SNMPv3 Working Group
INTERNET-DRAFT
draft-freeman-snmpv3-sec-00.txt, .ps

Stephen Borbash, Bill Freeman
Douglas Maughan
November 19, 1998

Simpler and More Secure Architectures for SNMPv3

Abstract

This document presents simpler and more secure architectures for SNMPv3 agents than the ones specified in RFCs 2271-2275. Agent security is improved by restricting each module's access to data, using the "principle of least privilege". The new agent architectures are analyzed in terms of software complexity as well as security, and are shown in some respects to be simpler.

Status of this memo

This document is being submitted to the IETF Simple Network Management Protocol version 3 (SNMPv3) Working Group for consideration as potential security architectures for SNMPv3. Comments are solicited and should be addressed to the authors and/or the SNMPv3 working group mailing list at `snmpv3@tis.com`.

Contents

List of Figures

List of Tables

1 Introduction

The RFCs 2271 through 2275 describe the new version of the simple network management protocol (SNMPv3). Unlike many networking protocols, these documents address not only what the message formats on the wire must be, but also what the software architecture must look like on the managers and agents.

The RFCs propose a certain software architecture, so they are open to debate about whether the architecture is secure. A software system providing security services must be carefully designed to prevent critical data from being compromised. SNMPv3 is such a system; it processes data that must be protected, along with the information needed to provide this protection.

In SNMPv3, certain modules in the software architecture must have access to sensitive data. For example, the module that performs encryption of the data must necessarily have access to plaintext as well as the encryption keys. There are other modules that do not need access to sensitive data. An example of this would be the dispatcher in the SNMPv3 agent. The dispatcher simply passes SNMP messages between modules, but does not perform any security related services itself.

According to the principle of least privilege, processes which do not need access to sensitive data should not have it. The complexity of the modules that handle sensitive data must be minimized. If the system is designed correctly, these modules are the only ones that can defeat the security services that are provided. Of course, any module that has access to any data in the system can prevent the system from functioning, but preventing denial of service is not a requirement of SNMPv3.

The goal of this document is to improve the software architecture for SNMPv3 by removing trust from modules which don't need to be trusted. We first examine how the RFC2271 architecture works. We illustrate several security problems which can arise from that architecture. Then we propose two new agent architectures, which solve some of the problems by reducing the access of several modules to sensitive data. Along the way we discuss the security benefits and concerns with our proposed changes.

2 RFC2271 Architectural Highlights

Figure 1, which is taken from RFC2271, shows the flow of data in the SNMPv3 agent. Each block is a module of the agent's implementation, and the arrows show which modules communicate with each other.

SNMPv3 aims to provide access control to the MIB (management information base) and provide some means of confidentiality to the SNMP PDU (protocol data unit) as it traverses the network. We briefly describe each module's responsibility:

The network is an interface to the User Datagram Protocol, or some other transport protocol.

The dispatcher module acts as a traffic cop; it delivers SNMP outgoing messages down to the network, and correctly routes incoming SNMP messages first to the right (to the message processor and security), then upward to whichever application is supposed to handle it.

The message processor is a stateful module; it remembers what requests this agent has or has not responded to, and decides whether messages need to be forwarded to the security module.

The security module is responsible for verifying signatures and decrypting incoming SNMP messages, and

```
 --------------------------------------------------------------------
|                                                                    |
|                                MIB                                 |
|____________________________________________________________________|
     ^                                  ^
     |          ---------------         |
     |         | SNMP Engine   |        |
     |          ---------------         |
     |         |               |        |
     v                                  v
 -----------   |  ---------    |  --------------      -----------
|           |    |         |     |              |    |           |
|  COMMAND  |  | |  ACCESS |   | | NOTIFICATION |    |   PROXY   |
| RESPONDER |    | CONTROL |     |  ORIGINATOR  |    | FORWARDER |
|___________|  | |_________|   | |______________|    |___________|
      ^                                 ^                  ^
      |        |               |        |                  |
   ---|--------                  -------|------------------|-----
  |   |                                 |                  |     |
      |                                 |                  |
  |   |    -----------------------------|                  |     |
      |   |                                                |
  |   |   |    --------------------------------------------|     |
      |   |   |
  |   v   v   v                                                  |
     ------------     ------------------     ----------------
  | |            |   |                  |   |                |   |
    | DISPATCHER |<->|     MESSAGE      |<->|    SECURITY    |
  | |____________|   |    PROCESSOR     |   |________________|   |
          ^          |__________________|
  |       |                                                      |
   -------|------------------------------------------------------
          |
          v
 --------------------------------------------------------------------
|                                                                    |
|                              NETWORK                               |
|____________________________________________________________________|
```

Figure 1: SNMPv3 Agent Architecture

signing and encrypting outgoing messages.

There are several application modules. The command responder responds to Gets and Sets. The notification originator forwards trap messages (which are typically generated by conditions in the operating system). The proxy forwarder application, which is not on all agents, forwards SNMP requests to devices on the network which dont speak SNMP.

The access control module is used to determine if a particular request for MIB access should be granted based primarily on who is accessing what and at what securityLevel. This module is supposed to be called by each application before accessing the MIB.

The MIB consists of variables which are being managed. Some examples are the IP routing table, the time since the last reboot, and the number of open connections. The ability to alter variables in the MIB without strong authentication was a weakness of SNMPv1, and one of the main reasons for SNMPv3.

For Get and Set messages, the basic protocol of SNMPv3 is as follows:

1. A command originator application on a manager sends a Get or Set PDU to a command responder application on an agent.

2. This PDU is protected in transit from replay, modification, and disclosure (by methods we don't discuss here).

3. The arriving PDU is authenticated and decrypted, then checked for whether the originator is allowed to perform the operation requested.

4. A signed and encrypted response PDU is sent back with data or an error code.

Now we examine in more detail what happens at the agent when the manager sends a Get, Getnext, Getbulk or Set Request. In order to concretely discuss our concerns, we have created two detailed figures, How an agent receives a request, and How an agent generates a response from the RFCs. These are displayed in Figures 2 and 3.

3 Security Concerns With the RFC2271 Architecture

As we can see from Figures 2 and 3, the various modules at the agent-device have access to various data which they receive as parameters to function calls. Much of this data, if changed by a module, could subvert the security that SNMPv3 is supposed to provide. For example, in item 12 in Figure 3, the dispatcher makes a call prepareOutgoingMessage() to the security module. The dispatcher can manipulate any of the parameters in this call, without fear that the security module will ever find out. In Table 1, we enumerate the different security-relevant variables which the five modules can touch. The ability to modify or ignore a variable without being detected and causing undesirable behavior is denoted by "yes". The ability to modify a variable without ill effect, or the inability to modify a variable, is denoted "no".

The ability to change these variables equals control over critical security information. Every module in the RFC2271 architecture has the ability to thwart the security that SNMPv3 is supposed to provide. In particular, the dispatcher and message processor have a surprising ability to cause problems. Note that a module doesn't have to be malicious to subvert security, only miscoded. Specific threats are described below.

```
                ^          8.success or errorIndication
  9.goes to MIB  |          ---------------------
                |         /                     \
                |        /                       \
                |       V                         \
   -----------------------------              ----------------
  |  Command Responder applic'n |             | Access Control |
  |_____________________________|             |________________|
     ^                  \                            ^
     |                   \                          /
     |                    \                        /
     |                     ----------------------
     |                    7. isAccessAllowed()
     |
     |
     | 6. processPDU()
     |
     |
     |
     |
     |     5.success or errorIndication  4.success or errorIndication
     |       -----------------            ---------------------
     |      /                \           /                     \
     |     /                  \         /                       \
     |    V                    \       V                         \
  ------------              -------------------             ----------
 | Dispatcher |            | Message Processor |           | Security |
 |____________|            |___________________|           |__________|
    ^     \                     ^       \                       ^
    |      \                   /         \                     /
    |       \                 /           \                   /
    |        -----------------             ----------------------
    |     2. prepareDataElements()       3. processIncomingMessage()
    |
    |
    |
    | 1. payload data with
    |    underlying headers
    |    stripped
    |

  ---------
 | Network |
 |_________|
```

Figure 2: Agent - Request Receipt Process (RFC 2271-2275)

```
                |
                | 10.data from MIB
                |
                v
   -----------------------------            ----------------
  |  Command Responder applic'n |           | Access Control |
  |_____________________________|           |________________|
     |
     |
     |
     | 11. returnResponsePDU()
     |
     |
     |
     |
     |
     |
     |     12. prepareOutgoingMessage()  13. generateResponseMessage()
     |       -----------------             ---------------------
     |      /                 \           /                     \
     |     /                   \         /                       \
     v    /                     v       /                         v
 ------------                 -------------------               ----------
| Dispatcher |               | Message Processor |             | Security |
|____________|               |___________________|             |__________|
    |     ^                    /       ^                          /
    |      \                  /         \                        /
    |       \                /           \                      /
    |        -----------------             ----------------------
    |     15.success or errorIndication  14.success or errorIndication
    |
    |
    |
    | 16. payload data to
    |    be encapsulated and
    |    sent onto network
    v

 ---------
| Network |
|_________|
```

Figure 3: Agent - Request Generation Process (RFC 2271-2275)

Table 1: RFC2271 - Agent Modules and Access to Variables

Variable	dispatcher	message processor	security	command responder application	access control
messageProcessingModel	yes	yes	yes	yes	no
securityModel	yes	yes	yes	yes	yes
securityName	yes	yes	yes	yes	yes
securityLevel	yes	yes	yes	yes	yes
contextEngineID	yes	yes	yes	yes	no
contextName	yes	yes	yes	yes	yes
pduVersion	yes	yes	yes	yes	yes
PDU	yes	yes	no	yes	yes (variable name)
securityStateReference	yes	yes	yes	yes	yes

Dispatcher: As the central hub in the SNMPv3 engine, the dispatcher has control over the data passed from the message processor to the application. The dispatcher can generate phony requests and send them up to the application. In cases where the securityLevel is "authNoPriv" it may decide not to route incoming messages toward security. Most alarmingly, the dispatcher handles incoming messages after the security module has decrypted and authenticated them, and outgoing messages before security has encrypted and signed them, so it can alter the bindings. Finally, there is nothing to ensure that the dispatcher does not send data directly to the network.

Message processor: As with the dispatcher, the message processor could change an incoming message in any way after it has been verified by the security module. This allows the message processor to defeat access control in the same manner described for the dispatcher above. RFC2271 trusts the message processor to have the security module process (and possibly encrypt) the data before returning it. There is nothing to ensure that the message processor actually did this, so it can defeat confidentiality.

Security Module: As the module responsible for verifying the incoming request's origin and authenticity, this unit could falsely verify a corrupt message, or change any of the security related information after decryption/verification. The security module can also fake encryption; nothing else within the architecture verifies that this was performed properly.

Application: Since the application has direct access to the MIB, it can act with impunity. The application can Set or Get any variable, and can return or set an incorrect variable. There is nothing that would force the application to call access control, or to honor the answer that access control returns.

Access Control: This module is responsible for determining if a particular MIB request is within an entity's view. A coding error could allow access regardless of credentials, or worse. Although the column in Table 1 is all "no", it can't be denied that access control depends on the proper coding of this module.

4 Goals for improving the security of SNMPv3

The previous section laid out detailed concerns about what problems can arise if all the modules are trusted with all the data. It is true that some of the modules must be trusted; we believe that security and access control must be. Not all of the modules need to be trusted, however. It is the thesis of this draft that trust in the other modules should (and can) be reduced.

It is worthwhile to state the goals for the changes we suggest. Ideally,

1. Sets are only performed by authorized managers
2. Signed responses from agents contain real MIB data
3. Private data are encrypted
4. Gets are only performed by authorized managers
5. Getbulk and Getnext are only performed by authorized managers

These are only specific instances of goals stated in the RFCs. Section 1.4 of RFC2271 says that the "principal threats against which the architecture should defend" include masquerade, modification of information, and message stream modification.

Among the threats, the most serious is the danger of an outsider doing a "secure" Set. (If you accept noAuthNoPriv for Sets, no one can help you.) Therefore, we concentrate here mostly on the command responder application in the agent. It is likely that similar schemes can be created to reduce trust in notification originators, or proxy forwarders.

Of course the command originator application on the manager is a big concern, but little can be done there, since the application is at the "top of the stack;" there is no MIB or access control on the manager and it is hard to imagine how to avoid having the manager's applications be trusted. It should be self-evident to SNMP designers that a corrupted manager defeats all the security in SNMPv3, so one can naturally expect them to pay close attention to security on that device.

5 A Slightly Changed Architecture for SNMPv3 Agents

Figure 4 shows an architecture for SNMPv3 that provides additional security with minimal changes. This model tries to keep the architecture similar to the original proposal in RFC 2271, while making changes to increase the chances of the system running the security functions properly. There are two fundamental changes.

First, the access control module is the only module that may access the MIB on behalf of an SNMP manager's request. Other components in the system, including processes outside SNMP, can access MIB variables they need in the course of doing their jobs. They can keep the MIB variables they use in their own address spaces, as opposed to in the kernel, where most MIB variables will reside.

In RFC2271, applications are supposed to ensure that requests for MIB access (to read or write) are authorized by calling the access control module. In RFC2271 it is possible for the application not to call access control, or to ignore the response. By placing access control "between" the applications and the MIB, we

```
 --------------------------------------------------------------------
|                                                                    |
|                                MIB                                 |
|____________________________________________________________________|
                                  ^
                                  |
                                  v
                ---------------------------------------
               |                                       |
               |            ACCESS CONTROL             |<---------
               |_______________________________________|         |
                 ^             ^                   ^             |
                 |             |                   |             |
                 v             v                   v             |
   ----------------    ----------------    ----------------      |
  |                |  |                |  |                |     |
  |    COMMAND     |  |  NOTIFICATION  |  |     PROXY      |     |
  |   RESPONDER    |  |   ORIGINATOR   |  |   FORWARDER    |     |
  |________________|  |________________|  |________________|     |
          ^                    ^                    ^            |
          |                    |                    |            |
          |   _________________|                    |            |
          |  |   ___________________________________|            |
          |  |  |                                                |
          v  v  v                                                |
   ----------------    ----------------    ----------------      |
  |                |  |                |  |                |     |
  |   DISPATCHER   |<->|    MESSAGE     |<->|    SECURITY    |<---
  |                |  |   PROCESSOR    |  |                |
  |________________|  |________________|  |________________|
           ^
           |
           v
 --------------------------------------------------------------------
|                                                                    |
|                              NETWORK                               |
|____________________________________________________________________|
```

Figure 4: Minimal Change SNMPv3 Agent Architecture

force access control to be consulted. The access control module, not the application, is trusted to control access to the MIB.

Second, we do not trust the dispatcher or the message processor to properly handle SNMPv3 security information. This is accomplished by having the SNMPv3 request signed by the manager, and having the security module verify this unaltered message. The job of the message processor for a SNMPv3 message is simplified, since the security module will know how to handle this format. Once a message is verified, security stores data about the pending request, and then the message is passed to an application. When the application makes a call to the access control module for MIB access, the access control module calls security to double-check that the request is authentic. This prevents a rogue application from accessing the MIB in a manner not allowed. The checks between access control and security can be greatly simplified if they share an address space (or part of their address space.)

The advantages of the 'slightly changed architecture' are that it reduces the trust in the dispatcher and message processor, and channels all SNMP-related accesses to the MIB through the access control module.

Problems remain with the 'slightly changed architecture'. For one thing, the application is still trusted to do things (other than access the MIB). Because the SNMP Working Group decided on a modular approach, it is possible that different applications will be produced by different developers yet all registering with the same engine. One may not care to run these applications as trusted processes. Here are two problems which could arise:

(1) An application could request and receive from access control a binding (a <variable, value> pair) which ought to go out encrypted over the network. When passing it down, it could change the security level to noAuthNoPriv or AuthNoPriv, and unfortunately the binding would be visible on the network.

(2) A Get request arrives, is decrypted by security, and when passed to the command responder application, is changed to a Set request by the application and then passed to access control.

In the next section we describe a method by which trust in the application is reduced, and problems such as these are avoided.

6 A Serialized Architecture for SNMPv3 Agents

Figure 5 shows a more secure architecture for SNMPv3. Note the serialized design of the modules. The serialization makes it clear how data will flow in this architecture. Security and access control cannot be bypassed here, as they could be in the RFC2271 architecture. The message processor and dispatcher will not touch any data after security has authenticated and decrypted it on incoming messages, and will not touch data on outgoing messages until wholeMsg has been signed by security.

The reader will notice the communication (four arrows) between security and access control. The purpose of these extra function calls is to reduce trust in the application, which lies between the two trusted modules. If we can succeed in trusting only access control and security, that would be ideal.

Our design reduces, but does not eliminate, the trust we put in the application. To be in accord with the modular design of the SNMPv3, we must allow the different application modules (command responder, notification originator, proxy forwarder) to be masters of their own tasks. Therefore, we chose a mechanism which still allows the application to do its own job, but forces the application to tell the trusted modules

```
 ---------------------------------------------------------------------
|                                                                     |
|                                MIB                                  |
|_____________________________________________________________________|
                               ^   |
                               |   v
                       ---------------------
  ---------------------|                   |----------------------
 |                     |   ACCESS CONTROL  |                      |
 |          --------->|___________________|<---------             |
 |         | requestAccessToMIB^  |requestAccessToMIB  |          |
 |         |       (IN)        |  |       (OUT)       |          |
 |  request |                  |  v                   |request    |
 |  Security|           ---------------------        |Access     |
 |  Verific-|          |                   |         |Control    |
 |  ation   |          |    APPLICATION    |         |Verific-   |
 |  (OUT)   |          |___________________|         |tion (IN)  |
 |          |      processPDU ^  | returnResponse    |           |
 |          |                 |  v                   |           |
 |          |           --------------------         |           |
 |          ----------|                    |-----------          |
 |                     |      SECURITY      |                     |
  -------------------->|____________________|<---------------------
 request      processIncomingMsg ^  | processIncomingMsg    request
 Security          (IN)          |  v      (OUT)        Access Control
Verification               ---------------------         Verification
   (IN)                   |                     |            (OUT)
                          |  MESSAGE PROCESSOR  |
                          |_____________________|
           prepareDataElements ^  | prepareDataElements
                   (IN)        |  v         (OUT)
                         ----------------------
                        |                      |
                        |      DISPATCHER      |
                        |______________________|
                  SNMP message ^  | SNMP message
                      (IN)     |  v     (OUT)
 ---------------------------------------------------------------------
|                                                                     |
|                              NETWORK                                |
|_____________________________________________________________________|
```

Figure 5: Serialized SNMPv3 Agent Architecture

enough information that they can perform double-checking.

As a result of some extra calls among security, access control and the application, we receive the following benefits:

(1) When access control goes to the MIB, it is on behalf of a principal which has those variables in its view. In other words, Sets and Gets are only performed by authorized managers.

(2) Sensitive MIB variables will be encrypted before being sent out on the network.

(3) Bindings whose signatures are checked by the manager are unchanged from when they emerged from the MIB at the agent.

None of these is ensured by the RFC2271 architecture, because all of the applications are trusted processes in that architecture. Note that there are many other properties which are desirable but harder to guarantee without major changes to the architecture. These are discussed in the next section.

Access control and security will maintain state, so that they may converse as shown in Figure 6.

```
                Incoming Get/Set Request      Outgoing Get/Set Response
                      ----------------        ----------------
                     | Access Control |     | Access Control |
                   --|________________|     |________________|<-
                  |          ^                       |          |
Is the applic'n   |          |                       |          | Is the applic.
lying? It asks    |          |                       |          | lying? It says
for something on  |          |                       |          | you gave it
behalf of         |          |                       |          | these bindings
securityName,     |          |                       V          | on behalf of
using this        |     ----------------        ----------------  | this
securityLevel     |  | Application    |      | Application    | | securityName
within this       |  |________________|      |________________| | using this
securityModel.    |          ^                       |          | securityLevel
                  |          |                       |          | within this
                  |          |                       |          | securityModel.
                  |          |                       |          |
                  |          |                       |          |
                  |          |                       V          |
                  |     ----------------        ----------------  |
                  |  |   Security     |      |   Security     | |
                   ->|________________|      |________________|-
```

Figure 6: Serialized Architecture - Secure Communications

On calls from the application to access control (incoming requests), access control will call down to security to ask if the bindings match those in the request it saw earlier. This will prevent an application from accidentally or intentionally modifying the variables requested for a Get, or even more importantly, the variable and value for a Set.

On calls from the application to security (outgoing responses), the command responder includes a pointer to the area where the outgoing bindings are. Security will call up to access control to confirm that these outgoing bindings are in fact what access control retrieved from the MIB. This ensures that a message signed by the security module contains authentic values from the MIB as passed to the application from access control.

Now we justify our claims of greater security.

(1) When access control goes to the MIB, it is on behalf of a principal which has those variables in its view. In other words, Sets and Gets are only performed by authorized managers. If a command responder application calls access control with a Get or Set request, access control verifies the bindings, securityModel, securityName and securityLevel with a separate call to security.

(2) Sensitive MIB variables will be encrypted before being sent out on the network. On an outgoing response, security verifies the securityLevel with a call to access control. This securityLevel is used by the security module to decide whether or not to encrypt the message. Since this variable was used by access control to determine the view of the MIB tree to use, only MIB variables in a view that does not use encryption (privacy) can be sent out in the clear.

(3) Bindings whose signatures are verified by the manager are unchanged from when they emerged from the MIB at the agent. Again, if a response message reaches the network, it must have passed through security. Security double-checked the bindings with access control before allowing the message to be passed down to the network.

Similar checks can probably be devised for the proxy forwarder application.

The serialized architecture clearly channels all SNMP MIB accesses through the access control module. It eliminates the trust in the dispatcher and message processor by ensuring that on incoming messages, they do not handle the messages after decryption and authentication, and on outgoing messages they do not handle the messages before encryption and signing.

In addition, the serialized architecture provides the three guarantees (above) which are not provided by the 'slightly changed architecture'. In the next section we compare the expected performance of the serialized architecture to that of RFC2271, and discuss residual security problems.

7 Analysis of the Serialized Architecture

Without trusting any user process in the agent except the security and access control modules, the serialized architecture achieves the three goals of the previous section. In this section, we argue that the serialized architecture improves security, and has little impact on performance.

In Table 2, we can see that the dispatcher and message processor cannot cause security breaches, and the application can change fewer variables. It is important to realize that this architecture is not completely secure. Because it can't help trusting the application modules to perform their specialized tasks, the serialized architecture can still be defeated by a rogue application. Consider the following situations.

(1) The command responder application may incorrectly interpret Getbulk. That is, the application may break up a Getbulk into a number of smaller requests, which do not correspond to the Getbulk, and then

Table 2: Serialized Architecture - Agent Modules and Access to Variables

Variable	dispatcher	message processor	security	command responder application	access control
messageProcessingModel	no	no	yes	yes	no
securityModel	no	no	yes	no	yes
securityName	no	no	yes	no	yes
securityLevel	no	no	yes	no	yes
contextEngineID	no	no	yes	yes	no
contextName	no	no	yes	yes	yes
pduVersion	no	no	no	yes	no
PDU	no	no	yes	no	yes (variable name)
securityStateReference	no	no	yes	no	no

later reassemble the response message. The serialized architecture does not prevent this, because access control and security do not know how the command responder breaks up a Getbulk. (If they knew and double-checked everything, there would be no role for the application; it would essentially be absorbed into the trusted modules.)

(2) If a message type other than the one that the security module knows how to decode is used, the message processor could change some of the security related parameters, such as the securityLevel, making possible the retrieval of something outside the manager's view. However, the serialized architecture will still prevent these bindings from being sent in the clear, if they required encryption. Also, data will not be sent unless an authorized user made the request.

(3) The application could change a Get to a Set, since the access control module does not check the message type. Since the Get requests will only have a place-holder in the binding values on the request, this would allow an application to set any MIB variable to the value of that place-holder - probably zeros. This problem can be solved by not giving any manager-principal permission to request both Sets and Gets. In other words, create a separate principal for Gets and Sets. In this case, if a Get request is authenticated, the application can not change it to a Set because that principal will not have Set access to any MIB variables.

These scenarios show that not all trust in the application has been removed. Nevertheless, because it trusts fewer modules than the RFC2271 architecture, security is improved.

In Table 3 we compare the processing time for a typical Get or Set Request. Note that in the data flow diagrams presented in section 2, there are 16 calls/returns. There would be four fewer calls if security were not called, i.e. if the securityLevel were noAuthNoPriv. A count of the arrows in the serialized architecture diagram of section 6 shows that at worst there are 16 calls/returns. In the noAuthNoPriv case, the calls between access control and security would be unnecessary, reducing the number by four.

The additional work required by the serialized architecture is not great; some functions have moved from one module to another. While the access control module has more to do in the serialized architecture, the application has correspondingly less to do. It is true that both security and access control are stateful, but in RFC2271 the message processor, application and security modules are stateful, so statelessness was evidently not an important goal in agent design.

Table 3: Number of Calls/Returns Generated by a Get or Set Request

securityLevel	RFC2271 architecture	serialized architecture
noAuthNoPriv	12	12
authNoPriv	16	16
authPriv	16	16

We expect the "message passing" burden not to differ much either. Passing messages between processes in different address spaces is costlier than within one space. In the implementation of a serialized agent, the security and access control modules should not be in the same address space as the untrusted modules. In RFC2271, the agent is divided into an "engine" and "applications which register with the engines", which suggests that they are separate processes. Therefore, we expect the costs for message passing between processes to be about the same in the serialized architecture as in RFC2271.

Since the number of calls is unchanged, and the total work done is much the same, we predict the serialized architecture would have performance comparable to the RFC2271 agent. If the implementor codes wisely, utilizing shared address spaces as noted in this document, the new architecture may actually be faster than the old one; however, since more state information is being maintained, the memory requirements will likely be higher.

8 Summary

In this document we demonstrated how security problems could result from the miscoding of modules in the SNMPv3 architecture described by RFC2271.

Many of these problems can be avoided by reorganizing the architecture, which affects the various module functions very little. We provided two such reorganizations. The first consists of two minor changes; the second places all the modules in a line, so that it is obvious how data must flow.

Our new architectures achieve better security, by (a) requiring that only two modules be coded with great care, instead of all five, and (b) guaranteeing that if this is done, certain undesirable situations do not arise.

Therefore, we recommend that these improvements be incorporated into SNMPv3.

A Data Flow for the Serialized Architecture

SNMP message (IN):

- Same as RFC2271 architecture

prepareDataElements (IN):

- The message processor sends the same information to the security module as in the RFC2271 architecture (messageProcessingModel, expectResponse, maxMessageSize, securityParameters, securityModel, securityLevel, wholeMsg, and wholeMsgLength.)

processIncomingMsg (IN):

- Similar to the RFC2271 architecture. For SNMPv3 messages, the security module should know where the fields passed from the message processor are located in the wholeMsg. The security module uses the fields from the authenticated wholeMsg instead of the ones passed from the message processor. For other versions, the security module can use the ones passed from the message processor, but this induces a level of risk that should be taken into account when setting the access permissions for principles using versions of SNMP other than the one the security module knows how to decode (non-SNMPv3).

processPdu:

- The application is passed the same information by the security module that the dispatcher passed in the RFC2271 architecture. Namely: messageProcessingModel, securityModel, securityName, securityLevel, contextName, pduVersion, PDU, maxSizeResponseScopedPDU, and stateReference.

requestAccessToMIB (IN):

- Here the application basically combines the functions of accessing the MIB and performing the isAccessAllowed function call. This function call will provide access control with the same information that was passed in both of those two calls, so the access control module can determine if access should be granted and actually perform the access to the MIB.
- Additional information must be passed to the access control module to allow the next two function calls to be performed. Namely the location of the start and stop of the variable bindings inside the original scopedPdu.

requestSecurityVerification (IN) (not for notification originators):

- Here the access control module calls the security module to verify that the string inside the scopedPdu that security verified matches the variable bindings that access control is using to access the MIB. This will prevent the application from modifying the variables requested, or changing the variables or values when performing a set.

- This check can be performed by passing a hash of the bindings to security, or passing the bindings themselves. This function call can be eliminated by using a memory area shared by the access control module and security to provide a secure means of passing this information.

requestSecurityVerification (OUT) (not for notification originators):

- This either returns true or false.

accessMIB (IN):

- This is simply the computer dependent access of the MIB that the application performed in the RFC2271 architecture.

accessMIB (OUT):

- This is the variable(s) for a Get/Getnext/Getbulk request, or the response code for a Set.

requestAccessToMIB (OUT):

- This returns the variable bindings to the application to be placed in a response message.

returnResponse:

- These are the same data that were passed from the application to the dispatcher in the RFC2271 architecture, along with some additional information for the security module to perform the next two function calls. This additional information is the location inside the scopedPdu that contains the list of variable bindings that was provided to the application by access control.

requestAccessControlVerification (IN):

- This step is the security module calling the access control module to ensure that the variable binding list is intact before applying a signature. This can be accomplished by providing a hash to this string, passing the string to access control, or simply having access control pass the original string back to security. If desired, the access control module could omit this string from the requestAccessToMIB response and only pass a place holder (possibly an array of zeros). Then the security module could get this information from the access control at this step. This would prevent the application from divulging this information either accidentally or on purpose. Again, this function call can be eliminated by using a memory area shared by the access control module and security to provide a secure means of passing this information.

responseAccessControlVerification (OUT):

- This simply returns a pass/fail, or the data to fill into the scopedPdu if the method above was applied.

processIncomingMsg (OUT):

- This is the same as this function call between the message processing module and security in the RFC2271 architecture.

prepareDataElements (OUT):

- This is the same as this function call between the message processor and the dispatcher in the RFC2271 architecture.

SNMP message (OUT):

- The SNMP message delivered to the network stack.

Security Considerations

This document describes potential security problems which could result from implementation of the SNMPv3 architecture as described in RFCs 2271-2275.

In addition, this document proposes two new architectures for consideration. These new architectures address the aforementioned security problems by limiting the access to data of certain modules within the existing SNMPv3 architecture.

References

[RFC-2271] D. Harrington, R. Presuhn, B. Wijnen, *An Architecture for Describing SNMP Management Frameworks*, RFC-2271, January 1998.

[RFC-2272] J. Case, D. Harrington, R. Presuhn, B. Wijnen, *Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)*, RFC-2272, January 1998.

[RFC-2273] D. Levi, P. Meyer, B. Stewart, *SNMPv3 Applications*, RFC-2273, January 1998.

[RFC-2274] U. Blumenthal, B. Wijnen., *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*, RFC-2274, January 1998.

[RFC-2275] B. Wijnen, R. Presuhn, K., *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)*, RFC-2275, January 1998.

Addresses of Authors

The authors can be contacted at:

Steve Borbash
Department of Defense
E-mail: sab@afterlife.ncsc.mil

Bill Freeman
Department of Defense
E-mail: wef@afterlife.ncsc.mil

Douglas Maughan
Phone: 301-688-0847
E-mail: wdm@tycho.ncsc.mil

National Security Agency
ATTN: R23
9800 Savage Road
Ft. Meade, MD. 20755-6000

Copyright Notice