Lightweight agents for intrusion detection ☆

Guy Helmer, Johnny S.K. Wong *, Vasant Honavar, Les Miller, Yanxin Wang

Department of Computer Science, Iowa State University, 225 Computer Science Building, Ames, IA 50011-1041, USA

Abstract

We have designed and implemented an intrusion detection system (IDS) prototype based on mobile agents. Our agents travel between monitored systems in a network of distributed systems, obtain information from data cleaning agents, classify and correlate information, and report the information to a user interface and database via mediators.

Agent systems with lightweight agent support allow runtime addition of new capabilities to agents. We describe the design of our Multi-agent IDS and show how lightweight agent capabilities allowed us to add communication and collaboration capabilities to the mobile agents in our IDS.

1. Introduction

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

Detecting intrusions (Denning, 1987) in a distributed system is a difficult problem. IDSs must analyze large volumes of data while not placing a significant added load on the monitored systems and networks. Data must be obtained from sources distributed around the computing system. Intrusions take place at all levels of the distributed system, from physical components up to applications; each level may require monitoring. Data from various sources must be aggregated and correlated to determine whether an intrusion is taking place. Intrusions should be detected as soon as possible to allow immediate and effective countermeasures to be executed.

Our research group is applying distributed knowledge networks (Honavar et al., 1998) and data warehouse techniques to IDSs. Distributed knowledge networks use agents for information retrieval and extraction, data transformation and knowledge discovery. Data warehouse technologies are used for data and knowledge organization and assimilation from heterogeneous physically distributed data and knowledge sources.

Software agents consist of program code and state and exist to perform tasks on behalf of a user with some degree of autonomy. A software agent's goal may require some degree of intelligence, allowing it to react to its environment, make plans to achieve its goal, maximize its utility, and/or modify its behavior over time (Honavar, 1998). Software agents may use mobility to travel to sources of data and remotely execute their tasks, resulting in a natural distribution of work and reduced communication overhead.

Lightweight agents are agents that accomplish their essential tasks with minimal code. They are dynamically updatable and upgradable, smaller, simpler, and faster to transport (due to their smaller size). For example, in distributed computing system, there are many operating systems. For an intrusion detection agent to effectively operate in these systems, it needs the ability to process the information in the system it will immigrate to and

---

* Corresponding author. Tel.: +1-515-294-2586; fax: +1-515-294-0258.

E-mail addresses: ghelmer@cs.iastate.edu (G. Helmer), wong@cs.iastate.edu (J.S.K. Wong), honavar@cs.iastate.edu (V. Honavar), lmiller@cs.iastate.edu (L. Miller), wangyx@cs.iastate.edu (Y. Wang).

carry the detection rules that will apply in the system which it will immigrate to. If the agent is designed not using lightweight agent concept, it will always carry all the rules that are needed for all systems and will be bigger and waste resources.

With lightweight agent concept in mind, the design will be based on simplicity and minimalism. An agent will only carry the primary features to make it lightweight; after it arrives the destination system, it will be upgraded and updated as necessary for the situation. Using this design objective, the system will save resources because many agents are needed to immigrate to monitored system. Lightweight agents will reduce the network bandwidth and CPU time required to move the agent. Voyager is the only commercial mobile agent platform currently supporting dynamic upgrading of agent, so we use Voyager as our lightweight agent platform. The lightweight agent design also allowed us to quickly add features to our IDS.

A mobile agent is a program that acts on behalf of a user or another program and is able to migrate from host to host on a network under its own control. The agent chooses when and to where it will migrate and may interrupt its own execution and continue elsewhere on the network. The agent returns results and messages in an asynchronous fashion (University of Ottawa, 2000). mobile agents do not require network connectivity with remote services to interact with them and network connections are used for one-shot transmission of data (the agent and possibly its state and data). Results in the form of data do not necessarily return to the user using the same communication trajectory, if indeed the results are to be returned at the originating site. Alternatively, the agent may send itself to another intermediate node and take its partial results with it. Results are delivered back to the user whose address the agent knows. Mobile agent implement the distribute architecture, and it is better than the client–server paradigm, which is the most common way of implementing distributed applications. In this model, a network connection must be established between client and server. This Paradigm breaks down under the situations dealing with highly distributed problems, slow and/or poor quality network connections, and especially in the maintenance of constantly changing applications. In a system with a single central server and numerous clients, there is a problem of scalability. When multiple servers become involved, the scaling problems multiply rapidly, as each client must manage and maintain connections with multiple servers. Finally, the protocol which a client and a server agree upon is by its very nature specialized and static. Often, specific procedures on the server are coded in the protocol and become a part of interface. Certain classes of data types are bound to these procedures and the result is a special network version of an application program interface. The interface is extensible, but only at the high cost of re-coding the application to provide for protocol version compatibility and software upgrades.

Mobile agents overcome all these inherent limitations in the client–server paradigm. First and foremost, the mobile agent paradigm shatters the very notion of client and server. With mobile agents, the flow of control actually moves across the network instead of using the request/response architecture of client–server paradigm. In effect, every node is a server in the agent network and the agent moves to the location where it may find the services it needs to run at each point in its execution.

The scaling of servers and connections then becomes a straightforward capacity issue, without the complicated exponential scaling required between multiple servers. The relationship between users and servers is coded into each agent instead of being pieced out across clients and servers. The agent itself creates the system, rather than the network or the system administrators. Server administration becomes a matter simply of managing systems and monitoring local load.

The problem of robust networks is greatly diminished for several reasons. The hold time for connection is reduced to only the time required to move the agent in or out of the machine. Because the agent carries its own credentials, the connection is simply a conduit and is not tied to user authentication or vulnerable to spoofing. No requests flow across the connection; the agent itself moves only once.

Last and most important, no application-level protocol is created by the use of agents. Therefore, compatibility is provided for any agent-based application. Complete upward compatibility becomes the norm rather than a problem to be tackled, and upgrading or reconfiguring an application may be done without regard to client deployment. Servers can be upgraded, services moved, load balancing interposed, and security policy enforced, without interruptions or revisions to the network and clients.

So mobile agents have many advantages for distributed architecture. Distributed mobile autonomous agents solve critical problems in intrusion detection, such as bandwidth, computing cycles on user's computers, efficiency, reliability, and they provide a general architecture for adding and integrating "components" into the system. Monolithic, centralized systems have several faults which may be overcome by the use of a distributed architecture. Mobile agents diminish the weaknesses that would be added by a client–server paradigm to a distributed architecture.

Network intrusion detectors typically use single sensors attached to network segments. However, local area networks have moved towards switched architectures which do not broadcast unicast frames to all network segments. Centralized sensors will miss traffic on segments to which the sensor is not attached in a switched

environment. Distributed agents solve this problem by monitoring network activity at each host.

Network intrusion detectors also have problems with high data rates. Centralized systems may miss packets under heavy load situations on Fast or Gigabit Ethernet networks. Distributed agents distribute the processing effort between the networked systems and likely would improve the chances that intrusions will be detected that would be missed by a centralized IDS.

The modular architecture of the agent-based IDS allows for the integration of intrusion detection components from other projects. For example, the SNORT network IDS (Roesch, 1999) is a small, fast, low-overhead sensor that watches for network packets that match signatures. The agent-based IDS can incorporate SNORT and provide network misuse detection at each host. Likewise, various small programs detect specific intrusions such as port scanning, broadcast pings (smurfs), logins at unusual times, and changes to files. These can be included in the distributed IDS via wrapper agents.

A centralized or centrally-governed IDS provides a single point of failure and a single target for an attack. The distributed agent architecture avoids a central point of failure. Autonomous agents may continue operating despite the failure of other agent servers or other failures in a system, which avoids the compromise of the entire IDS even if one component fails or is attacked (Mell and McLarnon, 1999). Mobile agents also may be capable of evading attackers and resurrecting themselves if killed.

A distributed agent-based IDS helps solve spatial problems in intrusion detection, where more than one host is involved in an intrusion. For example, in an "FTP bounce" attack, an attacker may use an anonymous FTP server on one host to spoof a command to a remote shell on the target host. Agents on both the anonymous FTP server and the target host could detect the spatially-separate events in the attack and correlate the events in near real-time.

Kumar (1995) lists shortcomings of IDSs. Viewed in a different way, the shortcomings provide a list of desirable features in an IDS.

Generic Architecture. The common intrusion detection framework (CIDF, Porras et al., 1999) specifies a generic architecture for an IDS and classifies the components of an IDS. A system of distributed mobile agents implements the IDS in a flexible way compatible with the CIDF architecture.

Efficiency. A distributed Multi-agent System obtains audit data at the appropriate levels of the distributed system and distributes the information processing and intrusion detection effort.

Portability. IDSs have tended to be developed with an orientation to an organization's security policy. Differences in security policies between organizations re-

sult in a lack of portability of an IDS. In a different sense, portability of the IDS with respect to operating systems and computer architecture is also an issue. Perl and Java, two interpreted languages, have been used to provide portability for IDSs. Autonomous agents for intrusion detection (AAFID, Balasubramaniyan et al., 1998) and our own project, MAIDS, are two such IDS's.

Upgradability. An IDS based on a component-based architecture such as that available in an agent-based system satisfies the upgradability and enhancement concern. New features can easily be added to such a system.

Maintenance. Maintaining and updating the learned knowledge used by components of an IDS would depend on the architecture of the components.

Performance benchmarks. Exhaustive quantitative performance evaluations of current IDSs in real-world environments do not exist. Vulnerability coverage is beginning to be addressed by vendors with the assistance of vulnerability assessment projects, including the Common Vulnerability Enumeration (Mann, 1999).

Testing. Kumar says, "there is no easy way to test intrusion detection systems." (Kumar, 1995) The MIT Lincoln Labs intrusion detection evaluation (Lippmann et al., 1998) was one of the first major tests of research IDSs. Attacks remain difficult to simulate, effectiveness of systems is difficult to evaluate without operating them under real world loads, and significant tuning and expertise tends to be necessary to operate systems.

The Multi-agent IDS is similar to other intrusion detection systems in that its effectiveness would need to be demonstrated in real-world settings.

This modular and extensible approach to building a system helps solve the complex problems in an IDS. It divides the problem into the aspects of information retrieval, classification, collaboration, and compilation. Agents have been developed for our system that retrieve information from distributed systems, classifies the data (either using embedded expert rules or machine learning techniques), and stores the data in a database.

In this paper, we briefly examine the design of our distributed IDS and then explain the addition of agent collaboration to the system by the use of dynamic aggregation. We show how dynamic aggregation provides a convenient mechanism for extending existing objects and allows us to quickly add new features to the system.

In the initial design of our IDS, agent communication was vertical: information traveled between a data gathering agent up to its associated mobile agent, and from there up to the user interface. In this design, though,

agents were not able to communicate horizontally with each other to cooperatively detect intrusions.

For horizontal communication, we added the concept of agent sensitivity, where an agent would become more sensitive to unusual events when intrusive events have been noticed by other agents. As an example of sensitivity, consider failed logins. A few login failures on a single host may be normal, such as when a user forgets her password. However, when an attacker has identified a target host (perhaps by portscanning, which is considered intrusive by our system), he may connect to the target host and try a few typical passwords. In this case, a loose temporal relation exists between the first event (portscanning) and the second event (failed login attempts). Agent sensitivity levels can provide a real-time correlation of related intrusions while allowing normal, individual events to pass without triggering alarms. We used dynamic aggregation to add sensitivity and horizontal communication to the agents.

Our implementation of the system currently includes:

- Static data cleaning agents that obtain information from system logs, audit data, and operational statistics and render the information in a common format;
- Low-level agents that monitor and classify ongoing activities, classify events, and pass on their information to mediators;
- Facets for the low-level agents that add cooperation to the agents;

- Data mining agents that use machine learning to acquire predictive rules for intrusion detection from system logs and audit data.

The Multi-agents Intrusion Detection Model is based on the idea of distributed knowledge networks. The architecture illustrated in Fig. 1 makes use of the following layers:

User Interface. The user interface allows control of agents, management of the list of monitored systems, and reports intrusions.
Database. The database stores data for training and off-line intrusion detection.
Data fusion and data mining. Agents at this level can fuse information from lower agents and mine knowledge from the database.
Mediators. The low-level agents are managed by mediators which control the systems visited by the agents, obtain the classified data from the agents, and route the data into the local database and to the user interface. As the system is developed, the mediators will apply data mining algorithms to the data in the database to connect individual events into a cohesive view of the elements involved in an attack.
Data gathering, base classification, and basic data mining. In the middle of the architecture, the low-level, mobile agents form the first line of intrusion detection. They periodically travel to each of their associ-
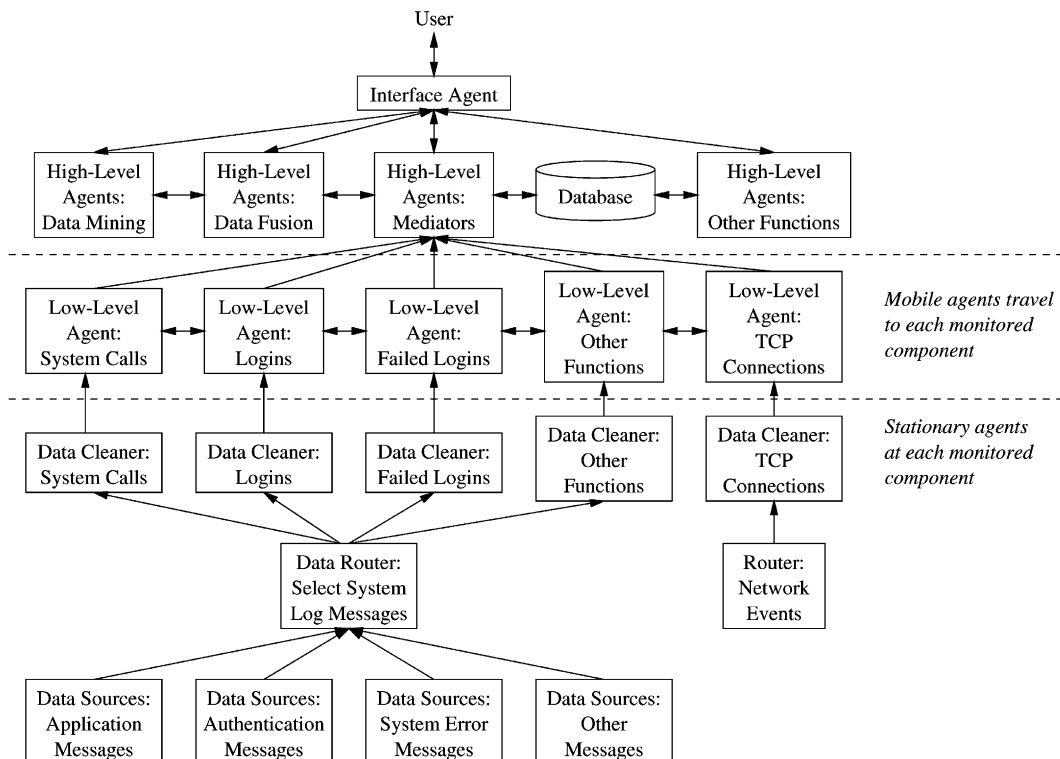

Fig. 1. Architecture of the Mobile-agents Intrusion Detection System.

ated data cleaning agents, obtain the recently gleaned information, and classify the data to determine whether singular intrusions have occurred.

Data cleaning and formatting. At the bottom of the tiered architecture, the system log routers and system activity agents read log files and monitor the operation of the systems. The routers feed data into the distributed data cleaning agents which previously registered their interest in particular events. Targeted data cleaning agents process data obtained from the routers and activity agents. They render the data into common data formats.

The hierarchical architecture for our IDS agent system has following advantages:

1. The implementation of agent is efficient. When low-level agent travel to monitored system, mediator parts need not to travel. Many low-level agents are generated and migrating to monitored system, the mediator part need not to be generated many times and need not to migrate. Much network bandwidth and CPU time are saved.
2. Layered system is easy to design and modify. Clear organization of the agents make system easy to maintain.
3. It provides platform indecency. The lower levels that need to contact with system logs is platform dependent. When new operating system is added, only lowest level agent-data collecting agents need to be added.

The delay on reporting intrusion of monitored components in hierarchical architecture will be trivial compare to the advantages we benefit from the hierarchical architecture. The agents can be lightweight in the hierarchical system architecture and performance is greatly improved.

As we further develop the system, multiple departmental-level systems can be monitored. Data warehousing can be used to combine the knowledge and data from the individual departments into an organization-wide view of attacks. The agent system of Fig. 1 is targeted to run at the departmental level of an organization. To provide enterprise-wide information about intrusions, the data from each departmental agent system will feed into a data warehouse as shown in Fig. 2.

Because the data warehouse will provide a global view of the intrusion detection systems, it supports not only the identification of attacks but also:

- helps administrators discover new attacks,
- trains system administrators about how attacks are mounted on their systems, and
- identifies weak points in the enterprise information systems.

A number of agent infrastructures exist to support agent systems. Generally, agents infrastructures provide agent servers, agent interfaces, and agent brokers. Agents servers may provide mobility and authentication. Agent interfaces are used by application programs to create and communicate with agents. Agent brokers provide naming and location services. The prototype mobile agent IDS has been built in the Java language using the Voyager Object Request Broker (ObjectSpace Inc., 1999) . Voyager provides the mobility, interfacing, and naming services needed to implement the agents in the IDS. Voyager is a platform that is easily to use, freely available on the web and 100% percent Java, so the applications developed based on Voyager are portable and compatible with multiple operating systems and hardware platforms. Voyager is also the only commercial mobile agent platform currently supporting dynamic aggregation, which is the feature we used to design our lightweight agent system. Based on these advantages, we used Voyager for our agent platform.
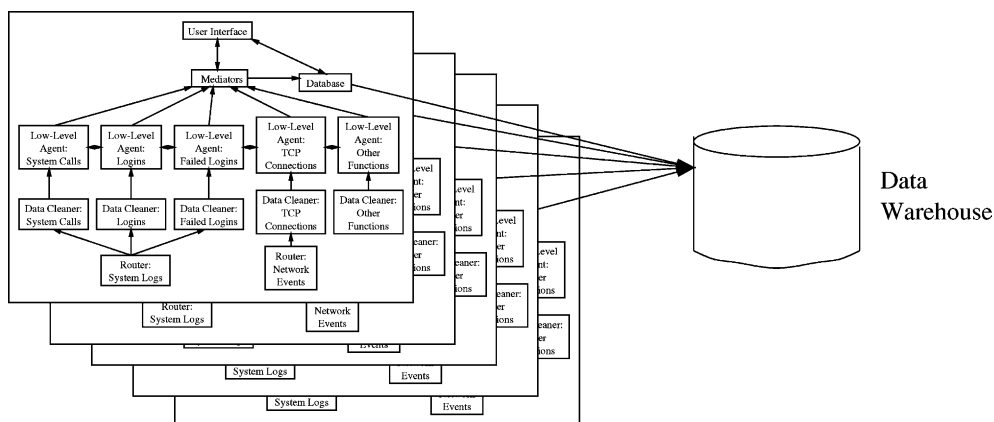

Fig. 2. Enterprise date warehouse for intrusion detection.

2. Related work

2.1. Distributed intrusion detection systems

The distributed intrusion detection system (DIDS) of the University of California—Davis (Mukherjee et al., 1994) used a combination of host and LAN monitors to observe system and network activity. A centralized director obtained information from the monitors to detect intrusions. DIDS is similar to our system in that it used multiple, distributed monitors, akin to our agents, and artificial intelligence algorithms to discover anomalous events. DIDS differs from our system in that the intelligence is purely centralized, and DIDS did not make use of any agent technology.

The CIDF (Reilly and Stillman, 1998) resembles our project general architecture. CIDF is a proposed standard being developed by a group composed of the Information Technology Office of the Defense Advanced Projects Agency, University of California—Davis, Information Sciences Institute, Odessey Research, and others. The CIDF nomenclature differs from ours, including the reconnaissance agents which correspond to our data gathering agents, the analysis agents correspond to our low-level agents and facets, and the decision-response agents correspond to our high-level agents. In general, the CIDF provides a model with which we can compare our system, but it does not specially related to agents.

2.2. Agents in intrusion detection systems

Several projects have investigated the use of agents in intrusion detection systems. The Computer Immunology Project, Java agents for meta-learning (JAM), and AAFID projects each examined the problem in different ways.

The AAFID project (Balasubramaniyan et al., 1998) at Purdue's COAST project is a flexible, distributed IDS infrastructure with some similarity to the MAIDS design. It has developed an IDS using Perl agents for fast prototyping and cross-platform compatibility. The communication between agents are incidental. Their project also analyzed the agent-based approach to intrusion detection. Our approach differs in the emphasis on the use of learning algorithms, data warehousing, and mobile agents. Our system is implemented in Java.

The Computer Immunology Project at the University of New Mexico (Warrender et al., 1999; Forrest et al., 1997, 1996) explored designs of IDSs based on ideas gleaned by examining animal immune systems. Small, individual agents would roam a distributed system, identify intrusions, and resolve the intrusions. One portion of the project developed a sense of self for security-related computer programs by observing the normal sets of system calls executed by the programs.

This sense of self can be used to detect intrusions by discovering when a program executes an unusual set of system calls. The Computer Immunology Project differs from our project by their focus on individual agents rather than an integrated system of cooperating Multi-agents.

The JAM Project at Columbia University (Lee and Stolfo, 1998; Stolfo et al., 1997) uses intelligent, distributed Java agents and data mining to learn models of fraud and intrusive behavior that can be shared between organizations. Our project differs in its concentration on intrusion detection within a single organization and its use of data warehousing to aggregate department data and provide organization-wide views of security information.

3. Lightweight agents

Dynamic aggregation allows runtime addition of new capabilities to agents. This section explains the addition of agent collaboration to the system by the use of dynamic aggregation.

We call the small, minimal agents *lightweight* because the agents implement a minimum of functionality, as opposed to *heavyweight* agents that include all functions that may ever be needed. Compared to heavyweight agents, lightweight agents are:

- Smaller,
- Simpler,
- Faster to transport (due to their smaller size),
- Dynamically updatable and upgradable.

Dynamic aggregation allowed us to add collaboration capabilities to the lightweight mobile agents and quickly add new features to the system. Our particular use of dynamic aggregation allows the agents in our IDS to inform each other about intrusive activity.

Each agent uses dynamic aggregation to manage its sensitivity level. The sensitivity level determines how sensitive an agent is to events which may not be considered intrusive under normal circumstances but which can be intrusive in the presence of related intrusions.

An example of the sensitivity issue is the problem of failed login attempts. A few login failures on a single host may be normal, such as when a user forgets her password. However, when an attacker has identified a target host (perhaps by portscanning, which is considered intrusive by our system), he may connect to the target host and try a few typical passwords. In this case, a loose temporal relation exists between the first event (portscanning) and the second event (failed login attempts). Agent sensitivity levels can provide a real-time correlation of related intrusions while allowing normal, individual events to pass without triggering alarms.

3.1. Adding agent capabilities with dynamic aggregation

Dynamic aggregation of objects offers three benefits (ObjectSpace Inc., 1999).

- The behavior of a binary-only object may be extended.
- An object may be customized in specific ways.
- An object's behavior may be extended at runtime, in ways that need not be specified at compile time.

To this list of benefits we add:

- An object is as small as it can be until new functionality is needed.

The Voyager terminology for objects used in dynamic aggregation is "facets". Voyager defines a primary object and its facets to be an "aggregate" that is managed as a single unit. Voyager's facet selection rules conveniently select a facet implementation tailored to a particular object. This allows specific customization of an object.

Voyager selects an implementation of a facet based on the name of the facet and the class name of the primary object. When a facet of a particular class is requested, Voyager searches for a facet that implements an interface by the facet class's name with an I prefixed. For example, if the facet by the name of AFacet were requested, Voyager would search for a class that implements the interface IAFacet.

Voyager searches objects for implementations of the facet's interface using the name of the primary object suffixed by the name of the facet. If a matching implementation is not found, Voyager searches using the primary object's superclasses. For example, if the primary object were of the class aClass (which extends java.lang.Object) and the facet AFacet were requested, Voyager would first look for a facet named aClassAFacet, then for ObjectAFacet, and finally AFacet.

When Voyager looks for an implementation for the facet, it searches the primary object's package and the facet's package.

3.2. Use of dynamic aggregation

Intelligent agents in the Multi-agents Intrusion Detection System are lightweight in that they are oriented towards gathering and classifying data. The low-level agents themselves communicate directly only to their related data gathering agents and mediators. Adding low-level agent communication in the IDS allows agents to fuse related data in real time and take advantage of knowledge about the security status of related components in the system. Dynamic aggregation provides a convenient way of adding communication between low-level agents without adding excess baggage to the agents themselves.

Inter-agent communication was implemented by the use of sensitivity facets as an aggregate to the low-level agents. Sensitivity facets are a family of objects that communicate intrusion information among themselves and use information about related intrusions to affect future decisions about intrusions.

The issue of failed attempts to login provides a good example of how the sensitivity facets are used. A few failed logins in a distributed system are expected as users tend to forget passwords, try wrong passwords on the wrong systems or mistype as they try to login. However, in the face of an attack, a few failed logins may signal the next step in an intrusion as the attacker tries commonly-used or default passwords after having identified systems that allow virtual terminal or file transfer connections from the network. The failed logins sensitivity facets listen for network connection events from unusual sources (which occur as the attacker identifies her targets), remember these events, and increase their sensitivity to failed logins for a period of time.

Agent collaboration in the IDS was not developed until after the agents had been designed, developed, and tested. Dynamic aggregation provided a way in which the agents could be extended without having to overload an agent with new features. However, the agents needed a bit of redesign to accommodate the sensitivity facets. Each adjustment facet must integrate itself into its primary agent to affect decisions on whether events are intrusive. The agents were modified to provide newly-obtained events to a listener (namely, the adjustment facet) for adjustment of the event's intrusion classification. The agents were also modified to notify a listener (the reporting facet) if an event was determined to be intrusive. This adjustment to the agent architecture makes the agents much more flexible and open to future enhancement by aggregation.

The reporting facets were designed to broadcast intrusion messages from their primary agent to other interested facets. The adjustment facets listen for intrusion messages from the reporting facets. Fig. 3 illustrates the flow of information between the agents and facets.

3.3. Communication in the IDS

Auditing information is exchanged between agents using subclasses of an Audit class. The Audit class provides methods for serializing an audit object in a table of keys and values, deserializing an audit object, storing an audit object in a database, and obtaining an audit object from a database. Subclasses of the Audit class fully describe events like logins, changes to important files, network connections, and processes executed. Low-level agents obtain audit objects from data gathering agents
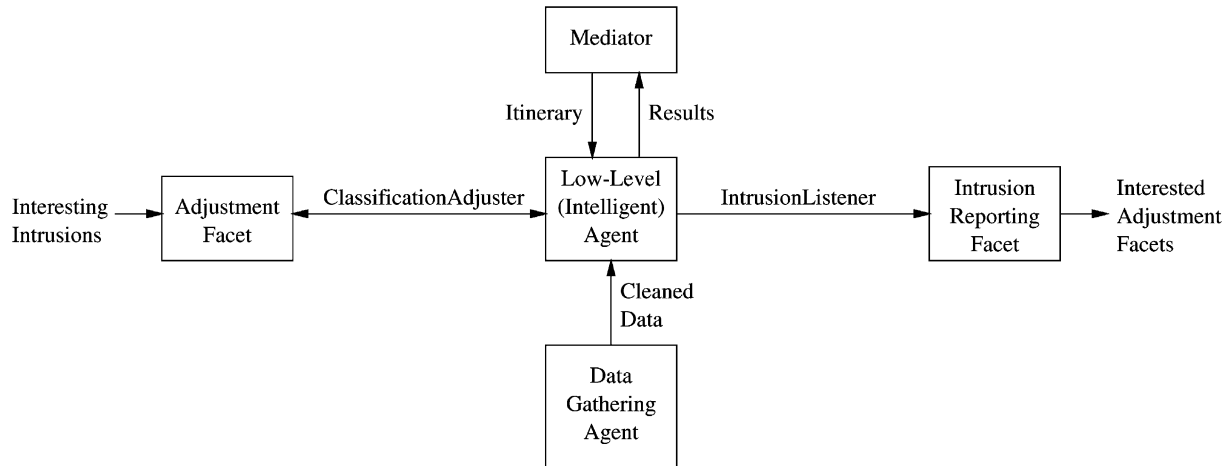
Fig. 3. Facets for the low-level agents.

and apply the first level of intrusion detection, using artificial intelligence or expert rules. Audit objects are passed vertically through the IDS (see Fig. 1).

Intrusion information is exchanged between facets using descriptive strings in ASN.l syntax. In the style advocated by Tim Bass (Bass, 1999), a small management information base (MIB) was designed to encapsulate information about intrusions. The MIB allows the low-level agents to communicate intrusion information at a higher, more abstract level and describe more knowledge about the intrusion. A sample of the ASN.l hierarchy of intrusions follows.

host Intrusions identified on a host computer
    host.priv_prog Intrusions against a privileged program
    host.auth Intrusions related to authentication
net Intrusions related to network services
    net.ip Intrusions via the IP network protocol
    net.ip.icmp Intrusions via the ICMP protocol
    net.ip.udp Intrusions via the UDP protocol
        net.ip.udp.service = nfs Intrusions via NFS
    net.ip.tcp Intrusions via the TCP protocol
        net.ip.tcp.service = login Intrusions via the rlogin protocol

Listeners can register interest in any portion of the hierarchy by specifying a prefix. For example, a listener could register the prefix `net.ip.tcp` to listen for all TCP-related intrusions.

3.4. Communicating between facets

The intrusions detected by each agent in the IDS were examined for related intrusions. Examples of such relations are shown in Table 1.

Reporting facets were constructed to encode intrusion information in the MIB described previously and send the reports to interested facets. Adjusting facets encode the relationships between intrusions by registering interest in related intrusions. Adjustment facets interpret the received intrusion messages to affect their sensitivity level and use their sensitivity level to adjust intrusion classifications of events via their related low-level agent.

3.5. Machine learning method used by low-level agent

In this project, we use distributed intelligent agent for intrusion detection. A lower-level layer of agents, just above the data cleaning agents in the system architecture, form the first level of intrusion detection. Using lightweight mobile agent technology, these agents travel to each of their associated data cleaning agents, gather recent information, and classify the data to determine whether suspicious activity is occurring. The agents are able to use a variety of classification algorithms, the choice of which will depend on data.

In our project, several data cleaning and low-level agents have been implemented. Our research in this part focuses on agents that monitors privileged programs

Table 1
Agents and their related intrusions

Agent	Related intrusion	Rationale
Untrusted connection; NetTCP	Attack on network daemon; Changed configuration file	Attacker will often attempt to login via the network after attacking a network daemon or changing a configuration file
Critical files	Attack on network daemon; NFS	Attacker may change a critical file to enable further attacks
Failed logins	Port scanning	Attacker may scan available network ports to find machines on which to try to login

and algorithms for detecting intrusion based the log data of the privileged programs. Programs that provide network service in distributed computing systems often execute with special privileges. For example, the popular sendmail mail transfer program operates with superuser privileges on UNIX systems. Forrest's project at the University of New Mexico (Forrest et al., 1996) developed databases of systems calls from normal and anomalous uses of privileged programs such as sendmail. Forrest's system call data is a set of files consisting of lines giving a process ID number and system call number. The files are partitioned based on whether they show behavior of normal or anomalous use of the privileged sendmail program running on SunOS 4.1. Forrest organized system call traces into sequence windows to provide context, and showed that a database of known good sequence windows can be developed from a reasonably sized set of non-intrusive sendmail executions. Forrest then showed that intrusive behavior can be determined by finding the percentage of system call sequences that do not match any of the known good sequences. We use the same data set with Forrest's to enable comparison with techniques used in related papers (Lee and Stolfo, 1998; Warrender et al., 1999).

We present a feature vector technique that improves on Forrest's technique because it does not depend on a threshold percentage of abnormal sequences. Our feature vector technique compactly summarizes the vast data obtained from each process, enabling longer-term storage of the data for reference and analysis. With respect to other rule learning techniques, our technique induces a compact rule set that is easily carried in lightweight agents. Our technique also may mine knowledge from the data in a way that can be analyzed by experts.

For details about our work in this aspect, please refer to our these papers (Helmer et al., 1998, 1999).

3.6. Lightweight agents for distributed intrusion detection

Distributed knowledge network include computational tools for accessing, organizing, transforming, and analyzing the contents of heterogeneous, distributed data and knowledge sources and for distributed problem solving and decision making. Our current design of distributed knowledge network consists of the following components: a mobile agent infrastructure; intelligent agent for information extraction, intrusion classification, coordination and control mechanisms for Multi-agent Systems, and distributed intrusion detection based on software fault tree (SFT) and color petri net (CPN) model.

Intrusion data from heterogeneous data sources resides on multiple hardware platforms and operating systems at different geographical locations. This requires a robust and flexible framework for interoperability between the various data sources and clients and good methods for understanding the temporal and spatial relationship of the distributed data.

For assimilation and processing of distributed intrusion information, understanding the temporal and spatial relationship of the distributed data, and correlating intrusion events, we use two methods:

The first method is adding communication between low-level agents, allowing agents to fuse related knowledge and take advantage of knowledge about the security status of related components in the system by dynamic aggregation and sensitivity facets. This has been introduced in detail above in Sections 3.1–3.4.

The second method is fusing the knowledge using SFT and CPN models, which enable understanding and capture of domain knowledge needed to accurately define the requirements of intrusion detection. We use SFT to model the combinations and sequences of events by which intrusions can occur. Then the SFT models of intrusions are used to create CPN designs for the detectors in the IDS. The CPN detector models are then mapped into implementation as mobile agents that form the distributed IDS. For details, please refer to our these papers (Helmer et al., 2001, submitted for publication).

The mobile agent intrusion detection architecture enables efficient, distributed knowledge fusion, intrusion event correlation, and effectively reduces false alarms.

4. Implementation

In this section, the overall implementation of the prototype IDS is discussed. Then, the implementation of dynamic aggregation to add communications between agents is presented.

4.1. IDS implementation

The prototype IDS has been implemented using Sun's Java Development Kit version 1.1 (Sun Microsystems, 2000) and ObjectSpace's Voyager Object Request Broker version 3 (ObjectSpace Inc., 1999).

Java was chosen as the development language because of its platform independence, security, and speed of development. Java's platform independence has allowed us to compile and execute completely shared code on any one of several platforms including Silicon Graphic's IRIX and Hewlett–Packard's HP-UX commercial operating systems and free operating systems including Free BSD and Linux. Java's security features include sandboxes for executing untrusted code, strict typing, bounds checking, byte-code verification, and code signing. Java's strict typing and object orientation assisted development by enforcing structure, allowing the addition of functionality by extending existing objects, and reducing time needed for debugging as compared to

projects of similar complexity that have been implemented in the C language.

The runtime performance of Java code is a concern, but has not been an obstacle to the IDS project. The performance has been sufficient to demonstrate operation of the prototype IDS. When additional performance has been required, just-in-time compilers that compile Java byte code into native machine code have satisfied the requirements.

The Voyager Object Request Broker was chosen for the prototype IDS because of its availability, development in Java as its native language, and support for mobile agents. Voyager provides mobility, message passing (independent of the target object's location), and naming services for Java objects, all of which are necessary for our prototype IDS. Voyager also supports extending lightweight agents via dynamic aggregation, or *facets*. Facets allow us to add capabilities to IDS agents, including agent communication and collaboration. We could also dynamically add countermeasures capabilities to agents. This would allow a running IDS system to adapt to new attacks by implementing new countermeasures.

The bottom tier of agents in the IDS is the set of stationary data cleaning agents. The stationary agents obtain information from sources including system logs, audit data, and operational statistics. The stationary agents convert the information into a common format for use by the higher levels of agents.

An example of a stationary agent is the `DGFailedLoginAgent`, which reads the system logs for reports of failed logins. It parses failed login messages from the system log into `Login` objects which describe when the failed login took place, which user account was used, and which computers were involved in the failed login.

The middle tier of agents in the IDS is the set of low-level agents. Low level agents are mobile agents that gather information from the stationary agents. Low-level agents process the gathered information to monitor and classify events. Low-level agents then pass on the information to their mediator.

An example of a low-level agent is the `FailedLoginAgent`, which visits each host's `DGFailedLoginAgent` to retrieve the recent list of failed logins. If the number of failed logins in the entire distributed system exceeds a threshold in a short period of time, the failed logins are flagged as an attempted intrusion.

The prototype detects intrusions including attempts at trying multiple passwords on multiple machines, unusual TCP network connections, changes in critical files (with the assistance of Tripwire (Kim and Spafford, 1994)), attacks against the sendmail mail transfer agent, and refused connections to unsafe network services.

Note, however, that the flow of information in this design is purely vertical (in terms of Fig. 1) and agents at each level are not cooperating or coordinating with each other. The existing low-level agents are then considered "lightweight" in the view that they do not have the capability to communicate directly with each other.

4.2. Adding cooperation to the agents

Sensitivity facets were created to add communications capabilities to the low-level agents. The Sensitivity facets implement the adjustment and intrusion reporting functions shown in Fig. 3. Mediators construct agents and attach Sensitivity facets to agents. The Sensitivity facets implement the reporting and adjusting functions described in Section 3.3. As agents gather information, the information is passed through the Sensitivity facets for reporting and adjusting.

The interface `ISensitivity` was defined with the methods:

`register( )` To subscribe to events in the associated agent,

`setIntrusionClasses( )` To subscribe to interesting intrusions from other agents,

`getSensitivity( )` To obtain the current sensitivity level,

`sendIntrusionMessage( )` To publish messages about intrusions, and

`recvIntrusionMessage( )` Through which intrusion messages are received by the facet.

A default implementation of the interface, `Sensitivity`, was created. The `register( )` method implementation subscribes to all events seen by its attached agent. The `setIntrusionClasses( )` method implementation defines which intrusions are interesting to this facet. The `getSensitivity( )` method implementation returns an indicator of how sensitive the facet is to suspicious events. The `sendIntrusionMessage( )` method implementation publishes information about an intrusive event to other listening Sensitivity facets. The `recvIntrusionMessage( )` method implementation receives messages about intrusive events from other Sensitivity facets and adjusts the facet's sensitivity level based on the significance of the event.

Fig. 4 shows a slice of the IDS with `Sensitivity` facets included. A specific `Sensitivity` facet, `NetTCPAgentSensitivity`, extends the basic `Sensitivity` facet to listen for intrusive activity including changes to files and buffer overflow attacks. If other agents report these intrusive activities, the sensitivity of the `NetTCPAgent` should be raised since the target may receive anomalous TCP connections in the near future which may be part of the intrusion. Because of Voyager's rules for attaching facets to an agent, when a `Sensitivity` facet is added to the `NetTCPAgent`,
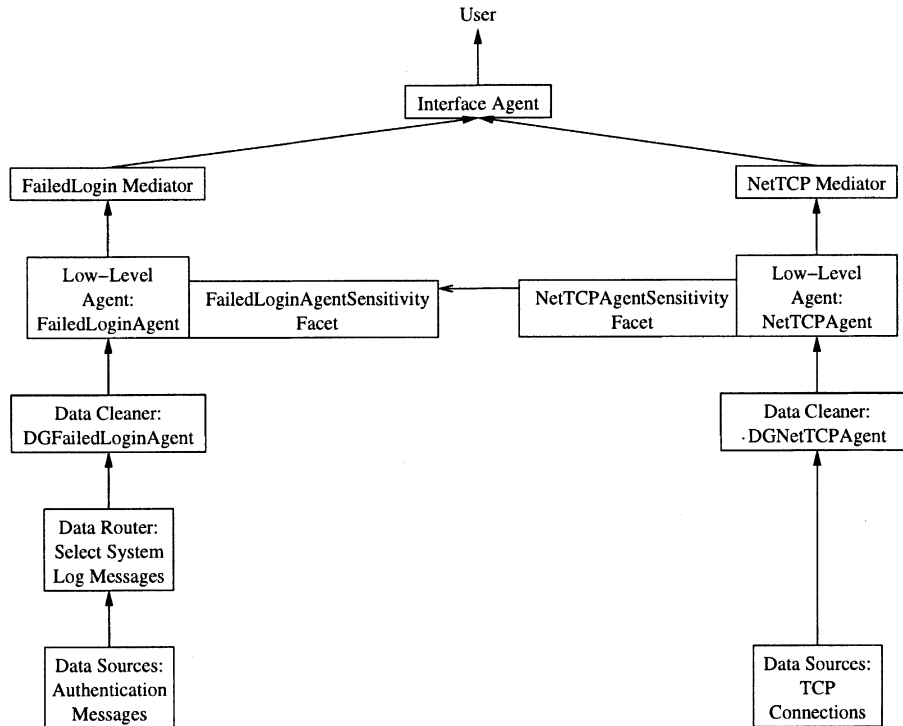
Fig. 4. A portion of the IDS with facets.

the `NetTCPAgentSensitivity` facet is added to the `NetTCPAgent`.

The `FailedLoginAgentSensitivity` facet listens for intrusive activity reports from the `NetTCP-Agent`. If the `NetTCPAgent` were to sense activity such as scanning for available telnet ports, the `FailedLoginAgentSensitivity` facet will raise is level of suspicion. An intruder may scan for available telnet ports, then connect to the discovered telnet ports and try to login to well-known accounts with typically-used passwords. If this were to happen, the `FailedLoginAgentSensitivity` facet would help detect the attack by lowering the threshold of acceptable failed logins.

4.3. Results

By only adding the sensitivity capability when it is required, the load on the network and speed of transmission of the agents is much better in the normal case.

In particular, the Java byte-compiled code for the `FailedLoginAgent` class is a total of 5665 bytes (2298 + 3367). The byte-compiled code for the `FailedLoginAgentSensitivity` class is a total of 5547 bytes (1540 + 4007). Permanently adding the sensitivity capability to the `FailedLoginAgent` would increase the size of the agent by 96%. Considering only the size of the code, the `FailedLoginAgent` is roughly half the size without the sensitivity facet as it would be with the sensitivity facet. At a constant transmission rate, the

time to transmit the lightweight agent by itself is about half the time required to transmit the agent with the facet.

The savings are less dramatic for the `NetTCPAgent`. The Java byte-compiled code for the `NetTCPAgent` class is a total of 7434 bytes (4067 + 3367). The byte-compiled code for the `NetTCPAgentSensitivity` class is a total of 4493 bytes (486 + 4007). Permanently adding the sensitivity capability to the `NetTCPAgent` would increase its size by 60%.

Our IDS system has not only good efficiency and fast response time but also high accuracy and low false alarm rate. We conducted several experiments.

First, we conducted an experiment to test the classification of sendmail system calls by a lower-level agent. We used the database of system calls from Forrest's project at the University of New Mexico to train our agent. For sendmail system calls, the average false alarm rate is only 0.83% using our method and the complexity of learned hypothesis is 8.6 rules on average. For details of the experiment, please refer to our other papers (Helmer et al., 1998, 1999).

Second, we conducted experiments to test the correlation ability of the IDS. Some distributed attacks were used to test the false alarm rate and correlation ability. One distributed attack used was the FTP bounce attack. The FTP bounce attack can be used to transfer data to a network port to which an attacker does not normally have access. One way to exploit this problem is to send data to a remote shell server that trusts the FTP host via

the FTP server. If the target trusts the FTP server, the rsh daemon will accept the data as if it were user input and execute the given command. To perform the experiment, we used one attacker machine which sent data to an FTP server, one relay machine which ran the vulnerable FTP server, and one machine running an rsh daemon that trusted the relay machine. All the FTP bounce attacks were successfully identified and no false alarms were reported in this experiment (Helmer et al., 2002, submitted for publication).

4.4. Discussion

In our project, we explored how to use lightweight distributed agent to implement an IDS. In this paper, we focus our discussion on agent architecture and the upgradability of agent.

Security problem is a big problem for agent system. Currently our system does not consider the security problem: anyone who can access the agent system can take control. In the future, we will explore this problem.

Newly discovered distributed attacks must be modeled and integrated into the IDS for distributed intrusion detection. Using our technique, a SFT analysis is performed to describe the new distributed intrusion. The SFTA is then translated to detector agents. Translation by hand is troublesome and tedious, so we have designed a system to automatically translate the SFTA to detection agents. For more details, please refer to our these papers (Helmer et al., 2001, submitted for publication).

5. Conclusions and future work

Extending lightweight agents provides a convenient mechanism for implementing a new form of communication in our IDS. By developing facets that implemented listening and reporting functions, a significant new feature was added to the IDS without negatively affecting the existing agent design or operation. Operation of the system is improved in the normal case, since the load on the system due to the size of the agents is reduced when no intrusions are present.

Extension of agents offers many possibilities for extending the IDS. A potential use for lightweight agent capabilities would be to add data mining capabilities to mediators in the IDS. A group of facets could be constructed that implement various data mining algorithms. Mediators could then dynamically add data mining algorithms for higher-level intrusion detection by adding appropriate facets.

The prototype MIB for intrusions should be formalized based on a taxonomy of intrusions. Using the ASN.l syntax for describing intrusions seems to hold promise for integration of IDSs and communication

between intrusion detection agents. Also, relationships between intrusions should be formally identified and encoded into facets. The prototype table of intrusion relationships was a first step in developing our sensitivity facets. Previous IDSs encoded these relationships as rules in an expert system (Mukherjee et al., 1994). Using agents and facets seems to offer a more dynamic, malleable way to deal with these relationships than encoding rules in an expert system.

An extension of the IDS system may be possible by using facets to implement data fusion to identify the source of an intrusion in real time. Some attacks identify the source IP address of the attacker, and at times even more information may be gathered from the source that help determine the identity of the attacker.

A possible further extension using facets would be to implement countermeasures to respond to intrusions. Facets could be designed that fuse intrusion information in real time. When the fused information meets certain criteria, the facets could direct the monitored system to take corrective or defensive action to deter or stop the intrusion. If the source of the intrusion is known (by the identification facets previously mentioned), countermeasures may be possible to prevent further attacks. Because facets are aggregated with their agents, the countermeasures facets would be operating on the system on which the actions would need to be executed. No additional communications overhead would be required to send commands to the target system to counter an attack.

References

Balasubramaniyan, J., Garcia-Fernandez, J.O., Isacoff, D., Spafford, E.H., Zamboni, D., 1998. An architecture for intrusion detection using autonomous agents. Technical Report COAST TR 98-05, Purdue University, Department of Computer Sciences.

Bass, T., 1999. Multisensor data fusion for next generation distributed intrusion detection systems. In: Proceedings, 1999 IRIS National Symposium on Sensor and Data Fusion, May.

Denning, D.E., 1987. An intrusion-detection model. Journal of IEEE Transactions and Software Engineering SE-13 (2), 222–232.

Forrest, S., Hofmeyr, S.A., Somayaji, A., 1997. Computer immunology. Journal of CACM 40 (10), 88–96.

Forrest, S., Hofmeyr, S.A., Somayaji, A., Longstaff, T.A., 1996. A sense of self for Unix processes. In: Pro-ssp96, Los Alamitos, CA, USA, 1996, pub-IEEE-CSP, pp. 120–128.

Helmer, G., Wong, J., Slagell, M., Honavar, V., Miller, L., Lutz, R., Wang, Y., 2002. Software fault tree and colored Petri net based specification, design, and implementation of agent-based intrusion detection systems. IEEE Transactions of Software Engineering, submitted.

Helmer, G., Wong, J., Slagell, M., Honavar, V., Miller, L., Lutz, R., 2001. A software fault tree approach to requirements analysis of an intrusion detection system. In: Proceedings, Symposium on Requirements Engineering for Information Security. Center for Education and Research in Information Assurance and Security, Purdue University, March.

Helmer, G., Wong, J.S.K., Honavar, V., Miller, L., 1998. Intelligent agents for intrusion detection. In: Proceedings, IEEE Information Technology Conference, Syracuse, NY, USA, September, pp. 121–124.

Helmer, G., Wong, J.S.K., Honavar, V., Miller, L., 1999. Feature selection using a genetic algorithm for intrusion detection. In: Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, FL, USA, July, pp. 1781.

Honavar, V., 1998. In: Williams, J., Sochats, K. (Eds.), Encyclopedia of Information Technology, chapter Intelligent Agents. Marcel Dekker, New York, USA.

Honavar, V., Miller, L., Wong, J.S.K., 1998. Distributed knowledge networks. In: Proceedings, IEEE Information Technology Conference, Syracuse, NY, USA, September 1998, pp. 87–90.

Kim, G.H., Spafford, E.H., 1994. Experiences with tripwire: Using integrity checkers for intrusion detection. In: USENIX Association, editor, pro-sans94, pub-USENIX:adr, April, pub-USENIX, pp. 89–101.

Kumar, S., 1995. Classification and Detection of Computer Intrusions. PhD thesis, Purdue University, West Lafayette, IN, USA, August.

Lee, W., Stolfo. S., 1998. Data mining approaches for intrusion detection. In: Pro-ss98, pub-USENIX.

Lippmann, R.P., Cunningham, R.K., Fried, D.J., Garfinkel, S.L., Gorton, A.S., Graf, I., Kendall, K.R., McClung, D.J., Weber, D.J., Webster, S.E., Wyschogrod, D., Zissman, M.A.,1998. The 1998 DARPA/AFRL off-line intrusion detection evaluation. In: Pro-raid98, Louvain-la-Neuve, Belgium.

Mann, D.E., Christey, S.M. 1999. Towards a common enumeration of vulnerabilities. In: Second Workshop on Research with Security Vulnerability Databases, Purdue University, West Lafayette, IN, USA, January.

Mell, P., McLarnon, M., 1999. Mobile agent attack resistant distributed hierarchical intrusion detection systems. In: Pro-raid99, Purdue, IN, USA, September.

Mukherjee, B., Todd Heberlein, L., Levitt, K.N., 1994. Network intrusion detection. Journal of IEEE Network, 8 (3), 26–41.

ObjectSpace, Inc., Dallas, TX. ObjectSpace Voyager Core Technology User Guide, 1999. Version 3.0.0.

Porras, P., Schnackenberg, D., Stuart, S.-C., Maureen, S., Wu, F., 1999. The common intrusion detection framework architecture. Online, http://www.gidos.org/drafts/architecture.txt.

Reilly, M., Stillman, M., 1998. Open infrastructure for scalable intrusion detection. In: Proceedings, 1998 IEEE Information Technology Conference, Syracuse, NY, USA, IEEE Press, pp. 129–133.

Roesch, M., 1999. Snort: Lightweight intrusion detection for networks. In: Pro-lisa99, Seattle, WA, USA, November, pub-USENIX.

Stolfo, S.J., Prodromidis, A.L., Tselepis, S., Lee, W., Fan, D., Chan. P.K., 1997. JAM: Java agents for meta-learning over distributed databases. In: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, Newport Beach, CA, USA, August, pp. 74–81.

Sun Microsystems. Java Development Kit Version 1.1.x. Online, May 2000. Available from: <http://java.sun.com/products/jdk/1.1/>.

University of Ottawa. Intelligent mobile agents research. Online, May 2000. Available from: <http://deneb.genie.uottawa.ca/webdata/research/>.

Warrender, C., Forrest, S., Pealmutter, B., 1999. Detecting intrusions using system calls: alternative data models. In: Pro-ssp99, IEEE Press, pp. 133–145.

Dr. Guy Helmer is a Senior Software Engineer at Palisade Systems, Inc., where he designs and builds network security and applications protocol management appliances, including the PacketHound, winner of a 2001 R&D 100 Award. He received his Ph.D. in Computer Science from Iowa State University in 2000, his M.S. in Computer Science from Iowa State University in 1998, and his B.S. in Computer Science from the South Dakota School of Mines and Technology in 1989. Guy spent seven years as a system programmer, network engineer, and system administrator at Dakota State University in Madison, S.D., one of Yahoo!'s Most Wired Universities in 1998, 1999, and 2000, where accomplishments included networking multiple campuses, establishing the first connection for the state government of South Dakota to the Internet, and engineering one of the first comprehensive high-speed dorm networks in the region. His research interests include operating system and network security, high-performance computation, software engineering, and software safety.

Dr. Johnny S.K. Wong is a Full Professor of the Computer Science Department, Iowa State University at Ames, Iowa, USA. His research interests include Operating Systems, Distributed Systems, Telecommunication Networks, Broadband-Integrated Services Digital networks, Concurrency Control and Recovery, Multimedia and Hypermedia Systems, Intelligent Multi-agents Systems, Intrusion Detection. He has been an investigator for research contracts with Telecom Australia from 1983 to 1986, studying the performance of network protocols of the ISDN. During this period, he has contributed to the study and evaluation of the communication architecture and protocols of ISDN. From 1989 to 1990, he was the Principal Investigator for a research contract with Microware Systems Corporation at Des Moines, Iowa. This involved the study of Coordinated Multimedia Communication in ISDN. In Summers 1991 and 1992, Dr. Wong was supported by IBM corporation in Rochester. While at IBM, he worked on the distributed computing environment (DCE) for the Application Systems. This involved the integration of communication protocols and distributed database concepts. Dr. Wong is also involved in the coordinated multimedia system (COMS) in Courseware Matrix Software Project, funded by NSF Synthesis Coalition Project to enhance engineering education. From 1993–1996, he is the working on a research project on a knowledge-based system for energy conservation education using multimedia communication technology, funded by the Iowa Energy Center. From 1995 to 1996, he was supported by the Ames Laboratory of the Department of Energy (DOE), working in Middleware for Multidatabases system.

He was involved in projects on Intelligent Multi-agents for Intrusion Detection and Countermeasures funded the Department of Defense (DoD), database generating and X-ray displaying on the World Wide Web Applications funded by Mayo Foundation. Currently, he is working on the CISE Educational Innovation: Integrated Security Curricular Modules and NSF SFS Program on Information Assurance, both funded by the National Science Foundation (NSF).

Dr. Vasant Honavar received a B.E. in Electronics Engg. from Bangalore University, India, an M.S. in Electrical and Computer Engg. from Drexel University, and an M.S. and a Ph.D. in Computer Science from the University of Wisconsin, Madison. He founded and directs the Artificial Intelligence Research Laboratory (www.cs.iastate.edu/~honavar/aigroup.html) in the Department of Computer Science at Iowa State University (ISU) where he is currently an associate professor. Honavar is also a member of the Lawrence E. Baker Center for Bioinformatics and Biological Statistics, the Virtual Reality Application Center, and the faculty of Bioinformatics and Computational Biology at ISU. His research and teaching interests include Artificial Intelligence, Machine Learning, Bioinformatics and Computational Biology, Grammatical Inference, Intelligent Agents and Multi-agent systems, Distributed Intelligent Information Networks, Intrusion Detection, Neural and Evolutionary Computing, Data Mining, Knowledge Discovery and Visualization, Knowledge Based Systems, and Applied Artificial Intelligence. He has published over 90 research articles in refereed journals, conferences and books, and has co-edited three books. He is a co-editor-in-chief of the Journal of Cognitive Systems Research published by Elsevier. His research has been partially funded by grants from the National Science Foundation, the National Security Agency, the Defense Advanced Research Projects Agency (DARPA), the US Department of Energy, the John Deere Foundation, the Carver Foundation, Pioneer Hi-Bred Inc., and IBM. Prof. Honavar is a member of ACM, AAAI, IEEE, and the New York Academy of Sciences.

Les Miller is a professor and chair of Computer Science at Iowa State University. His research interests include databases, data warehouses, integration of heterogeneous distributed data sources, and mobile agents. He was the editor of ISMM's International Journal of Microcomputer Applications from 1989 to 1999. He has also served as the vice president of the International Society for Computers and their Applications (ISCA).

Yanxin Wang is a PhD candidate in Computer Science Department at Iowa State University. She received her Master degree in Computer Science from Tsinghua University, China in 1996 and her Bachelor degree in Computer Science from Beijing University of Aeronautics and Astronautics, China in 1993. Her current research interests include network and network security, intrusion detection and operating systems.