Modeling Computer Attacks: An Ontology for Intrusion Detection

Jeffrey Undercoffer, Anupam Joshi, and John Pinkston

University of Maryland, Baltimore County
Department of Computer Science and Electrical Engineering
1000 Hilltop Circle, Baltimore, MD 21250
{undercoffer,joshi,pinkston}@umbc.edu

Abstract. We state the benefits of transitioning from taxonomies to ontologies and ontology specification languages, which are able to simultaneously serve as recognition, reporting and correlation languages. We have produced an ontology specifying a model of computer attack using the DARPA Agent Markup Language+Ontology Inference Layer, a descriptive logic language. The ontology's logic is implemented using DAMLJessKB. We compare and contrast the IETF's IDMEF, an emerging standard that uses XML to define its data model, with a data model constructed using DAML+OIL. In our research we focus on low level kernel attributes at the process, system and network levels, to serve as those taxonomic characteristics. We illustrate the benefits of utilizing an ontology by presenting use case scenarios within a distributed intrusion detection system.

1 Introduction

A central component of an IDS is the taxonomy employed to characterize 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 localized. The inherent problem with this approach is threefold:

i. In order to operate over instances of the data model characterized 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 lack 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, are not communicable between non-homogeneous systems, and their semantics are often vague and lack grounding in any formal logic.

G. Vigna, E. Jonsson, and C. Kruegel (Eds.): RAID 2003, LNCS 2820, pp. 113–135, 2003.

To mitigate the effects of theses problems, we suggest transitioning from taxonomies to ontologies. We construct a data model that characterizes the domain of computer attacks and 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 analyze this information. Gruber [17] defines an ontology as an explicit specification of a conceptualization. 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 (IDS) and IDS sensors.

Ontology representation languages 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) [11], enabling instances of the ontology to be operated over using formal and complete theorem provers.

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)[6], and its ability to enable interoperability between non-homogeneous IDS sensors, Kemmerer and Vigna [25] 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 illustrate the benefits of using ontologies by presenting an implementation of one being utilized by a distributed intrusion detection system. We have constructed our ontology using the Darpa Agent Markup Language + Ontology Inference Layer (DAML+OIL) [22] and have implemented its logic using DAMLJessKB [28], an extension to the Java Expert System Shell [13].

Although our IDS model is not the focus of this paper, we briefly describe it in order to provide context to the reader. Our IDS [23] is a two-phased, host based system. The first phase is an anomaly detector which detects aberrant behavior at the system level. We have instrumented the Linux kernel and gather 190 distinct attributes at the process, system and network levels, several times per second. We use *Principal Component Analysis* (PCA) [15] to reduce the dimensionality of the data set and then use *Fuzzy Clustering* [29] on the reduced data set in order to obtain clusters that model the quiescent state of the system. Once the baseline has been established, we use the Mahalanobis metric [5] as a dissimilarity measure in order to determine if subsequent data samples fall within the bounds of the normative state. The second phase of our IDS *reasons* over the subsequent samples of the feature set that fall outside of the bounds of the normative state, and possibly represent anomalous behavior. 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.

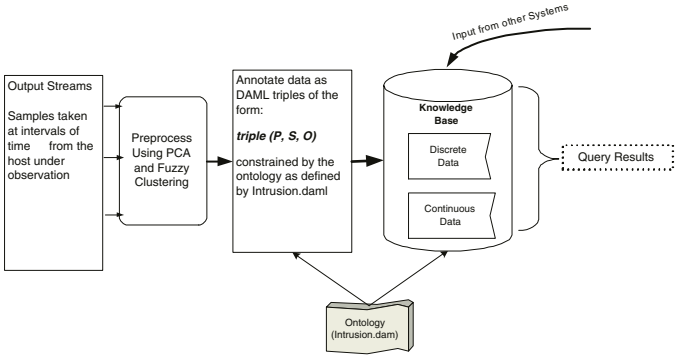

Fig. 1. Distributed IDS Framework

The goal of this work is to demonstrate the utility of ontologies and the overwhelming benefits that may be derived by the IDS research community by transitioning from taxonomies and their linguistic and symbolic representations, to ontologies and ontology representation languages.

The remainder of this paper is organized as follows: Section 2 presents related work in the area of attack taxonomies, attack languages and ontologies for intrusion detection. Section 3 details the motivation for transitioning from taxonomies to ontologies. Our ontology is presented in Section 4. Section 5 details our implementation and Section 6 provides a use case scenario illustrating the utility of using an ontology in detecting instances of a *Denial of Service, Mitnick* and *Buffer Overflow* attacks. We conclude with Section 7.

2 Related Work

There is little, if any, published research formally defining ontologies for use in Intrusion Detection. Raskin et al. [40] introduce and advocate the use of ontologies for information security. In stating the case for using ontologies, they claim that an ontology organizes and systematizes all of the phenomena (intrusive behavior) 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.

2.1 Related Work: Taxonomies

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

Landwehr et al. [31] present a taxonomy categorized according to genesis (how), time of introduction (when) and location (where). They include sub-categories of: *validation errors, boundary condition errors* and *serialization errors*, as a means of effecting an intrusion. We have incorporated these sub-categories into our ontology.

During the 1998 and 1999 DARPA Off Line Intrusion Detection System Evaluations [20,26,35], Weber 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 [33] 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 co-located 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 is comprised of properties and features that are observable by the target of the attack.

Ning et al.[37] 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. [1] and McHugh [36], the taxonomic characterization of intrusive behavior 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 [18] suggests an analysis of each layer of the TCP/IP protocol stack to serve as the foundation for an attack taxonomy. Consequently, we have endeavored to make our ontology as *target centric* as possible.

Aslam et al. [3] 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 is comprised of many of the attributes identified by Aslam et al.

Our intent is to not criticize the use of taxonomies. To 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 [43], taxonomies do not contain the necessary *meta-knowledge* required to convey modeling 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.

2.2 Related Work: Attack Languages

There are several *attack languages* proposed in the literature. These languages are often categorized as Event, Response, Reporting, Correlation, and Recognition Languages [8,9]. We concentrate on correlation, reporting and recognition languages because an ontology representation language is able to simultaneously provide the functionality of all three.

A. P-Best
The P-BEST Toolset [34] (Production-Based Expert System Toolset) 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. P-BEST supports the writing of rules for signature detectors. According to Doyle et al. [8], the P-BEST language lacks concepts that are specific to event recognition and consists solely of a formalism for expressing probabilistic and linguistic rules.

B. STATL
STATL [9] 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. In STATL, scenarios are *attacker centric*. This language provides constructs to represent an attack as a composition of states and transitions. The constructs are similar to those used in programming languages, which describing conditional, sequential and iterative events. STATL lacks constructs for combining sub-events into larger events. Reporting on the efficacy of various attack recognition languages, Doyle et al. [8] state: "STATL constitutes the most clearly defined language for use in attack recognition".

C. LogWeaver
LogWeaver [16] 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 type of events 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* [42]. Essentially, LogWeaver is a specification for a detection language, which defines a syntax and grammar for the end-user to use when writing signatures.

D. CISL
The Common Intrusion Detection Framework (CIDF) [24] started as a DARPA initiative in 1998. CIDF was an effort to develop protocols and application programming interfaces to give IDS research projects the ability to share information and resources and to enable IDS component reuse by multiple systems. The CIDF framework is comprised of components which exchange data in the form of a *GIDO* (generalized intrusion detection object) which are represented in a standard format. This standard format is specified in the Common Intrusion Specification Language (CISL) [10], a reporting language. The CIDF effort appears to have lost inertia, with many of its developers now working on the IETF's IDMEF.

E. BRO
Bro [39] is a real-time, network based IDS that utilizes the specialized "Bro Language", a detection language. The goal of the "Bro Language" is to express security policies in terms of scripts written within that language. In turn, the scripts consist of event handlers that specify what to do whenever a particular event occurs. According to Paxson, the scripts require environment specific tailoring.

F. Snort Rules
SNORT [41] is a network intrusion detection system that performs real time analysis and packet logging on IP networks. SNORT uses a detection language to define rules. The rules are two part: header and options. The header contains the rule's action and addressing information. The options section contains the alert message as well as specifying packet inspection criteria.

G. IDMEF
The Internet Engineering Task Force's proposed Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition [6] 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) [47].
The IDMEF assumes a hierarchal configuration of three IDS components: *sensors, analyzers, and managers*. Sensors are located at the bottom most level of the hierarchy. Sensors output data to analyzers, which in turn report up to a manager, located at the topmost 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.

2.3 XML in Comparison to DAML+OIL

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.

The ontology specification language DAML+OIL, is a descriptive logic language and is grounded in both model-theoretic[1] and axiomatic semantics[2] and has been "cooked" specifically for the Internet. Consequently it is able to:

i. Model the attributes and characteristics of a domain.
ii. Report the existence of an instance of the domain (model) in a manner that is "comprehensible" by any entity that possess the specific ontology.
iii. 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.

The following best explains the inadequacies of XML vis-á-vis DAML+OIL.

Humans are able to combine new facts with existing knowledge to derive new knowledge, computers are not. When a computer acquires new data in XML, it may be able

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

Table 1. Language Feature Comparison: DAML+OIL versus XML

Feature	**Description**	**DAML +OIL**	**XML**
bounded lists	Uses a first/rest structure to represent unordered bounded lists, with nil representing the end of the list.	Yes	No
cardinality constraints	minCardinality and maxCardinality	Yes	Yes
class expressions	Wherever a Class is referenced allows an expression involving *unionOf, disjointUnionOf,intersectionOf* or *complementOf*.	Yes	No
data types	e.g: numerical, temporal and string data types	Yes	Yes
defined classes	Allows new classes to be defined based on property values or other restrictions of an existing class.	Yes	No
enumerations	Allows specification of a restricted set of values for a given attribute to include *oneOf*	Yes	No
equivalence	Supports *equivalentTo* for classes, properties, and instances to support reasoning across ontologies and knowledge bases	Yes	No
extensibility	Allows new properties to used with existing classes.	Yes	No
formal semantics	Semantics have been expressed in both model-theoretic and axiomatic forms.	Yes	No
inheritance	Fully supports *subClassOf* and *subPropertyOf*	Yes	No
inference	Has constructs such as *TransitiveProperty, UnambiguousProperty, inverseOf*, and *disjointWith* for reasoning engines.	Yes	No
local restrictions	Allows restrictions to be associated with a Class/Property pairs.	Yes	No
qualified constraints	Allows expressions such as "all children of X are of type Y".	Yes	No
reification	Provides a standard mechanism for recording data sources, timestamps, etc., without intruding on the data model.	Yes	No

to respond, but only because of some other software which is not part of the XML specification. Although conforming to the XML specification, different systems may very well respond differently, given the same XML encoded data. If a computer acquires new data in DAML+OIL, it can generate entirely new information, solely based on the DAML+OIL standard. Given the same data, any system that conforms to the DAML+OIL specification will generate the same new information and conclusions. A set of DAML+OIL statements, in conjunction with the DAML+OIL specification, enables the conclusion of yet another DAML+OIL statement, whereas a set of XML statements, in conjunction with the XML specification, does not allow the conclusion of any other XML statements. To employ XML to generate new data, knowledge needs to be embedded in some procedural code, which is in stark contrast to DAML+OIL where the knowledge is explicitly stated in DAML+OIL statements.

Although XML supports sub types which are restrictions of extensions on a type, there are no classes. Consequently, there is no notion of inheritance. The following exemplifies the benefits of inheritance. Suppose that you wished to define an event of type X, that is an aggregation of two other events of types Y and Z. Furthermore, suppose that Y and Z are comprised of subclasses Y_1 and Y_2 and Z_1 and Z_2, respectively. If

this information were encoded in XML, we would need application logic that iteratively checked for all possible combinations of Y and Z to satisfy a query. If the same information were to be encoded in DAML+OIL, we would only need to query for the existence of X. Table 1 provides a feature by feature comparison between DAML+OIL and XML.

3 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 of a taxonomy is in order.

3.1 Characteristics of a Sufficient Taxonomy

A *taxonomy* is a *classification* system where the classification scheme conforms to a systematic arrangement into groups or categories according to established criteria [48]. Glass and Vessey [14] 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 [44], 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 [2], Lindqvist et al. [33], Krusl [30] and others have identified what they believe to be the requisite properties of a sufficient and acceptable taxonomy for computer security. Collectively, they have identified the following properties as essential to a taxonomy:

Mutually Exclusive. A classification in one category excludes all others because categories do not overlap.

Exhaustive. The categories, taken together, include all possibilities.

Unambiguous. The category is clear and precise so that classification is not uncertain, regardless of who is classifying.

Repeatable. Repeated applications result in the same classification, regardless of who is classifying.

Accepted. The taxonomy should be logical and intuitive so that it can become generally approved.

Useful. The taxonomy can be used to gain insight into the field of inquiry.

Comprehensible. The taxonomy should be useful to those with less than expert knowledge.

Conforming. The terminology of the taxonomy should comply with established security terminology.

Objectivity. The features must be identified from the object under observation where the attribute being measured should be clearly observable.

Deterministic. There must be a clear procedure that can be followed to extract the feature.

Specific. The value for the feature must be unique and unambiguous.

Upon review of the above list, we believe that, for our purposes, a sufficient and acceptable taxonomy must be: **Mutually Exclusive, Exhaustive, Unambiguous, Useful, Objective, Deterministic, Repeatable** and **Specific**. Hence, these requirements form the underpinnings of our ontology and were selected because they have been identified by the IDS community as essential. We did not adopt the property "Comprehensible" because the requirement that a taxonomic property be comprehensible dictates that those with less than expert knowledge should find the ontology and its taxonomy useful. We felt that this requirement has the potential to oversimplify and relax the structure of the ontology. We did not adopt the property "Accepted", due to the requirement that it be intuitive. The knowledge engineering process employed to build a viable ontology is often more than simple intuition and, at times, appears counter-intuitive.

3.2 Ontologies

According to Davis et al. [7], knowledge representation is a surrogate or substitute for an object under study. In turn, the surrogate enables an entity, such as a software system, to reason about the object. Knowledge representation is also a set of *ontological* commitments specifying the terms that describe the essence of the object. In other words, *meta-data* or data about data describing their relationships.

Frame Based Systems are an important thread in knowledge representation. According to Koller et al. [27], Frame Based Systems provide an excellent representation for the organizational structure of complex domains. Frame Based Languages, which support Frame Based Systems, include RDF [32], and are used to represent ontologies. According to Welty et al. [49], an ontology, at its deepest level, subsumes a taxonomy. Similarly, Noy and McGuinness [38] state that the process of developing an ontology includes arranging classes in a taxonomic hierarchy.

The relationship among data objects may be highly complex; however, at the the finest level of granularity, the *Knowledge Representation* of any object may be represented by an *RDF-S* (Resource Description Framework Schema) statement [4] which formally defines the RDF model as:

i. A set called *Resources*.
ii. A set called *Literals*.
iii. A subset of Resources called *Properties*.
iv. A set called *Statements*, where each element is a triple of the form:{*subject, predicate, object* }. Where *predicate* is a member of Properties, *subject* is a member of Resources, and *object* is either a member of Resources or a member of Literals.

Primarily, RDF-S is about defining class hierarchies (i.e.: taxonomies) and introduces the notions of *Class, Property, Domain* and *Range*. RDF and DAML+OIL extend RDF-S with richer modeling primitives. Figure 2 graphically illustrates the basic RDF-S model, where *(subject, predicate, object)*, which is the same as *(resource, property,*

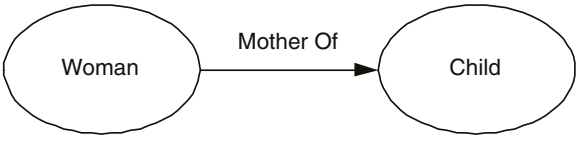

Fig. 2. RDF-S Relationship Graph

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF [
    <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
    <!ENTITY rdfs 'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#'>
    <!ENTITY daml_oil 'http://www.daml.org/2001/03/daml+oil#'>]>

<rdf:RDF
  xmlns:rdf ="&rdf;"
  xmlns:daml_oil ="&daml_oil;"
  xmlns:rdfs ="&rdfs;">

<daml_oil:ObjectProperty rdf:ID="Mother_Of">
  <daml_oil:range rdf:resource="#Child"/>
  <daml_oil:domain rdf:resource="#Woman"/>
</daml_oil:ObjectProperty>

<daml_oil:Class rdf:ID="Woman">
  <rdfs:subClassOf rdf:resource="&daml_oil;#Thing"/>
</daml_oil:Class>

<daml_oil:Class rdf:ID="Child">
  <rdfs:subClassOf rdf:resource="&daml_oil;#Thing"/>
</daml_oil:Class>

</rdf:RDF>
```

Fig. 3. DAML+OIL Specification for the Mother Child Relationship

resource[or literal value]), is illustrated by the *(Woman, Mother Of, Child)* relationship, where *Mother* is the subject, *Child* is the object and *Mother Of* is the predicate. Figure 3 illustrates the Mother Child relationship specified in DAML+OIL. It should be noted that a set of*N-triples*, an RDF-S graph, and a DAML+OIL specification are equivalent if they each describe the same ontology.

In applying ontologies to the problem of intrusion detection, the power and utility of the ontology is not realized by the simple taxonomic representation of the attributes of the attack. Instead, **the power and utility of the ontology is realized by the fact that we can express the relationships between collected data and use those relationships to deduce that the particular data represents an attack of a particular type**. Because ontologies provide powerful constructs that include machine interpretable definitions of the concepts within a specific domain and the relations between them, they may be utilized not only to provide an IDS with the ability to share a common understanding of the information at issue, but also to further enable the IDS, with an improved capacity, to reason over and analyze instances of data representing an intrusion.

Moreover, specifying an ontological representation decouples the data model from the logic of the intrusion detection system. The decoupling of the data model enables non-homogeneous 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 statements. Non-homogeneous IDSs do not need to run the same type of software and the sensors of a distributed IDS may monitor different aspects of an enterprise. A shared ontology enables these disparate components to operate as a coalition, sharing, correlating and aggregating each other's data.

4 Our IDS Ontology: Attributes of the Class Intrusion

In constructing our ontology, we conducted an empirical analysis [46] of the features and attributes, and their interrelationships, of over 4,000 classes of computer attacks and intrusions 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.

Figure 4 presents a high level view of our ontology. The 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 (i.e.: address from which system calls are made, total virtual memory size, etc) 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.

At the top most level of Figure 4 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 is comprised of 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.

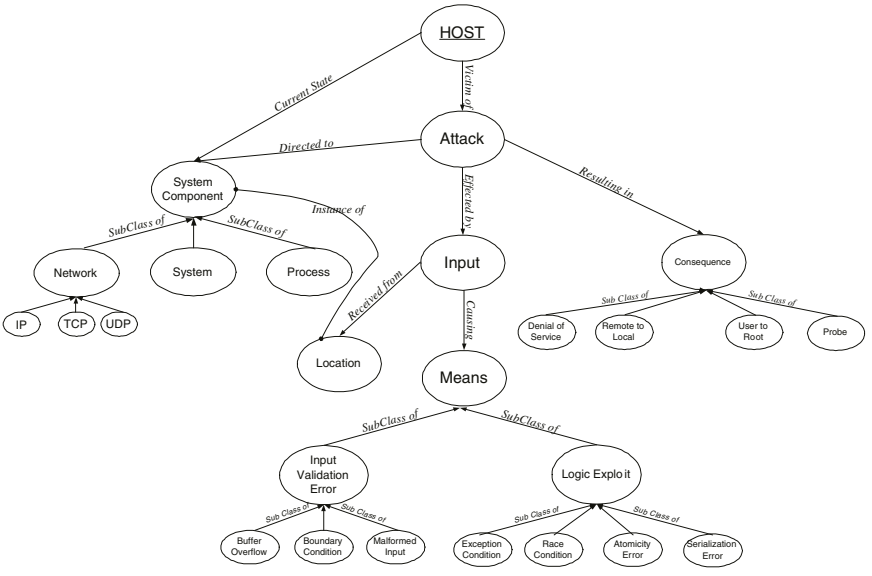

Fig. 4. Our IDS Ontology

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 is comprised of 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, 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) Serialization Error. An error that results from the improper serialization 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.

As previously stated, the properties of Mutual Exclusion, Exhaustive, Non-ambiguity, Usefulness, Objectivity, Determinism, Repeatability and Specificity are the overarching requirements that determine the taxonomic characteristics of our ontology. We believe that we have met these requirements predicated upon the following:

i. Mutual Exclusion. Each class in the ontology is disjoint from the other classes because none share an identical set of properties.
ii. Exhaustive. Our analysis of the available data indicates that computer attacks and intrusions are effected by some input, that is directed to some system component, causing some heretofore unintended system response (means), and results in some adverse system consequence. Our ontology captures these notions.
iii. Non-ambiguity. Each class in the ontology has a definite set of properties and restrictions.
iv. Usefulness. As will be exemplified in Section 5, Implementation, our ontology enables the conclusion (entailment) of new knowledge from seemingly disassociated facts.

v. Objectivity. The properties of the classes of our ontology are directly derivable from 190 distinct system features. This feature set characterizes system state at any particular time.
vi. Deterministic. The properties of each class obtainable from metrics associated with the Linux kernel.
vii. Repeatability. An instantiated object within our ontology will always be evaluated to the identical conclusion. Moreover, the same object will be evaluated to the same conclusions by any entity using the ontology.
viii. Specific. The property values for classes that define aberrant system behavior are unique and are limited to a set of 190 attributes.

5 Implementation

There are several *reasoning systems* that are compatible with DAML+OIL[12,19,28,21], which according to their functionality, may be classified as 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* [28] reasoning system. DAMLJessKB is employed to reason over instances of our data model that are considered to be suspicious. These suspicious instances are constrained according to our ontology and asserted into the knowledge base.

Upon initialization of DAMLJessKB, we parse the DAML+OIL statements representing the ontology, converting them into *N-Triples*, 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 all of the chains of implication derived from the ontology.

As will be illustrated shortly, additional information in the form of instances of the ontology is asserted into the knowledge base as facts.

5.1 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 ontology. Instances are asserted and de-asserted 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 granular that it requests an attack of a specific type, such as a Syn Flood:

```
(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 a medium level of 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, subclasses, and the relationships that hold between them. The query variable *?var*, which 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.

6 Using the Ontology to Detect Attacks: Use Case Scenarios

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

6.1 Denial of Service – Syn Flood

The DAML+OIL representation of an instance of a *Syn_Flood* attack is illustrated in Figure 5. 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.

```
<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"/>
```

Fig. 5. DAML+OIL Notation for an Instance of a Syn Flood Attack

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.

6.2 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.

In the following example, which is illustrated in figure 6, **Host B** is the ultimate target and **Host A** is trusted by **Host B**.

The attack is structured as follows:

i. The attacker initiates a Syn/Flood attack against **Host A** to prevent **Host A** from responding to **Host B**.
ii. 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**.

iii. 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**.
iv. **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 number of half open connections caused by the Syn/Flood attack, cannot send a *RST* message to **Host B** in response to the spurious Syn message.
v. 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**.
vi. **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.

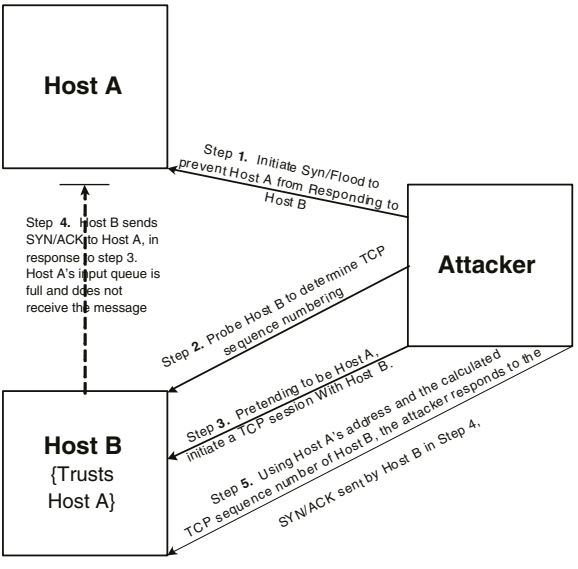

Fig. 6. Illustration of the Mitnick Attack

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 utilize a secure communications infrastructure that has been optimized for IDSs. We present such a communications infrastructure in [45].

Consider the case of the instance of the Syn Flood attack presented in Section 6.1, and that it was directed against **Host A** in our example scenario. Since the IDS responsible for

Host A is continually monitoring for anomalous behavior, asserting and de-asserting data as necessary, it detects the occurrence of an inordinate number of partially established TCP connections, and transmits the instance of the Syn Flood illustrated in Figure 5 to the other IDSs in its coalition.

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 de-asserted 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 7 lists the set of DAML+OIL statements describing those connections that were used in our experiments:

```
<IntrOnt:Connection rdf:about="&IntrOnt;00043"
         IntrOnt:IP_Address="130.85.112.231"
         IntrOnt:conn_time="15:42:59"
         IntrOnt:conn_date="02/22/2003"
         rdfs:label="00041"/>

<IntrOnt:Connection rdf:about="&IntrOnt;00043"
         IntrOnt:IP_Address="130.85.112.231"
         IntrOnt:conn_time="15:44:17"
         IntrOnt:conn_date="02/22/2003"
         rdfs:label="00043"/>
```

Fig. 7. DAML+OIL Notation for an Instances of Connections

Figure 8 illustrates the DAML+OIL notation specifying the Mitnick attack. Notice that it is a subclass of both the class defining a Denial of Service attack and the TCP subclass, with a restriction on the property indicating that the target of the attack has established a connection with the victim 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)))
```

```
<daml:Class rdf:about="&Intrusion;Mitnick"
   rdfs:label="P\_Mitnick">
    <rdfs:subClassOf>
      <daml:Restriction>
        <daml:onProperty rdf:resource=
              "&IntrOnt;Victim"/>
        <daml:hasValue rdf:resource="#true"/>
        <daml:toClass rdf:resource=
              "&IntrOnt;DoS"/>
      </daml:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <daml:Restriction>
        <daml:onProperty rdf:resource=
              "&IntrOnt;est_connections"/>
          <daml:hasValue rdf:resource=
              "#IP_Address"/>
          <daml:toClass rdf:resource=
              "&IntrOnt;TCP"/>
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>
```

Fig. 8. DAML+OIL Specification of the Mitnick Attack

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 de-assertions, 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.

6.3 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 of the boundaries of the Text segment. Figure 9 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>
```

Fig. 9. 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))
```

7 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. Secondly, 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

1. J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner. State of the Practice of Intrusion Detection Technologies. Technical Report 99tr028, Carnegie Mellon - Software Engineering Institute, 2000.
2. E. G. Amoroso. *Fundamentals of Computer Security Technology*. Prentice-Hall PTR, 1994.
3. T. Aslam, I. Krusl, and E. Spafford. Use of a Taxonomy of Security Faults. In *Proceedings of the 19th National Information Systems Security Conference*, October 1996.
4. D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3c.org/TR/rdf-schema/, 2003.
5. P. C.Mahalanobis. *On Tests and Meassures of Groups Divergence*. International Journal of the Asiatic Society of Bengal, 1930.
6. D. Curry and H. Debar. "intrusion detection message exchange format data model and extensible markup language (xml) document type definition. http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-10.txt, January 2003.
7. R. Davis, H. Shrobe, and P. Szolovits. What is Knowledge Representation? *AI Magazine*, 14(1):17 – 33, 1993.

8. J. Doyle, I. Kohane, W. Long, H. Shrobe, and P. Szolovits. Event Recognition Beyond Signature and Anomaly. In *2nd IEEE-SMC Information Assurance Workshop*, June 2001.
9. S. Eckmann, G. Vigna, and R. Kemmerer. STATL: An Attack Language for State-based Intrusion Detection. *Journal of Computer Security*, 10(1/2):71 – 104, 2002.
10. R. Feiertag, C. Kahn, P. Porras, D. Schackenberg, S. Staniford-Chen, and B. Tung. A Common Intrusion Specification Language. http://www.isi.edu/ brian/cidf/drafts/language.txt, June 1999.
11. R. Fikes and D. L. McGuinness. An Axiomatic Semantics for RDF, RDF-S, and DAML+OIL. http://www.w3.org/TR/daml+oil-axioms, December 2001.
12. G. Frank, J. Jenkins, and R. Fikes. JTP: An Object Oriented Modular Reasoning System. http://kst.stanford.edu/software/jtp.
13. E. J. Friedman-Hill. Jess, The Java Expert System Shell. http://herzberg.ca.sandia.gov/jess/docs/52/, November 1977.
14. R. L. Glass and I. Vessey. Contemporary Application-Domain Taxonomies. *IEEE Software*, pages 63 – 76, July 1995.
15. G. Golub and C. Loan. *Matrix Computations*. The Johns Hopkins University Press, 1989.
16. J. Goubault-Larrecq. An Introduction to LogWeaver (v2.8). http://www.lsv.ens-cachan.fr/ goubault/DICO/tutorial.pdf, September 2001.
17. T. F. Gruber. A Translation Approach to Portable Ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.
18. B. Guha and B. Mukherjee. Network Security via Reverse Engineering of TCP Code: Vulnerability Analysis and Proposed Solutions. In *IEEE Networks*, pages 40 – 48. IEEE, July/August 1997.
19. V. Haarslev and R. Moller. RACER: Renamed ABox and Concept Expression Reasoner. http://www.cs.concordia.ca/ faculty/haarslev/racer/index.html, June 2001.
20. J. W. Haines, L. M. Rossey, R. P. Lippman, and R. K. Cunningham. Extending the DARPA Off-Line Intrusion Detection Evaluations. In *DARPA Information Survivability Conference and Exposition II*, volume 1, pages 77 – 88. IEEE, 2001.
21. I. Horrocks, U. Sattler, and S. Tobies. Reasoning with Individuals for the Description Logic SHIQ. In *Proceedings of the 17th International Conference on Automated Deduction*, number 1831. Springer-Verlag, 2000.
22. J. Hendler. DARPA Agent Markup Language+Ontology Interface Layer. http://www.daml.org/2001/03/daml+oil-index, 2001.
23. A. Joshi and J. Undercoffer. On web semantics and data mining: Intrusion detection as a case study. In *Proceedings of the National Science Foundation Workshop on Next Generation Data Mining*, 2002.
24. C. Kahn, D. Bolinger, and D. Schackenberg. Communication in the Common Intrusion Detection Framework v 0.7. http://www.isi.edu/ brian/cidf/drafts/communication.txt, June 1998.
25. R. A. Kemmerer and G. Vigna. Intrusion Detection: A Brief History and Overview. *Security and Privacy a Supplement to IEEE Computer Magazine*, pages 27 – 30, April 2002.
26. K. Kendall. A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems. Master's thesis, MIT, 1999.
27. D. Koller and A. Pfeffer. Probabilistic Frame-Based Systems. In *Proceedings of the Fifteenth National Conference on Artifical Intelligence*, pages 580 – 587, Madison, Wisconsin, July 1998. AAAI.
28. J. Kopena. DAMLJessKB. http://edge.mcs.drexel.edu/ assemblies/software/damljesskb/articles/DAMLJessKB-2002.pdf, October 2002.
29. R. Krishnapuram, A. Joshi, O. Nasraoui, and L. Yi. Low-Complexity Fuzzy Relational Clustering Algorithms for Web Mining. In *IEEE transactions on Fuzzy Systems*, volume 9, August 2001.

30. I. Krusl. *Software Vulnerability Analysis.* PhD thesis, Purdue, 1998.
31. C. E. Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi. A Taxonomy of Computer Program Security Flaws. *ACM Computing Surveys*, 26(3):211 – 254, September 1994.
32. O. Lassila and R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, February 1999.
33. U. Lindqvist and E. Jonsson. How to Systematically Classify Computer Security Intrusions. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 154 – 163, May 1997.
34. U. Lindqvist and P. A. Porras. Detecting computer and network misuse through the production-based system toolset (p-best). In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 146 – 161. IEEE, May 1999.
35. R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman. Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition,2000*, pages 12 – 26.
36. J. McHugh. 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*, November 2000.
37. P. Ning, S. Jajodia, and X. S. Wang. Abstraction-Based Intrusion in Distributed Environments. *ACM Transactions on Information and Systems Security*, 4(4):407 – 452, November 2001.
38. N. F. Noy and D. L. McGuinnes. Ontology development 101: A guide to creating your fisrt ontology. Stanford University.
39. V. Paxson. Bro: A system for Detecting Network Intruders in Real Time. In *Proceedings of the 7th Symposium on USENIX Security*, 1998.
40. V. Raskin, C. F. Hempelmann, K. E. Triezenberg, and S. Nirenburg. Ontology in Information Security: A Useful Theoretical Foundation and Methodological Tool. In *Proceedings of NSPW-2001*, pages 53 – 59. ACM.
41. M. Roesch. Snort, version 1.8.3. availble via www.snort.org, August 2001. an open source NIDS.
42. M. Roger and J. Goubault-Larrecq. Log Auditing through Model Checking. In *Proceedings of 14th the IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 220 – 236, 2001.
43. S. Staab and A. Maedche. Ontology Engineering Beyond the Modeling of Concepts and Relations. In *Proceedings of the 14th European Congress on Artificial Intelligence*, 2000.
44. G. G. Sumpson. *Principals of Animal Taxonomy*. Columbia University Press, 1961.
45. J. Undercoffer, F. Perich, A. Cedilnik, L. Kagal, and A. Joshi. A Secure Infrastructure for Service Discovery and Access in Pervasive Computing. *Mobile Networks and Applications: Special Issue on Security*, 8(2):113 – 126, 2003.
46. J. Undercoffer and J. Pinkston. An Empirical Analysis of Computer Attacks and Intrusions. Technical Report TR-CS-03-11, University of Maryland, Baltimore County, 2002.
47. W3C. Extensible Markup Language. http://www.w3c.org/XML/, 2003.
48. WEBSTERS, editor. *Merriam-Webster's Collegiate Dictionary*. Merriam-Webster, Inc., tenth edition, 1993.
49. C. Welty. Towards a Semantics for the Web. www.cs.vassar.edu/faculty/welty/papers/dagstuhl-2000.pdf, 2000.