An Architecture of a Distributed Intrusion Detection System Using Cooperating Agents

Jaydip Sen¹, Indranil Sengupta¹, Piyali Roy Chowdhury²

¹Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur - 721302, INDIA
sen_jaydip@yahoo.com, isg@iitkgp.ac.in
²Department of Computer Science and Engineering
Future Institute of Engineering and Management, Kolkata-700150, INDIA
roychowdhury.piyali@gmail.com

Abstract- An Intrusion Detection System (IDS) is a security mechanism that is expected to monitor and detect intrusions into the computer systems in real time. The currently available intrusion detection systems have a number of problems that limit their configurability, scalability, and efficiency. There have been some propositions about distributed architectures based on multiple independent agents working collectively for intrusion detection. However, these distributed intrusion detection systems are not fully distributed as most of them centrally analyze data collected from distributed nodes resulting in a single point of failure. In this paper, we propose a distributed architecture with autonomous and cooperating agents without any central analysis component. The agents cooperate by using a hierarchical communication of interests and data, and the analysis of intrusion data is made by the agents at the lowest level of the hierarchy. This architecture provides significant advantages in design of an IDS in terms of scalability, flexibility, extensibility, fault tolerance, and resistance to compromise. We have developed a proof-of-concept prototype, and conducted experiments on the system. The results show the effectiveness of our system in detecting intrusive activities in any network of workstations.

I. INTRODUCTION

A secure computer system provides guarantees regarding the confidentiality, integrity, and availability of its objects (such as data, purpose, or services). However, systems generally contain design and implementation flaws that result in security vulnerabilities. An intrusion can take place when an attacker or a group of attackers exploits the vulnerabilities and thus damages the confidentiality, integrity or availability guarantees of a system. Intrusion Detection Systems (IDSs) detect some set of intrusions and execute some predetermined actions when an intrusion is detected.

Over the last decade, research in the field of intrusion detection has been heading towards a distributed framework of systems that do local detection and provide information to perform global detection of intrusions. These distributed frameworks of intrusion detection have some advantages over single monolithic frameworks [1]. Most of these distributed systems are hierarchical in nature. The local intrusions detection components look for local intrusions and pass their analysis results to the upper levels of the hierarchy. The components at the upper levels analyze the refined data from multiple lower level components, and attempt to establish a global view of the system state. However, such IDSs are not fully distributed systems because of the centralized data analysis performed at the higher levels of the hierarchy [2].

In this paper we propose an agent-based architecture for performing intrusion detection in a distributed fashion. By employing a suitable communication mechanism, the resource overhead is minimized in the intrusion detection process.

The rest of the paper is organized as follows: Section II describes some related works done on distributed intrusion detection, Section III briefly introduces the concept of agents and their interests, Section IV describes the architecture of the system, the communication mechanism of the agents, and some implementation aspects of the prototype. Section V highlights some experiments conducted and the results obtained, and the Section VI concludes the paper.

II. RELATED WORK

In this section, we briefly describe the some of the existing distributed IDS architecture. DIDS [4] is a distributed IDS consisting of host managers and LAN managers doing distributed data monitoring, and sending notable events to the DIDS director. These managers also do some local detection, passing the summaries to the director. The director analyzes the events to determine the security state of the system. This centralized nature of the director is clearly the bottleneck of the distributed approach of DIDS.

EMERALD [3] is a framework for distributed intrusion detection. It employs monitors at the levels of hosts, domains, and enterprises to develop an analysis hierarchy. It uses subscription-based communication scheme, both within and between the monitors. However, the inter-monitor subscription scheme is hierarchical in nature, thus limiting access to the events or results from the layer immediately below.

AAFID [5,8] is a distributed IDS developed in CERIAS at Purdue University. AAFID employs agents at the lowest level of the hierarchy for data collection and analysis. _Transceivers_ and _Monitors_ at the higher levels control the agents and monitor a global view of activities in the network. This hierarchy of data analysis components makes the system vulnerable to attack by malicious intruders. _The Fuzzy Intrusion Recognition Engine (FIRE)_ [6] is a network IDS that uses fuzzy systems to assess
malicious activities in a network. It utilizes AAFID architecture as the platform.

A prototype called the Hummingbird System [7] is developed at University of Idaho. Hummingbird is a distributed system that employs a set of Hummer agents, each assigned to a single host or a set of hosts. Each Hummer interacts with other Hummers in the system through a manager, a subordinate, and the peer relationships. Hummingbird is intended to allow a system administrator to monitor security threats on multiple computers from one central location.

An architecture of an intrusion detection system using a collection of autonomous agents has been proposed in [2]. In the cooperation and communication model proposed by the authors, agents request and receive information solely on the basis of their interests. They can specify new interests as a result of a new event or alert. This avoids unnecessary data flow among the agents. As a major advantage, the entire system is not compromised if an agent fails. Instead, there is a graceful degradation of system performance.

However, most of the hierarchical distributed IDSs have the following drawbacks:

1) **Analysis Hierarchy**: As there is a hierarchy in data analysis these systems are very difficult to modify. Changes may have to be made at many (if not all) levels if any new distributed attack is developed.

2) **Data Refinement**: When a module from a lower level sends results of analysis to a module at higher level, some data refinement is done. However, the knowledge of what events are important in a system-wide level is difficult to anticipate at the lower levels of the hierarchy, and thus data refinement may result in loss of some important information.

3) **Bulky Modules**: Intrusion analysis engines based on anomaly detection are large modules. They also consume significant amount of system resources in terms of CPU usage, disk I/O, and memory, as they have to analyze long audit trails. In most of the current IDSs, such components are present at all levels of hierarchy. They also present multiple points of failures.

4) **Passive Interactions**: The components of these IDSs interact with each other in a passive way. The lower level components generate data for the upper level components as per the rules driving them. There is no mechanism for a component to query other component on the basis of analysis that it has done. In EMERALD, the query mechanism is available between the adjacent levels of the hierarchy, and in AAFID it is allowed only within a host.

In our proposed model, we solve these problems by a mechanism of intelligent coordination among a collection of agents. The key effort in our approach is directed towards making the agents cooperate by communicating with each other. The intelligence in cooperation is attempted by communicating events and alerts to only those agents that are interested in them. The analysis of the data is done by the agents at the lowest level of the hierarchy. The agents at the higher level are lightweight components and they can communicate directly with other agents at the same level in the agent hierarchy. Thus in the event of failures of the agents at higher levels, the communication between the lower level agents will not be disrupted. This makes the system fault-tolerant.

### III. AGENTS AND THEIR REGISTRIES

#### A. Agents

In this section we briefly describe some properties of agents. Crossbie and Spafford [9,10] first pointed out the suitability of agents for intrusion detection purpose. Nwana and Woolridge [11] have defined an agent as a computational and physical entity situated in an environment (either virtual or real), which is able to act in the environment, and to communicate with other agents in the environment. It is also driven by internal tendencies (goals, beliefs etc.), and has an autonomous behavior, which is the consequence of its perception, its representation and its interaction with the environment and with other agents. The agents can be distinguished according to their ‘intelligence’ levels [12]. A cognitive agent is able to find a solution for a complex problem while communicating with other agents and interacting with its knowledge base. A reactive agent reacts quickly for a simple problem that does not require complex reasoning. A hybrid agent is a mixture of reactive and cognitive agents. It has some reflex (reactive evolution) to resolve repeated problems, and thinks about complex system situations. In our work, the agents have reasoning capabilities, autonomy, adaptability, communication, and cooperation in order to reach some intrusion detection goals.

#### B. Interests

An interest is defined as “a specification of data that an agent is interested in, but is not available to the agent because of the locality of data collection or because the agent was not primarily intended to observe those data” [2]. An agent may express an interest in the following two situations:

1) There may be more than one agents residing on the same host that need data from the same data source. If the overhead of the data access mechanism from the data source is more than the communication overhead of transferring the data between the agents, it will be more efficient if one agent obtains data and makes it available to the other agents.
2) When an agent is required to detect coordinated or distributed attacks, it may need data from agents residing in multiple hosts. The agent may not have access to the data sources in this case, and transfer of data from other agents is the only solution.

It is impossible for an agent to keep track of all other agents in a large network having thousand of hosts belonging to different domains. Thus, an agent may not know the existence or locations of agents that are collecting data in which it is interested. Hence propagation of interests of different agents in the network becomes mandatory. In our model, we have proposed hierarchical propagation of interests in the network. There may be different types of interests as follows:

1) Directed and Propagated Interests: In case of directed interests, an agent knows the host or domain from which it is interested in getting data. If the agent has no specific idea about the host or domain it is interested in, the interest is propagated across the entire enterprise network. Such an interest is termed as a propagated interest.

2) Local, Domain and Enterprise Level Interests: Interest can be local-level, domain-level, or enterprise-level depending on whether an agent is interested in getting data only from local host, local domain, or the entire enterprise network. The data that is requested by an agent in the form of an interest is delivered to it by the agent(s) servicing that interest. We propose data delivery using the same hierarchical framework as interest propagation. Direct data delivery without following the hierarchy may not be a feasible proposition in a large enterprise network. In case of data delivery through hierarchical framework, if the higher-level agents are degraded or disabled by an intruder, it will have a significant impact on the detection capability of the IDS. However, as there is no single point of failure, the intrusion detection activity will not stop altogether.

C. Registries

All hosts in the network maintain two major registries: Agent Registry and Interest Registry. The agent registry maintains information about the basic agents running in the hosts, the events they collect, and the alerts they generate. The interest registry is used to keep track of all interests originating from the basic agents in the host and the interests being serviced by the basic agents resident in that host.

IV. ARCHITECTURE OF THE MODEL

Our proposed model is a five-tier hierarchical structure of agents (Fig. 1). There are five types of agents at the five levels of hierarchy: Local Agents or Basic Agents (BAs), Workstation Coordinator Agents (WCAs), Domain Coordinator Agents (DCAs), Enterprise Coordinator Agent (ECA), and Security policy Manager Agent (SMA). The functions of these agents are described briefly.

A. Basic Agents

A workstation can be logically split into multiple entities performing different functions. Each entity is monitored by a set of BAs. Every BA is responsible for monitoring a part of the system resource, user system, and so on. These agents collect and analyze data flowing into or out of this workstation according to the analysis results. Every BA works independently and concurrently with respect to other BAs. The monitoring BA reports the situation detected on an entity, or gives an alarm to the local system user, or takes appropriate measures to prevent the perceived threat if necessary (Fig. 2). If a BA encounters a situation where it does not have appropriate data to ascertain whether an intrusion is taking place, it expresses an interest in that data and propagates that interest to the WCA on that workstation. The WCA then communicates with other BAs on that workstation and directs them to deliver the data of interest to that BA.

B. Workstation Coordinator Agents

Every workstation has one WCA running on it. The WCAs serve as links between the BAs and the DCAs. All WCAs residing in the adjacent hosts in the same domain have direct communication links between them. When a WCA receives an interest from a BA, it first checks the type of the interest. If the interest is a host-directed or a domain-directed interest to a domain different from the domain to which the WCA belongs, the interest is forwarded to the DCA above the WCA in the agent hierarchy. If the interest is a host-directed interest belonging to the same domain but not to the workstation on which the WCA resides, the interest is forwarded to the appropriate WCA of that host. Otherwise, it refers to the agent registry and determines if there is any agent on this host that can service that interest. If any such agent is found, the
WCA informs that agent to send the requested events or alerts to it and also updates the interest registry in the local host. If no agent on the host is found that can service the interest, and if the interest is a local-level interest, the WCA informs the requesting agent about its inability to service that interest. When a WCA receives an interest from the DCA above or a WCA in the same domain, it looks up the agent registry to determine if there is any agent on that host that can service this interest. If any such agent is found, the WCA requests that agent to send the required data to it, and updates the interest registry. If no such agent is found, the WCA discards the request.

On receiving data from a BA in the local host, the WCA looks up the interest registry and sends the data to the BAs whose interest matches the data received. If there is any agent belonging to a different domain that have registered an interest for the same data, the WCA forwards the data to the DCA that handles the further transfer of the data. When a WCA receives data from a DCA, it looks up the interest registry, and sends the data to all the BAs in the local host whose interest matches the data received.

C. Domain Coordinator Agents

Every domain has a DCA. The DCA in each domain has knowledge of all the WCAs in its domain and has direct communication links to each of them. Each DCA is also directly linked to the ECA. The main function of DCAs is to propagate interests and data among WCAs and BAs belonging to separate domains.

An interest received from the ECA can only be a propagated interest because any host-directed or domain-directed interests would have been taken care of by appropriate WCA or DCA. The propagated interest received from the ECA is sent to all hosts to which the DCA is connected. An interest received from a DCA of a different domain can be of two types: a host-directed interest to a host in this domain or a domain-directed interest to this domain. The interests of the first type are forwarded to the appropriate hosts, while those of the second type are forwarded to all the hosts in this domain.

An interest received from a WCA in the same domain of the DCA can be of three types: a host-directed interest to a host in a different domain or a domain-directed interest to a different domain or a propagated interest. The interests of the first and second types are sent to the appropriate DCA for further transfer. The interests of the third type are forwarded to the ECA. An interest directed to a host in the same domain is forwarded to that host. If the directed host is not in the same domain as that of DCA, the DCA forwards it to the ECA. Similarly, a domain-directed interest is sent to all hosts in the domain if the domain specified is its own domain, else it is forwarded to the ECA. Domain-level interests are forwarded to all hosts in its domain. Enterprise-level and propagated interests are not only sent to all hosts in its domain but also forwarded to the ECA.

Data received form the ECA is forwarded only to the host specified. Data received from a host in the domain of the DCA is sent to the DCA of that sent the interest.

D. Enterprise Coordinator Agent

The ECA is at the top of all DCAs in the agent hierarchy, monitoring all the domains under it. Its functionality is similar to but simpler than that of a DCA in the sense that it only has to keep track of the domains under it, and forward data and interests between them based on their nature.

E. Security Policy Manager Agent

The SMA manages the security policies specified by the site security officer. The site security officer interacts with the agents from a high level using security policies. Security policies specify the agents’ roles and responsibilities, and the behavior they should exhibit when they receive events or alerts.

F. Agent Communication Mechanism

The secure transmission of messages among different agents is of paramount importance in a distributed IDS. We have distinguished between two categories of communication while developing the communication system among the agents in our prototype IDS: (i) communication among the agents in the same host, (ii) communication among the agents residing in different hosts. We have used XML (eXtensible Markup Language) as the language for message passing among the agents. The important messages exchanged between the agents are:
The message communication between the agents in the same host need not be transmitted through the network layer. Mechanisms like pipes, message queues, shared memory can be used for this purpose. We have used shared memory mechanism as it allows a large volume of data to be shared and is efficient for one-to-many communication [5,8]. For message-communication between agents residing in different hosts, we propose a system where all communications take place through an Agent Management System (AMS). We have used AMS model of JADE [22] for this purpose. This reduces the cost of communication, as the methods of communication are not replicated for each agent.

We have implemented a Public Key Infrastructure (PKI) model for providing two-way authentication of messages and agents. All agents provide their digital certificates to the registry. The certificates are provided to each agent in advance. This prevents any malicious agent from authenticating itself to the registry. All messages sent among the agents are signed. Signature verification is done by obtaining the agent’s public key from the registry. The messages among the agents are encrypted by shared key encryption method and the PKI is used for the key management. The message communication system has been implemented using Reliable UDP (RUDP). RUDP has all the features of TCP except congestion control and slow start. This makes the communication between the agents faster without compromising on the reliability of communication.

G. Some Implementation Details

The proof-of-concept prototype has been built with JADE [13] and Java. JADE (Java Agent Development Environment) is a middleware developed by TILAB that enables faster development of multi-agent distributed applications based on the peer-to-peer communication architecture. JADE has been implemented fully in Java. It includes both the libraries (i.e. the Java classes) required to develop application agents, and the run-time environment that provides the basic services and that must be active on the host before agents can be executed. From the functional point of view, JADE provides all the basic services necessary for distributed per-to-peer applications. It allows each agent to dynamically discover other agents and to communicate with them by message passing mechanism. The agents communicate by exchanging asynchronous messages - the communication model universally accepted for distributed systems. Each agent is identified by a unique identifier and provides a set of services. An agent can register its services and search for other agents providing given services. It can control its life cycle also.

In our system, each agent has three behaviors: filtering, interaction, and deliberation. The filtering behavior filters security events. When an event occurs in the network, it is collected by an agent only if it matches with the event classes specified in the detection goal of the agent. The interaction behavior manages the interaction between the agents. It defines the mailbox of the agent, and the way the messages are received and enqueued for later interpretation. The deliberation behavior represents beliefs, goals, intentions, and knowledge of the agents. When an agent receives a detection goal, it updates a set of event classes to filter. When an event occurs, it is filtered by the filtering module and sent to the deliberation module. The deliberation module updates /creates the agent’s beliefs, and tests whether the belief matches with an attack signature. If it matches, then a detection goal is reached and a list of intentions are sent to the interaction module for execution.

Implementation of the agents under JADE framework involve the following steps:
1) Determination of the agent behaviors.
2) Implementation of the agent class (extending the existing classes of JADE).
3) Implementation of the agent meta-behavior by instantiating an existing class or introducing a new class and then instantiating it. The meta-behavior provides an agent with a self-control mechanism to dynamically schedule its behaviors in accordance with its internal state.
4) Instantiation of the agent class.
5) Initializing the agent acquaintances.
6) Deployment and activation of the agent.

V. EXPERIMENTS AND RESULTS

To test our prototype, we conducted experiments. In our experiments, we have used the KDD Cup 1999 intrusion detection contest data [14]. This data was compiled during 1998 DARPA intrusion detection and evaluation program by MIT Lincoln Lab. The original dataset consisted of 4.94 million records. There were 41 attributes for each record. Each record was also given a class label that specified the category of attack to which the record belonged. All the attacks were grouped into 4 major categories: i) Denial of Service (DoS), ii) Remote to User (R2U), iii) User to Root (U2R), and iv) Probing (Probe).

We constructed a dataset consisting of 15000 records by randomly selecting records from the original database, such that the number of data instances selected from each class was proportional to their frequencies in the original database. We added one more class of records that we call ‘normal’ class
apart from the 4 attack types mentioned above. The knowledge about these attacks is then distributed among the agents in the system.

We tested the performance of the prototype in a network of workstations each having Pentium 4 processor with 3GHz clock speed, 128 MB RAM, and Red Hat Linux version 9 as the operating system. The data-rate of the Ethernet was 100 MBPS. The agents were installed in the workstation in a hierarchical form (Section IV).

Using Ethereal network sniffer (a software to capture and analyze information being transmitted over a network), we monitored the network to determine the bandwidth consumption by the agents. We first monitored the network with no agents running on any workstation, so as to establish the baseline utilization. We then activated all the agents in each workstation and simulated attacks from different workstations. From the data collected by the sniffer, it was evident that the agents had very little bandwidth consumption. During the one-hour time when the agents were active on the workstations, we had only 15% increase in number of packets captured in the network by the sniffer compared with the one-hour period when the network was monitored without activating the agents. The average bandwidth consumption by the agents never exceeded 5% of the 100 MBPS Ethernet.

To test the CPU utilization by the agents, we selected some workstations and then activated some user programs on them. The IDS agents were then activated on those workstations, and some elementary attacks like password guessing were simulated on those workstations. During the thirty-minute analysis period, the maximum CPU utilization of the agents was found to be 12.54%, the average utilization of the entire period being 0.59%.

<table>
<thead>
<tr>
<th>Activity Type</th>
<th>Successful Detection Rate</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denial of Service</td>
<td>93.28%</td>
<td>11.42%</td>
</tr>
<tr>
<td>Remote to User</td>
<td>89.41%</td>
<td>16.53%</td>
</tr>
<tr>
<td>User to Root</td>
<td>85.63%</td>
<td>18.34%</td>
</tr>
<tr>
<td>Probe</td>
<td>92.26%</td>
<td>14.38%</td>
</tr>
<tr>
<td>Normal</td>
<td>96.30%</td>
<td>8.24%</td>
</tr>
</tbody>
</table>

Finally for testing the overall performance of the IDS, in terms of its attack detection capability and reduced rate of false positives, we simulated 37 different attacks from some workstations to other workstations in the network. Some of the attacks were not included in the knowledge base of the agents. This was done to test the ability of the IDS to identify novel attacks. Table 1 summarizes the results of the experiment. The results show that the performance of the prototype is quite encouraging particularly in terms of successful detection of attacks.

VI. CONCLUSION

In this paper, we have discussed some current IDSs and their limitations in terms of certain features like fault-tolerance. We have noted that flexibility, autonomy, adaptability, and distribution of operations are some of the important parameters to be addressed to build an effective IDS that can be used in real world to detect complex and distributed attacks. We have presented an architecture with agents that perform intrusion related data collection and analysis in a truly distributed manner. We have implemented a proof-of-concept prototype and conducted experiments on the system. The results show the effectiveness of the system in real-time detection of intrusions.

REFERENCES