TRIDSO: Traffic-based Reasoning Intrusion Detection System using Ontology

Lisa Frye
Computer Science Department, Kutztown University, Kutztown, USA
Email: frye@kutztown.edu

Liang Cheng and Jeff Heflin
Department of Computer Science and Engineering, Lehigh University, Bethlehem, USA
Email: {cheng,heflin}@cse.lehigh.edu

Many Intrusion Detection Systems are capable of detecting simple attacks. Complex attacks often consist of a sequence of multiple simple attacks and are more difficult to identify, requiring the knowledge of experienced network engineers. An ontology representation of the complex attacks can introduce meaning to the data collected while monitoring the network, allowing the data to be understood by computers and the human expertise implemented. This will allow an Intrusion Detection System to identify complex attacks when they occur on a network. The details of this ontological representation and its implementation in the Traffic-based Reasoning Intrusion Detection System using Ontology (TRIDSO) are described. A detailed example of how TRIDSO detected a complex attack is explained. Lastly, a performance evaluation of TRIDSO was conducted and analyzed. TRIDSO was able to identify a variety of complex attacks. Due to performance outcomes, the current implementation of TRIDSO is best utilized for post-attack detection, which provides valuable evidence for security managers.

Keywords: *Computer network security; Intrusion Detection System; Ontology*

ACM Classifications: *C.2.3, D.4.6, I.2.4*

1. Introduction

Detecting an attack against a network or host, also referred to as intrusion detection, is an active research area that does not have a definitive solution. Intrusion detection is a difficult task for a variety of reasons. First, the pure volume of data that requires analysis to detect an intrusion is daunting, often making this task unmanageable. Second, the lack of a common format for representing attack data makes it difficult to utilize multiple systems to assist with intrusion detection. Limiting the data analysis to one system makes intrusion detection more difficult. Lastly, the differences in each individual attack, and the daily introduction of new attacks, make it difficult to represent the attacks formally. This often requires each attack to have its own representation, not allowing for generic attack representation. This limits the ability for multiple systems to use one representation for the same attack. It also makes new attack identification difficult. Intrusion detection is often performed by an Intrusion Detection System (IDS) (Fuchsberger, 2005).

Manuscript received: 14 November 2012
Communicating Editor: Rafael Valencia-García

A simple attack is a single-step attack that is generally straightforward to perform, such as to ping all nodes in a network. The occurrence of a simple attack in a network may indicate that an attacker is just trying an easy attack. The assemblage of several simple attacks may indicate the occurrence of a more complex attack. In order to understand the way simple attacks may fit together to form a complex attack, it is necessary to consider their spatial and temporal properties. For instance, pings to hosts on the same network, with incrementing IP addresses, over a span of several days, may indicate a network manager doing simple management or troubleshooting tasks. Given the same set of pings, over a span of several minutes, typically indicates an attacker looking for available hosts to attack. It is necessary to see that these ping packets are generated from the same source host and also within the same time period.

Frye, Cheng, and Kaplan (2011) developed a methodology to detect complex attacks. The methodology described in Frye *et al* (2011) is the preliminary design for the formal representation defined in this paper. This methodology was extended using the ontological knowledge representation (Spyns, Meersman and Jarrar, 2001). The ontology development was based upon the family of complex attacks identified by attack trees in Frye *et al* (2011). The colouring scheme has not been implemented yet and will be implemented in the future.

Ontology was chosen as the basis for the IDS primarily due to its semantic expressiveness. This allows the IDS to identify attacks from the meaning of the network data. In particular, the ontology features leveraged most in the IDS are the advanced class definitions and inference capability. Some of the advanced class definitions that proved beneficial were the ability to specify two classes as disjoint and define classes using Boolean combinations (union and intersection). The disjoint class definitions were valuable in identifying packet types. For instance, a packet cannot be both a TCP packet and a UDP packet. Through the use of inference, additional knowledge can be learned from the network data allowing more advanced identification rules.

The primary goal of this research was to develop a Traffic-based Reasoning Intrusion Detection System using Ontology (TRIDSO) to detect complex attacks. New in this research is an approach to represent complex attacks formally. The formal representation of complex attacks was utilized to develop an ontology describing these attacks.

The ontology definitions were the basis of an IDS. This IDS, TRIDSO, represents a new type of IDS and therefore is an important contribution to the development of more sophisticated approaches to intrusion detection. The IDS developed was based on ontology and attacks were identified by evaluating all network traffic. This led to the detection of complex attacks, as well as attack attempts. Many existing IDSs do not detect complex attacks or attack attempts; most detect successful simple attacks. Attack attempts are helpful to security managers in implementing an appropriate security framework.

Lastly is a performance evaluation of TRIDSO. A fundamental concern in the current implementation of TRIDSO is the performance. The large response times make TRIDSO unusable for real-time attack detection; however, post-detection provides beneficial information to security managers. Any attack information analyzed by security managers assists in security decisions.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 discusses the system architecture for the approach developed in this research. The ontology developed is described in Section 4. Section 5 describes the implementation and a use case in detail. Section 6 describes the evaluation results. Conclusions and future work are provided in Section 7.

2. Related Work

Over the years there has been a significant amount of IDS research. A variety of methods have been suggested for the implementation of IDS. One avenue of research is the use of ontology. Ontology has been utilized in various aspects of security and intrusion detection.

Martimiano and Moreira (2006) developed an ontology, ONTOSEC, to identify security incidents that exploit existing vulnerabilities. The attack information used in the ontology was obtained from an existing IDS. The primary purpose of this ontology was to provide a common format that could be utilized by a variety of security tools. This system can be used to store attack data but it is not capable of detecting an attack. It used data obtained from other tools, including IDSs, to determine if a security incident occurred. It will examine sequences of security incidents based on one incident preceding or succeeding another incident.

One system that utilizes ontology to aid in intrusion detection is the Reaction after Detection (ReD) Project (Cuppens-Boulahia *et al*, 2009). The primary goal of ReD was to determine the most appropriate reaction, both short and long term, to an identified attack. A long-term reaction will consist of the deployment of new security policies to the network. An ontology was utilized in this approach to instantiate the new security policies and determine policy violations.

An ontology-based IDS was proposed by Undercoffer, Joshi and Pinkston (2003) to detect intrusions against hosts. This approach is anomaly-based, so baseline behaviour of the network is obtained and abnormal behaviour identified. The ontology was used to define the attack and its properties, including the consequences and means of the attack. This work was able to detect complex attacks but required an IDS to be installed at each host to be monitored and the maintenance of known vulnerabilities.

Context-aware alert analysis was researched by Xu, Xiao and Wu (2009). They argued that alert analysis for unified security management can be divided into three stages: alert collection, alert evaluation, and alert correlation. An ontology was developed that included the context, asset owner, vulnerability, threat and countermeasure for attacks. Alert correlation was achieved by adding behavioural information through the use of rules.

Vorobiev and Bekmamedova (2007) discussed how distributed firewalls and IDSs (F/IDSs), monitoring different hosts, must work together in a distributed manner. Several ontologies were developed, most of which were used to give a simplified and common vocabulary for security incidents and the distributed F/IDSs. These worked collaboratively to detect multi-phased, complex attacks. When a host identifies an attack, it shares this information with the other hosts in the framework, which then uses the shared information to detect a multi-phased, complex attack.

Vorobiev and Bekmamedova (2010) also developed an ontology-based system to detect complex attacks against software system, particularly gaming systems. Many existing security ontologies were leveraged in this work to develop a shared vocabulary for system components. The various security ontologies utilized were explained following a case study of a Mitnick attack against a gaming system.

A multiagent system using ontology was developed for Outbound Intrusion Detection (OID) by Mandujan (2005). The goal of an OID is to help protect remote systems. This work accomplished OID by taking advantage of the fact that many complex attacks are automated using scripts or executable programs. The system developed analyzed changes in the network traffic and the resources used by an automated attack tool. The ontology identified all elements about the

originating system, including automated attack tools, network traffic, signatures, sensors, and reactions, as well as their relationships.

Much of the previous work is focused on identifying vulnerabilities of systems and evaluating the threats against these targets. The work presented in this paper focused on the network traffic and not the vulnerabilities of targets. By doing this, it was possible to identify attacks and also attack attempts, even if the vulnerability didn't exist in the target node or network. This may have been the result of the service or application that is the target of the vulnerability not being implemented in the network, or the target may have been patched to resist the vulnerability, etc. It is important to note that attack attempts are just as important or meaningful as an actual attack. The attempts can alert the administrator to an attacker that is trying to penetrate their network or a node on their network. It also allows the administrator to prepare for future deployments, such as a user adding a web server to the network that may contain vulnerabilities.

The work here began with specific attack examples but evolved into more general cases. The rules developed for identifying complex, multi-phase attacks are generic, and will lead to the identification of any type of attack, including zero-day attacks. These rules will allow a family of complex, multi-phased attacks to be defined and detected. By representing these attacks ontologically, a more advanced and reusable representation of network attacks will be created.

3. System Architecture

There are existing IDSs that examine network traffic and identify possible attacks to the network. Many of the attacks identified by these IDSs are simple attacks or attacks consisting of one single attack. The IDS alarms when a single attack type is identified in the network traffic. Snort (2011) and Caswell, Beale and Baker (2007) is an example of this type of IDS; it identifies possible simple attacks by checking network traffic against rules and alarms if any traffic matches the rule.

A. Traffic Centric Architecture

Many IDSs identify intrusions by looking for data that is destined for a host with a vulnerability that the data can exploit. These systems only identify intrusions against known vulnerabilities. The network manager should not only be concerned with attacks against nodes that are vulnerable, but should also watch for any attack attempt by an intruder. This is important because a network manager cannot predict what users on the network will install or deploy.

For example, consider an intruder attempting to circumvent a vulnerability in web services. If there are no vulnerable systems on the network, then the attack attempt would be unsuccessful; however, this does not preclude the same attack becoming successful in the future. If a user installed a new web server that is vulnerable to the attack, the attack attempt would then become successful against this new web server.

To make the network more resistant to successful attacks, the network manager should analyze all attack attempts against the network. TRIDSO is based on using all network traffic. This allowed the system to detect all intrusion attempts, regardless if the intrusion was successful or not.

Another characteristic of many IDSs is that they detect attacks against hosts. These IDSs typically evaluate the current state of the host, or a change in state, which often requires software installation on the hosts. TRIDSO requires no software installation on the hosts. One primary differentiator of TRIDSO is that its attack detection is based on network events and not known vulnerabilities or host state.

Another advantage of basing attack detection on network data and not host data is the ability to identify packets destined for multiple nodes on the same network. Many times the targets of the simple attacks in a complex attack are different nodes, which makes the attack more difficult to detect. By examining all traffic data, these patterns can also be detected. TRIDSO is capable of detecting attacks against multiple hosts, such as a ping scan. Ontology can aid in this type of detection by taking advantage of its inference capability. Without ontology, a complex programmatic implementation would be required to identify some of the simple attacks, like the ping scan. Another advantage of ontology is its adaptability. Maintaining and enhancing a programmatic implementation is more difficult than an ontological implementation.

B. System Design

The system design, illustrated in Figure 1, consists of four subsystems: vulnerability, device, traffic and attack. The reasoner is necessary to query the knowledge base, which stores the ontologies and their instances. TRIDSO used the built-in reasoner of Jena (2012), an ontology development library.

The vulnerability subsystem contains data about existing vulnerabilities. The device subsystem consists of the device ontology and a mapper to convert device data to ontology instances. The ontology consists of devices in the network and their characteristics. Implementation of these subsystems will be future work.

The traffic subsystem deals with the raw network traffic data. A packet sniffer captures all network traffic, which is converted to ontology instances. This ontology represents the raw network traffic data in a variety of forms, such as individual frames or datagrams, packet streams, TCP connections, etc. Also part of the traffic subsystem is information about alerts found using an existing tool, Snort. The capture file is the input to Snort and the output is then used as the input to a mapper that creates ontology instances for all alerts found.

The attack subsystem consists of an ontology that describes attacks that can occur. This subsystem is unique in that it consists entirely of ontology files. There are two ontology files, one that describes simple attacks and one that describes complex attacks. The attack data is obtained from the traffic ontology. This information is used to create additional instances in the knowledge base, particularly to identify the occurrence of simple and complex attacks.

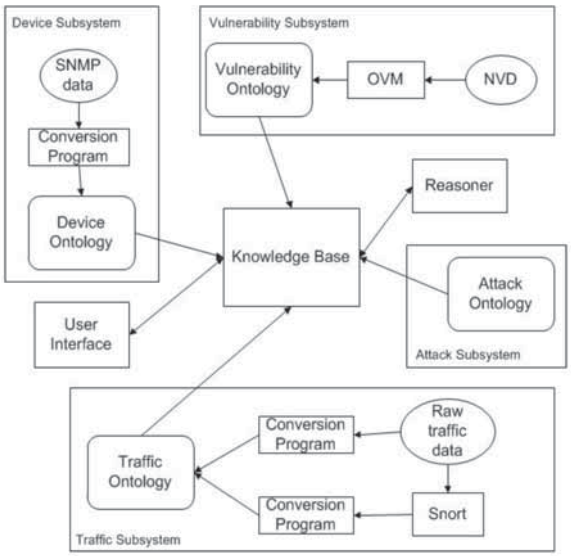

Figure 1: TRIDSO Architecture (arrows indicate the flow of data)

4. Ontology Development

The primary component of TRIDSO is the various ontologies. Each subsystem in TRIDSO includes at least one ontology to represent data necessary for that subsystem. The ontologies were written using OWL (2012). For the first phase of

development in TRIDSO, only the traffic and attack subsystems were implemented.

Ontology development is a cyclic process that begins with defining the domain's terms. The difficult part of this process is determining how to define each term, as a class, property or instance. The ontology development process for TRIDSO is defined by Frye (2013).The ontologies for traffic and attack subsystems are described here. The ontology definition files for TRIDSO are available online (Frye, 2012).

A. Traffic Ontology

Network traffic was captured using a packet capture utility. This data was converted to ontology instances in the traffic ontology. There are many different classes in the traffic ontology (see Figure 2). The primary class is the *Packet* class, containing the date and time for the packet.

Figure 2: Traffic Ontology Diagram

This *Packet* class is then broken down into the various layers according to the Internet Protocol Stack (Kurose and Ross, 2013). For instance, the *TCPPacket* class contains instances of all TCP packets found in the captured traffic data. This class contains properties specific to TCP packets, such as sequence number, acknowledgement number, and TCP flags; however, because of OWL's class relationships, it also inherits all the properties of its parent classes (*L4Packet*, *IPPacket*, *L2Packet* and *Packet*). The *Application* class contains the application layer protocol and data for ICMP and layer-4 packets (TCP and UDP).

The *PacketCollection* class was used to group common packet instances and classify them according to type. The types of concern to TRIDSO were identified in the *PacketType* class and were various denial of service flood attacks, ping scan, port scan and the teardrop attack. For instance, if there were multiple ping packets to the same node within a specified time frame, an instance was created in the *PacketCollection* class of *PacketType PingFloodType*. These instances were later used by TRIDSO to assist in attack identification.

The *PacketSequence* classes were utilized in identifying several packets that are meaningful if they occurred in a specified order. If order is not important, an instance was not created in these classes.

Packet Type	Packet Data	Ontology Class	Ontology Property
Any	Date and time	*Packet*	*dateTime*
ARP	Source MAC address	*L2Packet*	*hasSrcMAC*
ARP	Destination MAC address	*L2Packet*	*hasDestMAC*
IP	Source IP address	*IPPacket*	*hasSrcIP*
IP	Destination IP address	*IPPacket*	*hasDestIP*
IP	IP version	*IPPacket*	*ver*
IP	Packet length	*IPPacket*	*packetLen*
IP	Time to Live	*IPPacket*	*ttl*
TCP / UDP	Source port number	*L4Packet*	*l4SrcPort*
TCP / UDP	Destination port number	*L4Packet*	*l4DestPort*
TCP	Sequence number	*TCPPacket*	*tcpSeqNum*
TCP	Acknowledgement number	*TCPPacket*	*tcpAckNum*
TCP	Flags	*TCPPacket*	*tcpFlags*
ICMP	ICMP type	*ICMPPacket*	*icmpType*
ICMP	ICMP code	*ICMPPacket*	*icmpCode*

Table 1: Relationship Between Packet Data and Ontology Property

Various packet types were inferred in the traffic ontology. Most of the packet types used OWL restrictions to create their instances. The packet types of interest in TRIDSO were special packets using the protocols TCP and ICMP (Internet Control Message Protocol). As an example, a ping packet is an ICMP packet with an ICMP Type value of 8; therefore, the *PingPacket* class is the intersection of the *ICMPPacket* class and the restriction of the *ICMPType* property to be the value of 8. The OWL code for this class definition is shown in Figure 3. The remaining packet types in TRIDSO were handled in a similar fashion, by placing restrictions on properties of the *TCPPacket* or *ICMPPacket* classes.

```
<owl:Class rdf:ID="PingPacket">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#ICMPPacket"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#icmpType"/>
      <owl:hasValue rdf:datatype="&xsd;integer">8</owl:hasValue>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Figure 3: OWL code for the *PingPacket* class

The network traffic data that was captured was run through Snort. An alert was created in Snort when a simple attack was identified in the input file. The output of Snort, consisting of all the alerts identified, was also used to create ontology instances in the traffic ontology. An instance was created in the Alert class hierarchy for each alert identified by Snort. This allowed the system to take advantage of an existing tool to aid in its intrusion identification. Table 2 shows some of the relationships between the alert information generated from Snort and ontology properties.

Alert Type	Alert Information	Ontology Class	Ontology Property
Any	Date and time	*Alert*	*aDateTime*
Any	Identification	*Alert*	*aID*
Any	Description	*Alert*	*aDescription*
IP	Source IP address	*IPAlert*	*hasAlertSrcIP*
IP	Destination IP address	*IPAlert*	*hasAlertDestIP*
IP	Header length	*IPAlert*	*aIPHdrLen*
IP	Packet length	*IPAlert*	*aIPDgramLen*
TCP / UDP	Source port number	*L4Alert*	*aL4SrcPort*
TCP / UDP	Destination port number	*L4Alert*	*aL4DestPort*
TCP	Sequence number	*TCPPacket*	*aTCPSeqNum*
TCP	Acknowledgement number	*TCPPacket*	*aTCPAckNum*
TCP	Flags	*TCPPacket*	*aTCPFlags*
ICMP	ICMP type	*ICMPAlert*	*aICMPType*
ICMP	ICMP code	*ICMPAlert*	*aICMPCode*

Table 2: Alert Information's Relationship with Ontology Property

B. Attack Ontology

There were two ontologies created for the attack information in TRIDSO. The first was the attack ontology. This ontology was used to identify simple attacks. The instances were created by using inference via ontology constructs and SPARQL (2012), a query language for RDF (2011). Some instances were created using ontology inference by utilizing some advanced class definitions, such as restrictions, intersections, and unions.

The top-level of the attack ontology hierarchy includes four classes: *Availability*, *Recon*, *GainAccess*, and *ViewChangeData*. Each of the four classes in the top-level represents a high-level type of attack. The hierarchy for each of these classes was extended to include more detailed attacks of each type. One such hierarchy is depicted for the *Availability* class in Figure 4.

The inference capability provided by ontology was a key reason the IDS utilized ontology. As one example of the use of inference in TRIDSO, consider the *PingFlood* class. A ping packet instance, which is part of the traffic ontology, was created by defining a collection of all ping packets. The number of ping packets from the *PingPacket* class that occurred in a specified timeframe to the same node or network was determined. If the number of ping packets in the timeframe was above a threshold, then an instance was created in the *PacketCollection* class with a type of *PingFloodType*. The instances of the *PingFlood* class were the packet collections of *PingFloodType*. These were created using more inference; they were the intersection of the instances in the *PacketCollection* class that had the value of *PingFloodType* for the *pcType* property.

The instances in the super classes of the *PingFlood* class were also created using inference through OWL constructs. These instances were created by using taxonomic relationships between the classes. *PingFlood* is a subclass of the *Flood* class, so any instance in *PingFlood* was also an instance in *Flood*. Each node in Figure 4 is a subclass of its parent node, so each parent node inferred its

instances from its child node. Through these taxonomic relationships, instances were created in *Resources*, *DoS*, *Availability*, and *Attack*, all from the instances in the *PingFlood* class. If ontology was not used, a programmatic solution would be required to create these additional instances. Ontology greatly simplified this task.

C. Complex Attack Ontology

Complex attacks (Figure 5) are represented in a separate ontology, primarily for ease of organization and management. The complex attacks were built by exploiting inference in the attack ontology. The leaf nodes are the classes from the attack ontology; they are represented in this figure for discussion purposes. The only new classes defined for complex attacks were the *complexAttack* class and the four top-level classes.

Instances became part of the top-level classes through the use of Boolean combinations. If correlated instances existed in each leaf node, then an instance was created in the top-level class. The correlation drawn depended on the top-level class, or type of complex attack. For instance, if an instance existed in the *PingScan*, *NodeScan*, and *Availability* classes with a target IP address of the same node, then an instance was created in the *DoSComplex* class, indicating the existence of a complex denial-of-service attack.

The instances in these top-level classes leveraged inference with OWL constructs. The property *wasAttacked* has a range of *IPaddress*,

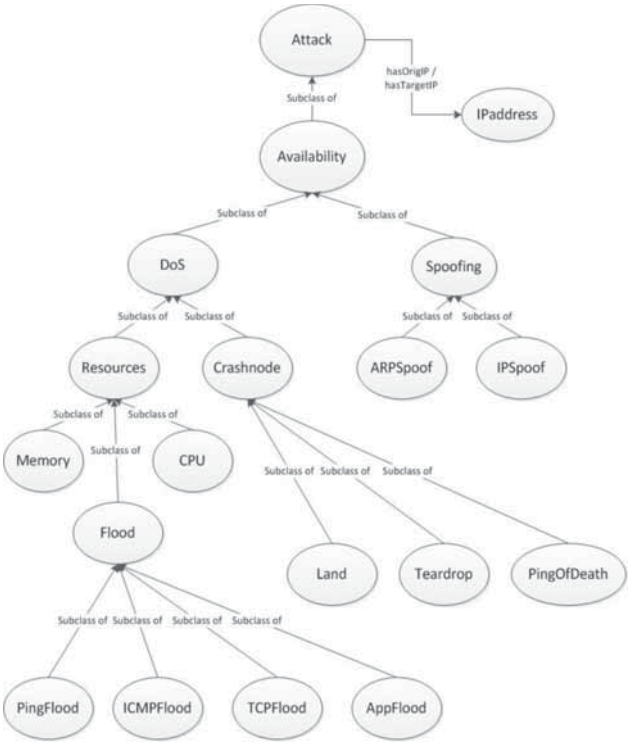

Figure 4: Availability Hierarchy in Attack Ontology Diagram

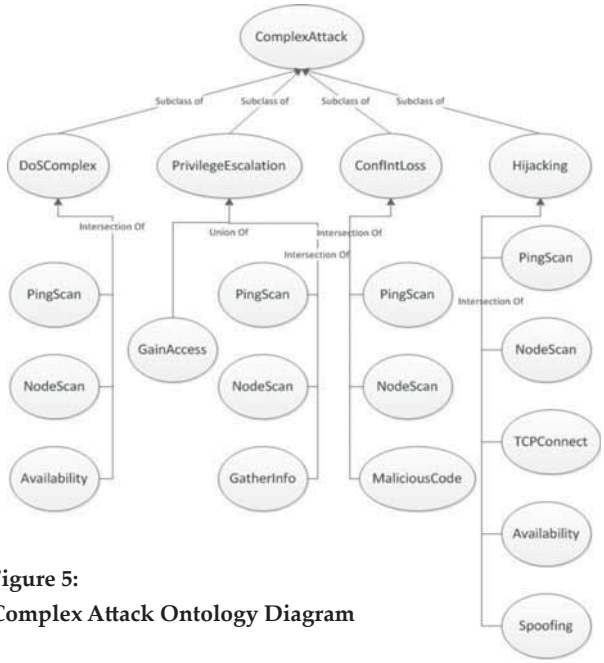

Figure 5:
Complex Attack Ontology Diagram

a domain of *Attack*, and is the inverse of *hasTargetIP*. Through the inverseOf OWL construct, an instance was inferred for the *wasAttacked* property for any IP address that had an instance in the *Attack* class with a target IP address of that IP address. Instances of the *DoScomplex* class were created through the intersection of the instances of the *IPaddress* class in the traffic ontology that had values of the *wasAttacked* property from the *PingScan*, *NodeScan*, and *Availability* classes. That means that there were instances in the *PingScan*, *NodeScan*, and *Availability* classes with the same destination IP address. These instances were identified using the someValuesFrom restriction in OWL. The OWL code for the *DoScomplex* class is shown in Figure 6.

```
<owl:Class rdf:ID="DoSComplex">
 <rdfs:subClassOf rdf:resource="#ComplexAttack"/>

 <owl:equivalentClass>
  <owl:Class>
   <owl:intersectionOf rdf:parseType="Collection">
     <owl:Class rdf:about="&traffic;NWaddressScanned"/>
     <rdf:Description rdf:about="&traffic;IPaddress"/>
     <owl:Restriction>
       <owl:onProperty rdf:resource="&attack;wasAttacked"/>
       <owl:someValuesFrom rdf:resource="&attack;PingScan"/>
     </owl:Restriction>
     <owl:Restriction>
       <owl:onProperty rdf:resource="&attack;wasAttacked"/>
       <owl:someValuesFrom rdf:resource="&attack;NodeScan"/>
     </owl:Restriction>
     <owl:Restriction>
       <owl:onProperty rdf:resource="&attack;wasAttacked"/>
       <owl:someValuesFrom rdf:resource="&attack;Availability"/>
     </owl:Restriction>
   </owl:intersectionOf>
  </owl:Class>
 </owl:equivalentClass>

</owl:Class>
```

Figure 6: OWL code for the *DoScomplex* class

D. A Set of Queries

When possible, OWL constructs were used so the power of ontology could be attained. Some of the OWL constructs utilized that simplified development and attack identification were taxonomic relationships and advanced class definitions through the use of Boolean expressions. There were aspects of the attacks that were complex making it difficult to represent using OWL constructs. When OWL constructs were not sufficient, then SPARQL was utilized, which has more flexibility. SPARQL is a query language, which resembles SQL for databases and includes the ability to dynamically add instances to the knowledge base. Many times SPARQL was required because it was necessary to select instances based on specific criteria and then narrow the results even more before inserting an instance into the knowledge base, which is also possible with SPARQL.

An aspect of the spatial domain of attacks is the location, physical or logical, in the network where the attack occurred. The simple attacks comprising one complex attacks may not all target the same node. It is not sufficient to simply examine the destination IP address of some specific types of packets; it may require determining the network address for these packets to see if the packets are part of the spatial domain of other packets. This was one example of the necessity of SPARQL, the creation of PingScanType packets in the *PacketCollection* class.

Ping scan packets were identified as ping packets to nodes in the same network. The network is identified by the IP address. For the current version of TRIDSO, only classful IP addresses were evaluated. A future version will also consider classless IP addresses. There are three classes of IP addresses, class A, class B and class C. The type of address is also determined by the IP address. The value of the first octet in the IP address will indicate the class for that IP address. All instances

to the same network were identified, which required instance values to be compared to each other. If the number of pings to the same network exceeded a threshold value, then the network address was added to the *IPaddress* class and an instance was created in the *PacketCollection* class of *PingScanType*. These instances were used to create instances in the *PingScan* class of the attack ontology. OWL was used to create the instances in the *PingScan* class, similar to the *PingFlood* instances.

Identifying a port scan attack against a specific node also necessitated the use of SPARQL. This required identifying a specified number of packets that were sent to the same destination IP address but different port numbers. With the varying port numbers, it was not possible to represent this using OWL. A SPARQL query was written, which is shown in Figure 7 that first selected and counted the number of packets sent to the same destination IP address but different ports. If this count, which represented the number of unique ports for that target, was over a specified threshold, then an instance was added to the *PacketCollection* class of type *PortScanType* for that target IP address. In this implementation of TRIDSO, the threshold was 0, which is reflected in the query. Future research will determine the appropriate value for this threshold, which will then be implemented in future versions of TRIDSO. This instance was also added to the *PortScan* class via OWL constructs indicating a port scan attack occurred.

```
INSERT
{  _:a rdf:type  attack:PacketCollection;
          attack:beginDate  ?beginDateTime;
          attack:endDate  ?endDateTime;
          attack:pcType traffic:PortScanType;
          attack:hasTargetIP ?destIP;
          attack:pcFrequency ?cnt .
} WHERE { {
   SELECT DISTINCT ?packet1 ?destIP
                    (MIN(?dateTime) as ?beginDateTime)
                    (MAX (?dateTime) as ?endDateTime)
          (count(?destIP) as ?cnt)
    { ?packet1 rdf:type traffic:L4Packet;
                    traffic:dateTime ?dateTime;
                    traffic:hasDestIP ?destIP;
                    traffic:l4DestPort ?l4DestPort1 .
      {
       SELECT ?packet2 ?destIP ?l4DestPort2 ?dateTime2
       { ?packet2  rdf:type traffic:L4Packet;
                      traffic:dateTime ?dateTime2;
                      traffic:hasDestIP ?destIP;
                      traffic:l4DestPort ?l4DestPort2 .
       } GROUP BY ?destIP
      }   FILTER ( ( ?packet1 != ?packet2) &&
           ( ?l4DestPort1 != ?l4DestPort2 ) ) .
    }   GROUP BY ?destIP
        HAVING (count(?destIP) > 0)
} }
```

Figure 7: A SPARQL rule to describe a node port scan

Another case where it was necessary to use SPARQL was the creation of instances for many of the simple attacks identified by Snort. For example, when Snort detects an attack, it creates information about the attack in a file. This file was parsed by TRIDSO at run time and instances were added to the knowledge base for each alert. Now it is necessary to parse the properties for each instance and determine the type of alert, often requiring the parsing of the alert classification or description generated by Snort. This required the use of regular expressions, which are supported by SPARQL. So, to create an instance for attacks where the attacker gained root access to a node, Snort would generate an alert classification containing the string "Administrator Privilege Gain". A SPARQL query was written to search for the classification property in the *Alert* class that contained the regular expression matching that string. For all instances found, a new instance was inserted by SPARQL into the *Attack* class with a type of *AdminPG*.

5. Implementation and Use Case Studies

The system was designed to use ontology and related tools to minimize customized code. This allows the features of ontology to be leveraged, allowing better adaptability and flexibility in attack detection. The majority of the customized code was to initially populate the knowledge base with traffic data using a mapper program.

A. System Implementation

TRIDSO was implemented using Java and Jena. Jena was chosen for the ontology aspects because it provides a leading implementation of SPARQL. The version of Jena utilized also includes support for SPARQL extensions, such as INSERT and *count*, which are necessary in the queries developed.

Before processing any data, an ontology model was created and populated with the system's ontology files. This created the knowledge base and allowed instances to be properly inferred as the knowledge base was dynamically populated with instance data.

Network data was captured using a packet capture utility. The program's customized code processed the capture file; for each packet read, the packet type was determined, such as TCP, UDP, IP or ARP, and the required data for that packet type was extracted. An instance was created in the knowledge base for the appropriate class of the *Packet* hierarchy in the traffic ontology.

The alert processing accomplished the same thing for the alert file and the *Alert* class hierarchy. For this to occur, the capture file was processed by Snort, which generated an alert file. The alert file was then processed by customized code creating instances in the knowledge base. Figure 8 shows the overall data flow for the traffic subsystem.

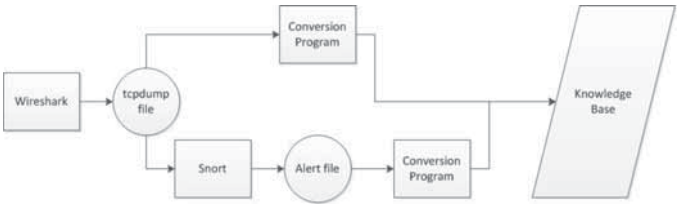

Figure 8: The data flow of the traffic subsystem

The majority of additional instances were either inferred in the ontology or created using SPARQL. Upon completion of the SPARQL queries, the knowledge base was ready to answer attack queries. One such query was to identify that a complex attack occurred.

B. Mitnick Attack Use Case

To test TRIDSO, real attacks were launched in a test environment. The data associated with the attacks was captured using packet capture software. Two use cases of the system, a complex denial of service and the Mitnick attack were explained in Frye, Cheng and Heflin (2012).

The Mitnick attack (Northcutt, 2001) is an example of a hijacking attack. The steps in the Mitnick attack are:

1. Find active hosts to identify a target machine
2. Identify TCP connections on the active hosts
3. Predict the TCP sequence number for a TCP connection
4. Take one host in the TCP connection offline

5. Insert the source machine into the TCP connection by spoofing the host that was taken offline in step 4

The specific attacks used in each step may vary from attack to attack. The Mitnick attack used to test TRIDSO utilized common attacks for each step.

The first step in the Mitnick attack is to identify an active host using a ping scan. When an attacker launches a ping scan attack, the data captured will show ping packets to many IP addresses on the same network. A ping packet is an ICMP packet with a message type of 8 (echo). A *PingPacket* class was defined in the traffic ontology that was all instances in the *ICMPPacket* class with a value of 8 for the *icmpType* property. If the number of these packets to nodes in the same network was over a threshold value, then an instance was inserted in the *PacketCollection* class of type *PingFloodType*. This required some advanced reasoning and required the use of a SPARQL query, which is shown in Figure 9. The ontology was then able to infer the existence of an instance in the *PingFlood* class, which was simply defined as all instances in the *PacketCollection* class of type *PingFloodType*. This was done with a class definition using the intersectionOf OWL construct, similar to the class definition for the *PingPacket* class shown in Figure 3.

```
INSERT
{
  _:a rdf:type attack:PacketCollection;
      attack:beginDate ?beginDateTime;
      attack:endDate ?endDateTime;
      attack:pcType traffic:PingFloodType;
      attack:hasTargetIP ?destIP;
      attack:pcFrequency ?cnt .
}
WHERE { {
  SELECT  ?destIP   (MIN(?dateTime) as ?beginDateTime)
            (MAX (?dateTime) as ?endDateTime)
            (count(?destIP) as ?cnt)
     WHERE {?pack rdf:type traffic:PingPacket;
              traffic:dateTime ?dateTime;
              traffic:hasDestIP ?destIP .
     }  GROUP BY ?destIP
        HAVING (count(?destIP) > 0)
} }
```

Figure 9: A SPARQL rule to describe a *PingFlood*

The second step in the attack is to discover an active TCP connection for that port as well as the second host in the connection. For this particular attack a node scan was used to identify active ports. Then the ontology was utilized to determine if there was an active TCP connection for each identified port.

Predicting the TCP sequence number is the third step. This prediction attack utilizes the creation and quick termination of many TCP connections to the same host. Software can be used to accomplish this, which will then also determine what sequence number should be used for the next segment in the TCP connection. Knowing this information will allow the attacker to create their own TCP segment to send to the unknowing target. This attack was identified in TRIDSO by detecting many TCP packets with the RST flag set to the same host. This necessitated comparing data for two different packets at the same time, which required a SPARQL query.

The fourth step is to take one host in the TCP connection offline so it can be impersonated in the TCP connection. This is typically accomplished using a denial of service attack (originally a Syn Flood attack) against the identified host. Another common denial of service attack is a ping flood attack, where the attacker will just send enough ping packets to the targeted host that it is consumed with responding to these ping requests. This is the denial of service attack that was used in the test Mitnick attack. Identification of this attack was similar to the ping scan except instead of looking for a threshold number of ping packets to hosts on the same network, the ping packets needed to be sent to the same host. This was accomplished via ontology by identifying *PacketCollection*

instances of type *PingFloodType* as explained previously for the ping scan. An instance was then inferred to exist in the *PingFlood* class.

The last step is pretending to be the host that is now offline due to the denial of service attack. This is usually done with an IP spoofing attack. For the IP Spoofing attack element, the *Stream* class is the key. The *L3Stream* includes instances for observed packets with the same source and destination, essentially creating a mapping of IP address to MAC address. When a new *L3Stream* is inserted for the IP Spoofed packet(s), the IP-MAC address mapping is different. Historical data is used to identify the IP Spoofed correlation as an anomaly, indicating the occurrence of an *IPSpoof* attack, causing the creation of an instance in the IPSpoof class in the attack ontology.

Figure 10 shows the Attack hierarchy in the ontology after detection of the simple attacks that comprise the Mitnick attack. The nodes shaded with horizontal lines indicate the simple attacks just described that were detected from the captured traffic data. The nodes shaded with vertical lines were all inferred from the ontology.

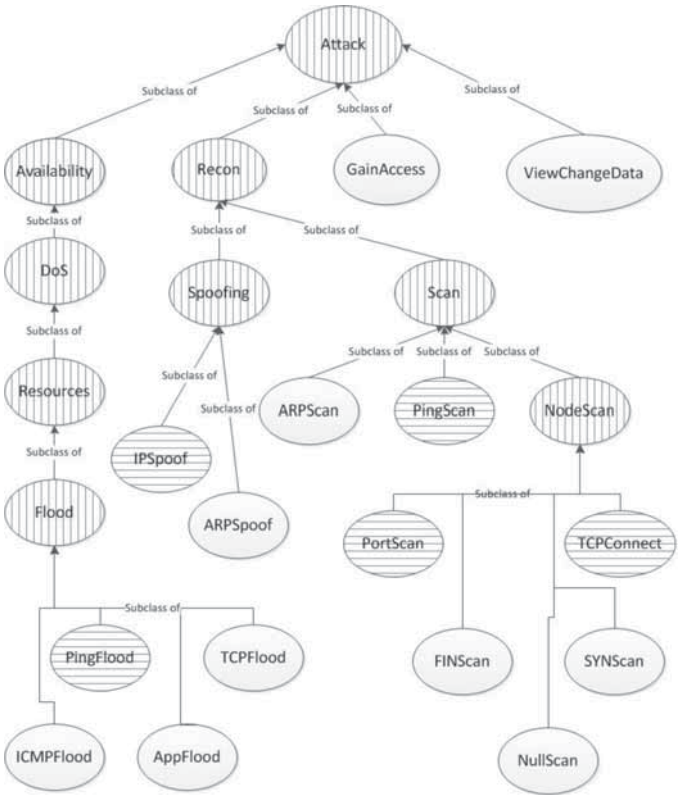

Figure 10: The ontology hierarchy for the Mitnick attack

At this point, the knowledge base consists of instances for all the shaded nodes in Figure 9. A hijacking attack is defined as the intersection of the following: a) a network that was scanned for IP addresses, b) a host on that network that had attacks of node scan and a TCP connection attack, and c) a host with a TCP connection with the host from step b that was attacked with availability and spoofing attacks. The class definition for the *Hijacking* class performed this intersection using the intersectionOf OWL construct. The Mitnick attack is a specific example of a hijacking attack. TRIDSO will indicate that a hijacking complex attack occurred and not be specific about the type of hijacking attacks, Mitnick in this case.

6. Evaluation Results

TRIDSO was evaluated using three methods. The first method compared TRIDSO results with a state-of-the-art IDS. The second method analyzed the response times. The third method was a scalability evaluation. The evaluations were conducted on data sets that focused on the two use

cases described in the previous section. The data sets were captured in a test network established for this research.

The state-of-the-art system used to compare with TRIDSO results was Snort. This system was chosen because it is current and commonly used in existing networks. For the complex denial of service use case, Snort detected only the simple attacks that comprised the complex attack. The TRIDSO output included the detection of the simple attacks as well as the identification of the occurrence of the complex DoS. In the Mitnick attack, Snort again only detected the simple attacks (the ping and port scans). Once again, TRIDSO detected the simple attacks but also detected the occurrence of a hijacking attack.

When evaluating the response times, two in particular were analyzed, the load time and the query time. The load time refers to the time it took to load the instances into the knowledge base from the raw data files. The query time is the time it took for a SPARQL query to execute.

As illustrated in Table 3, the load time increases as the number of packets and alerts increase. A direct equation cannot be drawn from these results as the load time varies based not only on the size of the data set but also on the types of packets in the raw data and the associated class definition. As the data set size increases, the load time becomes undesirable. This will require attention in future research to allow TRIDSO to be utilized with large data sets.

	Input (numbers)		Load Time (ms)	
Data Set	**Packets**	**Alerts**	**Ontology Definition Files**	**Raw Data Instances**
Ping scan	12	4	503.907	27.652
Port scan	100	0	429.295	3151.427
Complex DoS	314	8	417.031	674.658
Hijacking	550	164	438.941	18,696.442

Table 3: Load Time Performance for Trial Data Sets

The response time of a query used in TRIDSO was also evaluated. Table 4 shows the response time to add alert instances to the knowledge base, which were instantiated using SPARL queries. These results show that this response time is directly related to the number of alerts generated from the raw data. As the number of alerts increases, the time to add the alert-related instances to the knowledge base increases.

	Response Time (ms)		
Data Set	**Number of Alerts**	**Query Time to Add Alert Instances**	**Query Time to Add Alert-related Attack Instances**
Port scan	0	0	66.91
Ping scan	4	27.652	103.74
Complex DoS	8	674.658	1,687.90
Hijacking	164	18,696.440	63,450.320

Table 4: Alert Query Response Time Performance for Trial Data Sets

The response times to add instances for the *PacketCollection* class to the knowledge base, which used SPARQL queries, is shown in Table 5. These instances are added by first selecting instances from a variety of classes based on specific criteria and then adding these instances of the *PacketCollection* class to the knowledge base. The table shows the number of instances returned from the SELECT query, the number of instances actually added to the knowledge base, and the response time to do the entire query, which includes the SELECT and the insertion. These queries are complex and lead to unacceptable response times for a real-time system. This overhead of SPARQL helps to demonstrate why OWL constructs were used when possible.

	Response Time (ms)		
Data Set	**Number of instances in SELECT clause**	**Number of instances inserted**	**Query Time to Add Instances**
Ping scan	35	6	2,542,030.124
Port scan	344	5	3,230,520.202
Complex DoS	1124	13	10,815,152.304
Hijacking	1902	51	8,771,351.157

Table 5. Query Response Time Performance for Trial Data Sets.

The last evaluation method was the scalability of the system. This was evaluated by analyzing the overall system response time for different size data sets. Figure 11 shows the total response times for attack detection. The size of the knowledge base includes all the ontology instances, with a variety of values shown for each data set. The raw value is the number of instances added from the raw network data and total is the number of total instances in the knowledge base for the data set. The other number for the data set is the total number of triples in the knowledge base.

As this figure demonstrates, the total response times are measured in minutes. This is unacceptable for any type of network system. It prevents TRIDSO from being utilized for real-time attack detection.

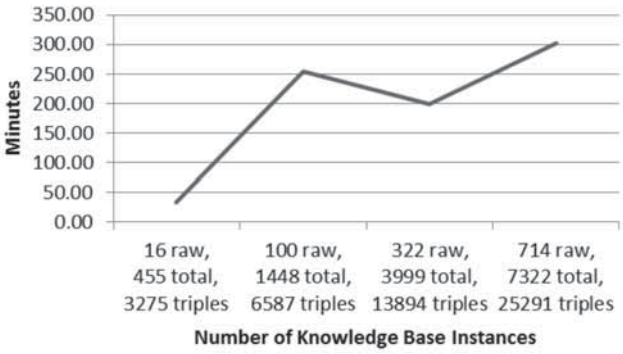

Figure 11: The run time performance of TRIDSO

TRIDSO is still a valuable tool for network managers. Security is a continuous process. It is imperative for the network manager to learn about attacks against the network, even if it is post-attack detection. It is better to identify attacks after they occur, allowing for prevention techniques to be applied and prevent future attacks, than to not learn about the attack. Scalability of TRIDSO will garner significant attention in future work.

TRIDSO utilized the Jena framework for the ontology API and knowledge base. It was chosen for its availability and ease-of-use. Jena is acceptable for use in a proof-of-concept system but is not a

scalable environment. A full implementation of TRIDSO would require a different environment, which would immediately impact the response times and scalability.

7. Conclusions, Future Work and Limitations

Network attacks occur on a daily basis, often going undetected. With the number of users relying on networks increasing at a rapid rate, both for personal and business reasons, it is imperative that networks and services be available at all times. A successful attack against a network often makes the network or services unavailable, making attack detection imperative in today's networks.

There are many types of Intrusion Detection Systems available with most of them being only able to detect simple attacks, such as scanning for an available host or a vulnerable port. Many attacks are complex, consisting of several simple attacks conducting in sequence. The development of an IDS capable of detecting complex attacks would be a significant contribution to the area of attack detection.

A newly developed system, TRIDSO, monitors the network traffic looking for the occurrence of complex attacks. The attack detection is based on reasoning capabilities of ontology. Three ontologies, in two subsystems, were developed and incorporated into TRIDSO, allowing for an IDS capable of detecting complex attacks while providing adaptability and flexibility in the system. The development of the ontologies to describe specific traffic and attack concepts provides an approach to representing generic attacks. This general representation of attacks, described by the ontologies, is a contribution to intrusion detection.

The Mitnick attack was explained to demonstrate how TRIDSO was able to detect complex attacks. By using ontology to infer new instances and SPARQL to create new instances, simple attacks were identified. If the simple attacks comprising a complex attack were in the TRIDSO knowledge base, then TRIDSO detected that complex attack and provided relevant information to the network manager.

TRIDSO was evaluated using data sets captured in a test environment. Evaluation results for TRIDSO demonstrate that the system is capable of detecting complex attacks. It is not yet capable of functioning as a real-time IDS; however, it can still serve as a valuable resource for network managers. Future research will focus on the response times and scalability of the system.

The ontologies and system implementation of TRIDSO will continue. Refinement of the three ontologies, traffic, attack and complex attack, will be one area of continued development. The area that will garner a significant amount of future work will be the scalability of the system. Methods will be evaluated to help decrease the response times, such as selecting a different ontology environment and implementing some multiprocessing concepts. As the evaluation process proceeds, it is hopeful that the generalized representation of the complex attacks will lead to the identification of unexpected complex attacks. Completion of TRIDSO, allowing for the identification of generalized complex attacks would be ground-breaking work for intrusion detection research.

References

CASWELL, B., BEALE, J. and BAKER, A. (2007): *Snort IDS and IPS Toolkit*. Burlington, MA: Syngress Publishing, Inc.

CUPPENS-BOULAHIA, N., CUPPENS, F., LÓPEZ DE VERGARA, J.E., VÁZQUEZ, E., GUERRA, J. and DEBAR H. (2009): An ontology-based approach to react to network attacks. *International Journal of Information and Computer Security* **3**(3/4): 280–305.

FRYE, L. (2012): Frye's Network Management Research. http://faculty.kutztown.edu/frye/res/index.html. Accessed 12 January, 2012.

FRYE, L. (2013): Ontology Development for an Intrusion Detection System. *28th Annual Conference of the Pennsylvania Computer and Information Science Educators (PACISE'13)*, East Stroudsburg, PA.

FRYE, L., CHENG, L. and HEFLIN, J. (2012): An Ontology-Based System to Identify Complex Network Attacks. *First IEEE International Workshop on Security and Forensics in Communication Systems, part of IEEE International Conference on Communications 2012*, Ottawa, Canada.

FRYE, L., CHENG, L. and KAPLAN, R. (2011): A Methodology to Identify Complex Network Attacks. *The 2011 International Conference on Security and Management (SAM'11) at The 2011 World Congress in Computer Science, Computer Engineering, and Applied Computing (WORLDCOMP'11)*, Las Vegas, NV.

FUCHSBERGER, A. (2005): Intrusion Detection Systems and Intrusion Prevention Systems. *Information Security Tech. Report* 10(3): 134–139.

JENA – A Semantic Web Framework for Java. http://jena.sourceforge.net/index.html. Accessed 5 January, 2012.

KUROSE, J.F. and ROSS, K.W. (2013): *Computer Networking: A Top-Down Approach*, Sixth Edition. New York, Addison-Wesley.

MANDUJANO, S. (2005): An Ontology-supported Outbound Intrusion Detection System. *Proceedings of the 10th Conference on Artificial Intelligence and Applications, Taiwanese Association for Artificial Intelligence (TAAI 2005)*, Kaohsiung, Taiwan.

MARTIMIANO, L.A.F. and MOREIRA, E. (2006): The evaluation process of a computer security incident ontology. *The 2nd Workshop on Ontologies and their Applications (WONTO'06)*, Ribeirão Preto, SP, Brazil.

NORTHCUTT, S. (2001): *Network Intrusion Detection: An Analyst's Handbook*. Sams Publishing.

RESOURCE DESCRIPTION FRAMEWORK (RDF): http://www.w3.org/RDF/. Accessed 3 November, 2011.

SNORT: http://www.snort.org/. Accessed 18 October, 2011.

SPARQL QUERY LANGUAGE FOR RDF: http://www.w3.org/TR/rdf-sparql-query/. Accessed 21 January, 2012.

SPYNS, P., MEERSMAN, R. and JARRAR, M. (2002): Data modeling versus Ontology engineering. *ACM SIGMOD Record* 31(4): 12–17.

UNDERCOFFER, J., JOSHI, A. and PINKSTON, J. (2003): Modeling Computer Attacks: An Ontology for Intrusion Detection. VIGNA, G., JONSSON, E. and KRUEGEL, C. (Ed.), *The Sixth International Symposium on Recent Advances in Intrusion Detection*, Springer.

VOROBIEV, A. and BEKMAMEDOVA, N. (2007): An Ontological Approach Applied to Information Security and Trust. *18th Australasian Conference on Information Systems (ACIS 2007)*, Toowoomba, Queensland, Australia.

VOROBIEV, A. and BEKMAMEDOVA, N. (2010): An Ontology-Driven Approach Applied to Information Security. *Journal of Research and Practice in Information Technology* 14: 61-76.

W3C SEMANTIC WEB WEB ONTOLOGY LANUAGE: http://www.w3.org/2004/OWL/. Accessed 23 January 2012.

XU, H., XIAO, D. and WU, Z. (2009): Application of Security Ontology to Context-Aware Alert Analysis. *Eighth IEEE/ACIS International Conference on Computer and Information Science (ICIS 2009)*, Shanghai.

Biographical Notes

Lisa Frye *is a professor at Kutztown University of Pennsylvania and chair of the Computer Science and Information Technology Department. Her research interests include networks, network security, ontology, and computer science education. She received a Ph.D. in computer science from Lehigh University. Her previous industry experience includes network server manager at Kutztown University, systems engineer at Electronic Data Systems and system support analyst at Unisys Corporation.*

Lisa Frye

Liang Cheng *is an associate professor at Lehigh University. He has advised six Ph.D. students to their graduation and two postdocs. Dr Cheng has been the Principal Investigator (PI) and a Co-PI of fifteen projects supported by the U.S. National Science Foundation, Defense Advanced Research Projects Agency, Department of Energy, and state and industry sponsors. His research focuses on wireless ad hoc networks and system modeling and analytics.*

Liang Cheng

Dr Jeff Heflin

Dr Jeff Heflin *is an associate professor in the Department of Computer Science and Engineering at Lehigh University. His specific research interests include establishing semantic interoperability between heterogeneous information systems, scalable ontology reasoning, and developing formal theories of distributed ontology systems. He has been involved in the design of many important Semantic Web languages, including SHOE, DAML+OIL, and OWL. In 2004, he received an NSF CAREER award to study the theory and algorithms of distributed ontologies. He is a Senior Member of AAAI and an area editor for the Journal of Web Semantics. He was a guest editor for four journal issues and served as co-program chair for ISWC 2012. Dr Heflin received his B.S. in computer science from the College of William and Mary. He received his M.S. and Ph.D. in computer science from the University of Maryland.*