Developing an SNMP Manager for UNIX System Security

by

Satian Chokdeepanich

Faculty of Engineering

December 2000

Developing an SNMP Manager for UNIX System Security

A thesis

submitted to the Faculty of Engineering

by

Satian Chokdeepanich

in partial fulfillment of the requirements

for the degree of

Master of Engineering in Broadband Telecommunications

Advisor: Dr. Sudhiporn Patumtaewapibal

Assumption University

Bangkok, Thailand

December 2000

"DEVELOPING AN SNMP MANAGER FOR UNIX SYSTEM SECURITY"

By

Mr.Satian Chokdeepanich

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Engineering
Majoring in Broadband Telecommunications

Examination Committee:

1. Dr.Sudhiporn Patumtaewapibal (Advisor) .....
2. Dr.Kittiphan Techakittiroj (Member) .....
3. Dr.Nutawut Nupairoj (Member) .....
4. Dr.Pipat Hiranvanichakorn (MUA Representative) .....

Examined on: November 10, 1999
Approved for Graduation on: December 2, 2000

Faculty of Engineering
Assumption University
Bangkok Thailand
November / 1999

ABSTRACT

This thesis designs and implements a Simple Network Management Protocol (SNMP) manager for UNIX system to improve the security for unauthentication and unauthorized access.

By using the SNMP to communicate with UNIX server/agent and UNIX client/manager, the server manager can monitor and control the Management Information Base (MIB) of any logged files (e.g. authentication logged file , daemon logged file and kernel logged file) of an agent. Tkined, an Interactive Network Editor based on the Tcl/Tk programming language is used to contact the appropriate remote SNMP agent to retrieve the desired information. Tkined uses a Graphical User Interface (GUI) running on UNIX X-Windows to display the information. This manager interaction together with GUI forms a powerful management tool for network administrators.

ACKNOWLEDGEMENTS

I would like to acknowledge many people for their generous support during the preparation of this thesis, especially Dr.Sudhiporn Patumtaewapibal, my advisor, and Prof. Dr. Ir. Piet Demeester , co-advisor, for his continuos guidance, encouragement, support and for his constructive suggestions to the thesis.

I also would like to thank Dr. Pipat Hiranvanichakorn, Dr. Natawut Nupairoj and Dr. Kittiphan Techakittiroj for joining the committee and for their helpful suggestion.

I would like to thank Dr. Nick Marly, Dr. Kittiphan Techakittiroj, Stefaan Vanhastel, and my colleagues from the ATLANTIS NETWORK LABORATORY, Department of Information Technology, University of Gent, Belgium.

Special appreciation is due to my family for their continuous encouragement. Above all, I am forever grateful to my parents whose willingness to invert in my future has enabled me to achieve my educational goal.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION

This master thesis deals with Network Monitoring and Management using the Simple Network Management Protocol Version 1 (SNMPv1), which is an application level protocol on UDP/IP (User Datagram Protocol/ Internet Protocol). We develop an SNMP Manager Application to monitor and control the security of UNIX systems for unauthenticated and unauthorized access. Our SNMP Manager Application is implemented by using Tkined, an Interactive Network Editor based on Tcl/Tk programming language.

In chapter 2, we give an overview of the Simple Network Management Protocol Network-Management System and Network Management Application (NMA). The theory of Simple Network Management Protocol Version 1 (SNMPv1) is elaborated in chapter 3 where we explain the basic concept and protocol specification of SNMPv1. In chapter 4, we consider the auth Group managed-object in the Management Information Base (MIB) which contains the authentication information (authentication logged file) and the design of the SNMP Manager for monitoring and controlling the auth Group managed-object MIB of hosts running the appropriated remote agent server. Finally, in Appendix A, we explain the implemented programming language according to the manager desired function.

CHAPTER 2. SNMP NETWORK-MANAGEMENT SYSTEM

A network-management system is a collection of tools for network monitoring and control. A network-management system consists of additional hardware and software implemented among existing network components. A network-management system is designed to view the entire network as an unified architecture, with the unique address and labels assigned to each point and the specific attributes assigned to each element in the system. The active elements of the network provide regular feedback of status information to the network-control center. Details can be found in [1],[11].

2.1 Network-Management Configuration

Figure 1 depicts the architecture of a network-management system. Each network node contains a collection of software in the diagram as network-management entity (NME). Each NME performs the following network management tasks:

- Stored statistics locally
- Responds to commands from the network-control center, including commands to :
    - Transmit collected the statistics to the network-control center.
    - Change a parameter (e.g. a timer used in a transport protocol)
    - Provide status information (e.g. parameter values, active links)
    - Generate artificial traffic to perform a test

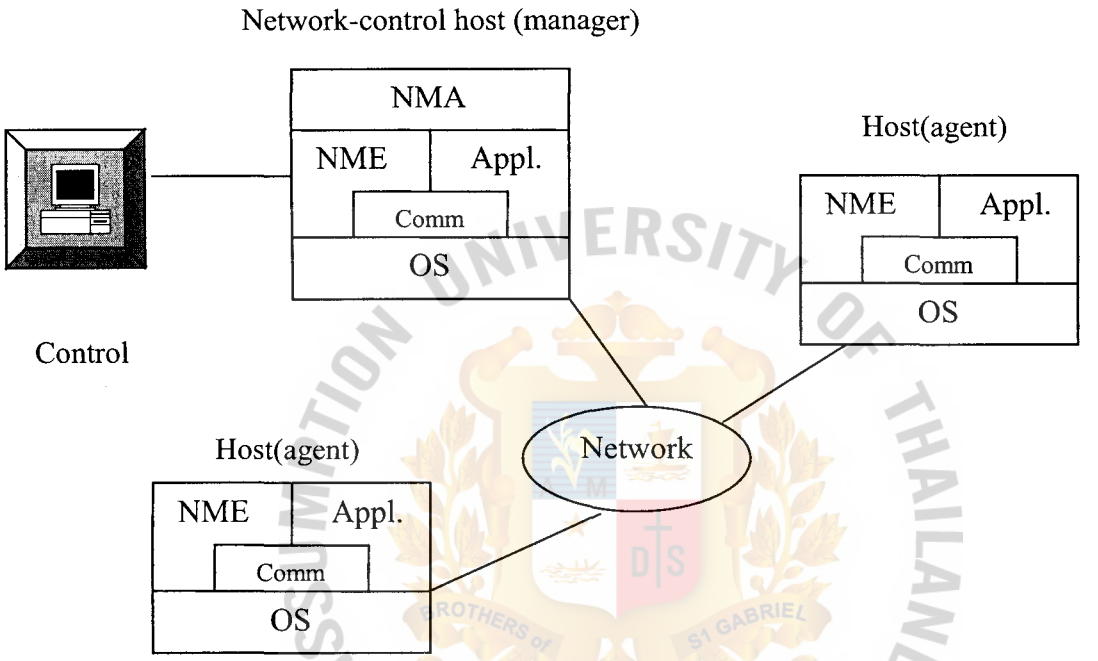
At least one host in the network is designated as the network-control host, or manager. The manager includes a collection of software called the network-management application (NMA). The NMA includes an operator interface to allow an authorized user to manage the network. The NMA responds to the user commands by displaying information and issuing commands to NMEs throughout the network. This communication is carried out using the application-level network-management protocol that employs the communication architecture in the same fashion as any other distributed application.

For an agent, each node in the network includes an NME which is for purposes of network management.

2.2 TCP/IP Network-Management Architecture

The model of network management that is used for TCP/IP (Transmission Control Protocol / Internet Protocol) network management includes the following key elements:

- Management station
- Management agent
- Management information base (MIB)
- Network-management protocol



NMA = network-management application
NME = network-management entity
Appl. = Applications
Comm = communication software
OS = Operating System

Figure 1 Elements of a Network-Management System

The management station is typically a stand-alone device but is capable to be implemented on a shared system. In either case, the management station serves as the interface for human network manager for the network-management system. The management station will have, at minimum:

- An interface by which network manager may monitor and control the network
- The capability of translating the network manager's requirements into the actual monitoring and control of remote elements in the network
- A database of information extracted from the MIBs of all the managed entities in the network

The other active element in the network-management system is the management agent. The management agent responds to request for information and requests for actions from the management station. It may asynchronously provide the management station with important and unsolicited information.

The means by which resources in the network may be managed is to represent these resources as objects. Each object is, especially, a data variable that represents one aspect of the management agent. The collection of objects is referred to as a management information base (MIB). The MIB functions as a collection of access points at the agent for the management station. A management station performs the monitoring function by retrieving the value of MIB objects. A management station can cause an action to take place at an agent or can change an agent's configuration settings by modifying the value of specific variables.

The management station and agents are linked by a network-management protocol. The protocol used for the management of TCP/IP networks is the Simple Network Management Protocol (SNMP), which includes the following key capabilities:

- Get : enables the management station to retrieve the value of objects at the agent
- Set : enables the management station to set the value of objects at the agent

- Trap : enables an agent to notify the management station of significant events

There are no specific guidelines in the standards as to the number of management stations per network. In general, it is prudent to have at least two systems capable of performing the management-station function, to provide redundancy in case of failure. As long as SNMP remains relatively "simple" that number can be quite high – certainly, in the hundreds.

CHAPTER 3. Simple Network Management Protocol Version 1 (SNMPv1)

The SNMP network management protocol is the application level protocol for allowing the Network Management Station (NMS) to get or read-retrieve and set or write-alter the managed objects in each agent's MIB. The SNMP also defines the Trap mechanism that allows an agent to transmit unsolicited alarm messages for certain predefined conditions. [3],[9],[10] and [11] contains more information of SNMP.

The network management stations and the agents communicate to each other by exchanging SNMP messages. The NMS and the agents receive all the SNMP Message except the Trap-PDU at well-defined UDP port 161 of the socket interface. The Trap-PDU SNMP Messages are received by the NMS at well-defined UDP port 162.

3.1 Authentication and Authorization

The SNMP administrative concerns involve the valid authentication and proper authorization of communicating between NMSs and agents. Authentication is the process of verifying the message sender's identity. Authorization is the verification of the access-level for a particular authenticated message received by the agent.

SNMP uses a trivial authentication mechanism. The key component of a trivial authentication scheme is the use of a community name. The community name is a string that represents the set of NMSs and agents that belong to a common and known group.

After an agent receives a command, it checks the community field. Then it does a string compare operation of the community name field in the received message against the community name string stored in its configuration. If they match, the message is

considered authentic by the protocol and is passed on for further processing. If the two strings are not successfully compared, the received message is discarded.

Once the SNMP message has been authenticated, the level of access then needs to be determined. Every member of the community knows which objects in the MIB can be accessed by the other members. This group of objects is referred to as the view. Two access modes for the objects are in the view: read-only and read-write.

3.2 The SNMP Message

The protocol specifies the command and response messages that can be used for the various dialogues between the NMS and its agents. Figure 2 shows the message and its fields. Details of protocol stack can be found in [11].

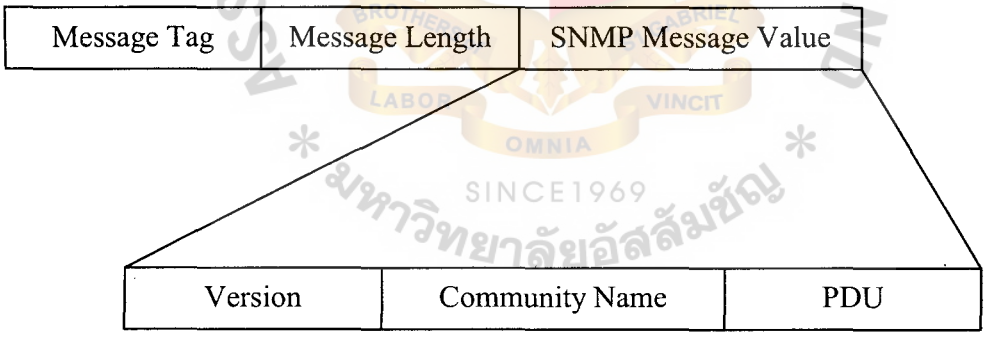


Figure 2 The SNMPv1 Message

The message field contains three required sub-fields ;

- The Version Field
- The SNMP Community Name Field
- The Protocol Data Unit (PDU)

SNMP version checking is a simple mechanism that does not allow for any negotiation. If the version field of a received message is incorrect, the message is discarded.

The Community Name Field is the octet string that contains the community name used in the authentication process. An agent contains a list of valid community strings to compare with community string of the received SNMP message. If a match occurs, the message is processed. If no match occurs, the message is discarded.

SNMPv1 specifies that the Protocol Data Unit must be one of five supported types:

- GetRequest-PDU
- GetNextRequest-PDU
- GetResponse-PDU
- SetRequest-PDU
- Trap-PDU

Figure 3 shows the common structure for imperative commands (get, get-next, and set) and the response command (get-response) and the four fields that each contains:

- The Request ID Field
- The Error Status Field
- The Error Index Field
- The Variable Bindings List

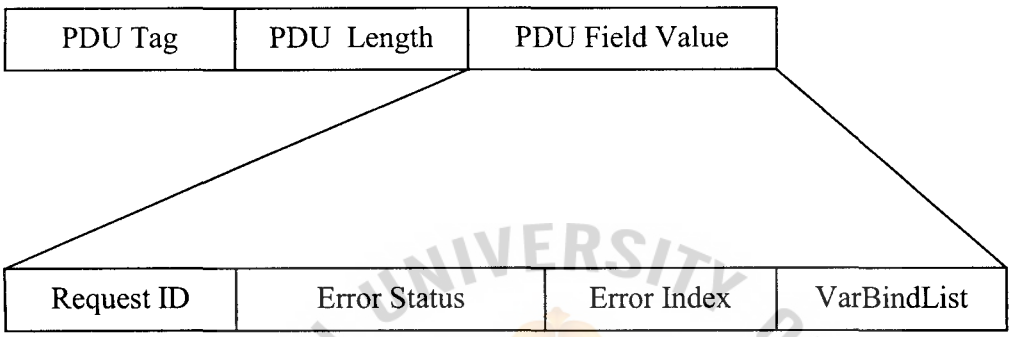


Figure 3 Command PDU Format

The Request ID Field is an integer that numbers the requests sent from the NMS to the agent. This field is necessary for matching a subsequently received GetResponse-PDU from the agent. The onus of reliable SNMP message sequencing is solely on the management side. The RequestID field can handle the error condition of duplicate responses where two recently received responses have the same RequestID value, and the NMS can use a timer facility to guage non-responses. An unknown RequestID would cause the NMS to discard the message.

The ErrorStatus Field is only used by the agent in sending a GetResponse-PDU. It indicates the status of the previous command. If it is non-zero, an error condition has occurred. In all other PDUs this field should have the value of zero.

Table 1 shows the six values for the ErrorStatus Field.

Table 1 Error Status Field

Status Name	Status Value
NoError	0
TooBig	1
NoSuchName	2
BadValue	3
ReadOnly	4
GenError	5

The ErrorIndex Field is used in the GetResponse-PDU to provide more information on error conditions detected by the agent. The ErrorIndex points to the first variable in the Variable Bindings List that caused the error condition. Its value is the position of the variable within the list, starting at one.

The variable Binding List, often abbreviated as VarBindList, is the list of instance of the managed objects that are operated on by the message's command.

Figure 4 shows the fields and the positions of the elements within the Variable Bindings List.

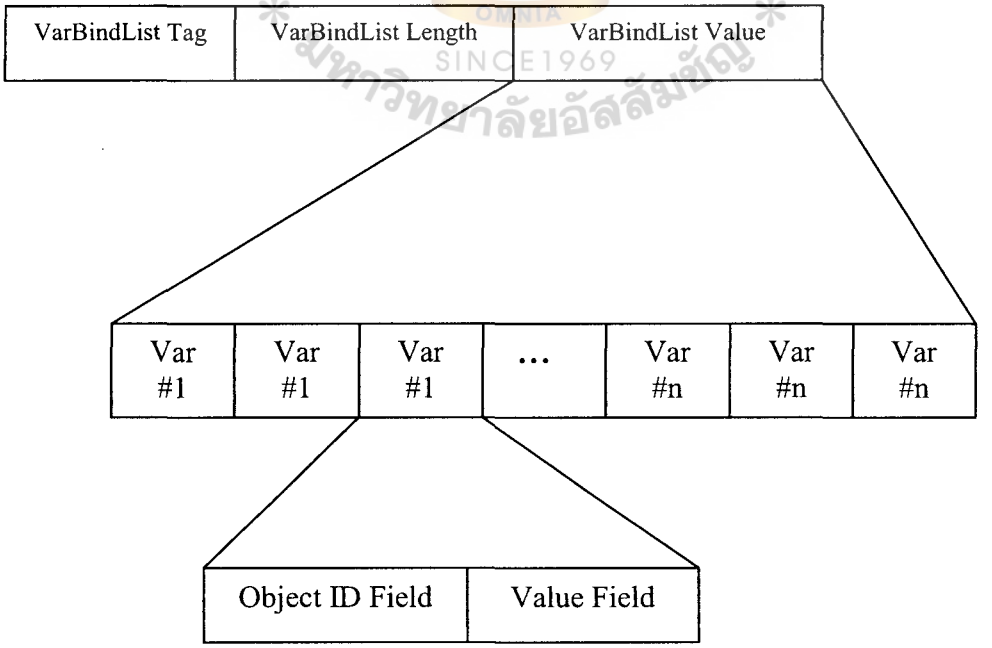


Figure 4 Variable Binding List Format

The Trap-PDU has a different structure from the former PDU tpes. The Trap-PDU has six mandatory fields for its format;

- The Enterprise Field
- The Agent Address Field
- The Generic Trap Field
- The Specific Trap Field
- The Time Stamp Field
- The Variable Binding List

Figure 5 shows the six fields of Trap-PDU used in SNMPv1.

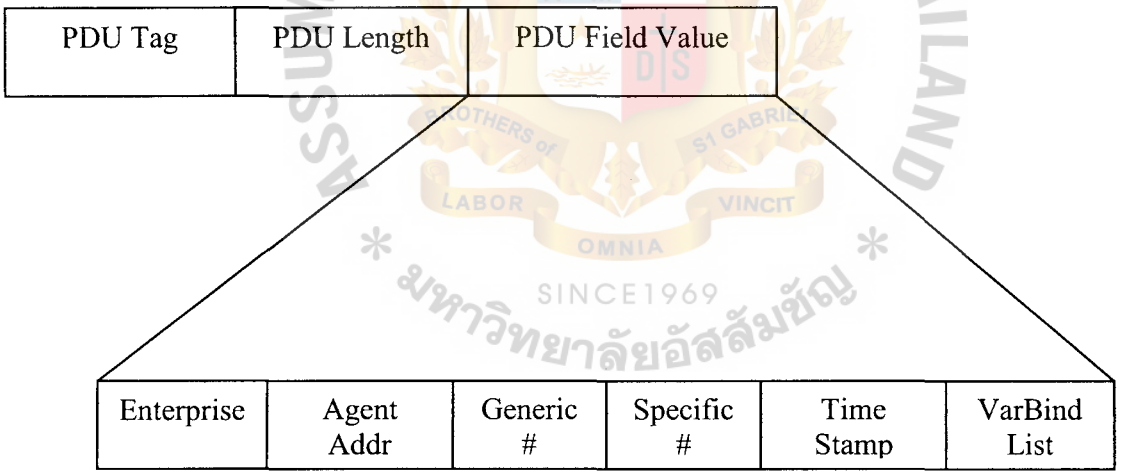


Figure 5. Trap-PDU Format

The Enterprise Field contains the Object Identifier for the network device generating the Trap.

The Agent Address Field is the agent's IP address. This further identifies the trap sender to the NMS that must receive and process the trap.

The Generic Trap Field contains an integer value that represents one of the standard predefined traps for SNMP. [3] defines seven generic traps:

- coldStart
- warmStart
- linkDown
- linkUp
- authenticationFailure
- egpNeighborLoss
- enterpriseSpecific

The Specific Trap Field contains that trap value defined for a particular enterprise. The values in the Specific Trap field are implementation dependent and are defined in the vendor's MIB that is implemented for that device.

The Time Stamp Field contains the "time" the trap was generated. This value is the number of time ticks that have elapsed since the agent was initialized. This value is in units of hundredths of a second.

The Variable Bindings List contains supplemental implementation information when included in the Trap-PDU. The variable bindings field contains "interesting information" associated with each generic or specific trap. This field contains zero or more variable binding pairs, which each pair contains an object name and the object value. The significance of the Variable Bindings List is implementation-specific, and its contents would be unique to each generic and specific trap.

3.2.1 The GetRequest-PDU

The NMS uses the GetRequest-PDU to retrieve the specific values of the known managed objects from the target agent's MIB. The valid response to this command is the GetResponse-PDU from the agent with the current values for those instances of the successful retrieval.

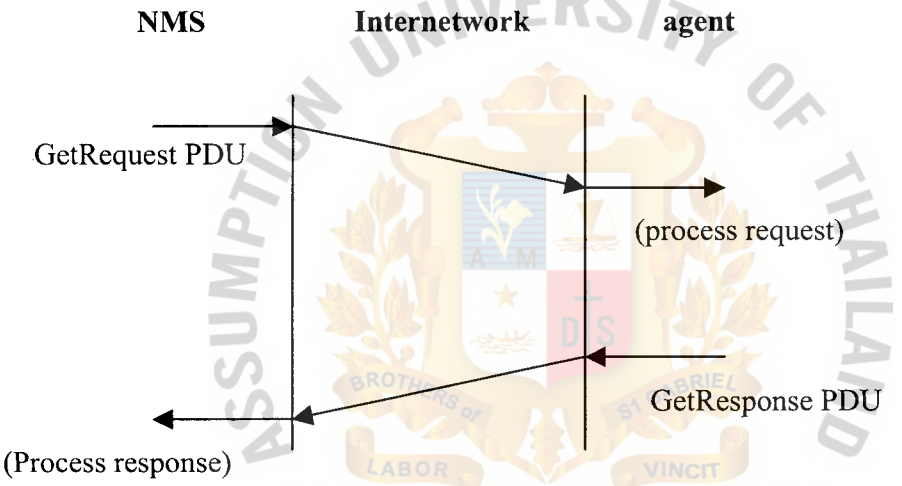


Figure 6. GetRequest Message

Figure 6 shows the NMS sending the GetRequest-PDU to the agent then processes the request and sends the GetResponse-PDU back to the NMS. After NMS processes the response, the query is complete.

First, the agent verifies that each object specified in the variable bindings list of the command exists. Each object's access mode must be compatible: it must be either read-only or read-write. Attempting to read a non-accessible object, such as the aggregate types of a table name or entry(row definition), is not valid. Another cause for an incorrect fetch is the instrumentation code that retrieves the value detects an internal failure.

The first incorrected object encountered causes the agent to construct the GetResponse-PDU with the appropriate errorStatus and errorIndex fields set. If the agent cannot find an object, the errorStatus is set to noSuchName. If the object is an aggregate type, the errorStatus is also set to noSuchName. And if the instrumentation code detects a failure, genErr is the errorStatus used. The errorIndex field is set to the offset of the first object that caused the error. Then the agent will process the following objects.

The agent also can detect the failure condition for the case when the construction of GetResponse-PDU results in a PDU of greater than maximum allowable SNMP message size. In this case the GetResponse-PDU is identical to the received GetRequest-PDU, except that the errorStatus is set to tooBig and errorIndex field is assigned a value of zero.

If all of the managed object values are successfully found and retrieved, and the resulting GetResponse-PDU is of acceptable size, the errorStatus field is assigned the value of noError and the errorIndex field is set to zero.

In every case that the GetResponse-PDU is sent, the request ID field is set to the value retrieved from the corresponding previously received GetRequest-PDU. The message is then sent back to the NMS that originated the command.

3.2.2 The GetNextRequest-PDU

Figure 7 shows the NMS sending the GetNextRequest-PDU across the network and having the agent process it. The agent then sends the GetResponse-PDU that the NMS receives and processes to complete the query.

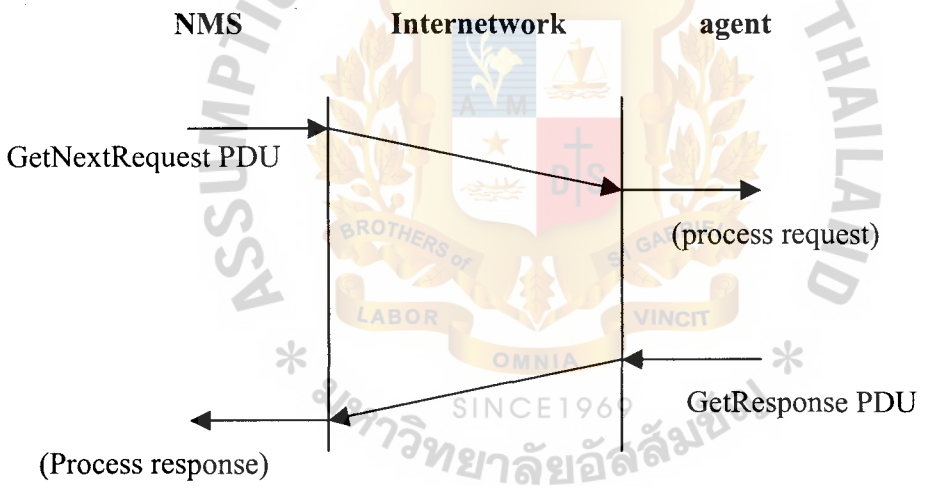


Figure 7. GetNextRequest Message

The GetNextRequest command is similar to GetRequest, except that the agent attempts to retrieve the lexicographically next larger value than the managed object instance requested. The use of this command is primarily for tree traversal and determining the elements of a table not known beforehand.

The agent verifies that an object lexicographically larger than the next object specified exists in the variable bindings list. The next managed object must have the proper access mode ; read-only or read-write. If the instrumentation code that retrieves the value detects a failure, the agent records the error condition.

The first incorrected object encountered causes the agent to construct the GetResponse-PDU with the appropriate errorStatus and errorindex fields set. If the agent cannot find the next valid object, the errorStatus is set to noSuchName. If the instrumentation code detects a failure, the genErr errorStatus is used. The errorIndex field is set to the offset of the first object that caused the error. Then the agent will process the following objects.

The agent can also detect the failure condition for the case when the construction of GetResponse-PDU results in a PDU of greater than the maximum allowable SNMP message size. In this case the GetResponse-PDU is identical to the received GetNextRequest-PDU, except that the errorStatus is toobig and the errorIndex field is assigned a zero value.

If all the managed object values are successfully found and retrieved, and the resulting GetResponse-PDU is of acceptable size, the errorStatus field is assigned the value of noError and the errorIndex is set to zero. The variable bindings list of the GetResponse-PDU contains the variable name and value of the object representing the successor to each of the variable name requested. Each successor name and value represents the next object in the lexicographical ordering that has the proper access value for the particular MIB view.

In every case that the GetResponse-PDU is sent, the requestID field is set to the value found from the matching GetNextRequest-PDU received from the NMS. The message is then sent back to the NMS that originated the command.

3.2.3 The SetRequest-PDU

Figure 8 shows the NMS sending the SetRequest-PDU and having the agent process it. The agent then sends the GetResponse-PDU to the NMS who processes the returned information.

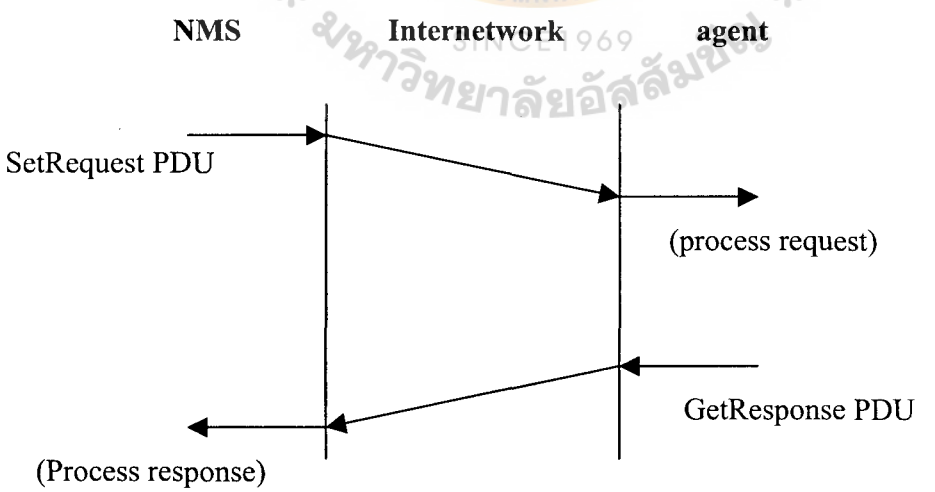


Figure 8 SetRequest Message

The SetRequest-PDU command is similar to the GetRequest-PDU, except that the agent attempts to set the value of the managed object instances specified instead of just reading it.

The agent verifies that each managed object exists and that it possesses a proper access mode of write-only or read-write. If the instrumentation code that sets the value detects a failure, the agent records the error condition.

The first incorrected object encountered causes the agent to construct the GetResponse-PDU with the appropriate errorStatus and errorIndex fields set. If the agent cannot find the object, the errorStatus is set to noSuchName. If the ASN.1 syntax was in error, the errorStatus of badValue is set in the response. If the instrumentation code detects a failure, the genErr errorStatus is used. The errorIndex field is set to the offset of the first object in the VarBindList that caused the error.

The agent can also detect the failure condition for the case when the construction of an otherwise successful GetResponse-PDU causes a PDU of greater than maximum allowable SNMP message size. In this case, the GetResponse-PDU is identical to the received SetRequest-PDU, except that the errorStatus is set to tooBig and the errorIndex field is assigned a value of zero.

If all the managed object values are successfully found and set, and the resulting GetResponse-PDU is of acceptable size, the erorStatus field is assigned the value of noError and the errorIndex field is set to zero. The variable bindings list of the GetResponse-PDU contains the variable name(s) and the new values for those objects.

3.2.4 The GetResponse-PDU

The agent sends the GetResponse-PDU after it has processed a GetRequest-PDU, a GetNextRequest-PDU, or a SetRequest-PDU. The NMS receiving the GetResponse-PDU needs to check the values of the fields in the message and then process the pertinent data.

The requestID field. It correlates the response to a previously sent command. Next, the errorField is checked to see if the command has been successful. If the error staus in this field is noError, the variable bindings list is processed. If the error status is non-zero, the error is noted and logged.

3.2.5 The Trap-PDU

Traps are asynchronous notifications an agent can send to the Network Management Station to inform the NMS of an extraordinary event. These extraordinary events are predefined in the MIB and must be known to both the agent and the NMS within the enterprise. Figure 9 shows the agent sending the Trap-PDU across the network to the NMS for processing.

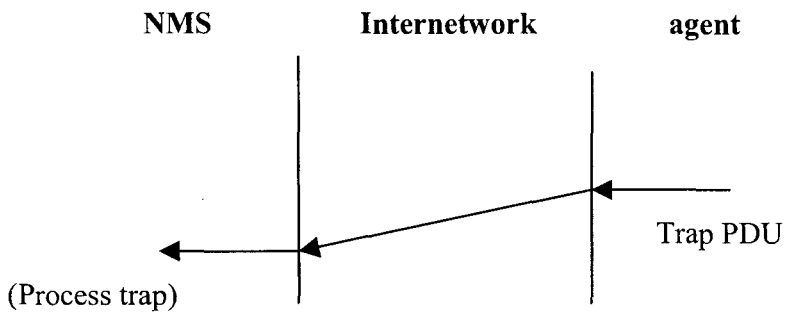


Figure 9. Trap Message

The generic trap field contains one of the traps predefined by SNMP. The seven traps defined are:

- cold Start (0) : This trap signifies that the agent's device, called the sending protocol entity, is reinitializing it self such that the agent's configuration or the protocol entity implementation may be altered.
- WarmStart (1) : This trap signifies that the agent's device is reinitializing itself such that neither the agent's configuration nor the protocol entity implementation is altered.
- LinkDown (2) : This trap message signals that the agent's device recognizes a failure in one of the communication links presented in its configuration. The Trap-PDU contains the name and value of the ifIndex instance for the affected interface as the first element of its variable bindings.
- LinkUp (3) : This trap message signals that the agent's device recognizes that one of the communication links presented in the agent's configuration has come up. The Trap-PDU contains the name and value of the ifIndex instance for the affected interface as the first element of its variable bindings.
- AuthenticationFailure (4) : This trap signifies that the agent is the addresses of a protocol message that is not properly authenticated. Although implementations of the SNMP must be capable of generating this trap, they must also be capable of suppressing the emission of such traps through an implementation-specific mechanism such as a configuration parameter.
- EgpNeighborLoss (5) : This trap signifies that an Exterior Gateway Protocol (EGP) neighbor for whom the agent's device was an EGP peer has been marked down and

the peer relationship no longer pertains. The first element in the variable bindings is the name and value of the egpNeighAddr instance for the affected neighbor.

- EnterpriseSpecific (6) : This trap signals that an agent has recognized that some predefined, enterprise-specific event has occurred. The specific-trap field identifies the particular trap that occurred.

CHAPTER 4. Network Management Application Design

This section gives the detailed design of our network management application (NMA) for the management station to manage the network enabling monitoring the security of UNIX system by using Tkined (Interactive Network Editor Based on Tcl/Tk). By running this application, we allowed the manager to monitor and control the Management Information Base (MIB) of auth group managed objects at the appropriated agent. In this application, the manager's screen will display all the nodes in the network. If the unauthenticated and unauthorized users try to access any nodes in the network, the manager will display that node by alarming or blinking it on the manager's screen. The manager can also display all the unauthenticated and unauthorized access' information. The "Tcl/Tk" code for implementing in this Network Management Application is given in Appendix A.

4.1 UNIX Security Managed Object

The manager will send the SNMP message including the IP address, the value of object identifier MIB and control information to the destined agent. For UNIX System, auth group MIB (as shown in Table 2) consists of the information about Authentication Time, Authentication System Name, Authentication Program Name and Authentication output. This information is concerning about the security of UNIX system [2], [5], [6] and [8].

4.2 Network Management Function

In our Network Management Application, we implemented a module is called "SNMP-Security" into Tools main menu in Tkined as shown in Figure 10.

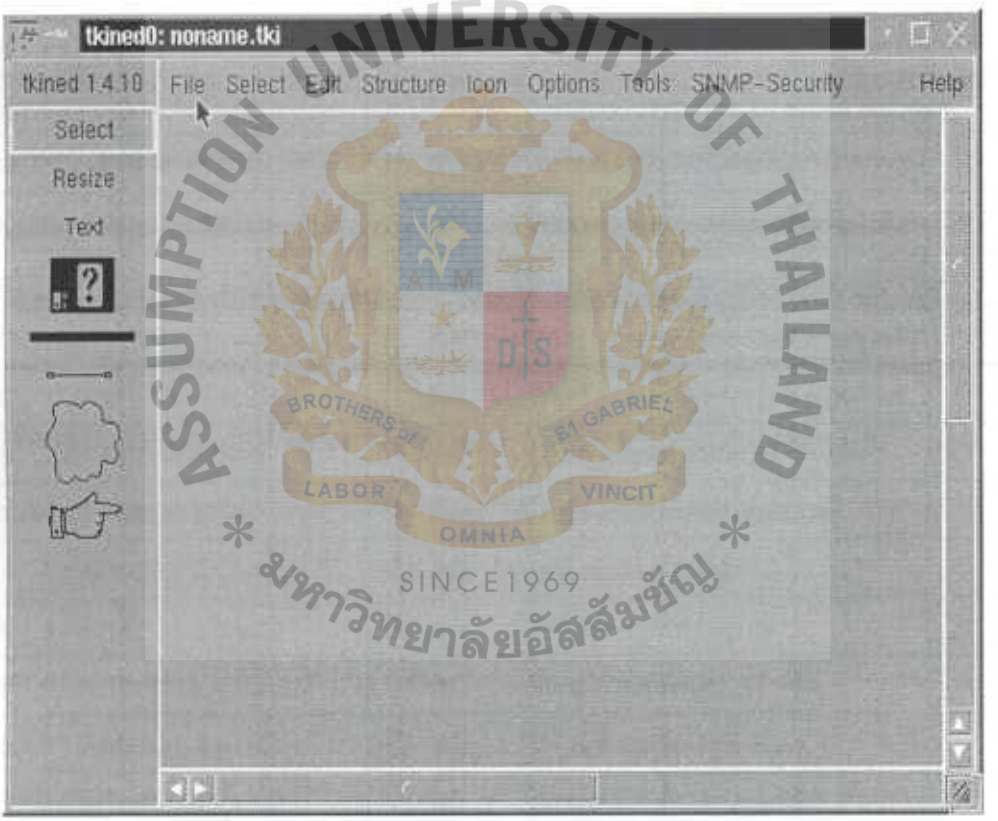


Figure 10. SNMP-Security Module

Table 2. auth Group MIB

Table 2. auth Group MIB

OBJECT	OBJECT IDENTIFIER	ACCESS	DESCRIPTION
index1	1.3.6.1.4.1.4210.10.1.1.1.1.0	RO	Index of Log File No. 1
filename1	1.3.6.1.4.1.4210.10.1.1.1.2.0	RO	File name of the log file
time1	1.3.6.1.4.1.4210.10.1.1.1.3.0	RO	Time of the event
sysname1	1.3.6.1.4.1.4210.10.1.1.1.4.0	RO	System name of the event
prog1	1.3.6.1.4.1.4210.10.1.1.1.5.0	RO	Program used to generate
message1	1.3.6.1.4.1.4210.10.1.1.1.6.0	RO	Message generated by above program
enable1	1.3.6.1.4.1.4210.10.1.1.2.1.0	RW	To enable the filtering rule
arg1t1	1.3.6.1.4.1.4210.10.1.1.2.2.0	RW	First argument used to filtering data
arg2t1	1.3.6.1.4.1.4210.10.1.1.2.3.0	RW	Second argument used to filtering data
arg3t1	1.3.6.1.4.1.4210.10.1.1.2.4.0	RW	Third argument used to filtering data
reset1	1.3.6.1.4.1.4210.10.1.1.2.5.0	RW	Used to reset the filtering argument
fileReset1	1.3.6.1.4.1.4210.10.1.1.3.0	RW	Used to reset all the file content
fileBackup1	1.3.6.1.4.1.4210.10.1.1.4.0	RW	Used to generate backup file
index2	1.3.6.1.4.1.4210.10.1.2.1.1.1.0	RO	Index of Log file No.2
filename2	1.3.6.1.4.1.4210.10.1.2.1.1.2.0	RO	File name of the log file
time2	1.3.6.1.4.1.4210.10.1.2.1.1.3.0	RO	Time of the event
sysname2	1.3.6.1.4.1.4210.10.1.2.1.1.4.0	RO	System name of the event
prog2	1.3.6.1.4.1.4210.10.1.2.1.1.5.0	RO	Program used to generate
message2	1.3.6.1.4.1.4210.10.1.2.1.1.6.0	RO	Message generated by above program
enable2	1.3.6.1.4.1.4210.10.1.2.2.1.0	RW	To enable the filtering rule
arg1t2	1.3.6.1.4.1.4210.10.1.2.2.2.0	RW	First argument used to filtering data
arg2t2	1.3.6.1.4.1.4210.10.1.2.2.3.0	RW	Second argument used to filtering data
arg3t2	1.3.6.1.4.1.4210.10.1.2.2.4.0	RW	Third argument used to filtering data
reset2	1.3.6.1.4.1.4210.10.1.2.2.5.0	RW	Used to reset the filtering argument

Table 2. auth Group MIB

OBJECT	OBJECT IDENTIFIER	ACCESS	DESCRIPTION
fileReset2	1.3.6.1.4.1.4210.10.1.2.3.0	RW	Used to reset all the file content
fileBackup2	1.3.6.1.4.1.4210.10.1.2.4.0	RW	Used to generate backup file
index3	1.3.6.1.4.1.4210.10.1.3.1.1.1.0	RO	Index of Log file No.2
filename3	1.3.6.1.4.1.4210.10.1.3.1.1.2.0	RO	File name of the log file
time3	1.3.6.1.4.1.4210.10.1.3.1.1.3.0	RO	Time of the event
sysname3	1.3.6.1.4.1.4210.10.1.3.1.1.4.0	RO	System name of the event
prog3	1.3.6.1.4.1.4210.10.1.3.1.1.5.0	RO	Program used to generate
message3	1.3.6.1.4.1.4210.10.1.3.1.1.6.0	RO	Message generated by above program
enable3	1.3.6.1.4.1.4210.10.1.3.2.1.0	RW	To enable the filtering rule
arg1t3	1.3.6.1.4.1.4210.10.1.3.2.2.0	RW	First argument used to filtering data
arg2t3	1.3.6.1.4.1.4210.10.1.3.2.3.0	RW	Second argument used to filtering data
arg3t3	1.3.6.1.4.1.4210.10.1.3.2.4.0	RW	Third argument used to filtering data
reset3	1.3.6.1.4.1.4210.10.1.3.2.5.0	RW	Used to reset the filtering argument
fileReset3	1.3.6.1.4.1.4210.10.1.3.3.0	RW	Used to reset all the file content
fileBackup3	1.3.6.1.4.1.4210.10.1.3.4.0	RW	Used to generate backup file
codeNo	1.3.6.1.4.1.4210.10.1.50.1.1.1.0	RO	Number of mapping filename
filename	1.3.6.1.4.1.4210.10.1.50.1.1.2.0	RO	File name of mapping number
fileCode	1.3.6.1.4.1.4210.10.1.50.2.1.0	RW	File name of the daemon rule
syn1	1.3.6.1.4.1.4210.10.1.50.2.2.0	RW	Argument no.1 of the input to the daemon
syn2	1.3.6.1.4.1.4210.10.1.50.2.3.0	RW	Argument no.2 of the input to the daemon
syn3	1.3.6.1.4.1.4210.10.1.50.2.4.0	RW	Argument no.3 of the input to the daemon

Table 2. auth Group MIB

OBJECT	OBJECT IDENTIFIER	ACCESS	DESCRIPTION
interval	1.3.6.1.4.1.4210.10.1.50.2.5.0	RW	Interval used to check
create	1.3.6.1.4.1.4210.10.1.50.2.6.0	RW	Flag used to set to create a new daemon
			process
psIndex	1.3.6.1.4.1.4210.10.1.50.3.1.1.1.0	RO	Index of running Daemon Process
psPid	1.3.6.1.4.1.4210.10.4.50.3.1.1.2.0	RO	Process ID of the running Daemon
psTime	1.3.6.1.4.1.4210.10.1.50.3.1.1.3.0	RO	Time interval for repeated check of the
			daemon
psFilename	1.3.6.1.4.1.4210.10.1.50.3.1.1.4.0	RO	File name used to check by daemon proc.
psSyn1	1.3.6.1.4.1.4210.10.1.50.3.1.1.2.0	RO	Argument no.1 of the filtering rule
psSyn2	1.3.6.1.4.1.4210.10.1.50.3.1.1.6.0	RO	Argument no.2 of the filtering rule
psSyn3	1.3.6.1.4.1.4210.10.1.50.3.1.1.7.0	RO	Argument no.3 of the filtering rule
killProcess	1.3.6.1.4.1.4210.10.1.50.3.2.0	RW	Indicate Process ID of the process which
			we want to terminated
killAll	1.3.6.1.4.1.4210.10.1.50.3.3.0	RW	Set to 1 to kill all daemon process

Object	OID	Access	Description
authIndex	1.3.6.1.4.1.4210.10.1.1.1.1.1.0	RO	Row number of the log file
authTime	1.3.6.1.4.1.4210.10.1.1.1.1.2.0	RO	Time of the generated EVENT of the AUTH log file
authSysname	1.3.6.1.4.1.4210.10.1.1.1.1.3.0	RO	The System name of the EVENT
authProg	1.3.6.1.4.1.4210.10.1.1.1.1.4.0	RO	The program name used to generate the EVENT
authOutput	1.3.6.1.4.1.4210.10.1.1.1.1.5.0	RO	The output generated by the Authen program.
authLastread	1.3.6.1.4.1.4210.10.1.1.2.0	RW	The number of ROW(index) that manager read for the last access used to index the next used EVENT for the filtering result.
authTotal	1.3.6.1.4.1.4210.10.1.1.3.0	RO	The total line of the auth_log file
authReset	1.3.6.1.4.1.4210.10.1.1.4.0	RW	Set to '1' : to reset the auth_log file, calling the function to delete all the content of the system specific auth_log file.
logFilename	1.3.6.1.4.1.4210.10.1.100.1.0	RW	Numerical substitution display used log file name. '1' for Authentication Logfile '2' for others
syntax 1	1.3.6.1.4.1.4210.10.1.100.2.0	RW	The first syntax used to filtering concerning log file.
syntax 2	1.3.6.1.4.1.4210.10.1.100.3.0	RW	The second syntax used to filtering the concerning log file .
andflag	1.3.6.1.4.1.4210.10.1.100.4.0	RW	Set to '1' : filtering rule = syntax 1 AND syntax 2 Set to '2' : filtering rule = syntax 1 OR syntax 2
updateflag	1.3.6.1.4.1.4210.10.1.100.5.0	RW	Set to '1' : used Lastread flag to indicate the starting point for filtering Set to ' 0 ' : start from the beginning of the file.
filterreset	1.3.6.1.4.1.4210.10.1.100.6.0	RW	Set to ' 1' : reset the filtering content to default.
resultIndex	1.3.6.1.4.1.4210.10.1.101.1.1.1.0	RO	The row number (index) of the filtering result event.
resultTime	1.3.6.1.4.1.4210.10.1.101.1.1.2.0	RO	The time of the filtering result event.
resultSysname	1.3.6.1.4.1.4210.10.1.101.1.1.3.0	RO	The system name of the filtering result event.
resultProg	1.3.6.1.4.1.4210.10.1.101.1.1.4.0	RO	The program used of the filtering result event.
resultOutput	1.3.6.1.4.1.4210.10.1.101.1.1.5.0	RO	The output of the filtering result event.
resultUpdate	1.3.6.1.4.1.4210.10.1.101.2.0	RW	Set to '1' : to update tha data concerning with log file with filtering syntax applied.

In SNMP-Security Module, before starting this module the manager has to verify himself by enter the passphase. The passphase is used to open the manager's private as shown in figure 11.

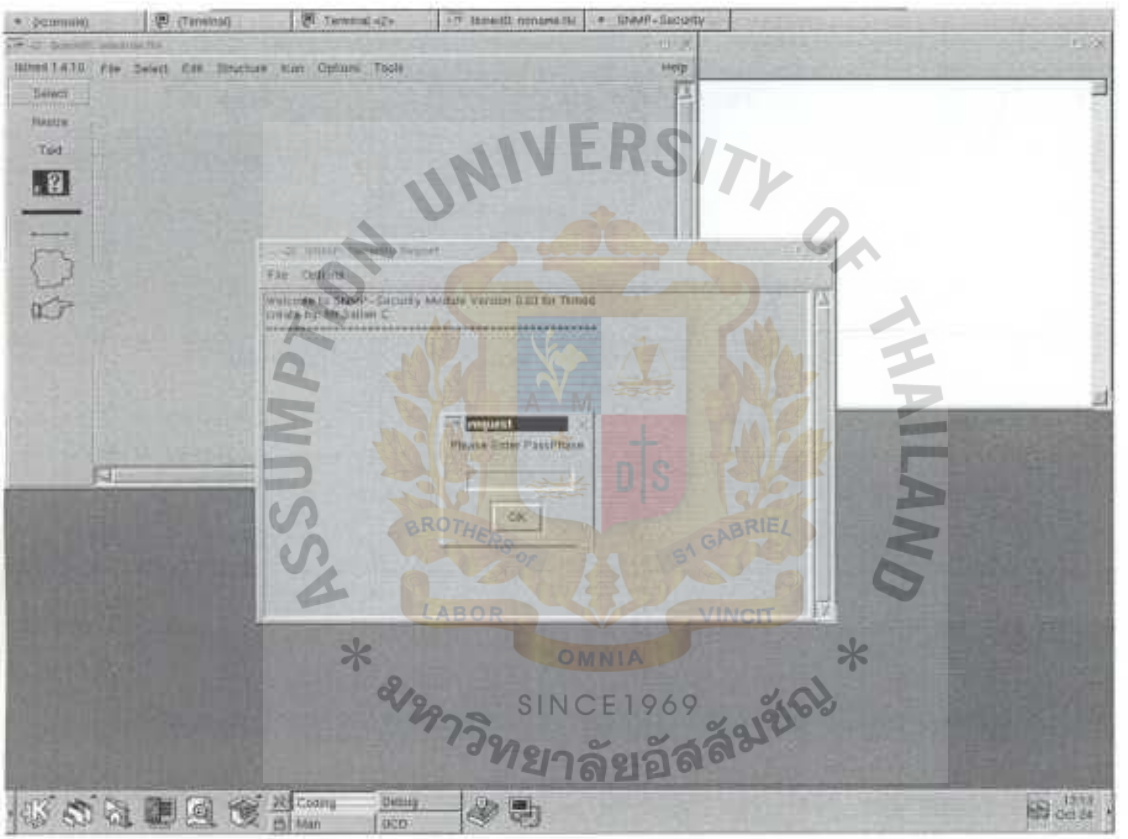


Figure 11 SNMP-Security Module

In SNMP-Security Module, we design our system to monitor and control the managed object at the destination. There are ten functions in this group as shown in figure 12:

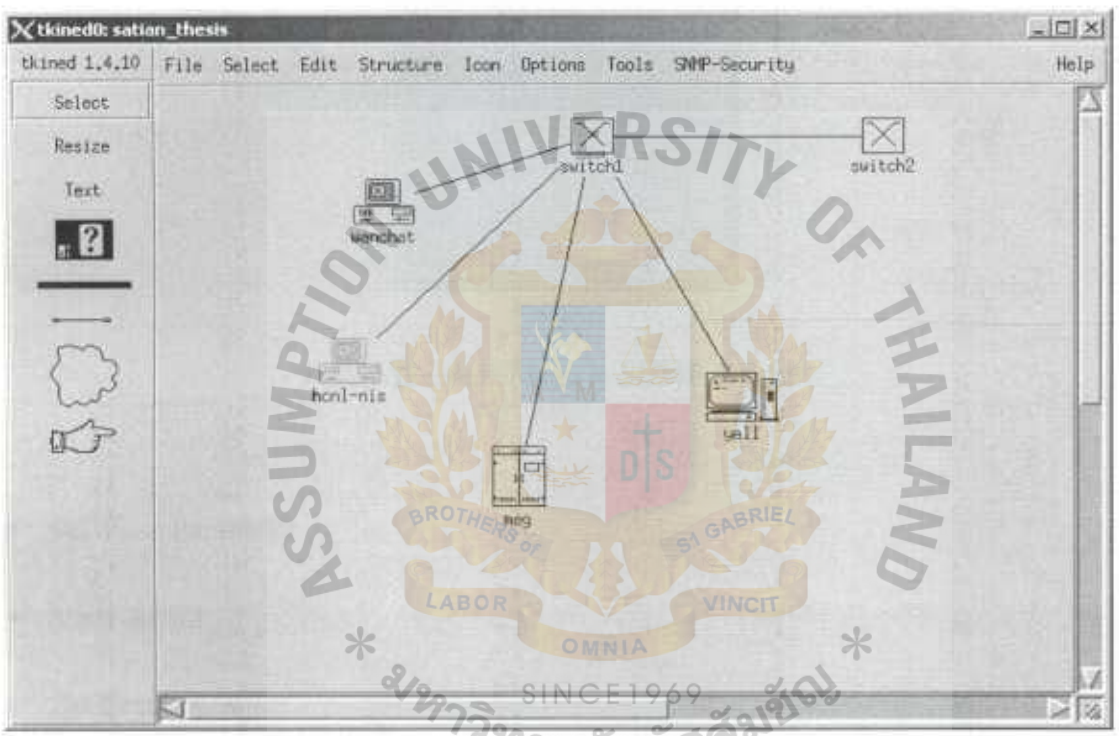


Figure 12 Main Menu in SNMP-Security

- Set View Parameter
- View Result
- Set Event Monitor
- List Event Monitor
- Kill Event Monitor
- Reset Log File
- Check View
- Event Color Setting
- Set SNMP/PGP Parameter

- Help SNMP – Security
- Delete SNMP – Security

4.2.1 Set View Parameter

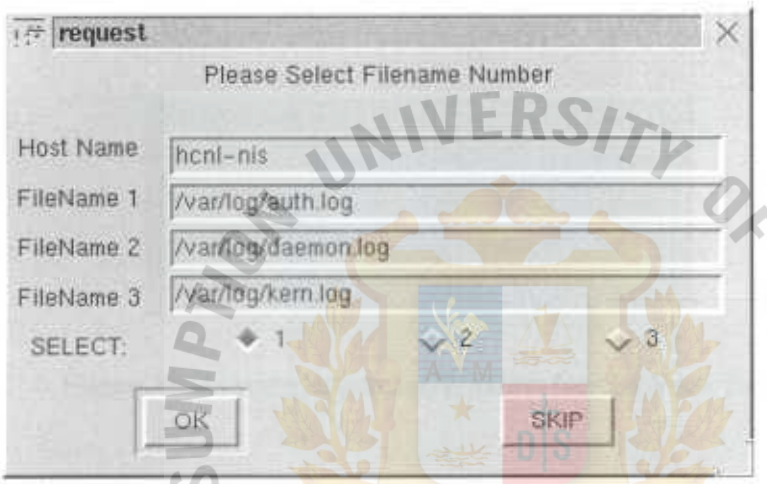


Figure 13 Sub Menu in Set View Parameter

The manager must select the file name (auth.log , daemon.log and kern.log) that would like to get the information as shown in figure14.

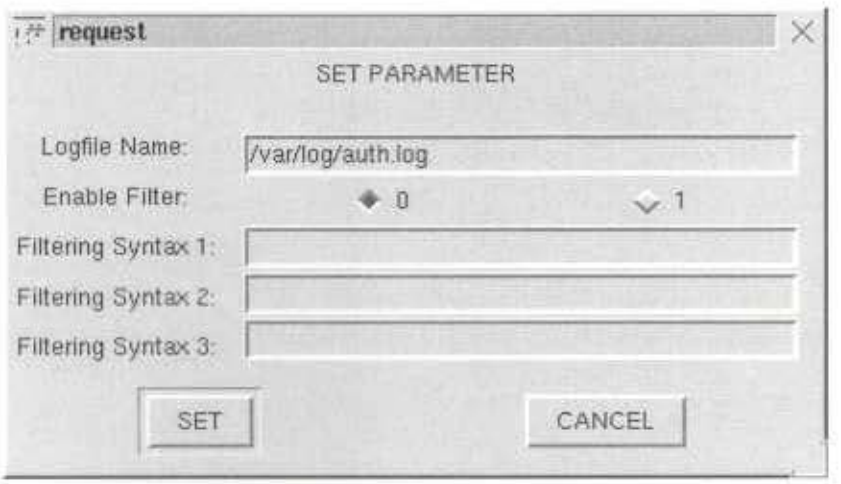


Figure 14 Sub Menu in Set View Parameter

The manager need to select to use the filter operation, if need please select 1and 0 for unused. .Then the manager can fill the filtering word in Filtering Syntax 1 or 2 or 3 as shown in the figure 14. Finally, the acknowledgement window will display for acknowledge the set view parameter menu as shown in figure no. 15.

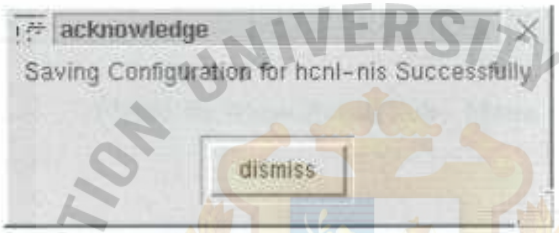


Figure 15 Acknowledgement Window for Set View Parameter

4.2.2 View Result

This sub-menu is used for displaying the result event according to Set View Parameter Setting. Before getting the result display, the manager has to select the Filename Number which to be displayed as shown in the figure 16. The view result will be shown as in the figure no. 17.

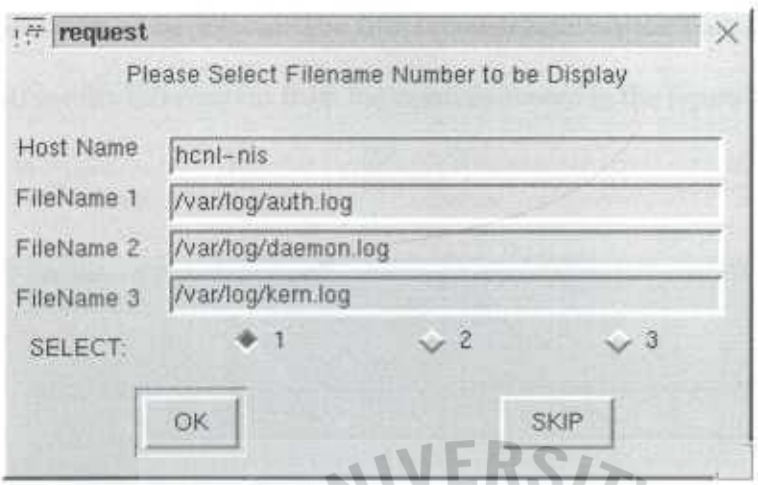


Figure 16 View Result Sub - Menu.

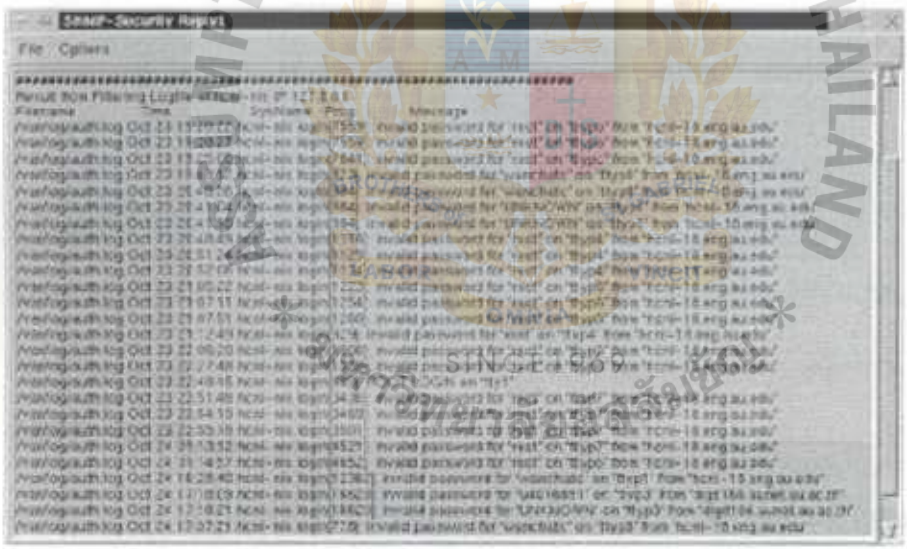


Figure 17 Result View Display.

4.2.3 Set Event Monitor

The manager must select the Filename number which would like to monitor. The second, third and forth requested data are Monitor Syntax 1, 2 and 3 which the manager can enter

the word that needs to be filtered. The fifth requested data is the time interval which the manager will get the information from the agent as shown in the figure 18.

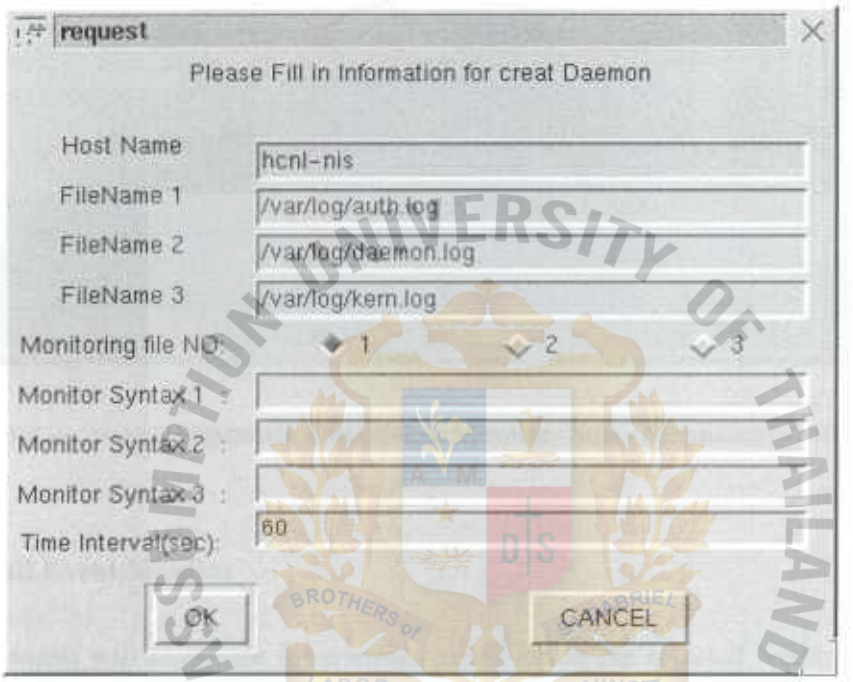


Figure 18 Set Event Monitor Sub Menu.

4.2.4 List Event Monitor

This sub-menu used to display the job of the event monitor as shown in the figure no. 19.

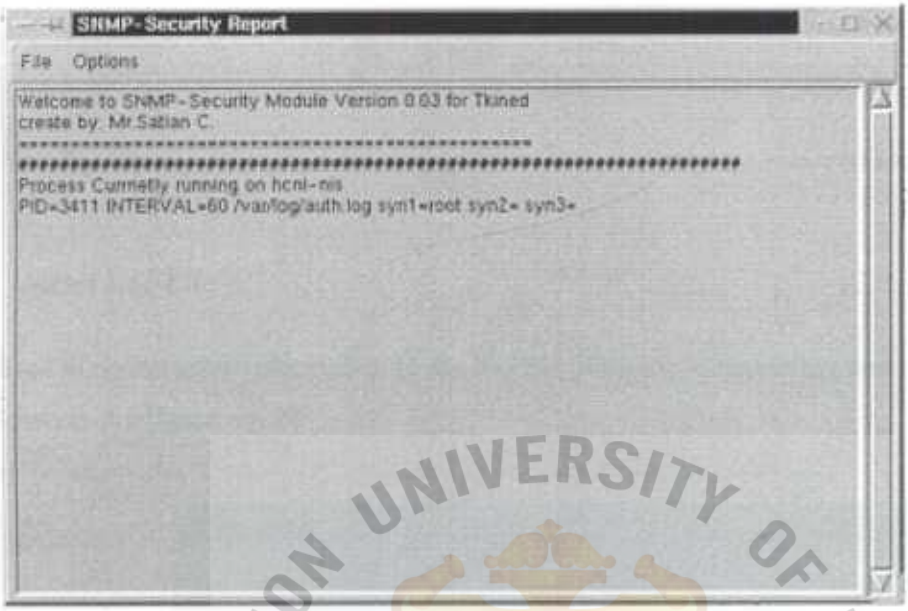


Figure 19 List Event Monitor Sub - Menu.

4.2.5 Kill Event Monitor

This sub-menu will terminate the monitor event job at the selected Node. After the the manager activated this sub-menu, the acknowledge window will display the selected node (for killing) as shown in figure 20.

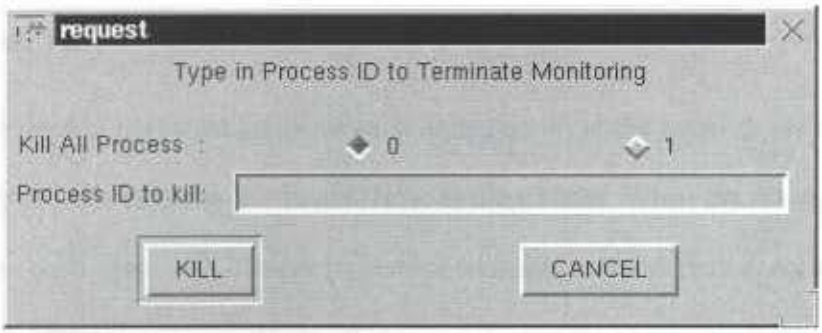


Figure 20 Kill Event Monitor Sub - Menu.

4.2.6 Reset Log File

This used to reset all the information in the log file (auth.log, daemon.log and kern.log) A1 shown in the figure no. 21

.

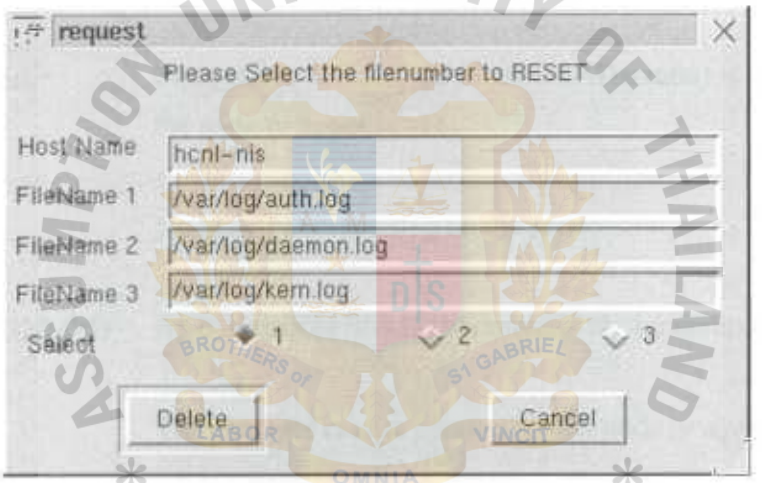


Figure 21 Reset Log File Sub - Menu.

4.2.7 Check View

This can be used to check the status of each node (agent) in the network according to the color setting of node status in the Event Color Setting Menu. When the manager activate this function, each agent will display its status according to the default color setting. This menu will check all the agent software that is installed at the agent whether it is supported by our application.

4.2.8 Event Color Setting

The manager can set the color of each status as shown below (figure 22) :

- NORMAL Status — Compatible SNMPD (UCD-SNMP Daemon) is running on the node or agent
- NO SNMPD — No SNMPD is running on the node
- NO COMPAT SNMPD — No compatible SNMPD is running on the node or agent.
- Unreachable — The manager cannot connect to that node or agent.
- Event Alert — The node will be blinking when the filter event has occurred.

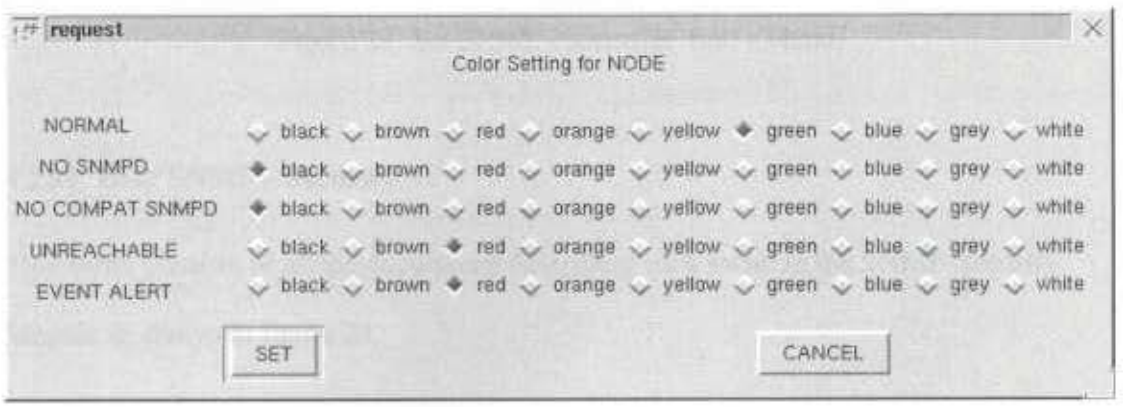


Figure 22 Event Color Setting Sub - Menu.

4.2.9 Set SNMP Parameter

This menu is used for setting the specification of SNMP Version 1 parameter concerning the community name, UDP Port, Time out, Retry, Window size, Delay and PGP Manager name and passphase as shown in figure 23.

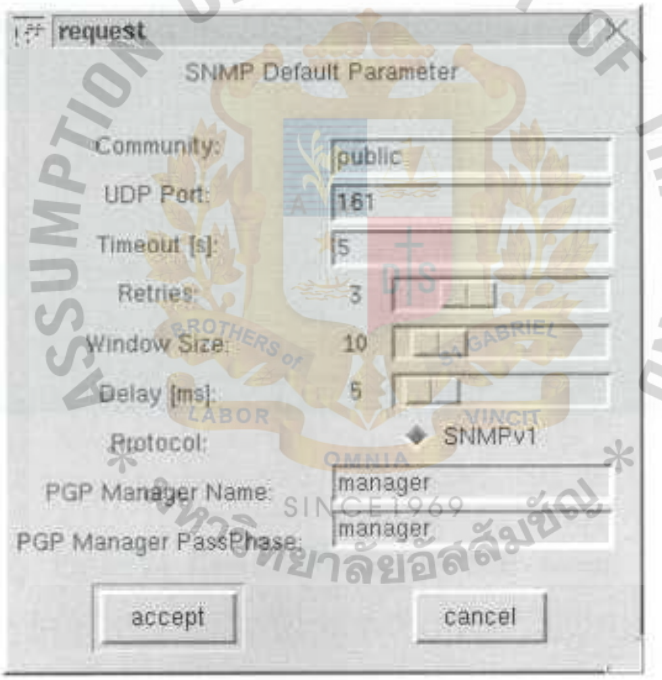


Figure 23 Set SNMP Parameter Sub - Menu.

4.2.10 Help SNMP – Security

This menu consists of all the documents describing each menu in the SNMP-Security Module as shown in figure 24.

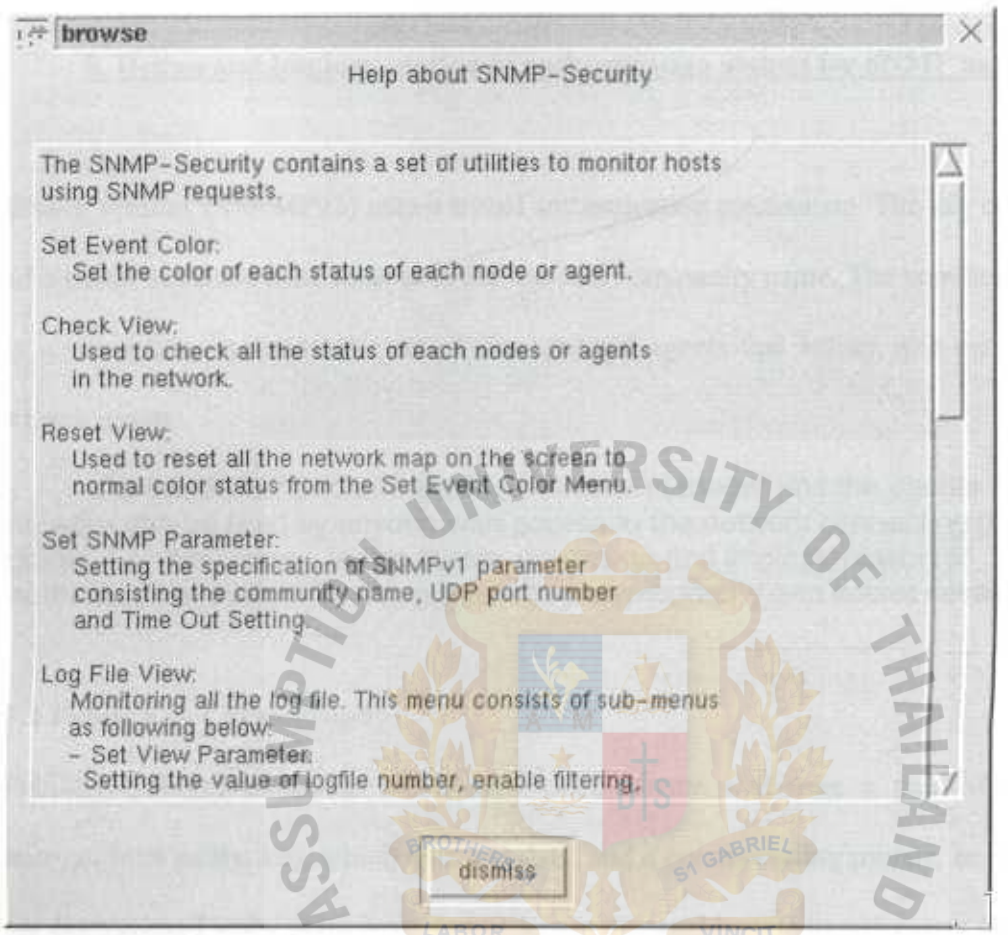


Figure 24 Help SNMP-Security Sub - Menu.

4.2.11 Delete SNMP – Security

This menu is used for terminating the SNMP – Security Module.

5. Design and Implementation of authentication system for SNMP party.

SNMP version 1 (SNMPv1) uses a trivial authentication mechanism. The key component of a trivial authentication scheme is the use of a community name. The community name is a string that represents the set of manager and agents that belong to a common and known group.

As the SNMP packet traveling between manager and the agents on the network can be read by anyone with access to the network connecting those points. For the privacy, in this thesis, we design and implementation of authentication system by using public key cryptography and secret session key.

5.1 Public Key Cryptography

Public key cryptography is an asymmetrical scheme that uses a pair of keys for encryption: a public key, which encrypts data, and a corresponding private, or secret key for decryption. Firstly, publish the public key to the world while keeping your private key secret. Anyone with a copy of your public key can then encrypted information that only you can read. Only the person who has the corresponding private key can decrypt the information.

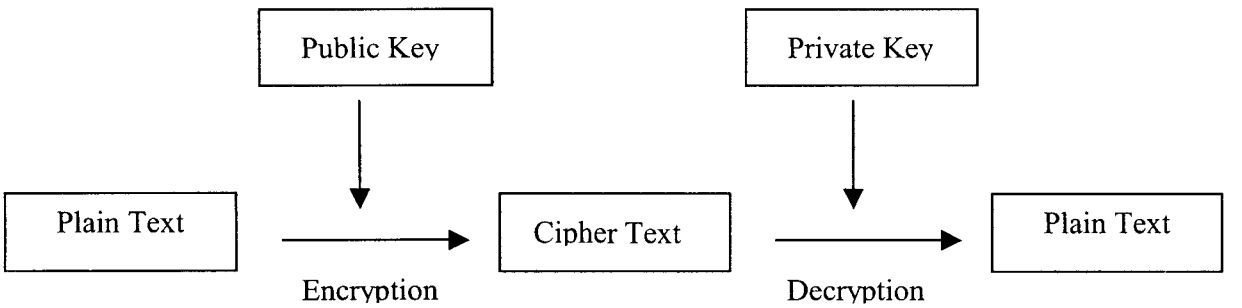


Figure 25 Public Key Encryption

The primary benefit of public key cryptography is that it allows people who have no preexisting security arrangement to exchange message securely. The need for sender and receiver to share secret keys via some secure channel is eliminated; all communications involve only public keys and no private key is ever transmitted or shared.

5.2 Session Key

When the SNMP authentication has been completed. Manager is sure he is talking to agent and Agent is sure he is talking to manager. Furthermore, the two of them will also have established a secret session key for use in the upcoming conversation.

The point of using session key for each new connection is to minimize the amount of traffic that gets sent with user's secret keys and public keys.

5.3 Design the authentication for SNMP Party.

5.3.1 Using Public Key Cryptography.

In this thesis, we design the authentication for SNMP Party by using the public key cryptography as shown in the figure 26.

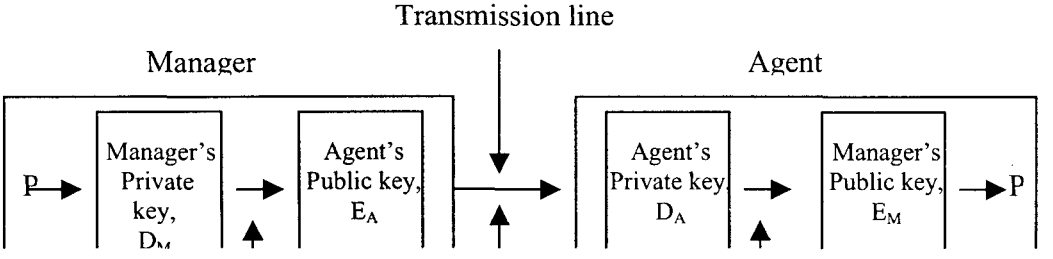


Figure 26 Manager to Agent Scheme

From the figure no. 26, assuming manager can send the packet (P) , setrequest or getrequest, to the agent by transmitting $E_A(D_M(P))$. Note carefully that manager knows his own (private) decryption key, D_M , as well as agent's public key, E_A .

When agent receives the message, the agent transforms the message using the agent's private key, D_A , as shown in Fig. 26. Then the agent decrypted the message using the manager's public key , E_M, to get the original message ,P.

We used the facility PGP, Pretty Good Privacy, to do the public key cryptography to generate a key pair consisting of a private key and public key. As its name implies, only manager have access to his private key, but in order to correspond with the agent the manager need a copy of the agent's public key and the agents need a copy of the manager's public key.

5.3.2 Session Key

After the conversation by the public key cryptography completed, the agent will send the requested information with the session key. Only the manager will know the meaning of the session key. In this thesis, we just implemented the simple session key as the offset the character by the value of the session key.

6. CONCLUSION

Using Tkined, we can implement a powerful network management application for network administrator. Particularly in this thesis, the developed network management application can monitor and control all the information of the unauthorized access from the management information base of the authentication log file at the hosts (agents) in the network.

On UNIX System – being a multi-user system – it is very important to monitor the login of users as the root or super user. The network administrator can monitor any unwanted access by using this network management application to secure UNIX systems.

In addition, this work also can be extended for monitoring and controlling of the other files in the directory /var/log, e.g. mail.log and so on. We hope this application will be useful for the UNIX network administrator.

BIBLIOGRAPHY

[1] Craig Hunt, TCP/IP Network Administration, O'REILLY & Associates Inc., 2 nd .edition,1998.

[2] Daniel A. Tauber, The Complete LINUX Kit, SYBEX Inc., USA, 1995.

[3] J.Case , M.Fedor , M.Schoffstaff , and J. Davin, "The Simple Network Management Protocol" , RFC 1157, University of Tennessee at Knoxville, Performance Systems International, Performance Systems International, and MIT Laboratory for Computer Science, May 1990.

[4] K. McClooghrie, "Management Information Base for Network Management of TCP/IP-based Internet" RFC 1156, Performance Systems International and Hughes LAN Systems, May 1990.

[5] Kevin Reichard, The Linux Internet Server, MIS:Press Newyork USA, 1997.

[6] Lars Wirzenius, Linux System Administrators' Guide 0.4, The Linux Documentation Project, Saturday 21 September, 1996.

[7] M. Rose, and K. McClooghrie, " Structure and Identification of Management Information for TCP/IP-based Internets" RFC 1155, Performance Systems International and Hughes LAN Systems, May 1990.

[8] Olaf Kirch, The Linux Network Administrators' Guide,Kattreinstr. 38, 64295 Darmstadt, Germany, 1994.

[9] Sean Harnedy, TOTAL SNMP, Prentice Hall PTR, 2nd edition, 1998.

[10] Sidnie Feit, SNMP, McGraw-Hill International Editions, USA, 1995.

[11] William Stallings, SNMP, SNMPv2, and CMIP "The Practical Guide to Network-Management Standards", Addison-Wesley Publishing Company,in 1995.

LIST OF ACRONYMS

SNMP	Simple Network Management Protocol
SNMPv1	Simple Network Management Protocol Version 1
MIB	Management Information Base
Tcl/Tk	Tool Command Language / Tool Kit
GUI	Graphical User Interface
TCP/IP	Transmission Control Protocol / Internet Protocol
UDP/IP	User Datagram Protocol / Internet Protocol
NMA	Network Management Application
NME	Network Management Entity
NMS	Network Management Station
OS	Operating System
PDU	Protocol Data Unit
auth	authentication

APPENDIX A

Computer code for Chapter 4

The following code runs in "Tcl/Tk". It demonstrates an implementation of the network management application in Chapter 4.

```
#!/bin/sh
# the next line restarts using scotty -*- tcl -*- \
exec scotty2.1.10 "$0" "$@"
#
# snmp_security.tcl -
#
# Simple SNMP security monitoring scripts for Tkined.
# Coder:  Mr.Satian  Chokdeepanich
package require Tnm 2.1

ined size
LoadDefaults snmp monitor
SnmpInit SNMP-Security

##
## Set Default MIB file using by Tkined
##
mib load /usr/local/share/snmp/mibs/LOGS-MIB.txt
##

##
## Initilize Report View Window
##
writeln "Welcome to SNMP-Security Module Version 0.03 for Tkined"
writeln "create by: Mr.Satian C."
writeln "=============================================="
##

if {[info exists default(interval)]} {
   set interval $default(interval)
} else {
   set interval 60
}
if {![info exists default(graph)]} {
   set default(graph) false
```

```
}
global first
set first 0

proc createjob { } {
global time
set pid [job create -command "trapcheck" -interval $time]
##writeln "pid = $pid"
}

##
## Sesskey encode
##

proc sessenc { sesskey plaintext } {

  set enctext [exec /usr/lib/tkined1.4.10/apps/sessenc $sesskey $plaintext]
  return $enctext
}

##
## Sesskey decode
##

proc sessdec { sesskey enctext } {

  set plaintext [exec /usr/lib/tkined1.4.10/apps/sessdec $sesskey $enctext]
  return $plaintext
}


## Flash the icon of the object given by the ip address.
## Keep track of all ip addresses to flash in an array.
##

proc flash {op {ip ""}} {

   static jobid
   static ipaddrs

   switch $op {

      list {
         if {[info exists ipaddrs]} {
```

```
        return [lsort [array names ipaddrs]]
    } else {
        return {}
    }
}

add {
    if {$ip == ""} return
    if {![info exists ipaddrs($ip)]} {
        set ipaddrs($ip) 1
    } else {
        incr ipaddrs($ip) 1
    }
    if {![info exists jobid]} {
        set jobid [job create -command "flash run" -interval 10000]
    }
    return
}

remove {
    if {![info exists ipaddrs]} return
    if {![info exists ipaddrs($ip)]} return
    incr ipaddrs($ip) -1
    if {$ipaddrs($ip) <= 0} {
        unset ipaddrs($ip)
    }
    if {[llength [array names ipaddrs]] == 0} {
        $jobid destroy
        unset jobid
        unset ipaddrs
        writeln "All highlights deleted."
        writeln
    }
    return
}

reset {
    if {![info exists ipaddrs]} return
    if {[info exists ipaddrs($ip)]} {
        unset ipaddrs($ip)
    }
    if {[llength [array names ipaddrs]] == 0} {
        $jobid destroy
        unset jobid
        unset ipaddrs
        writeln "All highlights deleted."
```

```
        }
        return
    }

    run {
        foreach ip [array names ipaddrs] {
            IpFlash $ip [expr {$ipaddrs($ip) < 10 ? $ipaddrs($ip) : 10}]
        }
    }
  }
}


##
## Create a chart. Honours the monitor.graph default definition.
##

proc CreateChart {id x y} {
  global default
  if {$default(graph) != "true"} {
        set id [CloneNode $id [ined create STRIPCHART] $x $y]
  } else {
        set id [CloneNode $id [ined create GRAPH]]
  }
  return $id
}

##
## Tell tkined to restart a command when a saved map gets loaded.
##

proc save { cmd } {
  set cmdList [ined restart]
  if {[lsearch $cmdList $cmd] >= 0} return
  lappend cmdList $cmd
  ined restart $cmdList
}

##
## Remove a command from the list of commands saved in the tkined map.
##

proc forget { cmd } {
  set cmdList ""
  foreach c [ined restart] {
        if {$c != $cmd} {
```

```
            lappend cmdList $c
        }
    }
    ined restart $cmdList
}

##
## Restart a monitor job by calling cmd. Build the list of objects
## from tkined using ined retrieve commands.
##

proc restart { cmd args } {

    global interval

    if {! [catch {expr [lindex $args 0] + 0}]} {
        set itv  [lindex $args 0]
        set jid  [lindex $args 1]
        set args [lrange $args 2 end]
        set old_interval $interval
        set interval [expr int($itv)]
    }

    foreach id $args {
        catch {ined -noupdate clear $id}
        catch {ined -noupdate clear $id}
    }
    catch {eval $cmd $args}

    if {[info exists old_interval]} {
        set interval $old_interval
    }
}

##
## Monitor a SNMP variable in a stripchart. This version uses two
## procedures. The ShowVariable proc is called by the job scheduler
## to send SNMP requests at regular intervals. SNMP responses are
## processed by ShowVariableProc.
##

proc ShowVariableProc {id status vbl} {
    global cx
    if {$status != "noError"} {
        ined -noupdate values $id 0
        ined -noupdate attribute $id $cx(descr,$id) \
```

```
                "$cx(descr,$id) [lindex $vbl 0]"
        return
    }
    set now [mib scan sysUpTime [lindex [lindex $vbl 0] 2]]
    set syntax [lindex [lindex $vbl 1] 1]
    set value  [lindex [lindex $vbl 1] 2]
    set val 0
    switch -glob $syntax {
        Gauge -
        INTEGER -
        Unsigned32 -
        Integer32 {
            set val $value
        }
        Counter* {
            if {$now > $cx(time,$id)} {
                set val [expr $value - $cx(value,$id)]
                set val [expr $val / ( ($now-$cx(time,$id)) / 100.0 )]
            } else {
                set val 0
            }
        }
    }
    ined -noupdate values $id $val
    ined -noupdate attribute $id $cx(descr,$id) "$cx(descr,$id) $val"
    set cx(value,$id) $value
    set cx(time,$id) $now
    set txt "[ined -noupdate name $id] $cx(descr,$id)"
    MoJoCheckThreshold $id $txt $val
}

proc ShowVariable { ids } {
    global cx
    foreach id [MoJoCheckIds ShowVariable $ids] {
        $cx(snmp,$id) get "sysUpTime.0 $cx(name,$id)" \
                "ShowVariableProc $id %E \"%V\""
    }
}

##
## Start a new monitoring job for the device given by ip. The args
## list contains node ids with the snmp variable oid to display.
##

proc start_snmp_monitor { ip args } {
    global cx interval
```

```
    set ids ""
    set re_args ""

    while {$args != ""} {
        set id [lindex $args 0]
        set var [lindex $args 1]

        if {$id != ""} {
            set s [SnmpOpen $id $ip]
            if {[catch {$s get "sysUpTime.0 $var"} vbl]} {
                set name [ined -noupdate name $id]
                set addr [ined -noupdate address $id]
                set msg "Unable to restart monitor job on $name"
                if {$addr != ""} {
                    append msg " \[$addr\]"
                }
                writeln "$msg.\nSNMP get failed: $vbl\n"
                $s destroy
                set args [lrange $args 2 end]
                ined delete $id
                continue
            }
            set cx(time,$id)  [mib scan sysUpTime [lindex [lindex $vbl 0] 2]]
            set cx(value,$id) [lindex [lindex $vbl 1] 2]
            set cx(snmp,$id) $s
            set cx(name,$id) $var
            set cx(descr,$id) [mib name [mib oid $var]]
            lappend ids $id

            ined -noupdate attribute $id $cx(descr,$id) $cx(descr,$id)
            ined -noupdate label $id $cx(descr,$id)
        }

        set args [lrange $args 2 end]
        append re_args " \$$id $var"
    }

    if {$ids != ""} {
        set jid [job create -command [list ShowVariable $ids] \
                -interval [expr $interval * 1000]]
        save "restart start_snmp_monitor $interval $jid $ip $re_args"
    }
}

##
```

```
## Prepare a monitoring job for the objects in list and the SNMP variable
## given by varname. This proc is also called by other scripts so changes
## must be done very carefully...
##


proc MonitorVariable {list varname} {
   ForeachIpNode id ip host $list {
        set s [SnmpOpen $id $ip]
        set syntaxList {
           Gauge Gauge32 Counter Counter32 Counter64
           Unsigned32 INTEGER Integer32
        }
        set vars ""

        set err [catch {$s get $varname} vbl]
        if {$err || [lindex [lindex $vbl 0] 1] == "noSuchInstance"} {
           if {[catch {
                $s walk vbl $varname {
                   set syntax [lindex [lindex $vbl 0] 1]
                   if {[lsearch $syntaxList $syntax] >= 0} {
                        lappend vars [lindex [lindex $vbl 0] 0]
                   } else {
                        set name [mib name [lindex [lindex $vbl 0] 0]]
                        writeln "$host: $name of base type $syntax ignored"
                   }
                }
           } msg]} {
                ined acknowledge "Monitor Variable $varname on $ip: $msg"
                $s destroy
                continue
           }
        } else {
           set syntax [lindex [lindex $vbl 0] 1]
           if {[lsearch $syntaxList $syntax] >= 0} {
                lappend vars [lindex [lindex $vbl 0] 0]
           } else {
                set name [mib name [lindex [lindex $vbl 0] 0]]
                writeln "$host: $name of base type $syntax ignored"
           }
        }

        if {$vars == ""} {
           ined acknowledge \
                   "$varname is either not available or of wrong type."
           continue
```

```
        }

        set args $ip
        set i 0
        foreach var $vars {
           set nid [CreateChart $id [expr 30+$i] [expr 30+$i]]
           ined -noupdate attribute $nid "SNMP:Config" [$s configure]
           lappend args $nid
           lappend args $var
           incr i
        }
        $s destroy

        eval start_snmp_monitor $args
   }
}

##
## This procedure used to check weather there are SNMPD agent
## which compatible with LOGS MIB brance running on
## the specific host selected or weather the host exists or not.
##
## return value ('0' == OK) ('1' == NOSNMPD)
##              ('2' == NOCOMPATSNMPD) ('3' == UNREACHABLE)
##
proc HostCheck { id ip } {

   if {[catch {icmp -size 0 -timeout 5 echo $ip} rtt]} {
     writeln "Could not send icmp packet: $rtt"
     writeln
   }
   set rtt [lindex [join $rtt] 1]
   if {$rtt < 0} {
     return 3
   }

   set res ""
   set s [SnmpOpen $id $ip]
   if {[catch {
       $s walk x 1.3.6.1.4.1.4210.10.1.1.4 {
       ##writeln "value x = $x"
       lappend res [lindex [lindex $x 0] 2]
       }
     } msg]} {
     $s destroy
     return 1
```

```
    #no SNMPD running
  }
  $s destroy
  if { $res == "" } {
    ##writeln "message = $msg"
    return 2
    #no compatible SNMPD
  }
  return 0
}

##
## Host Colored
##
proc color_host { id ip host flag } {
  global color
  upvar $flag lflag
  set lflag [HostCheck $id $ip]

  if { $lflag == 0 } {
    ined color $id $color(OK)
  }
  if { $lflag == 1 } {
    ined color $id $color(NO_SNMPD)
    ined acknowledge "Unable to contact to $host !!NO SNMPD RUNNING!!"
  }
  if { $lflag == 2 } {
    ined color $id $color(NO_COMPAT_SNMPD)
    ined acknowledge "Unable to contact to $host !!NO COMPATIBLE SNMPD
RUNNING!!"
  }
  if { $lflag == 3 } {
    ined color $id $color(UNREACHABLE)
    ined acknowledge "Unable to contact to $host !!UNREACHABLE HOST!!"
  }
}

##
## color toggle
##
proc color_alert { id color1 color2 } {
  static hostcolor
  set hostcolor [ined color $id]

  if { [lsearch $hostcolor $color1] == 0 } {
    ined color $id $color2
```

```
    } elseif { [lsearch $hostcolor $color2] == 0 } {
      ined color $id $color1
    } else {
      ined color $id $color1
    }
}



##
## initlization color parameter for given event
##
proc color_init { } {
    global color

    set color(OK) green
    set color(NO_SNMPD) black
    set color(NO_COMPAT_SNMPD) black
    set color(UNREACHABLE) red
    set color(EVENT) red
}
color_init

##
## color parameter setting
##
proc "Event Color Setting" { list } {
    global color

    set result [ined request "Color Setting for NODE" \
      [list [list "NORMAL          " $color(OK) radio black brown red orange yellow green
blue grey white ] \
          [list "NO SNMPD        " $color(NO_SNMPD) radio black brown red orange
yellow green blue grey white ] \
              [list "NO COMPAT SNMPD" $color(NO_COMPAT_SNMPD) radio black
brown red orange yellow green blue grey white ] \
          [list "UNREACHABLE     " $color(UNREACHABLE) radio black brown red
orange yellow green blue grey white ] \
          [list "EVENT ALERT     " $color(EVENT) radio black brown red orange yellow
green blue grey white ] ] \
      [list SET CANCEL] ]
    if { [lindex $result 0] == "CANCEL" } return
    set color(OK) [lindex $result 1]
    set color(NO_SNMPD) [lindex $result 2]
    set color(NO_COMPAT_SNMPD) [lindex $result 3]
    set color(UNREACHABLE) [lindex $result 4]
```

```
    set color(EVENT) [lindex $result 5]
}
##
## This procedure used to ease ForeachIpNode also return current NODE
## in the list using.
##
proc ForeachIpNode2 { id ip host lcomp list body } {
    upvar $id lid
    upvar $ip lip
    upvar $host lhost
    upvar $lcomp comp
    foreach comp $list {
        if {[ined type $comp] == "NODE"} {
            set lid [ined id $comp]
            set lip [GetIpAddress $comp]
            set lhost [ined name $comp]
            if {$lip == ""} {
                set host [lindex [ined name $comp] 0]
                ined acknowledge "Can not lookup IP Address for $host."
                continue
            }
            uplevel 1 $body
        }
    }
}

##
## This command used for retrieve the log file from the remote host using
## SNMP protocol.
##
proc "Set View Parameter" { list } {
    ##global logname usefilter syntax1 syntax2 syntax3
    ##global andflag updateflag color
    global color

    "Check View" $list
    set node [lindex $list 0]
    set ip [GetIpAddress $node]

    set res ""
    ForeachIpNode2 id ip host lcomp $list {

      if { [ined color $id] != $color(OK) } {
        ined acknowledge "Unable to Contact to $host Agent"
        continue
      }
```

```
    set mylist [list [ined retrieve $id]]
    "Retrieve Session Key" $mylist
    set sesskey [ined attribute $id "sesskey"]
    set s [SnmpOpen $id $ip]

    set namelist ""
    if {[catch {$s walk x 1.3.6.1.4.1.4210.10.1.50.1.1 {
      lappend namelist [lindex [lindex $x 0] 2]
      }
        } msg] != 0} {
        ined acknowledge "Unable to Connect to $host"
        ##ined color $id $color(UNREACHABLE)
        $s destroy
        continue
    }
    if {$namelist == ""} {
      ##ined color $id $color(NO_COMPAT_SNMPD)
      ined acknowledge "Unable to Contact Agent"
        $s destroy
        continue
    }
    $s destroy

    set file_name(1) [lindex $namelist 3];set file_name(1) [sessdec $sesskey
$file_name(1)]
    set file_name(2) [lindex $namelist 4];set file_name(2) [sessdec $sesskey
$file_name(2)]
    set file_name(3) [lindex $namelist 5];set file_name(3) [sessdec $sesskey
$file_name(3)]

    set selectfile 1
    set select [ined request "Please Select Filename Number" \
    [list [list "Host Name" $host request] \
    [list "FileName 1" $file_name(1) request] \
    [list "FileName 2" $file_name(2) request] \
    [list "FileName 3" $file_name(3) request] \
    [list "SELECT:" $selectfile radio 1 2 3] ] \
    [list "OK" "SKIP"]]

    if { [lindex $select 0] == "SKIP" } {
      continue
    }
    set selectfile [lindex $select 5]

    set s [SnmpOpen $id $ip]
    if {[catch {
```

```
    $s walk x 1.3.6.1.4.1.4210.10.1.$selectfile.2 {
       lappend res [lindex [lindex $x 0] 2]
    }
     } msg]} {
     ined acknowledge "Unable to contact to $host :unreachable HOST"
     $s destroy
     ##ined color $id $color(UNREACHABLE)
     continue
}
if { $res == "" } {
     ined acknowledge "Unable to contact to $host :unable to conatact SNMPD"
     ##ined color $id $color(NO_COMPAT_SNMPD)
     $s destroy
     continue
}
$s destroy

set enable    [lindex $res 0];set enable        [expr $enable - $sesskey]
set argument1 [lindex $res 1];set argument1   [sessdec $sesskey $argument1]
set argument2 [lindex $res 2];set argument2   [sessdec $sesskey $argument2]
set argument3 [lindex $res 3];set argument3   [sessdec $sesskey $argument3]

set result [ined request "SET PARAMETER" \
      [list [list "Logfile Name:" $file_name($selectfile) request ] \
      [list "Enable Filter:" $enable radio 0 1  ] \
      [list "Filtering Syntax 1:" $argument1 request ] \
      [list "Filtering Syntax 2:" $argument2 request ] \
      [list "Filtering Syntax 3:" $argument3 request ] ] \
      [list SET CANCEL ] ]
if {[lindex $result 0] == "CANCEL"} {
   continue
}

set enable     [lindex $result 2];set enable    [expr $enable + $sesskey]
set argument1   [lindex $result 3];set argument1 [sessenc $sesskey $argument1]
set argument2   [lindex $result 4];set argument2 [sessenc $sesskey $argument2]
set argument3   [lindex $result 5];set argument3 [sessenc $sesskey $argument3]

set s [SnmpOpen $id $ip]
if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.$selectfile.2.1.0 \
    $enable]]} msg] {
      error $msg
}
if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.$selectfile.2.2.0 \
 $argument1]]} msg] {
   error $msg
```

```
    }
    if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.$selectfile.2.3.0 \
     $argument2]]} msg] {
       error $msg
    }
    if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.$selectfile.2.4.0 \
     $argument3]]} msg] {
       error $msg
    }
    $s destroy
    ined acknowledge "Saving Configuration for $host Successfully"
  }
}

##
## This command used for Update All the result Kept IP address in array
## using by previous menu "Set Parameter"
##
proc "Update Result (All)" { list } {
  global logname usefilter

  set iplist [array name logname]
  set list [ined retrieve]

  ForeachIpNode id ip host $list {
    set flag [lsearch $iplist $ip]
    if { $flag == -1 } {
          continue
    } else {
          if { $usefilter($ip) == "YES" } {
            set s [SnmpOpen $id $ip]
         if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.101.2.0 1]]} msg] {
            error $msg
         }
         $s destroy
          }
    }
  }
}
##
## This command used for Update the result MIB branch of selected ID
##
##
proc "Update Result" { list } {

  global logname usefilter
```

```
   "Check View" $list
   set iplist [array name logname]
   ForeachIpNode id ip host $list {
    set flag [lsearch $iplist $ip]
    if { $flag == -1 } {
       continue
    } else {
       if { $usefilter($ip) == "YES" } {
           set s [SnmpOpen $id $ip]
        if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.101.2.0 1]]} msg] {
           error $msg
        }
        $s destroy
         }
    }
   }
}

##
## This command used for display the result of the Filtering Data..
##
##
proc "View Result" { list } {

   global color
   "Check View" $list
   ForeachIpNode id ip host $list {

     set mylist [list [ined retrieve $id]]
     "Retrieve Session Key" $mylist
     set sesskey [ined attribute $id "sesskey"]
     set s [SnmpOpen $id $ip]
     set namelist ""
     if {[catch {$s walk x 1.3.6.1.4.1.4210.10.1.50.1.1 {
       lappend namelist [lindex [lindex $x 0] 2]
     }
     } msg] != 0} {
          ined acknowledge "Unable to Connect to $host"
       ined color $id $color(UNREACHABLE)
          $s destroy
          continue
     }
     if {$namelist == ""} {
       ined color $id $color(NO_COMPAT_SNMPD)
          $s destroy
          continue
```

```
    }

    $s destroy

    set file_name(1) [lindex $namelist 3];set file_name(1) [sessdec $sesskey
$file_name(1)]
    set file_name(2) [lindex $namelist 4];set file_name(2) [sessdec $sesskey
$file_name(2)]
    set file_name(3) [lindex $namelist 5];set file_name(3) [sessdec $sesskey
$file_name(3)]

    set selectfile 1
    set select [ined request "Please Select Filename Number to be Display" \
            [list [list "Host Name" $host request] \
              [list "FileName 1" $file_name(1) request] \
              [list "FileName 2" $file_name(2) request] \
              [list "FileName 3" $file_name(3) request] \
              [list "SELECT:" $selectfile radio 1 2 3] ] \
          [list "OK" "SKIP"]]

    if { [lindex $select 0] == "SKIP" } {
      continue
    }
    set selectfile [lindex $select 5]

    set s [SnmpOpen $id $ip]
    set count 0
    writeln
"#####################################################################"
    writeln "Result from Filtering Logfile of $host IP $ip"
    $s walk x 1.3.6.1.4.1.4210.10.1.$selectfile.1 {
      incr count
    }
    set num_row [expr $count / 6]
    set one 1;set two 0;set three 0;set four 0;set five 0; set six 0
    set innercount 1
    $s walk x 1.3.6.1.4.1.4210.10.1.$selectfile.1 {
      if { $one == 1 } {
        if { $innercount == $num_row } {
          set innercount 1; set one 0; set two 1
          continue
        }
        incr innercount
      }
      if { $two == 1 } {
        if { $innercount == $num_row } {
```

```
      set filename($innercount) [lindex [lindex $x 0] 2]
      set filename($innercount) [sessdec $sesskey $filename($innercount)]
      set innercount 1; set two 0; set three 1
      continue
    }
    set filename($innercount) [lindex [lindex $x 0] 2]
    set filename($innercount) [sessdec $sesskey $filename($innercount)]
    incr innercount
  }
  if { $three == 1 } {
    if { $innercount == $num_row } {
      set time($innercount) [lindex [lindex $x 0] 2]
      set time($innercount) [sessdec $sesskey $time($innercount)]
      set innercount 1; set three 0; set four 1
      continue
    }
    set time($innercount) [lindex [lindex $x 0] 2]
    set time($innercount) [sessdec $sesskey $time($innercount)]
    incr innercount
  }
  if { $four == 1 } {
    if { $innercount == $num_row } {
      set sysname($innercount) [lindex [lindex $x 0] 2]
      set sysname($innercount) [sessdec $sesskey $sysname($innercount)]
      set innercount 1; set four 0; set five 1
      continue
    }
    set sysname($innercount) [lindex [lindex $x 0] 2]
    set sysname($innercount) [sessdec $sesskey $sysname($innercount)]
    incr innercount
  }
  if { $five == 1 } {
    if { $innercount == $num_row } {
      set prog($innercount) [lindex [lindex $x 0] 2]
      set prog($innercount) [sessdec $sesskey $prog($innercount)]
      set innercount 1; set five 0; set six 1
      continue
    }
    set prog($innercount) [lindex [lindex $x 0] 2]
    set prog($innercount) [sessdec $sesskey $prog($innercount)]
    incr innercount
  }
  if { $six == 1 } {
    if { $innercount == $num_row } {
      set message($innercount) [lindex [lindex $x 0] 2]
      set message($innercount) [sessdec $sesskey $message($innercount)]
```

```
            set innercount 1; set six 0
            continue
          }
          set message($innercount) [lindex [lindex $x 0] 2]
          set message($innercount) [sessdec $sesskey $message($innercount)]
          incr innercount
        }
      }
      $s destroy
      writeln "Filename          Time              SysName   Prog          Message"
      for {set count 1} {$count <= $num_row} {incr count} {
        writeln "$filename($count) $time($count) $sysname($count) $prog($count)
$message($count)"
      }
    }
}



##
## This is the command used to monitor number of rows of totalrow variable
## of the SNMP agent.
##
proc "Log File Monitor" { list } {

    static varname

    if {![info exists varname]} {
          set varname 1
    }
    set varname 1
    set res [ined request "Please choose logfile number: (1=AUTH)" \
              [list [list "Log File:" $varname  ] ] \
              [list start cancel] ]
    if {[lindex $res 0] == "cancel"} return

    set varname ""
    foreach var [lindex $res 1] {
          lappend var 3.0
          set var [linsert $var 0 1.3.6.1.4.1.4210.10.1]
          set var [join $var .]
          if {[catch {mib oid $var}]} {
            ined acknowledge "Unknown Log File variable \"$var\"."
            continue
          }
```

```
        lappend varname $var
    }

    foreach var $varname {
        MonitorVariable $list $var
    }
}

##
## Command to Reset the Log file in the concerning host
## using SNMP command to set OID 1.3.6.1.4.1.4210.10.1.FileNumber.4.0
##
proc "Reset Log File" { list } {
  global color
  ForeachIpNode id ip host $list {

    if { [ined color $id] != $color(OK) } {
      ined acknowledge "Unable to Contact to $host Agent"
      continue
    }
    set mylist [list [ined retrieve $id]]
    "Retrieve Session Key" $mylist
    set sesskey [ined attribute $id "sesskey"]
    set s [SnmpOpen $id $ip]

    set namelist ""
    if {[catch {$s walk x 1.3.6.1.4.1.4210.10.1.50.1.1 {
      lappend namelist [lindex [lindex $x 0] 2]
      }
      } msg] != 0} {
      ined acknowledge "Unable to Connect to $host"
      $s destroy
      continue
    }
    if {$namelist == ""} {
      ined acknowledge "Unable to Contact Agent"
      $s destroy
      continue
    }
    $s destroy
    set file_name(1) [lindex $namelist 3];set file_name(1) [sessdec $sesskey
$file_name(1)]
    set file_name(2) [lindex $namelist 4];set file_name(2) [sessdec $sesskey
$file_name(2)]
    set file_name(3) [lindex $namelist 5];set file_name(3) [sessdec $sesskey
$file_name(3)]
```

```
    set interval 60; set file 1
    set select [ined request "Please Select the filenumber to RESET" \
    [list [list "Host Name" $host request] \
    [list "FileName 1" $file_name(1) request] \
    [list "FileName 2" $file_name(2) request] \
    [list "FileName 3" $file_name(3) request] \
    [list "Select    " $file radio 1 2 3]] \
    [list "Delete" "Cancel"]]

    if {[lindex $select 0] == "Cancel"} {
      continue
    }
    set file [lindex $select 5]
    set s [SnmpOpen $id $ip]

    if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.$file.3.0 \
     [expr 1 + $sesskey]]]} msg] {
       error $msg
    }
    $s destroy
    ined acknowledge "File Reset Successfully"
  }
}

##
## Read Filter Parameter from remote HOST
##

proc filter_read { id ip logname syntax1 syntax2 andflag updateflag } {

upvar $logname llogname
upvar $syntax1 lsyntax1
upvar $syntax2 lsyntax2
upvar $andflag landflag
upvar $updateflag lupdateflag

   set res ""
   set s [SnmpOpen $ip $ip]
   if {[catch {
        $s walk x 1.3.6.1.4.1.4210.10.1.100 {
          lappend res [lindex [lindex $x 0] 2]
        }
      } msg]} {
    ined acknowledge "SNMP ERROR  !!$msg!!"
    return
```

```
    }
    set llogname [lindex $res 0]
    set lsyntax1 [lindex $res 1]
    set lsyntax2 [lindex $res 2]
    set landflag [lindex $res 3]
    set lupdateflag [lindex $res 4]
    $s destroy
}

##
## Command used to kill Blink Alert Job and remove from the
## alert_computer array.
##

proc kill_blink { id ip } {
global alert_computer color

  set listalert [array name alert_computer]
  if { [lsearch $listalert $ip] != -1 } {
    $alert_computer($ip) destroy
    ined color $id $color(OK)
    unset alert_computer($ip)
  }
}


##
## Command used to MONITOR the filtering EVENT and
## display the appropriate from setting.
##

proc monitor { id ip host logname {syntax1 0} {syntax2 0} {andflag 0} } {

global color loutput alert_computer rowcount change


    ##
    ## change syntax back from 0 to ""
    if { $syntax1 == 0 } {
      set syntax1 ""
    }
    if { $syntax2 == 0 } {
      set syntax2 ""
    }

    #ined acknowledge "change flag in monitor proc  == $change"
```

```
##
## update using SET operation if change equal 1
if { $change == 1 } {
 set s [SnmpOpen $id $ip]
 ##
 ##if there are change in any field we will reset the LastRead to 0
 if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.$logname.2.0 0]]} msg] {
  ined acknowledge "Fail set AuthLastRead"
 }



 if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.100.1.0 $logname]]} msg] {
  ined acknowledge "Fail set Syntax1"
 }
 if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.100.2.0 $syntax1]]} msg] {
  ined acknowledge "Fail set Syntax2"
 }
 if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.100.3.0 $syntax2]]} msg] {
  ined acknowledge "Fail set Syntax1"
 }
 if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.100.4.0 $andflag]]} msg] {
  ined acknowledge "Fail set And_Flag"
 }
 if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.100.5.0 1]]} msg] {
  ined acknowledge "Fail Set UpdateData Flag"
 }
 set change 0
 $s destroy
}
#end of updating process

##
## update data
set s [SnmpOpen $id $ip]
if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.101.2.0 1]]} msg] {
 error $msg
}
$s destroy

set count 0
## count variable used to collect total element found on WALK
set result ""
##
## get result data
set s [SnmpOpen $id $ip]
```

```
if {[catch {
  $s walk data 1.3.6.1.4.1.4210.10.1.101.1 {
    set data [lindex [lindex $data 0] 2]
    lappend result $data
    incr count
  }
  } msg]} {
  ined acknowledge "Fail Get (Walk) Data"
}
$s destroy

set totalrow 0
if {![info exist loutput]} {
  set loutput(0) "just test"
}
#writeln "Total Filter Element Result row*5 == $count"
#writeln "totalrow == $totalrow" (debug)

arrangewalkdata totalrow $ip $result

##now the data are kept in variable OUTPUT in the form of
##global loutput(rownumber,ipnumber) == DATA
##and increasingly until unset the rowcount($ip) == totalunreadrow to 0
## (require user intervention)


##put on effect BLINK on concerning computer
if { $totalrow > 0 } {
  writeln "You've got new EVENT alert"
  #ined color $id $color(EVENT)
  if { ![info exists alert_computer] } {
    set jobid [job create -command "color_alert $id $color(EVENT) black" -interval
200]
    set alert_computer($ip) $jobid
  } else {
    ## if the array exist, check weather there is already running job
    ## of the IP address, if NO create a new JOB blink if YES just skip
    set alert_list [array name alert_computer]
    if { [lsearch $alert_list $ip] == -1 } {
      set jobid [job create -command "color_alert $id $color(EVENT) black" -interval
200]
      set alert_computer($ip) $jobid
    } else {
      writeln "already alert NO adding JOB"
    }
  }
```

```
    }
    set totalrow 0

    #writeln "all job run for alert == [array name alert_computer]"
    #set jobid [job create -command "color_alert $id $color(EVENT) black" -interval 200]

    #writeln "id of the computer == $id"
    #writeln "jobname == $jobid"
    #writeln "jobconfigure == [job0 configure]"
    #set nodevalue [lindex [lindex [job0 configure] 1] 1]
    #writeln "node value from search == $nodevalue"

    ## $loutput used to kept the answer from the query SNMPD and
    ## will not delete until user intervention.. (be careful)
    #if { [info exists loutput(0,$ip)] } {
    #  writeln "really result == $loutput(0,$ip)"
    writeln "total result row for $ip == $rowcount($ip)"
    #}
}


##
## arrangewalkdata Proc used to arrange data from WALK
## and put to some global variable

proc arrangewalkdata { totalrow ip result } {

global loutput
global rowcount
## $rowcount($ip) array used to kept the number of ROWs in the array LOG file
## not concern with the exact size of the Real Log File. require user intervention
## to reset the size (be careful)

upvar $totalrow ltotalrow
#upvar $output loutput

    set rowcount(0) 0
    set loutput(0) ""
    set totalelement [llength $result]
    set ltotalrow [expr $totalelement/5]

    #writeln "In the arrange Walk data totalelement == $totalelement"
    if { ![info exists rowcount($ip)] } {
      set rowcount($ip) 0
    }
    for {set i 0} {$i < $totalelement} {incr i} {
```

```
    if { [expr $i / $ltotalrow] == 0 } {
      set loutput([expr $i % $ltotalrow + $rowcount($ip)],$ip) [lindex $result $i]
    } elseif { [expr $i / $ltotalrow] == 1 } {
      lappend loutput([expr $i % $ltotalrow + $rowcount($ip)],$ip) [lindex $result $i]
    } elseif { [expr $i / $ltotalrow] == 2 } {
      lappend loutput([expr $i % $ltotalrow + $rowcount($ip)],$ip) [lindex $result $i]
    } elseif { [expr $i / $ltotalrow] == 3 } {
      lappend loutput([expr $i % $ltotalrow + $rowcount($ip)],$ip) [lindex $result $i]
    } elseif { [expr $i / $ltotalrow] == 4 } {
      lappend loutput([expr $i % $ltotalrow + $rowcount($ip)],$ip) [lindex $result $i]
    }
  }
  incr rowcount($ip) $ltotalrow
  #writeln "now rowcount variable, total row for $ip = $rowcount($ip)"
  #reach manimum count
}

##
## Command to Create Jobs used to Monitoring of the concerning
## hosts in the map.
##

proc "Set Event Monitor" { list } {
  global color time first

  "Check View" $list
  set node [lindex $list 0]
  set ip [GetIpAddress $node]

  set res ""
  ForeachIpNode id ip host $list {

    if { [ined color $id] != $color(OK) } {
      ined acknowledge "Unable to Contact to $host Agent"
      continue
    }
    set mylist [list [ined retrieve $id]]
    "Retrieve Session Key" $mylist
    set sesskey [ined attribute $id "sesskey"]
    set s [SnmpOpen $id $ip]

    set namelist ""
    if {[catch {$s walk x 1.3.6.1.4.1.4210.10.1.50.1.1 {
      lappend namelist [lindex [lindex $x 0] 2]
      }
      } msg] != 0} {
```

```
    ined acknowledge "Unable to Connect to $host"
    $s destroy
    continue
  }
  if {$namelist == ""} {
    ined acknowledge "Unable to Contact Agent"
    $s destroy
    continue
  }
  $s destroy
  set file_name(1) [lindex $namelist 3];set file_name(1) [sessdec $sesskey
$file_name(1)]
  set file_name(2) [lindex $namelist 4];set file_name(2) [sessdec $sesskey
$file_name(2)]
  set file_name(3) [lindex $namelist 5];set file_name(3) [sessdec $sesskey
$file_name(3)]

  set selectfile 1


  set interval 60
  set select [ined request "Please Fill in Information for creat Daemon" \
  [list [list "Host Name" $host request] \
  [list "FileName 1" $file_name(1) request] \
  [list "FileName 2" $file_name(2) request] \
  [list "FileName 3" $file_name(3) request] \
  [list "Monitoring file NO:" $selectfile radio 1 2 3] \
  [list "Monitor Syntax 1  :" "" request] \
  [list "Monitor Syntax 2  :" "" request] \
  [list "Monitor Syntax 3  :" "" request] \
  [list "Time Interval(sec):" "$interval" request] ] \
  [list "OK" "CANCEL"]]


  if { [lindex $select 0] == "CANCEL" } {
    continue
  }

  set filecode    [lindex $select 5]
  set argument1   [lindex $select 6]
  set argument2   [lindex $select 7]
  set argument3   [lindex $select 8]
  set interval    [lindex $select 9]
  if {$first == 0} {
    set time [expr $interval * 950]
    createjob
```

```
        set first 1
    }
## encoding part
    set filecode    [expr $filecode + $sesskey]
    set argument1   [sessenc $sesskey $argument1]
    set argument2   [sessenc $sesskey $argument2]
    set argument3   [sessenc $sesskey $argument3]
    set interval    [expr $interval + $sesskey]
    set creat       [expr 1 + $sesskey]

## real setting through snmpd
    set s [SnmpOpen $id $ip]

## set filecode
    if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.50.2.1.0 \
      $filecode]]} msg] {
        error $msg
    }
## set syn1
    if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.50.2.2.0 \
      $argument1]]} msg] {
        error $msg
    }
## set syn2
    if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.50.2.3.0 \
      $argument2]]} msg] {
        error $msg
    }

## set syn3
    if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.50.2.4.0 \
      $argument3]]} msg] {
        error $msg
    }
## set interval
    if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.50.2.5.0 \
      $interval]]} msg] {
        error $msg
    }
## set to create daemon
    if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.50.2.6.0 \
      $creat]]} msg] {
        error $msg
    }

    $s destroy
```

```
    ined acknowledge "Saving Configuration for $host Successfully"
  }
}



##
## Show Event "Command used to Show the Error EVENT captured by Monitoring
## program, data will be kept in "loutput(rownum,ip)" variable and the
## total row will be kept in "rowcount(ip)" variable
##
proc "List Event Monitor" { list } {
  global color
  "Check View" $list
  ForeachIpNode id ip host $list {
    if { [ined color $id] != $color(OK) } {
      ined acknowledge "Unable to Contact to $host Agent"
      continue
    }

    set mylist [list [ined retrieve $id]]
    "Retrieve Session Key" $mylist
    set sesskey [ined attribute $id "sesskey"]

    set s [SnmpOpen $id $ip]
    set count 0
    $s walk x 1.3.6.1.4.1.4210.10.1.50.3.1 {
      ##writeln "x = $x"
      incr count
    }
    if {$count == 0} {
      ined acknowledge "No Daemon currently running on $host"
      continue
    }
    set num_row [expr $count / 7]
    set one 1;set two 0;set three 0;set four 0;set five 0; set six 0; set seven 0
    set innercount 1
    $s walk x 1.3.6.1.4.1.4210.10.1.50.3.1 {
      if { $one == 1 } {
        if { $innercount == $num_row } {
          set innercount 1; set one 0; set two 1
          continue
        }
        incr innercount
      }
      if { $two == 1 } {
```

```
  if { $innercount == $num_row } {
    set pspid($innercount) [lindex [lindex $x 0] 2]
    set pspid($innercount) [expr $pspid($innercount) - $sesskey]
    set innercount 1; set two 0; set three 1
    continue
  }
  set pspid($innercount) [lindex [lindex $x 0] 2]
  set pspid($innercount) [expr $pspid($innercount) - $sesskey]
  incr innercount
}
if { $three == 1 } {
  if { $innercount == $num_row } {
    set pstime($innercount) [lindex [lindex $x 0] 2]
    set pstime($innercount) [expr $pstime($innercount) - $sesskey]
    set innercount 1; set three 0; set four 1
    continue
  }
  set pstime($innercount) [lindex [lindex $x 0] 2]
  set pstime($innercount) [expr $pstime($innercount) - $sesskey]
  incr innercount
}
if { $four == 1 } {
  if { $innercount == $num_row } {
    set psfilename($innercount) [lindex [lindex $x 0] 2]
    set psfilename($innercount) [sessdec $sesskey $psfilename($innercount)]
    set innercount 1; set four 0; set five 1
    continue
  }
  set psfilename($innercount) [lindex [lindex $x 0] 2]
  set psfilename($innercount) [sessdec $sesskey $psfilename($innercount)]
  incr innercount
}
if { $five == 1 } {
  if { $innercount == $num_row } {
    set pssyn1($innercount) [lindex [lindex $x 0] 2]
    set pssyn1($innercount) [sessdec $sesskey $pssyn1($innercount)]
    set innercount 1; set five 0; set six 1
    continue
  }
  set pssyn1($innercount) [lindex [lindex $x 0] 2]
  set pssyn1($innercount) [sessdec $sesskey $pssyn1($innercount)]
  incr innercount
}
if { $six == 1 } {
  if { $innercount == $num_row } {
    set pssyn2($innercount) [lindex [lindex $x 0] 2]
```

```
            set pssyn2($innercount) [sessdec $sesskey $pssyn2($innercount)]
            set innercount 1; set six 0; set seven 1
            continue
          }
          set pssyn2($innercount) [lindex [lindex $x 0] 2]
          set pssyn2$innercount) [sessdec $sesskey $pssyn2($innercount)]
          incr innercount
        }
        if { $seven == 1 } {
          if { $innercount == $num_row } {
            set pssyn3($innercount) [lindex [lindex $x 0] 2]
            set pssyn3($innercount) [sessdec $sesskey $pssyn3($innercount)]
            set innercount 1; set seven 0
            continue
          }
          set pssyn3($innercount) [lindex [lindex $x 0] 2]
          set pssyn3$innercount) [sessdec $sesskey $pssyn3($innercount)]
          incr innercount
        }

      }
      $s destroy
      ##writeln "PSPID / PSTIME / PSFILENAME / PSSYN1 / PSSYN2 / PSSYN3"
      writeln
"#####################################################################"
      writeln "Process Currnetly running on $host"
      for {set count 1} {$count <= $num_row} {incr count} {
        writeln "PID=$pspid($count) INTERVAL=$pstime($count)\
 $psfilename($count) syn1=$pssyn1($count) syn2=$pssyn2($count)\
syn3=$pssyn3($count)"
      }
   }

}
##
## this procedure used to set the AUTH_LASREAD field
##
proc "Auth Last Read Setting" { list } {

   ForeachIpNode id ip host $list {

     #check for Availability
     color_host $id $ip $host flag

     #skip if the computer is not SNMPD compatible
     if { $flag != 0 } {
```

```
      continue
    }

    set s [SnmpOpen $id $ip]

    ##
    ## get data to display
    if { [catch { $s get 1.3.6.1.4.1.4210.10.1.1.2.0 } rowlastread] } {
      ined acknowledge "error SNMP process"
    }
    set rowlastread [lindex [lindex $rowlastread 0] 2]

    set result [ined request "Auth Row Number (Last Read)" \
       [list [list "ROW NUMBER" $rowlastread] ] \
       [list "set values" cancel] ]
    if { [lindex $result 0] == "cancel" } return
    set rowlastread [lindex $result 1]

    if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.1.2.0 $rowlastread]]} msg] {
      ined acknowledge "Fail set AuthLastRead"
    }
    $s destroy
  }
}


##
## Count the number of connections to a TCP service and display the
## result in a stripchart.
##

proc tcp_service_user { ids } {
  global cx

  set ids [MoJoCheckIds tcp_service_user $ids]
  if {$ids == ""} return

  foreach id $ids {
      set count 0
      foreach ipaddr $cx(addr,$id) {
        if {[catch {
            $cx(snmp,$id) walk x tcpConnState.$ipaddr.$cx(port,$id) {
              set tcpConnState [lindex [lindex $x 0] 2]
              if {$tcpConnState == "established"} { incr count }
            }
        }]} continue
```

```
        }
        ined -noupdate attribute $id "tcp established" "$count $cx(name,$id)"
        ined values $id $count
    }
}

proc start_tcp_service_user { ip id port } {
    global cx interval

    set cx(snmp,$id) [SnmpOpen $id $ip]
    set cx(port,$id) $port
    set cx(addr,$id) ""
    if [catch {netdb services name $cx(port,$id) tcp} cx(name,$id)] {
        set cx(name,$id) "port $cx(port,$id)"
    }

    if [catch {
        $cx(snmp,$id) walk x ipAdEntAddr {
            lappend cx(addr,$id) [lindex [lindex $x 0] 2]
        }
    } msg] {
        writeln "Can not read IP address table of host $ip: $msg"
        catch {$cx(snmp,$id) destroy}
        return
    }

    ined -noupdate attribute $id "tcp established" $cx(name,$id)
    ined -noupdate label $id "tcp established"

    set jid [job create -command [list tcp_service_user $id] \
            -interval [expr $interval * 1000]]
    save "restart start_tcp_service_user $interval $jid $ip \$$id $port"
}


##
## Set the parameters (community, timeout, retry) for snmp requests.
##

proc "Set SNMP/PGP Parameter" {list} {
    SnmpParameterlocal $list
}

##
## Set the parameters (community, timeout, retry) for snmp requests.
##
```

```
proc SnmpParameterlocal {list} {

   global snmp_community snmp_timeout snmp_retries snmp_window snmp_delay
   global snmp_port snmp_protocol snmp_context snmp_user
   global manager passphase

   set result [ined request "SNMP Default Parameter" \
         [list [list "Community:" $snmp_community entry 20] \
          [list "UDP Port:" $snmp_port entry 10] \
          [list "Timeout \[s\]:" $snmp_timeout entry 10] \
          [list "Retries:" $snmp_retries scale 1 8] \
          [list "Window Size:" $snmp_window scale 0 100] \
          [list "Delay \[ms\]:" $snmp_delay scale 0 100] \
          [list "Protocol:" $snmp_protocol radio SNMPv1] \
             [list "PGP Manager Name:" $manager entry 10] \
             [list "PGP Manager PassPhase:" $passphase entry 10] ] \
         [list accept cancel] ]

   if {[lindex $result 0] == "cancel"} return

   set snmp_community [lindex $result 1]
   set snmp_port      [lindex $result 2]
   set snmp_timeout   [lindex $result 3]
   set snmp_retries   [lindex $result 4]
   set snmp_window    [lindex $result 5]
   set snmp_delay     [lindex $result 6]
   set snmp_protocol  [lindex $result 7]
   set manager        [lindex $result 8]
   set passphase      [lindex $result 9]
}


##
## Set the default/or update jobs parameters for monitoring jobs.
##
proc "Set Monitor Parameter" {list} {
   global interval
   global default

   set result [ined request "SNMP-Security Monitoring Parameter" \
         [list [list "Interval \[s\]:" $interval entry 8] ] \
         [list "set value" cancel] ]

   if {[lindex $result 0] == "cancel"} return
```

```
    set interval [lindex $result 1]
    set default(graph) true

    if {$interval < 1} {
        set interval 1
    }
}

##
## Reset all the View show in the window
##
proc "Reset View" { list } {

    set list [ined retrieve]
    ForeachIpNode id ip host $list {
        ined color $id black
    }
}

##
## Check all the Node in the Map and set the color concerning with type
## of the NODE
## RED = unreachable
## WHITE = no compatible snmpd
## BLACK = reachable + compatible snmpd running
##
proc "Check View" {list} {
global color
    set list [ined retrieve]
    #writeln "list == $list"
    ForeachIpNode id ip host $list {
      ined color $id orange
      set flag [HostCheck $id $ip]
      if { $flag == 0 } {
        ined color $id $color(OK)
        continue
      } elseif { $flag == 1 } {
        ined color $id $color(NO_SNMPD)
        continue
      } elseif { $flag == 2 } {
          ined color $id $color(NO_COMPAT_SNMPD)
          continue
      } elseif { $flag == 3 } {
        ined color $id $color(UNREACHABLE)
        continue
      }
```

```
    }
    #endof(ForeachIpNode)
}

##
## Reset Event Monitor Job..
##

proc "Kill Event Monitor" {list} {

  set killall 0
  set code 0
  ForeachIpNode2 id ip host lcomp $list {

    set mylist [list [ined retrieve $id]]
    "Retrieve Session Key" $mylist
    set sesskey [ined attribute $id "sesskey"]
      set select [ined request "Type in Process ID to Terminate Monitoring" \
        [list \
        [list "Kill All Process  :" $killall radio 0 1] \
        [list "Process ID to kill:" "" request]] \
        [list "KILL" "CANCEL"]]

      set status    [lindex $select 0]
      set killall   [lindex $select 1]
      set code      [lindex $select 2]; set code [expr $code + $sesskey]
      if {$status == "CANCEL"} {
        break
      }

##if notKillall
      if {$killall == 0} {
        set s [SnmpOpen $id $ip]
        if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.50.3.2.0 \
          $code]]} msg] {
            error $msg
        }
        $s destroy
        ined acknowledge "Terminate Successfully"
        continue
      }
##if killall
      set s [SnmpOpen $id $ip]
      if [catch {$s set [list [list 1.3.6.1.4.1.4210.10.1.50.3.3.0 \
        [expr 1 + $sesskey]]]} msg] {
          error $msg
```

```
        }
        $s destroy
        ined acknowledge "Terminate Successfully"
        continue

    }
}

##
## Encoding with pgp program
##
proc encode {input our_name our_passphase receipt_name} {
    if {[catch [set fd [open "/tmp/input_plain_text" "w+"]] ] == 0} {
        ined acknowledge "Fail open/create input_plain new file"
        return "error"
    }
    puts $fd $input
    close $fd

    catch [exec pgp +batchmode +force {/tmp/input_plain_text} \
    -esa -z $our_passphase -o {/tmp/output.asc} \
    $receipt_name -u $our_name] mesg
    ##writeln "message = $mesg"

    if {[catch [set fd [open "/tmp/output.asc" "r"]] ] == 0} {
        return "error"
    }
    fconfigure $fd -blocking 0
    set buffer [read $fd]
    close $fd
    ##exec rm "/tmp/input_plain_text"; exec rm "/tmp/output.asc"
    return $buffer
}
##
## Decode with pgp program
##
proc decode {input our_passphase} {
    if {[catch [set fd [open "/tmp/input_enc_text" "w+"]] msg] == 0} {
        ined acknowledge "Fail open/create new file"
        return
    }
    puts $fd $input
    close $fd

    catch [exec pgp +force +batchmode -z $our_passphase \
    -o "/tmp/output" "/tmp/input_enc_text"] msg
```

```
  ##writeln "message = $msg"

  if {[catch [set fd [open "/tmp/output" "r"]] msg] == 0} {
    return "error"
  }

  if {[catch [fconfigure $fd -blocking 0] mesg] != 0} {
    return "error"
  }
  set buffer [read $fd]
  close $fd
  exec rm /tmp/output; exec rm /tmp/input_enc_text
  return $buffer
}



##
## Retrieve Session Key..
## This proc used to get the session key of the given server.
##
proc "Retrieve Session Key" {list} {

  global passphase manager
  ForeachIpNode id ip host $list {

    set pgp [ined attribute $id "pgp-name"]
    if {[catch {set sockfd [socket $ip 20000]} mesg]
    != 0} {
      ined acknowledge "Fail connect to \"$host\"  \"$mesg\""
      continue
    }
    set buffer [encode "GETCODE" $manager $passphase $pgp]
    puts $sockfd  $buffer
    flush $sockfd

    set buffer ""
    fconfigure $sockfd -blocking 0
    while {$buffer == ""} {
      set buffer [read $sockfd]
    }
    close $sockfd

    set buffer [decode $buffer $passphase]
    ined attribute $id "sesskey" $buffer
    if {$buffer == "error"} {
```

```
      ined acknowledge "Fail to Retrieve SessionKey from $host"
        continue
    }
    ##ined acknowledge "Successfully Retrieve SessionKey from $host"
    continue
  }
}

##
## Init passphase/manager value
##
proc init_pass_mana { } {
  global manager passphase

  set manager manager
  set passphase ""
}
init_pass_mana

##
## Request passphase for this manager
##
proc get_pass {} {
  global passphase
  set result \
  [ined request "Please Enter PassPhase" \
  [list \
  [list "" $passphase entry 15] ] \
  [list "OK"]]

  set passphase [lindex $result 1]
}
get_pass

## Trap Check
## data will be kept in the following form
## <IP> <FILE NUMBER> <SIZE>
proc trapcheck { } {
  global color
  set list [ined retrieve]
  set res [catch {set fd [open /tmp/trap  r]}]
  if { $res != 0 } {
    set fd [open /tmp/trap w]
  }
  while {[eof $fd] == 0} {
    gets $fd data
```

```
    set myip   [lindex $data 0]
    set myfile [lindex $data 1]
    set mysize [lindex $data 2]
    ForeachIpNode id ip host $list {
      if {$ip == $myip} {
        ined color $id $color(EVENT)
        ined request "Alert TRAP receive" \
        [list \
        [list "HOST Name  :" $host request] \
        [list "File Number:" $myfile request] \
        [list "File Size  :" $mysize]] \
        [list "OK"]]
      }
    }
  }
  close $fd
  set fd [open /tmp/trap w]
  close $fd
}

##
## Display some help about this tool.
##

proc "Help SNMP-Security" {list} {
   ined browse "Help about SNMP-Security" {
        "The SNMP-Security contains a set of utilities to monitor hosts" \
        "using SNMP requests." \
        "" \
        "Set Event Color:" \
        "    Set the color of each status of each node or agent." \
        "" \
        "Check View:" \
        "    Used to check all the status of each nodes or agents" \
        "    in the network." \
        "" \
        "Reset View:" \
        "    Used to reset all the network map on the screen to" \
        "    normal color status from the Set Event Color Menu." \
        "" \
        "Set SNMP Parameter:" \
        "    Setting the specification of SNMPv1 parameter" \
        "    consisting the community name, UDP port number" \
        "    and Time Out Setting." \
        "" \
        "Log File View:" \
```

```
        "   Monitoring all the log file. This menu consists of sub-menus" \
        "   as following below:" \
        "   - Set View Parameter:" \
        "     Setting the value of logfile number, enable filtering," \
        "     Filtering Syntax 1, Filtering Syntax 2, AndFlag and" \
        "     Update Event Only." \
        "   - Update Result (All):" \
        "     Updating all the parameters in every nodes or agents" \
        "     which are settled in the Set View Parameter" \
        "   - Update Result:" \
        "     Updating all the parameters in the selected node which" \
        "     is settled in the Set View Parameter" \
        "   - View Result:" \
        "     Show the result event according to Set View Parameter" \
        "     Setting." \
        "" \
        "Event Monitor:" \
        "   Setting the parameter of event that display on the monitor." \
        "   This menu consists of sub-menu as following below:" \
        "   - Log File Monitor:" \
        "     Display the size of log file by graphical." \
        "   - Set Event Monitor :" \
    "     Setting the parameter that want to be monitored." \
        "   - Show Event :" \
        "     Display the event according to the setting parameter." \
        "   - Reset Event :" \
        "     Reset all the events in the Show Event." \
        "   - Set Monitor Parameter :" \
        "     Set the retrieve interval time." \
        "" \
        "Log File Control:" \
        "   Allowing to control the log file by deleting all the" \
        "   information or indicate the number of row for starting" \
        "   display the inforamtion in the log file." \
        "" \
        "Job Control:" \
        "   Displaying and modifying the jobs within the network." \
    }
}

##
## Delete the menus created by this interpreter.
##

proc "Delete SNMP-Security" {list} {
```

```
    global menus monjob rowcount loutput alert_computer color pid

    ## unset variable
    if { [info exists monjob] } {
      unset monjob
    }
    if { [info exists rowcount] } {
      unset rowcount
    }
    if { [info exists loutput] } {
      unset loutput
    }
    if { [info exists alert_computer] } {
      unset alert_computer
    }
    if { [info exists color] } {
      unset color
    }

    if {[job info] != ""} {
          set res [ined confirm "Kill running monitoring jobs?" \
                [list "kill & exit" cancel] ]
          if {$res == "cancel"} return
    }

    DeleteClones
    ##"Reset View"
    set list [ined retrieve]
    ForeachIpNode id ip host $list {
       ined color $id black
    }

    foreach id $menus { ined delete $id }
    exit
}

##
## Monitor Job Info
##

proc "Job Info" { list } {
    MoJoInfo
}

##
## Modify the state or the interval of a running job.
```

```
##

proc "Modify Job" { list } {
   MoJoModify
}

set menus [ ined create MENU "SNMP-Security" \
            "Set View Parameter"  \
         "View Result" ""  \
            "Set Event Monitor"  \
            "List Event Monitor"  \
         "Kill Event Monitor" "" \
            "Reset Log File" "" \
         "Check View" \
         "Event Color Setting" "" \
         "Set SNMP/PGP Parameter"  \
            "Help SNMP-Security" "Delete SNMP-Security"  \
            ]
```