Distributed Intrusion Detection for Computer Systems Using Communicating Agents

Major Dennis J. Ingram(1), H Steven Kremer(2), and Neil C. Rowe(3)

(1) Marine Corps Warfighting Laboratory,
3255 Meyers Ave.
Quantico, VA 22134 USA
(703) 784-1330, ingramd@mcwl.quantico.usmc.mil

(2) Space and Naval Warfare Systems Center
53560 Hull Street
San Diego, CA 92152-5001
(619) 553-8542, kremer@spawar.navy.mil

(3) Code CS/Rp, Department of Computer Science
U.S. Naval Postgraduate School
Monterey, CA 93943 USA
(831) 656-2462, rowe@cs.nps.navy.mil

Abstract

Intrusion detection for computer systems is a key problem of today's Internet connected society. Almost all government agencies, businesses, schools, and most homes use the Internet, and many of their service providers use the Windows NT operating system, which has known vulnerabilities. The availability of many "hacker toolkits" provides even the novice computer enthusiast with the ability to penetrate an Internet site. Most commercially available intrusion detection systems provide for a distributed mechanism, but require that information be transmitted to a central location for detection processing. This centralized mechanism is a single point of failure and provides a prime target for an intruder. The work presented here demonstrates that detection agents can be run in a distributed fashion with each one running independently of the others while cooperating and communicating to provide a truly distributed detection mechanism without a single point of failure. There is no central processing location; all agents process the information available to them independently and relay any suspicious activity to other agents on the network. The agent software runs simultaneously with other user and system software without noticeable consumption of system resources, and without generating an overwhelming amount of network traffic during an attack.

1. Background

Computer security in today's networks is one of the fastest expanding areas of the computer industry because protecting resources from intruders is an arduous task that must be automated to be efficient and responsive [Hale, 1998; GAO, 1996]. Most intrusion-detection systems currently rely on some type of centralized processing to analyze the data necessary to detect an

intruder in real time [Lunt, 1993]. A centralized approach can be vulnerable to attack. If an intruder can disable or bypass the central detection system, then most, if not all, protection is subverted.

Intrusions generally fall into two categories: misuse and anomalies. Misuse attacks exploit some vulnerability in the system hardware or software to gain unauthorized access. Many of these attacks are well documented and are easily detected by computer systems, but new ones are constantly being discovered. Anomalies are harder to detect since they often originate from an inside user who already has access to the system. They are characterized by deviations from normal user behavior, and detection requires some type of user profiling to establish a normal behavior pattern and then look for deviations from the pattern.

Several types of detection systems are commercially available. These can be used individually or can be combined to provide more protection. A host-based system resides on a single host computer. It uses audit logs or network traffic analysis of a single host for processing and detection. This type of system is limited in scope since it is only able to use its own local host's environment, and cannot detect simultaneous attacks against multiple hosts. A network-based system is a dedicated computer, or special hardware platform, with detection software installed. It is placed at a strategic point on a network (like a gateway or subnetwork) to analyze all network traffic on that particular segment. It can scan data traffic for known attack patterns. It can also determine if incoming Internet Protocol (IP) addresses originate from outside its subnet. This system can detect attacks against multiple hosts on a single subnet, but it usually cannot monitor multiple subnets at one time. It also cannot detect any host-based attack that does not pass through it.

Distributed network-based systems allow detection software modules to be placed throughout the network with a central controller collecting and analyzing the data from all the modules. This provides a robust mechanism for detecting intrusions across several subnets and several hosts. But it usually requires a dedicated computer to act as the central controller; centralization can make it vulnerable to attack. Examples are AID [Sobirey and Richter, 1999], a client-server architecture that consists of agents residing on network hosts and a central monitoring station; AAFID [Zamboni et al, 1998], a hierarchy of components that detect basic operations and report to a transceiver, which performs some basic analysis on the data and sends commands to the agents; CMDS™, a commercial product from Science Applications International Corporation (SAIC) [Proctor, 1996] that is a real-time audit reduction and alerting system that uses an expert system and statistical profiling to analyze audit records; and EMERALD [Neumann, 1999] which monitors large distributed networks with analysis and response units called monitors.

2. Design of a distributed intrusion-detection system

We describe here the design and implementation of a distributed network-based intrusion-detection system using a set of agents ("IDAgent"s), one for each processor, and a communications mechanism for them, built on top a simple single-computer intrusion-detection sensor. We have concentrated on agent communication and coordination with only a relatively simple set of detection criteria. Our goal was to explore the distributed processing aspects of

intrusion problems. Further details are in [Ingram, 1999] and [Kremer, 1999], available online from http://www.cs.nps.navy.mil/people/faculty/rowe/index.html.

2.1 Software choices

The IDAgent base design is implemented in Java® version 1.1.8 to be platform independent. Initial tests of the communications mechanisms were done on Windows NT 4.0 workstations and Windows NT 4.0 server and Linux version 5.2. Several trial runs were also conducted in a mixed environment with NT 4.0 workstation, NT Server, and Linux 5.2, running together. Platform independence is difficult for intrusion detection because of the many platform-specific mechanisms used. For example, a Windows NT system log is processed differently from one on a Linux or Sun® workstation. For this reason, we chose a single platform for final testing, and since the military is migrating primarily to a Windows NT environment, it was chosen.

Figure 1 shows a block diagram of the IDAgent components needed for a single host. All major components of the agent are constructed as threads that run concurrently. The main components and data structures are: Controller module, TCP Receiver, UDP Receiver, TCP Transmitter, UDP Transmitter, Agent Window Manager, Host Sensor, Log Sensor, Message class, and Contact List of known agents. The Agent Window Manager is the user interface to the IDAgent. It contains a display frame of 500 by 320 pixels for a text display area. The lower portion contains an area with control buttons on a colored background of green, yellow, or red, depending on the current alert level of the IDAgent. The control buttons allow displaying of debugging data, the current contact list of known agents, alert messages that caused a change to the alert status, and a status which indicates a numerical value of the current alert level.

2.2 Controller module

The controller is the brain behind the IDAgent. Once the main program initializes all variables and the Window manager is started, the Controller thread is started; it suspends processing when there is no activity to reduce system processor usage. Any activity in any sensor, receiver, transmitter, or Window Manager of the IDAgent will activate the controller so that it may analyze incoming messages from other agents and internal sensors and determine if an intrusion is in progress. The controller updates the Alert status of the agent depending on the messages it receives. The Alert status can range from 0.0 to 1.0 and represents the likelihood that an intrusion is taking place. Alert levels of 0.0 to 0.4 will display a green indicator in the user display indicating a normal status; 0.4 to 0.7 will show a yellow indicator; and above 0.7 will display red.

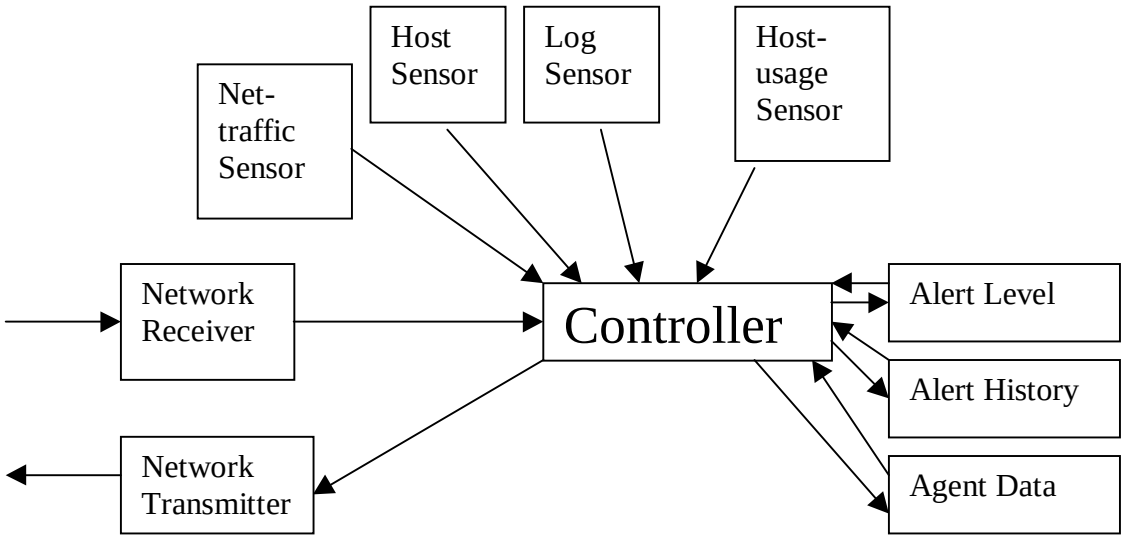

Figure 1: IDAgent Block Diagram

The Alert level is increased depending on the "weight" of the message. All messages are initially sent with a weight value of 0.0; this prevents the message from affecting the alert level until the controller has analyzed the message. Once analyzed the message will either be saved for future reference or the weight and alert level will increase. Each message is analyzed when it arrives. If an attack is suspected, then an appropriate weight value is assigned to that message, which in turn increases the alert level of the IDAgent. If the message is not considered an attack, then the message weight remains 0.0.

The normalized increase in the alert level is inverse-exponential as the alert value increases. N = ((1.0 – A) * W) + A), where A = the current alert level, W = the weight value of a message, and N = the new alert value. The alert level will approach 1.0 if many suspicious alerts are received. If no alerts are received within two transmit intervals (10 minutes in the current implementation), the alert level will decrease following a negative exponential curve. N = (A * Degradation Factor) where degradation factor is a fraction (0.9 in the current implementation).

If important information is received from an internal sensor, its agent's controller constructs a message and sends it to other agents in the network to notify them of an event or action that is taking place on its own host. Messages are not forwarded in the network to prevent duplicate message traffic. For an example, assume in a network of twenty computers that the agent on computer nine detects a failed login attempt. Its controller analyzes the attempt and constructs an alert message with 0.0 weight that is sent to all nineteen other computers. Now should the person attempt a login on another computer, it too would be detected and sent to all agents.

The current implementation includes only basic intrusion capabilities for testing obtained from the Log Sensor (see below), an interface between the audit and the IDAgent that recognizes login attempts. This data is passed to the agent where it is processed by the controller. The controller analyzes each failed login attempt and calculates a fraction of failed attempts as compared to the total attempts. This calculation is done both for attempts on a single host and the network as a

whole. If either the host or network fraction reaches the threshold value for the agent, an alert message is constructed with a weight value of (fraction – threshold). To overcome the problem that occurs with small login ratios (1 login attempt and 1 failure is 100% failure rate), we use the formula ((LoginFailures / TotalAttempts) – (1 / TotalAttempts)). Another internal sensor included in the agent is the host sensor, which uses the contact list of known agents to determine if an agent has stopped responding. The host sensor monitors how many remote agents have contacted it and checks to make sure they are all still functioning. If the number of agents not responding reaches a threshold, a message is sent to the controller.

2.3 Transmitters and receivers

The User Datagram Protocol (UDP) Transmitter thread generates a UDP packet that contains the port number that the agent's Transmission Control Protocol (TCP) receiver is listening on and identification of the local host on which it is running. It sends this packet as a broadcast message to the network on a port number determined at startup. This broadcast occurs periodically based on the Transmit Interval variable. The thread initiates the first contact and maintains the contact between all agents.

The UDP Receiver's primary function is to receive the UDP broadcast messages of other agents and process them. The broadcast port number by default is 8000. Although any unused port number may be designated at system startup, all agents in the network must be running on the same UDP port number. The UDP Receiver establishes a listener on the given port and waits for a broadcast message from another IDAgent. If a message is not in the correct format, an exception is generated and the broadcast is discarded. Otherwise, a contact record is created with the remote agent's identifying information, the port number of its TCP receiver, and the time that the message was received. The contact record is placed in a list of known contacts that is used by the controller and transmitter when sending messages. Each time a broadcast is received, the new contact record is compared to the contact list. If a match is found, the timestamp and the TCP receiver port number of the original record are updated. The port number is updated when a restarted agent is now listening on a different port. The timestamp allows the controller to determine the last time that an agent contacted it to aid the host sensor in detecting a non-responding host. If an agent fails to broadcast for 3.5 transmit intervals, it is considered by other hosts to be non-responding and may result in an alert being generated.

The Transmission Control Protocol (TCP) Transmitter thread sends messages between agents. A message contains information that an agent needs to report an attack. Since the delivery of such a message helps in the detection of an intruder, some guarantee of delivery must be expected. The User Datagram Protocol (UDP) Transmitter does not provide such assurance, but the TCP protocol is connection-oriented and does [Courtios, 1998]. In the current configuration, the transmitter will deliver any message to all known agents on the contact list. It establishes a connection with the remote agent's TCP Receiver, transmits all currently available messages, and closes the connection.

The TCP Receiver thread picks a port to listen on that is not being used by any other components of the computer on which the agent resides. It returns this port to a global variable in the IDAgent so the UDP transmitter explained above will be able to access it to tell other agents

which port the receiver is listening on. When a message is received, the TCP Receiver queues it for the controller in the message-in queue and continues listening for additional messages. A TCP socket connection must be established between two agents for the message transfer to take place. If a connection cannot be established, the sending agent becomes aware of the problem and can report it to its own controller.

2.4 Message and ContactList class data structures

A Message Class defines message objects that can be constructed and transmitted from one host to another. The class contains a message code, a data field for a description of the message, an identifier, a target address, a source address, a time stamp, and a message weight. The message code indicates why the message was sent. The string data field relates to the code and provides additional description of the code. The identifier supplies operands, if any, for the code. For example, if a message were for a failed login attempt, the identifier would store the account name. The target address is the Internet address and host name of the recipient. The source address is that of the current host. Each message is given a timestamp at origination. The message weight is the relative importance of the message as determined by the controller following the methods described earlier. In the current configuration of the IDAgent, the message size is 787 bytes when the host name is eight characters.

The ContactList class is a data structure storing information about other known agents. It consists of an InetAddress, a port number, and a timestamp. The InetAddress is a Java data type that contains the Internet address and host name of a remote host. The port number is the port that the remote host has a TCP receiver listening on. The timestamp contains the last contact time of a remote agent.

2.5 Host sensor and alert parser

The function of the host-sensor thread is to determine if any remote agents are not broadcasting using the UDP transmitter. It checks the contact list of known agents and compares the time of last contact to the current time. If the host has not responded, an alert message is generated and placed in the controller queue. The controller will calculate a message weight based on previous messages. It will then use the message weight, of this internally generated message, to determine if there is sufficient evidence to update the system alert level. The fraction of hosts not currently responding determines the weight, to limit false alerts when an agent is stopped or restarted by an administrator. The host sensor does not cause any external messages to be generated. It is assumed that each IDAgent will detect a non-responding host, and therefore external messages would be redundant.

The alert parser thread is a utility thread that maintains the internal messages that the controller uses to detect intrusions. The alert parser runs approximately every thirty minutes or six broadcast intervals. It looks through a list of old alerts and discards any over twenty-four hours old. It scans a list of recent alerts and places any over twelve hours old into the old alert list. This allows the controller to run more efficiently when it only needs to scan recent events. For the current configuration, only the recent alerts are used for processing. The alerts over twelve hours old were included for future sensor capabilities and are not currently used.

2.6 Windows NT event logs

The log sensor is another independent sensor thread that automatically retrieves all login attempts from the system log and passes them to the internal log sensor threads.

There are three Windows NT event logs. The Application event log records user-application events, both those selected by the application and system diagnostic events. All processes running under NT have the ability to log events to the application event log. The use of the application log by security programs needs to be expanded to include additional events. The System event log records events logged by Windows NT system services, drivers, and kernel mode events. The Security event log records Windows NT system security and system auditing events. The three event log files are located in the \WINNT\SYSTEM32\CONFIG directory. Each event log record is stored in the EVT binary record format and is only accessible using the Windows event-logging interface. The event log record fields are: Time Generated, User, Computer Name, Event Identifier, Event Source, Event Type, Event Categories, and Message Strings. We conducted experiments with just the security event log.

Twelve functions of the event logging interface [Microsoft Corp., 1999] can be used by application programs. OpenEventLog gets a handle to an already-open event log, opened by enabling auditing, while CloseEventLog closes it and releases its resources. ReadEventLog reads the specified event log. GetOldestEventLogRecord gets the index of the latest record and GetNumberOfEventLogRecords returns the number of records. RegisterEventSource returns a handle of a log file to be used by ReportEvent to write an event entry to an event log. This functionality allows a user program to report events to the event logging service for recording in the Application event log. DeregisterEventSource closes the log. BackupEventLog copies the log before clearing to an archive; ClearEventLog clears all the events in a log file. OpenBackupEventLog opens a backup log file. NotifyChangeEventLog allows the logging service to signal to an event object created by CreateEvent, when a record is written to a specified log file. It allows the near real-time processing of security-event records as they are written, before they can be altered or deleted.

[Staniford-Chen et al, 1999] argues for the importance of a common record format to allow intrusion data to be shared across a network. To this end, we create data records with fields: Date & Time Generated, Event ID, Event Type, Event Category, User Name, Computer name (Domain), Login Type, and Workstation. The format of the fields is ASCII text and they are comma-separated. The records are of variable length and fields reported are dependent on the record Event Identifier.

2.7 Implementation of the log sensor

The Audit Log Sensor was written in C, using Microsoft Visual C++ 5.0 using the Microsoft Visual Studio 97 environment. The executable program size is approximately 100 kilobytes. It makes requests of the event logging service by calling the abovementioned functions to perform reading, backing up, and clearing of the security event log. The API (interface) Dynamic Link Library defines the functions and is used by user-mode Win32 processes to access the security-event log. The log can only be accessed by an user account with the “manage auditing and

security log" privilege set. The event logging service can also access remote system' event logs by using Remote Procedure Calls but these calls were not explored. Figure 2 shows a block diagram. When important information is received from the sensor, the IDAgent's controller for that machine will construct a message and send it to other agents in the network to notify them of a security event or action.

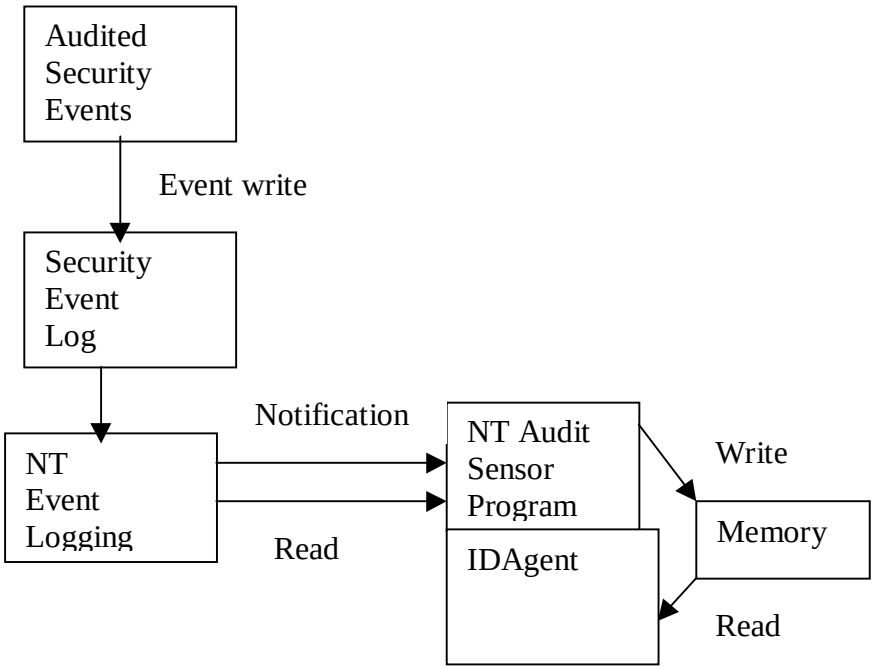

Figure 2: Context of the NT Audit Log Sensor

The Main routine of the program uses the function NotifyThread to create a separate program thread to monitor the security event log and then wait for input. The NotifyThread routine was adapted from examples from [Murray, 1998]. Within the thread, NotifyChangeEventLog associates the event file handle with an event object. The thread uses the WaitforSingleObject to detect when a new event log entry is written.

OpenEventLog then obtains a handle to the already open log. GetOldestEventLogRecord gets the index of the latest record and passes it to ReadEventLog to read the log. The Event Identifier finds the records of interest. The record is read in and reformatted to omit extraneous data, and is transferred to the IDAgent program. CloseEventLog closes the log and this releases the resources. The event object is then reset by ResetEvent to wait for the next event.

3. Testing of our implementation

3.1 Testing of the Log Sensor

We performed a variety of tests suggested by those of comparable systems [Durst, 1999; Puketza et al, 1997]. The NT Audit Log Sensor was tested on a dedicated NT workstation where NT security events of login success, login failure, and locked account occurred and were logged.

The Log Sensor was notified through the NotifyChangeEventLog function when something was written to the NT security-event log, and if the event was of the types mentioned, it was parsed, reformatted, and passed to the IDAgent. Initial standalone tests required the simulation of an IDAgent by a simple program that read the data and wrote to a disk file where the complete transfer could be checked. The Log Sensor provides real-time parsing of the security-event records to accomplish the following checks (admittedly incomplete) using Event Identification as the primary event-record key.

Detection: Audit Policy changed or Event Log cleared.
Action: Monitor the Security Event Log entry for Event Identification 517 where the user name is other than the Security Manager. The Log Sensor provides a record for Event Identification 517, "Event Log cleared" from the first record to the new log file after the log has been cleared. Monitor the Security Event Log entries for Event Identification 612 where the user name is other than the Security Manager. The sensor provides the record for Event Identification 612, "Audit Policy Changed".

Detection: Attempt to exploit default NT user accounts, Administrator and Guest.
Action: Monitor the Security Event Log entries for Event Identification 529 where the account name is any spelling variation of account "Administrator" or "Guest". The sensor provides the record for Event Identification 529, "Login Failure".

Detection: Multiple Login Failures suggesting intrusion attempt.
Action: Monitor the Security Event Log entries for Event Identification 529. Compare the number from a workstation over a period of time. The sensor provides the record for Event Identification 529, "Login Failure".

Detection: Account lockouts suggesting intrusion attempt.
Action: Monitor the Security Event Log entries for Event Identification 539 and Event Identification 644. The sensor provides the record for Event Identification 539, "Account locked" and for Event Identification 644, "User Account Locked".

Detection: User account added, deleted, or modified.
Action: Monitor the Security Event Log entries for Event Identifications 630, 624, and 642. Sensor provides the record for Event Identification 630, "User account delete", 624, "User account created", and 642, "User Account Modified".

3.2 Restricted network testing

To test the networking facilities, a program was designed that uses some simple detection mechanisms along with all the necessary components to transmit and receive data over the Internet. The first test was performed in the early stages of code development. Only a basic agent skeleton with TCP Transmitter and TCP Receiver classes was used in order to make an initial determination of network overhead usage. Follow-on tests were then conducted with other parts of the agent operating to get the full effect on network and CPU utilization. Finally, simulated alert messages were sent to see how the agents reacted and what impact this reaction would have on CPU utilization of the host.

Initial throughput testing of the IDAgent was conducted on a closed network of three Micron 166Mhz Pentium computers, each running the Windows NT 4.0 operating system as server or workstation. Two of the machines were configured as workstations with 32MB of RAM, and one was configured as a server with 64MB of RAM. To prove portability of the basic agent, tests were also performed on the same machines running the Linux operating system version 5.2. However, no other portions of the testing were done on Linux, and the results were only used to show portability of the agent to other platforms.

The agents had no detection capabilities or message processing capabilities during the initial testing. Only the network-bandwidth utilization was compared. Using an Observer® network-packet "sniffer" (a software program used to capture and analyze information being transmitted over a network), We monitored the network to determine the average bandwidth utilization for the three agents on a 10Mbps Ethernet 10BaseT network. The bandwidth measurement includes usage resulting from network polling, broadcasts, and network overhead on both the Windows NT and Linux operating systems. The IDAgents were configured to send 5,000 static messages of approximately 155 bytes each to each of the other agents. With three agents running, 45,000 messages or approximately 6.975 Megabytes of data was transmitted. The test was repeated three times to get an average transmit time. We found an average network-bandwidth utilization of 8% with an average standard deviation of 1% and a maximum of 2%. The transmission of 45,000 messages took approximately 110 seconds, and the average bandwidth never exceeded 10% of the 10-megabit Ethernet network. The results were very encouraging since it will rarely be expected that an agent will need to transmit 5000 messages in such a short time.

3.3 General network testing

The second test was conducted on an open network in the Computer Science Department at the Naval Postgraduate School. The subnet we used is the same one used by most students, faculty, and researchers in the department. We used the same three computers and added four additional workstations, all running the Windows NT operating system version 4.0 workstation. The IDAgent was fully configured and included login detection and host failure detection as described earlier. Any successful or unsuccessful login attempts generate a message from the agent sent to all other hosts that have the IDAgent running; a broadcast message is sent by each host at five-minute intervals to update the contact list of known agents. Some general assumptions were made for this test to determine what an adequate number of login attempts should be. We estimated the number of daily login alerts based on a ten-user network. We assumed each user performs a login approximately three times a day. We assumed each user locks the computer screen an additional four times a day, requiring a password to unlock it, and generating an authentication alert. Windows NT authenticates users on the network who map a drive to a shared resource, which also generates an authentication alert on login since the resource is still open during a screen lock. It is assumed for this scenario that each user maps two network drives: one for shared applications and one for a shared file storage location. An expected login failure rate of 15% is set as the threshold in the IDAgent to reduce false alerts. With these assumptions, a network of ten users will generate approximately 130 login alerts per day, which will average approximately 16.25 logins per hour.

Using the same network sniffer as earlier, we monitored the network with no agents running for one hour to establish baseline utilization. We then ran seven IDAgents on the network for one hour, generating over 50 login alerts and producing over 400 message transmissions. This is three times more than the average number for an hourly period in our assumptions.

Figures 3 and 4 show the average number of network packets per second, average number of broadcasts per second, and percentage of network-bandwidth utilization for the one-hour period both with and without agents running. The average number of packets sent while the agents were running was actually slightly lower than without. The average number of broadcast messages increased slightly as expected; the average network utilization decreased slightly as shown in Figure 4, which was not expected. However, looking at the percentage of bandwidth used, the slight drop is insignificant when compared to the total bandwidth available. The "average maximum utilization" averages the peak bandwidth usage for each ten-second interval. This average went up slightly from 1.9 to 2.3, which indicates that the packet transmissions show more short bursts of data. From the data collected, it appears that the IDAgent has little effect on bandwidth in an open network. During the hour that the agents were running, approximately 64,000 packets were captured. Of those packets, only 5,391 were from one of the computers running an IDAgent, about 8%; the rest were from normal network activity.

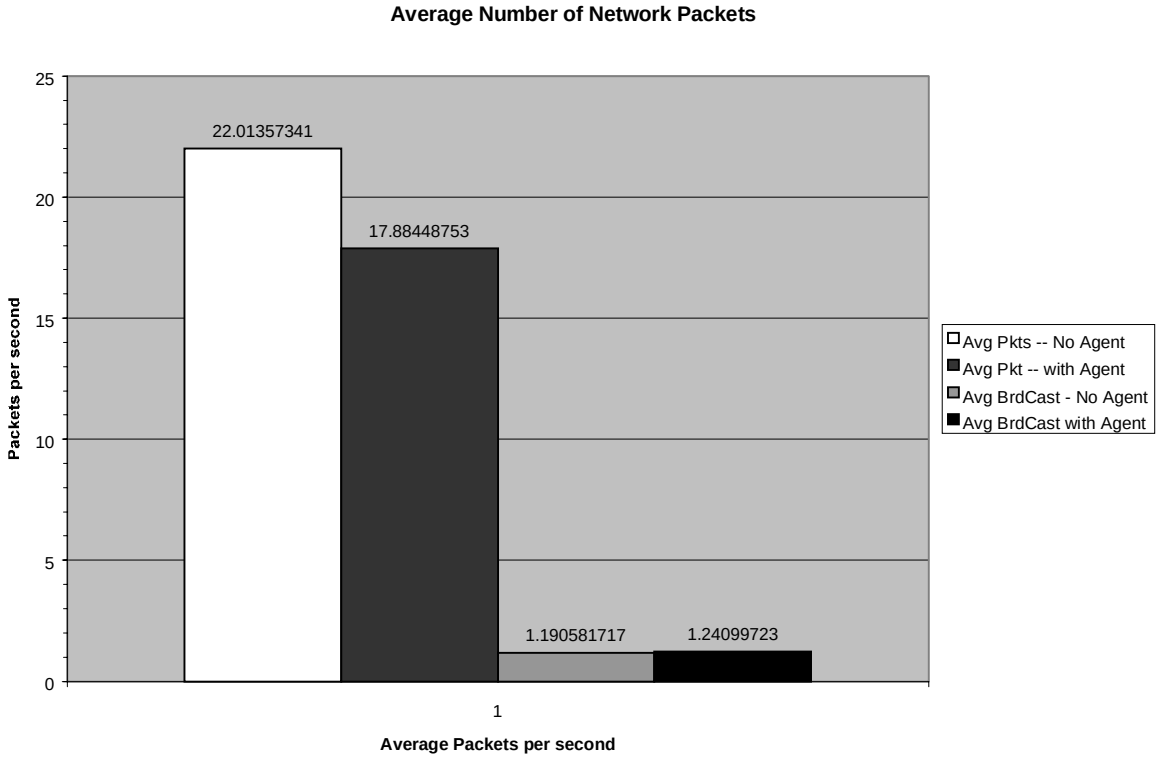

Figure 3: Average Packets and Broadcasts

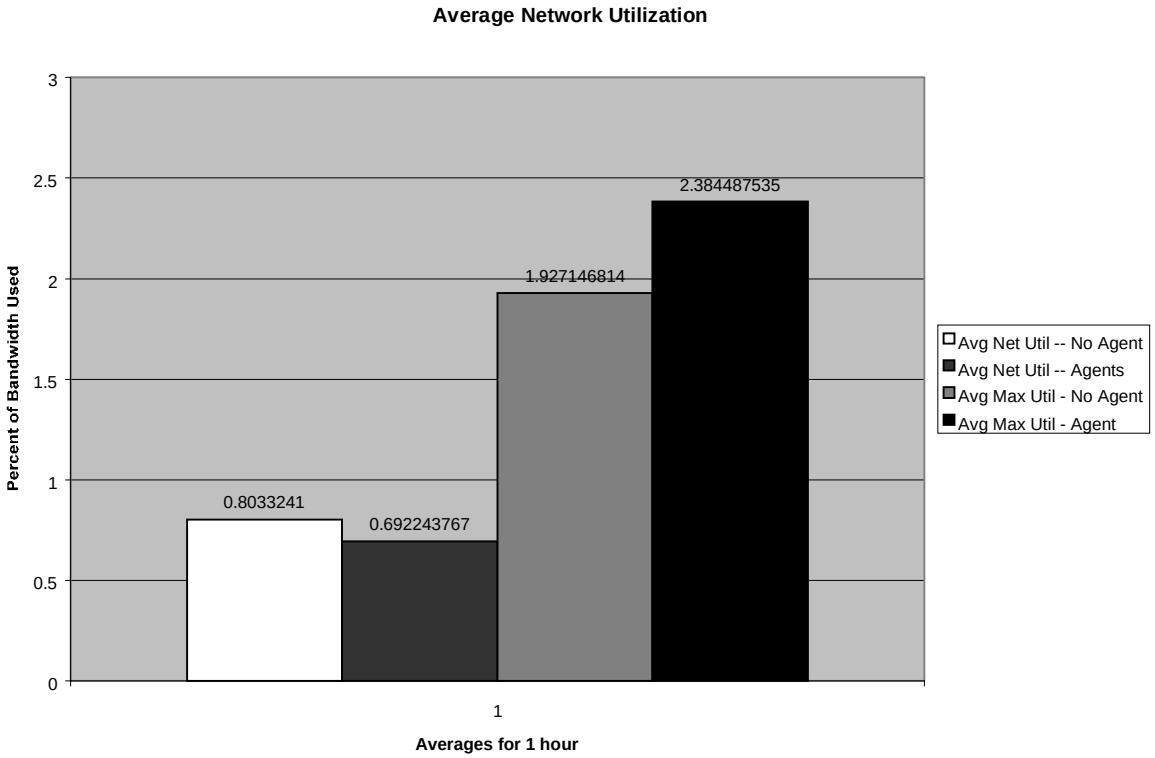

Figure 4: Percent Network Bandwidth Utilization

3.4 CPU utilization tests

Another test was conducted using the Windows NT performance monitor and logging tool. We were able to log and graph the CPU utilization over time with an IDAgent running to see its impact. We configured the performance monitor to log processor usage for user programs and started one IDAgent; no other user programs were running on its computer. An IDAgent was also started on another host and login alerts were generated from both computers. During the thirty-minute analysis period, approximately 20 alerts were generated. The maximum CPU utilization by the agent was 8.145%. The average utilization over the entire period was 0.329%. There are several small usage periods, when the IDAgent was active in receiving and sending messages.

3.5 Simulated attacks

Several scenarios were used to test the reaction of the IDAgent. Three computers were used. In the first scenario, all computers had several successful logins from users, then one computer had a series of unsuccessful login attempts on a single account. In the second scenario, several successful login attempts were performed on each computer, followed by a series of unsuccessful attempts. In the third scenario, all three computers were used, and many rapid consecutive unsuccessful login attempts were made from a single administrator account on one machine.

After allowing all three agents to run for several minutes with no activity, two successful logins were made on each computer followed by an attack on machine three. The attacker produced six

successive login failures. Figure 5 shows the login attempts and reactions of the agents with their corresponding alert level changes. Machine three responded differently because its weight calculation was based on attempts being made on its own host, while the other two machine calculations were based on attempts throughout the entire network because the messages originated from another machine. The result is a higher alert level on the machine where the attack is taking place.

Total # of Attempts	Total # of Failures	Machine #1 Message Weight	Machine #1 Alert level	Machine #2 Message Weight	Machine #2 Alert level	Machine #3 Message Weight	Machine #3 Alert level
7	1	0.0	0.1	0.0	0.1	0.0	0.1
8	2	0.0	0.1	0.0	0.1	0.1	0.19
9	3	0.072	0.1648	0.072	0.1648	0.249	0.392
10	4	0.150	0.290	0.150	0.290	0.35	0.605
11	5	0.2136	0.4417	0.2136	0.4417	0.4214	0.771
12	6	0.2666	0.5905	0.2666	0.5905	0.475	0.880

Figure 5: Alert Levels for Single Target Attack

The second scenario was much like the first but with an attacker attempting to login on to all three machines simultaneously instead of just one. Figure 6 shows the results of the test. The alert levels for all machines were very close together since login failures were spread across all hosts. The second machine reached a yellow alert level of 0.423 on the twenty-first login attempt with three local failed attempts, four remote failed attempts, and fourteen successful logins. The remaining machines reached a yellow alert level of 0.486 after two more attempts, one successful and one failing. A total of twenty-three logins were attempted, eight login failures and fifteen successful logins.

Total Login Attempts	Machine #1 Login Failures	Machine #2 Login Failures	Machine #3 Login Failures	Machine #1 Message Weight	Machine #1 Alert Level	Machine #2 Message Weight	Machine #2 Alert Level	Machine #3 Message Weight	Machine #3 Alert Level
11	1	0	0	0.0	0.1	0.0	0.1	0.0	0.1
13	1	1	0	0.0	0.1	0.0	0.1	0.0	0.1
14	1	2	0	0.0	0.1	0.05	0.1450	0.0	0.1
16	1	2	1	0.0375	0.1337	0.0375	0.177	0.0375	0.1337
17	1	2	2	0.0852	0.2076	0.0852	0.247	0.0852	0.2076
19	2	2	2	0.1131	0.2972	0.1131	0.3324	0.1131	0.2972
21	2	3	2	0.1357	0.393	0.1357	0.423	0.1357	0.393
23	2	3	3	0.1543	0.486	0.1543	0.512	0.1543	0.486

Figure 6: Alert Levels for Multiple Target Attack

The IDAgent is designed to suspend transmission of messages for a short period of time if it comes under a repeated attack, to prevent a flood of network traffic from its own messages. To test this, three test machines were started and 40 rapid login attempts were made against an administrator account on a single host. After transmitting 25 messages to the other agents, the IDAgent being attacked continued to log the attack, but it did not continue transmitting messages until five minutes after the attack had stopped. Agent response was successful: The attacked machine had an alert level of 1.0, the highest that can be reached, while both remaining agents had an alert level of 0.999.

4. Conclusions

This paper has proposed distributed nonhierarchical autonomous agents as an intrusion-detection mechanism. Testing with an implementation of such an agent in this environment showed that neither CPU utilization nor network utilization were heavily loaded by the IDAgent. Even with over 50 login attempts within one hour, the network traffic, broadcasts, and processing did not interfere with normal computer and network operations. The IDAgent was also able to detect several scenarios of login attempts from both a single host and multiple hosts, and escalated the alert level of each agent appropriately.

We are currently investigating further ideas to improve the IDAgent approach:

- Security or secure message handling could prevent blocking of messages or generation of false messages.
- Authentication could determine if an agent that is responding is really a trusted agent or a piece of malicious software.
- IDAgent could run as a NT "service" to prevent IDAgent from terminating and leaving its system vulnerable.
- The Log Sensor could be executed as a system service, started at system boot-up under NT System Services.
- The Log Sensor could filter more false alarms, and should periodically report to the IDAgent that it is alive.
- Other sensors could scan for network traffic patterns, known attacks, or other system log entries; the agent was written in a modular fashion to allow such sensor threads to be included easily.
- The systems could be reactive to an attack to prevent entry and provide countermeasures.

References

Barrus, Joseph D. “Intrusion Detection In Real Time In A Multi-Node, Multi-Host Environment”, M.S. thesis, Computer Science Department, Naval Postgraduate School, September 1997.

CINCPACFLT, Navy Administration Message: “Information Technology for the 21st Century,” R300944, March 1997.

Courtios, Todd, Java Networking & Security, Prentice-Hall, Inc., Upper Saddle River, NJ, 1998.

Durst, Robert. Terrence Champion, Brian Witten, Eric Miller, and Luigi Spagnuolo, "Testing and Evaluating Computer Intrusions Detection Systems", Communications of the ACM, July 1999, Vol. 42, No. 7, 53-61.

Hale, Ron, "Intrusion Crack Down", Information Security, August 1998.

GAO (Government Accounting Office), "Information Security: Computer Attacks at Department of Defense Pose Increasing Risks," GAO/AIMD-96-84, May 22, 1996.

Ingram, Dennis J., "Autonomous Agents for Distributed Intrusion Detection in a Multi-host Environment", M.S. thesis, Computer Science Department, Naval Postgraduate School, September 1999.

Kremer, H Steven, "Real-time Intrusion Detection for Windows NT Based on Navy IT-21 Audit Policy," M.S. thesis, Software Engineering Curriculum, Naval Postgraduate School, Monterey, CA, September 1999.

Lunt, Theresa F. "A Survey of Intrusion Detection Techniques", *Computers & Security*, 12:405-418, 1993.

Microsoft Corporation. "MSDN Online", http://msdn.microsoft.com/library/sdkdoc/winbase/, May 1999.

Murray, James D. *Windows NT Event Logging*, O'Reilly, 1998.

Neumann, Peter, G., and Phillip A. Porras, "Experience with EMERALD to Date", Proceedings 1st USENIX Workshop on Intrusion Detection and Network Monitoring Santa Clara, CA, April 1999.

Proctor, Paul E., "Computer Misuse Detection System (CMDS) Concepts", Science Applications International Corporation (SAIC), May 1996.

Puketza, Nicholas, Mandy Ching, Ronald A. Olsson, and Biswanath Mukherjee. "A Software Platform for Testing Intrusion Detection Systems", *IEEE Software*, pp. 43-51, Sept.-Oct. 1997.

Staniford-Chen, Stuart, Brian Tung, and Dan Schnackenberg. "The Common Intrusion Detection Framework (CIDF)", http://seclab.cs.ucdavis.edu/cidf/papers/isw.txt, August 30, 1999.

Sobirey, Michael, and Birk Richter, "The Intrusion Detection System AID", Brandenburg University of Technology at Cottbus, http://www-rnks.informatik.tu-cottbus.de/~sobirey/aid.e.html.

Zamboni, Diego, Jai Sundar Balasubramaniyan, Jose Omar Garcia-Fernandez, David Isacoff, and Eugene Spafford, "An Architecture for Intrusion Detection Using Autonomous Agents", COAST Technical Report 98/05, COAST Laboratory, Purdue University, June 1998.