A Target-Centric Ontology for Intrusion Detection

John Pinkston, Jeffrey Undercoffer, Anupam Joshi and Timothy Finin
University of Maryland, Baltimore County
Department of Computer Science and Electrical Engineering
1000 Hilltop Circle, Baltimore, MD 21250
{pinkston, undercoffer, joshi, finin}@umbc.edu

Abstract

We have produced an ontology specifying a model of computer attacks. Our ontology is based upon an analysis of over 4,000 classes of computer intrusions and their corresponding attack strategies and is categorized 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 favor of ontologies, for the classification of computer attacks and intrusions. We have specified our ontology using DAML+OIL and have prototyped it using DAMLJessKB. We present our model as a target-centric ontology and illustrate the benefits of utilizing an ontology in lieu of a taxonomy, by presenting a use case scenario of a distributed intrusion detection system.

1 Introduction

Based upon empirical evidence we have produced a model of computer attacks categorized by: the system component targeted, the means and consequence of attack, and the location of the attacker. Our model is represented as a *target-centric* ontology, where the structural properties of the classification scheme is in terms of features that are observable and measurable by the target of the attack or some software system acting on the target's behalf. In turn, this ontology is used to facilitate the reasoning process of detecting and mitigating computer intrusions.

Traditionally, the characterization and classification of computer attacks and other intrusive behaviors have been limited to taxonomies. Taxonomies, however, lack the necessary and essential constructs needed to enable an intrusion detection system (IDS) to reason over an instance that is representative of the domain of a computer attack. Alternatively, ontologies 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 the software system with a greater ability to reason over and analyze this information.

As detailed by Allen, et al. [2], and McHugh [22], 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 [10] suggests an analysis of each layer of the TCP/IP protocol stack to serve as the foundation for an attack taxonomy.

As an alternative to a taxonomy, we propose a data model implemented with an ontology representation language such as the Resource Description Framework Schema (RDFS) [26] or the DARPA Agent Markup Language + Ontology Inference Layer (DAML+OIL) [1]. We illustrate the benefits of using ontologies by presenting an implementation of our ontology being utilized by a distributed intrusion detection system. Accordingly, we have specified our target-centric ontology in DAML+OIL and have implemented it using DAMLJessKB [17], an extension to the Java Expert System Shell [7].

Because IDS's 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. Consequently, our ontology 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 analyzed over 4,000 computer vulnerabilities and the corresponding attack strategies employed to exploit them.

The remainder of this paper is organized as follows: Section 2 presents related work in the form of alternative attack taxonomies as well as presenting related work in the area of ontologies for intrusion detection. Section 3 presents the characteristics of a sufficient taxonomy. Section 4 details the motivation for abandoning taxonomies in favor of ontologies. Our target-centric attack taxonomy is presented in Section 5. Section 6 details our implementation and Section 7 provides an example scenario illustrating the utility of the ontology within a distributed intrusion detection system. We conclude with Section 8.

2 Related Work

As previously stated, most of the existing research in the area of the classification of computer attacks is limited to taxonomies.

Because a taxonomy is contained within an ontology we address the research in the area of defining intrusion taxonomies before we address ontologies. Accordingly, this section is subdivided, with Subsection 2.1 presenting related work in the area of taxonomies for intrusion detection and Subsection 2.2 presenting related work in the area of ontologies for intrusion detection.

2.1 Related Work: Taxonomies

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

In [19] Landwehr et al., 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, which we incorporate into our ontology as the means of an attack.

During the 1998 and 1999 DARPA Off Line Intrusion Detection System Evaluations [12] [21] [15] Weber provided a taxonomy defining the categories of consequence, to include *Denial of Service*, *Remote to Local* and *User to Root*, which we incorporate into our work.

Lindqvist and Jonsson [20] 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.

2.2 Related Work: Ontologies

There is little, if any, published research formally defining ontologies for use in Intrusion Detection.

Raskin et al. [25], introduce and advocate the use of ontologies for information security. In arguing the case for using ontologies, they state 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.

In commenting on the IETF's IDMEF, Kemmerer and Vigna [14] state *"it is a but a first step, however additional effort is needed to provide a common ontology that lets IDS sensors agree on what they observe"*.

3 Characteristics of a Sufficient Taxonomy

At this point, a clear understanding of the definition, purpose and objective of a taxonomy is in order. Accordingly, a *taxonomy* is a *classification* system where the classification scheme conforms to a systematic arrangement into groups or categories according to established criteria [31]. Glass and Vessey [9] contend that taxonomies provide a set of unifying constructs so that the area of interest can be systemically 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 [27] 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 [3], Lindqvist, et al. [20] and Krusl [18] each 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, Exhaustive, Unambiguous, Repeatable, Accepted, Useful, Comprehensible, Conforming, Objective, Deterministic and Specific. Accordingly, as an ontology subsumes a taxonomy these characteristics form the underpinnings of our work.

4 From Taxonomies to Ontologies: *The case for ontologies in Intrusion Detection*

Ning et al. [23], 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. Consequently, this model is not readily interchangeable and reusable by other systems.

The Intrusion Detection Working Group of Internet Engineering Task Force (IETF) has proposed the Intrusion Detection Message Exchange Requirements [33] which, in addition to defining the requirements for the Intrusion Detection Message Exchange Format, also specifies the architecture of an IDS. The Intrusion Detection Message Exchange Format Data Model (IDMEF) and accompanying Extensible Markup Language Document Type Definition [4] is a profound effort to establish an industry wide data model which defines computer intrusions. The IDMEF, however, has its shortcomings. Specifically, it uses XML which is limited to a syntactic representation of the data model and does not convey the semantics, relationships, attributes and characteristics of the objects which it represents.. This limitation requires that each individual IDS interpret and implement the data model programmaticaly.

According to Davis et al. [5], 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. [16], Frame Based Systems provide an excellent representation for the organizational structure of complex domains. Frame Based Languages, which support Frame Based Systems, include RDF, and are used to represent ontologies. According to Welty et al. [32], an ontology, at its deepest level, subsumes a taxonomy. Similarly, Noy and McGuinness [24] state the process of developing an ontology includes arranging classes in a taxonomic hierarchy.

In applying ontologies to the problem of intrusion detection, the power and utility of the ontology is not realized by the simple 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**. 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 non-homogeneous IDS's 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 IDS's in the form of a set of DAML+OIL (or RDF) statements. If the re-

cipient 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 utilized 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 analyze instances of data representing an intrusion. Moreover, within an ontology, characteristics such as cardinality, range and exclusion may be specified and the notion of inheritance is supported.

5 Target-Centric Ontology *Attributes of the Class Intrusion*

In constructing our ontology, we relied upon an empirical analysis [30] of the features and attributes, and their interrelationships, of over 4,000 classes of computer attacks and intrusions. Figure 1, presents a high level view of our ontology. The attributes of each class and subclass (denoted by ellipses) are not shown because it would make the illustration unwieldy.

At the top most level we define the class *Host*. Host has the properties *Current State* which is defined by the class *System Component* and *Victim of* which is defined by the class *Attack*. As defined in Section 4 the property, also called the predicate, defines the relationship between a subject and an object.

The System Component class is comprised of the following subclasses:

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

2. 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.

3. 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:

1. 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.
2. User Access. The attack results in the attacker having access to services on the target system at an unprivileged level.
3. Root Access. The attack results in the attacker being granted privileged access to the system, consequently having complete control of the system.
4. 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 the attributes *Received from* and *Causing* where *Causing* defines the relationship between the *Means* of attack and some input. We define the following subclasses for *Means* of attack:

1. 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.
2. 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.

6 Implementation

There are several *reasoning systems* that are compatible with DAML+OIL. According to their functionality, reasoning systems can be classified into two types, backward-chaining

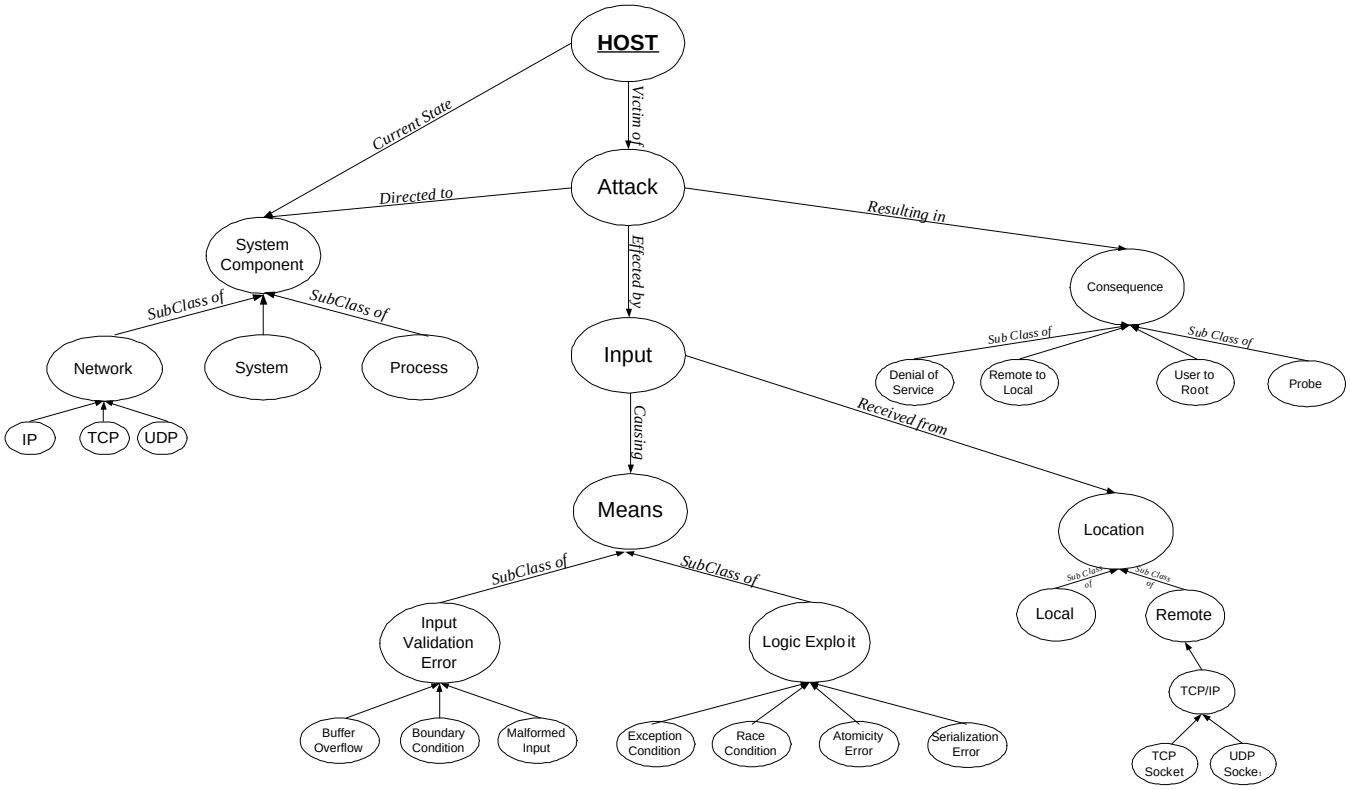

Figure 1: Target Centric Ontology

and 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. Available reasoning systems include: Stanford's *Java Theorem Prover* [6], Drexel's *DAML-JessKB* [17] and the *Renamed ABox and Concept Expression Reasoner* [11].

We have prototyped the logic portion of our system using the *DAMLJessKB* [17] reasoning system, an extension to the *Java Expert System Shell* (JESS) [7]. JESS is a Java implementation of the *C Language Integrated Production System* (CLIPS) [8]. 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 target-centric ontology and asserted into the knowledge base.

Upon initialization of DAMLJessKB we converted the DAML+OIL statements representing the ontology 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.

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

Figure 2 lists the DAML+OIL statements defining the class *Attack* and it properties *Directed To*, *Resulting In* and *Effected By*. These properties correspond to the edges between the node labeled *Target* and the nodes labeled *System Component*, *Input* and *Consequence* respectively, in Figure 1.

```
<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 2: DAML+OIL Statements Defining the Class Attack and its Properties: Directed To, Resulting In and Effected By

Figure 3 presents the DAML+OIL notation for the class *System Component*, its subclass *Network*, and Network's subclass *TCP*. Figure 4 lists the DAML+OIL notation for some of the attributes of the class *TCP*.

Figure 5 details the specification of the class *Consequence* while Figures 6 and 7 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 1 illustrating our ontology, is a subclass of both Denial of Service and TCP and, as stated

```
<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 3: DAML+OIL Statements Specifying the Class System Component and its Subclass, 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 4: 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 5: DAML+OIL Specification of the Class Consequence

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

Figure 6: DAML+OIL Statements Specifying the Denial of Service Subclass

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

```
<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 7: DAML+OIL Statements Specifying the SynFlood Subclass

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. 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 holds between them. The variable *?var*, 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 a Distributed Attack

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 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 8, **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 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.
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 where it believes 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.

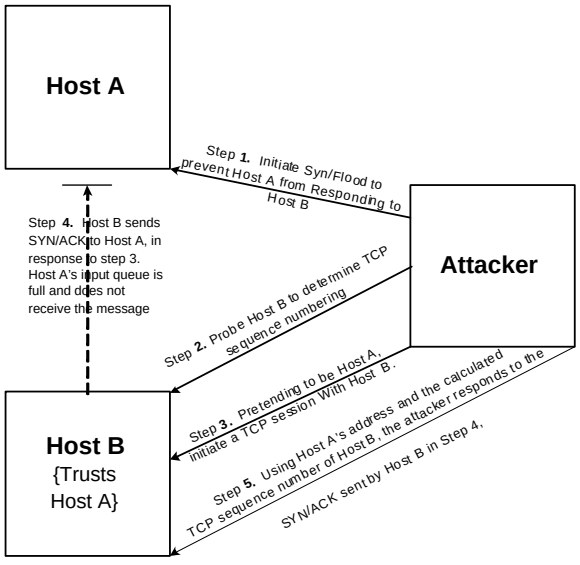

Figure 8: Illustration of the Mitnick Attack

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 explains the utility of our ontology, as well as the importance of forming coalitions of IDSs. In our IDS model, we form coalitions of IDS services each of which is responsible for specific parts of an enterprise or domain. For example, one IDS service may be responsible for a specific host, while another is responsible for a group of hosts, while yet still another is responsible for monitoring network traffic. The IDS's all share a common ontology and utilize a secure communications infrastructure that has been optimized for IDS's. We present such a infrastructure in [13, 28, 29].

Consider the case of the instance of the Syn Flood attack presented in Section 6 and that it was directed against **Host A** in our example scenario. As the IDS responsible for **Host A** is continually monitoring for anomalous behavior, asserting and de-asserting data as necessary, it will detect the occurrence of an inordinate number of partially established TCP connections, and transmit the instance of the Syn Flood to the other IDS's in its coalition.

That instance is repeated below:

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

<IntrOnt:Syn_Flood rdf:about="&IntrOnt;00038"
     IntrOnt:Exceed_T="true"
     IntrOnt:int_time="20021212 154312"/>
```

This instance is converted into a set of *N-Triples* and asserted into the knowledge base of each IDS in the coalition. 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. This situation, especially in conjunction with **Host B** being subjected to a series of probes meant to determine its TCP sequencing, could be the prelude to a distributed attack the current connections and pending connections are also asserted into the knowledge base.

The following is a set DAML+OIL statements describing connections:

```
<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"/>
```

In order to detect an Mitnick type attack, we include the following DAML+OIL statements that partially specify an ontology of the Mitnick attack (the class is identified as P_Mitnick for partial):

```
<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>
```

The ontology is partial because the Mitnick attack has the additional property that the connection time with the victim must be greater than or equal to the time of the denial of service attack. An instance of this ontology will be instantiated provided that there exists an instance of a denial of service attack that has the same unique identifier as that of an established connection. In fact there will be an instance created in each case where this condition holds. In our prototype, we check each instance to determine if the time of the connection is greater than or equal the time of the attack.

The following rules are used to check each instance:

```
(defrule isMitnick

(PropertyValue
(p http://security.umbc.edu/IntrOnt#P\_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 rule will fire and event number 00038, the instance of the intersection of the connection and the denial of service attack, will be displayed.

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 the *Exced_T* property of the *Network* class was 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 IDS's in the coalition. In turn, these IDS's converted the DAML+OIL statements to 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.

The ontology specifying the Mitnick class states that it is the intersection of both the *DoS* and *Connection* classes. Because each IDS instantiates an instance when this constraints imposed by intersection is true, we need to examine each instance to ensure that $Connection\ Time \geq Intrusion\ Time$.

8 Conclusion and Future Work

We have analyzed vulnerability and intrusion data derived from CERT advisories and NIST's ICAT meta-base resulting in the identification of the components (network, kernel, application and other) most frequently attacked. We have also identified the most common means and consequences of the attack as well as the location of the attacker. Our analysis shows that non-kernel space (non operating system) applications, running as either root or user, are the most frequently attacked and are attacked remotely. The most common means of attack are exploits. According to the CERT advisories issued in response to severe vulnerabilities, *root* access is the most common consequence of an exploit whereas the ICAT data shows *denial of service* to be the most common consequence.

Our analysis was conducted in order to identify the observable and measurable properties of computer attacks and intrusions. Accordingly, we have developed a target-centric ontology characterized by *System Component, Means of Attack, Consequences of Attack* and *Location of Attacker*. We have stated the case for replacing simple taxonomies with ontologies for use in IDS's and have presented an initial ontology specifying the class *Intrusion*. Our ontology is available at: http://security.cs.umbc.edu/Intrusion.

We have prototyped our ontology using the DAMLJessKB, which has some limitations. We intend to either modify DAMLJessKB in order to make it a full and complete reasoner or use Stanford's *Java Theorem Prover* [6] or *Rename ABox and Concept Expression Reasoner* [11].

References

[1] DARPA Agent Markup Language+Ontology Interface Layer. http://www.daml.org/2001/03/daml+oil-index, 2001.

[2] Julia Allen, Alan Christie, William Fithen, John McHugh, Jed Pickel, and Ed Stoner. State of the Practice of Intrusion Detection Technologies. Technical Report 99tr028, Carnegie Mellon - Software Engineering Institute, 2000.

[3] Edward G. Amoroso. *Fundamentals of Computer Security Technology*. Prentice-Hall PTR, 1994.

[4] D. Curry and H. Debar. Intrusion detection message exchange format data model and extensible markup language (xml)document type definition. draft-ietf-idwg-idmef-xml-07.txt, January 2003. expires July 31, 2003.

[5] Randall Davis, Howard Shrobe, and Peter Szolovits. What is knowledge representation? *AI Magazine*, 14(1):17 – 33, 1993.

[6] Gleb Frank, Jessica Jenkins, and Richard Fikes. Jtp: An object oriented modular reasoning system. http://kst.stanford.edu/software/jtp.

[7] Ernest J. Friedman-Hill. Jess, the java expert system shell. http://herzberg.ca.sandia.gov/jess/docs/52/, November 1977.

[8] Joseph Giarratano and Gary Riley. *Expert Systems Principles and Programming*. PWS Publishing Company, third edition, 1998.

[9] Robert L. Glass and Iris Vessey. Contemporary application-domain taxonomies. *IEEE Software*, pages 63 – 76, July 1995.

[10] Biswaroop Guha and Biswanath Mukherjee. Network Security via Reverse Engineering of TCP Code: Vulnerability Analysis and Proposed Solutions. In *IEEE Networks*, pages 40 – 48. IEEE, July/August 1997.

[11] Volker Haarslev and Ralf Moller. RACER: Renamed ABox and Concept Expression Reasoner. http://www.cs.concordia.ca/ faculty/haarslev/racer/index.html, June 2001.

[12] Joshua W. Haines, Lee M. Rossey, Richard P. Lippman, and Robert 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.

[13] Lalana Kagal, Jeffrey Undercoffer, Anupam Joshi, and Tim Finin. Vigil: Enforcing Security in Ubiquitous Environments . In *Grace Hooper Celebration of Women in Computing 2002*, 2002.

[14] Richard A. Kemmerer and Giovanni Vigna. Intrusion detection: A brief history and overview. *Security and Privacy a Supplement to IEEE Computer Magazine*, pages 27 – 30, April 2002.

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

[16] Daphne Koller and Avi Pfeffer. Probabilistic Frame-Based Systems. In *Proceedings of the Fifteenth National Conference on Artifical Intelligence*, pages 580 – 587, Madison, Wisconsin, July 1998. AAAI.

[17] Joe Kopena. DAMLJessKB. http://edge.mcs.drexel.edu/assemblies/software/damljesskb/ articles/DAMLJessKB-2002.pdf, October 2002.

[18] Ivan Krusl. *Software Vulnerability Analysis.* PhD thesis, Purdue, 1998.

[19] Carl E. Landwehr, Alan R. Bull, John P. McDermott, and William S. Choi. A taxonomy of computer program security flaws. *ACM Computing Surveys*, 26(3):211 – 254, September 1994.

[20] Ulf Lindqvist and Erland Jonsson. How to systematically classify computer security intrusions. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 154 – 163. IEEE, May 1997.

[21] Richard Lippmann, David Fried, Isaac Graf, Joshua Haines, Kristopher Kendall, Davind McClung, Dan Weber, Seth Webster, Dan Wyschogrod, Rober Cunningham, and Marc 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, January 2000.

[22] John 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.

[23] Peng Ning, Sushil Jajodia, and Xiaoyang Sean Wang. Abstraction-based intrusion in distributed environments. *ACM Transactions on Information and Systems Security*, 4(4):407 – 452, November 2001.

[24] Natalya F. Noy and Deborah L. McGuinnes. Ontology development 101: A guide to creating your fisrt ontology. Stanford University.

[25] Victor Raskin, Christian F. Hempelmann, Katrina E. Triezenberg, and Sergei Nirenburg. Ontology in information security: A useful theoretical foundation and methodological tool. In *Proceedings of NSPW-2001*, pages 53 – 59. ACM, ACM, September 2001.

[26] RDF. Resource description framework (rdf) schema specification, 1999.

[27] George Gaylord Sumpson. *Principals of Animal Taxonomy*. Columbia University Press, 1961.

[28] Jeffrey Undercoffer, Filip Perich, Andrej Cedilnik, Lalana Kagal, and Anupam Joshi. A Secure Infrastructure for Service Discovery and Access in Pervasive Computing. *Mobile Networks and Applications: Special Issue on Security*, (2):113 – 126, 2003.

[29] Jeffrey Undercoffer, Filip Perich, and Charles Nicholas. Shomar: An architecure for distributed intrusion detection services. University of Maryland Baltimore County, Department of Computer Science and Electrical Engineering, 2002.

[30] Jeffrey Undercoffer and John Pinkston. An empirical analysis of computer attacks and intrusions. Technical Report TR-CS-03-11, University of Maryland, Baltimore County, 2002.

[31] WEBSTERS, inc, editor. *Merriam-Webster's Collegiate Dictionary*. Merriam-Webster, Inc., tenth edition, 1993.

[32] Chris Welty. Towards a semantics for the web. Vassar College, 2000.

[33] M. Wood and M. Erlinger. Intrusion detection message exchange requirements. draft-ietf-idwg-requirements-08, August 2002.