A Collaborative Approach to Situational Awareness for CyberSecurity

M. Lisa Mathews, Paul Halvorsen, Anupam Joshi and Tim Finin
Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, MD, USA
{math1, pmghalvorsen, joshi, finin}@umbc.edu

Abstract— **Traditional intrusion detection and prevention systems have well known limitations that decrease their utility against many kinds of attacks. Creating a new system that collaboratively combines information from traditional and nontraditional sensors to produce new, relevant signatures is one way to deal with these limitations. In this paper, we present a framework that uses this collaborative approach, as well as the details for a network traffic based classifier that shows promise for detecting malicious traffic.**

Keywords- cybersecurity; intrusion detection; situational awareness; information extraction

I. INTRODUCTION

As we incorporate computers into more aspects of our lives, security attacks that target these systems become more invasive and damaging. Intrusion detection and prevention systems (IDPSs) are one way to safeguard the cyber-systems we use, but they have limitations. Current state-of-the-art IDPSs perform a simple analysis of host or network data and then flag an alert. Only known attacks whose signatures have been identified and stored in some form can be discovered by most of these systems. Many times an attack is only revealed after some amount of damage has already been done. Also, traditional IDPSs are point-based solutions incapable of utilizing information from multiple data sources and have difficulty discovering newly published or zero-day attacks. Recent security attacks follow a low-and-slow intrusion pattern where, instead of doing as much damage as quickly as possible, the goal is to remain undetected for as long as possible and slowly weaken a system's defenses. Traditional intrusion detection and prevention systems have difficulty discovering and stopping these types of attacks.

To address these issues, we are developing a semantic approach to intrusion detection that uses traditional as well nontraditional sensors collaboratively [1]. Traditional sensors include hardware or software such as network scanners, host scanners, and IDPSs like Snort and Norton AntiVirus. Potential nontraditional sensors include sources such as online forums, blogs, and vulnerability databases which contain textual descriptions of proposed attacks. After analyzing the data from these sensors, the information extracted is added to a knowledge base that contains rules or policies used to identify the situation or context in which an attack can occur. By having different sources collaborate to discover potential security threats and create additional signatures, the resulting situational-aware IDPS would be better equipped to stop creative attacks such as those that follow a low-and-slow intrusion pattern. This paper describes this collaborative approach to cybersecurity and details a specific module we have built to identify potential attacks using a network traffic flow classifier.

II. BACKGROUND

An Intrusion Detection System (IDS) is a set of tools that runs passively in the background to determine if components of a system, as reflected in the network or host monitoring data, are behaving maliciously. When an IDS runs passively, it notes potential security breaches and logs them or notifies an operator but takes no action to prevent or mitigate the problem. For example, if an IDS detects the unauthorized transfer of packets over the network, it takes no action against the flow of traffic or the hosts on the network. Active systems, referred to as Intrusion Prevention Systems (IPSs), seek to stop malicious behavior and traffic before harm is done. These two systems usually work in conjunction to form and IDPS. Figure 1 shows the general flow to current IDPSs.

The system architecture designed for the collaborative situation-aware IDS [1] comprises data streams from individual sensors that are potentially deployed enterprise wide, and sometimes even across enterprise boundaries. This data can be used directly, or after some analysis, as will be discussed later in this paper specifically for network traffic flow data. The collaborative component is provided by a combination of an ontology, knowledge base, and reasoner working together. The data streams include the traditional and nontraditional sensors and can be viewed as different channels that provide useful information related to an attack. Relevant information and events from these data streams are extracted and represented in the ontology. These are asserted into the knowledge base. The integration and reasoning over this aggregated data enables collaborative detection of complex attacks.

Our ontology is an extension of one developed earlier by our group [2, 3] to describe events related to cybersecurity. The ontology consists of three main classes: 'means', 'consequences', and 'targets'. The 'means' class describes various methods of executing an attack; the 'consequences' class describes the possible outcomes of an attack; and the 'target' class contains the information regarding the system under attack.

This research was partially supported by MURI award FA9550-08-1-0265 from AFOSR. The work of M. Lisa Mathews was supported by the National

COLLABORATECOM 2012, October 14-17, Pittsburgh, United States
Copyright © 2012 ICST
DOI 10.4108/icst.collaboratecom.2012.250794

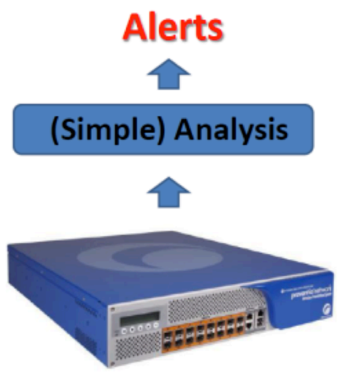

Figure 1. Current state-of-the-art intrusion detection and prevention systems perform relatively simple analyses over a stream of data collected by custom hardware or software systems to detect potential security breaches and alert an operator or users.

The information extracted from the different data streams is asserted using the terms in the ontology. The entities collected from different data sources can be properties of a class in the ontology (e.g., port number, IP address, OS version, hardware platform etc.) or objects of the classes defined in the ontology. The knowledge base is built up by encoding the information as Web Ontology Language (OWL [15]) and Resource Description Framework (RDF [16]) assertions. We serialize these using Notation-3 (N3 [17]) triples of the form (*s*ubject *p*redicate *o*bject) that asserts that the relation *p* holds between *s* and *o*.

For example, Figure 2 shows the free text description from the CVE-2012-2557, which is available from the National Vulnerability Database[1] (NVD). Our text processing module can analyze this description and assert the fact that the software product *Internet Explorer 6* has the *use-after-free vulnerability* into the knowledge base. In our ontology, the 'user-after-free vulnerability' is an instance of the class 'Backdoor', which is a subclass of 'MaliciousCodeExecution', which in turn is a subclass of 'Means' class. The reasoner would be able to automatically deduce that it was looking at the means of some potential attack. The sensor information is used to continuously update the knowledge base.

The reasoning logic module takes inputs from different data streams, the knowledge asserted into the knowledgebase, and rules expressed in the security ontology to infer the possibility of a threat or attack. The rules typically come from domain experts. For example, computer forensics experts detect many complex attacks by combing evidence from various different logs and traces. Such rules can be expressed in our ontology and an appropriate rule language (in our case, Jena [18] rules). The reasoner logic looks at the rules from the knowledge base and information gathered to flag an alert, giving the means, consequences, and targets of the potential attack. The knowledge base that is built up by asserting the ontology is used by these rules to derive chains of implications. Instances are asserted into the knowledge base as events occur. For instance, consider the IE6 vulnerability described in Figure 2. A

[1] http://nvd.nist.gov/

Use-after-free vulnerability in Microsoft Internet Explorer 6 through 8 allows remote attackers to execute arbitrary code via a crafted web site that triggers access to a deleted object, aka "cloneNode Use After Free Vulnerability."

Figure 2. This note from CVE-2012-2557 is a example of the kind of text that can be analyzed to detect concepts, entities, events and relations that are relevant to intrusions.

```
{ IDPS:scannerLog      IDPS:hasBrowser                ?Browser
  IDPS:gatewayLog      IDPS:hasURL                    ?URL
  ?URL                 IDPS:hasSymantecRating         "unsafe"
  IDPS: scannerLog     IDPS:hasOutboundConnection     "true"
  IDPS:WiresharkLog    IDPS:isConnectedTo          ?IPAddress
  ?IPAddress           IDSP:isZombieAddress           "true"}
=>
{ IDPS:system          IDPS:isUnderAttack     "user-after-free
                                               vulnerability"
  IDPS:attack          IDPS:hasMeans          "Backdoor"
  IDPS:attack          IDPS:hasConsequence    "Unautorized
                                                RemoteAccess"}
```

Figure 3. This rule, serialized as N3, asserts RDF triples describing a potential attack based on the presence of triples representing the state of the system and recent events.

rule that accounts for this threat, like the one in Figure 3, could say that if an affected version of Internet Explorer is running (as detected by a host based sensor), that the user has visited a previously unvisited site (as detected by an application level gateway) that has a negative reputation (as reported by commercial providers such as Symantec), and that a connection has subsequently been opened to machine in a known range of zombie addresses (as detected by a Wireshark and SORBS), an attack is likely occurring.

The knowledge base can also be dynamically queried by an analyst using the SPARQL [19] RDF query language. SPARQL queries consist of triple patterns consisting of a subject, predicate and object that are URIs, literals or variables (terms beginning with a '?', along with conjunctions, disjunctions, and optional patterns. If there are any triples in the knowledge base that match the query either as the result of an assertion of a fact or a derivation of rules resulting from the chain of implication, the value of those triples will be returned.

Let us now focus on the sensing modules that form elements of this collaborative system. Traditionally, most of the signatures used by IDSs/IPSs are static and will only look for a very specific pattern in the traffic (i.e. a packet every three seconds), but will miss anything not in that specific signature (i.e. a packet every five seconds). Due to this limitation, a large number of signatures are needed to block a wide range of malicious traffic. Snort comes preconfigured with 16,000 rules and currently has over 20,000 additional signatures available for download [5].

These signatures can be labeled as traffic or data signatures depending on what information is extracted from a packet. Traffic signatures observe patterns in a traffic flow in order to detect anomalies using patterns that fall into two primary categories: header information and flow information. Typical elements of header information used are ports, IP addresses, and

Time to Live (TTL) values. Ports and IP addresses are used to identify a flow, which is a grouping of packets that are sent or received by the same application.

Data signatures examine all of the contents of a packet to detect malicious code. This has an advantage over traffic signatures since some attacks only require very few packets or can hide by mimicking benign traffic. However, the deep packet inspection (DPI) required for these signatures to detect malicious activity is a time-consuming process involving data collection followed by string comparisons. The information produced by this type of inspection could also lead to privacy concerns.

Another limitation of IDSs/IPSs relying heavily on signatures is that they only look for very specific patterns in traffic. New attacks will not be discovered efficiently since the corresponding traffic has not been observed before, meaning the appropriate signature has not been created. Attacks can also be tailored to either mimic legitimate traffic or to circumvent these signatures.

III. Related Work

IDPSs have been around for some time, and new ones continue to appear for commercial use. Some of these are open source, such as Snort, and some are commercially available, such as Norton AntiVirus by Symantec. In order to identify different traffic flows, IDSs generally use signatures that are basically a set of rules that determine whether network traffic can be labeled as benign or malicious.

Many data sources on the Web today are composed of information related to intrusions and attacks in different levels of verbosity. Mulwad et al. [6] described a system for extracting information about vulnerabilities and cyber attacks from different unstructured data sources like vulnerability description feeds (CVE, CCE, CPE, CVSS, XCCDF, OVAL), hacker forums, chat rooms, blogs, etc., and informing the expert about it. Khadilkar et al. [7] explain the importance of a semantic model for information representation and present an ontology for the National Vulnerability Database. They demonstrate the building of a knowledgebase from a structured data set and its usage. Undercoffer [8] presents a host based intrusion detection system, making use of ontological representation of the intrusions and attacks, which performs better than the conventional signature-based intrusion detection system.

In their research Wright et al. (2006) perform traffic classification between different protocols, including FTP, HTTP, HTTPS, SSH, SMTP, etc. They were able to achieve an average true positive detection rate of 99.66% detection rates with an average false positive rate of 1.2% [9]. Using their technique, Zhang et al. (2011) were able to achieve an average detection rate of 92.26% with an average false positive rate of 1.29% [10]. The authors achieved this by collecting data passively on a wireless local area network for 60 seconds and applying a hidden markov model.

Both Auld et al. (2007) and Li et al. (2007) present research that classifies traffic using stateful collection and classification [11, 12]. Stateful, in this context, refers to the fact that the traffic flows consist of both inbound and outbound connections, where stateless connections refer to only a single direction. Auld et al. (2006), using a Bayesian Neural Network and 28 classifiers, were able to achieve an average detection rate of 96.6% with an average false positive rate of 1.02%. Li et al. (2007), using support vector machines and nine classifiers, were able to achieve an average detection rate of 96.92% and an average false positive rate of 6.59%.

IV. Traffic Flow Classifier

We have designed a situationally-aware IDS that gathers data from traditional and non-traditional sensors and performs analysis over this data. This can produce new facts and information from which the context or situation of the system can be deduced. This in turn would be used by rules or policies that would flag an alert during a potential intrusion. In this work, we sought to build upon this framework by incorporating a network traffic flow classifier. Figure 4 depicts the system architecture of the combined frameworks.

One particular sensor is the network traffic data of a machine. We have developed an analysis routine that is in essence a traffic flow classifier – it looks at only the packet headers to decide if the traffic is likely a part of some malicious activity. In this way, the users' privacy is protected because the data portion of a packet is where the sensitive information, such as user names, passwords, email contents, or page contents, will transported. In traditional systems, where the intent is to directly detect a specific attack, not including this information could lead to decreased detection rates if the IDPS misses malicious payloads. However, in our approach this is just one of the pieces of evidence that goes into the collaborative decision making.

A key goal of our work is to develop a generic, high-level approach that can generate evidence for potential intrusions and use that evidence to detect malicious traffic. This is unlike current IDSs, which are looking for very specific forms of attacks, such as botnets or security fuzzers. This research brings together several different techniques in order to detect several forms of attacks. Each of these techniques is used in conjunction with one another to produce a single classification, which is simply a decision on whether the traffic indicates an attack. To do this, we observed the following different patterns in the traffic: the number of unique ports being accessed, the number of lower value ports being accessed, how periodic the packets are, how rapid the packets are being observed, the average time to live (TTL) value, and the standard deviation of TTL values.

While each of these indicators has been used before, this is the first time they have been brought together in a single package and used with a machine learning component. The work done for this paper has produced a proof of concept collector and classifier. Our experiments were performed on a closed network, using several virtual machines (VMs) with the host running the collector and classifier.

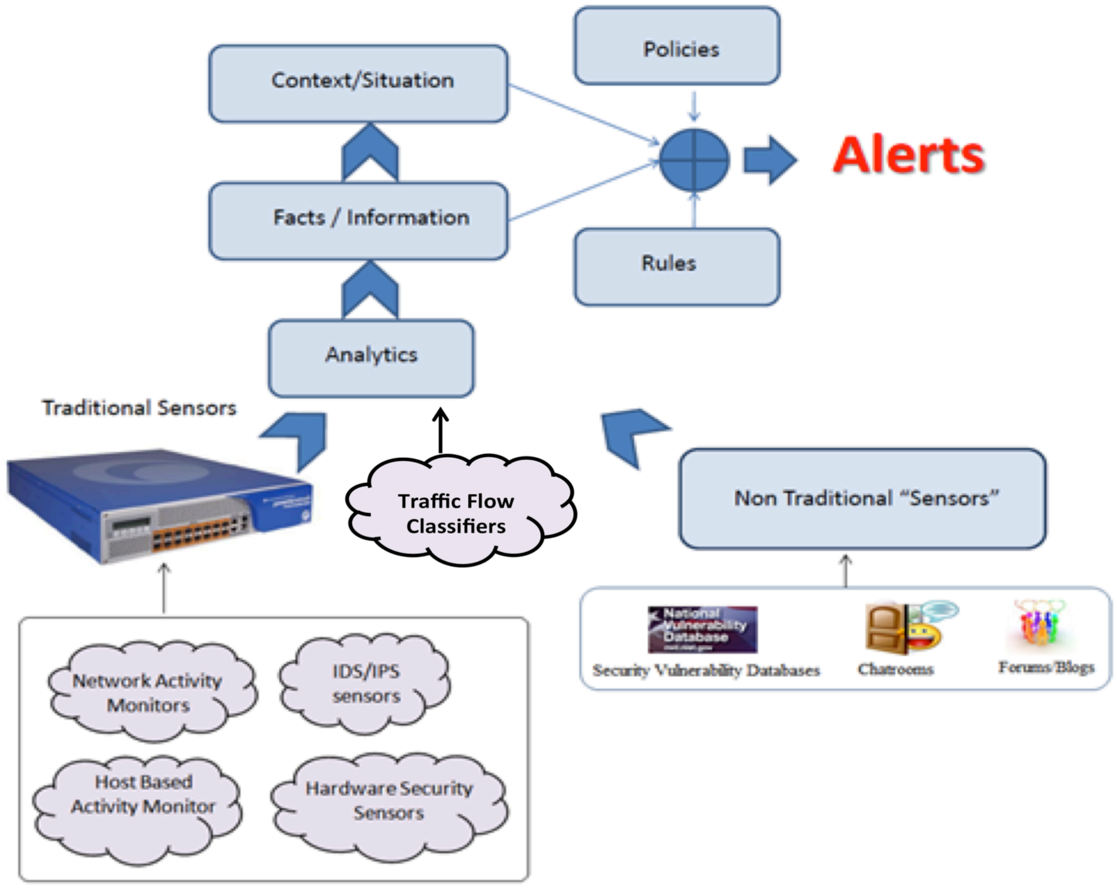
Policies
Context/Situation
Alerts
Facts / Information
Rules
Analytics
Traditional Sensors
Traffic Flow Classifiers
Non Traditional "Sensors"
National Vulnerability Database
Security Vulnerability Databases
Chatrooms
Forums/Blogs
Network Activity Monitors
IDS/IPS sensors
Host Based Activity Monitor
Hardware Security Sensors

Figure 4. Our architecture supports situation awareness by integrating and analyzing data extracted from multiple sources, both traditional and non-traditional. The results are represented in a knowledge-based system and reasoned over to detect potential attacks.

The current techniques that only use traffic signatures [11, 12] are stateful detectors, whereas the method chosen here is stateless. Stateful detectors observe a single flow of data going in both directions. This requires a greater amount of processing, as an inbound packet must be connected to an outbound packet, which may introduce false positives or false negatives into the set, if they are not properly matched. With a stateless detector, our research does not need to make this connection and thus will not include the extra variable of attempting to match inbound and outbound connections.

The IP header contains most of the information that will be needed to classify the traffic as benign or malicious. The version number is needed to determine the format of the header and IP address; the total length is useful since certain attacks require a certain pattern of data, leading to patterns in data size; the TTL is utilized with the understanding that smaller TTLs are indicators of malicious activity; and the source and destination addresses are used to help determine the session and the malicious agent. The source and destination ports are the only fields needed from the TCP header (which are also in the UDP header) to help identify sessions and determine whether many ports are seen in a short period of time, which might indicate malicious activity. Using these protocol headers, packets can be grouped together into sessions, which can then be analyzed to determine if they are malicious or benign.

V. System Setup

A. Network Traffic Simulation

The network used in this research was made up of several VMs and a single host for monitoring the traffic. The VMs were a mixture of Ubuntu 12.04 and Debian 6.0.5 hosted on Virtual Box. The host for the VMs is Ubuntu 12.04. The VMs were used to send and receive malicious traffic and the host was used to monitor and classify the traffic. Benign traffic was sent by some of the VMs, but was mostly obtained from browsing known benign websites, ssh (secure shell, used to established remote command line access) sessions, and games.

Malicious and benign traffic were simulated through several different tools described below. The malicious traffic was simulated using tcpreplay [13] and a Backtrack VM [14], while the benign traffic was simulated using tcpreplay and performing normal tasks that involve sending and receiving packets. All of this traffic is collected by the machine hosting the VMs and is then processed to determine if the traffic is malicious or not. For both the simulated malicious and benign traffic, tcpreplay was automated using python scripts.

The traffic simulated via tcpreplay was automated with two python scripts and a configuration file. The first python script prepared the new pcap files (the pcap files containing the IP address to be used for the test), and the second replays the packets described in the new pcap files. The configuration file describes which pcap files to use and what IP addresses to send them from. Since tcpreplay can simulate traffic from multiple IP addresses while residing on a single host, only one VM is needed to send the different pcap traces.

B. Traffic Collection

To collect the traffic, C++ with a MySQL back-end server was used. C++ was chosen over Java and scripting languages because it is much faster due to the fact that it runs natively after compilation, rather than in a VM. It was also chosen due to its ability to manipulate memory, which allows for fast parsing of the incoming packets that is necessary to keep up with the volume of traffic. C++ was chosen over C for its ability to manipulate strings. For C++ to collect the data, libpcap is utilized, which is the same library used by Wireshark and other traffic collection programs and provides the raw bytes for all traffic on a given interface. The ability of C++ to manipulate memory is then applied to pull out all of the different parts of the packet. Utilizing libmysqlcppconn, these parts are then added to the MySQL back-end server to be saved for further processing. The collection aspect for this research only deals with details from the IP header and the port numbers from the protocol headers.

C. Traffic Learning and Classificiation

The learning and classification of the traffic is done using Java and Weka, an artificial intelligence (AI) training tool that takes formatted data files to train a number of AI algorithms used for classifying data. Weka has a java interface that allows a developer to use Java to implement systems to train and classify data. Using Weka, various AI techniques were implemented in conjunction with several signatures to classify the traffic as either benign or malicious.

The signatures that were used included looking at the ports used, the time to live (TTL) values, the timing of the packets, and the domain of origin for the packets. Regarding the ports, the number of unique ports observed and the number of unique ports specifically observed below port 1024 (the reserved ports) were calculated. A higher number of ports observed indicates malicious traffic. The reason this assumption can be made is that legitimate traffic will remain on a small number of ports for communication, as this is more efficient then switching ports. Malicious users may want to touch more ports to discover which ones are vulnerable.

TTL values are checked for anomalous patterns as well. For the TTL, the number of TTL values below 50, the number of unique values, the mean value, and the standard deviation of the TTL values are calculated. The idea here is that most applications set the TTL value to the highest possible value. Other than mapping a system, there are very few applications where a low TTL value is an advantage. The reason that legitimate traffic would not use a low TTL is to prevent it from dropping packets on their way to the client, whereas a malicious user may want to map the network.

The timing of the packets is also monitored. Here the number of packets within a given amount of time, the average number of packets per second, the total time a session was open, and the mean time between packets are calculated. The idea behind this is that legitimate traffic tends to be more random and spaced out while malicious traffic will be concentrated. It is concentrated because in order to exploit most vulnerabilities, a burst of packets is needed. A malicious user may also want to get the payload uploaded as quickly as possible before their traffic can be blocked or the system patched. Legitimate traffic tends to be slower and more random. Most traffic will be user generated, such as browsing web pages.

After all the data has been classified, python is used to graph the results. The matlibplot library is used along with python to graph the true positive vs. false positive rates of the classification algorithms.

VI. SIMULATIONS & RESULTS

Three primary classifiers where chosen for this research along with sub categories for each. The classifiers use features based on ports, the timing of the packets, and time to live (TTL). The classifiers were first looked at individually, then as a whole.

The traffic was first broken into sessions which were then classified as either malicious or not. These sessions were determined by three factors: the source and destination IP addresses, the source port, and the timing of the packet. Only the source port is used to determine sessions since an application will usually run on a single port but may be routed to multiple ports on the destination for load balancing. Attackers will also touch multiple ports as they attempt to scan a remote host for vulnerabilities. The start and end of sessions are also determined by a 60 second gap between sent or received packets. This research also focuses on stateless detection. Stateful detection determines sessions by using traffic that goes both to and from a host. With stateless detection, this algorithm only needs to observe one-way traffic saving computational time.

The first classifier looked at the ports that were accessed. The results are displayed in Figure 5. Using just the ports as an indicator, high true positive rates are only possible with high false positive rates. This is noted when the highest observed true positive rate of 0.846 only occurs when there is a false positive rate of 0.598. When reducing the false positive rate to acceptable levels, the true positive rate also drops significantly. If the false positive rates are reduced to below 0.100 the true positive rates are reduced to below 0.388.

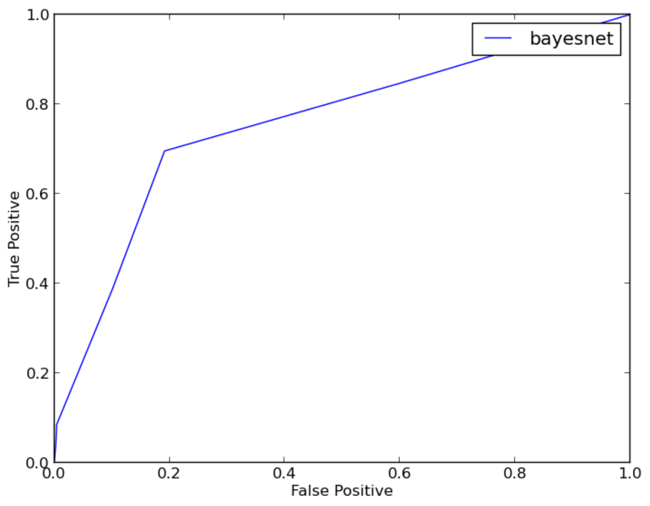

Figure 5. Port Only Classification

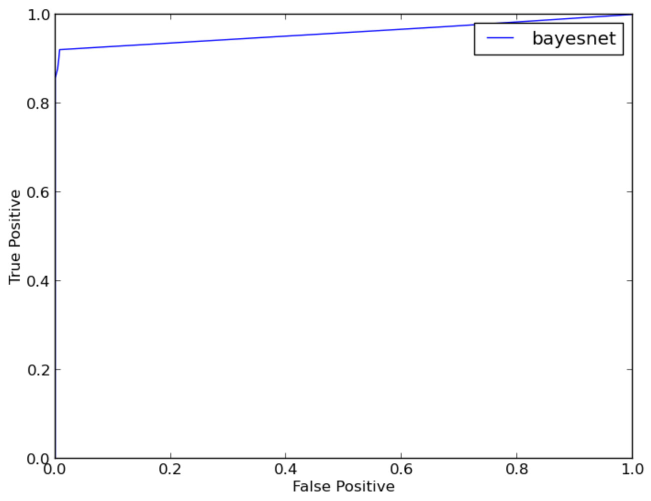

Figure 6. Timing Only Classification

When observing the ports, this algorithm determines the number of unique destination ports and the unique number of reserved ports observed per session. The reserved ports are those below port number 1024. This is to detect scanners that look for security holes and for specific applications running on a remote host. The low detection rates could result from the fact that only scanners will cause this classification to trigger.

Most attacks focus on a single port and a single application running on the remote host. Benign traffic will sometimes span multiple unique ports for load balancing on remote hosts. Between malicious and benign traffic, the number of unique ports is fairly equal at 1.004 unique ports for malicious traffic and 1.338 unique ports for benign traffic. Benign traffic also had a greater number of unique ports below port number 1024 at an average of 0.611 unique ports for benign traffic and 0.187 unique ports for malicious traffic.

Looking at just the timing of the packets, there is a noticeable increase in accuracy. These results are displayed in Figure 6. To achieve the highest observed true positive rates of 0.906 the false positive rates are 0.786. In comparison to the port only classification, a true positive rate of 0.812, there is a false positive rate of 0.567. To reduce the false positive rate below 0.100, the true positive rate is reduced to 0.521.

The timing of the packets was determined by observing the average time between packets, the standard deviation (a unique statistic to this research) of the time between packets, the number of packets observed in the connection, and the total duration of the session. Using these classifiers produces a better detection rate than the ports classifier. The timing classifiers work since initial attacks tend to be very short lived bursts of traffic going in one direction. Previously, it was stated that this is stateless detection, but the single direction is important since benign traffic tends to have to wait for acknowledgments periodically which will slow the transmission timing. Benign traffic sessions also tend to be longer and contain more packets.

Malicious traffic tends to have a more regular interval between packets compared to benign traffic. The average standard deviation for malicious traffic is 0.209 seconds, and the average standard deviation of benign traffic is 3.334 seconds. Malicious traffic tends to be more consistent for the following two reasons. First, the attacker will send a large amount of traffic all at once aiming to exploit a target as fast as possible. The second reason is that if a host is calling back to an attacker, the traffic involved tends to be fairly evenly spaced so the attack knows when the host will call back. Benign traffic tends to be more random as it usually is determined by human interaction.

The third classifier used observed the TTL values and continues to show improvements over the other two, as shown in Figure 7. To achieve the highest true positive rate observed of 0.972, there is an observed false positive rate of 0.671. To compare with the other classifiers, achieving a false positive rate of less than 0.100 means the true positive rate is reduced to 0.921. The TTL values showed the greatest accuracy in detection among the three primary classifiers. This classifier observed the TTL value of the packet by calculating the number of unique TTL values, the average TTL value, and the number of TTL values below 20 hops. Looking at the raw data, the only real classifier that played any significant role was the average TTL value. The majority of the sessions only had a single TTL value used. Simulated malicious traffic for this research contained an average TTL value of 64.647 hops, while benign traffic had an average of 72.805 hops.

After combining all three classifiers, there is a significant increase above the others individually. These results are displayed in Figure 8. To achieve the highest true positive rate observed of 0.999, the false positive rate is 0.742. Compared with the individual classifiers with a true positive rate of 0.846, the false positive rate is 0.033. This is the lowest false positive rate of the individual classifiers. To achieve a false positive rate less than 0.100, the true positive rate is 0.933.

Putting all three classifiers together yields the greatest accuracy. Each classifier provides values that are unique to different types of attacks and different types of benign traffic. By combining the values in each of these classifiers, a more specific signature is created for benign and malicious traffic. The more specific the signature, the greater the detector accuracy.

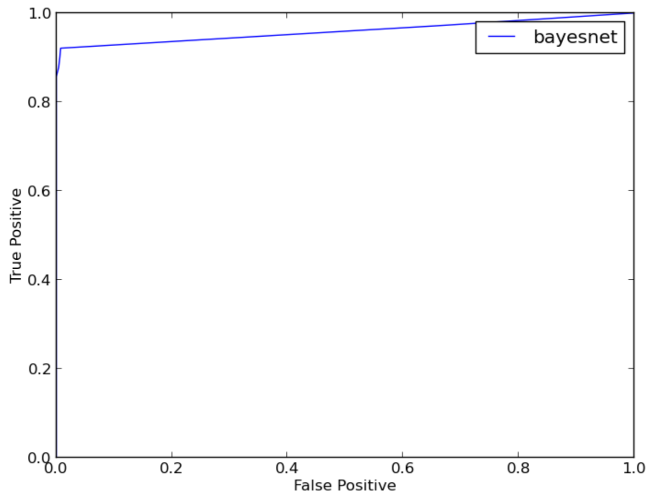

Figure 7. TTL Only Classification

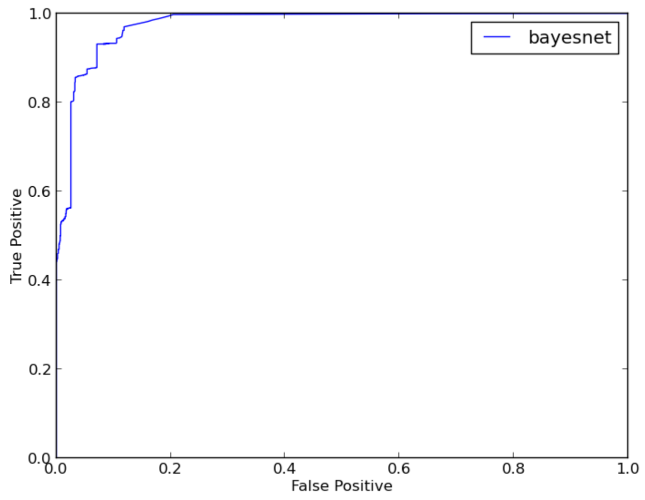

Figure 8. Combining Port, Timing, and TTL Classification

VII. Conclusion

We presented a collaborative approach to detecting attacks that integrates information from multiple sources and represents it in a knowledge base. Each source is analyzed, and corresponding facts are described using terms from an ontology which are inserted into a common knowledge base. These facts, along with rules that are similar to forensic analysis, are used to detect potential attacks. We also described a new analysis of traffic flow information to flag potential attacks using parameters including the number of destination ports, the number of reserved ports observed per session, the timing of packets, and the TTL values of packets seen in incoming network traffic.

Compared to existing approaches, we use less information, which is the result of the choice to make this algorithm stateless. This means it is only collecting and analyzing the traffic moving in one direction, helping to reduce detection time as it requires less processing to not have to relate incoming and outgoing packets. Compared to others, this new algorithm is also more generic, suggesting when traffic is potentially malicious and indicative of an attack. This fits into our collaborative system, since this is just one of the sources of evidence used to detect a potential attack. Several of the other proposed techniques observe specific protocols or search for specific types of attacks. Further, by looking at stateless information of network traffic flows, and not examining the data payloads, we are also more respectful of privacy. Despite these design choices that have the potential to make detection less accurate, we obtain a true positive rate comparable to a recent work by Zhang et al. [10] (92.45% vs. 92.26%) though our false positive rate (7.09% vs. 1.29%) was a bit higher.

References

[1] S, More, M. Mathews. A. Joshi, and T. Finin; , "A Knowledge-based Approach to Intrusion Detection Modeling," *IEEE Symposium on Security and Privacy Workshops,* pp.75-81, May 2012.

[2] J. Undercoffer, A. Joshi, T. Finin, and J. Pinkston, "Using DAML+OIL to classify intrusive behaviours," *The Knowledge Engineering Review*, vol. 18, pp. 221– 241, 2003.

[3] J. Undercoffer, A. Joshi, and J. Pinkston, "Modeling Computer Attacks: An Ontology for Intrusion Detection," in *Proc. 6th Int. Symposium on Recdent Advances in Intrusion Detection*. Springer, September 2003.

[4] "Microsoft Internet Explorer 6 through 8 vulnerability cve-2012-2257", http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-2557.

[5] Sourcefire. Snort. http://www.snort.org/, 2012.

[6] V. Mulwad, W. Li, A. Joshi, T. Finin, and K. Viswanathan, "Extracting Information about Security Vulnerabilities from Web Text," in *Proc. of the Web Intelligence for Information Security Workshop*. IEEE Computer Society Press, pp. 257-260, Aug. 2011.

[7] V. Khadilkar, J. Rachapalli, and B. Thuraisingham, "Semantic web implementation scheme for national vulnerability database," Technical Report UTDCS-01-10, Dept. of Computer Science, Univ. of Texas at Dallas, 2010.

[8] J. Undercofer, "Intrusion Detection: Modeling System State to Detect and Classify Aberrant Behavior," Ph.D. dissertation, Univ. Maryland, Baltimore County, February 2004.

[9] C. V. Wright, F. Monrose, and G. M. Masson, "On inferring application protocol behaviors in encrypted network traffic", *J. Mach. Learn. Res.*, vol. 7, pp.2745-2769, Dec. 2006.

[10] F. Zhang, W. He, X. Liu, and P. G. Bridges, "Inferring users' online activities through traffic analysis," in *Proc. of the fourth ACM conf. on wireless network security*, New York, NY, 2011, pp. 59-70.

[11] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian neural networks for internet traffic classification," *Neural Networks, IEEE Transactions on*, vol.18, no.1, pp.223-239, Jan. 2007.

[12] Z. Li, R. Yuan, and X. Guan, "Accurate classiffcation of the internet traffic based on the svm method," in *Communications, 2007. ICC '07. IEEE International Conference on*, pp. 1373-1378, June 2007.

[13] Tcpreplay. Tcpreplay. http://tcpreplay.synfin.net/wiki, 2010.

[14] BackTrack. Backtrack linux. http://www.backtrack-linux.org/, 2011.

[15] OWL Web Ontology Language Overview. http://w3.org/TR/owl-features

[16] RDF. Resource Description Framework. http://www.w3.org/RDF/

[17] N3. Notation 3 Logic. http://www.w3.org/DesignIssues/Notation3.html

[18] Jena. Apache Jena. http://jena.apache.org/index.html

[19] SPARQL. SQARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/