An Ontology-Based System to Identify Complex Network Attacks

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

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

Abstract— **Intrusion Detection Systems are tools used to detect attacks against networks. Many of these attacks are a sequence of multiple simple attacks. These complex attacks are more difficult to identify because (a) they are difficult to predict, (b) almost anything could be an attack, and (c) there are a huge number of possibilities. The problem is that the expertise of what constitutes an attack lies in the tacit knowledge of experienced network engineers. By providing an ontological representation of what constitutes a network attack human expertise to be codified and tested. The details of this representation are explained. An implementation of the representation has been developed. Lastly, the use of the representation in an Intrusion Detection System for complex attack detection has been demonstrated using use cases.**

Keywords- Computer network security; Intrusion Detection System; Ontology

I. 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) [1].

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 [2] developed a methodology to detect complex attacks. The methodology described in [2] is the preliminary design for the formal representation defined in this paper. This methodology was extended by defining the formal representation utilizing ontology [3]. The ontology development was based upon the family of complex attacks identified by attack trees in [2]. The coloring scheme has not been implemented yet; that will become part of the probability of the attack that is part of the future work.

The primary goal of this research was to develop a Traffic-based Reasoning Intrusion Detection System using Ontology (TRIDSO) to detect complex attacks. The first contribution is a thorough explanation of a formal method to represent complex attacks. This will in turn provide an approach to represent generic attacks, which will allow new attacks to be identified. The second contribution is the development of an Intrusion Detection System using ontology to describe specific traffic and attack concepts. TRIDSO represents a new type of IDS and therefore is an important contribution to the development of more sophisticated approaches to intrusion detection. The last contribution is the validation of TRIDSO via use cases.

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 evaluation results. Conclusions and future work are provided in Section 6.

II. 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.

One system that utilizes ontology to aid in intrusion detection is the Reaction after Detection (ReD) Project [4]. 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 [5] to detect intrusions against hosts. This approach is anomaly-based, so baseline behavior of the network is obtained and abnormal behavior 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 [6]. 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 behavioral information through the use of rules.

Vorobiev and Bekmamedova [7] 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.

A multiagent system using ontology was developed for Outbound Intrusion Detection (OID) by Mandujan [8]. 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.

III. System Architecture

There are existing IDSs that examine the 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 [9, 10] 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. The system developed, TRIDSO, was based on all network traffic. This allowed the system to detect all intrusion attempts, regardless if the intrusion was successful or not.

B. System Design

The system design, illustrated in Fig. 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. The built-in reasoner of Jena [11], an ontology development library, was used for TRIDSO.

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.

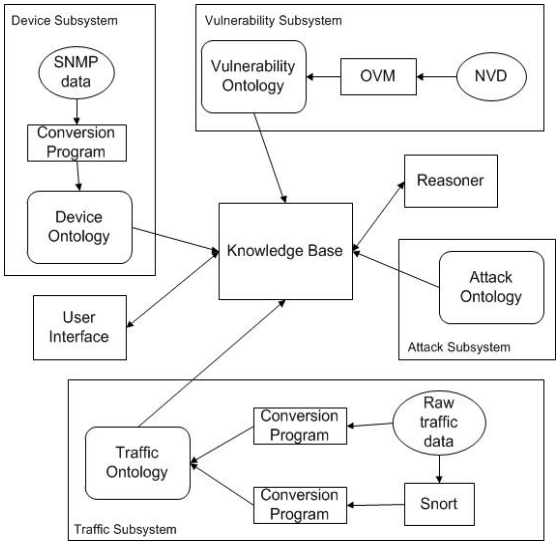

Figure. 1. TRIDSO Architecture (arrows indicate the flow of data)

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. 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.

IV. 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 [12]. For the first phase of development in TRIDSO, only the traffic and attack subsystems were implemented. The ontologies for each of these subsystems are described here.

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 Fig. 2). The primary class is the Packet class, containing the date and time for the packet. This class is then broken down into the various layers according to the Internet Protocol Stack [13]. For instance, the IPPacket class contains instances of all IP packets found in the captured traffic data.

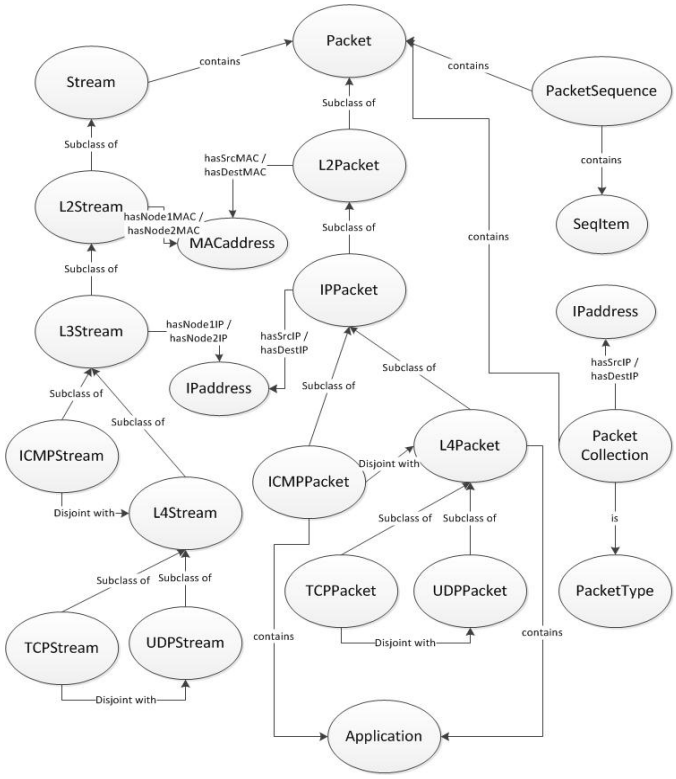

Figure 2. Traffic Ontology Diagram

The Application class contains the application layer protocol and data for ICMP and layer-4 packets (TCP and UDP).

The Stream class of the traffic ontology contains information about past and present flows between two nodes in the network. This includes ARP, ICMP, TCP and UDP flows. The instances in the Stream classes were used to identify specific simple attacks, such as the attacker identifying an existing TCP connection in order to conduct a man-in-the-middle attack. These instances were also used to identify some poison attacks; the existing ARP-to-IP address translations must be known, which were retrieved from the L3Stream class, to identify if an ARP poison has occurred (a new IP address was returned in response to an ARP request for an existing ARP address).

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. 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.

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 remaining packet types in TRIDSO were handled in a similar fashion, by placing restrictions on properties of the TCPPacket or ICMPPacket classes.

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.

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 [14], a query language for RDF [15]. 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 Fig. 3.

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 Fig. 3 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

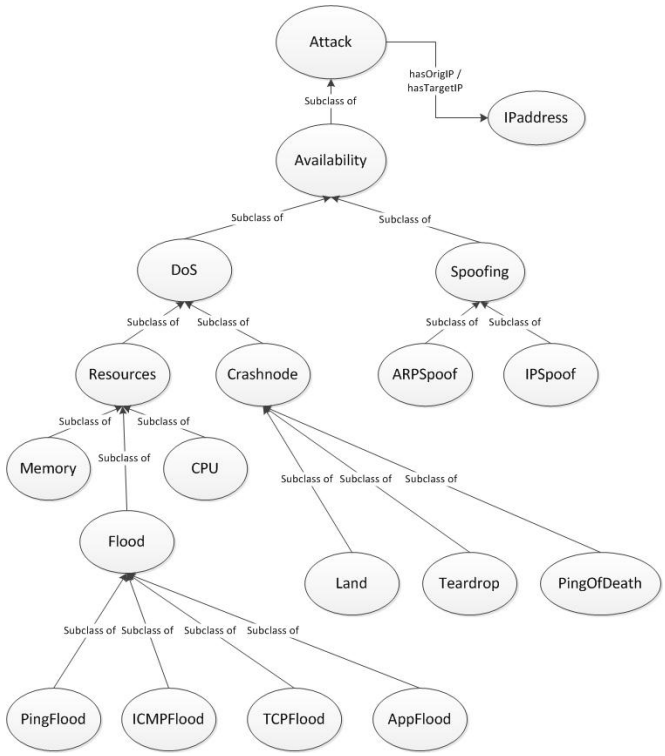

Figure 3. Availability Hierarchy in Attack Ontology Diagram

in Resources, DoS, Availability, and Attack, all from the instances in the PingFlood class. Without the use of ontology, a query for Availability attacks would not return any results.

C. Complex Attack Ontology

Complex attacks (Fig. 4) 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. 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.

This was done using inference with OWL constructs. The property wasAttacked has a range of IPaddress, a domain of Attack, and is the inverseOf hasTargetIP. 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. These instances were identified using the someValuesFrom restriction in OWL.

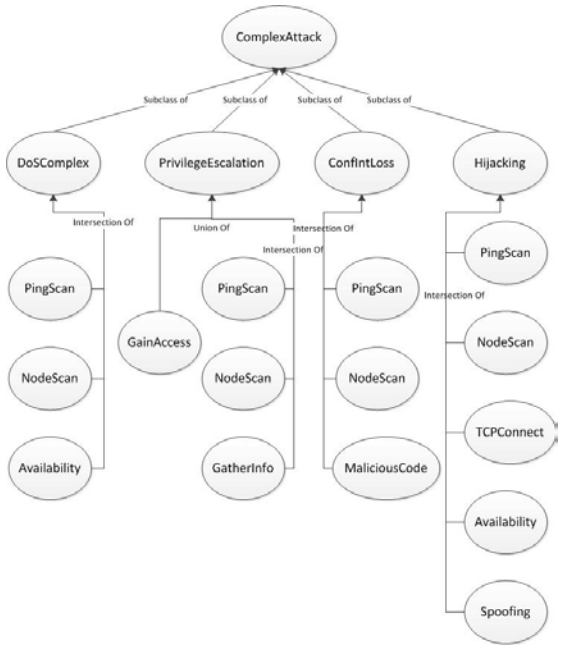

Figure 4. Complex Attack Ontology Diagram

V. 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 developed using Java and Jena.

A. System Implementation

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; each packet was extracted and an instance generated in 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.

The majority of additional instances were either inferred in the ontology or created using SPARQL. When possible, OWL constructs were used, as discussed in the previous section, so the power of ontology could be attained. When OWL constructs were not sufficient, then SPARQL was utilized. SPARQL is a query language, which resembles SQL for databases and includes the ability to dynamically add instances to the knowledge base.

An example of the necessity of SPARQL was 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. 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.

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. Use Cases

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 are explained.

Complex Denial of Service Attack: A complex denial of service attack includes the simple attack steps taken by an attacker to perform a denial of service against a node or network. The attack elements that comprise this complex attack are the attacker finding an available node(s) in the network (a ping scan), possibly finding an open port on the node (a node scan) and then launching some type of a denial of service attack on that device or network (availability attack), such as a ping flood, making the network card unable to process legitimate requests.

To detect this type of complex attack, an instance was created for a PacketCollection of type PingFloodType because ping packet instances were found to the same node. The number of pings to the same node was above a threshold value thus creating an instance of PingFlood in the attack ontology. Determination of the optimal threshold value to use in TRIDSO is continuing work.

Inference in the attack ontology, because of subclass definitions, created an instance of the Availability class. For this test, a static instance of the PingScan and NodeScan were created.

For the complex attack classes, an instance of PingScan, NodeScan, and Availability to the same node, caused an instance of the DoSComplex class, indicating the occurrence of a complex denial of service attack. TRIDSO correctly identified this complex attack.

The Mitnick Attack: A classic complex attack is the Mitnick attack [16]. This complex attack is an example of a hijacking attack. The attack consists of a denial of service attack, originally a Syn Flood attack, predicting the TCP sequence number, and IP spoofing. This example demonstrates how TRIDSO will detect this complex attack.

The Syn Flood and TCP sequence number prediction attacks both utilize the creation, and quick termination, of many TCP connections to the same host. This is detected in TRIDSO by looking at the number and length of TCP connections, which are TCPStream instances in the traffic ontology, and the number of these instances to the same node. If the occurrence

is above a determined threshold, an instance is created for a SynFlood. Because of the similarity in how these attack elements are conducted, this class will be used for both of these attack elements, making no differentiation between the specific attack elements, in this case.

For the IP Spoofing attack element, the Stream class will once again be the key. The L3Stream will include 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 will be used to identify the IP Spoofed correlation as an anomaly, indicating the occurrence of an IPSpoof attack, causing the creation of an instance of the IPSpoof class in the attack ontology.

The existence of a SynFlood, also indicating the possibility of a TCP sequence number prediction attack, and an IPSpoof instance to the same node indicates the possibility of a Mitnick complex attack. Since 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.

VI. 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.

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

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. Determining the optimal threshold value for identifying occurrences of various flood-type attacks, such as a ping flood, will be future work. As the development of the ontologies near completion, system evaluation will be done. 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

[1] A. Fuchsberger. Intrusion Detection Systems and Intrusion Prevention Systems. *Information Security Tech. Report*, vol. 10, issue 3, pp. 134-139, Jan. 2005.

[2] L. Frye, L. Cheng, and R. Kaplan. 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, July 18-21, 2011.

[3] P. Spyns, R. Meersman, and M. Jarrar. Data modeling versus Ontology engineering. *ACM SIGMOD Record*, vol. 31, issue 4, pp. 12-17, Dec. 2002.

[4] N. Cuppens-Boulahia, F. Cuppens, J. E. López de Vergara, E. Vázquez, J. Guerra, and H. Debar. An ontology-based approach to react to network attacks. *International Journal of Information and Computer Security*, vol. 3, issue 3/4, pp. 280-305, Jan. 2009.

[5] J. Undercoffer, A. Joshi, and J. Pinkston. Modeling Computer Attacks: An Ontology for Intrusion Detection. G. Vigna, E. Jonsson, and C. Kruegel (Ed.), *The Sixth International Symposium on Recent Advances in Intrusion Detection*, pp.113-135, Springer, 2003.

[6] H. Xu, D. Xiao and Z. Wu. Application of Security Ontology to Context-Aware Alert Analysis. *Eighth IEEE/ACIS International Conference on Computer and Information Science (ICIS 2009)*, Shanghai, pp. 171-176, June 1-3, 2009.

[7] A. Vorobiev and N. Bekmamedova. An Ontological Approach Applied to Information Security and Trust. *18th Australasian Conference on Information Systems (ACIS 2007)*, Toowoomba, Queensland, Australia, Dec. 5-7, 2007.

[8] S. Mandujano. 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, Dec. 2-3, 2005.

[9] "Snort". Retrieved October 18, 2011, from http://www.snort.org/.

[10] B. Caswell, J. Beale, and A. Baker. Snort IDS and IPS Toolkit. Burlington, MA: Syngress Publishing, Inc., 2007.

[11] "Jena - A Semantic Web Framework for Java". Retrieved January 5, 2012, from http://jena.sourceforge.net/index.html.

[12] "W3C Semantic Web Web Ontology Lanuage". Retrieved January 23, 2012, from http://www.w3.org/2004/OWL/.

[13] J. F. Kurose and K. W. Ross. Computer Networking: A Top-Down Approach, Fifth Edition. New York, Addison-Wesley. 2010.

[14] "SPARQL Query Language for RDF". Retrieved January 21, 2012, from http://www.w3.org/TR/rdf-sparql-query/.

[15] "Resource Description Framework (RDF)". Retrieved November 3, 2011, from http://www.w3.org/RDF/.

[16] S. Northcutt. Network Intrusion Detection: An Analyst's Handbook. Sams Publishing, 2001.