Multi-Level Intrusion Detection System (ML-IDS)*

Youssif Al-Nashif, Aarthi Arun Kumar, Salim Hariri
Center for Autonomic Comp., ECE Dept., the Univ. of Arizona, Tucson, AZ 85721
{alnashif, aarthi83, hariri}@ece.arizona.edu

Yi Luo, Ferenc Szidarovsky
System and Industrial Engineering, The Univ. of Arizona, Tucson, AZ 85721
{luo1, ferenc}@email.arizona.edu

Guangzhi Qu
Department of Comp. Sci. and Engineering, Oakland University, Rochester, MI 49309
gqu@oakland.edu

Abstract

As the deployment of network-centric systems increases, network attacks are proportionally increasing in intensity as well as complexity. Attack detection techniques can be broadly classified as being signature-based, classification-based, or anomaly-based. In this paper we present a multi level intrusion detection system (ML-IDS) that uses autonomic computing to automate the control and management of ML-IDS. This automation allows ML-IDS to detect network attacks and proactively protect against them. ML-IDS inspects and analyzes network traffic using three levels of granularities (traffic flow, packet header, and payload), and employs an efficient fusion decision algorithm to improve the overall detection rate and minimize the occurrence of false alarms. We have individually evaluated each of our approaches against a wide range of network attacks, and then compared the results of these approaches with the results of the combined decision fusion algorithm.

1. Introduction

Recently, the need for an efficient network defense system has become extremely critical. The Internet is pervading almost every aspect of life and business, and along with this exponential growth comes the critical need to secure these systems from unauthorized disclosure, transfer, modification, or destruction is vital. The increase in the number of attacks and their complexity is due to an increase in the number of applications with vulnerabilities and the number of attackers equipped with fast networks and processing units. Cyber attacks can spread very rapidly. For example, Botnet is a system where bots spread by installing themselves on internet-connected computers. These bots respond to external commands and then execute coordinated attacks. In 2004, a large Botnet containing about 25,000 bots was capable of transmitting junk data at a combined rate of 5 Gbps, enough to take a company network offline. A Botnet larger than a million nodes is capable of sending 22 to 24 Gbps of data. Complex network attacks like these present a significant threat to the security of information infrastructure and can lead to catastrophic results. Network attacks typically exploit vulnerabilities in networks, system software and protocols. For example, some attacks misuse network resources' limitations, protocol vulnerabilities, or application vulnerability to reach their goals. Furthermore, these attacks also vary in their speed, complexity, and dynamicity.

Network defense systems, whether they are signature based or anomaly based, can be classified according to the attacker's misuse type. The first category of network defense systems detects misuse of network resources' limitations (e.g., probing, flooding, and network based DOS). Those defense systems use flow level information to make decisions and detect the attacks such as PeakFlow X, Mazu Profiler, and the AND system [27]. The second type focuses on detecting attackers' misuse of protocol's vulnerabilities [29]. In this type, the system detects attacks using information collected from the protocol headers. Current detection systems from this type are mainly statefull firewalls (e.g., IPTABLES, Cisco PIX, and Linksys WRT54GS) and it protects against attacks such as DoS attacks and SYN flooding attacks. It is worth to mention that the statefull firewall has compatibility problems with the use of TCP window scaling which is shipped with the new OS such as Microsoft Vista or Linux with kernel 2.6.8 and above [36] and [37]. The last type of network defense systems is the one that detects misuses of application's

*The research presented in this paper is supported in part by National Science Foundation via grants numbers CNS-0615323 and IIP-0758579, and by Electronic Warfare Associates, Inc. via the grant number ewagsi-07-LS-0002, and it is conducted as part of the NSF Center for Autonomic Computing at University of Arizona.

vulnerabilities by analyzing information collected from the packet payloads. The most known system of this type is SNORT [38]. In most current network defense systems [27],[38],[31], each packet is inspected to find attack signatures in the payloads (e.g., buffer overflow exploits).

In this paper we present a multilevel intrusion detection system (ML-IDS), which is capable of detecting any type of network attacks. The current implementation of ML-IDS uses two types of attack detection techniques: flow based and protocol based techniques. The flow-based approach employs a set of rules that, if violated, flags the traffic flow as being anomalous. The protocol behavior approach views network protocol operation as being a finite state machine, and flags illegal sequences of state transitions as being anomalous. The current protocol behavior analysis module is not affected by the use of TCP window scaling. A fusion module that implements the least square algorithm is used to combine the decision produced by each type of anomaly analysis and thus significantly reduce the attack detection error. To automate the control and management activities of the ML-IDS (dynamic change in the monitoring, configuration and action policies), we use Autonomia's Component Management Interface and Component Runtime Manager to autonomize the ML-IDS. We have successfully used Autonomia, an autonomic control and management environment being developed at UA, to automate the performance, fault and security management of network centric systems [3][4]. In what follow, we discuss in further detail our approach to implement each module of the ML-IDS and evaluate its performance against a wide range of network attacks.

The rest of this paper is organized as follows. In section 2, we describe related work in the area of network defense and compare our work with different approaches. Section 3 presents the ML-IDS architecture. Section 4 illustrates the design and the current implementation of the ML-IDS and its functional components. Experimental results that evaluate the effectiveness and performance of the ML-IDS are introduced in section 5. Finally, in section 6 we conclude, and present future research activities.

2. Related Work

Intrusion detection can be broadly classified as signature based, classification based and anomaly based systems. The last two techniques are significantly different from the signature-based approaches in that they do not rely on a database of fixed 'signatures' to identify the malicious activity.

Signature-based detection techniques such as SNORT [38], USTAT[2], NSTAT[3] and P-Best[4], are limited in that they cannot detect new attacks. Also, once a new attack is identified, there is a significant time-lapse before the signature database is updated. Also this type of techniques is well known for its high number of false positive alerts.

Classification-based detection techniques on the other hand have normal and abnormal data sets, and use data mining techniques to train the system [5][6][7]. This creates fairly accurate classification models, when compared with signature-based approaches and they are thus extremely powerful in detecting known attacks and their variants. However, they are still not capable of detecting unknown attacks. ADAM [39] uses a combination of association rules mining and classification to discover attacks in a TCPdump audit trail. Another approach is the use of rare class predictive models such as Credos [8], PN-Rule [9], SHRINK [12] and Smoteboost[10]. [39][11] use association rules to detect intrusions, while others use cost sensitive modeling. SHRINK [12] uses a combination of rare class prediction and cost sensitive modeling for misuse detection.

Anomaly-based detection techniques build a model of normal behavior, and automatically classify statistically significant deviations from the normal as being abnormal [13][14][15]. The advantage of this approach is that it is possible to detect unknown attacks. However, there is a potential for having a high rate of false positive alarms generated when the knowledge collected about normal behaviors is inaccurate. Supervised learning approaches to anomaly detection involve training a system on a known set of normal data and testing with a different data set to determine whether the new data is normal. Examples of such techniques are IDES [16], NIDES [17], EMERALD [18]. [19] and [20] estimate parameters of a probabilistic model over the normal data and compute how well new data fits into the model. Unsupervised techniques, such as those based on statistical approaches [21], clustering [22], outlier detection schemes [23] and state machines [24], on the other hand, detect anomalous behavior without training data. In [25], Shon, T. et. al. use a hybrid approach that combines supervised and unsupervised learning mechanisms to perform anomaly detection.

In ML-IDS, each of the individual levels can use any of the detection techniques discussed above. In the current implementation of ML-IDS, both classification based detection and anomaly based detection have been used. Since in our system we are focusing on automating the process of detection and protection, the signature based detection has been avoided due to its tendency to have high amount of false positive alerts.

3. ML-IDS Architecture

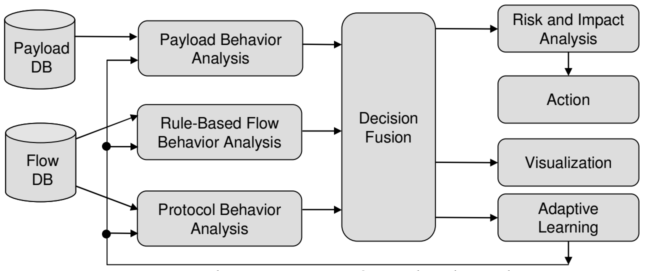


Figure 1. Architecture of multi-level IDS

In what follows, we give an overview of the ML-IDS modules. The main modules (see **Figure 1**) are the Online Monitoring and Filtering, Multi-level Behavior Analysis, Decision Fusion, Action, Visualization, and Adaptive Learning.

3.1. Online Monitoring and Filtering Engine

Our assumption is that any network attack causes a certain anomalous behavior scenario, and it is imperative to be able to accurately identify this anomalous behavior. To meet this challenge, ML-IDS utilizes decision correlation metrics to measure the attributes of the network that will most likely identify an attack in progress. This method greatly reduces the number of features to be monitored and the size of monitored data, and thus decreasing the impact of the monitoring modules on the network's operation. The online monitoring module uses existing software tools to monitor network and host traffic. For simplicity, the online monitoring modules can be viewed as databases.

3.2. Multi-level Behavior Analysis

The main objective of using multiple levels to analyze the behavior of network traffic is to decrease the incidence of false alarms, while achieving a high detection rate. During each observation period, T, Rule-based behavior, Protocol behavior and Payload analysis are used to detect anomalous events in network operations at different level of granularities. If an anomalous behavior is detected by any of the three systems, then an alert is sent to the Decision Fusion module. **Figure 2** shows the operation of ML-IDS, where network behaviors are inspected on all levels of granularity. Payload behavior is inspected to detect abnormal behaviors caused by malicious codes and application misuse. Flows are inspected to detect traffic abnormal behaviors. Headers are inspected to insure that the packets in the network are not exploiting protocol vulnerabilities and strictly follow protocol state transitions.

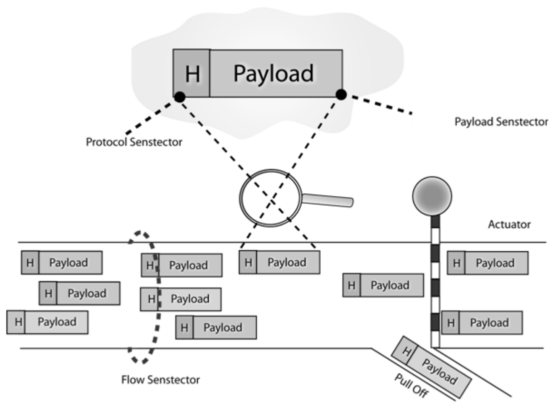


Figure 2. Multi-level inspection

a) Rule-based Flow Behavior Analysis module uses a set of rules that are generated using a supervised learning approach to detect network attacks. We use extensive network traffic to train the rule-based flow detection module about how the network should behave normally with respect to the identified features for different observation windows. The use of multiple observation windows provides the information required to detect a wide range of network attacks with varying rates.

b) Protocol Behavior Analysis module inspects protocol packet headers (e.g., TCP, UDP, or ICMP) to detect anomalies. Every network protocol can be modeled as a sequence of finite state transitions that defines its normal behavior. Thus, any break in the rules of the state machine is considered as malicious behavior.

c) Payload behavior Analysis module inspects payloads to detect malicious codes or application misuse. The payload data is filtered, reduced for fast look-up, and the system is trained with a complete profile of normal network traffic within an observation window.

3.3. Decision Fusion Module

This module fuses the decisions produced by each type of behavior analysis modules. With a large number of alerts being generated by each level of the behavior analysis, it is necessary to fuse the information generated in order to increase the detection rate and also reduce the overhead of processing false alarms. As will be explained later, least square linear regression algorithm is used to fuse the decisions generated by each type of behavior analysis modules.

3.4. Risk and Impact Analysis Engine

When the risk and impact analysis engine receives one or more alerts from the Behavior Analysis Module, it determines what part of the system will be affected by the detected attack. It also consults with its knowledge repository to determine the impact of each protection action on the network operations; it determines the impact of shutting down completely a port, router, or other network components and how does that compare to impact of the attack itself.

3.5. Action Engine

The action engine is used to proactively prevent or mitigate the impact of attacks. Once the action engine receives the result from the risk and impact analysis, it consults with its knowledge repository to determine the appropriate automatic actions to be executed. Those automatic actions are dependent on the type of network resources and their capabilities.

3.6. Visualization module

This module is used to give administrators a better understanding of the activities undergoing in the network. It also allows them to interact with the network resources and inject their own actions in the system. These actions could be manual or semi-automated.

3.7. Adaptive Learning

The ultimate goal of our ML-IDS is to use an unsupervised learning algorithm to identify changes in network operations. These changes may be significantly different from the current baseline models of normal operations. If there is a discrepancy in the decisions given by each level, then the adaptive learning module will use this discrepancy to re-train the errant system in real time. The adaptive learning module will use supervised learning to generate a new set of rules that model the current normal network behaviors that will be used to re-train the Rule-based behavior analysis system. For the Payload and Protocol behavior systems, the adaptive learning module will add the mis-detected patterns and expand the normal and the abnormal profiles repository. In addition, this module will perform a comprehensive analysis of network, host, and payload traffic data that might lead to the discovery of a new anomalous behavior that was not detected by one or more of the levels. As a result of this analysis, the supervised learning module may be activated again to generate new rules, or to expand the trained profiles so as to capture new types of anomalous behaviors. The system administrator can also manually train the supervised learning module on newly discovered attacks that were not detected by the system.

4. ML-IDS Modules

In this section, we present the currently implemented modules of the ML-IDS.

4.1. Monitoring and Filtering Engine

We perform online monitoring and filtering to collect traffic data, and to identify an appropriate set of network features to be analyzed. This is essential to keep the overhead and the impact on network operations at a minimum, while being able to accurately detect attacks. Feature selection is also needed to remove redundant information and hence reduce false positive alarms since redundant information is a source of noise for the learning algorithm [26]. We use two correlation metrics to select features based on the desired decision variable: Decision Independent Correlation (DIC) and Decision Dependent Correlation (DDC) [26]. DIC is the ratio between the mutual information of two features and the entropy of one feature. DDC is a correlation metric to quantify the information redundancy between features and with respect to a decision variable. We use the feature selection algorithm that we have developed in AND prototype to identify the best features required for efficient anomaly detection [27]. The algorithm first uses information theoretic measures to remove irrelevant information from among the various network features. It then eliminates redundancy from the features to be selected.

4.2. Multi-level Behavior Analysis

4.2.1. Rule-based behavior analysis

The rule-based behavior analysis system uses an anomaly-based approach to detect network attacks. This system observes network traffic at the flow level, which is at a higher granularity than our protocol or payload behavior analysis systems.

a) Training:

In order to build a baseline model for normal network behavior, we applied a rule learning algorithm (RIPPER) [40]. This algorithm efficiently constructs the baseline models. For instance, for 300K network traffic records, it took about 4 hours to generate the baseline model consisting of around 120 rules.

b) Detection

During the detection stage, the filtered network flows' records are concurrently analyzed using several observation windows. For each window, we have one module to analyze and correlate the network records with respect to its observation window time t_w. In the current implementation of this module, we use three windows to analyze network traffic.

In each window, network features are analyzed as a time series of consecutive t_w windows. All connections falling within one window and having the same source address, destination address, and destination port are aggregated together into one aggregated network flow. This process significantly reduces the number of connections analyzed for anomaly detection.

In addition, the time series analysis of aggregated network flows with respect to multiple windows significantly improves the detection rate. Assume that for an attack to succeed, steps S_a, S_b, S_c, S_d need to be taken, and each of these steps will make the network go into states B_a, B_b, B_c, B_d, respectively (**Figure 3**). We consider that each of the four steps behaves normally within t_w. Thus, even if two or three of these events are analyzed, their combined behavior will be considered normal if they are analyzed within separate windows, i.e., ($B_a \cup B_b$, $B_a \cup B_c$, $B_b \cup B_c$, and $B_a \cup B_b \cup B_c$.)

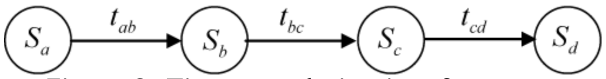
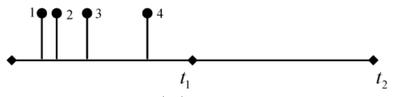

$$\begin{tikzcd} S_a \arrow[r, "t_{ab}"] & S_b \arrow[r, "t_{bc}"] & S_c \arrow[r, "t_{cd}"] & S_d \end{tikzcd}$$

Figure 3. Time correlation in a four-step attack

However, when all four events $B_a \cup B_b \cup B_c \cup B_d$ are analyzed within one window, the anomaly caused by the attack will be detected. Hence, this type of attack will be detected only if the size of the time window $t_w \geq t_{ab} + t_{bc} + t_{cd}$. To further improve the accuracy of our detection mechanism, we have used several windows to analyze the network flows. Some attacks are launched at a very high rate and thus can be detected by smaller windows, while slower types of attacks need much larger windows in order to detect the anomaly triggered by this type of attack. For example, we analyze network flows collected within two windows t_1 and t_2, where $t_2 = 2 \cdot t_1$. as shown in **Figure 4**.



(a)

Event	Src. Bytes	Dest. Bytes
1	10	10
2	30	20
3	40	30
4	60	40

(b)

Figure 4. Time window example

We assume that four network flows, or events, with the same source address, destination address, and destination port have occurred in the first instance of t_1. The anomaly condition for t_1 is $\sum src\_bytes > 120$ and $\sum dest\_bytes < 150$. And the anomaly condition for t_2 is $\sum src\_bytes > 240$ and $\sum dest\_bytes < 300$. It is obvious that the anomaly caused by this attack can be detected using the first time-window t_1 and not by the second time-window t_2.The choice of the window intervals has been done based on a combination of Monte Carlo and Bi-Section methods.

4.2.2. Protocol behavior Analysis

Network protocols define a set of rules, conventions and specifications for communication between network devices. The protocol behavior for anomaly-based detection of network attacks involves building a complete profile of normal network traffic that implements a particular protocol. In this work, we apply our approach to analyze the anomalous behavior of the Transmission Control Protocol (TCP) [30].

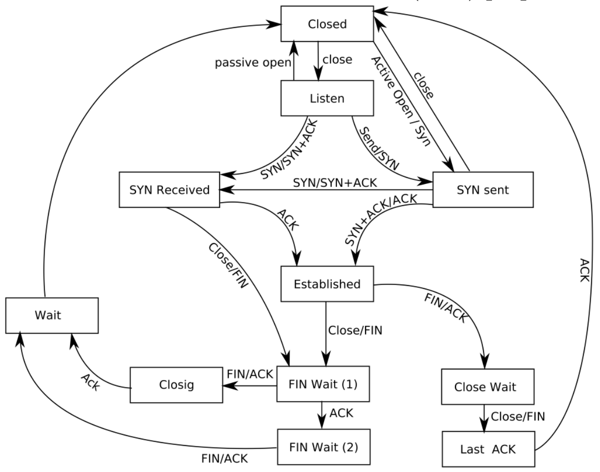


Figure 5. TCP state diagram

The Transmission Control Protocol is a reliable transport protocol used to establish a connection between two communicating processes. Our approach exploits the inherent properties of the TCP packet header. In previous work, the operation of the TCP protocol has modeled the protocol as a finite state machine [29]. In our Protocol behavior analysis approach we focus on analyzing protocol transition sequences of length n during a window interval. When an attack exploits the TCP header, it typically generates illegal state transitions that can be detected by our protocol transition analysis. For example, in order to establish a TCP connection, the protocol needs to complete the following state transitions. Listen – SYN_Sent – SYN_Received – Established. This involves the following sequence of actions SYN – SYN-ACK – ACK (see **Figure 5**). In TCP SYN flood attack, there are several SYN – SYN_ACK transitions, which result in a large number of half-opened connections. However, the SYN – SYN-ACK – ACK state transition is not completed. We study these transitions using n-gram analysis [33],[34], and a sliding time-window, to detect any attacks that are based on using TCP protocol as soon as they occur.

In Anagram[31], Stolfo et. al. used a mixture of higher order n-grams to detect anomalies in the packet payload. We use n-gram analysis to study the behavior patterns of TCP flags for distinct connections over a specified time window. N-gram analysis is a technique that uses a sub-sequence of n items from a given sequence that has been used in studying system calls [32-34]. In our system, we generate n-grams by sliding windows of length n over a stream of packets, and collecting the sequences of TCP flags per network connection. Analysis of the state transitions that occur during a given window allows us to accurately detect any type of TCP protocol attacks. We are currently extending this approach to detect any type of attacks that use UDP and ICMP protocols.

a) The Bloom Filter approach

We have extended the use of a Bloom filter [35] to efficiently detect the occurrence of intrusions. A Bloom Filter is a probabilistic data structure that is space as well as memory efficient. Its primary function is to represent a set of elements and to efficiently support membership queries.

There is a clear trade-off between the size of the Bloom filter and the probability of a false positive. When $\boldsymbol{n}$ keys are inserted into a bit-vector of length $\boldsymbol{m}$ using $\boldsymbol{k}$ hash functions, the probability that a particular bit is still 0 is:

$$p=(1 - 1/m)^{kn}$$

Therefore, the probability of the occurrence of a false positive is:

$$P = (1-(1 - 1/m)^{kn}))^{k}$$

This value can be approximated to:

$$P = (1- e^{-kn/m})^{k}$$

This value is minimized for a value of

$$k = \ln 2 \times m/n$$

The value of $\boldsymbol{k}$ determines the number of hash functions to be used, and it needs to be an integer. In order to calculate the number of hash functions needed to efficiently model our normal traffic, while still maintaining a high detection rate, we chose the false probability rate of the bloom filter to be 0.01%. We used seven distinct hash functions based on the number of elements in our training data set, and our desired false positive rate of 0.01%. We chose seven one-way additive and rotative general-purpose string hashing algorithms in the training as well as the testing phases of our system [41].

b) Training

In the training phase, we collected several types of normal TCP traffic, in order to build the normal profile. We use an open source software tool to collect traffic packets. We then filter the header information, and collect the TCP flags for every connection established within a specified time window. The sequences of TCP flags for every connection within a specified time window are then framed into higher order n-grams. This allows us to examine the TCP state transitions for each connection at different granularities. By using different window sizes, as well as varying the gram length, it is possible to detect very fast, or slow-occurring transition violations.

The window-based training is performed using a window size of ten seconds, which was found to be optimal for detecting most types of network attacks. During the training phase, $\boldsymbol{n}$-grams of a wide range of normal TCP header patterns were collected during window $\boldsymbol{T}$.

The $\boldsymbol{n}$-gram patterns within the specified time-window are the input to the Bloom Filter. For every connection within the time-window, the patterns were fed to the hash functions in order to generate index locations to uniquely identify them. After aggregation of the collected traffic data, we had approximately 3.5 million patterns that were divided into over 100000 records based on the source and destination of the patterns within the specified time interval. We also generated 12 types of TCP attacks over the same time period, for every connection within the specified time window. We collected almost 1.5 million patterns, and they were also divided into approximately 100000 records based on the source and destination of the patterns within the specified time interval. It was clearly obvious that the distribution of malicious patterns is distinct, and is greater in volume and intensity when compared with normal patterns. These patterns were fed two distinct Bloom filters, one for the normal patterns and the second for the abnormal patterns. The filter sizes were computed for a desired false positive rate of 0.01% as was discussed earlier.

c) Detection Algorithm

Once the training phase was complete, we first tested our system off-line using test data, and then by deploying it for real time detection. The packets are collected, and the sequences of TCP flags for every connection within a specified time window are framed into higher order ***n***-grams. The ***n***-gram patterns for each connection within the specified time-window are the input to the Bloom Filter. For every connection within the time-window, the patterns were fed to the hash functions in order to generate index locations to uniquely identify them. These indices are then looked up in both the normal, as well as abnormal Bloom filters.

For example, in the SYN flood attack, when there is a sequence of repeated SYN – SYN_ACK transitions, without an ACK transition, it will be flagged as being anomalous. The detection algorithm is described below in **figure 6**.

```
NC:      number of normal patterns
AC:      number of abnormal patterns
UC:      number of unknown patterns
P:       Pattern
N:       Normal
A:       Abnormal
U:       Unknown
t1, t2:  Time window boundaries
T:       Threshold

If ( t1 < time_stamp < t2)
    For each connection
        For each pattern
            If ( p == N )
                NC++
            Else If ( p == A )
                AC++
            Else If (((p!=N) && (p!=A)) ||
            ((p==N) && (p==A)))
                UC++
            score = NC / (NC + AC + UC)

            If (score > T)
                Connection is Normal
            Else if (A>U)
                Connection is Abnormal
            Else
                Further inspection needed
```

Figure 6. Protocol Behavior Detection Algorithm

Each pattern is compared against the normal and the abnormal Bloom filters. The patterns that are present in either the Normal or Abnormal filters are flagged accordingly. Those patterns or sequences that do not occur in either filter are flagged as unknown. For every connection occurring within the specified time window, we then count the number of normal, abnormal and unknown packets. We were able to establish a heuristic threshold based on a connection's score with respect to the threshold. If the sequences in the connection have not been seen before, we perform further analysis in order to make an accurate decision. We observe the frequency of connections within the window, and also the intensity of the data, in order to make a decision on patterns that our system has not seen before. If the intensity of the traffic is greater than a heuristically determined threshold, it is flagged as abnormal. This scheme has reduced the occurrence of false alarms to almost zero.

4.3. Decision Fusion

Due to the uncertainty in each of the analysis levels, the possibility of false negative and false positive in their decisions is infallible. A better performance than using the individual levels can be obtained by applying an appropriate decision fusion technique to the multi-level system. In the current system design, three detection techniques are applied: Rule-based flow analysis, protocol behavior analysis, and payload behavior analysis. The most appropriate way to do the fusion is by computing a linear combination of the binary decisions for all observation windows of the multi-level system. The single level decision will be characterized by records either 0 for normal or 1 for abnormal. We assume that the different levels with different observation windows have all the knowledge required to identify any attack. There are several alternative methods to find the best linear combination coefficients. An attractive approach is the least squares technique, in which the independent variables are the individual decisions of the three levels, and the dependent variable is the "true" nature of the entry obtained from the training data [42]. Using least squares technique for each connection key and time-window, we can obtain a fused decision.

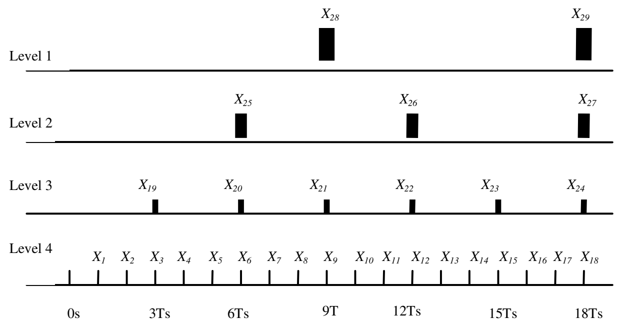

Level 1
X_{28}
X_{29}
Level 2
X_{25}
X_{26}
X_{27}
Level 3
X_{19} X_{20} X_{21} X_{22} X_{23} X_{24}
Level 4
X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9 X_{10} X_{11} X_{12} X_{13} X_{14} X_{15} X_{16} X_{17} X_{18}
0s
3Ts
6Ts
9T
12Ts
15Ts
18Ts

Figure 7. Alerts from multiple levels with different time windows

Figure 7 describes an example of alerts generated from four different detection systems which have different alert reporting periods. We consider the four systems have time windows: 10 seconds, 30seconds, 60 seconds and 90 seconds respectively. The decision of normal or abnormal for each systems is made at the end of its time window. For those systems we consider 180 seconds window to be the common time frame for fusion.

Focusing on one time frame which has 180 seconds as shown in **Figure 7**, we assume that the point ***i*** (***i*** = 1, 2, 3, 4,…, 18) represents the end of every 10 second time interval, the point ***i*** (***i*** = 19, 20, …, 24) represents the end of every 30 second time interval, the point ***i*** (***i*** = 25, 26, 27) represents the end of every 60 second time interval and the point ***i*** (***i*** = 28, 29) represent the end of every 90 second time interval. The true value of being normal or abnormal of the each key for this time frame is known. Suppose we have the key ***j*** (***j*** =1, 2, 3, 4… ***m***), $X_{j,i}$ (***i*** = 1, 2, 3, 4,…, 29; ***j*** =1, 2, 3,…, ***m***) are the decisions (0,1) of being normal or abnormal at the point ***i*** for key ***j*** in frame ***h***, y_j (***j*** =1, 2, 3,…, ***N***) are the true value of the normal or abnormal. then:

$$X_j = [x_{j,1}, x_{j,2}, \ldots, x_{j,n}]$$

$$F(X_j) = \sum_{i=1}^{n} w_i \cdot x_{j,i} = y_j$$

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\mathbb{Z} \cdot W = Y$$

$$\mathbb{Z}^T \cdot \mathbb{Z} \cdot W = \mathbb{Z}^T \cdot Y$$

$$W = \left(\mathbb{Z}^T \cdot \mathbb{Z}\right)^{-1} \cdot \mathbb{Z}^T \cdot Y$$

4.4. Action Module

One of the major problems in current IDS systems is the ability of automating actions. This is due to the high number of false positive alerts that those systems have. After testing ML-IDS for weeks in the lab, the system gave almost zero false positive alerts. This encouraged the building of an automated action module. The action module was designed to have high maintainability, scalability, and compatibility. Thus, this module was divided into three submodules: Action recommendation module, Action Translation module, and Resource connectivity module.

The action recommendation module is responsible of translating the alerts and the results of risk and impact analysis into high level actions understandable by human beings. The action translation module, is responsible of parsing the high level action into an action that is understandable by network resources. It also identifies the set of network resources that are going to take the action and their types. The last module, which is the resource connectivity module, is in charge of maintaining the control communication channel to the network resource for better connectivity. The control and management of these modules was implemented using autonomia [1].

5. Experimental Results and Evaluation

In what follows, we describe the experimental results of the Rule-based behavior analysis, and the protocol analysis systems and evaluate the effectiveness of our approach.

Our experimental test bed environment consists of 2 Cisco 2800 series routers, a Central 10/100M switch and 15 PCs. These computers are configured into 3 sub networks. Routine network services and applications such as web browsing, email and instant messenger are running on the test bed. We use our attack library to inject viruses, worms, and different kinds of attacks within the test bed. The network services are monitored and analyzed using Autonomia agents. The collected traffic is fed to the rule-based behavior analysis system and the protocol behavior analysis system in parallel. The list of attacks that the current implementation of ML-IDS can detect is shown in Table 1.

Table 1. Attacks detected by ML-IDS

Attack Category	***Results***
Scanning	Detected
Passive Scanning	Not detected
Exploits	Not detected
R2L	Detected
DoS Attack	Detected
Worm	Detected

5.1. Rule-based flow behavior analysis

In the current ML-IDS prototype, we used 290,872 records of traffic to train the Rule-based flow behavior analysis system. Among these records, there are 70,089 normal records and 220,783 abnormal records. The normal records represent normal traffic activities such as web browsing, network time protocol, DNS, NetBIOS, file transferring, audio, video, and secure shell.

This type of analysis can accurately detect and protect against scanning, DoS, Worm and R2L attacks. However, the current implementation does not collect and analyze host workloads and processes, and consequently we were not able to detect attacks that exploit operating system vulnerabilities (e.g., exploits such as Ownstation, snooqer, killthemessenger, smb/rpc nuke, rpc dcom, octopus, jolt2).

5.2. Protocol behavior analysis system

When the normal network traffic, as well as the malicious traffic are injected into the system, the traffic is framed into ***n***-grams and compared against both Bloom Filters simultaneously. We have trained the normal Bloom Filter using 3498681 patterns, and the abnormal Bloom Filter with 1106653 patterns. We have used a wide range of normal traffic activities including web browsing, instant messenger, file transfer, audio and video streaming, voip and secure shell.

This type of analysis can accurately detect and protect against scanning, DoS, Worm and R2L attacks. The computational complexity of the operations on a Bloom filter is O (1), and hence it has very low computational overhead.

5.3. Decision Fusion System

We did not encounter any false positive alarms from attacks that manipulate TCP header traffic in our current implementation of this system.

This algorithm was tested with real traffic for four different IDS systems that have different alert periods. **Table 2** shows the performance of this algorithm with seven different data sets collected from the four IDS systems. It is obvious that the fusion algorithm is effective and has a better detection rate than the individual levels.

6. Conclusion and Future Work

In this paper, we presented an autonomic network defense system that can detect and protect against network attacks with high detection rate and very low false alarms. We have used two separate systems that analyze network traffic on different granularities, and the fusion module intelligently combines the results of both systems.

The current ML-IDS prototype implementation could not detect attacks that require payload monitoring and host attacks because the ML-IDS system monitors only packet header and network flow information. We are currently implementing the remaining modules of the system such as the Payload behavior analysis, Risk and Impact analysis, and Adaptive learning modules.

Table 2. Fusion Algorithm Results

Data Sets	Detection Rate				
	IDS 1	IDS 2	IDS 3	IDS 4	Fusion
Set 1	69.1	27.5	19.0	19.0	96.8
Set 2	67.1	30.0	23.0	23.0	88.9
Set 3	67.0	29.9	23.0	23.0	97.0
Set 4	67.0	30.1	23.0	23.0	70.5
Set 5	67.0	30.0	23.0	23.0	87.1
Set 6	66.9	30.1	23.0	23.0	96.9
Set 7	66.8	30.0	23.0	23.0	75.6

7. References

[1] S. Hariri, G. Qu, H. Chen, Y. Al-Nashif, M. Yousif, "Autonomic Network Security Management: Design and Evaluation", *ACM Transactions on Autonomous and Adaptive Systems - Special Issue on Adaptive Learning in Autonomic Communication*, 2007.

[2] K. Ilgun. "USTAT: A Real-Time Intrusion Detection System for UNIX", *Master Thesis*, University of California, Santa Barbara, November 1992.

[3] G. Vigna and R. Kemmerer, "NetStat: A Network-Based Intrusion Detection Approach", In *Proceedings of the 14th Annual Information Theory: 50 Years of Discovery Computer Security Application Conference*, Dec. 1998.

[4] U. Lindqvist, P.A. Porras, "Detecting Computer and Network Misuse through the Production-Based Expert System Toolset (P-BEST)", In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. pp. 146 - 161.

[5] Wenke Lee and Salvatore Stolfo, "Data mining approaches for intrusion detection", In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[6] Richard P. Lippmann and Robert K. Cunningham, "Improving intrusion detection performance using keyword selection and neural networks", *Computer Networks*, 2000, 34:597–603.

[7] Jianxiong Luo, "Integrating fuzzy logic with data mining methods for intrusion detection", *Master's thesis*, Department of Computer Science, Mississippi State University, 1999.

[8] M. Joshi and V. Kumar, "Credos: Classification using ripple down structure (a case for rare classes)", In *Proceedings of 19th International Conference on Data Engineering*, Bangalore, India, 2003.

[9] Ramesh Agarwal and Mahesh V. Joshi, "PNrule: A new framework for learning classifier models in data mining (a case-study in network intrusion detection)", *Technical Report TR 00-015*, Department of Computer Science, University of Minnesota, 2000.

[10] Aleksander Lazarevic, Nitesh V. Chawla, Lawrence O. Hall, and Kevin W.Bowyer, "Smoteboost: Improving

the prediction of minority class in boosting", *Technical Report 2002-136*, AHPCRC, 2002

[11] Stefanos Manganaris, Marvin Christensen, Dan Zerkle, and Keith Hermiz, "A data mining analysis of rtid alarms", In *Proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection RAID*, West Lafayette, IN, 1999.

[12] Mahesh V. Joshi, Vipin Kumar, and Ramesh C. Agarwal, "Evaluating boosting algorithms to classify rare classes: Comparison and improvements", In *ICDM*, pages 257–264, San Jose, CA, 2001

[13] Ertoz, L., Eilertson, E., Lazarevic, A., Tan, P., Srivastava, J., Kumar, V., Dokas, P., "The MINDS - Minnesota Intrusion Detection System", *Next Generation Data Mining*, MIT Press, 2004.

[14] Dorothy E. Denning, "An intrusion-detection model", *IEEE Transactions on Software Engineering*, SE-13:222–232, 1987.

[15] Harold S. Javitz and Alfonso Valdes, "The nides statistical component: Description and justification", *Technical report*, Computer Science Laboratory, SRI International, 1993.

[16] T.F. Lunt and R Jagannathan, "A Prototype Real-Time Intrusion-Detection Expert System" In P*roceedings of the IEEE Symposium on Security and Privacy*, 1988, pp. 18-21.

[17] D. Anderson, T. F. Lunt, H. Javitz, A. Tamaru, and A. Valdes, "Detecting unusual program behavior using the statistical component of the next-generation intrusion detection expert system NIDES", *Technical Report SRI-CSL-95-06*, Computer Science Laboratory, SRI International, 1995.

[18] P.A. Porras and P.G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances", In *Proceedings of the National Information Systems Security Conference*, 1997, pp. 353-365.

[19] K. Sequeira, M. Zaki, "ADMIT: Anomaly-base Data Mining for Intrusions", *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, July 2002.

[20] N. Ye, "A Markov Chain Model of Temporal Behavior for Anomaly Detection", *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics.*

[21] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney, "Practical automated detection of stealthy portscans", *Journal of Computer Security*, 10:105–136, 2002.

[22] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo, "A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data", *Data Mining for Security Applications*, 2002.

[23] Charu C. Aggarwal and Philip S. Yu, "Outlier detection for high dimensional data", In *SIGMOD Conference*, 2001.

[24] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou, "Specification based anomaly detection: A new approach for detecting network intrusions", In *ACM Conference on Computer and Communications Security*, 2002.

[25] Shon, T. and Moon, J., "A hybrid machine learning approach to network anomaly detection" *Inf. Sci.* 177, 18 Sep. 2007, 3799-3821.

[26] G. Qu, S. Hariri, and M. S. Yousif, "A new dependency and correlation analysis for features." *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 9, 2005 pp. 1199–1207.

[27] Y. Al-Nashif, G. Qu, H. Chen, S. Hariri, and A. Arun Kumar, "Autonomic Network Defense (AND) System: Design and Analysis", Submitted to *IEEE Transactions on Secure and Dependable Computing.*

[28] H. Chen, Y. Al-Nashif, G. Qu, and S. Hariri, "Self-Configuration of Network Security", the *11th IEEE International EDOC Conference*, Annapolis, Maryland U.S.A. 15-19 October 2007.

[29] InSeon Yoo and Ulrich Ultes-Nitsche, "Towards Run-time Protocol Anomaly Detection and Verification", *Proc. of the 1st International Conference on E-Business and Telecommunication Networks*, Setubal, Portugal, 25-28 August, 2004.

[30] RFC793, "Transmission control protocol", September 1981. DARPA Internet Program Protocol Specification.

[31] Ke Wang, Janak J. Parekh, Salvatore J. Stolfo "Anagram: A Content Anomaly Detector Resistant To Mimicry Attack", In *Proceedings of the Nineth International Symposium on Recent Advances in Intrusion Detection(RAID 2006)*

[32] Staniford-Chen, S., V. Paxson, and N. Weaver, "How to 0wn the Internet in Your Spare Time", In *USENIX Security*, 2002.

[33] Christodorescu, M. and S. Jha. "Static Analysis of Executables to Detect Malicious Patterns", In *USENIX Security Symposium*, Washington, D.C. 2003.

[34] Vargiya, R. and P. Chan., "Boundary Detection in Tokenizing Network Application Payload for Anomaly Detection", in *ICDM Workshop on Data Mining for Computer Security (DMSEC),* Melbourne, FL, 2003.

[35] Bloom, B.H., "Space/time trade-offs in Hash Coding with Allowable Errors", *Communications of the ACM*, 1970. **13**(7): p. 422-426.

[36] IOS Firewall and Microsoft Windows Vista TCP Window Scaling, Document ID: 71613, 2006–2007 Cisco Systems, Inc.

[37] Network connectivity may fail when you try to use Windows Vista behind a firewall device, Article ID:934430, Sept. 26, 2007. Microsoft Corp.

[38] Martin Roesch, "Snort - Lightweight Intrusion Detection for Networks", *13th Systems Administration Conference - LISA* 1999.

[39] Daniel Barbara, NingningWu, and Sushil Jajodia, "Detecting novel network intrusions using bayes estimators", In *Proceedings of First SIAM Conference on Data Mining*, Chicago, IL, 2001.

[40] W. W. Cohen. "Fast effective rule induction", *In Proceedings of the Twelfth International Conference on Machine Learning*, Lake Tahoe, California, 1995

[41] Arash Partow, http://www.partow.net/programming/hashfunctions/#AvailableHashFunctions

[42] Christopher M. Bishop, "Pattern Recognition and Machine Learning", Springer, 2007.