Secure SNMP

by

Eugene B. Belostotsky

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirement for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

February 6, 1997

Copyright 1997 Eugene B. Belostotsky. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce,
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author

Department of Electrical Engineering and Computer Science

February 6, 1997

Certified by

Michael Bove
Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Theses
Secure SNMP

by

Eugene B. Belostotsky

Submitted to the
Department of Electrical Engineering and Computer Science

February 6, 1997

In Partial Fulfillment of the Requirement for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This thesis proposes a user-based security mechanism for the Simple Network Management Protocol. The security mechanism is designed within the SNMP architecture. It protects against masquerade, modification of information, disclosure and message stream modification threats. The security mechanism uses public-key cryptography for key distribution, and private-key cryptography for user authorization, message authentication and encryption. The security mechanism is implemented within the WinSNMP architecture, and is transparent to the manager applications using WinSNMP API. The Mock Agent is used to simulate a real SNMP agent supporting this security mechanism.

Thesis Supervisor: Michael Bove
Title: Associate Professor
I thank Professor Michael Bove for his willingness to supervise my thesis on long-distance basis.

I thank Eclipse group of Printer Business Unit of Digital Equipment Corporation for their support. I also thank DEC for providing me with the fellowship.

Most of all, however, I want to thank my parents, Boris and Larisa, for bringing me to this wonderful country and for pointing me in the right direction so many times. They supported me throughout my years at MIT. I would not have gotten to this point without them.
Table of Contents

1 Introduction .................................................................................................... 6
2 Background ..................................................................................................... 9
  2.1 The Simple Network Management Protocol .................................... 9
  2.2 WinSNMP ............................................................................................... 12
  2.3 Security ................................................................................................. 13
3 The SNMPv2z Design .................................................................................. 16
  3.1 Security Threats ..................................................................................... 16
  3.2 Goals of Security Mechanism ............................................................... 19
  3.3 High-level Model ................................................................................... 19
  3.4 Security Algorithms in the SNMPv2z ................................................. 20
  3.5 Secret Key Transfer Mechanism ............................................................ 23
4 The SNMPv2z Implementation .................................................................. 25
  4.1 Overall Picture ....................................................................................... 25
  4.2 The SNMPv2z Message Format ............................................................ 27
  4.3 The SNMPv2z Manager ........................................................................ 28
    4.3.1 The Microsoft Windows Programming Model ............................ 28
    4.3.2 The WinSNMP Programming Model ............................................ 29
    4.3.3 The SNMPv2z Within a WinSNMP implementation .................... 31
    4.3.4 Extensions to WinSNMP’s Local Database .................................. 33
    4.3.5 Security Enhancements to WinSNMP Implementation ................ 36
  4.4 SNMPv2z Agent .................................................................................... 39
    4.4.1 The SNMP Mock Agent ................................................................. 39
    4.4.2 The SNMPv2z Mock Agent ............................................................. 40
      4.4.2.1 The SNMPv2z MIB ................................................................ 40
      4.4.2.2 Authorized Users Table ......................................................... 42
      4.4.2.3 Security Enhancements to the SNMP Mock Agent ............... 43
5 Alternate Approaches .................................................................................. 44
6 Analysis ........................................................................................................ 46
  6.1 Existing Problems ................................................................................... 46
  6.2 Testing Strategy and Results ................................................................ 48
7 Conclusion ..................................................................................................... 49

Bibliography .................................................................................................... 51
List of Figures

Figure 1: SNMP Formats ........................................................................................................ 12
Figure 2: WinSNMP Architecture ......................................................................................... 13
Figure 3: Symmetric and Public-key Algorithms ................................................................. 15
Figure 4: Secret Key Transfer Mechanism ........................................................................... 23
Figure 5: SNMPv2z Community String ................................................................................ 28
Figure 6: WinSNMP Implementation for SNMP and SNMPv2z ........................................ 32
Figure 7: A Sample $NP\_WSNMP\_INI$ File ..................................................................... 34
Figure 8: SNMPv2z Additions to Local Database ............................................................... 35
Figure 9: Security MIB ....................................................................................................... 41
Chapter 1

Introduction

The work described in this thesis was done while working in Digital Equipment Corporation's Printer Systems Engineering group (part of Components and Peripherals division). Printer Systems Engineering group provides complete network printing solutions to corporate customers - printers and software needed to print to and manage them. I was working in the group that develops software to remotely manage DEC's network printers. To conduct actual communication with network printers we use the Simple Network Management Protocol.

The Simple Network Management Protocol (SNMP) is the industry-standard network device management protocol[1]. Its popularity lies in its simplicity (thus the name). The SNMP architecture contains a manager and one or more agents. Each agent has a database of relevant information, which can be queried or changed by the manager.
The simplicity of the SNMP lies in the fact that manager-agent communication requires connectionless service (for example, UDP on top of IP) and takes the form of very simple GET or SET requests[1].

Although the SNMP's simplicity is its main advantage, it is also "too simple" for some problems. One problem with the SNMP is that it is not secure - SNMP messages are sent cleartext so that anybody having access to the physical wires can catch and modify them. This is particularly unsettling when a message carries an important information that will significantly affect manager's or agent's behavior (in the case of GET and SET requests, respectively). Potential harmful effects depend on a particular agent, but may include reconfiguring a router, changing default paper tray in a printer, and many others.

There have been several proposed security mechanisms for the SNMP, but none has been accepted unanimously by the industry so far. So, vendors of network printers and management software use proprietary protocols to securely manage their printers from remote sites. This diversity of security models makes it impossible for a network management software to manage different network printers. The result is that network administrators cannot use one network printer management software package to manage several different-brand network printers, which is exactly the goal of the project I was working on at DEC.

In this thesis I propose a security mechanism for the Simple Network Management Protocol (version of the SNMP using this mechanism is labeled SNMPv2z) that is relatively simple and has a clear relation to existing security models used in current
network environments. Although my immediate goal is to develop a security mechanism for managing DEC's network printers, I hope that my mechanism will be considered by the whole SNMP community.
Chapter 2

Background

In this chapter I present a high-level description of the Simple Network Management Protocol. Also presented is a brief overview of different types of security algorithms.

2.1 The Simple Network Management Protocol

Implicit in the SNMP architectural model is a collection of network management stations and network elements. Network management stations execute management applications which monitor and control network elements. Network elements are devices such as hosts, gateways, terminal servers, and the like, which have management agents responsible for performing the network management functions requested by the network management stations[9]. The Simple Network Management Protocol is used to
communicate management information between the network stations and the agents in the network elements. Consistent with SNMP being "simple", the exchange of SNMP messages requires only an unreliable datagram service (e.g., UDP over IP).

Resources in the network may be managed by representing these resources as objects. Each object is, essentially, a data variable that represents one aspect of the managed agent. An object can contain an information of any type, as long as the information conforms to a standard (ASN.1) encoding rules. Among the types of data available are integers, strings, nulls, and octet-streams. Complex types can be created by creating structures, including two-dimensional table, of these basic types. Other common types are also included in the standard, among which are the IP addresses and 32-bit integers[8].

Objects inside the agent are organized in a tree-like fashion, with a specific "address" assigned to each leaf of a tree (which stores an instance of an object). These addresses are known as Object Instance Identifier (OID), and whole tree itself is known as a Management Information Base (MIB)[10]. For example, the OID of the object describing the agent host system is 1.3.6.1.2.1.1.1. Most SNMP agents contain a standard MIB, known as MIB-2. This common MIB contains a standard data tree as set by the standards committee for SNMP and contains data such as the network peripheral name, its locations, and its contact person. Some branches of the MIB-2 are only standardized across systems of a particular class (e.g., a common set of objects is used for the management of network printers). One branch of the MIB-2 allows entry into the private realm. Companies, such as IBM, Hewlett-Packard, DEC, and many others, go to a central
location and ask for their own sub-branch of the private branch in the MIB-2. This scheme allows these companies to store their own private information. This is needed when a piece of data not existing in the MIB-2 standard is required by one of company’s products. When such a case occurs, the companies can simply add the missing information to their private branch, and since they know which OID is assigned to it (as it is assigned by the companies internally), they can reference it easily.

A management station performs the monitoring function by retrieving the value of MIB objects (by doing GET request). A management station can cause an action to take place at an agent or can change the configuration settings at an agent by modifying the value of specific variables (by doing SET request).

There are numerous other SNMP details, unimportant to my thesis, that are available from RFC 1155 [8], RFC 1157 [9], RFC 1213 [10]. One thing that is important is the structure of SNMP message. In Figure 1 you see the formats of generic SNMP message, SNMP request and SNMP response messages[2]. Most fields of the SNMP message are self-evident (after all, this is "simple" protocol), and detailed descriptions can be found in above RFCs. "Community" deserves special recognition; it is a relationship between an SNMP agent and a set of SNMP managers that defines message authentication and access control[2]. In other words, an agent can set up several communities, each with access to only part of the MIB (these parts may be overlapping). In theory, SNMP manager must use a specific community name to get an access to a specific part of the MIB. In practice, almost no one uses this mechanism for several reasons: there is no real message authentication or authorization (community name is sent cleartext)
mechanism[1]. In this thesis I propose another security mechanism which does not have above flaws.

<table>
<thead>
<tr>
<th>Version</th>
<th>Community</th>
<th>SNMP PDU (Protocol Data Unit)</th>
</tr>
</thead>
</table>

(a) SNMP message

<table>
<thead>
<tr>
<th>PDU type</th>
<th>Request ID</th>
<th>Error Status</th>
<th>Error Index</th>
<th>Variable Bindings</th>
</tr>
</thead>
</table>

(b) GetRequest, SetRequest PDU

<table>
<thead>
<tr>
<th>PDU type</th>
<th>Request ID</th>
<th>Error Status</th>
<th>Error Index</th>
<th>Variable Bindings</th>
</tr>
</thead>
</table>

(c) GetResponse PDU

<table>
<thead>
<tr>
<th>OID</th>
<th>value 1</th>
<th>OID 2</th>
<th>value 2</th>
<th>...</th>
<th>OID n</th>
<th>value n</th>
</tr>
</thead>
</table>

(d) Variable Bindings

*FIGURE 1: SNMP Formats*

### 2.2 WinSNMP

The Windows SNMP API specification defines a programming interface for network management applications running under the Microsoft Windows family of GUI/operating system products, enabling those applications to make use of a logically external SNMP engine or service layer[12]. WinSNMP provides a single interface to which application developers can program and multiple SNMP software vendors can conform. Figure 2 shows where WinSNMP fits in one possible scenario of end-to-end
SNMP connectivity from an entity acting in a manager role to an entity acting in an agent role.

Nothing in the WinSNMP specification attempts to dictate how an implementation will actually execute the communications process with remote entities. Possible approaches include the following: Embedded Stack, Proprietary Stack API, Windows Sockets API, Multi-Protocol API, RPC.

FIGURE 2: WinSNMP Architecture

2.3 Security

There has been a lot of research in cryptography in the last several decades, but most of it is theory. There are not that many security algorithms that survive in the real world, either due to their impracticality, logical errors or other real-world obstacles unforeseen in the laboratory environment[3]. For the practical problem, it is very important to choose an algorithm which provides adequate security for the particular problem, otherwise there might be too little security, which is almost the same as no
security at all, or too much security, which can take computational power from the
problem which needs security.

In addition to providing confidentiality, cryptography is often asked to do other
tasks. Security mechanisms can be asked to provide authentication services - it should be
possible for the receiver of a message to ascertain its origin; an intruder should not be able
to masquerade as someone else. Authentication services are often coupled with integrity
checking - it should be possible for the receiver of a message to verify that it has not been
modified in transit; an intruder should not be able to substitute a false message for a
legitimate one[5]. Security mechanisms can also be asked to provide authorization services
- enforcing access rights of authenticated users.

There are two different types of security algorithms: secret and public. Secret
algorithms are inadequate by today's standards: a changing group of users can not use
them because every time a user leaves the group everyone else must switch to a different
algorithm, if someone accidentally reveals a secret everyone must change their
algorithm[4]. Instead of securing an algorithm, modern cryptography keeps security in the
keys that are used by publicly known algorithms.

There are two types of key-based public algorithms (see Figure 3): symmetric and
public-key. Symmetric algorithms are algorithms where the encryption key can be
calculated from the decryption key and vice versa. In most symmetric algorithms, the
encryption and decryption keys are the same. These algorithms, also known as secret-key
algorithms, require that the sender and receiver agree on a key before they can
communicate securely. Public-key algorithms are designed so that the key used for
decryption cannot be calculated from the encryption key. These algorithms are called "public-key" because the encryption key can be made public - anybody (who knows public key) can encrypt a message but only a specific person with the corresponding decryption key (private key) can decrypt the message.

(a) Encryption and decryption with one (secret) key

(b) Encryption and decryption with two different (one can be public) keys

FIGURE 3: Symmetric and Public-key Algorithms

While there is a small number of different types of algorithms, there are a lot of different ways how these algorithms can be used to achieve a particular level of security in a specific situation. For example, a Coca-Cola secret shared between several people requires a different security mechanism than sending secure mail messages over the Internet. The job of the person developing a security mechanism for a commercial product is not to design security algorithms from scratch, but rather to design a security scheme that uses already existing algorithms in most cost-efficient and practical way.
Chapter 3

The SNMPv2z Design

This chapter presents the design of SNMPv2z manager and agent entities. First, I list common threats to a network management protocol. Based on the threats, I define several goals for my security mechanism. Finally, I present high-level model and the detailed design of the SNMPv2z security mechanism.

3.1 Security Threats

Several of the classical threats to network protocols are applicable to the network management problem and therefore are applicable to any SNMP security model. The principal threats against which SNMP security mechanism should provide protection are[3]:

...
1. **Masquerade**

The masquerade threat is the danger that management operations not authorized for some user may be attempted by assuming the identity of another user that has the appropriate authorizations.

2. **Modification of Information**

The modification threat is the danger that some unauthorized entity may alter in-transit SNMP messages generated on behalf of an authorized user in such a way as to effect unauthorized management operations, including falsifying the value of an object.

4. **Disclosure**

The disclosure threat is the danger of eavesdropping on the exchanges between managed agents and a management station.

4. **Message stream modification**

The SNMP protocol is typically based upon a connectionless transport service which may operate over any subnetwork service. The re-ordering, delay, replay of messages can and does occur through the natural operation of many such subnetwork services. The message stream modification threat is the danger that messages may be maliciously re-ordered, delayed or replayed to an extent which is greater that can occur through the natural operation of a subnetwork service, in order to effect unauthorized management operations.

All security threats described above can be aimed towards either a manager or an agent (agents are affected by SET requests and managers are affected by GET responses).
However, the SNMP security means much more to an agent than to a manager. The reason is that the SNMP philosophy assumes that agents reside in various network elements, some of which do not have a lot of RAM or computational power. This is why SNMP agents are supposed to be very simple, in terms of their memory and computational requirements. An SNMP agent is a passive entity - it does exactly what SNMP request asks it to do, without any consideration for consequences of these actions. For example, a SET request that results in the router being rebooted can be a disaster for a whole LAN, but SNMP agent will perform this action mindlessly. This is why we need to make sure that only authorized manager entities can send such radical (or any) SET requests.

A manager application, on the other hand, usually resides in a workstation or a PC. Thus, it does not have memory or computational limitations that an agent does. Hopefully, a manager is programmed such that it will not do anything radical without considering its options, which may include alerting network administrator or some other responsible party.

Because an SNMP agent is a “helpless” entity and a manager application can be made very powerful, my security mechanism is geared toward protecting an agent and assumes that incoming requests may be created or modified by a hostile manager entity. Later, we will model an agent as a server and a manager as a client. From this point of view, it is also much more important to protect a server (SNMP agent) than a client (SNMP manager).
3.2 Goals of Security Mechanism

Based on the foregoing account of threats in the SNMP network management environment, the goals of this SNMP security model are:

1a. To provide user authorization service such that an agent’s MIB is accessible only to authorized management entities.

1b. To provide verification of the identity of the user on whose behalf a received SNMP message claims to have been generated.

2. To provide verification that each received SNMP message has not been modified during its transmission through the network.

3. To provide that the contents of each received message are protected from disclosure. This may not be necessary in all situations (for example, there is no secret data passed to/from a network printer).

4. To provide protection against delayed or retransmitted SNMP message that can result in unauthorized management operation.

These goals should be achieved without complicating the protocol too much. In other words, our supreme goal is to leave SNMP simple.

3.3 High-level Model

Leaving the SNMP simple means that a security mechanism should be simple, otherwise the complexity of security mechanism will deter network industry professionals.
from using SNMPv2z at all. I view SNMPv2z architecture in the context of the prevalent computing model - client/server.

The basic model consists of one (or more) servers and one (or more) agents who need to access information stored in a server. There is a limited number of users who are allowed to log into the server. The list of authorized users is kept in securely-held password file, which is maintained by a system administrator. To log into the server a user needs to supply a username and a password. My assumption is that this model is very familiar to any computer user/programmer and that its use in SNMPv2z will not complicate SNMP too much.

In SNMPv2z realm, a server is an agent and a client is a management station. An agent keeps a password file of all users who can access its MIB through SNMPv2z. To send SNMPv2z requests a user needs to run a management application, supplying his/her username and the password (which will be checked against agent's password file). This looks very similar to the client remotely logging into the server - a client/server (in this case manager/agent) model which is very popular in today's computing environments.

### 3.4 Security Algorithms in the SNMPv2z

With above goals in mind, I propose a new security mechanism for the SNMP. Considering that SNMP should be simple, I will adopt existing SNMP administrative infrastructure (as opposed to much more complex administrative infrastructure proposed by several experimental specifications of next generation of the SNMP)[11]. The
advantage of this approach is that SNMPv2z entities will be downward compatible - able to deal with both the SNMPv2z and the SNMP.

To support user authorization each SNMPv2z agent contains a password table of authorized users - these are the users who are allowed access to agent's MIB. Before any communication between a user (acting as a manager) and an agent commences, a user must be authorized, through a mechanism described later in the paper.

To support data integrity I will use a message digest algorithm. A digest is calculated over an appropriate portion of the SNMP message and included as part of the message sent to the recipient. Recipient recalculates the digest (using the same message digest algorithm) and compares the result to the digest included in the message (calculated by the sender). If both digests are the same then the message has not been modified while it was traveling over the network. This mechanism, however, does not tell the recipient who the sender is -- anybody who knows the algorithm (and it is known to public) can send a message to the SNMP agent. It also means that any sniffer who knows the digest algorithm can catch the message, change the data included in the message, recalculate the digest and insert it into the message in place of the old digest. In this scenario, the receiver would not have any clue that a message has been tampered with.

To support message authentication and data integrity a secret key (digest secret key) is inserted into SNMP message prior to computing the digest. The secret key is shared by both sender and receiver. After the message arrives, a receiver inserts the secret key into it, calculates the digest and compares it to the digest calculated by the sender. If both digests are equal then the message is authenticated and data integrity is assured. The
scenario described at the end of last paragraph is impossible in this case because the sniffer does not know sender's secret key. However, a sniffer can still catch a message and look at the data included in the message (since the data is passed clear-text).

To support confidentiality, I will use a standard data encryption algorithm (DES). DES is a symmetric algorithm, which requires a secret key (DES secret key). I could use the same secret key for both digest and DES algorithms, but having two separate keys leads to a more secure system.

Avoiding message stream modification requires efforts by both a manager and an agent. I propose a very simple strategy that requires little overhead. My strategy takes advantage of the SNMP specifications not specifying how a manager should choose request IDs for the SNMP messages; most SNMP implementations use request Ids to differentiate between different SNMP messages. In SNMPv2z, a manager uses increasing request IDs, starting from one. An agent saves the request ID of every message it processes from a particular user. To check for message stream modification, an agent checks the request ID of the current message against the request ID of the last message from this user. If current request ID is greater than old request ID then agent processes the message as usual. If the opposite is true, however, then agent knows that message stream modification has occurred and the message is discarded.
3.5 Secret Key Transfer Mechanism

As you see above, both digest and encryption algorithms I use are symmetric - both sender and receiver must know the secret keys. Barring manual intervention, the secret keys have to be transferred over the network. Unlike Kerberos and other systems with trusted third party, SNMPv2z entities must negotiate secret keys themselves. A manager, prior to querying an agent, has to make up secret keys and somehow send them to the agent through a secure channel. This can be accomplished by using public-key cryptography: a manager uses agent's public key to encrypt a message (containing digest and DES secret keys) that only this particular agent can decrypt. Figure 4 describes how manager starts secure communication with an agent.

(1) Manager requests Agent's public key.

\[ \text{Manager} \rightarrow \text{Agent} \ \{\text{GET Public Key}\} \]

(2) Manager generates random digest and DES secret keys, encrypts them using Agent's public key, and sends them to Agent.

\[ \text{Manager} \rightarrow \text{Agent} \ \{\text{SET (Secret Keys)}_{ PublicKey}\} \]

Agent decrypts Manager's secret keys using its private key.

(3) Both Manager and Agent use the secret keys for user authorization and message authentication.

*FIGURE 4: Secret Key Transfer Mechanism (response messages are not shown)*
Using public-key cryptography for key distribution and private-key cryptography for encryption/authentication is popular in commercial products. Digital Equipment Corporation used similar strategy in designing Digital Internet Tunnel. This product allows remote PCs and servers to connect to private networks via the Internet as if they were connected by a single secure wire.
Chapter 4

The SNMPv2z Implementation

This chapter presents Windows implementation of SNMPv2z manager and agent entities. After SNMPv2z message format is discussed, I describe my implementation of SNMPv2z manager within WinSNMP programming model. Finally, I discuss how an existing SNMP agent can be modified to serve as SNMPv2z agent.

4.1 Overall Picture

To show the feasibility of my design I need to implement two separate entities: SNMPv2z manager and SNMPv2z agent. The implementation of these entities very much depends on the development platform. Since my project in DEC is being developed for Windows environment (Windows NT and Windows 95), I had to develop SNMPv2z entities using Microsoft's Visual C++ for Windows.
Since my development time was limited, I decided to focus on the main theme of my thesis - the SNMP security - and try to use as many ready-to-use pieces (whose purpose is not central to my thesis) as possible. There are three ready-to-use libraries/packages that I use:

1). WinSNMP library

This is a library that provides WinSNMP API to a manager application using it. This library takes care of low-level details of SNMP such that the application only worries about building SNMP message and then WinSNMP library does the rest (sending the message, receiving it, notifying application about received message, etc). DEC licenses WinSNMP library from American Computer Company (ACE*COM).

In the course of choosing SNMP low-level library I realized that WinSNMP standard is widely-used in the SNMP management applications developed for Windows platform. In order for my security mechanism to be successful I need to make it transparent to management applications conforming to the WinSNMP standard; in other words, I need to build my security mechanism into the WinSNMP implementation.

2). RSA library

This library implements all standard security protocols: RSA, DES, MD5, etc. It is provided for free by RSA Inc. to be used for educational purposes. I decided to use this library to be sure that I am using tested implementations of security algorithms.

3). The SNMP Mock Agent

One of my colleagues at DEC developed SNMP Mock Agent for Windows platform. Its original purpose was to be a testing tool for the SNMP manager.
applications. I decided to use it because it gives me the base on which to build SNMPv2z Mock Agent.

4.2 The SNMPv2z Message Format

My primary goal in designing SNMPv2z is to make it simple. Practically, it means that it has to be compatible with the SNMP, which in turn means that I have to use SNMP message format. The problem is that SNMPv2z requires some extra information (for example, message digest) to be passed in each message. The only solution is to pass extra information as part of an existing field of the SNMP message. A good candidate for an "expandable" field is the community string: it is part of every message, it does not have a specific length and it can contain any characters (even unprintable ones).

Figure 5 shows the components of the community string in SNMPv2z. The first byte denotes if OIDs' values are encrypted using RSA (public-key algorithm), DES (private-key algorithm) or not encrypted at all. Note that SNMPv2z always calculates the message digest by default. An agent's ID is passed in the second byte of the community string. Third byte stands for the length of person's username on whose behalf the message is being sent. The next series of bytes contains the username itself. Sixteen bytes of message digest follows (size of digest is fixed by MD5 - message digest algorithm I use). Finally, to be downward compatible we need to pass an original community string. It is used by SNMPv2z agent for the same purpose that it is used by SNMP agent - to specify access rights to specific part(s) of agent's MIB.
4.3 The SNMPv2z Manager

After describing Windows and WinSNMP programming model, I discuss how SNMPv2z security mechanism is implemented within WinSNMP API.

4.3.1 The Microsoft Windows Programming Model

The Microsoft Windows environment model provides for both synchronous (function call does not return until it finishes its work) and asynchronous (applications are "driven" by the receipt and processing of asynchronous message events, like user clicking a mouse) programming model. While most of WinSNMP API is synchronous, one very important part of it is not. In order for WinSNMP library to be able to process more than one SNMP message at a time, it cannot stall after a message is sent and before a corresponding response is received. However, it has to have a way to notify an application when the response is finally received. Fortunately, Windows was designed for this programming model. All SNMP manager application has to do is to pass a window's (which it creates for GUI purposes) handle and a specific message ID to the WinSNMP...
library. When WinSNMP library wants to notify the application that an SNMP response has been received it sends the message to the window created by the application.

4.3.2 The WinSNMP Programming Model

Since my security mechanism is built into a WinSNMP implementation, I will first describe the WinSNMP programming model. A session, a unit of resource and communications management, is used to manage a link between a WinSNMP management application and a WinSNMP interface implementation. A well-behaved WinSNMP application will use the session construct to logically organize its operations and to minimize resource requirements on the implementation.

When an application wants to construct a message it first asks WinSNMP implementation to create a resource for each of the message's parts. These resources are allocated inside a WinSNMP implementation's space and accessed by a management application via handles. We are particularly interested in two types of resources: network entities and community strings. They can be interpreted differently by a WinSNMP implementations, depending on the mode it is operating in. In a regular, UNTRANSLATED, mode a WinSNMP implementation expects a network entity to be an IP address, and community string to be any string that SNMP agent, hopefully, recognizes. In a user-friendly, TRANSLATED, mode a network entity is a nickname for an IP address (for example, "printer in my office" instead of 16.34.144.7), and a community string is nickname for a real community string (in case the agent defines some hard-to-remember community strings and managing applications wants to use easy-to-
remember strings). In the user-friendly mode, a WinSNMP application does the translation via the local database, whose implementation depends on a particular WinSNMP implementation. The ACE*COM's WinSNMP implementation uses .INI file format (standard Windows way to save text information between executions of a program) as its local database architecture.

As I discuss later in this chapter, SNMPv2z security mechanism operates in user-friendly, TRANSLATED, mode of the WinSNMP implementation. To understand how an SNMP manager application uses WinSNMP API in a user-friendly mode, I present the following sequence of steps outlining a “life” of a simple session.

1. Application calls SnmpStartup to attach WinSNMP dynamic library and make required WinSNMP-related initializations.

2. Application creates a window and defines a message ID, which when received by the window means that a response to SNMP request has been received by the WinSNMP implementation.

3. Application creates a session by calling SnmpOpen with the window's handle and the message ID defined above. A handle to newly-created session resource is returned. All functions related to communication within this session require the session's handle as an input argument.

4. Application calls SnmpStrToEntity passing it a user-friendly name of the agent network entity that it wants to query. A handle to a newly-created network entity resource is returned.
5. Application calls SnmpStrToContext passing it a user-friendly community string.
   A handle to newly-created community resource is returned.

6. Application builds the rest of the message by calling appropriate WinSNMP functions
   and passing it required handles.

7. Application calls SnmpSendMsg to send a message.

8. Application's window receives a message when the WinSNMP library receives a
   response to the message.

9. Application calls SnmpRecvMsg to get a message.

10. Application calls SnmpClose to close the session.

11. Application calls SnmpCleanup to detach SNMP dynamic library and free any
    remaining WinSNMP-related resources.

   Of course, many messages can be sent within a session's context; many sessions
   can exist within a window's context; many windows can exist within an application's
   context.

4.3.3 The SNMPv2z Within a WinSNMP Implementation

   Figure 6 shows how a management application is designed to interact with the
   WinSNMP library and how it actually happens in the SNMPv2z. To make the SNMPv2z
   look like the SNMP to a manager application I need to make my security mechanism
   transparent. If I had access to the source code for a WinSNMP implementation, I would
   modify it to include the security mechanism. However, I do not have this privilege. So, I
need a “fake” WinSNMP library which would intercept all WinSNMP calls, add security enhancements, and call ACE*COM’s WinSNMP library. This is exactly what SNMPv2z library does.

(a) SNMP Manager - WinSNMP interaction

(b) SNMPv2z Library Within WinSNMP Architecture

FIGURE 6: WinSNMP implementation for SNMP and SNMPv2z
4.3.4 Extensions to WinSNMP's Local Database

Network management protocols are rarely used interactively. More often, a manager application polls network entities (using GET request) and takes some action when unexpected results are received. An action might involve writing to a log file, displaying a warning message or sending a SET request to the network entity. If my security mechanism limited a managing application to be run interactively then SNMPv2z would be doomed from the start. On the other hand, for a manager application to be batch-processed, some configuration information may need to be stored before an application is ran.

It seems that the WinSNMP specification was designed with this idea in mind, thus the existence of a local database. Figure 7 shows a sample np_wsnmp.ini file - local database in ACE*COM's implementation of the WinSNMP. Note the structure of an entry in Entities section. For each user-friendly name the local database provides the SNMP version to be used with this agent, its IP address, timeout between retries, and number of retries (number of times a copy of a message is sent if no reply is received within a timeout).

[Entities]
; Entries look like this:
; name = version,ipaddress,timeout,retry
; Where:
; name is a "friendly" name for SNMP_TRANSLATED mode
; version is the SNMP version of the agent (only 1 for now)
; ipaddress is a standard 4-byte dotted string IP address
; timeout is the timeout interval in miliseconds
; retry is the retry count in units
hp4si_in_my_office = 1,16.34.144.7,3000,3
hp5si_in_my_office = 1,16.34.144.8,3000,3
[Contexts]
; Entries look like this:
; name = version,communityString
; Where:
; name is a "friendly" name for SNMP_TRANSLATED mode
; version is the SNMP version of the agent (only 1 for now)
; communityString is a text only community string in the agent
readall = 1,public
readwrite = 1,system

[MIB Variables]
; Entries look like this:
; name = oid, type, access, status
; Where:
; name is a "friendly" name for the object
; (usually right from the MIB, but can be anything)
; oid is the object or instance identifier
; type is a brief indication of the data type of the variable
; access is read-only (ro), read-write, not-accessible (na)
; status is mandatory (m), deprecated (d), obsolete (e)
sysDescr = 1.3.6.1.2.1.1.1.0,ds,ro,m
sysObjectID = 1.3.6.1.2.1.1.2.0,oid,ro,m

**FIGURE 7: A Sample NP_WSNMP.INI File**

SNMPv2z takes advantage of the local database and makes logical extensions to its structure. Figure 8 shows how the information is stored for security-enabled connections: section *Entities* is modified and a new section *Users* is added. In addition to SNMPv1 entities shown in Figure 7, section *Entities* now contains entities for SNMPv2z agent entities. SNMPv2z entities contain the same type of information as SNMPv1 entities plus several new pieces of data. The new pieces of data include: root of agent's security MIB (described later), a user's name on whose behalf the connection is to be established, and a flag indicating whether data sent to this agent should be encrypted or not. Also note that SNMP version is now "2z".
There are two ways to provide the password corresponding to the username. If the local database file is stored securely, then the password can be provided in Users section of np_wsnmp.ini (as is the case for user gene in Figure 8). Otherwise, there should be no password associated with a username in Users section, and in due time SNMPv2z will ask for the password interactively.
4.3.5 Security Enhancements to WinSNMP Implementation

Here is how the SNMPv2z library operates within the WinSNMP programming model described earlier in this chapter. SNMPv2z library intercepts all WinSNMP calls, does appropriate security enhancements (if any) and calls ACE*COM’s WinSNMP library for low-level SNMP implementation. Since the SNMPv2z is transparent, a manager application “thinks” that it calls the WinSNMP library directly.

Steps from section 4.3.2 are repeated here with SNMPv2z security enhancements following each step:

1. Application calls SnmpStartup to attach WinSNMP dynamic library and make required WinSNMP-related initializations.

   The SNMPv2z library creates and initializes to null its internal data structures:

   user password array and a linked list of authorized security-enabled connections.

2. Application creates a window and defines a message ID, which when received by the window means that a response to SNMP request has been received by the WinSNMP implementation.

   No WinSNMP calls are involved in this step - thus no security enhancements.

3. Application creates a session by calling SnmpOpen with the window's handle and the message ID defined above. A handle to newly-created session resource is returned.

   All functions related to communication within this session require the session's handle as an input argument.

   SnmpOpen call is directly routed to the WinSNMP library.

4. Application calls SnmpStrToEntity passing it a user-friendly name of the agent
network entity that it wants to query. A handle to a newly-created network entity resource is returned.

This is when user authorization happens. To authorize a user the SNMPv2z library sends a sequence of SNMP messages to the agent, thus taking a role of a manager application while authorization is taking place. The SNMPv2z library creates a new window and uses the WinSNMP library to send a sequence of three messages. This sequence of messages serves two purposes: to authorize the user with the given password and to pass secret keys from a manager to an agent (see previous chapter for the description of secret key transfer mechanism). Before the following sequence of messages is executed, the SNMPv2z library checks user password array for the user’s password. If it is not there, np_wsnmp.ini is checked. Finally, if all of the above fails, a user is prompted for his/her password interactively. The username and the password are added to user password array. asks for the user’s password if it is not in the np_wsnmp.ini file.

Message #1: GET request to receive agent’s ID and public key.

Message #2: SET request to send username, password and secret keys encrypted with agent’s public key.

Message #3: GET request to find out if user is authorized.

This message sequence is synchronous, and will result in error if any of the messages result in error. If the user is authorized, a linked list of authorized security-enabled connections is augmented with the information about the just established connection.

5. Application calls SnmpStrToContext passing it a user-friendly community string.
A handle to newly-created community resource is returned.

SnmpStrToContext call is directly routed to the WinSNMP library.

6. Application builds the rest of the message by calling appropriate WinSNMP functions and passing it required handles.

These calls are directly routed to the WinSNMP library.

7. Application calls SnmpSendMsg to send a message.

The SNMPv2z library checks if the connection is authorized. If it is, then the SNMPv2z community string is created using format described in section earlier in this chapter. Depending on the required security level, OID values may need to be encrypted.

8. Application's window receives a message when the WinSNMP library receives a response to the message.

No WinSNMP calls are involved in this step - thus no security enhancements.

9. Application calls SnmpRecvMsg to get a message.

SNMPv2z library checks if the connection is authorized. If it is, the SNMPv2z library performs message authentication and data integrity checks. Depending on the required security level, OID values may need to be decrypted.

10. Application calls SnmpClose to close the session.

SnmpClose call is directly routed to the WinSNMP library.

11. Application calls SnmpCleanup to detach SNMP dynamic library and free any remaining WinSNMP-related resources.

The SNMPv2z library deallocates its global data structures.
Note that if an agent does not support SNMPv2z then message #1 (in step 4) would return an error since neither agent ID nor public key exist in non-SNMPv2z agents. One other note: based on my assumption (in chapter 3) that SNMP agents are not hostile entities, there is no danger in passing them secret keys even if a user is not authorized.

4.4 SNMPv2z Agent

To test the security mechanism I need an SNMPv2z agent. Fortunately, I did not have to build SNMPv2z agent from scratch. Instead, I modified the SNMP Mock Agent, developed by my colleague, to be SNMPv2z compatible. The SNMP Mock Agent was developed in Microsoft’s Visual C++ environment for the Windows 95 and Windows NT platforms.

4.4.1 The SNMP Mock Agent

The reason for the creation of the SNMP Mock Agent is a lack of proper testing hardware/firmware for the SNMP management software. Since creation of the actual hardware takes a long time, a simulated testing environment is needed if management software is to be developed concurrently with the development of network entities that need to be managed. The idea of the SNMP Mock Agent is simple: the Mock Agent simulates a live SNMP agent across the network, programmed with the expected data.

The Mock Agent and the management program communicate via the network, just as the real SNMP agent would. The management program formulates and sends SNMP requests to the Mock Agent, and it expects a reply which the Mock Agent provides.
Mock agent logs all SNMP activity addressed to it in a special log window and in a special log file.

To be a really useful testing tool, the Mock Agent cannot be static; it should change its state, namely the values of some OIDs in its MIB. To achieve this level of programmability, the Mock Agent run from scripts crafted by the testing person. Because each testing situation is unique, it would not be feasible to force the testing person to modify the actual code for the Mock Agent for each test case. A very simple script language is therefore used to program the Mock Agent.

In order to run the SNMP Mock Agent, one needs to supply a name of the “main” script file. The script file defines the data in Agent’s MIB and may include other script files. Upon receiving SNMP requests from the network, the Mock Agent should examine its own data and respond correctly.

### 4.4.2 The SNMPv2z Mock Agent

This section describes how I transformed the SNMP Mock Agent into the SNMPv2z Mock Agent.

#### 4.4.2.1 The SNMPv2z MIB

An SNMPv2z agent needs to hold some security-specific information such as its public/private key, agent ID, secret keys of authorized users who are managing it. This information is saved in a new branch of MIB, which I call security MIB. For now, this
MIB is a part of the private MIB. If the SNMPv2z were to become a standard, then security MIB would exist in the standard MIB (i.e. MIB-2).

### Security MIB

- **X.1** Agent's ID (R)
- **X.2** Public Key (R)
- **X.3** Private Key
- **X.4** Placeholder for Username (W)
- **X.5** Placeholder for Password (W)
- **X.6** Placeholder for Digest Secret Key (W)
- **X.7** Placeholder for DES Secret Key (W)
- **X.8** Result of Authorization (R)
- **X.9** Authorized Users Table
  - **X.9.1** Entry
    - **X.9.1.1** Username
    - **X.9.1.2** Password
    - **X.9.1.3** Digest Secret Key
    - **X.9.1.4** DES Secret Key
    - **X.9.1.5** Encryption Flag
    - **X.9.1.6** IP Address
    - **X.9.1.7** Last Request ID

**FIGURE 9: Security MIB (access rights are in parenthesis)**

Figure 9 shows the structure of the security MIB. The agent ID and public/private key pair is generated when the agent is started. Each entry in the table of users includes: username and his/her password (set by system administrator by some method outside of SNMP's scope), digest and DES secret keys (set by message #2 in step 4 of the authorization sequence described in section 4.3.5), flag indicating if OID values sent to/from this user are encrypted or not, user's IP address (to differentiate the same user...
communicating from different manager entities) and request ID of the last message from this user.

Since so much information in the security MIB is secret, only some fields are accessible to outside network entities. The only two items that can be read are agent's ID and its public key. The only items that could be written are placeholders for username, password, and secret keys. The rest of the security MIB is not accessible to outside entities.

### 4.4.2.2 Authorized Users Table

The SNMP's specifications are such that an SNMP GET (SET) request message has to contain the exact "address" (OID) of the data queried (set). Moreover, an SNMP response message cannot contain anything but the data requested (set). This makes the SNMP simple, but it presents some problems for the SNMPv2z. To authorize a user, a manager application has to send his/her username and password to an agent. The only way to do that in the SNMP framework is to send them in a SET request. The problem is that a manager does not know their OIDs because it does not know where this user's entry appears (if at all) in the authorized users table.

The solution is to have placeholders for username, password and secret keys so that their exact "addresses" are known to any SNMPv2z manager. When they are set, the SNMPv2z Mock Agent is triggered to try to match username and password to one of the entries in the table. If there is no match, the placeholders are cleared and Result of Authorization is set to zero. Otherwise, Result of Authorization is set to this user's
digest secret key (so that only the right user can decode), and new entry is added to the authorized users table. The agent moves values from placeholders to the new table entry (clearing placeholders after that). It also sets Encryption Flag based on SNMPv2z community string, sets user's IP address and initializes Last Request ID to zero.

4.4.2.3 Security Enhancements to the SNMP Mock Agent

The only major modifications needed to convert an SNMP agent into an SNMPv2z agent is to add security filters for incoming requests and outgoing responses.

When the SNMPv2z Mock Agent receives a request message it looks at its version. If it is SNMPv1, then the SNMPv2z Mock Agent behaves exactly like the SNMPv1 Mock Agent. If it is SNMPv2z, then message integrity is checked by calculating message digest and comparing it to the message digest included in the request's community string. Depending on the user's security level, OID values may need to be decrypted. If a request message passes security filter, then Last Request ID is set to the request ID of current request.

Before SNMPv2z response message is sent back, the message digest is calculated and included in the community string. OID values may need to be encrypted depending on the security level of the particular user.
Chapter 5

Alternate Approaches

There are two major SNMPv2z design alternatives I had. The first one has to do with the choice of the type of message security algorithm. It does not have to be symmetric. However, public-key cryptography (the other choice) is much more computationally intensive than symmetric algorithms. This would burden agents because, as I discussed before, they are not designed to carry on heavy computational load. A small advantage of this approach is that there are no secret keys to distribute, thus there is no need for secret key transfer mechanism.

The choice and distribution of secret keys is the second major area where there are many design alternatives. There are two types of secret keys: long-lived and session. The secret key distribution mechanism depends on secret keys’ type. The SNMPv2z would not be very secure had it used long-lived secret keys. The reason is that a lot of time can
pass between the times when secret keys are changed. This means that a sniffer has all that time to try to figure out these secret keys. It does not mean that he will succeed, but he would have a higher chance of success that would be the case if the keys are changed more often. Long-lived secret key can be a user’s login password; they are distributed by system administrator. The problem with user’s login password is that users can not login into most SNMPv2z agents (for example, network printers). Long-lived secret keys can also be distributed manually. However, this approach is very burdensome since any time a new user wants to send an SNMPv2z message, or an old user want to send a message to a new SNMPv2z agent entity, a manual intervention of system/network administrator is required. Because of above problems I decided to use short-lived (session) secret keys, which are distributed by a mechanism within the SNMP framework.

A big part of the SNMPv2z Mock Agent design is the structure and the choice of storage for security-related information. I chose to structure and store it in the MIB framework because it has all the features we would need in a security database and it seems logical to use standard SNMP database, MIB, for the SNMP security mechanism. However, security information can be stored in an encrypted database stored separately from the MIB. The other possibility is to store users’ passwords in the password file, and to store the rest of the information in the MIB.
Chapter 6

Analysis

In this section I discuss problems that the SNMPv2z has, my testing strategy and results.

6.1 Existing Problems

The SNMPv2z security mechanism protects against classical threats that any network management protocol faces. However, it does not provide perfect security (if that exists). The hardest job of an engineer using cryptography is to choose a security mechanism/algorithm that is right given the level of security required. I think my security mechanism is more than adequate for the SNMP's security problems.

In theory, there is a clear way to make the SNMPv2z more secure. Note that currently the security mechanism does not provide user authentication service (it does
provide user authorization and message authentication). User authentication service would make sure that agent X responding to an SNMPv2z request is really agent X and not agent Y impersonating agent X. Currently, there is no way to prevent someone from impersonating an SNMP agent. The only reason someone may want to do that is to prevent an SNMPv2z agent from being managed. If users were authenticated, however, this threat would be avoided.

One solution that would protect against one entity impersonating another is to have a trusted third party which would certify the authenticity of users and network entities. The third party certifying agent is the heart of Kerberos security system. However, network architecture in which Kerberos is used is very different from the SNMP architecture. Kerberos is used in static (or very slowly changing) network environment where third party can really be trusted because it exists in the safe hands of Athena administrators. The SNMP architecture may be similar to Athena architecture, but it also may be very different: new network entities added/removed often and no trusted third party exists. Thus, although user authentication is a theoretical possibility, it would be very hard to implement it for a generic SNMP architecture.

There are some other minor problems with the SNMPv2z. Currently, I assume that request IDs can increase infinitely. For all practical purposes, this is not a problem because size of request ID field is not limited in any way by the SNMP specifications and depends on an SNMP implementation. However, in the future there should be a mechanism to inform SNMPv2z agents of request ID count being restarted at one.
Currently, two (or more) authorizations proceeding at the same time may cause an error because there is only one place where authorization's result is written. So, even if an authorization succeeds, its result may be overwritten by some other authorization going on at the same time. The SNMPv2z gets around this problem by re-executing the authorization sequence (after a brief timeout) if an error occurs.

My implementation does not allow the same user establishing two sessions with the same agent from the same manager. This is rarely needed, and the solution to this problem would mean significantly complicating the security MIB.

6.2 Testing Strategy and Results

It is very hard and time consuming to test how secure a security mechanism is. The reason is that it is impossible to prove that a security mechanism works; it is only possible to guess that it works because it has not been broken. Fortunately, I do not have to test security of my algorithms because I used original RSA algorithms.

The biggest test I had to perform is to show that my security mechanism works within the SNMP framework. My testing environment included several SNMP manager applications using WinSNMP, SNMPv2z Mock Agent and several SNMP agents. By using my implementation of WinSNMP, which uses SNMPv2z, manager applications were able to manage SNMP agents using SNMP and SNMPv2z agent using SNMPv2z. In other words, the security mechanism was transparent to manager applications, which was one of the goals of this project.
Chapter 7

Conclusion

As the Simple Network Management Protocol is used more and more widely in various network environments, the need for the secure SNMP becomes more and more apparent. Industry experience shows that as SNMP agents become more complex, companies limit their use of the SNMP, especially SET operations, because of its lack of security. To correct this problem, many companies implement proprietary security mechanisms, thus solving their own problems, but hindering chances of some security algorithm becoming the industry standard. In this thesis, I propose a simple security mechanism that fits well within the SNMP architecture. Although, it was designed for use by the Digital Equipment Corporation, I hope that it will serve as one of the prototypes considered for the standard security mechanism.

I think that there are (at least) three conditions/restrictions that need to be observed in a successful SNMP security mechanism: the security mechanism should be
within the SNMP architecture, it should be *simple*, and there is no trusted third party that could be used as user certifying agent. Given these conditions, I designed the security mechanism that uses public-key cryptography for key distribution and private-key cryptography for message authentication and user authorization. The SNMPv2z also provides message encryption service and protects against message stream modification.

To simplify the transition of existing SNMP manager applications to the SNMPv2z, my security mechanism needs to be transparent to manager applications. I implemented the security mechanism within the WinSNMP implementation, such that manager applications using WinSNMP API do not have to be rewritten.

I modified existing SNMP Mock Agent to serve as SNMPv2z Mock Agent. One major modification was the addition of the security MIB. I successfully used SNMPv2z with existing WinSNMP manager applications communicating with the SNMPv2z Mock Agent.

If I were starting this project over again, I would not have designed the SNMPv2z or my experiments differently. There is one thing, however, that I would have done differently - I would have used WinSNMP package with public source code. This would simplify SNMPv2z implementation because there would be no need for fake WinSNMP library; I would truly build my security mechanism into the WinSNMP implementation.
References


