Distributed Intrusion Detection for Computer Systems Using Communicating Agents

CAPT 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 the Internet, and the Windows NT operating system has a number of vulnerabilities. The work presented here demonstrates that independent detection agents under Windows NT can be run in a distributed fashion, each operating mostly independent of the others, yet cooperating and communicating to provide a truly distributed detection mechanism without a single point of failure. The agents can run along with 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 the central detection system, then most, if not all, protection is subverted.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2000		**00-00-2000 to 00-00-2000**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Distributed Intrusion Detection for Computer Systems Using Communicating Agents	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Marine Corps Warfighting Laboratory,3255 Meyers Avenue,Quantico,VA,22134**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**27**	

We describe here the design and implementation of 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. All major components of the agents are constructed as threads to allow them to run concurrently. The main components are written in Java and are a 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 Log Sensor is a process in C that continually checks the Windows NT Security Event Log for a variety of suspicious events (like repeated failed login attempts), and alerts the IDAgent processes immediately via pipes when it finds something, making use of Windows NT utility routines. This approach to security is more aggressive than that specified under Navy IT-21 policy, and we will discuss some of its advantages.

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.

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 records of a single host for processing and analysis. This type of system is limited in scope since it is only able to see its own 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 Internet Protocol (IP) addresses that originate 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 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 requires a dedicated computer to act as the central controller; centralization can make it vulnerable to attack.

Today's hackers use several categories of attacks ranging from simple to very complex. The basic categories are listed in Figure 1.

One-to-one	Attacker uses a single machine to attack a single target machine. Example: sendmail bugs.
One-to-many	Attacker uses a single machine to attack many targets. Example: probes, denial of service attacks.
Many-to-one	Attacker divides assault among multiple outside machines to attack a single victim. This is difficult to detect because multiple connections from multiple sources look more innocent than multiple connections from a single source. Example: SYN flood using IP spoofing to deny services.
Many-to-many	Many collaborating attackers divide the tasks of probing/attacking multiple victims. This poses the same challenge as the "many-to-one" case with the added complexity of multiple target machines. This kind of attack is very difficult to detect. Example: "Smurf" attack from multiple sources.

Figure 1: Attack Types (from [Durst, 1999])

[Barrus, 1997] defined ten basic requirements for a good intrusion detection system:

1. A system must recognize any suspect activity or triggering event that could potentially be an attack.
2. Escalating behavior on the part of an intruder should be detected at the lowest level possible.
3. Components on various hosts must communicate with each other regarding level of alert and intrusions detected.
4. The system must respond appropriately to changing levels of alertness.
5. The detection system must have some manual control mechanisms to allow administrators to control various functions and alert levels of the system.
6. The system must be able to adapt to changing methods of attack.
7. The system must be able to handle multiple concurrent attacks.
8. The system must be scalable and easily expandable as the network changes.
9. The system must be resistant to compromise, able to protect itself from intrusion.
10. The system must be efficient and reliable.

2. Related work

There are hundreds of systems available that perform intrusion detection, intrusion prevention, and system security checking. Many perform well and provide a robust detection mechanism, but few run in a fully distributed environment. Of those that are distributed, many are Unix-based systems and will not run on Windows NT platforms. There are fewer still that are portable between operating systems.

The systems most similar to the one presented in this paper all have one major difference from it, in that they are hierarchical in nature. This places the highest vulnerabilities at the upper level of the hierarchy. Degrading or disabling a top-level monitor would severely limit the detection

capability of the system. None of these systems mention the use of an alert level to determine if an attack is in progress.

AID [Sobirey and Richter, 1999] is a client-server architecture that consists of agents residing on network hosts and a central monitoring station. Information is collected by the agents and sent to the central monitor for processing and analysis. It currently has implemented 100 rules and can detect ten attack scenarios. The prototype monitor is capable of handling eight agents. This system currently runs only on UNIX-based systems.

The AAFID architecture [Zamboni et al, 1998] appears the most similar to the one we propose. AAFID is designed as a hierarchy of components with agents at the lowest level of the tree performing the most basic functions. The agents can be added, started, or stopped, depending on the needs of the system. AAFID agents detect basic operations and report to a transceiver, which performs some basic analysis on the data and sends commands to the agents. A transceiver may transmit data to a transceiver on another host. If any interesting activity takes place, it is reported up the hierarchy to a monitor. The monitor analyzes the data of many transceivers to detect intrusions in the network. A monitor may report information to a higher-level monitor. The AAFID monitors still provide a central failure point in the system. AAFID has been developed into two prototypes: AAFID, which had many hard-coded variables and used UDP as the inter-host communication, and AAFID2, which was developed completely in PERL and is more robust. They run only on Unix-based systems.

CMDS™ is a commercial product from Science Applications International Corporation (SAIC) [Proctor, 1996]. It is a real-time audit reduction and alerting system that uses an expert system and statistical profiling to analyze audit records. The system uses distributed daemons running on host machines to monitor audit files. Information is sent to the CMDS central server for analysis by a rule-based expert system. It also uses a hierarchical architecture with several CMDS servers reporting to a higher CMDS system. It currently supports the operating systems SunOS, Windows NT, Solaris, Trusted Solaris, Ops Intel Workstation, Data General DSO, HP/UX, IBM LAN Server, Raptor Eagle Firewalls, ANS Interlock Firewalls, and SunOS BSM. This program appears to be robust across many platforms.

EMERALD [Neumann, 1999] is a system developed by SRI International with research funding from DARPA. The EMERALD project will be the successor to Next-Generation Intrusion Detection Expert System (NIDES). It is designed to monitor large distributed networks with analysis and response units called monitors. Monitors are used sparingly throughout the domain to analyze network services. The information from these monitors is passed to other monitors that perform domain-wide correlation, obtaining a higher view of the network. These in turn report to higher-level enterprise monitors that analyze the entire network. EMERALD is a rule-based system. The target operating system has not been stated, but it is being designed as a multi-platform system. EMERALD provides a distributed architecture with no central controller or director; since the monitors are placed sparingly throughout the network, they could miss events happening on an unmonitored section.

3. Design of a distributed intrusion-detection system

In designing our IDAgent approach, we 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 contained in the M.S., theses [Ingram, 1999] and [Kremer, 1999], both of which are available online from http://www.cs.nps.navy.mil/people/faculty/rowe/index.html.

3.1 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. One problem discovered was that platform independence was difficult to achieve for an intrusion-detection system because many of the mechanisms used to detect intrusions, such as system logs and system alert facilities, are specific to a certain platform. For example, a collection of data from a system log is processed differently on a Windows NT platform than it is 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 2 shows a simple block diagram of the IDAgent components needed for a single host. All major components of the agent are constructed as threads to allow them to 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.

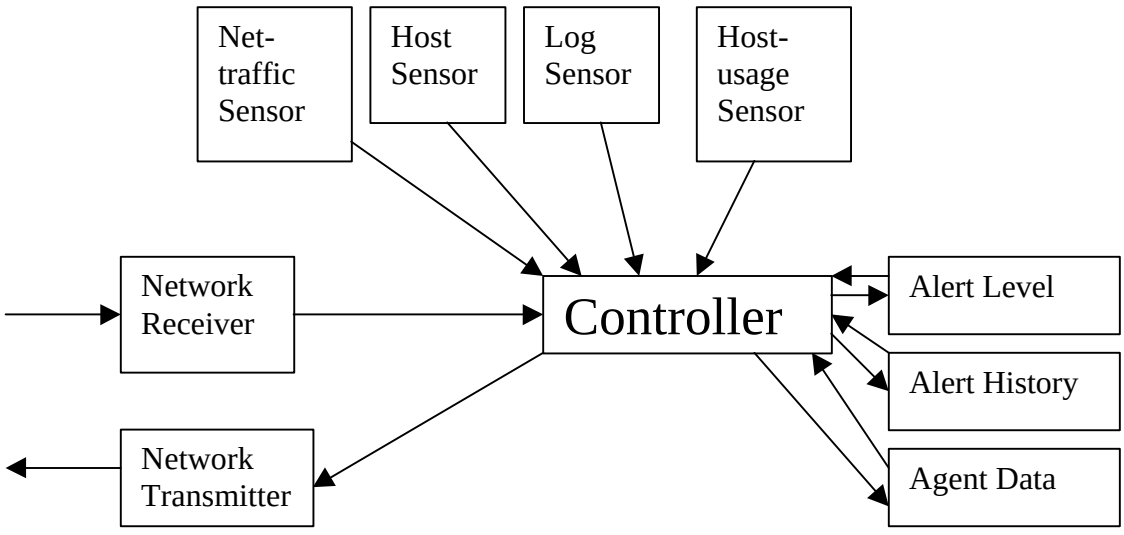

Figure 2: IDAgent Block Diagram

The Agent Window Manager (written initially by Major Jim Breitinger for another project) was used after modifications for 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 the generation of debug data for testing, the display of the current contact list of known agents, a display of alert messages that caused a change to the alert status, and a status which indicates a numerical value of the current alert level.

3.2 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 itself when there is no activity. 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; 0.4 to 0.7 will show a yellow indicator; and above 0.7 will display red.

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, as shown in Figure 3, 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 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). Figure 4 shows the decrease over time.

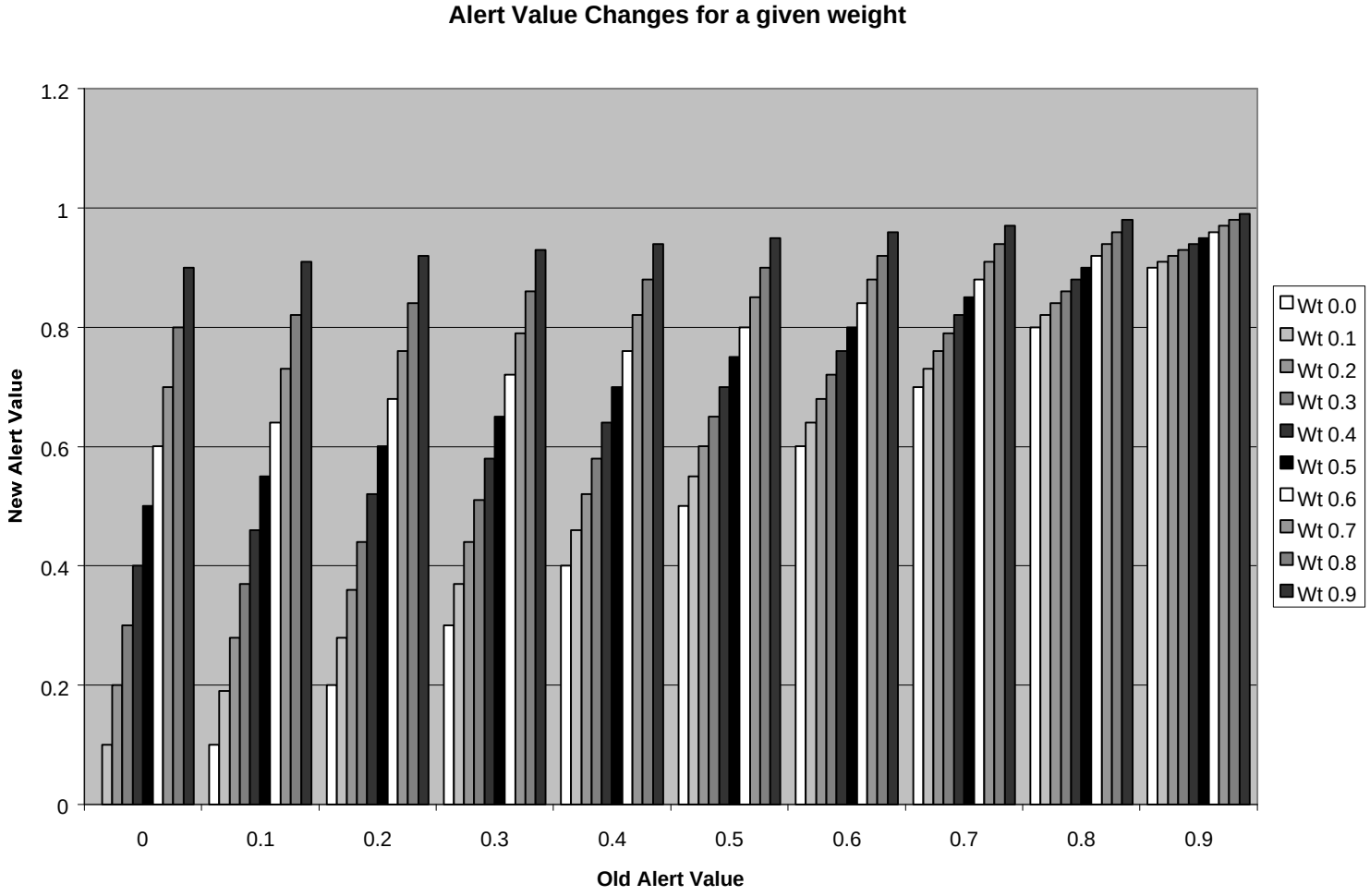

Figure 3: Alert-Level Increase

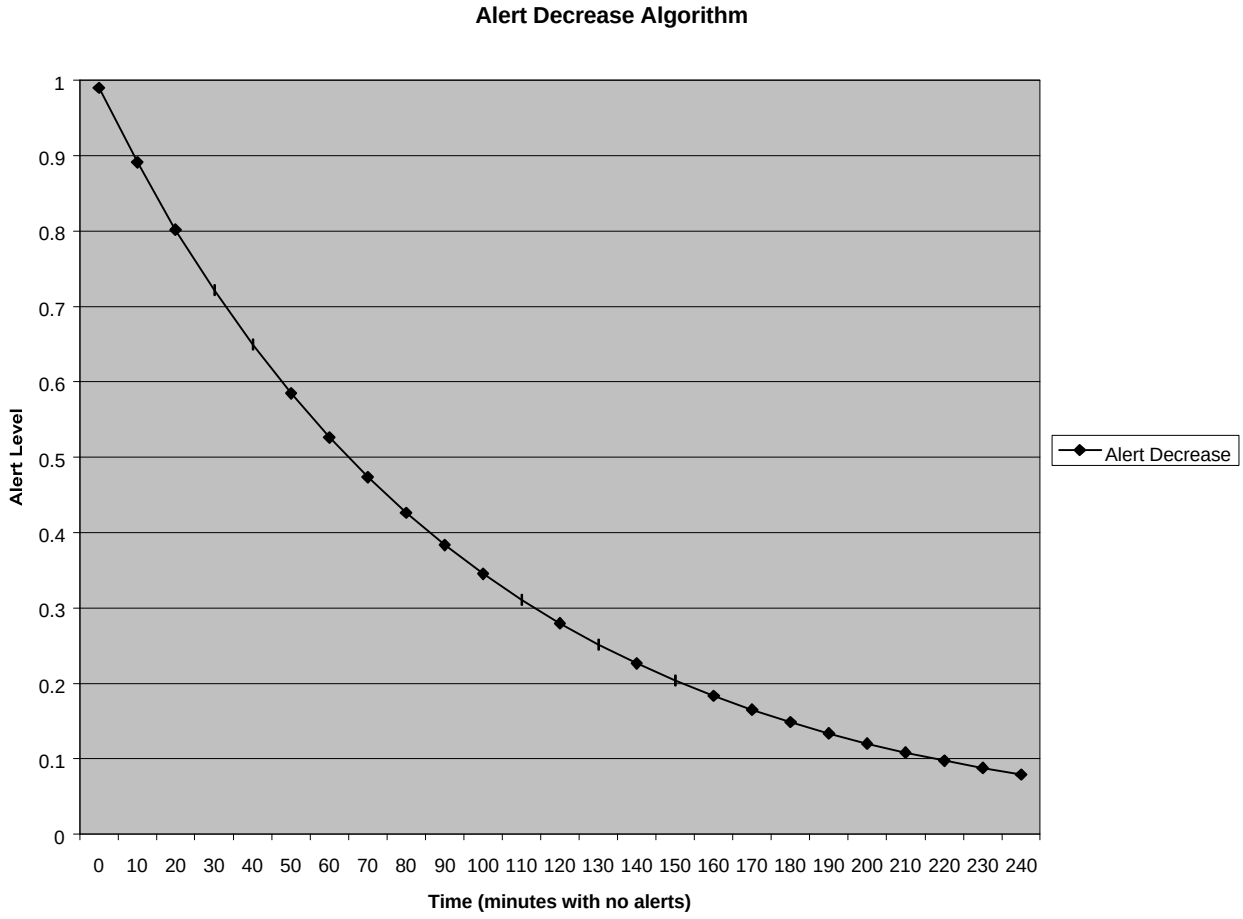

Figure 4: Alert-Level Decrease

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 (section 4), 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

following formula = ((LoginFailures / TotalAttempts) – (1 / TotalAttempts)). Figure 5 shows the effect of this calculation.

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.

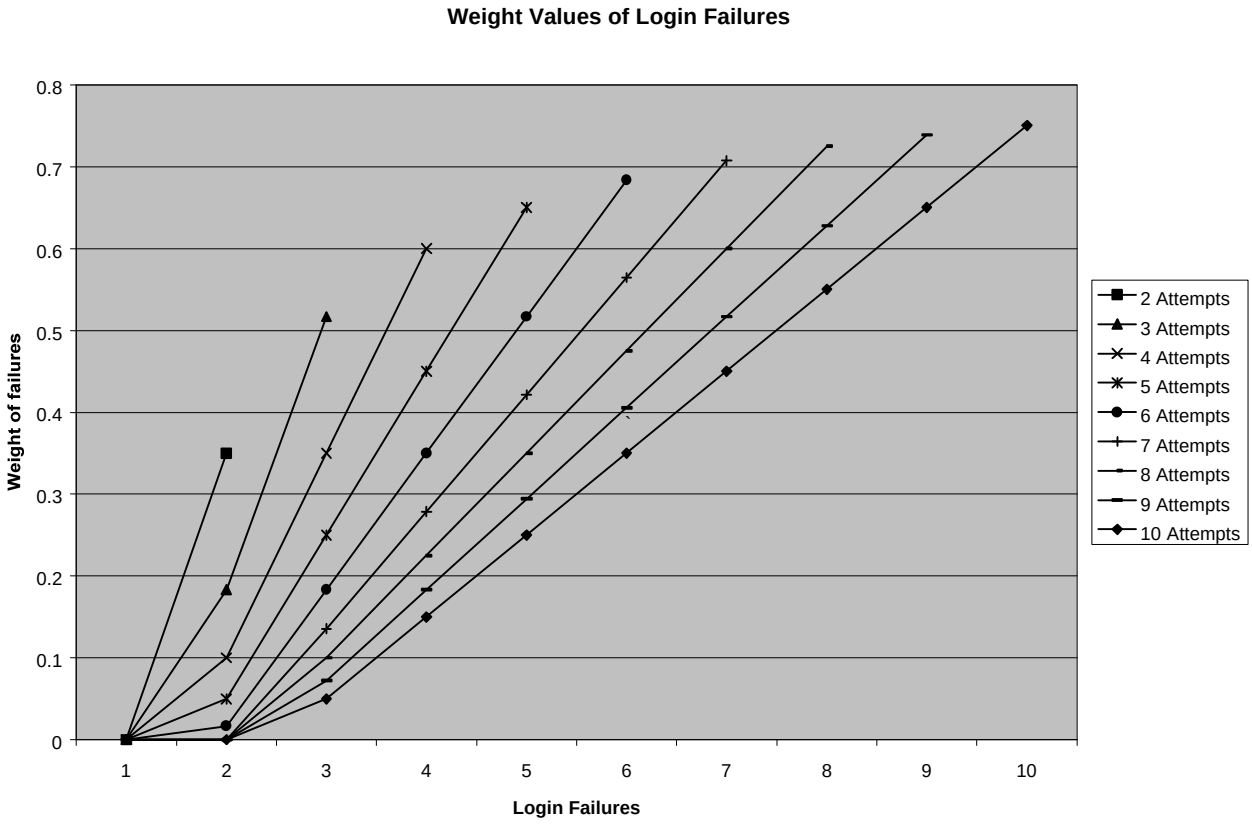

Figure 5: Login-Failure Calculation

3.3 UDP Transmitter and receiver

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 every five minutes. 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 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 in case an agent was restarted and is now listening on a different TCP 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 this may result in an alert being generated.

3.4 TCP transmitter and receiver

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 should become aware of the problem and can report it to its own controller.

3.5 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 in section 3. 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.

3.6 Host sensor

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.

3.7 Alert parser

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.

3.8 Audit Log Sensor

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 thread. It will be discussed in more detail in the next section.

4. The audit Log Sensor for intrusions

The Navy has issued an order “IT-21” [CINCPACFLT, 1997] providing direction for migration to the Microsoft Windows NT 4.0 operating system for workstations and network servers, to be completed by December 1999. IT-21 was written considering the available NT audit tools that, up to NT Service Pack 4.0 with the addition of Security Configuration Manager, have been meager indeed. Without the advantage of commercial security tools specifically able to implement an audit policy across an entire network, any additions to the IT-21 audit policy could not have been implemented. As written, the IT-21 audit policy is only an NT audit capability guide, not a true security audit policy.

It is inappropriate to write a security-policy specification based only on available tools. While the IT-21 guide is an NT security implementation guide, without an NT security policy specification guide, the implementation guide by default dictates policy. Perhaps the fault lies not in the design of the implementation guide but in the lack of an NT security policy specification guide.

The limitations of the NT audit capabilities should not dictate audit policy. The size and the maintenance problems of audit files should not result in a tradeoff of size versus manageability and therefore limit auditing. The number of false alarms should not be the reason to turn off a segment of auditing but rather filtering rules should limit the number of false alarms. Another limitation on IT-21 audit policy is that there is no provision for reporting from the audit logs after the audit data has been collected.

The manual analysis of audit files has been successful in the past when only one system was being evaluated, but the continual growth and expansion of networks makes that no longer feasible. If better analysis tools are not developed to analyze and report on host-based intrusions and misuse, we might as well turn off all auditing altogether, for without automated analysis all we will see is what we have always seen from the auditing logs – highly unusual events that stand out. Such events will not appear for a trusted individual who is determined to compromise sensitive data and undermine the security of the military network.

In designing the Log Sensor we therefore looked at a broader class of intrusive activities than are currently addressed under IT-21.

4.1 Windows NT event logs

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. We conducted experiments with just the security event log.

The event log record fields are:

> The Time Generated field contains the date and time of the event measured in seconds since Jan. 1, 1970, at 00:00.
>
> The User is the user account associated with the event.
>
> The Computer Name is the computer on which the event occurred.

The Event Identifier is the code number of reported event. This value is specific to the event source.

The Event Source is the source name, application, service, driver, or subsystem that reported the event.

The Event Type classifies the type of event. The Event Types of information, warning, and error are application-log types, while success-audit and failure-audit are security-log types.

The Event Categories identifies logical categories to which this event belongs.

The Message Strings describe the event in more detail.

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.

Here is an example of a security-event record. Event Identifier 539, an account being locked out, was created by three logins attempts to an account with an incorrect password when the set limit was two attempts. The NT Event Viewer program shows the record, which is an input to the NT Audit Log Sensor.

Date	07/15/1999	
Time	7:03:19 PM	
User	NT AUTHORITY\SYSTEM	
Computer	D871WS01	
Event Identifier	539	! 539 == Account Locked Out
(Event) Source	Security	
(Event) Type	Failure Audit	
(Event) Category	Login/Logoff	
Detail view:		
Description:	*Event specific*	
Reason:	Account locked out	
User Name:	admin	! Name of user
Domain:	D871WS01	! Domain (if server, else computer)

Login type:	2	! 2 == local, 3 == remote
Login Process:	NWGINA	! Process submitting login request
Authentication package:	MICROSOFT_AUTHENTICATION_PACKAGE_V1_0	
		! Program used to authenticate login
Workstation Name:	D871WS01	

[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. An example record for the same event above is:

Date	7/15/1999	
Time	19:03:19 PM	
Event Identification	539	
Event Type	16	! == Failure Audit
Event Category	2	! == Login/Logoff
User	admin	
Computer	D871WS01	!Domain
Login	2	!(local 2 or remote 3)
Workstation	D871WS01	

4.2 Implementation of the log sensor

Our Log Sensor 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 in this program. Figure 6 shows a block diagram of the sensor interface between the NT Audit Sensor and the IDAgent.

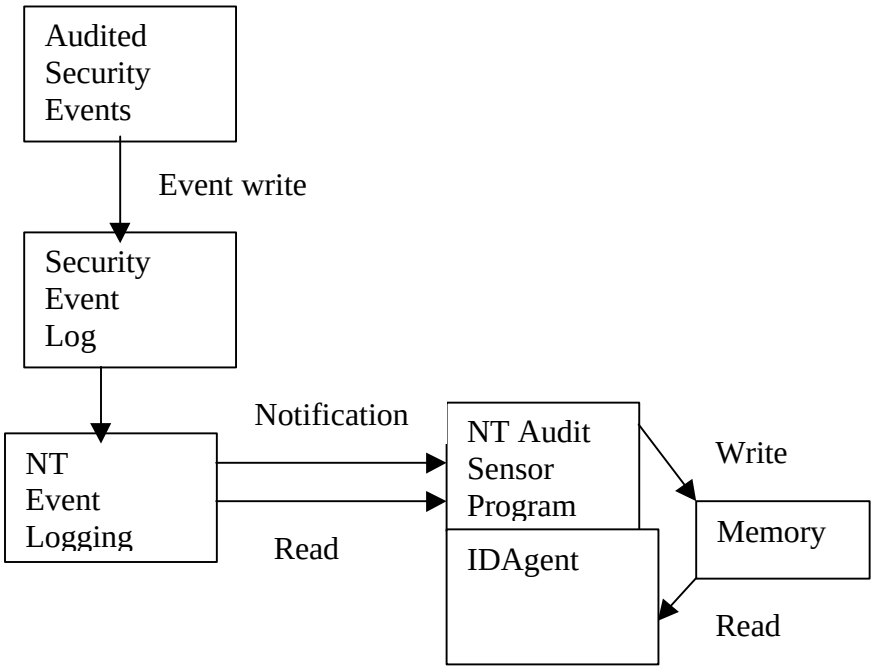

Figure 6: Context of the NT Audit Log Sensor

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.

The Audit Log Sensor was written in C, using Microsoft Visual C++ 5.0. It was developed, compiled and linked using the Microsoft Visual Studio 97 environment. The executable program size is approximately 100 kilobytes.

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 via a pipe, which is a section of shared memory that processes use for communication. CloseEventLog closes the log and this releases the resources. The event object is then reset by ResetEvent to wait for the next event.

5. Testing of our implementation

We performed a variety of tests suggested by those of comparable systems [Durst, 1999; Puketza et al, 1997].

5.1 Testing of the Log Sensor

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 written to a pipe. Initial standalone tests required the simulation of an IDAgent by a simple program that read the pipe 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.
Description: Verify that Audit Policy changes reflect the organization policy and that the Event Log is only cleared by the Security Manager.
Action: Monitor the Security Event Log entry for Event Identification 517 where the user name is other than the Security Manager. 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. Sensor provides the record for Event Identification 612, “Audit Policy Changed”.

Detection: Attempt to exploit default NT user accounts, Administrator and Guest.
Description: The Administrator and Guest accounts that come with the initial NT system have been renamed by the security policy. Any Login Failure events with these account names indicate an attempted intrusion.
Action: Monitor the Security Event Log entries for Event Identification 529 where the account name is any spelling variation of account “Administrator” or “Guest”. Sensor provides the record for Event Identification 529, “Login Failure”.

Detection: Multiple Login Failures suggesting intrusion attempt.
Description: Multiple login failure events in a short period of time indicate a possible attempted intrusion. More weight should be given to a remote failure than a local failure.
Action: Monitor the Security Event Log entries for Event Identification 529. Compare the number from a workstation over a period of time. Sensor provides the record for Event Identification 529, “Login Failure”.

Detection: Account lockouts suggesting intrusion attempt.
Description: Accounts are locked out when a defined number of unsuccessful login attempts occur. Attempts to login after lockout will result in additional lockout events. May indicate a possible attempted intrusion, but false alarms are possible.
Action: Monitor the Security Event Log entries for Event Identification 539 and Event Identification 644. 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.

Description: User accounts are changed. May indicate a misuse, but false alarms are possible.
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”.

The following is an example output file, in the format described earlier, from an early test run of one complete pass of the Log Sensor through a security event log. The log was interspersed with records of the type the program was capable of detecting and was created specifically for this test. All records of the types checked for were correctly identified.

8/31/99,17:43:46,612,8,6
8/31/99,16:52:30,624,8,7,test_u,D871WS01,admin
8/31/99,16:52:17,630,8,7,test_u,D871WS01,admin
8/31/99,16:50:42,612,8,6
8/31/99,16:49:19,624,8,7,test_user,D871WS01,admin
8/31/99,16:47:46,630,8,7,Test_acct_lockout,D871WS01,admin
8/31/99,16:47:24,612,8,6
8/31/99,15:14:52,529,16,2,admin,D871WS01,2,D871WS01
8/31/99,15:11:24,529,16,2,kremer,D871WS01,2,D871WS01
8/31/99,15:11:14,529,16,2,kremer,D871WS01,2,D871WS01
8/31/99,14:54:18,612,8,6
8/31/99,14:53:11,612,8,6
8/31/99,14:44:42,612,8,6
8/31/99,13:48:53,529,16,2,Administrator,D871WS01,2,D871WS01
8/31/99,13:48:42,529,16,2,Guest,D871WS01,2,D871WS01
8/31/99,13:28:3,539,16,2,Test_acct_lockout,D871WS01,2,D871WS01
8/31/99,13:28:1,529,16,2,Test_acct_lockout,D871WS01,2,D871WS01
8/31/99,13:28:1,644,8,7,Test_acct_lockout,D871WS01
8/31/99,13:27:58,529,16,2,Test_acct_lockout,D871WS01,2,D871WS01
8/30/99,8:20:50,529,16,2,ADMINTRJ,SPAWAR SSC ,3,\\FINA
8/26/99,15:18:52,529,16,2,admin,ANDROMEDA,2,D871WS01
8/26/99,15:18:39,529,16,2,admin,ANDROMEDA,2,D871WS01
8/26/99,15:18:27,529,16,2,admin,ANDROMEDA,2,D871WS01
8/26/99,14:57:46,517,8,1

The Audit Log Sensor was then executed in a networked environment as described below, writing its output record written to an output file rather than to a pipe because we did not have the time to debug the implementation of the pipe to the Controller.

5.2 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. Figure 7 shows the results.

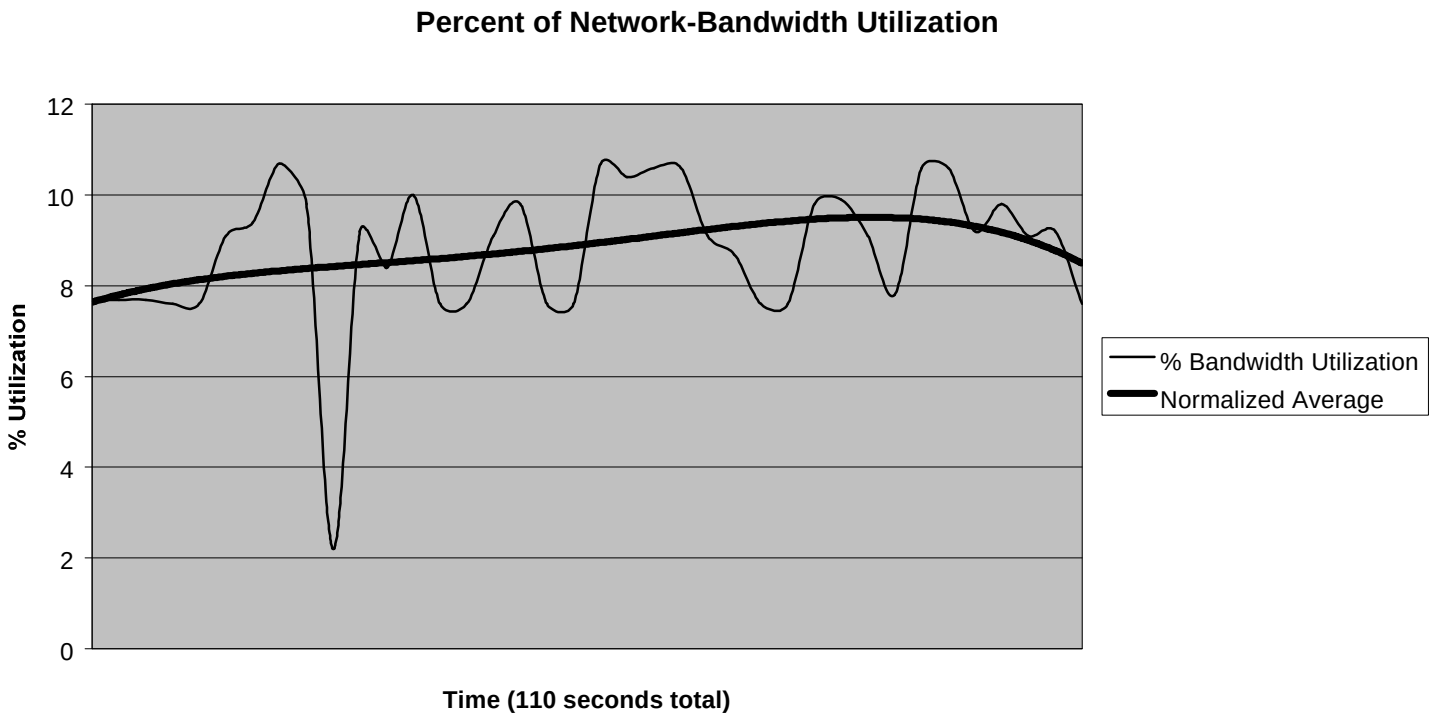

Figure 7: Network Bandwidth Utilization

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.

5.*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 in section 3. 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. Figure 8 shows the expected login alerts and the acceptable login failure rate for other numbers of users.

Users	Logins /Day	Locks/Day	Mapped Drives	Estimated Alerts	Acceptable Failure Rate
10	3	4	2	130	19.5
20	3	4	2	260	39
30	3	4	2	390	58.5
40	3	4	2	520	78
50	3	4	2	650	97.5
100	3	4	2	1300	195

Figure 8: Estimated Login Alerts

Using the same network sniffer as in earlier testing, 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 9 and 10 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 10, 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 consumption in an open network. During the one-hour time 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 remaining 92% were from normal network activity.

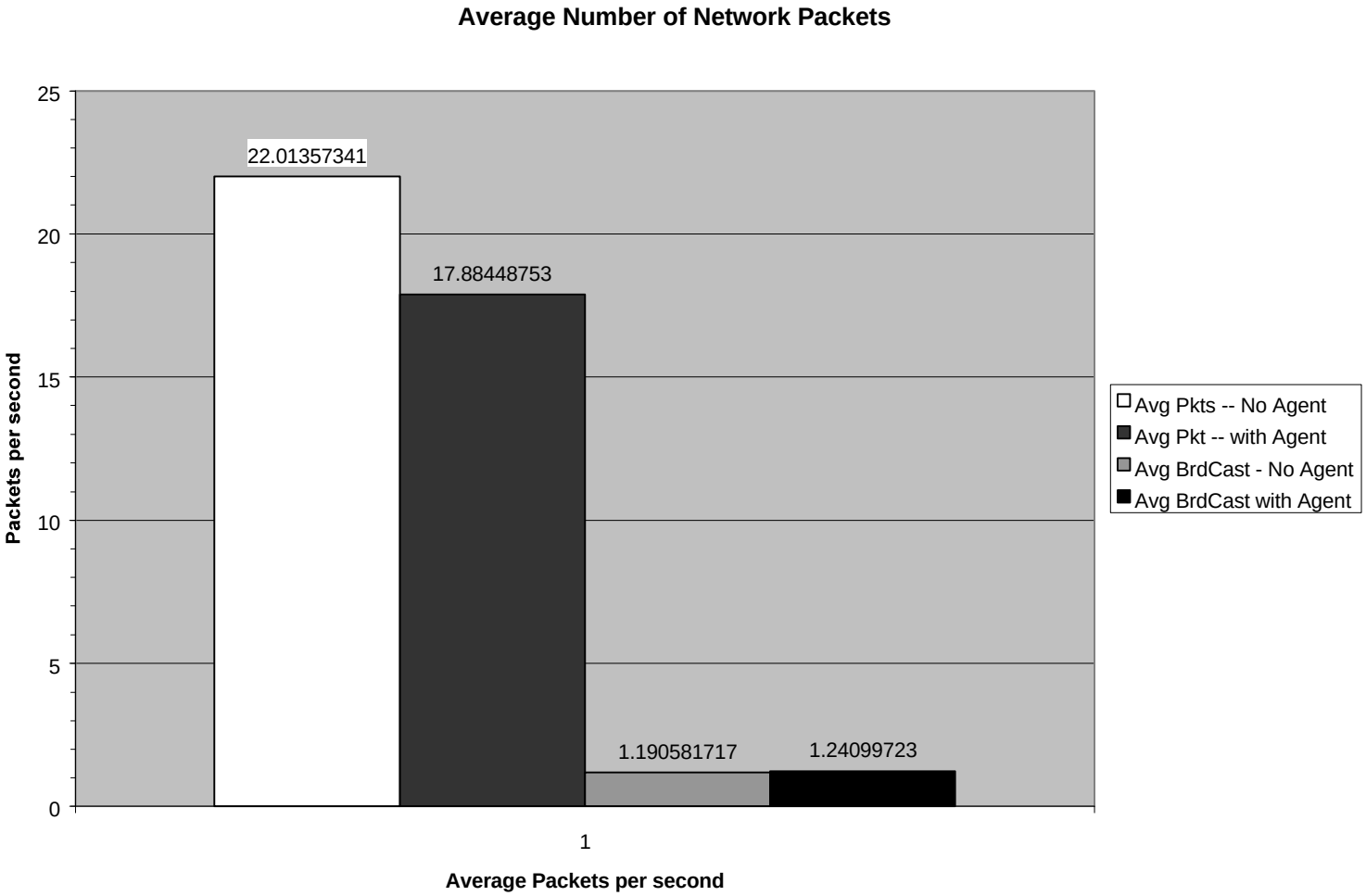

Figure 9: Average Packets and Broadcasts

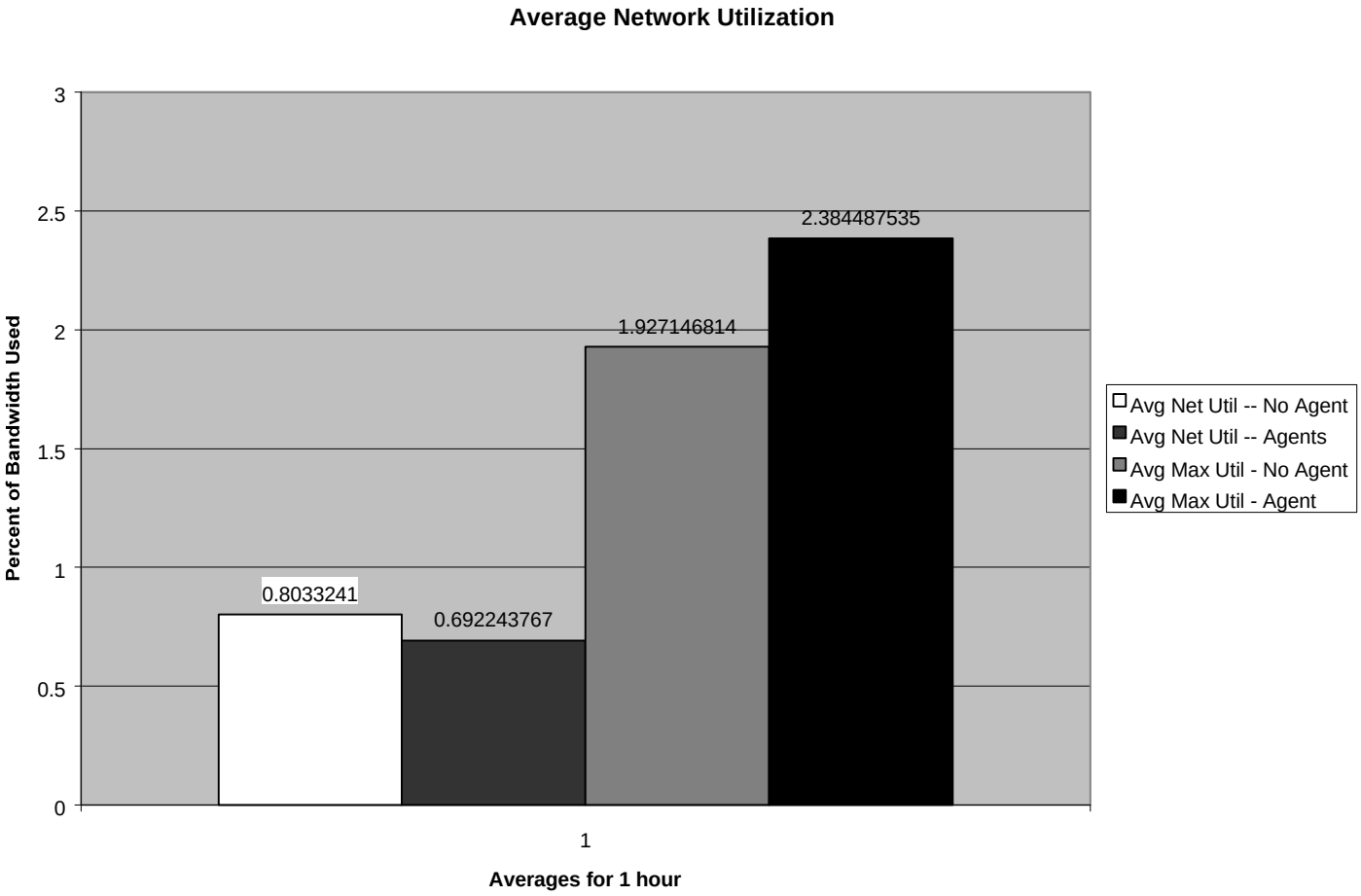

Figure 10: Percent Network Bandwidth Utilization

5.4 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. Figure 11 shows that the maximum CPU utilization of 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.

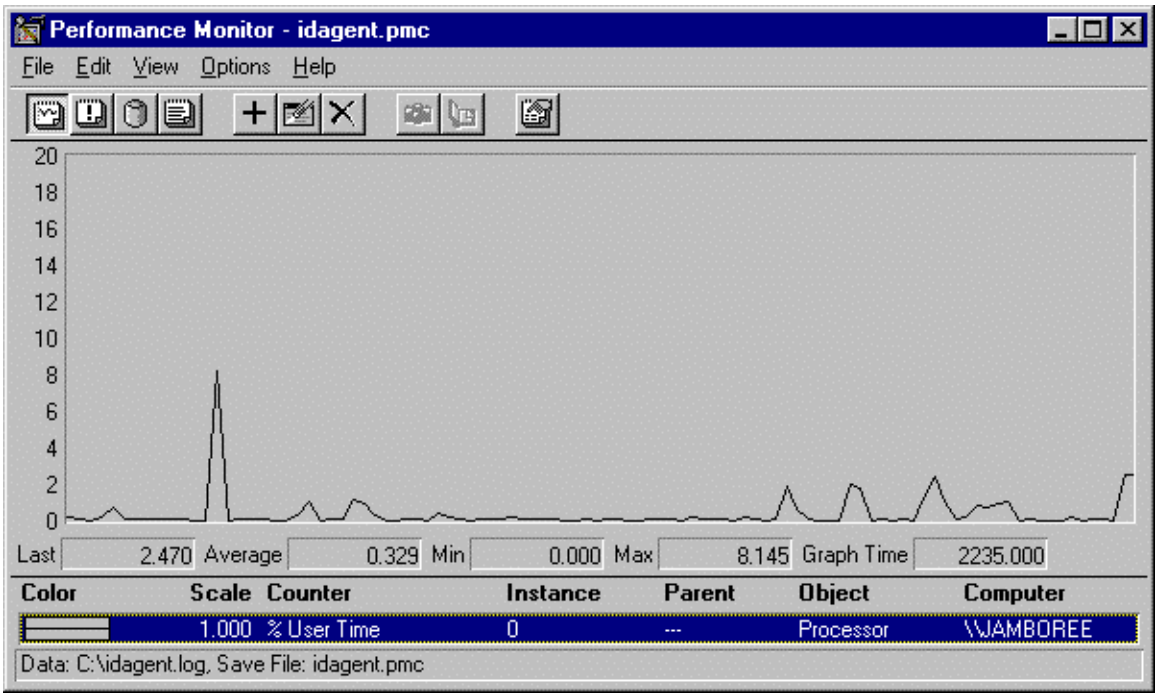

Figure 11: CPU Utilization

5.5 Simulated attack scenarios

Several scenarios were used to test the reaction of the IDAgent. Three computers were used. In the first scenario, all three 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. Table 3 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 (See section 3 for calculation details). 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 12: 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 13 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 13: 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.

6. 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.

6.1 Fulfillment of system requirements

We can assess our system in terms of the ten basic requirements for a good intrusion detection system listed in section 1.

The System Must Recognize Suspect Activity of a Potential Attack: The prototype system could effectively recognize failed logins, both on a single host and across distributed hosts. To recognize other types of activity, sensors would have to be written. The modular design of the IDAgent allows the straightforward integration of new sensors.

Escalating Behavior Should Be Detected at the Lowest Level Possible: The requirement to detect an intruder at the lowest level possible is very subjective. Triggering an alert the instant a failed login occurs would generate a large number of false positive alerts; waiting until an attack is absolutely certain might be too late. The threshold values in the IDAgent allow the level of detection to be adjusted to meet requirements. We believe our IDAgent detected login attacks at an appropriate level.

*There Must Be Interhost Communication Regarding Intrusions and Alert Level*s: The IDAgent program was designed specifically to meet this requirement. Its transmitter and receiver components are the means of communication, and the Message Class data structure carries the information between hosts.

There Must Be Appropriate Response to Changing Alert Level: This requirement was not implemented in the current configuration of the IDAgent.

The System Must Incorporate Manual Control Mechanisms for Administrators: The user interface for the IDAgent includes some control for debugging and determining the status of the agent. There are no controls for resetting thresholds or other parameters, but they could easily be added.

The System Must Be Adaptable to Changing Methods of Attack: This requirement was only partially met because only login sensors were written. Multiple sensors would be needed to detect changing attack methods.

The System Must Be Able to Handle Multiple Concurrent Attack Threads: IDAgent is a multi-threaded application that is capable of detecting multiple attack scenarios. If multiple login attacks were taking place, the IDAgent should be able to detect all suspicious activity.

The System Must Be Scalable and Easily Expandable: This requirement is fully met by IDAgent. To scale to a large network, you simply start agents on the added hosts. Expandability is allowed through the modular design of the agent.

The System Must Be Resistant to Compromise and Able to Protect Itself from Intrusion: This is left as future work. Java® provides many built-in security features, though none were incorporated yet.

The System Must Be Efficient and Reliable: Determination of efficiency was one of the primary goals of this work and has been adequately achieved in this prototype. Network bandwidth consumption and CPU utilization were both tested. The system was reliable under our limited testing.

6.2 Future work

Some of the following would provide for a more robust agent for future work and testing.

The current agent does not incorporate security or secure message handling to prevent blocking of messages or generation of false messages. Java® does provide built-in encryption mechanisms that could be used.

How does one determine if an agent that is responding is really a trusted agent or a piece of malicious software used by a hacker? Some form of authentication should be used to ensure security.

Running the IDAgent as an application under Java required a few work-arounds during testing. If the IDAgent was running and the user logged off, the IDAgent would terminate leaving no protection. The answer to this problem is to run IDAgent as an NT service. This was done successfully; however, the user interface cannot be seen or accessed making it difficult to monitor the agent. These problems would have to be overcome to successfully use the agent in a live network.

The version of Java used in this implementation is an interpreted language and as such runs much slower than an application written in a lower level language. Java was sufficient for prototyping and allowed rapid development of the communication portions of the agent. However, other languages should be researched.

There must be a response to an attack or intrusion to prevent entry. The system should be reactive.

The IDAgent tested here had limited sensor capability. It could detect user login attempts and when another agent was not responding. 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 threshold values in the agent were set based on our knowledge of network administration. Testing on a live network would allow the adjustment of the threshold values to better match the nature of the users in the network.

A configuration file that would allow an administrator to change parameters, variables, and threshold values without modification of the IDAgent would be a beneficial addition to the system.

The Log Sensor should be executed as a system service, started at system boot-up under NT System Services. It can be resident on all NT workstations and servers in the network. Since the security architecture and the auditing services are the same on servers and workstations, the Log Sensor can operate on both.

The Log Sensor currently recognizes only a small number of the events that can be audited, but it could be easily extended to handle more. Such events are set by the audit-policy settings which are mandated by the security policy. Programming to read the events and convert to a common format is needed, but the examples contained in the program should make that simple. A common audit format would allow data sharing amongst a variety of security tools.

The Log Sensor could filter more false alarms if desired. A graphical user interface allowing selective filtering options would be helpful. The sensor monitors the security log only, but it could be adapted to monitor the application and system logs with additional threads.

To ensure that the Log Sensor is not sabotaged or crashed, a message indicating the sensor is alive should be send periodically to the IDAgent. If the IDAgent does not get the expected message, an appropriate response should be generated. File access auditing should be set for the Log Sensor so that any attempt to delete or take ownership of the sensor would be recorded.

Archive backup and clearing of the event logs are maintenance functions that could easily be added to the Log Sensor.

7. 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.