Lightweight Agents For Intrusion Detection*

Guy Helmer, Johnny S. K. Wong, Vasant Honavar, Les Miller
Department of Computer Science
Iowa State University
{ghelmer,wong,honavar,lmiller}@cs.iastate.edu

November 27, 2000

Abstract

We have designed and implemented an intrusion detection system prototype based on mobile agents. Our agents travel between monitored systems in a network of distributed systems, obtain information from data cleaning agents, classify & 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 intrusion detection system and show how lightweight agent capabilities allowed us to add communication and collaboration capabilities to the mobile agents in our intrusion detection system.

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 [3] in a distributed system is a difficult problem. Intrusion detection systems 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 [7] and data warehouse techniques to intrusion detection systems. 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 & 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

*Funded in part by the Department of Defense.

its environment, make plans to achieve its goal, maximize its utility, and/or modify its behavior over time [6]. 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). Lightweight agents allowed us to quickly add features to our intrusion detection system.

Distributed mobile autonomous agents solve several critical problems in intrusion detection and 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 [8][9].

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 intrusion detection system allows for the integration of intrusion detection components from other projects. For example, the SNORT network intrusion detection system [20] is a small, fast, low-overhead sensor that watches for network packets that match signatures. The agent-based intrusion detection system 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 intrusion detection system via wrapper agents.

A centralized or centrally-governed intrusion detection system 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 [15]. Mobile agents also may be capable of evading attackers and resurrecting themselves if killed.

A distributed agent-based intrusion detection system 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 [11] lists shortcomings of intrusion detection systems. Viewed in a different way, the shortcomings provide a list of desirable features in an intrusion detection system.

Generic Architecture. The Common Intrusion Detection Framework [18] (CIDF) specifies a generic architecture for an intrusion detection system and classifies the components of an intrusion detection system. A system of distributed mobile agents implements the intrusion detection system 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. Intrusion detection systems have tended to be developed with an orientation to an organization's security policy. Differences in security policies between organizations result in a lack of portability of an intrusion detection system. In a different sense, portability of the intrusion detection system 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 intrusion detection systems. AAFID [1] and our own project, MAIDS, are two such IDS's.

Upgradability. An intrusion detection system 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 intrusion detection system would depend on the architecture of the components.

Performance Benchmarks. Exhaustive quantitative performance evaluations of current intrusion detection systems 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 [14].

Testing. Kumar says, "there is no easy way to test intrusion detection systems." [11, page 35] The MIT Lincoln Labs intrusion detection evaluation [13] was one of the first major tests of research intrusion detection systems. 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 intrusion detection system 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 intrusion detection system. 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 intrusion detection system 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;

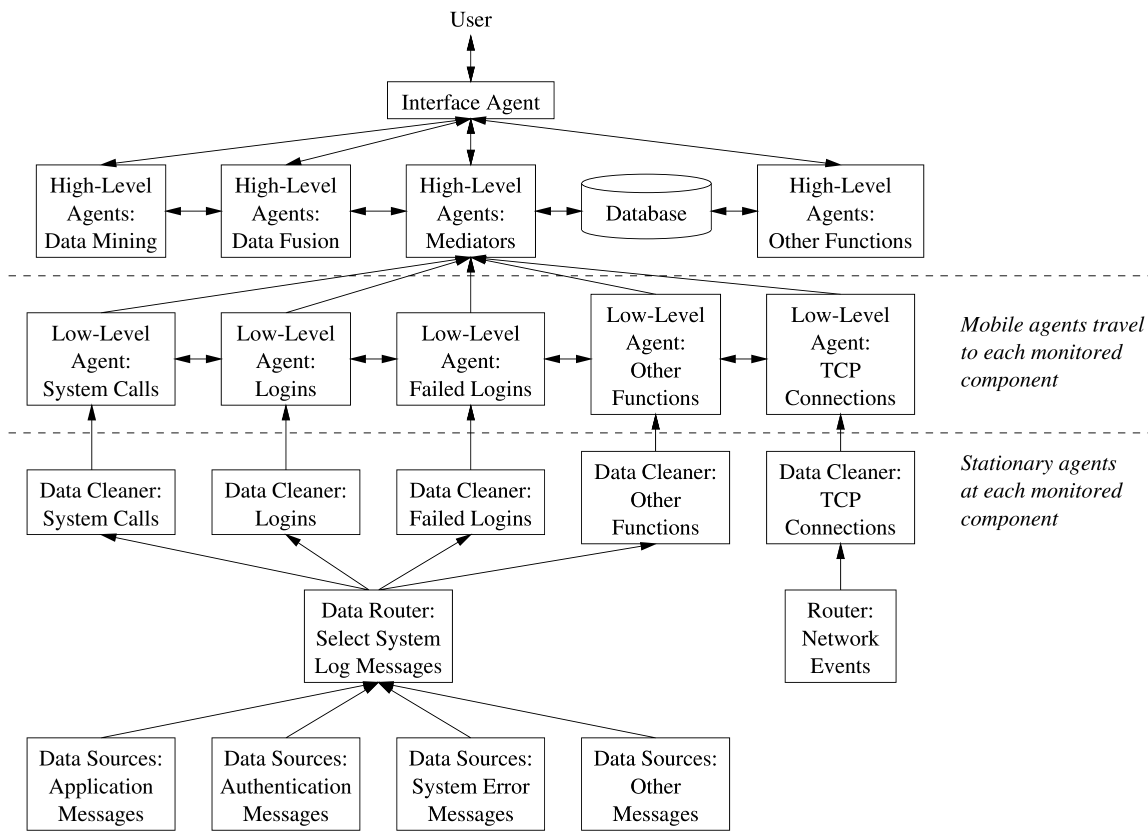

Figure 1: Architecture of the Mobile Agents Intrusion Detection System

- 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 Figure 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.

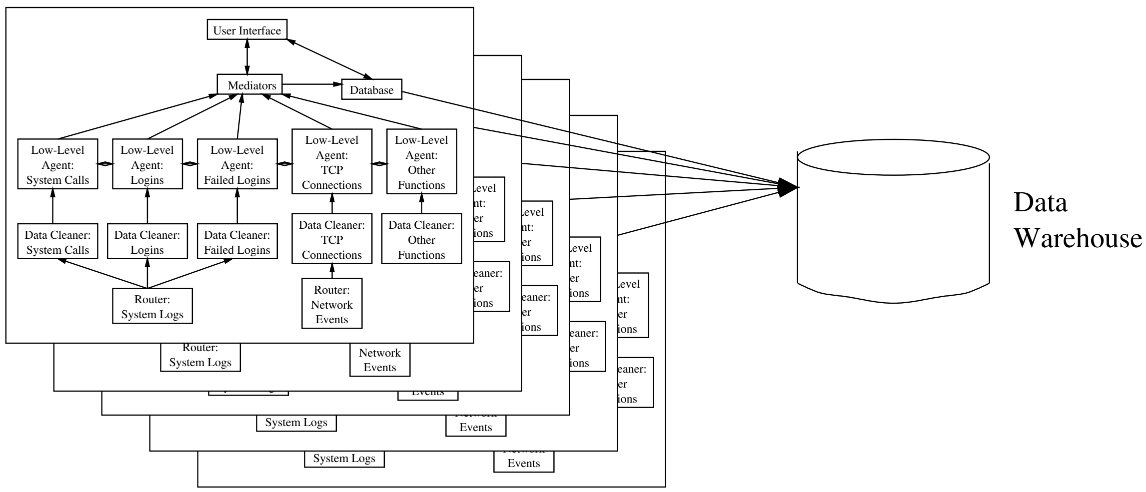

Figure 2: Enterprise data warehouse for intrusion detection

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 associated 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.

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 Figure 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 Figure 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 intrusion detection system has been built in the Java language using the Voyager Object Request Broker [17]. Voyager provides the mobility, interfacing, and naming services needed to implement the agents in the IDS.

2 Related Work

2.1 Distributed Intrusion Detection Systems

The Distributed Intrusion Detection System (DIDS) of the University of California-Davis [16] used a combination of host and LAN monitors to observe system & 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 Common Intrusion Detection Framework (CIDF) [19] 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 Autonomous Agents for Intrusion Detection (AAFID) projects each examined the problem in different ways.

The Computer Immunology Project at the University of New Mexico [23][4][5] explored designs of intrusion detection systems 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 [12][21] 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.

The AAFID project [1] at Purdue's COAST project has developed an intrusion detection system using Perl agents for fast prototyping and cross-platform compatibility. 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.

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 intrusion detection system 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 [17].

- 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.

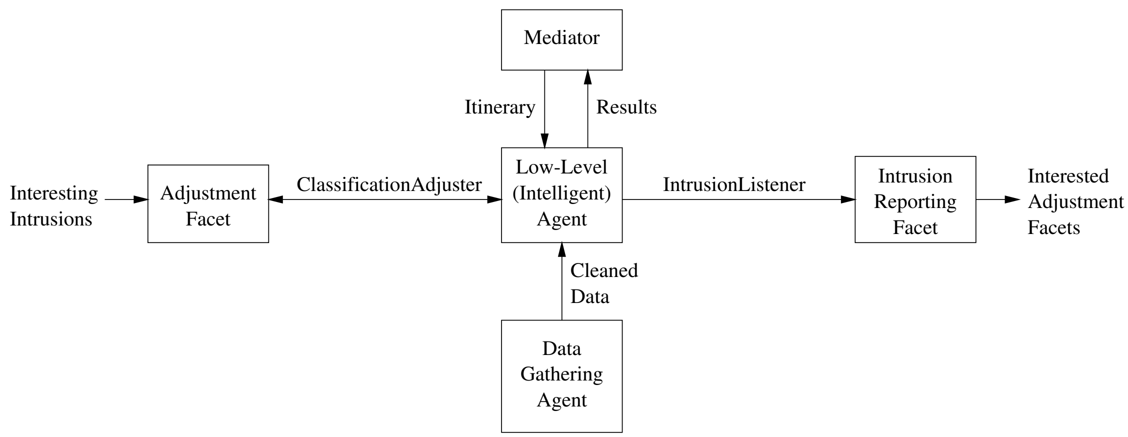

Figure 3: Facets for the low-level agents

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

Intrusion information is exchanged between facets using descriptive strings in ASN.1 syntax. In the style advocated by Tim Bass [2], 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.1 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.

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.

Table 1: Agents and their related intrusions

Agent	Related Intrusion	Rationale
Untrusted Connections; 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

4.1 IDS Implementation

The prototype IDS has been implemented using Sun's Java Development Kit (JDK) version 1.1 [22] and ObjectSpace's Voyager Object Request Broker version 3 [17].

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 FreeBSD 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 (JIT) 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 [10]), 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 Figure 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 Figure 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.

Figure 4 shows a slice of the intrusion detection system 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`, the `NetTCPAgentSensitivity` facet is added to the `NetTCPAgent`.

The `FailedLoginAgentSensitivity` facet listens for intrusive activity reports from the `NetTCPAgent`. 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

Intrusion events should be infrequent, so the intrusion detection system should rarely require the sensitivity capability. 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.

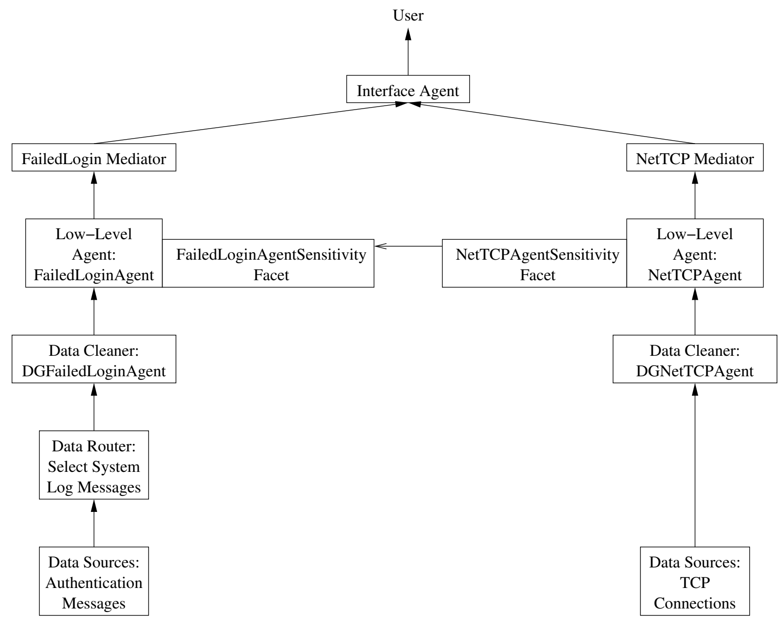

Figure 4: A portion of the intrusion detection system with facets

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%.

5 Conclusions and Future Work

Extending lightweight agents provides a convenient mechanism for implementing a new form of communication in our intrusion detection system. 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 intrusion detection system. A potential use for lightweight agent capabilities would be to add data mining capabilities to mediators in the intrusion detection system. 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.1 syntax for describing intrusions seems to hold promise for integration of intrusion detection systems 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 intrusion detection systems encoded these relationships as rules in an expert system [16]. 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

[1] Jai Balasubramaniyan, Jose Omar Garcia-Fernandez, David Isacoff, E. H. Spafford, and Diego Zamboni. An architecture for intrusion detection using autonomous agents. Technical Report COAST TR 98-05, Purdue University Department of Computer Sciences, 1998.

[2] Tim Bass. Multisensor data fusion for next generation distributed intrusion detection systems. In *Proceedings, 1999 IRIS National Symposium on Sensor and Data Fusion*, May 1999.

[3] Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, February 1987.

[4] Stephanie Forrest, Steven A. Hofmeyr, and Anil Somayaji. Computer immunology. *Communications of the ACM*, 40(10):88–96, October 1997.

[5] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128, Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[6] Vasant Honavar. *Encyclopedia of Information Technology*, chapter Intelligent Agents. Marcel Dekker, New York, USA, 1998. J. Williams and K. Sochats, eds.

[7] Vasant Honavar, Les Miller, and Johnny S. K. Wong. Distributed knowledge networks. In *Proceedings, IEEE Information Technology Conference*, pages 87–90, Syracuse, NY, USA, September 1998.

[8] Wayne Jansen, Peter Mell, Tom Karygiannis, and Don Marks. Applying mobile agents to intrusion detection and response. Technical Report Interim Report - 6416, National Institute of Standards and Technology, October 1999.

[9] Wayne Jansen, Peter Mell, Tom Karygiannis, and Don Marks. Mobile agents in intrusion detection and response. In *Proceedings of the 12th Annual Canadian Information Technology Security Symposium*, Ottawa, Canada, June 2000.

[10] Gene H. Kim and Eugene H. Spafford. Experiences with tripwire: Using integrity checkers for intrusion detection. In USENIX Association, editor, *Proceedings of the Third Annual System Administration, Networking and Security Conference (SANS III)*, pages 89–101, Berkeley, CA, USA, April 1994. USENIX.

[11] Sandeep Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue University, West Lafayette, IN, USA, August 1995.

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

[13] R. P. Lippmann, R. K. Cunningham, D. J. Fried, S. L. Garfinkel, A. S. Gorton, I. Graf, K. R. Kendall, D. J. McClung, D. J. Weber, S. E. Webster, D. Wyschogrod, and M. A. Zissman. The 1998 DARPA/AFRL off-line intrusion detection evaluation. In *Proceedings of the First International Workshop on Recent Advances in Intrusion Detection (RAID98)*, Louvain-la-Neuve, Belgium, 1998.

[14] David E. Mann and Steven M. Christey. Towards a common enumeration of vulnerabilities. In *2nd Workshop on Research with Security Vulnerability Databases*, West Lafayette, IN, USA, January 1999. Purdue University.

[15] Peter Mell and Mark McLarnon. Mobile agent attack resistant distributed hierarchical intrusion detection systems. In *Proceedings of the Second International Workshop on Recent Advances in Intrusion Detection (RAID99)*, Purdue, IN, USA, September 1999.

[16] Biswanath Mukherjee, L. Todd Heberlein, and Karl N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, May/June 1994.

[17] ObjectSpace, Inc., Dallas, TX. *ObjectSpace Voyager Core Technology User Guide*, 1999. Version 3.0.0.

[18] Phil Porras, Dan Schnackenberg, Stuart Staniford-Chen, Maureen Stillman, and Felix Wu. The common intrusion detection framework architecture. Online, 1999. `http://www.gidos.org/drafts/architecture.txt`.

[19] Mark Reilly and Maureen Stillman. Open infrastructure for scalable intrusion detection. In *Proceedings, 1998 IEEE Information Technology Conference*, pages 129–133, Syracuse, NY, USA, 1998. IEEE Press.

[20] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of the Thirteenth Systems Administration Conference (LISA 99)*, Seattle, WA, USA, November 1999. USENIX.

[21] Salvatore J. Stolfo, A. L. Prodromidis, S. Tselepis, W. Lee, D. Fan, and P. K. Chan. JAM: Java agents for meta-learning over distributed databases. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 74–81, Newport Beach, CA, USA, August 1997.

[22] Sun Microsystems. Java Development Kit Version 1.1.x. Online, May 2000. `http://java.sun.com/products/jdk/1.1/`.

[23] Christina Warrender, Stephanie Forrest, and Barak Pealmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 133–145. IEEE Press, 1999.