <rdfs:subClassOf rdf:resource="&IntrOnt;DoS"/>
     <rdfs:subClassOf rdf:resource="&IntrOnt;TCP">
       <daml:Restriction>
          <daml:onProperty rdf:resource=
                "&IntrOnt;Exceed_T"/>
          <daml:hasValue rdf:resource="#true"/>
        </daml:Restriction>
       </rdfs:subClassOf>
</daml:Class>
```

Figure 11 DAML+OIL statements specifying the *Syn Flood* subclass

```
(defrule isSynFlood
(PropertyValue
(p http://www.w3.org/1999/02/22-rdf-syntax-ns#type)
(s ?var)
(o http://security.umbc.edu/IntrOnt#SynFlood))
=>
(printout t ``A SynFlood attack has occurred.'' crlf
            ``with event number: `` ?var))
```

The query could be of coarse granularity, asking for all attacks of a specific class, such as *Denial of Service.* Accordingly, the following query will return all instances of an attack of the class *Denial of Service.*

```
(defrule isDOS
(PropertyValue
(p http://www.w3.org/1999/02/22-rdf-syntax-ns#type)
(s ?var)
(o http://security.umbc.edu/IntrOnt#DoS))
=>
(printout t ``A DoS attack has occurred.'' crlf
            ``with ID number: `` ?var))
```

Finally, the following rule will return instances of any attack, where the event numbers that are returned by the query need to be iterated over in order to discern the specific type of attack:

```
(defrule isConseq
(PropertyValue
(p http://www.w3.org/1999/02/22-rdf-syntax-ns#type)
(s ?var)
(o http://security.umbc.edu/IntrOnt#Conseq))
=>
(printout t ``An attack has occurred.'' crlf
            ``with ID number: `` ?var))
```

These varying levels of granularity are possible because of DAML+OIL's notion of classes and subclasses and the relationships that hold between them. The query variable *?var*, which

```
<Intrusion:Host rdf:about="&IntrOnt;00035"
 Intrusion:IP_Address="130.85.112.231"
 rdfs:label="00035">
<Intrusion:resulting_in rdf:resource=
                   "&IntrOnt;00038"/>
</Intrusion:Host>

<Intrusion:Syn_Flood rdf:about="&IntrOnt;00038"
 Intrusion:Exceed_T="true"
 Intrusion:time="15:43:12"
         Intrusion:date="02/22/2003"
 rdfs:label="00038"/>
```

Figure 12 DAML+OIL notation an instance of a *Syn Flood* attack

corresponds to the subject, contained in each of the queries, is instantiated with the subject whenever a predicate and object from a matching triple is located in the knowledge base.

7 Using the ontology to detect attacks: use-case scenarios

To test our implementation and experiment with it, we simulated instances of our ontology in DAML+OIL notation, and asserted them into the knowledge base. We then ran our queries against the knowledge base.

7.1 Denial of Service – Syn Flood

The DAML+OIL representation of an instance of a *Syn_Flood* attack is illustrated in Figure 12. The first statement indicates that an event numbered 00035 has occurred, which has the *resulting_in* property instantiated to an instance of a *Syn Flood* that is uniquely identified as event number 00038.

When the knowledge base was queried for instances of *Denial of Service* (DoS) attacks, the following was returned:

```
The event number of the intrusion is:
http://security.umbc.edu/Intrusion#00038
The type of intrusion is:
http://security.umbc.edu/Intrusion#Syn_Flood
The victim's IP address is:
130.85.112.231
The time and date of the event:
15:43:12 hours on 02/22/2003
```

It is important to note that we only queried for the existence of a *Denial of Service* attack, we did not specifically ask for *Syn Flood* attacks. The instance of the *Syn Flood* attack was returned because it is a subclass of *Denial of Service.*

7.2 The classic Mitnick-*type attack*

This subsection provides an example of using our ontology as it operates within a coalition of distributed IDSs to detect the *Mitnick* attack. This particular attack is a distributed attack consisting of a *Denial of Service* attack, TCP sequence number prediction and IP spoofing.

The following example of a distributed attack illustrates the utility of our ontology.

The *Mitnick* attack is multi-phased; consisting of a *Denial of Service* attack, TCP sequence number prediction and IP spoofing. When this attack first occurred in 1994, a *Syn Flood* was used to effect the denial of service; however, any denial of service attack would have sufficed.

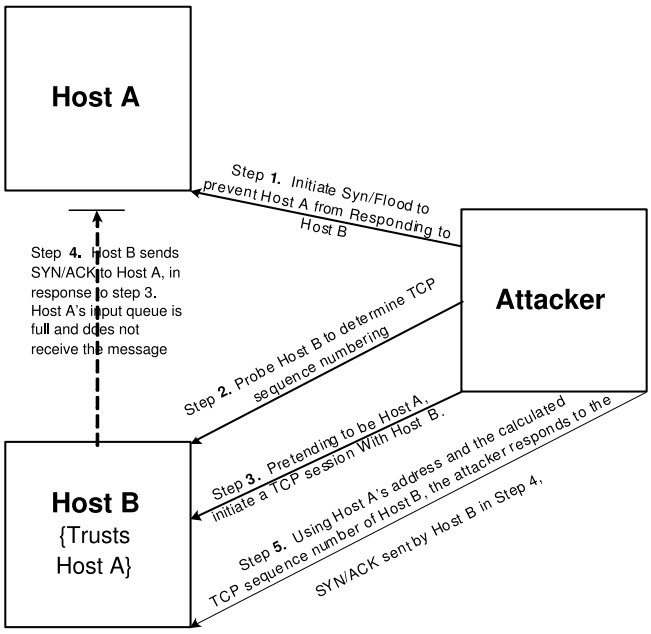

Figure 13 Illustration of the *Mitnick* attack

In the following example, which is illustrated in Figure 13, **Host B** is the ultimate target and **Host A** is trusted by **Host B.**

The attack is structured as follows:

1. The attacker initiates a *Syn Flood* attack against **Host A** to prevent **Host A** from responding to **Host B.**
2. The attacker sends multiple TCP packets to the target, **Host B**, in order to be able to predict the values of TCP sequence numbers generated by **Host B.**
3. The attacker then pretends to be **Host A** by spoofing **Host A**'s IP address, and sends a Syn packet to **Host B** in order to establish a TCP session between **Host A** and **Host B.**
4. **Host B** responds with a SYN/ACK to **Host A.** The attacker does not see this packet. **Host A**, since its input queue is full due to the number of half-open connections caused by the *Syn Flood* attack, cannot send an *RST* message to **Host B** in response to the spurious Syn message.
5. Using the calculated TCP sequence number of **Host B** (recall that the attacker did not see the SYN/ACK message sent from **Host B** to **Host A**) the attacker sends an *Ack* with the predicted TCP sequence number packet in response to the SYN/ACK packet sent to **Host A.**
6. **Host B** is now in a state of belief that a TCP session has been established with a trusted host **Host A.** The attacker now has a one-way session with the target, **Host B**, and can issue commands to the target.

It should be noted that an intrusion detection system running exclusively at either host will not detect this multi-phased and distributed attack. At best, **Host A**'s IDS would see a relatively short-lived *Syn Flood* attack, and **Host B**'s IDS might observe an attempt to infer TCP sequence numbers, although this may not stand out from other non-intrusive but ill-formed TCP connection attempts.

The following example illustrates the utility of our ontology, as well as the importance of forming coalitions of IDSs. In our model, all of the IDSs share a common ontology and utilise a secure communications infrastructure that has been optimised for IDSs. We present such a communications infrastructure in Undercoffer *et al.* (2003).

Consider the case of the instance of the *Syn Flood* attack presented in Section 7.1, and that it was directed against **Host A** in our example scenario. Since the IDS responsible for **Host A** is continually monitoring for anomalous behaviour, asserting and retracting data as necessary, it detects the occurrence of an inordinate number of partially established TCP connections, and

```
<IntrOnt:Connection rdf:about="&IntrOnt;00038"
 IntrOnt:IP_Address="130.85.112.231"
 IntrOnt:conn_time="20021212 154417"/>

<IntrOnt:Connection rdf:about="&IntrOnt;00101"
 IntrOnt:IP_Address="202.85.191.121"
 IntrOnt:conn_time="20021212 151221"/>

<IntrOnt:Connection rdf:about="&IntrOnt;00102"
 IntrOnt:IP_Address="68.54.101.78"
 IntrOnt:conn_time="20021212 150152"/>
```

Figure 14 DAML+OIL notation for instances of connections

```
<daml:Class rdf:about="&Intrusion;P\_Mitnick"
   rdfs:label="P_Mitnick">
    <daml:intersectionOf rdf:parseType=
                              ''daml:collection''>
      <daml:Class rdf:about="&IntrOnt;DoS"/>
      <daml:Class rdf:about="&IntrOnt;Connection"/>
    </daml:intersectionOf>
</daml:Class>
```

Figure 15 DAML+OIL specification of the *Mitnick* attack

transmits the instance of the *Syn Flood* illustrated in Figure 12 to the other IDSs in its coalition. It should be noted that outgoing TCP connections (or UDP multicasts) can still be used by **Host A** to transmit data to other IDSs during a *Syn Flood* attack.

This instance is converted into a set of *N-Triples* and asserted into the knowledge base of each IDS in the coalition. (Note: those same *N-Triples* will be retracted when the responsible IDS transmits a message stating that the particular host is no longer the victim of a *Syn Flood* attack.) Since this situation, especially in conjunction with **Host B** being subjected to a series of probes meant to determine its TCP sequencing, is anomalous and may be the prelude to a distributed attack, the current and pending connections are also asserted into the knowledge base.

Figure 14 lists the set of DAML+OIL statements describing those connections that were used in our experiments.

Figure 15 illustrates the DAML+OIL notation specifying the *Mitnick* attack. The class is identified as P_Mitnick for partial because the Mitnick attack has the property that the connection time with the victim must be greater than or equal to the time of the denial of service attack.

DAML+OIL, like any other notation language, does not have the functionality to perform mathematical operations. Consequently, when querying for the existence of a Mitnick type of attack, we must define a rule that tests for concomitance between the DoS attack and the establishment of the connection with the target of the DoS attack. The following query performs that test:

```
(defrule isMitnick
(PropertyValue
(p    http://security.umbc.edu/IntrOnt#Mitnick    )(s    ?eventNumber)(o
``true''))
(PropertyValue
(p     http://security.umbc.edu/IntrOnt#Int_time)(s     ?eventNumber)(o
?Int_Time))
```

```
(PropertyValue
(p http://security.umbc.edu/IntrOnt#Conn_time)(s ?eventNumber)
(o ?Conn_Time))
=>
(if (>= ?Conn_Time ?Int_Time) then
(printout t ``event number: `` ?eventnumber `` is a Mitnick Attack:
crlf)))
```

This query makes the correlation between event Number 00043, the connection occurring at 15:44:17, with the host at IP address 130.85.112.23, and event number 00038, the *Denial of Service* attack. The query, in conjunction with the other queries, produced the following response:

```
The synflood attack is:
http://security.umbc.edu/Intrusion#00038
The dos attack is:
http://security.umbc.edu/Intrusion#00038
The event number of the connection is:
http://security.umbc.edu/Intrusion#00043
The mitnick attack is:
http://security.umbc.edu/Intrusion#genid21
A connection with 130.85.112.231 was
made at 15:44:17 on 02/22/2003
```

where event number *genid21* was generated through a chain of implication based upon events 00038 and 00043 and the specification of the Mitnick attack in the ontology.

At this point, it is important to review the sequence of events leading up to the discovery of the Mitnick attack. Recall that the IDS responsible for the victim of the Syn Flood attack queried its knowledge base for an instance of a *DoS* denial of service attack. The query returned an instance of a Syn Flood, which was instantiated solely on the condition that a *Syn Flood* is a subclass of both the *DoS* and *Network* classes restricted to the value of *Exced_T* being true.

The instance (its properties) of the Syn Flood attack was transmitted in the form of a set of DAML+OIL statements to the other IDSs in the coalition. In turn, these IDSs converted the DAML+OIL notated instance into a set of *N-Triples* and asserted them into their respective knowledge bases. As a Syn Flood is a precursor to a more insidious attack, instances of established and pending connections were asserted into the knowledge base. As the state of the knowledge base is dynamic, due to the assertions and retractions, the rule set of each IDS is continually applied to the knowledge base.

Finally, the instance of the Mitnick attack was instantiated by the knowledge base, based upon the existence of both the instance of the TCP connection and the instance of the DoS attack.

7.3 Buffer overflow attack

The ``C'' strcpy() function is one of several functions that needs to be bounded in order to prevent a buffer overflow attack. A buffer overflow attack occurs when deliberately constructed code is placed onto the stack frame, overwriting the return address from the current function. When a function is called, input parameters to the function, the frame pointer(ebp register) and the return address (the current eip+the length of the call instruction) are pushed onto the stack. Like all instructions, they are located in the *Text* address space of memory.

As previously stated, we have instrumented the Linux kernel and are able to intercept any given process at each system call and examine the contents of its registers and stack frame. Consequently, we are able to define the characteristics of a buffer overflow attack such that the instruction pointer references a memory location that is outside the boundaries of the Text segment. Figure 16 presents the DAML+OIL notation for the class *Buffer Overflow* and one of its properties.

```
<daml:Class rdf:about="&IntrOnt;Buff_OF"
   rdfs:label="Buff_OF">
  <rdfs:subClassOf rdf:resource=
        "&IntrOnt;R_to_L"/>
     <rdfs:subClassOf rdf:resource=
          "&IntrOnt;U_to_R">
       <rdfs:subClassOf rdf:resource=
            "&IntrOnt;Process">
          <daml:Restriction>
          <daml:onProperty rdf:resource=
               "&IntrOnt;EIP_out_Txt"/>
          <daml:hasValue rdf:resource="#true"/>
        </daml:Restriction>
       </rdfs:subClassOf>
</daml:Class>

<rdf:Property rdf:about="&IntrOnt;EIP_out_Txt"
 rdfs:label="EIP_out_Txt">
        <rdfs:domain rdf:resource="&IntrOnt;
              Buff_OF"/>
        <rdfs:range rdf:resource="&IntrOnt;
              BooleanValue"/>
</rdf:Property>
```

Figure 16 DAML+OIL notation specifying the *Buffer Overflow* subclass

Similar to the previous two examples, querying the knowledge base with the following will yield all instances of a buffer overflow:

```
(defrule isBufferOverflow
(PropertyValue
(p http://www.w3.org/1999/02/22-rdf-syntax-ns#type)
(s ?var)
(o http://security.umbc.edu/IntrOnt#Buff_OF))
=>
(printout t ''A Buffer Overflow has occurred.'' crlf
          ''with ID number: '' ?var))
```

8 Conclusion and future work

We have stated the case for transitioning from taxonomies and the languages (event, correlation and recognition) employed by them to ontologies and ontology representation languages for use in intrusion detection systems. We have constructed and have presented an initial ontology, which is available at `http://security.cs.umbc.edu/Intrusion.daml`.

We have used the ontology specification language DAML+OIL to implement our ontology and to distribute information regarding system state within a distributed coalition. In the Mitnick example, the ontology (DAML+OIL) and an inference engine was initially employed as an event recognition language, by discerning that a type of Denial of Service attack was taking place. Further, DAML+OIL was used as a reporting language to communicate that fact to other systems. Finally, the ontology (DAML+OIL) and the inference engine were used as an event aggregation language to fuse the existence of the Denial of Service attack, a network connection and session establishment to deduce that a Mitnick-type attack had occurred. Moreover, the only prerequisite for the disparate systems with the distributed coalition is that they share the same ontology.

We are continuing our research, initiating attacks in a controlled environment in order to capture their low-level kernel attributes at the system, process and network levels in order to further specify our ontology.

References

Allen, J, Christie, A, Fithen, W, McHugh, J, Pickel, J and Stoner, E, 2000, ''State of the practice of intrusion detection technologies'' Technical Report 99TR028, Carnegie Mellon Software Engineering Institute.

Amoroso, EG, 1994, *Fundamentals of Computer Security Technology* Prentice-Hall.

Anderson, JP, 1980, ''Computer security, threat monitoring, and surveillance'' Technical report, James P. AndersonCo.

Aslam, T, Krusl, I and Spafford, E, 1996, ''Use of a taxonomy of security faults'' *Proceedings of the 19th National Information Systems Security Conference*.

Boley, D, 1998, ''Principal direction divisive partitioning'' *Data Mining and Knowledge Discovery* **4**(2) 325–344.

Mahalanobis, PC, 1930, ''On tests and measures of groups divergence'' *International Journal of the Asiatic Society of Bengal* NN–NN.

Curry, D and Debar, H, 2003, ''Intrusion detection message exchange format data model and extensible markup language (xml) document type definition'' available at `http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-10.txt`.

Denning, DE, 1987, ''An intrusion detection model'' *IEEE Transactions on Software Engineering* **13**(2) 222–232.

Doyle, J, Kohane, I, Long, W, Shrobe, H and Szolovits, P, 2001, ''Event recognition beyond signature and anomaly'' *2nd IEEE-SMC Information Assurance Workshop*.

Eckmann, S, Vigna, G and Kemmerer, R, 2002, ''STATL: an attack language for state-based intrusion detection'' *Journal of Computer Security* **10**(1/2) 71–104.

Fikes, R and McGuinness, DL, 2001, ''An axiomatic semantics for RDF, RDF-S, and DAML+OIL'' available at `http://www.w3.org/TR/daml+oil-axioms`.

Frank, G, Jenkins, J and Fikes, R, 2003, ''JTP: an object oriented modular reasoning system'' available at `http://kst.stanford.edu/software/jtp`.

Friedman-Hill, EJ, 1977, ''Jess, the Java Expert System Shell'' available at `http://herzberg.ca.sandia.gov/jess/docs/52/`.

Gartner, Inc., 2003, ''Gartner information security hype cycle declares intrusion detection systems a market failure''.

Glass, RL and Vessey, I, 1995, ''Contemporary application-domain taxonomies'' *IEEE Software* 63–76.

Golub, GH and Loan, CFV, 1989, *Matrix Computations* The Johns Hopkins University Press.

Goubault-Larrecq, J, 2001, ''An introduction to LogWeaver (v2.8)'' available at `http://www.lsv.ens-cachan.fr/~goubault/DICO/tutorial.pdf`.

Gruber, TF, 1993, ''A translation approach to portable ontologies'' *Knowledge Acquisition* **5**(2) 199–220.

Guha, B and Mukherjee, B, 1997, ''Network security via reverse engineering of TCP code: vulnerability analysis and proposed solutions'' *IEEE Networks* 40–48.

Haarslev, V and Moller, R, 2001, ''RACER: Renamed ABox and Concept Expression Reasoner'' available at `http://www.cs.concordia.ca/~faculty/haarslev/racer/index.html`, June.

Haines, JW, Rossey, LM, Lippman, RP and Cunningham, RK, 2001, ''Extending the DARPA off-line intrusion detection evaluations'' *DARPA Information Survivability Conference and Exposition II* **1** 77–88.

Hofmeyr, S, Forrest, S and Somayaji, A, 1998, ''Intrusion detection using sequences of system calls'' *Journal of Computer Security* **6** 151–180.

Horrocks, I, Sattler, U and Tobies, S, 2000, ''Reasoning with individuals for the description logic SHIQ'' *Proceedings of the 17th International Conference on Automated Deduction*.

Hendler, J, 2001, ''DARPA agent markup language+ontology interface layer'' available at `http://www.daml.org/2001/03/daml+oil-index`.

Kemmerer, RA and Vigna, G, 2002, ''Intrusion detection: a brief history and overview'' *Security and Privacy: A Supplement to IEEE Computer Magazine* 27–30.

Kendall, K, 1999, ''A database of computer attacks for the evaluation of intrusion detection systems'' Master's thesis, MIT.

Kopena, J, 2002, ''DAMLJessKB'' available at `http://edge.mcs.drexel.edu/ assemblies/software/damljesskb/ articles/DAMLJessKB-2002.pdf`.

Krishnapuram, R, Joshi, A, Nasraoui, O and Yi, L, 2001, ''Low-complexity fuzzy relational clustering algorithms for Web mining'' *IEEE transactions on Fuzzy Systems* **9**.

Krusl, I, 1998, ''Software vulnerability analysis'' Ph.D. thesis, Purdue.

Landwehr, CE, Bull, AR, McDermott, JP and Choi, WS, 1994, ''A taxonomy of computer program security flaws'' *ACM Computing Surveys* **26**(3) 211–254.

Lee, W and Stolfo, SJ, 1998, ''Data mining approaches for intrusion detection'' *Proceedings of the 7th USENIX Security Symposium*.

Lindqvist, U and Jonsson, E, 1997, ''How to systematically classify computer security intrusions'' *Proceedings of the 1997 IEEE Symposium on Security and Privacy* 154–163.

Lindqvist, U and Porras, PA, 1999, ''Detecting computer and network misuse through the production-based system toolset (p-best)'' *Proceedings of the 1999 IEEE Symposium on Security and Privacy* 146–161.

Lippmann, R, Fried, D, Graf, I, Haines, J, Kendall, K, McClung, D, Weber, D, Webster, S, Wyschogrod, D, Cunningham, R and Zissman, M, 2000, ''Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation'' *Proceedings of the DARPA Information Survivability Conference and Exposition, 2000* 12–26.

McHugh, J, 2000, ''Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory'' *ACM Transactions on Information and System Security* **VOL**(iss) NN–NN.

Ning, P, Jajodia, S and Wang, XS, 2001, ''Abstraction-based intrusion in distributed environments'' *ACM Transactions on Information and Systems Security* **4**(4) 407–452.

Raskin, V, Hempelmann, CF, Triezenberg, KE and Nirenburg, S, 2001, ''Ontology in information security: a useful theoretical foundation and methodological tool'' *Proceedings of NSPW-2001* 53–59.

Roger, M and Goubault-Larrecq, J, 2001, ''Log auditing through model checking'' *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)* 220–236.

Simpson, GG, 1961, *Principals of Animal Taxonomy* Columbia University Press.

Staab, S and Maedche, A, 2000, ''Ontology engineering beyond the modeling of concepts and relations'' *Proceedings of the 14th European Conference on Artificial Intelligence*.

Undercoffer, J, Perich, P, Cedilnik, A, Kagal, L and Joshi, A, 2003, ''A secure infrastructure for service discovery and access in pervasive computing'' *Mobile Networks and Applications: Special Issue on Security* **8**(2) 113–126.

Undercoffer, J and Pinkston, J, 2002, ''An empirical analysis of computer attacks and intrusions'' Technical Report TR-CS-03–11, University of Maryland, Baltimore County.

W3C, 2003, ''Extensible Markup Language'' available at http://www.w3c.org/XML/.

W3C, 2001, ''Extended backus-naur form grammer for n triples'', available at `http://www.w3.org/2001/sw/RDFCore/ntriples/`.

Websters, Inc, 1993, *Merriam-Webster's Collegiate Dictionary* Merriam-Webster, Inc.

Wulf, W, 2001, ''Statement before the House Science Committee, U.S. House of Representatives, given 10 October.