

Chapter 8

A Multiagent-based Intrusion Detection System with the Support of Multi-Class Supervised Classification

Mei-Ling Shyu and Varsha Sainani

Abstract The increasing number of network security related incidents have made it necessary for the organizations to actively protect their sensitive data with network intrusion detection systems (IDSs). IDSs are expected to analyze a large volume of data while not placing a significantly added load on the monitoring systems and networks. This requires good data mining strategies which take less time and give accurate results. In this study, a novel data mining assisted multiagent-based intrusion detection system (DMAS-IDS) is proposed, particularly with the support of multi-class supervised classification. These agents can detect and take predefined actions against malicious activities, and data mining techniques can help detect them. Our proposed DMAS-IDS shows superior performance compared to central sniffing IDS techniques, and saves network resources compared to other distributed IDS with mobile agents that activate too many sniffers causing bottlenecks in the network. This is one of the major motivations to use a distributed model based on *multiagent* platform along with a *supervised classification* technique.

8.1 Introduction

The growing importance of network security is shifting security concerns towards the network itself rather than being just host-based. Security services are evolving into network-based and distributed approaches to deal with heterogeneous open platforms and support scalable solutions. Intrusion detection is the process of identifying network activities that can lead to a compromise of security policy. Intrusion detection systems (IDSs) must analyze and correlate a large volume of data collected from different critical network access points [12]. This task requires an IDS to be able to characterize distributed patterns and to detect situations where a sequence of intrusion events occur in multiple hosts. In addition, intrusion prevention techniques

Mei-Ling Shyu and Varsha Sainani
Department of Electrical and Computer Engineering, University of Miami, Coral Gables, FL 33124, USA, e-mail: shyu@miami.edu, v.sainani@umiami.edu

such as user authentication, authorization, encryption, defensive programming and IDSs are often used as another wall to protect computer systems.

The two main intrusion detection techniques are *misuse detection* and *anomaly detection*. Misuse detection systems [2][10] use patterns of well known attacks or weak spots of the system to match and identify known intrusions. Misuse detection techniques in general are not effective against novel attacks that have no matched rules or patterns yet. On the other hand, *anomaly detection* systems observe flag activities that deviate significantly from the established normal usage profiles as anomalies or in other words as intrusions. Anomaly detection techniques can be effective against unknown or novel attacks since no prior knowledge about specific intrusions is required. However, anomaly detection systems tend to generate more false alarms than misuse detection systems because an anomaly can just be a new normal behavior [1][9].

As accuracy is the essential requirement for an IDS, its extensibility and adaptability are also critical in today's network computing environment. There can be multiple weak points for intrusions to take place in a network system. For example, at the network level, malicious IP packets can crash a host, and at the host level, vulnerabilities can occur in system software which can be exploited to execute an illegal root shell. Since malicious activities at different intrusion points are normally recorded in different data sources, an IDS needs to be extended to incorporate additional modules that specialize in certain components of the network systems. Hence, IDSs need to be adaptive in such a way that frequent and timely updates are possible.

Our research aims to develop a more systematic and automated approach for building IDSs. We have developed a set of tools that can be applied to a number of tasks such as capturing data, extracting features, classifying them into known and unknown attack categories, and ultimately stopping the ongoing malicious activity. We take a data-centric point of view and consider intrusion detection as a data analysis process. The central theme of our approach is to apply data mining techniques to the extensively gathered data to compute a model that accurately captures the actual behavior and patterns of the intrusions and normal activities. This approach significantly reduces the need to manually analyze and encode intrusion patterns, as well as the guesswork in selecting statistical measures for normal usage profiles. The resultant model is more effective because it is computed and validated using a large amount of network data. This necessity of analyzing large amounts of data and finding its nature requires data mining strategies. The instant at which the sign of an attack is concluded, necessary actions are required which can be accomplished by intelligent agents. Hence, the integration of agent technology and data mining techniques makes an IDS more autonomous and efficient.

The remaining part of this chapter is organized as follows. Existing work is discussed in Section 2. Section 3 presents the design of our proposed DMAS-IDS. Section 4 describes our experimental setup, and the results of the performance analysis are given in Section 5. Finally, Section 6 concludes the paper.

8.2 Existing Work

Various distributed intrusion detection architectures using the multiagent design methodology and the data mining techniques have been developed. These approaches widely range from being comprised entirely of mobile agents like the MANET system [5][13], being merely a collection of static agents as in [17], or a combination of both as in DIDMA system [7].

IDSs have undergone rapid development in both power and scope in the last few years. Recently, the agent concept has been widely used in distributed environments because it provides many favorable characteristics including scalability, adaptability, graceful degradation of service, etc. as compared to the non-agent based systems. Most of the distributed agent-based IDSs introduced more traffic into their residing network, and therefore the communication protocol between various entities is also an important aspect that has to be considered. At the same time, most of the designed agent-based IDSs require comparatively high processing power in local machines to run the agents and other supportive software. Hence, a lightweight agent system with low network traffic generation requirements is needed. This can be accomplished with the use of appropriate data mining strategies.

Data mining techniques such as classification can be useful for both misuse detection and anomaly detection. In network intrusion detection, classification can be applied to classify network data consisting of malicious behaviors, and several existing approaches such as RIPPER, Naive Bayes, and multi-Bayes classifiers have been successfully used to detect malicious virus code. There are alternative classification approaches which can be effectively utilized for intrusion detection purposes. With intrusions, it is observed that over the time, the user establishes profile based on the numbers and types of commands they execute. Data mining classifier approaches like SOM (Self Organizing Maps) and LQM (Learning Vector Quantization) can be utilized for reducing dimensionality of these numbers. Furthermore, nearest neighbor classifier approaches based on SOM and LQM can be used to refine the collected network data in intrusion detection. A number of such systems have been developed which utilized the fast and efficient computation and pattern matching strategies of data mining to compliment the low cost and lightweight agent system architectures.

One of the well known examples of applying distributed agent design methodology in the intrusion detection domain is the Distributed Intrusion Detection System (DIDS). DIDS attempts to build a distributed system based on monitoring agents that reside at every host in the network. Distributed systems present both advantages and disadvantages. On one hand, the system utilizes the real-time traffic information from various sources, in the form of data from various host monitors or to assess the security status of its residing network. However, on other hand, the systems' scalability is poor for large networks as an increasing number of hosts monitoring the network also significantly increases the work load of the DIDS director agent. Additionally, the data flow between host monitors and the director agent may generate significantly high network traffic overheads. In [7], a system called DIDMA (Distributed Intrusion Detection using Mobile Agents) attempted to overcome the scalability issues inherent in the original DIDS architecture by employing mobile agents

in the data analysis task. Thus, by decentralizing data analysis, DIDMA hoped to significantly neutralize the effects of the scalability issues.

Another well known system called *BODHI* [8] was designed for heterogeneous data sources based on the techniques such as supervised inductive distributed function learning and regression. It focuses on the guarantee for correct local and global data models having least network communication. It was implemented in Java and offers message exchanges and runtime environments of agent systems for the execution of mobile agents at each local site. A central facilitator agent takes care of initializing and coordinating the data mining tasks.

A Java-based multi-agent system (*JAM*) [18] is designed to be used for meta-learning distributed data mining. In this system, each site agent builds a classification model, where different agents built their classifiers using different techniques. JAM also provides a set of meta-learning. Once combined together, the classifiers are computed with the central JAM system coordinating the execution of these modules to classify data sets at all data sites simultaneously.

In [19], a distributed agent-based IDS analyzes anomalies to detect and identify the denial-of-services (DoS) and data theft attacks. It also attempts to respond to intrusions in real time by sending out alerts to the designated network administrator when network intrusions are detected. One of its main drawbacks is the design complexity of its comprising agents, since each agent must take on almost all work load of network traffic sniffing, data parsing, and intrusion detection. In addition, its data mining techniques are less powerful since they are capable of detecting only a limited number of attacks.

8.3 The Proposed DMAS-IDS Architecture

In this study, we present a novel data mining assisted multiagent-based intrusion detection system (DMAS-IDS) architecture. DMAS-IDS integrates a multi-class supervised classification algorithm and the agent technology in network intrusion detection. It utilizes high accuracy and speed response of the *Principal Component Classifier (PCC)* [16][20] at the first layer of our proposed architecture. Once the results from PCC classification are obtained, agents communicate them to the second layer. The second layer of the DMAS-IDS architecture is integrated with the *Collateral Representative Subspace Projection Modeling (C-RSPM)* classifier [14] which includes collateral class modeling, class ambiguity solving, and classification. Results from this stage of classification are further analyzed by the agents and policies are derived which are communicated to the next layer using our own designed low cost and low response time agent communication protocol. The architecture is further elaborated in the following sections.

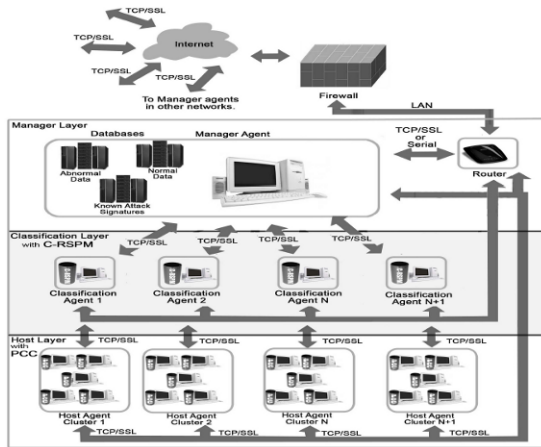


Fig. 8.1: The system architecture of the proposed DMAS-IDS

8.3.1 Agent Architecture

Fig. 8.1 presents our hybrid layered multiagent-based IDS architecture [15] which constitutes three layers called Host, Classification, and Manager layers. Each of these layers comprises of deliberative agents which are well aware of each other’s presence and are capable of communicating with each other using our developed communication scheme.

8.3.1.1 Host Layer

This layer marks the entry point of our proposed architecture. These end-user machines are workstations constituting the network, and they also act as the host agents inspecting each incoming network connection. Virtually, every machine in a network can be considered as a Host Agent. Host Agents collect network connection information and classify these connections as ‘normal’ or ‘abnormal’. Here, the *Principal Component Classifier (PCC)* is used [16][20].

Each Host Agent belongs to one Classification Agent (in the second layer), to which it reports the connections that PCC classifies them as ‘abnormal’. The responsibilities of these agents are (i) capturing network traffic, (ii) detecting abnormal activities in these connections, (iii) passing the classification results to its Classification Agent, (iv) properly responding to these abnormal activities for intrusion detection, and (v) passing a subset of the normal connection instances and the abnormal connection instances to the Manager layer to be saved in a database for the purpose of re-training the classifier at a later time.

8.3.1.2 Classification Layer

The second layer of the DMAS-IDS architecture is the Classification Layer. The responsibilities of the Classification Agents are (i) responsible for a set of Host Agents, (ii) classifying the abnormal connection instances found in their host machines into known attack types, and (iii) passing the classification results to the Manager Agent. Each Classification Agent is facilitated with a misuse detection algorithm called the *Collateral Representative Subspace Projection Modeling (C-RSPM)* [14]. This is important as the attack type will determine how the IDS should respond to the attack to safeguard the data in the network.

Unlike the Host Agents, dedicated machines are needed to run the Classification Agents so that they have enough processing power to handle all classification requests of their Host Agents. Additionally, they have to generate ‘policies’ upon the instances which are identified as attacks. Once the policy is created, it is communicated to the next layer called the Manager Layer. All the Classification Agents present in the network add their policies to the policy repository present in the ‘Database’ in the Manager Layer.

8.3.1.3 Manager Layer

This layer, in terms of contemporary agent models, is the same as the planning layer. The Manager Agent is in charge of the entire system, performing several support tasks for the system. Its responsibilities include (i) assigning a Host Agent to the specific Classification Agent, (ii) assisting the Classification Agents in managing their host machines and the tasks related to them, and (iii) managing the routers and firewalls in the network.

The main task performed by the Manager Agent is to take the policies from the Classification Agents throughout the network. After receiving the policies, the Manager Agent broadcasts them to every agent present in the network. Once the policy is received by all Host Agents, the corresponding Host Agent who initiated the request implements the policy. This functionality is important as the Classification Agents can prevent or lessen the effects of a possible attack by managing resources in those nodes that they expect to be affected by the incoming attack, such as bandwidth, communication ports, and connection authorization.

8.3.1.4 Agent Communication

A simple and manageable communication scheme that utilizes a discrete number of *KQML* performatives is designed to accommodate the goals of all the aforementioned agents. Agents use these messages to register with the upper level of agents and communicate the results of the tasks they are responsible of performing.

As soon as the Classification Agent comes online, it registers itself with the Manager Agent using the “REGISTER” performative. The same registration process applies to the Host Agent when it has to register with the Classification Agent. However, since the Host Agent needs to know to which the Classification Agent it

belongs as soon as it comes online, it uses the “RECOMMEND-ONE” performative to ask the Manager Agent. The Manager Agent then uses the “TELL” performative to reply to the Host Agent with the available Classification Agent’s IP. As mentioned earlier, the Host Agent diagnoses each incoming connection using PCC and uses the “EVALUATE” performative to communicate this result to the Classification Agent. The Classification Agent further classifies any abnormal connection into a known attack type and derives a policy based on it. It uses the “REPORT” performative to communicate the corresponding policy to the Manager Agent. Finally, the Manager Agent broadcasts this policy to all other agents present in the network using the “BROADCAST” performative. In our proposed DMAS-IDS architecture, the TCP/IP Secure Socket Layer (SSL) was adopted for the implementation of the secure communication channel among the agents to provide total privacy and authentication capabilities. The adoption of cryptographic communication services is important for making the distributed multiagent IDS architecture immune against attacks.

8.3.2 Principal Component Classifier (PCC)

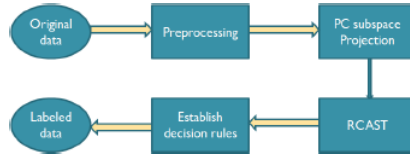
In the classification module of C-RSPM, each classifier is called the Principal Component Classifier (PCC) [20] (as shown in Fig. 2(a)). PCC will classify a connection as either normal (non-intrusion) or abnormal (possible intrusion). PCC basically goes through four basic steps of classification: (i) preprocessing, (ii) Principal Component Subspace Projection, (iii) Automatic Representative Component Selection, and (iv) Establishment of the Decision Rules.

In the preprocessing step, the average and standard deviation of the instances in the normal class are calculated, and these statistical characteristics are used to normalize the data. Let L be the set of normalized training data instances, $i = 1, 2, \dots, p$, $j = 1, 2, \dots, L$, $\bar{\mu}_i$ and s_{ii} be the sample mean and the variance of i^{th} row of the trimmed matrix \mathbf{X} respectively, and \mathbf{x}_{ij} ($i=1, 2, \dots, p$, $j=1, 2, \dots, L$) be the elements in matrix \mathbf{X} . Define the normalized un-trimmed data set consists of p features as shown in Equation (8.1) and its corresponding column vectors as presented in Equation (8.2), where Equation (8.3) is used for normalization. In order to decide the percentage of the data instances that can be regarded as outliers, a Parzen window is used to decide which data instances are retained and which are removed as outliers according to a defined ‘rtd’ factor. The ‘rtd’ factor is chosen corresponding to the center of a Parzen window where the maximum accuracy is reached. If there is a tie, then the first one is chosen as the default one.

$$\mathbf{Z} = \{\mathbf{z}_{ij}\}, i = 1, 2, \dots, p, j = 1, 2, \dots, L. \quad (8.1)$$

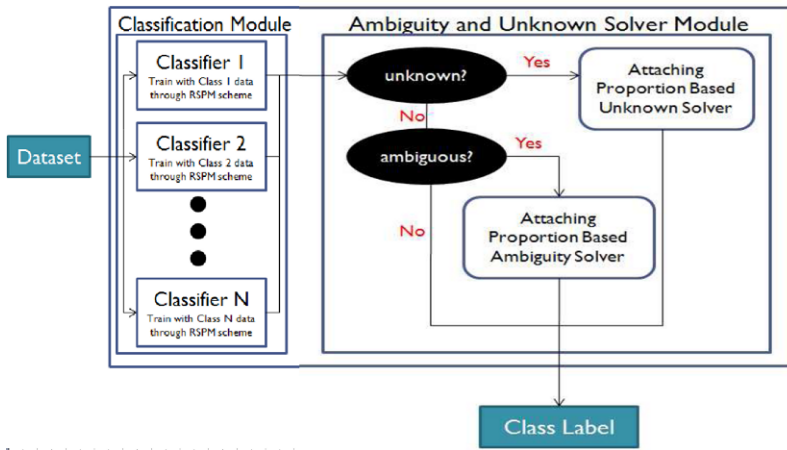
$$\mathbf{Z}_j = (\mathbf{z}_{1j}, \mathbf{z}_{2j}, \dots, \mathbf{z}_{pj})', j = 1, 2, \dots, L. \quad (8.2)$$

$$\mathbf{z}_{ij} = \frac{\mathbf{x}_{ij} - \bar{\mu}_i}{\sqrt{s_{ii}}}. \quad (8.3)$$



(a) Principal Component Classifier Architecture

C-RSPM Framework



(b) Collateral representative subspace projection modeling for supervised classification architecture

Fig. 8.2: PCC and C-RSPM

Next step is to automatically select the representative principal components (PCs). In the PC space, only those dimensions that have positive eigenvalues are selected. Before this, we need to perform the projection for the data instances from the original space to the PC space. That is, each retained training data instance is projected to a subspace by using those PCs derived from the normal class. Let $\mathbf{E}_i = (e_{i1}, e_{i2}, \dots, e_{ip})^T$ be the i^{th} eigenvector, and $(\lambda_1, \mathbf{E}_1), (\lambda_2, \mathbf{E}_2), \dots, (\lambda_p, \mathbf{E}_p)$ be the p eigenvalue-eigenvector pairs of the robust correlation matrix \mathbf{S} . Also, let \mathbf{Y} be the projection of \mathbf{Z} onto the p -dimensional eigenspace (shown in Equation (8.4)) consisting of \mathbf{Y}_j column vectors (given in Equation (8.5)). Then, a sample score value of the normalized training data instance vector can be computed using Equation (8.6). We define the p score row vectors of \mathbf{Y} , representing the distribution of the p eigenspace features of all the un-trimmed, normalized, and projected training

instances, as $\mathbf{R}_i = (\mathbf{y}_{i1}, \mathbf{y}_{i2}, \dots, \mathbf{y}_{iL})$, $i=1,2,\dots,p$. Following this, a lambda score value is computed for each PC.

$$\mathbf{Y} = \{\mathbf{y}_{ij}\}, i = 1, 2, \dots, p, j = 1, 2, \dots, L. \quad (8.4)$$

$$\mathbf{Y}_j = (\mathbf{y}_{1j}, \mathbf{y}_{2j}, \dots, \mathbf{y}_{pj})', j = 1, 2, \dots, L. \quad (8.5)$$

$$\mathbf{y}_{ij} = \mathbf{E}_i' \mathbf{Z}_j = e_{i1} \mathbf{z}_{1j} + e_{i2} \mathbf{z}_{2j} + \dots + e_{ip} \mathbf{z}_{pj}. \quad (8.6)$$

In the attempt to generate a better predictive model, the set of available score row vectors is refined by eliminating those possessing extremely insignificant or null variability, i.e., extremely little standard deviation. Next, a distance measure is defined as shown in Equation (8.7).

$$\mathbf{c}_j = \sum_{m \in M} \frac{(\mathbf{y}_{mj})^2}{\lambda_m}. \quad (8.7)$$

Finally, the classification decision rules can be established, where the decision rules to classify each of the data instances \mathbf{X}'_j , $j=1,2,\dots,N'$, are based on the selected threshold value C_{thresh} . The details of the steps of establishing the decision rules and determination of C_{thresh} can be found in [14]. In summary, we classify the j^{th} testing data instance as abnormal if $\mathbf{c}'_j > C_{\text{thresh}}$.

8.3.3 Collateral Representative Subspace Projection Modeling (C-RSPM)

Among various data mining techniques, *supervised classification* has become an essential tool that has been applied successfully in diverse research areas including *network intrusion detection systems*. Since it shows promising results with a number of data sets as compared to other available algorithms, it is utilized in our proposed framework. The system architecture of our previously developed C-RSPM classifier [14] is illustrated in Fig. 2(b). As can be seen from this figure, it includes the *Classification* module and *Ambiguity Solver* module.

The Classification module is composed of an array of deviation classifiers (i.e., one PCC for each class) which are executed collaterally. That is, each of the classifiers receives and classifies the same testing instance simultaneously [14]. The basic idea of the C-RSPM classifier is that each classifier is trained with the data instances of a known class in the training data set. Thus, training the C-RSPM classifier consists basically of training each individual classifier to recognize the instances of each specific class. In the ideal case, a testing data instance will be classified as 'normal' to only one classifier's training data instances. However, in realistic situations, a testing data instance may be classified as 'normal' by multiple classifiers or none of the classifiers considers it as 'normal'. To address these two scenarios, the *Ambiguity Solver module* is introduced. Ambiguity Solver captures and coordinates classifica-

tion conflicts and also provides an extra opportunity to improve the classification accuracy by estimating the true class of an ambiguous testing data instance.

There can be one more scenario for class ambiguity, which arises with a simple fact that no classifier can ensure the 100% classification accuracy, as in most cases, a data set would contain classes with very similar properties. Thus it makes it difficult for a classifier to identify the correlative differences between these classes with small feature differences.

Unlike many other algorithms which upon encountering such issues simply resort to solutions such as the random selection of a class label from among the ambiguous class labels, the C-RSPM classifier attempts to properly address the issues by defining a class-attaching measure called *Attaching Proportion* for each of the ambiguous classes. The goal of solving such an ambiguity issue via attaching proportion measure is to label an ambiguous testing data instance with the label of the classifier exhibiting the lowest *Attaching Proportion* value. Additionally, the *Attaching Proportion* can be viewed as a measure of the degree of normality of a data instance with respect to a class, indicating the percentile of normality the data instance under analysis is associated to the corresponding class.

8.3.4 Policy Derivation at Classification Agents

Ideally, an IDS should aim at not only detecting the intrusions but also stopping them as soon as they are detected. Our proposed architecture takes this requirement of the IDSs into consideration as well. As mentioned above, the Classification Agent derives a policy after it concludes the type of the abnormal connection, and then reports this policy, rather than the attack type, to the Manager Agent.

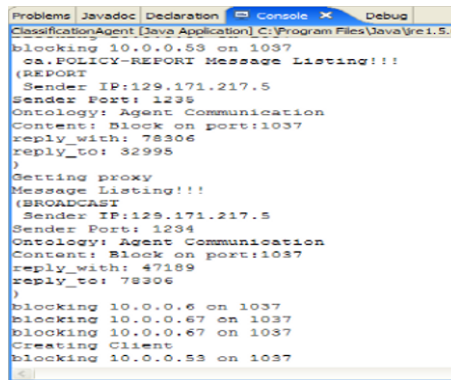
Policy derivation plays a very important role in the scenarios when the same attacker generates different attacks to different hosts in the same cluster at the same time instant or at different time instants. If the Classification Agent just reports the attack types, it does not serve the purpose, as the Manager agent may recommend a common measure for all different attacks at different hosts. In such situations, the policies prove their importance since policies provide the solutions to the ongoing attacks. This could be better explained with the example as follows. If an attacker A makes one attack called 'attack1' at host HA1 and the other attack called 'attack2' at host HA2, and both hosts belong to Classification Agent CA. Understanding the properties of 'attack1', it is enough to block the access of the attacker to a particular port; whereas for 'attack2', it may be required to block the access of the attacker to a particular server in the network. It is very much expected that in real case scenarios, we cannot just go blocking each doubtful machine. Hence, it is required to have case specific solutions, and policies can well provide acceptable solutions for this. Policies add more dynamicity and adaptability to an IDS by providing attack specific solutions.

To demonstrate how such types of policies can be established, in our experiments, three types of policy options are defined. Of course, more policies can be defined

to accommodate different network intrusions. Following gives the three pieces of policies that are defined in our experiments.

1. Block;
2. Block on a port; and
3. Block on a server.

As soon as the Classification agent concludes the attack type, it looks at the features of the attacks, chooses one of the three policy options, and then reports this to the Manager Agent. Fig. 8.3 shows a screen dump where incoming instances are being blocked to the port which has been diagnosed as a penetration point in the network.



```
Problems Javadoc Declaration Console x Debug
ClassificationAgent [Java Application] C:\Program Files\Java\jre1.5.0_
blocking 10.0.0.53 on 1037
ca.POLICY-REPORT Message Listing!!!
(REPORT
  Sender IP:129.171.217.5
  Sender Port: 1235
  Ontology: Agent Communication
  Content: Block on port:1037
  reply_with: 78306
  reply_to: 32995
)
Getting proxy
Message Listing!!!
(BROADCAST
  Sender IP:129.171.217.5
  Sender Port: 1234
  Ontology: Agent Communication
  Content: Block on port:1037
  reply_with: 47189
  reply_to: 78306
)
blocking 10.0.0.6 on 1037
blocking 10.0.0.67 on 1037
blocking 10.0.0.67 on 1037
Creating Client
blocking 10.0.0.53 on 1037
```

Fig. 8.3: Policy derivation and implementation

8.4 Experimental Setup

In order to assess the overall performance of our DMAS-IDS together with the our previously developed communication protocol in a realistic scenario, a prototype of the proposed architecture was implemented using Java RMI package in the particular lipe RMI [11], which allows a number of machines to communicate with each other at the TCP level. We have conducted evaluations in terms of scalability-related criteria such as network bandwidth and system response time for the analysis of our implemented protocol and proved its performance in our previous study [15]. Fig. 8.1 shows our DMAS-IDS network testbed, where the different types of agents were placed in mutually exclusive machines within the testbed in a manner such that any communication among the agents could only be realized through the generation of network traffic rather than local traffic within the same machine.

In order to test our framework, various experiments were organized to assess the performance of (i) the response time characteristics of the proposed architecture in terms of agent communication with supervised classification scheme, and (ii) the accuracy of classification and policy derivation of the agents. Both the training

and testing data sets were acquired from network traffic data generated in our own testbed. To evaluate the performance of the proposed framework, the classifiers were trained offline with the data generated from the testbed. Here, ten types of traffic were generated, which include 5,000 normal connections and 100 connections for each type of abnormal traffic classes. The generated ‘normal’ connections provide a proper quantity of data and transfer them during a 5-second interval; whereas typical abnormal connections generate a large amount of data in a short span of times (varying for all types of attacks) continuously. For example, connections sending extremely huge packets are used to simulate the ‘ping of death’ attacks; connections with a lot of packets in a short time duration simulate the ‘mail bombing’ attacks; connections which try to access the reserved ports are simulated as the ‘Trojan infections’; connections transmitting a number of large packets in a short time are used to simulate the ‘buffer-overflow’ cases, etc.

For processing these data sets, a number of tools were employed including our own previously developed tools. As mentioned above, in JAVA’s JPCAP [3][4][6], packages were used in our program to capture the packets from the network interface card and TCP trace was used to transform these packets into data instances. This traffic was generated using our own developed traffic generator, which is capable of generating a number of myriad attacks by simply varying its input parameters. The TCP trace extracts 88 attributes from each TCP connection like elapsed time, number of bytes, and segments transferred on both ways, etc. We have also utilized our own developed feature extraction technique to extract 46 features from the output of TCP trace, which are useful for the proposed DMAS-IDS. These features include some basic, time based, connection-based, and ratio-based network features. Out of these 46 features, 14 are real time rates which are employed for our experiments.

The focus of our testbed based experiments is on network attacks based on the TCP network protocol, since a great majority of the attacks are either executed via or rely on a certain degree on the TCP protocol. This is due mostly to TCP’s frangibility and instability. Please note, however, that our proposed architecture is not limited to the detection of simply TCP based network attacks. The network protocol is simply one of many categorical features employed to describe a network connection.

Having our tools installed in all of the machines in our testbed, we start the traffic generator on one of the machines serving as the sender and have another machine as the receiver where we have our Host Agent running. This Host Agent captures the incoming packets by reading it from the network card in the machine and caches it in the memory at the interval of every 5 seconds. Next, it calls the TCP trace software which reads the dump files and converts them to the data instances. Having the data instance from the TCP trace, the Host Agent calls our feature extraction tool to extract the required features from the data instance. After retrieving the resulting features from each data instance (i.e., a network connection), the Host Agent classifies it using PCC.

We had saved all the classification parameters for each of the classes into text files in the training phase. This helps us classify data instances in real time. PCC [20] classifies the data instances only in ‘normal’ and ‘abnormal’ classes and returns this label to the Host Agents. If the data instance is classified as ‘normal’,

then it is saved with the Host Agent, but if it is classified as ‘abnormal’, then the Host Agent further sends it to its Classification Agent on another machine using the ‘EVALUATE’ message. Upon receiving the ‘EVALUATE’ message with the abnormal instance information, the Classification Agent calls C-RSPM [14] to further analyze this data instance to classify it to a predefined known attack class. This label is used by the Classification Agent to derive its policy by looking at the features of that attack. The Classification Agent then places this policy into the ‘REPORT’ message and connects it to the Manager Agent which is located on another different machine. Upon receiving the message from the Classification Agent, the Manager Agent broadcasts it to all the Classification Agents in the network which further send it out to its Host Agents using the ‘BROADCAST’ message.

8.5 Results and Analysis

Our implemented agent architecture (excluding C-RSPM) has made its mark by proving its scalability, low cost, and low response time in our previous study [15]. It was evaluated with a number of experiments and a total of 506 agents were simulated altogether. We simulated 500 host agents and 5 classification agents along with one manager agent. The experiments over this setup were conducted in order to evaluate its performance in terms of its bandwidth requirements and response time. From [15], it can be observed that the system scaled linearly in terms of the attack response time. The highest response time was of 95 seconds for 100 host agents, 5 classification agents, and 1 manager agent, which is absolutely negligible. This also proved how fast our agents respond and hence how fast they can deliver the task assigned. Also, the bandwidth consumption results of this system were less than 1 MBit/sec, which is almost negligible as well. This demonstrates our communication protocol enables information exchange with low overhead and system scalability. This makes our proposed system low cost which is definitely a desirable feature for any distributed system. It is also clear from the results that in our proposed architecture, the performance of the system will not deteriorate too much with the increase in the number of attacks, which is justified by its low bandwidth consumption and quick response time behavior.

Next, the performance of C-RSPM was evaluated in [14]. As mentioned above, it is capable of performing high accuracy supervised classification and outperforms many other classification algorithms. C-RSPM has shown excellent performance with our testbed data and it has shown that it achieved 100% accuracy for some of the classes for our attack data generated from the testbed, and 99.97% on average for other data with the standard deviation varying only up to 0.09. This was motivating enough for us to integrate it with our agent system and build the proposed DMAS-IDS architecture.

Having proven the performance of our prior developed sub-systems, we next move to demonstrate the performance of the entire DMAS-IDS architecture together. As described above, we generated the attack data for 5900 instances and each of these instances was classified in real time, being captured from the network

data card of the machine and then going through each agent layer and respective classifiers. Fig. 4(a) shows the times consumed by each of the 100 instances continually going through the whole cycle of classification and communication, differentiated by the attack types. As can be seen from this figure, the data instances from all types of attacks, on average, result in the same response time irrespective of the attack types. This is definitely a desirable feature. If an IDS performs good for some of the attack types and not in the same way for the others, it can not be considered as a good system. Also, the longest time taken is 40 seconds in our experiments, which shows that our proposed DMAS-IDS architecture shows its promise by addressing an ongoing attack within 40 seconds after it has been detected as an intrusion. The time is measured from when an attack is detected and until the last ‘BROADCAST’ message has reached the Host Agent.

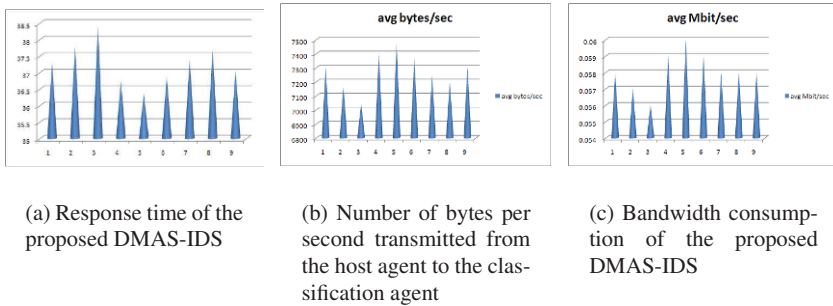


Fig. 8.4: Experimental results

Let’s now look at the bandwidth consumed by this architecture. As depicted by Fig. 4(b) and Fig. 4(c), the maximum bandwidth consumed by the system is 0.06 Mbits/sec, which is very low as well. This proves that even if the number of attacks increases exponentially, our system will still result in considerably fair performance. Also, if more machines are connected to the network, our proposed DMAS-IDS architecture will still withstand the load and deliver results. Therefore, it can be clearly seen that our proposed DMAS-IDS architecture is scalable, fast, and efficient. It provides an option over contemporary firewalls which are only good at blocking traffic for pre-known malicious connections.

8.6 Conclusion

In this paper, a novel distributed multiagent IDS architecture called DMAS-IDS is presented, which incorporates the desirable features of the multiagent design methodology with highly accurate, fast, and lightweight *PCC* Classifier and *C-RSPM* schemes. Even as a larger number of agents are introduced into the network, our proposed DMAS-IDS architecture provides effective communication between

its two comprising layers, through an efficient agent communication scheme that requires only a small and manageable generation of network traffic overhead as shown in the experimental results. A key concept in the design of the proposed DMAS-IDS architecture was to show the integration of the agent technology with the data mining strategies, and to prove how both compliment each other. To test the feasibility of a realistic employment and all the salient features of the proposed DMAS-IDS architecture, a private LAN testbed is built to facilitate both the generation of realistic normal and anomalous network traffic data and some common network attack generation tools, to appropriately validate the performance of proposed DMAS-IDS when compared to other well-known anomaly detection and supervised classification methods, and to assess the important scalability aspects of the proposed architecture such as system response time and agent communication generated network traffic overhead. From our experimental results, it can be concluded that the proposed DMAS-IDS architecture yields promising results, indicating a linear scalability and low response time including both agent communication and classification response times. These results are indicative that our proposed DMAS-IDS architecture provides many favorable characteristics such as being lightweight, good scalability, and least degradation of service.

References

1. Garuba M., Liu C., Fraites D.: Intrusion techniques: Comparative study of network intrusion detection systems. Fifth International Conference on Information Technology, New Generations, 2008.
2. Ilgun K., Kemmerer R. A., Porras P. A.: State transition analysis: A rule-based intrusion detection approach. *IEEE Trans. Softw. Eng.* 21, 3, pages 181-199, 1995.
3. JAMA (2008) Available at: <http://math.nist.gov/javanumerics/jama/>
4. Java Agent Development Framework (2008). Available at: <http://jade.tilab.com/>
5. Jin X., Zhang Y., Zhou Y., Wei Y.: A novel IDS agent distributing protocol for MANETs, V.S. Sunderan et al. (Eds.), *ICCS 2005, LNCS 3515*, pages 502-509, 2005.
6. JPCAP (2008) Available at: jpcap.sourceforge.net/javadoc/index.html
7. Kannadiga P., Zulkernine M.: DIDMA: A distributed intrusion detection system using mobile agents, *Proceedings of Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks*, pp. 238-245, 2005.
8. Kargupta H., Park B., Hershberger D., Johnson E.: Advances in distributed and parallel knowledge discovery, chapter 5, *Collective Data Mining: A New Perspective Toward Distributed Data Mining*. AAAI/MIT Press, 2000.
9. Klusch M., Lodi S., Moro G.: The role of agents in distributed data mining: Issues and benefits. *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology (IAT'03)*, 2003.
10. Kumar S., Spafford E. H.: A software architecture to support misuse intrusion detection. In *Proceedings of the 18th National Conference on Information Security*. 194-204, 1995.
11. liperMI (2006). Available at <http://lipermi.sourceforge.net/>
12. Marhusin M., Cornforth D., Larkin H.: An overview of recent advances in intrusion detection. *CIT*, 2008.
13. Pahlevanzadeh, B., Samsudin, A.: Distributed hierarchical IDS for MANET over AODV+, *IEEE International Conference on Telecommunications and Malaysia International Conference on Communications*, pages 220-225, May 14-17, 2007.

14. Quirino T., Xie Z., Shyu M.-L., Chen S.-C., Chang L.: Collateral representative subspace projection modeling for supervised classification. The Proceedings of 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), pages 98-105, 2006.
15. Sainani V., Shyu M.-L.: A hybrid layered multiagent architecture with low cost and low response time communication protocol for network intrusion detection systems. The IEEE 23rd International Conference on Advanced Information Networking and Applications, Accepted for publication, 2009.
16. Shyu M.-L., Chen S.-C., Sarinnapakorn K., Chang L.: Principal component-based anomaly detection scheme. Foundations and Novel Approaches in Data Mining, pages 311-329, Springer-Verlag, Vol. 9, 2006.
17. Spafford E., Zamboni D.: Intrusion detection using autonomous agents. Computer Networks 34, 4, 547-570, 2000.
18. Stolfo S., Prodromidis A., Tselepis S., Lee W., Fan D., Chan P.: JAM: Java agents for meta-learning over distributed databases. Proceedings of KDD-97, pages 74-81, Newport Beach, California, USA, 1997.
19. Vaidehi K., Ramamurthy B.: Distributed hybrid agent based intrusion detection and real time response system. Proceedings of the First International Conference on Broadband Networks, pages 739-741, 2004.
20. Xie Z., Quirino T., Shyu M.-L.: A distributed agent-based approach to intrusion detection using the lightweight PCC anomaly detection classifier. Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC'06), pages 446-453, 2006.