CHAPTER 4

SNMPv3 Framework

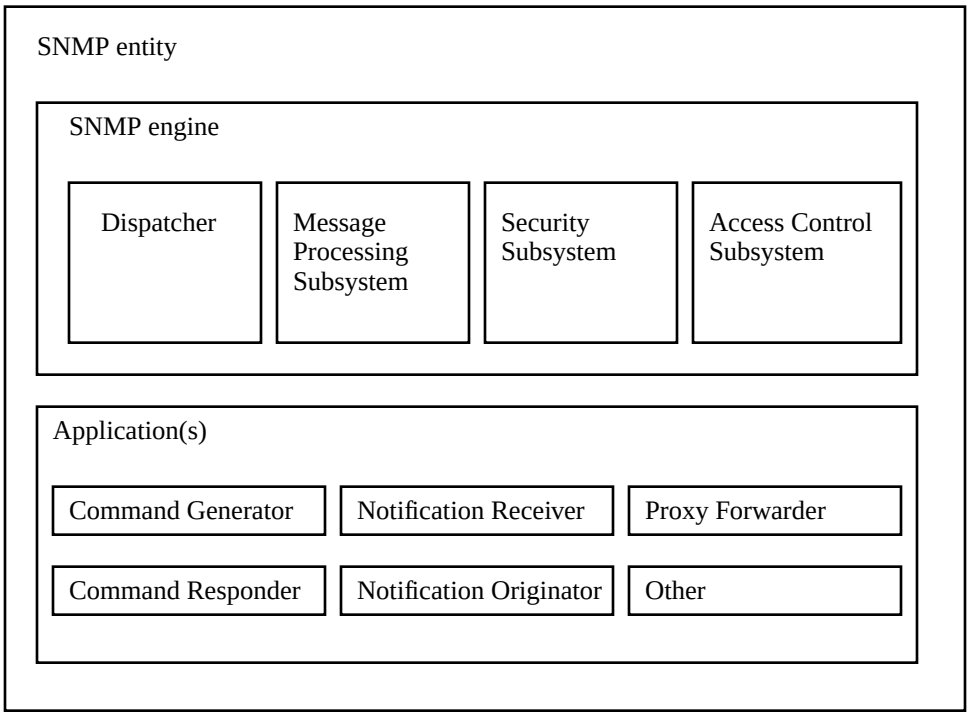
This chapter will describe the architecture that the SNMPv3 Framework is defined under. It will also show new textual conventions that have been defined for SNMPv3, along with a new SNMPv3 message format. The goal for this chapter is to prepare the reader for the following SNMPv3 chapters covering notification and proxy forwarding configuration, security, and view-based access control.

4.1 Architecture Overview

An architecture was originally defined by RFC 2271 for describing SNMP Management Frameworks, and the SNMPv3 Framework is a concrete "instantiation" of this architecture. Conceptually SNMPv3 is nothing more than an extension of SNMP to address two major areas, administration and security. A major goal for SNMPv3, though, is to support a module architecture that can be easily extended. This way, for example, if new security protocols are advanced they can be supported by SNMPv3 by defining them as separate modules. Hopefully this will allow us to avoid having to buy books on SNMPv4 in the future.

How important is it to understand this architecture in order to use SNMPv3? Probably not terribly. However, the SNMPv3 Framework is fairly simple and introduces some terminology that will be used later on.

What we used to call SNMP Agents and SNMP Managers, we now call an SNMP entity. An SNMP entity is made up of two pieces: an SNMP engine and SNMP applications.



4.1.1 SNMP Engine

As you can see from the above diagram, an SNMP engine is made up of the following components:

- Dispatcher
- Message Processing Subsystem
- Security Subsystem
- Access Control Subsystem

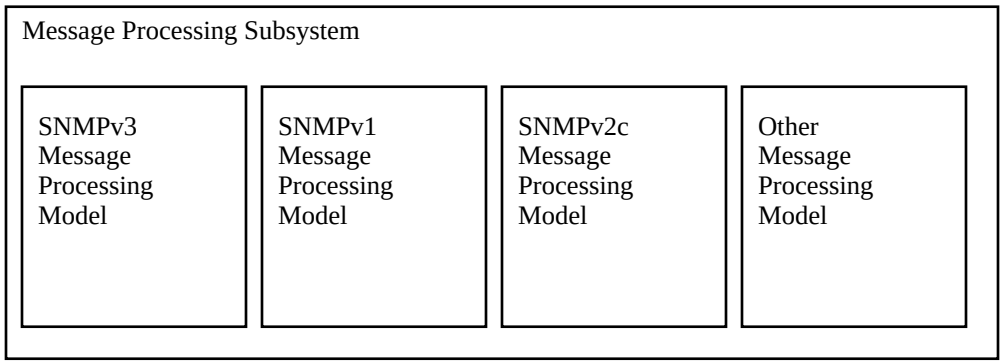
4.1.1.1 Dispatcher

The Dispatcher is responsible for sending and receiving messages. When a message is received, the Dispatcher tries to determine the version number of the message and then passes the message to the appropriate Message Processing Model. If the message cannot be parsed so

that the version can be determined, then the snmpInASNParseErrs counter is incremented and the message is discarded. If the version is not supported by the Message Processing Subsystem, then the snmpInBadVersions counter is incremented and the message is discarded. The dispatcher is also responsible for dispatching PDUs to applications, and for selecting the appropriate transports for sending messages.

4.1.1.2 Message Processing Subsystem

The Message Processing Subsystem is made up of one or more Message Processing Models. The following diagram shows a Message Processing Subsystem that supports models for SNMPv3, SNMPv1, SNMPv2c, and something that we will call “Other.”



The Message Processing Subsystem is responsible for

1. Preparing messages to be sent.
2. Extracting data from received messages.

Let’s walk through a simple case where the Dispatcher receives a valid SNMPv3 message from the line. The Dispatcher determines the version of the message and forwards it to the SNMPv3 Message Processing Model. The SNMPv3 Message Processing Model then processes the message by extracting information from it. It then calls the Security Subsystem to decrypt the data portion of the message (if needed) and make sure the message is properly authenticated. At that point the Dispatcher will forward the PDU portion of the message to the appropriate SNMP application (more about that later).

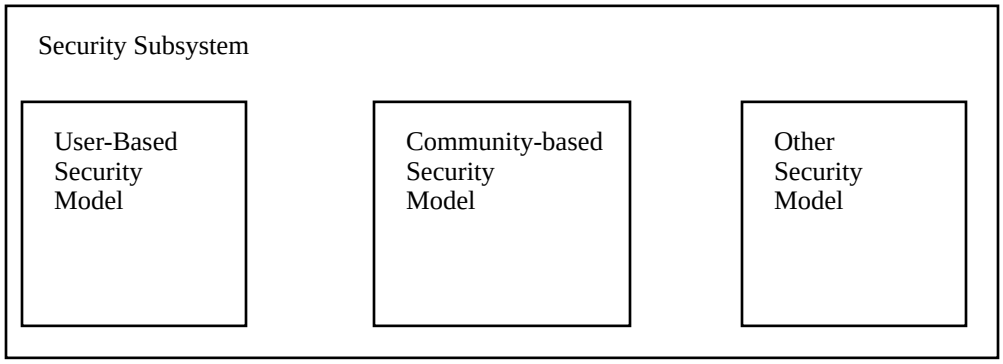
This architecture allows additional models (like “Other”) to be added. These additional models may be enterprise specific or future standards. In any case, the Dispatcher will need to be able to parse the messages to determine the version (and then map the version number to a Message Processing Model).

4.1.1.3 Security Subsystem

The Security Subsystem provides security services such as

1. Authenticating messages.
2. Encrypting/decrypting messages for privacy.

The following diagram shows a Security Subsystem that supports models for SNMPv3, a Community-based Model, and something we will call “Other.” The Community-based Model would support SNMPv1 and SNMPv2c.



A Security Model defines among other things

1. The security threats against which it protects.
2. The services it provides.
3. The security protocols used to provide services such as authentication and privacy.

The User-Based Security Model will be described in detail in the SNMPv3 security chapter. It protects SNMPv3 messages from the following potential security threats:

- An authorized user sends a message that gets modified in transit by an unauthorized SNMP entity. For example, an authorized user may send a message to set the operational state of a port to `up`. Someone might maliciously try to capture the message, modify it so the message sets the operational state of the port to `testing`, and then put the message back on the wire.

- An unauthorized user trying to masquerade as an authorized user. For example, someone might try to perform management operations (such as change the operational state of a port) that they don't have authorization for by pretending to be an authorized user.
- Modifying the message stream. SNMP is typically based on UDP, which is a connectionless transport service. Messages could potentially be captured and reordered, delayed, or possibly replayed at a later time. For example, if a Set operation were captured and replayed in the future, it could conceivably change the desired configuration. By checking the timeliness of messages, this threat can be minimized.
- Eavesdropping. By allowing messages to be encrypted, someone eavesdropping on the line won't be able to make sense of what they see. This feature is essential for carriers that need to protect against sensitive data, such as billing information, from being eavesdropped on.

The User-Based Security model currently defines the use of HMAC-MD5-96 and HMAC-SHA-96 as the possible authentication protocols and CBC-DES as the privacy protocol. Future authentication and privacy protocols may be added.

SNMPv1 and SNMPv2c Security Models provide only weak authentication (community names) and no privacy.

This architecture allows additional Security Models (like "Other") to be added. These additional models may be enterprise specific or future standards. Authentication and privacy protocols supported by Security Models are uniquely identified using Object Identifiers. Any IETF standard protocols for authentication should have an identifier defined within the snmpAuthProtocols subtree. Any IETF standard protocols for privacy should have an identifier defined within the snmpPrivProtocols subtree. Enterprise specific protocols should have their identifiers defined within the enterprise subtree.

The snmpAuthProtocols subtree is defined as

{iso (1) org (3) dod (6) internet (1) snmpV2 (6) snmpModules (3) snmpFrameworkMIB (10) snmpFrameworkAdmin (1) snmpAuthProtocols (1)} or, 1.3.6.1.6.3.10.1.1

usmHMACMD5AuthProtocol identifies the HMAC-MD5-96 authentication protocol and has the value {snmpAuthProtocols 2}.

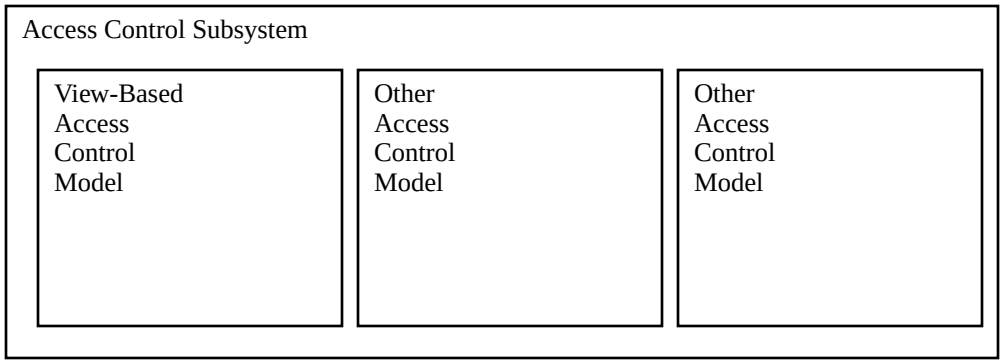
usmHMACSHAAuthProtocol identifies the HMAC-SHA-96 authentication protocol and has the value {snmpAuthProtocols 3}.

The snmpPrivProtocols subtree is defined as 1.3.6.1.6.3.10.1.2.

usmDESPrivProtocol identifies the CBC-DES symmetric encryption protocol and has the value {snmpPrivProtocols 2}.

4.1.1.4 Access Control Subsystem

The responsibility of the Access Control Subsystem is straightforward: determine whether access to a managed object should be allowed. Currently one access control model, View-Based Access Control Model (VACM), has been defined. VACM will be described in detail the SNMPv3 View-Based Access Control chapter. As the following diagram shows, the SNMPv3 Framework allows additional Access Control Models to be defined in the future.



As we will be seeing later, with VACM you can control which users and which operations can have access to which managed objects.

So within the SNMPv3 Framework, who calls the Access Control Subsystem? Any SNMP application which needs to access managed objects. Currently that would be any Command Responder or Notification Originator application. What that means in simpler terms:

1. When an SNMP Get, Get-Next, Get-Bulk, or Set PDU is being processed, the Access Control Subsystem needs to be called to make sure the MIB objects specified within the variable bindings are allowed to be accessed.
2. When a Notification (either an SNMPv2-trap or Inform) is being generated, the Access Control Subsystem needs to be called to make sure the MIB objects specified for the variable bindings are allowed to be accessed.

4.1.2 Applications

For SNMPv3, when we refer to applications, we are referring to internal applications within an SNMP entity as opposed to what you might normally think of, such as a network management application to do trending or configuration. These internal applications do things like generate SNMP messages, respond to received SNMP messages, generate notifications, receive notifications, and forward messages between SNMP entitites. Currently there are five types of applications defined:

1. Command Generators — generate SNMP commands to collect or set management data.
2. Command Responders — provide access to management data. For example, processing Get, Get-Next, Get-Bulk and Set PDUs are done by a Command Responder application.
3. Notification Originators — initiate Trap or Inform messages.
4. Notification Receivers — receive and process Trap or Inform messages.
5. Proxy Forwarders — forward messages between SNMP entities.

The SNMPv3 Framework allows other applications to be defined over time. From this list, you can see that Command Generators and Notification Receivers are what we used to think of as part of an SNMP Manager, while Command Responders and Notification Originators are what we used to think of as part of an SNMP Agent.

4.2 New Textual Conventions

4.2.1 SnmpEngineID

This type is used to represent an SNMP engine's administratively unique identifier. Since each SNMP entity contains a single SNMP engine, this will also uniquely identify an SNMP entity within an administrative domain.

An SnmpEngineID resolves to an OCTET STRING between 5 and 32 bytes long. If the first bit of an SnmpEngineID value is 0, then the value was supplied by the vendor and has the following format:

1. The first four octets are set to the device's SNMP management private enterprise number as assigned by the Internet Assigned Numbers Authority (IANA). For example, Cisco's private enterprise number is 9. For a Cisco device the first four octets would be assigned "00000009."
2. The following 8 bytes are assigned in an enterprise-specific method. For example, a ven-

dor may use the IP address of the entity padded with 4 random bytes. Or it may use a MAC address padded with 2 random bytes. Or could use one of several different proprietary methods.

If the first bit of an SnmpEngineID is 1, then the value has the following format:

1. The first four octets are set to the device's SNMP management private enterprise number as assigned by the Internet Assigned Numbers Authority (IANA),with the first bit being set to 1. For example, Cisco's private enterprise number is 9. For a Cisco device the first four octets would be assigned "80000009."
2. The fifth octet is used to indicate how the rest of the octets are used. Its values are

   - 0, reserved.
   - 1, the following four octets are an IPv4 address. The IP address used is the lowest, non-special address assigned to the device.
   - 2, the following sixteen octets are an IPv6 address. The IP address used is the lowest, non-special address assigned to the device.
   - 3, the following six octets are a MAC address. The MAC address used is the lowest MAC address assigned to the device. It is represented in canonical order.
   - 4, the remaining octets are administratively assigned text. The length of the text is enterprise specific and can be at most 27 octets.
   - 5, the remaining octets are administratively assigned hex values. The length of the octet string is enterprise specific and can be at most 27 octets.
   - 6 - 127, reserved.
   - 128 - 255, the meaning of these values is enterprise specific. For example, Cisco could define 128 to refer to a Cisco specific algorithm. BayNetworks on the other hand can define a different meaning for this. The maximum remaining length of the SnmpEngineID can be 27 octets.

As you can see from this the length of a SnmpEngineID can be anywhere between five and thirty-two octets.

4.2.2 SnmpSecurityModel

This type is used to identify a security model being used. It resolves to an INTEGER and has the following values defined:

- 0, reserved for "any."
- 1, SNMPv1.
- 2, SNMPv2c.

- 3, User-Based Security Model (USM).
- 4 - 255, reserved for standards-track security models. These values will be managed by the Internet Assigned Numbers Authority (IANA).
- Values greater than 255 can be used to specify enterprise-specific models. An enterprise-specific security model can be defined as

enterprisedNumber * 256 + securityModel

For example, given Cisco's enterprise number, 9, Cisco could define enterprise-specific security models with identifiers in the range of 2304 through 2559. This scheme allows enterprises to define up to 255 enterprise-specific security models. This scheme also will support up to 8,388,606 enterprises (given a 32-bit value: first bit needs to be 0, the next 23 bits can be used to identify an enterprise, the final 8 bits are used to define a security model).

In reality, are new security models going to be defined? Very rarely, if ever. Are enterprises going to define their own security models? Possibly, but probably not vendors that have to be concerned with interoperability. Government agencies, though, might take advantage of this.

4.2.3 SnmpMessageProcessingModel

This type is used to identify the message processing model used to process an SNMP message. It resolves to an INTEGER and can have one of the following values:

- 0, SNMPv1.
- 1, SNMPv2c.
- 2, SNMPv2u and SNMPv2*.
- 3, SNMPv3.
- 4 - 255, reserved for standards-track message processing models. These values will be managed by the Internet Assigned Numbers Authority (IANA).
- Values greater than 255 are handled exactly the same way as with the SnmpSecurityModel type to allow enterprise-specific message processing models. An enterprise- specific message processing model can be defined as

enterpriseNumber * 256 + messageProcessingModel

Again, as with the security model example, since Cisco's enterprise number is 9, Cisco could define enterprise-specific message processing models with identifiers in the range of 2304 through 2559. And as with the security model, this scheme allows enterprises to define up to 255 enterprise-specific message processing models.

4.2.4 SnmpSecurityLevel

This type defines the three security levels that can be used. They are

- `noAuthNoPriv` (1), SNMP messages are sent without authentication and without privacy.
- `authNoPriv` (2), SNMP messages are sent with authentication but without privacy.
- `authPriv` (3), SNMP messages are sent with authentication and with privacy.

4.2.5 SnmpAdminString

This resolves to an OCTET STRING and can be up to 255 bytes long. This is used to represent administrative information, preferably in human-readable form. An SnmpAdminString is encoded in UTF-8 format, which for 7-bit US-ASCII will be identical.

SnmpAdminString is used throughout the SNMPv3 MIBs that we will be looking at in later chapters. It is basically used to represent textual information, such as a user name or an identifier string.

4.2.6 SnmpTagValue

SnmpTagValue resolves to an OCTET STRING. It is expected that its values be human-readable text, such as things like “router” or “host,” but can really be any OCTET STRING as long as it doesn’t contain a delimeter character (space, tab, carriage return or linefeed).

We’ll see in the SNMPv3 applications chapter an example of using SnmpTagValue objects to identify which entities to send notifications and forward messages to. While the general use of an SnmpTagValue object is to select table entries, its use is application and MIB specific.

4.2.7 SnmpTagList

SnmpTagList resolves to an OCTET STRING. It is used to represent a list of tag values. Delimeter characters (space, tab, carriage return, or linefeed) are used to separate tag values in a list. *Note*: Only a single delimeter character should be used between two tag values.

While none of the tag values within the list may be a zero-length OCTET STRING, it is perfectly valid for the SnmpTagList object to be an empty string. This would simply represent a tag list with no tag values. Example values for an SnmpTagList object could be “router Cisco,” “router,” etc.

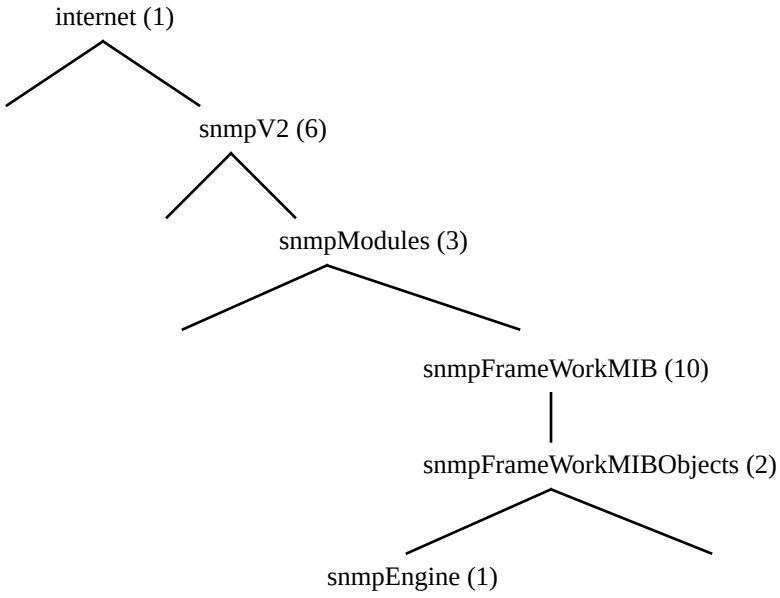
We’ll see in the SNMPv3 applications chapter an example of using SnmpTagList objects in conjunction with SnmpTagValue objects to identify the target systems to send notifications and forward messages to. Again, as with SnmpTagValue objects, its use is application and MIB specific.

4.2.8 KeyChange

KeyChange resolves to an OCTET STRING. A KeyChange object is used to change the private keys used for authentication and privacy. We'll be looking at how this is done in detail in the SNMPv3 Security chapter.

4.3 The snmpEngine Group

Several MIB variables have been defined to provide information about an SNMP engine. The snmpEngine group is defined under the MIB tree:



which defines snmpEngine to have the OBJECT IDENTFIER 1.3.6.1.6.3.10.2.1.

Table 4-1 snmpEngine Group

Object	Type	Access
snmpEngineID	SnmpEngineID	read-only
snmpEngineBoots	INTEGER	read-only
snmpEngineTime	INTEGER	read-only
snmpEngineMaxMessageSize	INTEGER	read-only

snmpEngineID: uniquely identifies an SNMP engine within an administrative group. Since there is a one to one mapping between an SNMP engine and an SNMP entity, this is also used to uniquely identify an SNMP entity.

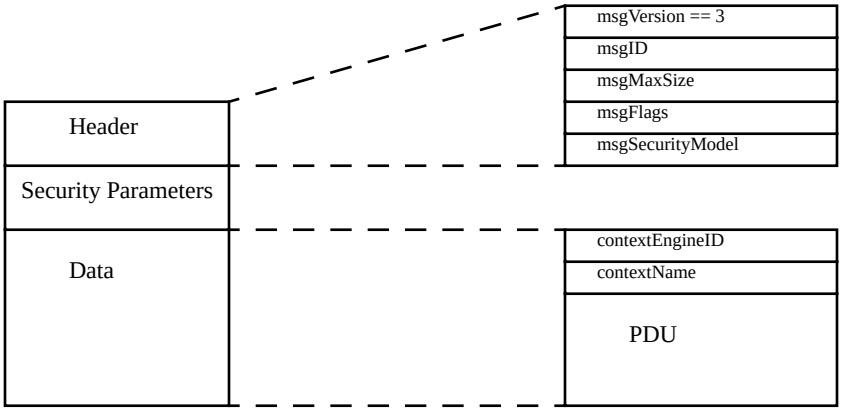
snmpEngineBoots: number of times an SNMP engine has either been started or re-initialized since snmpEngineID was last configured.

snmpEngineTime: number of seconds since the value of the snmpEngineBoots object last changed. If incrementing this value causes it to exceed its maximum value (2147483647, which is roughly 68 years) it will wrap back to zero and snmpEngineBoots will be incremented by one. Future network management applications need to be aware of this or we might end up with a year 2067 problem!

snmpEngineMaxMessageSize: maximum size in octets of an SNMP message which this SNMP engine can send or receive. This is determined by the minimum of the MMS (maximum message size) values supported among all of the transports available to and supported by the engine. This value can range between 484 (any implementation must be able to support an SNMP message size of at least this value) and 2^{31}-1.

4.4 SNMPv3 Message Format

A new format has been defined for SNMPv3 messages. An SNMPv3 message contains among other things an SNMPv2 PDU either encrypted or in plain text, security information, and the context the message should be processed in. The format for the message is



The header is made up of the following:

- msgVersion, a value of 3 identifies the version of the message as an SNMPv3 message.

- msgID (message identifier), this is an integer value that is used to coordinate request and response messages between two SNMP entities. The use of this is similar to the use of the request identifier within a PDU. The request identifier is used by SNMP applications to identify the PDU. The msgID is used by the engine to identify the message which carries a PDU.

  Note: One of the security threats that SNMPv3 tries to protect against is where a valid message is captured and replayed later. By guaranteeing that msgID values are not reused and that each message is identified by a unique value, this threat can be eliminated. One possible implementation to generate unique msgID values is to use the low-order bits of snmpEngineBoots as the high-order portion of the msgID value and a counter value for the low-order portion of msgID. This will protect against an SNMP entity generating the same msgID value after a device reboots. It will also guarantee that msgID values won't repeat until after 65,535 messages (2^{16}-1) have been generated.

- msgMaxSize (maximum message size), an integer value which indicates the maximum message size that the sender can support. This value is used to determine how big a response to a request message can be. This can have values ranging from 484 through 2^{31}-1.

- msgFlags (message flags), a 1-byte value that contains flags that indicate whether the message can cause a Report to be generated and the security level the sender had applied to the message before it was sent on the wire. The 3 bits defined are reportableFlag, authFlag, and the privFlag.

  If the reportableFlag is set, then a Report PDU can be sent back to the original sender (more on Report PDUs later). All messages that can be responded to (such as a Get PDU or an Inform PDU) are automatically treated as if reportableFlag is set to 1. All messages that are unacknowledged (such as a Report PDU, a Response PDU, or an SNMPv2-trap PDU) are automatically treated as if reportableFlag is set to 0. The reportableFlag is only used if the PDU portion of a message cannot be decoded, for example, if a PDU cannot be decrypted because of an invalid encryption key.

  The authFlag and privFlag are used to indicate the security level. This can indicate the message was sent with no authentication and no privacy, authentication and no privacy, or authentication and privacy. The receiver of the message must apply this same security level when the contents are processed.

- msgSecurityModel (message security model), an integer value which identifies the message security model that the sender used to generate this message. The receiver, obviously, must use the same security model to perform security processing for the message. The possible values for this are defined by the SnmpSecurityModel type. Since enterprise-specific security models may be implemented, the mapping of this value to the desired security model within an SNMP engine may need to be done in an implementation-dependent way.

The security parameters provided depend on the security model being used. These values are passed directly to the security model that maps to the msgSecurityModel field in the header portion of the message.

The data portion of the message is either encrypted or in plain text. The data portion is encrypted if the privFlag with the header portion is set. Whether encrypted or in plain text, the

data portion contains both context information and a valid SNMPv2c PDU (either Get, Get-Next, Get-Bulk, Set, Response, Inform, Report, or SNMPv2-trap).

The context information includes both a context engine identifier and a context name. Given this information, the proper context for which this PDU should be processed can be determined. *Note*: If a Request PDU (Get, Get-Next, Get-Bulk, Set) contains a context engine identifier that doesn't equal the SNMP engine's administratively unique identifier (snmpEngineID), then the Proxy Forwarder application will attempt to forward the message to the appropriate target. We will see how this is done in the SNMPv3 applications chapter.

4.5 Additional SNMP Statistics

Three additional 32-bit counters are defined under a snmpMPDStats subtree. These counters provide information about the number of packets that an SNMP engine dropped because the packets referenced unknown security models, had invalid or inconsistent components, or had PDUs that could not be processed.

snmpUnknownSecurityModels: the number of received packets that an SNMP engine dropped because they referenced either an unknown or unsupported security model.

snmpInvalidMsgs: the number of received packets that an SNMP engine dropped because they contained either invalid or inconsistent components within the SNMP message.

snmpUnknownPDUHandlers: the number of received packets that an SNMP engine dropped because the PDU could not be processed. Internally, applications must register for a combination of a context engine identifier and a PDU type.

4.6 Reports

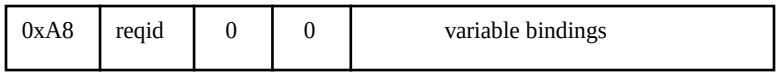
Report PDUs were defined for SNMPv2 but never used. For SNMPv3 they provide engine to engine communication and are processed directly by the SNMPv3 Message Processing Model. They allow an SNMP engine to tell another SNMP engine that an error was detected while processing an SNMP message. This (in theory anyway) allows the original SNMP engine to send a corrected SNMP message. Report PDUs are generated to report the following types of problems:

1. A Response message could not be generated.
2. A message was received with the authFlag cleared and the privFlag set.
3. An error occurred while providing authentication and privacy services for an incoming message.

Reports are also used for discovery and time synchronization purposes. An example of this will be shown in the SNMPv3 security chapter.

There have been some concerns addressed over the SNMPv3 mailing list on whether reports open a device up to denial of service attacks. A denial of service attack is where an attacker can use forged unauthenticated reports to confuse a receiving SNMP engine so that it does not talk correctly with other SNMP engines. The problem with dealing with a denial of service threat is distinguishing between intentionally invalid (malicious) messages and normal failures.

While the Report PDU was defined by RFC 1905 as part of SNMPv2, it was never used until SNMPv3. The format for a Report PDU is



where

- The PDU type 0xA8 indicates a Report PDU.
- reqid is either the request identifier of the message that triggered the report, or zero if the request identifier cannot be extracted (for example, if the PDU cannot be decrypted).
- The variable bindings will contain a single object identifier and its value. This is used to determine the problem that the report is identifying.

4.7 Summary

This chapter provided an overview of the SNMPv3 Framework, including an introduction to the subsystems that make up the framework. It also described new textual conventions that were added for SNMPv3, and the SNMPv3 message format. This chapter finsihed up by providing a description of how Reports are used by SNMPv3.