1998-06

A Distributed Autonomous-Agent Network-Intrusion Detection and Response System

Barrus, Joseph

Monterey, California. Naval Postgraduate School

A Distributed Autonomous-Agent Network-Intrusion Detection and Response System

Joseph Barrus
Enable Incorporated
11440 W. Bernardo Ct., Suite 290
San Diego, CA 92127
jbarrus@enableinc.com
Neil C. Rowe
Code CS/Rp
Naval Postgraduate School
Monterey, CA 93943

Abstract

We propose a distributed architecture with autonomous agents to monitor security-related activity within a network. Each agent operates cooperatively yet independently of the others, providing for efficiency, real-time response and distribution of resources. This architecture provides significant advantages in scalability, flexibility, extensibility, fault tolerance, and resistance to compromise.

We also propose a scheme of escalating levels of alertness, and a way to notify other agents on other computers in a network of attacks so they can take preemptive or reactive measures. We designed a neural network to measure and determine alert threshold values. A communication protocol is proposed to relay these alerts throughout the network. We illustrate our design with a detailed scenario.

This paper appeared in the Procedings of the 1998 Command and Control Research and Technology Symposium, Monterey CA, June-July 1998.

Introduction

The enormous growth of the Internet has not come without problems. There are more systems to attack as well as more systems from which to launch an attack. These attacks take advantage of the flaws or omissions that exist within the various operating systems and software that run on the many hosts in the network.

When an intruder attacks a system, the ideal response would be to stop his activity before he can do any damage or access sensitive information. This would require recognition of the attack as it takes place in real time. There are few automated methods to perform this recognition. Most real-time methods are applied manually by human observation by explicitly looking for the attack as a result of previous analysis or due to some triggering event of suspicious behavior, probably recognized by blind luck. Recognition of an attack is further complicated in a networked environment. A single suspect behavior on a single host in a network may not warrant any serious action. However, repeated suspect behavior across several hosts in a network may indeed suggest an attack, with a response definitely warranted. This would be very difficult for a human to recognize.

The Distributed Autonomous Agent Network Intrusion Detection and Response System, a collection of

autonomous agents running on the various hosts within a heterogeneous network, provides the foundation for a complete solution to the complexities of real-time detection and response in such an environment [Barrus, 1997]. These agents monitor intrusive activity on their individual hosts. Each agent can be configured to the operating environment in which it runs. In addition, other third party tools -- LAN monitors, expert systems, intruder recognition and accountability, and intruder tracing tools -- can be "plugged in" to augment the basic host-monitoring infrastructure. The agents communicate with each other relaying alert messages utilizing a hierarchical scheme of escalating levels of alertness.

Architecture

Figure 1 below is a simplified diagram of the proposed architecture. Autonomous agents sit atop various host machines in a network, communicating with each other via a defined protocol over a TCP/IP network.

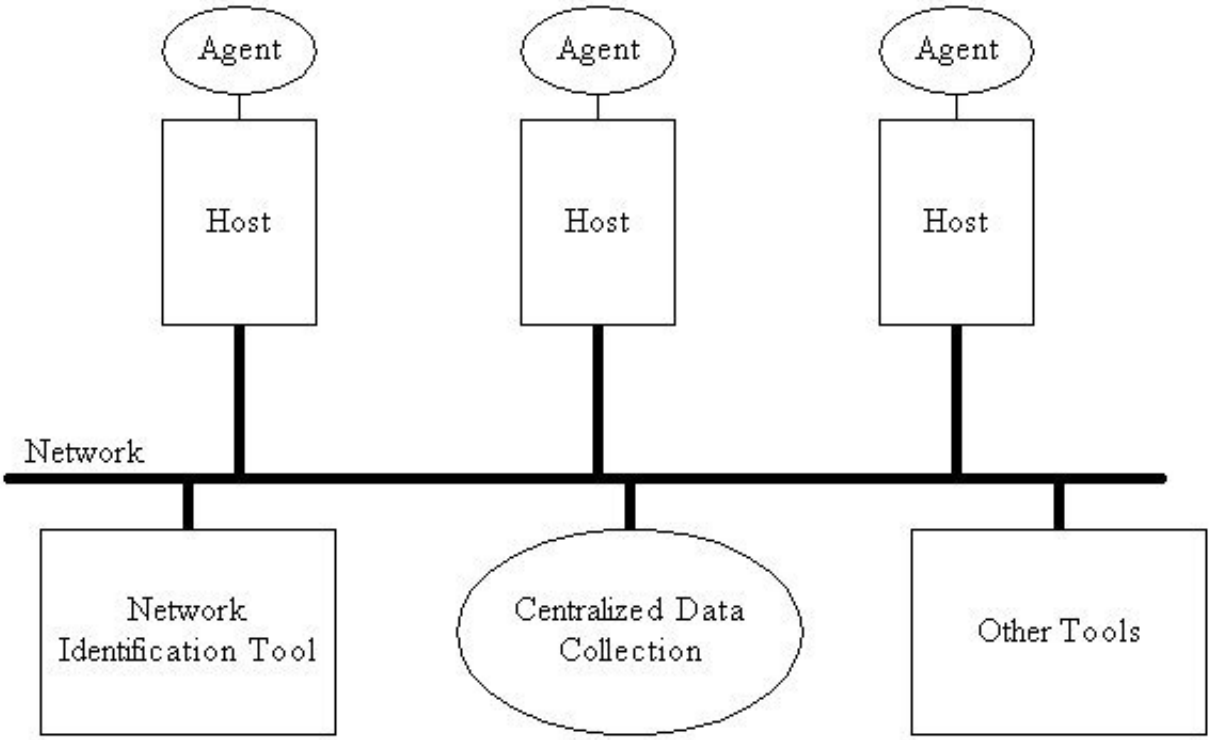

Figure 1. Distributed Autonomous Agent Network Intrusion Detection and Response System

The agents are processes that run on each host, monitoring that host for intrusive activity and communicating with each other in a coordinated response to this activity. They are written and compiled specifically for the platform upon which they sit.

The Network Identification Tool is a separate mechanism to recognize coordinated attacks distributed through the network. Alert messages generated from host monitors will be filtered through this tool. It can also make use of centralized data collection.

Attack Taxonomy

Intrusions fall into two categories, namely Misuse and Anomalous Behavior. Figure 2 below shows the attack

hierarchy. The forks below the rectangles represent inheritance.

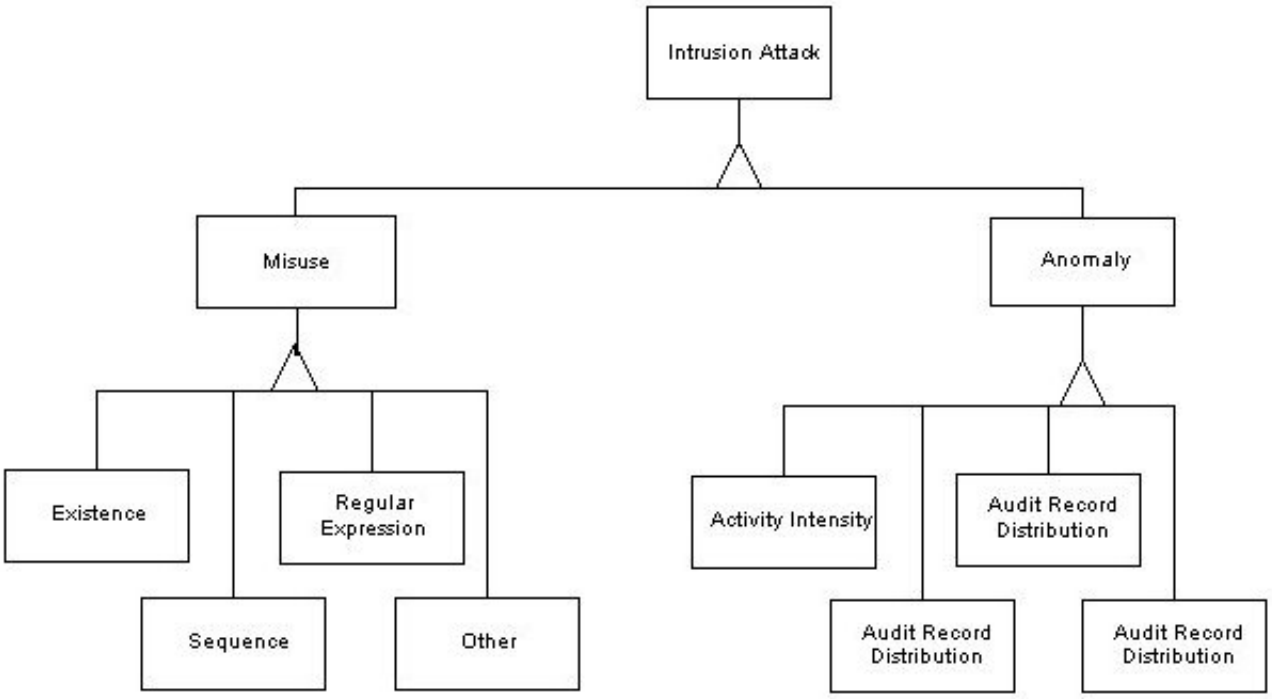

Figure 2. Intrusion Attack Class Hierarchy.

Misuse Behavior

Misuse comprises attacks that are already known and whose behavior can be specified. Therefore, recognizing an attack would involve comparing current behavior against a set of pre-defined attack scenarios. Matching of behavior occurring early in the scenario would indicate a possible or aborted attack. Misuse includes:

- Attempted break-in
- Masquerade attack
- Penetration of the security control system
- Leakage
- Denial of Service
- Malicious use

In the scheme of [Kumar, 1995], intrusions can be represented by an event or series of events. It is the relationship of these events to one another that provides the basis to recognize differing attack types. He offers four higher level classifications of attack types under which the above types can be placed:

- Existence of changes in system state
- Sequential patterns of events
- Regular expression patterns of events
- Patterns that require embedded negation and generalized selection

Anomalous Behavior

.

Anomalous Behavior describes attacks involving unusual use of the system resources. This kind of behavior is recognized by statistical methods that capture information about a user on a system and develops a profile of that user's behavior. When behavior is observed that significantly deviates from the normal behavior, then the system can respond. Threshold values must be established for response. Kumar suggests the following categories:

- Activity intensity rate
- Audit record distribution of particular activities
- Categorical distribution of activities
- Ordinal measurable activities

Alert level hierarchy

We propose two main factors should affect alert level: danger and transferability. Both will vary considerably over time.

Danger and Transferability

The danger can be defined as the potential damage that can occur if the attack at a particular stage is allowed to continue, and is measured by comparing the kind and stage of an attack against the frequency of false positives. Danger values (with nominal values in parentheses) are:

- (1) Minimal: The event cannot be deterministically associated with a known attack, but could be an early stage of some known attack profile or could be the early stage of some anomalous behavior.
- (2) Cautionary: The event indicates a secondary or later stage of a known attack, but one that still cannot be determined to be part of a known attack. It may also be the result of an accumulation of anomalous behavior having increased from the minimal stage, but not enough for serious concern or the event does determine a known attack, but the potential damage of the attack is minimal.
- (3) Noticeable: The event definitely identifies a known attack with moderate damage potential or the accumulation of anomalous behavior has reached a level where it is likely that some hostile action is taking place.
- (4) Serious: The event definitely identifies a known attack with high damage potential or the accumulation of anomalous behavior has reached a point that it is either disrupting service or is highly indicative that an attack is occurring with serious consequences.
- (5) Catastrophic: The event identifies a known attack that, if left to continue, may result in catastrophic losses.

The transferability can be defined as the applicability of an attack in other nodes in a network, and assessed by comparing the kind of attack against the existence of similar operating environments in the network. Transferability values (with nominal values in parentheses) are:

- (1) None: The attack or potential attack is occurring in an environment unique in the network.
- (2) Partial: The attack is occurring in a common operating environment or common piece of software, but the version of which may be different enough that applicability may be questionable
- (3) Full: The attack is occurring in a operating environment or piece of software that is common across the network. Other systems in the network are definitely vulnerable to the same attack.

. We propose that the product of danger and transferability should determine the Alert Level of an attack in a network. That is:

*Alert Level := Danger * Transferability*

With the values given above, the Alert Level will range from 1 to 15. The general style of alert response to a particular system state can be tied to this value, using two thresholds:

- Normal state: No notification to other nodes is necessary.
- Partial alert: Partial notification to other nodes is performed. Notification is sent to only the nodes that are determined to be relevant or where the relevancy is non-deterministic. This may result in notification of all nodes, particularly in the case where the danger and transferability values are both three. Full notification may also result if the restricted set of nodes cannot be determined from the node registry.
- Full alert: Full notification to other nodes is performed. The only time values can fall into this range are when they apply to all nodes in the network and the danger level is at least serious.

The two higher alert levels should trigger responses. For example: notification to other machines in the network of an attack; notification of an administrator; starting the logging of certain activities; or taking action to halt the attack.

Setting Thresholds with a Neural Network

Setting of the thresholds for warning actions as a function of alert level is tricky. Hosts should not be "crying wolf" or alerting unnecessarily for trivial circumstances, but on the other hand, should not overlook real possibilities for serious attacks. We designed a trainable neural network to implement this thresholding. The network requires eight inputs from statistics over a given time period:

- Number of break-in attempts.
- Number of masqueradings.
- Number of successful break-ins.
- Number of cases of leakage of secrets.
- Number of cases of impaired system availability due to malicious users.
- Number of overtly malicious actions witnessed.
- Number of Trojan horses found.
- A "nonuniqueness" factor rating how similar this site is to others.

Training uses specific cases tagged by a security expert with one of the three alertness levels. For instance, an expert told us that 1 penetration, 1 incident of maliciousness, and 2 break-ins, on a site similar to others, should result in a full alert; 1 break-in, 1 availability problem, and 2 incidents of maliciousness on a site only somewhat similar to others should result in a partial alert. Training was fast and accurate on our test cases. For instance, when an expert specified 42 alert-level judgments involving a total of 243 wide-ranging security-violating incidents, our network after training was able to provide 0.97 recall with 0.84 precision. Training took only three seconds of CPU time. Thus it appears the network is a quick and accurate tool for consolidating a range of competing factors.

Network Identification

To recognize coordinated attacks across a network, we need separate tools. One must be a network identification tool that can resolve users down to a common network identifier. Ko *et al*. have devised an algorithm for distributed recognition and accountability that could prove useful for performing this association [Ko *et al*., 1993]. Such an algorithm could be run on a separate host to perform analysis to recognize coordinated attacks and send out alerts to affected hosts as necessary. Also, the various hosts that generate their own alert messages could filter these through this tool to see if a particular attack can be associated with any suspected ongoing coordinated attacks.

Communications

Agents must communicate alert information to one another via messages. New messages may signal a change in Alert Level or details of ongoing intrusive activity. The following information needs to be included:

- Generating Host: The host machine generating the message and upon which the suspected attack is occurring.
- Alert Level: The alert level value as determined by the product of the Danger and Transferability.
- Attack Type: The attack object class.
- Network ID: The network identifier for the source of the attack if that can be determined
- Security Code: A digital signature that proves this message is coming from a host within the system and is not being spoofed.

Broadcasting of messages would be the easiest. But a TCP/IP network would require a point to point protocol. Therefore, a host name file needs to be maintained by each host in the network.

One problem associated with communications between the hosts is availability. If a host is unavailable to receive or send messages, it degrades the effectiveness of the intrusion detection system. ongoing attack. The importance of sending and receiving message should increase with the load on a host so messages will transfer properly.

However, a host could become so overloaded that it cannot send and receive messages, and that fact should raise the Alert Levels in other hosts in the network so they can protect themselves against a similar denial of service attack. This can be done by having all computers periodically send "I'm still running" messages to one another, so that the absence of one of these messages would act as a positive message indicating a denial of service attack, and result in raising the Alert Levels of other computers for this kind of attack.

In addition, messages are needed when changing configuration settings, for product upgrades, for adding additional hosts to the host file, or in requesting additional information from hosts.

Agent design

Figure 3 shows the object model of the agent design using the OMT methodology notation [Rumbaugh, 1991]. The forks below the rectangles represent inheritance.

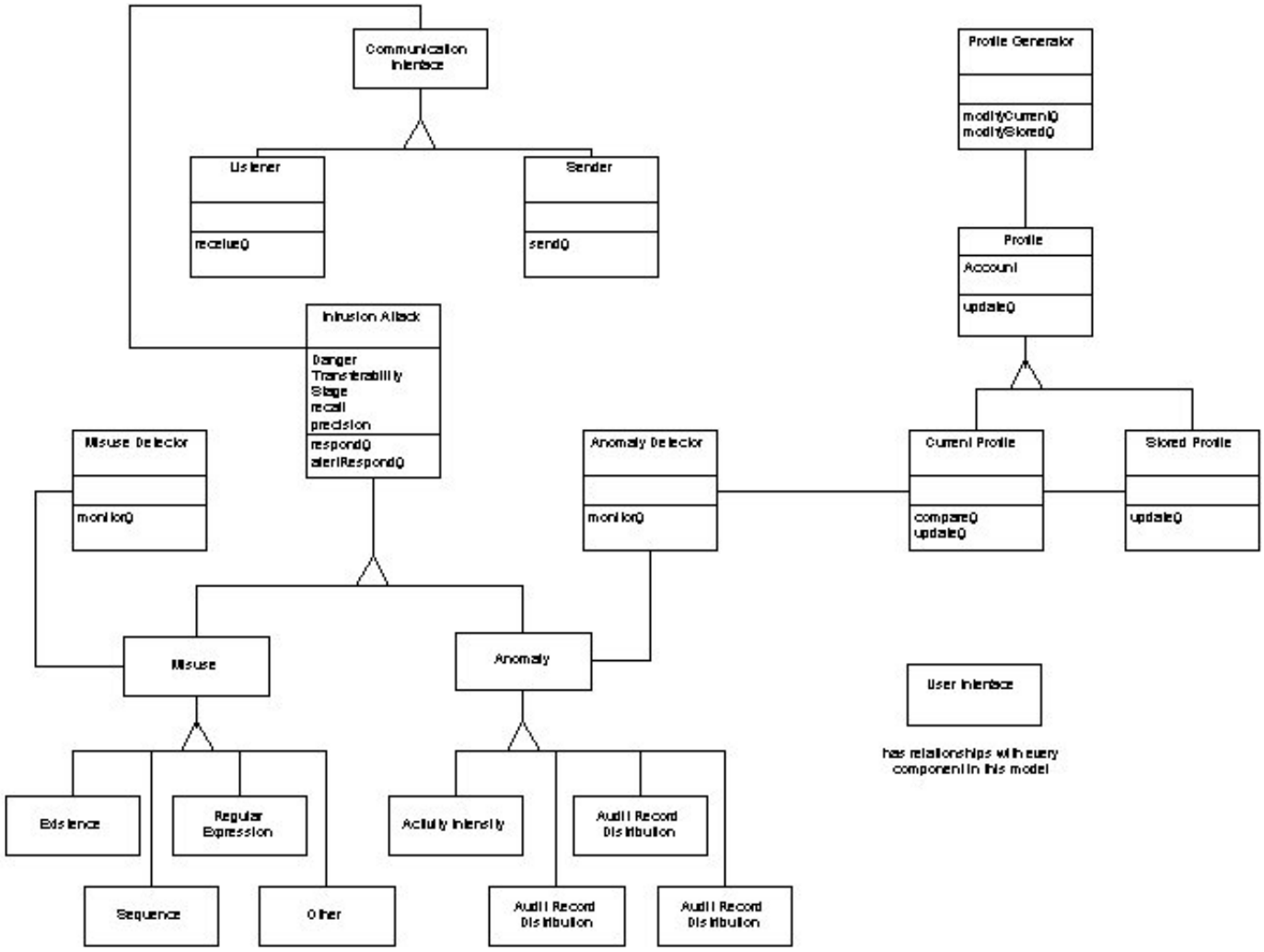

Figure 3. Object Model of Autonomous Agent Design

The elements in Figure 3 are:

- Communication Interface: This is an abstract superclass for the two types of communications.
- Listener: This listens for and receives incoming messages from other agents. It then processes them according to the type of attack.
- Sender: This sends messages (alerts) to other agents. It packages the alert, identifies the locations of other agents, and calls the network software to send the message.
- Intrusion Attack: Common attributes and methods for all intrusion types are placed here. Stage is an internal attribute used to track the phases of an attack. Changes in this value trigger changes to the danger value, which change the alert level for that attack and trigger additional responses. Recall and precision are static values set by administrators using the neural network. The method respond() responds to an internal change in alert level and alertRespond() responds to receipt of alert level messages for attacks of that type.
- Misuse: This covers misuse attacks.
- Anomaly: This covers anomaly attacks.
- Attempted Break in, Activity Intensity, etc. These handle various attack types, overriding any general

implementations at the superclass level. The number and identity of these classes is dynamic and new ones added as needed. Objects at this level will communicate with the Listener portion of the Communications Interface to pass information for alerts.

- Misuse Detector: This monitors the system resources and audit files to find and identify possible attacks. If it finds one, it will instantiate an object of the appropriate class and call the appropriate method to handle it.
- Anomaly Detector: This monitors the system resources for anomalous behavior. It calls the Profile of various accounts to compare itself to the stored Profile. If the current profile differs significantly, it will instantiate an anomaly object of the appropriate class and call the appropriate method to handle it.
- Profile Generator: This continuously tracks and measures the system resources and audit files.
- Current Profile: This represents the profile for some determined period up through the current time.
- Stored Profile: This is the historical, statistical record of an account's behavior against which the current profile can be compared to determine whether the current behavior is anomalous. This profile must be updated by merging in the current profile data periodically.

Each agent is responsible for monitoring a single host in the network. Once an attack has been identified, an occurrence of it is instantiated from the attack class hierarchy, and is get up to date by that agent through continued monitoring and through coordination with agents on other hosts by alert-level messages, handled by alertRespond().

Example Intrusion Scenario

To show how this design works we give an example. This will illustrate the identification and response to a coordinated *Doorknob Rattling* attack in a network using a simultaneous sweep from multiple sources.

Figure 4 shows the layout of a small sample network. Host 1 and Host 2 are UNIX platforms running an autonomous agent, while Host 3 is a Windows NT platform running an autonomous agent. Host 4 is a UNIX platform running a network monitor. Each host communicates with the others by the sockets provided by the operating system using the TCP/IP protocol. The listener is a daemon process that retrieves messages coming into a TCP/IP port on a host, and passes them to the agent software.

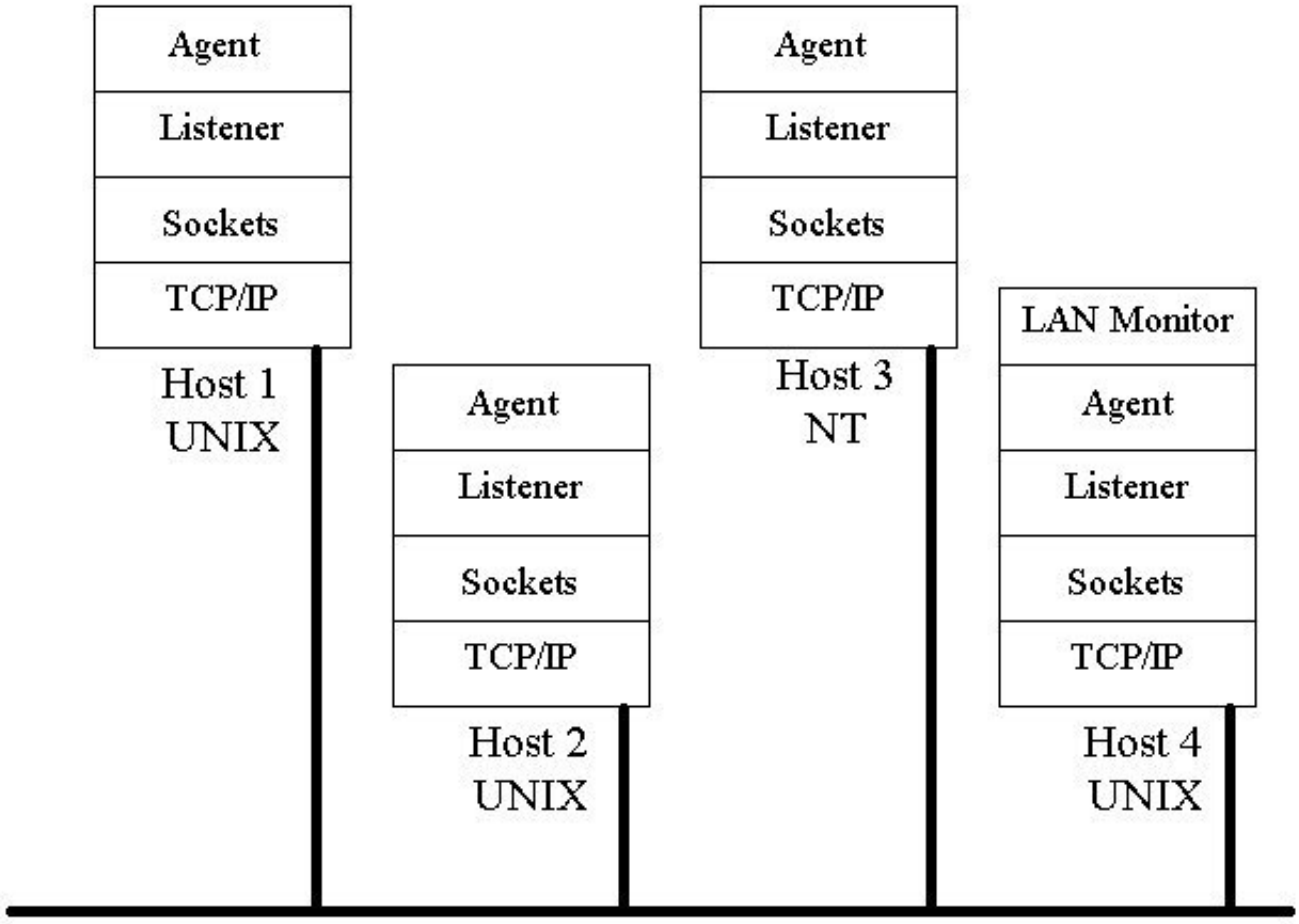

Figure 4. Example Network Subjected to Doorknob Attack

Each host:

- Responds to notifications of this attack type with alert level values of 6-10 by monitoring logins more closely (running a special process to do this) and accumulating evidence.
- Responds to notifications of this attack type with alert level values of 11-15 by paging the system administrator.
- Responds to continued attempts on that host, at least six, while alert level is in a state of 6-10, by raising the danger level to three.
- Responds to a successful illegal login on that host by raising the danger level to a value of four.

The host with the monitor is configured to respond to a doorknob attack by setting the danger level to two. Each host has a host file with information about the other hosts including:

- Host_1: OS: UNIX Type: Agent
- Host_2: OS: UNIX Type: Agent
- Host_3: OS: Windows NT Type: Agent
- Host_4: OS: UNIX Type: Monitor

An attack scenario might proceed with the LAN monitor on Host 4 recognizing that attempts to login to

various hosts originate from the same source. An instance of a Doorknob Attack subclass in the Intrusion Attack hierarchy is created by the agent's Misuse Detector interfacing with the LAN Monitor on the same host.

It then determines that the danger level of the newly created Doorknob Attack object has a value of two, since it is not yet sure this is an attack. Then the Misuse Detector determines which machines would be affected. In this case, it would be all of the machines, as both UNIX and Windows NT operating systems have analogous login procedures. The Misuse Detector then sets the transferability value to three since all machines in the network are affected, for an Alert Level of six. This implies partial notification. The Misuse Detector calls the respond() method of the Doorknob Attack object and this causes notification to occur by sending messages to all the hosts in the host file whose operating systems are UNIX or Windows NT.

Each host responds to that Alert Level based on how the response was configured for it. In this example, all the hosts are configured identically. So the message is received via the socket mechanism in a network port of each host machine. This socket forks a process to handle receipt of that message. This process passes the information in the message to the Listener object in the agent via its receive() method. The Listener object then calls the alertRespond() static method of the Doorknob Attack class to respond by monitoring the logins more closely to determine if logins continue to come from the same source and to start accumulating evidence of this attack.

Host 1 sees six more attempted logins on its machine and raises the danger level to three. But just logging in causes no damage if stopped there, and the transferability level remains the same. The Alert Level now is raised to nine. Messages are sent out from Host 1 to 2 and 3. However, suppose Host 2 sees a successful login from this source, and raises the danger level to four to indicate "very serious." With the Alert Level now at twelve, a full alert is appropriate, so all hosts will receive a notification. When Host 1 and Host 3 get a notification with an Alert Level of 12, they could page their respective administrators to take corrective action. Or they might start filtering out all packets coming from this source, or start tracing the source of the attack

Requirements Satisfaction

Our system satisfies many requirements for a good intrusion-detection system.

Recognition

The system must recognize initial potentially suspect behavior (the triggering event): The Misuse Detector and the Anomaly Detector will recognize suspicious behavior. In addition, central data collection allows for the recognition of distributed attacks.

Escalating Behavior

The system must recognize escalating suspicious behavior on a single host or across hosts: The Misuse Detector and the Anomaly Detector are both continuously seek and recognize escalating behavior. The attack object classes also permit instantiation of continuing attack threads.

Response

The system components must take the appropriate action that corresponds with a level of alertness: The attack class hierarchy contains a response method that is defined for each attack referring to the level of

alertness, attack type and stage of attack.

Manual Control

The system must allow manual intervention: The User Interface allows for direct manipulation of each object in the model. Furthermore, each agent can act as the master to the system at any given time. This would allow the system administrator access at any given node in the network. The system administrator can use any machine to intervene and propagate the results to all the other relevant hosts.

Adaptability

The system must respond quickly to the ever changing methods of attack: The existence of an attack hierarchy allows for the addition of new attack types. Since each response is kept in a separate attack class, modifications can be applied independently on other parts of the system.

Concurrency

The system must be able to handle concurrent attacks: The architecture provides for this as each agent can deal with concurrent attacks on its own machine. The attack hierarchy allows for further concurrency by providing for the concurrent instantiation of attack objects. Whenever a unique attack is recognized, its existence is recorded in an instantiation of an attack class of that attack type. The Misuse and Anomaly Detectors can keep track of and communicate with each instantiated attack object. Each instantiation can be realized on the system as a separate process.

Scalability

The system must scale as the network grows: Since the agents run independently of each other, they can be added without adversely affecting the performance of the other agents. Unlike other systems that monitor network traffic at a single point in the network where the increase in data might cause serious performance problems, independent agents only deal with the data for one system. As new hosts are added to the network, the number of messages received by any one agent should increase linearly.

But will the processing of these messages increase linearly? Each agent processes each message on a first-come, first-served basis. There is no need to sort messages or process messages together. The time to handle a message should be fairly short, so the total number of forked processes at any given time should peak at a number based on the number forks that can be done within the average process time of a message. This should fall well within the capabilities of each machine. If a host is unable to receive a message, the sender can retry for some set number of time.

Resistance to Compromise of Intrusion Software Itself

The system must protect itself from unauthorized use or attack: An encrypted signature to each message provides protection from spoofed messages. And any attempt to access the system itself can be thwarted by the addition of an attack class of this type of attack. Furthermore, the heterogeneous nature of the agents makes it more difficult to define a single attack that could affect all the agents in the system.

Efficiency and Reliability

The system should be efficient, reliable, and resistant to degradation in a heterogeneous platform environment:. Only the most appropriate code need be executed and only on the machine where the attack is

taking place. If any node agent goes down, the system still runs in full; only the machine upon which the agent was running becomes vulnerable. The distribution of the load supports resistance to degradation. And each agent is implemented specifically for the platform upon which it runs, but can communicate with each other regardless of platform.

Extensibility

The system must be extensible: As new attack types are defined, they can be added to the various agent implementations and only to those agents to which this type of attack is possible.

Flexibility

The system must be flexible: Each host can perform any subset of the functionality of the system. Each administrator can train the neural as per their own criteria. They can choose which attack types to monitor, or write their own responses

Conclusions

Intrusion detection and management is a relatively new idea. Robust yet efficient solutions are few. Our system offers a solution addressing the major issues: scalability, flexibility, extensibility, fault tolerance, and efficiency. We used a distributed architecture composed of autonomous agents and an object-oriented design to permit real-time response, to mitigate ongoing damage and provide the possibility of preemptive responses. Alert-level notification messages permit cooperative responses. Our approach should be an improvement on the preponderantly single-host systems of today.

References

[Barrus, 1997] Joseph D. Barrus. *Intrusion Detection in Real Time in a Multi-Node, Multi-Host Environment*. M.S. thesis, Naval Postgraduate School, Monterey, CA September 1997.

[Kumar, 1995] Sandeep Kumar. *Classification and Detection of Computer Intrusions*. Department of Computer Sciences, Purdue University. Ph.D. Dissertation, 1995.

[Ko *et al*., 1993] Calvin Ko, Deborah A. Frincke, Terrence Goan Jr, L. Todd Heberlein, Karl Levitt, Biswanath Mukherjee and Christopher Wee. Analysis of An Algorithm for Distributed Recognition and Accountability. 1st ACM Conference on Computer and Communications Security, Dept. of Computer Science, University of California, Davis, November 1993.

[Rumbaugh, 1991] James Rumbaugh, *Object-Oriented Modeling and Design*. Prentice-Hall International, New Jersey, 1991.

[Crosbie and Spafford, 1995] Mark Crosbie and Gene Spafford. Active Defense of a Computer System using Autonomous Agents. Technical Report No. 95-008, COAST Group, Dept. of Computer Sciences, Purdue University, February 15, 1995.

Go up to paper index