The Simple Times™

THE QUARTERLY NEWSLETTER OF SNMP TECHNOLOGY, COMMENT, AND EVENTS℠
VOLUME 4, NUMBER 1 JANUARY, 1996

The Simple Times is an openly-available publication devoted to the promotion of the Simple Network Management Protocol. In each issue, ***The Simple Times*** presents technical articles and featured columns, along with a standards summary and a list of Internet resources. In addition, some issues contain summaries of recent publications and upcoming events. For information on submissions, see page 16.

In this Issue:

The Simple Times is available via both electronic mail and hard copy. For information on subscriptions, see page 16.

The User-based Security Model

Glenn W. Waters
Bell-Northern Research Ltd.

The User-based Security Model for SNMPv2 (USEC) provides an administrative framework through which multiple levels of security for SNMPv2 protocol interactions can be defined. It achieves this with a minimum of overhead on the management and agent entities.

This article gives an overview of the concepts and security features of the USEC model. The USEC model also defines mechanisms to handle proxy agents that will be covered in a future article in ***The Simple Times***.

The *user*, a well known paradigm for computer users, is the basis of all protocol interactions in the USEC model. Conceptually, the user model is very analogous to the user-id/password that is used to login to a computer. The user, identified by a userName (the "user-id") is used to identify who is accessing information at an agent. A key (the "password") is used to ensure that the user is authentic. Optionally, a second key may be used to ensure privacy. The user-id/password model also grants a user-id a set of privileges. Similarly, the USEC model associates a set of access rights with an user.

Implicit in the user-id/password model is that some entity (machine or human) has knowledge of the user-id and password that may be used to login to a particular computer. The USEC model maintains that paradigm in that the shared knowledge of a user must be known to both the entity that wishes to access an SNMPv2 agent and to the agent itself.

Goals and Constraints

The specific goals of the USEC model with respect to security, are:

- to provide verification that each received SNMPv2 message has not been modified during its transmission such that an unauthorized management operation might result;

- to provide verification of the identity on whose behalf the SNMPv2 message claims to have been generated;

- to provide verification of timeliness of received messages; and,
- optionally, to ensure that the contents of SNMPv2 messages are protected from disclosure from unauthorized sources.

The USEC model specifically does not attempt to protect against denial of service attacks and against traffic analysis.

The goals and non-goals defined here are, in practice, the same as those defined for the now-historical SNMPv2 party model specified in RFC 1446.

Definition of a User

A user has the following attributes:

- *userName*, an octet string that represents the identity of the user;
- *authProtocol*, an indication of whether messages sent on behalf of this user can be authenticated, and if so, the type of authentication protocol which is used;
- *authPrivateKey*, if messages sent on the behalf of this user can be authenticated, the private authentication key for use with the authentication protocol;
- *privProtocol*, an indication of whether message sent on behalf of this user can be protected from disclosure, and if so, the type of privacy protocol which is used; and,
- *privPrivateKey*, if messages sent on behalf of this user can be protected from disclosure, the private privacy key for use with the privacy protocol.

Those attributes must be known to both the sender and the receiver of a communication.

The USEC model defines the use of MD5 as the authentication protocol and the use of DES as the privacy protocol. The keys associated with both the MD5 and DES protocols are 128-bit values. To provide an interface that is more user-friendly than 128-bit keys, USEC model specifies the use of a password to key algorithm as was originally defined by Steve Waldbusser as part of the SNMPv2 party model.

Other Definitions

Agents that support the USEC model must maintain three objects:

- *agentID*, which is a 12 octet identifier that is unique among all agents in an administrative domain;
- *agentBoots*, which is a count of the number of times the agent has rebooted/re-initialized since the agentID was last configured (each time the agent is re-initialized the value of agentBoots must be incremented by one); and,
- *agentTime*, which is the number of seconds since agentBoots was last incremented.

Quality of Service (qoS) is defined as the level of security that is afforded a particular message. The defined qoS levels are:

- no authentication (noAuth) and no privacy (noPriv);
- with authentication (auth) and no privacy (noPriv); and,
- with authentication (auth) and with privacy (priv).

A message's qoS is included within the message's header, and qoS is also used to indicate messages that may result in a report PDU being generated.

Time Window is a value that specifies the window of time in which an authenticated message generated on behalf of a user is valid. The same value of the Time Window, 150 seconds, is used for all users.

Local Configuration Datastore (LCD) is a locally defined (conceptual) datastore that holds a set of information about (locally known) SNMPv2 users and other associated information (e.g., access control). Each SNMPv2 entity maintains an LCD. An LCD may potentially be required to hold information about multiple SNMPv2 agent entities (e.g., in a manager that communicates with multiple SNMPv2 agent entities), and as such the agentID should be used to distinguish the information associated with a particular agent entity in the LCD.

Message Format

The format of a message using the USEC model is the same as the SNMPv1 message specified in RFC 1157, except that:

- the version number is changed to 2; and,
- the data component contains either a PDU or an OCTET STRING containing an encrypted PDU.

In addition, the SNMPv1 community string component in the message, termed the *parameters* component in the USEC model, contains a set of administrative information for the message.

An SNMPv2 message is an ASN.1 value with the following syntax:

```
Message ::=
    SEQUENCE {
        version
            INTEGER { v2u(2) },

        parameters
            OCTET STRING,

        data
            CHOICE {
                plaintext PDUs,
                encrypted OCTET STRING
            }
    }
```

The first octet in the parameters component is the *model*, where the value 1 refers to that USEC model. In that case, the *parameters* field contains several values encoded in network-byte order:

- *qoS*, the message's quality-of-service;
- *agentID*, the unique identifier for the agent;
- *agentBoots/agentTime*, the message's timestamp;
- *maxSize*, the maximum message size which the sender of this message can receive using the same transport domain as used for this message;
- *userLen/userName*, the user on whose behalf this message is sent;
- *authLen/authDigest*, the authentication digest; and,
- *contextSelector*, the context selector, which in combination with agentID identifies the SNMPv2 context containing the management information referenced by the SNMPv2 message.

The *plaintext* field contains an SNMPv2 PDU as defined in RFC 1905, whilst the *encrypted* field contains the encrypted form of an SNMPv2 PDU.

Contexts and Context Selectors

An SNMPv2 context is a collection of management information accessible by an SNMPv2 agent. An item of management information may exist in more than one context. An SNMPv2 agent potentially has access to many contexts. Each SNMPv2 message contains a context selector which unambiguously identifies an SNMPv2 context accessible by the SNMPv2 agent whose agentID is contained in the message.

A context is termed a local SNMPv2 context if it is realized by an SNMPv2 entity that uses locally-defined mechanisms to access the management information identified by the SNMPv2 context.

The term remote SNMPv2 context is used at an SNMPv2 manager to indicate a SNMPv2 context which is not realized by the local SNMPv2 entity (i.e., the local SNMPv2 entity uses neither locally-defined mechanisms, nor acts as a proxy SNMPv2 agent to access the management information identified by the SNMPv2 context). The USEC model also defines proxy SNMPv2 contexts.

The combination of an agentID value and a context selector provides for a unique identification of a context within an administrative domain.

Error Reporting

While processing a message, an SNMPv2 agent entity may determine that the contents of the message header is unacceptable according to one of the requirements of the USEC model. Rather than just discarding the received message, and forcing the management entity to await a timeout period to detect that an error condition has occurred, the agent may generate a message containing SNMPv2's new `report` PDU.

When the agent entity detects an error (e.g., the received message has a bad authentication digest) and the qoS indicates that a `report` may be generated, then after incrementing the appropriate error statistic, a `report` PDU message is generated. The `report` is generated with the same user and context as the received message and is sent to the same transport address as the received message. All error reports, except those generated due to a not-in-time-window error condition, are unauthenticated (i.e., qoS is noAuth/noPriv). To allow a management entity to authentically synchronize its time with the agent's time, those error reports generated due to a not-in-time-window error condition are authenticated (i.e., qoS is Auth/noPriv).

Upon receiving a `report` PDU a management entity may perform any error recovery actions that are appropriate, such as performing automatic error recovery (i.e.: clock synchronization) or notifying the management station operator of the error condition.

The report flag in the qoS may only be set if the message contains a `get`, `get-next`, `get-bulk`, or `set` operation. The report flag should never be set for a message that contains a `response`, `inform`, `trap`, or `report` operation. Furthermore, a `report` PDU is never sent by an SNMPv2 entity acting in a manager role.

Message Authenticity

To ensure message integrity and to verify the identity of the user on whose behalf a message is sent, the MD5 message digest algorithm described in RFC 1321 is used as follows:

- the user's 128-bit secret key, *authPrivateKey*, known to both the sender of the message and the receiver of the message is inserted into and appended to the SNMPv2 message.

- using the MD5 message digest algorithm, a 128-bit digest (checksum) is computed over the entire message and appended secret. The computed digest is inserted into the message, replacing the secret value, and the resulting message is transmitted to the recipient.

- the recipient of the message replaces the 128-bit digest with the secret value, saving the digest for later use, and appends the secret value to the received message. Using the MD5 message digest algorithm, a 128-bit digest is computed over the entire message and the appended secret. The computed digest value is compared to the received digest value and if they are equal then the message's integrity is intact and its identity of origin is deemed to be authentic.

Any message that is not authentic is discarded, possibly causing a `report` PDU message to be generated.

In comparison, the party model did not append the 128-bit key to the message before the digest was computed. This usage of MD5 is called *Keyed-MD5*, and, according to security experts, it cryptographically strengthens the algorithm. Furthermore, a user's keys are *localized* for each agent. This gives superior security properties, as outlined in the next article.

Timeliness of Delivery

Each SNMPv2 agent is the authoritative source of two time values, *agentBoots* and *agentTime*, which taken together provide an indication of time at that agent. When a manager wishes to authentically communicate with the agent, it must include its notion of both of these values in the message. On receipt of a message at the agent, the agentBoots and agentTime values are checked to ensure that they are within an acceptable time window (150 seconds) of the agent's current time.

When an agent generates a message, its current values of agentBoots and agentTime are always included in the message. When an authentic message is received by the manager, the agentBoots and agentTime values in the message are checked to ensure that they are within an acceptable time window of the manager's local values for the agent's time.

Any message that is not within the acceptable time window is discarded, possibly causing a report PDU message to be generated.

Replay Protection

As discussed previously, each SNMPv2 agent must maintain three objects, agentID, agentBoots, and agentTime.

The agentID value is used to protect against attacks in which a message from a manager is replayed to different agent and/or messages from one agent are replayed as if from a different agent. Since agentID is unique within an administrative domain and the agentID is included in the portion of a message that is authenticated, the same message from/to different agents will contain a different MD5 digest, even if the same authPrivateKey is used. This prevents messages from being cross-played.

To protect against replay, authentic messages are checked to ensure that they are timely. A management entity will accept a received message as timely:

- if the received value of agentBoots is greater than the local notion of agentBoots; or

- if the received value of agentBoots is equal to the local notion of agentBoots and the received agentTime is not more than 150 seconds less than the local notion of agentTime.

Simply stated, the timely message is one that contains an indication of time that may be greater than the local notion of the agent's time and is no more than 150 seconds older than the local notion of the agent's time.

An agent entity will accept a message as timely if the message contains a value of agentBoots equal to the agent's current value of agentBoots and the message contains a value of agentTime that is within 150 seconds of the agent's current agentTime. The agent checks that the received time is strictly within the 150 second window to protect against a manager irresponsibly using an indication of time that is at some arbitrary point in the future, thus allowing captured messages to be replayed until they are no longer timely.

In order for the mechanisms described here to reliably protect against reply attacks, the indication of time at an agent must be ever increasing once the agent is installed and running. Through the use of agentBoots, a non-volatile clock which ticks at all times (even when the agent is powered down) is not required at the agent. The agentBoots value is simply incremented when the agent re-initializes and the agent's indication of time has then advanced, thus providing protection against replay attacks.

Both agentID and agentBoots must be stored in non-volatile storage. If the agent cannot determine the values of agentID or agentBoots then the value of agentBoots should be set to its maximal value (4294967295).

When an agent is first installed, it sets its local values of agentBoots and agentTime to zero. If agentTime ever reaches its maximum value (2147483647) then agentBoots is incremented, as if the agent had rebooted, and agentTime is reset to zero and starts incrementing again. If agentBoots reaches it maximum value (4294967295) manual intervention is required and the agent must be physically visited and re-configured, either with a new agentID value, or with new secret values for all users known to that agent. Note that it would take 136 years of the agent rebooting once a second for this condition to ever arise!

Privacy

To ensure that messages are protected from disclosure from unauthorized sources, the USEC model employs the Data Encryption Standard (DES) in the Cipher Block Chaining (CBC) mode of operation. A 128-bit privacy key, known by both the sender and receiver of a message, is used to encrypt the PDU portion of the message prior to sending. Upon receiving an encrypted message the same privacy key is required to successfully decrypt the PDU portion of the message.

Time Synchronization

Time synchronization is required by a management entity in order to proceed with authentic communications with an agent entity (including being able to check the authenticity of a received `trap`). A management entity has achieved time synchronization with an agent entity when the management entity has obtained local values of agentBoots and agentTime from the agent that are within the agent's time window. In addition to keeping a local version of agentBoots and agentTime, a manager must also keep one other local variable, *latestReceivedAgentTime*. This value records the highest value of agentTime that was received by the manager from the agent and is used to eliminate the possibility of replaying messages that would prevent the manager's notion of the agentTime from advancing.

In order for a manager to become synchronized with an agent, the manager should set its local values of the agent's clocks (agentBoots, agentTime, and latestReceivedAgentTime) to zero. A subsequent authenticated message sent to the agent will cause an authentic error report to be returned to the manager. The error report, indicating that the manager sent a message that was not in the agent's time window, contains authentic values of the agent's clocks that may be used to update the manager's local notion of the agent's clocks. The manager may then continue with timely authentic communications, and, in particular, may re-send the authentic message that caused time synchronization to occur.

The manager and agent each have a clock that advance independently. For a manager to remain synchronized with the agent, its notion of the agent's time must remain current. If the manager's clock does not advance at the same rate as the agent's clock, then over time the manager's notion of the agent's time could vary enough that time synchronization will be lost. To prevent this condition, the manager's notion of the agent's time is updated (i.e., synchronized) whenever a message is received that is authentic, timely, and more recent than any other message received from that agent. Thus, a manager that maintains communication with an agent should never lose time synchronization with that agent.

Discovery

In order to communicate with an SNMPv2 agent that supports the USEC model, a management entity must know the value of the agent's agentID value. The agentID may be learned by sending a noAuth/noPriv retrieval communication to the agent with the agentID set to all zeros (binary). Since the context, identified by the combination of the agentID and contextSelector, is invalid due to an all-zero agentID, the agent's response to this message will be a unknown-context error `report` PDU that contains the agent's agentID value in the parameters field. If authentic communications are required, then the time synchronization procedure should then be used.

Access Policy

For a particular SNMPv2 context to which a user has access using a particular qoS, that user's access rights are given by a list of authorized operations (e.g., `get`, `set`, and so on), and for a local context, a read-view and a write-view. The read-view is the set of managed object instances that may be accessed by the user during a retrieval or notification operation. The write-view is the set of managed object instances that may be accessed by the user when performing a `set` operation.

USEC MIB module

The USEC model defines a MIB module that contains objects for basic agent instrumentation (e.g., agentID, agentBoots, and so on) and for USEC statistics.

Informs, Managers, and Agents

One of the new features in SNMPv2 is the `inform` PDU, which is used to transmit management information from one "application" to another. Each of these applications act in a manager role for the purposes of sending and receiving the inform; however, the sending application must have access to information in a MIB view of an entity acting in an agent role. That implies that an application that sends an `inform`:

- is a *dual-role entity*, namely, it acts as both an agent entity and a management entity;
- must have access to the agent's MIB view; and,
- must have access to parts of the agent's instrumentation, specifically, to the agent's authoritative time indicators.

The concept here builds upon the fact that every (network) component should have an agent which holds its management instrumentation, and this is just as true for management applications as for any other network component. Such management instrumentation is, of course, accessible through an agent. Thus, every application should have an agent which holds its management instrumentation, and therefore every application has an associated agent with a MIB view which is used for the purpose of that application sending an `inform`.

With this definition, some may ask why not consider an `inform` as an "acknowledged trap". The answer is related to two of the age-old SGMP/SNMP philosophies:

- keeping the "cost-of-entry" requirements on an agent to a minimum so that even the simplest of devices can afford to implement an agent; and,
- controlling the amount of SNMP traffic generated in a network according to the needs of the management applications, i.e., only a subset of the devices in the network will generate unsolicited SNMP messages upon the occurrence of some network error. (Note that it is this philosophy which has always been the impetus behind SNMP's paradigm of *trap-directed polling*.)

It is consistent with these philosophies for certain high-end devices, which have previously been considered to be agents (e.g., RMON probes or routers), to now be considered to be dual-role entities capable of sending informs. What would be problematic is if every device in the network were considered to be a dual-role entity.

The USEC model requires that an application that wishes to authenticate received informs must synchronize its time with the sending application's associated agent. (This requirement is the same as is required to authenticate received traps.)

Past, Present, and Future

Considerable work has been done on USEC since its initial publication in Spring of last year. In particular, while the security of USEC has been strengthened, (e.g., through the use of localized-keys and keyed-MD5), many of the procedures have been streamlined (e.g., clock synchronization).

As of this writing there are four independently-developed and interoperable implementations of the USEC model, two commercial and two openly-available. Given the straight-forward design of USEC, we anticipate other implementations to be available in 2Q96.

The core of the USEC model is quite stable; however, in early 1996 we will begin development of a remote configuration MIB module for USEC. Rather than using the design team approach enjoyed by the USEC model, an open mailing list `<usec-mib-request@fv.com>` is now established for this purpose.

Optimizing Key Distribution

Uri Blumenthal, N. C. Hien, Bert Wijnen
IBM T.J. Watson Research Center

This article describes a key management approach for networks in which several manager entities are communicating securely with a large number of agent entities. In this case, it is possible to make a reasonable compromise to cut down on the number of keys and their storage requirement, with little impact on security.

In the context of SNMP's User-based Security Model, the approach described is exploited by using *localized keys*.

Introduction

First, we define the class of problems this approach can offer a solution to. Then we outline the approach, using one application as an example. Then we evaluate the advantages and drawbacks.

The whole idea stems from the observation that, on one hand, it is highly desirable to minimize the number of keys or passwords in use when humans are involved; while on the other hand, it is very desirable for every machine to have its own key, which it shares with as few correspondents as possible.

A reasonable compromise seems to be to allow a user to have one key, and localize it for every machine he wants to access, via a cryptographically-strong one-way function.

Such compromise is most needed in SNMP's User-based Security Model, where a few network management stations talk to a great many agents, often on behalf of human users. Thus, SNMP will be used as an example during description of this approach, although the approach is not limited to SNMP.

Conditions of Applicability

Consider a network whose management topology is logically viewed as a star, or several overlapping stars. One or more superior entities need to communicate with a large number of subordinate entities (endpoints), but these endpoints don't correspond with each other. It is necessary that these endpoints have some kind of unique names or other publicly known identification marks. For example, in the context of Simple Network Management Protocol, there are large numbers of managed devices (endpoints) containing agents, and one or more management stations communicating with these devices on behalf of network operators and administrators.

A second consideration is to avoid direct key negotiation, and instead initialize cryptographic secrets via an off-line process. Key management subsequently occurs based on actions taken by the superior entities. For example, in the case of SNMP, cryptograhic secrets for agents are generated when an entity is initially configured, and management stations subsequently orchestrate key updates.

Finally, consider an environment in which the superior entities are relatively few and relatively secure. Preferably, these entities only need to retain key information when they are active (i.e., performing management tasks on behalf of an operator or administrator). However, this is not a requirement.

A Compromise

Some protocols allow the administrator to choose between security and usability. One can have:

- a usable configuration with a few keys, which is prone to security failures; or,
- a very secure configuration which no human will be able to use due to the requirement to have a different key for every agent; or,
- anything in between.

However, it is worth nothing that when dealing with large networks, having just a few different keys doesn't solve the problem, and having an individual key for every agent, alas, is not feasible.

Regardless, considering the possible insecurity of Agent installations and their possibly vast number, it is desirable to ensure that if one (or more) agents are compromised, then communications between the NMS and other agents aren't also compromised. Clearly, if each agent had its own key, it would solve the problem, but the price is too high to have each NMS carrying all the keys for all the Agents that it may need to talk to. While it wouldn't be unreasonable to require this from a computer, such burden is too heavy for human operators. On the other hand, if each operator had just one key which is stored on every agent for which access is authorized, then when one agent falls, the whole network security is compromised.

So to achieve both security and convenience, it would be ideal for an operator to have a single key, and every agent to have its own unique key that is somehow derived from that operator's key.

An obvious way to achieve such compromise is to apply a one-way function to the user's key. Such a one-way function needs to be both unique to each agent, and be straightforward and simple at the same time, so a NMS has no problem constructing it as necessary.

Let's say we have a user `A` with key `Ka`, and an agent `S` with an identifier `Is`. Then a localized key `Ka,s` for this user and agent combination can be derived via `Ka,s = Fs(Ka)`, where `Fs(x)` is a well-known one-way function unique for agent `S`.

Let us define `Fs(x) = F(x,Is)`, where `F(x,y)` is based on any well-known cryptographically secure hash-function, such as MD5 (RFC 1321) or SHS. If we choose MD5, as it is already used in SNMP, then:

```
F(x,y) = MD5(x || y || x)
```

So when a user `A` needs to access an agent `S`, it enters his key `Ka` and Agent's identification `Is`. Based on this, the NMS derives the key:

```
Ka,s = MD5(Ka || Is || Ka)
```

and uses this key for all the communications done on behalf of this user to this agent.

When a key for a user A is to be changed, the NMS gets the new key, localizes it using the above formula for every relevant agent, and sends the new keys for each agent via appropriate channels.

Evaluation and Conclusions

SNMP requires that each agent stores the keys for every user it needs to communicate with. By using our approach, the keys stored in any agent can't be derived from keys in possession of any other agent, neither can the user's key be derived from any agent's key, or several of them. This protects the network when one agent (or more) is compromised. There is no attack known to us, that can break the localization scheme.

Of course, any user needs only know their own key, rather than having to know the key for each agent that may be communicated with. As such, we have achieved the goal of each operator having but a single key.

This article shows a convenient way to distribute and manage keys for some special network configurations, one example of which is SNMP. We showed, that our approach indeed achieves its goal to allow each subordinate entity to have its own key, which is not in any tractable relation with any other subordinate key, nor it is feasible to determine the generating key from subordinate's keys.

Notes on Implementing SNMPv2u

Shawn A. Routhier
Epilogue Technology Corporation

This article describes some experience gained from writing an implementation of SNMPv2u (USEC). The code base utilized for this implementation was Epilogue's product *Envoy*. The base version included support for SNMPv1 and SNMPv2c with some code being copied from an older version that supported the *party* model from the original SNMPv2.

Envoy is a source code product we supply to other companies as a toolkit; our customers then add more code to produce their own network manager or agent. While *Envoy* includes a manager and agent as sample programs, it is the toolkit that is the product. This fact means that most of my effort is spent on the core engine code, and that the manager is a simple command-line program used primarily for demonstration purposes.

For this project I upgraded the core code to understand the SNMPv2u protocol as well as updating the sample agent and manager to use the core to communicate using SNMPv2u. I implemented most of the base protocol, including the authentication services and report features, but did not implement either the privacy or proxy services. I deferred implementing the proxy service due to time constraints. The privacy service is a bigger problem, as it uses the DES encryption algorithm which makes it difficult to export (from the US) in source form. (Unfortunately there probably isn't an appropriately strong algorithm that is exportable.)

Engine Additions And Modifications

Because SNMPv2u basically adds a layer for authentication and security, most of my code didn't require modification. The required changes were limited to the routines that encode and decode packets and the routine that constructs the packet structure to be encoded. I also needed to add MD5 code for computing the authentication digest. However, none of the core agent or manager functions were changed. That is, there were no change to either agent-specific functions such as processing PDUs, determining what objects to process, calling the method routines, and merging the results back into the response pdu; nor to manager-specific functions, such as determining which objects to retrieve, and displaying the response.

The *packet constructor* routine needs to validate some new arguments such as the user-name and the quality-of-service option. It then gathers other information such as the user's authentication key and inserts all of this into the packet structure under construction.

The *encode routine* now needs to determine if the authentication routine needs to be invoked, and, if so, to insert the digest into the packet properly.

The *decode routine* required more work than the other two routines. While its basic function remains the same, the routine must now handle the authentication step (including timestamp manipulation) as well as processing the report pdu.

Finally, I needed to add the MD5 digest support for the encode and decode routines to use. I started with digest code from an older version of *Envoy* but needed to update it to allow for appending the key to the end of the packet in an efficient fashion. (Unlike earlier versions of various approaches to SNMP security, SNMPv2u uses Keyed-MD5.)

Other Modifications and Additions

In addition to the protocol engine changes, I also needed to add routines to manipulate the new database and to modify the manager to use the new packet constructor routine. Due to the limited nature of my manager this had almost no impact. A more sophisticated manager may require more changes to take advantage of some new features such as the report PDU and auto-discovery.

The databases that are needed are for users, contexts, agents and access rights. For a simple agent these can be small and simple (or even non-existent) while a manager will require larger and more complicated tables. For the current project, I choose to use the same relatively simple routines for both my manager and agent sides. In a production environment I would likely use two different schemes.

Additional Comments

Two significant complaints with the original SNMPv2 were: its potentially large amount of non-volatile memory requirements, and the difficulty in dealing with remote clocks. SNMPv2u simplifies both these issues by having a single clock per agent and by using report PDUs along with a small MIB module to allow a manager to gain the required timestamp information. As a consequence, it is considerably easier to implement time synchronization under SNMPv2u than under the *party* mechanism.

Another issue that surfaced during the discussions about the orignal SNMPv2 was the difficulty in changing objects associated with one of the parties being used to authenticate a packet. SNMPv2u neatly sidesteps this issue by mandating that an agent cache the necessary values during the decode step. This simplifies both the manager and the agent.

Under SNMPv2u, key generation procedures are slightly more involved, but are still straightforward to implement. (Unlike earlier versions of various approaches to SNMP security, SNMPv2u uses localized keys.) I imagine that many agents will pre-compute localized keys and save them, while managers may attempt to save space by maintaining a user with a single key and computing a localized key on the fly either per packet or, more likely, per "management session".

Some Conclusions

Implementing the SNMPv2u protocol is neither difficult or complex, the complexity being shifted away from the protocol engine towards a supporting database.

I believe that any successful authentication and security protocol will need to allow simple agents to avoid most of the complexity of the database code. The protocol should also require minimal resources from such simple agents.

Although I found that implementing SNMPv2u was easier than implementing the original SNMPv2, I don't think that the original SNMPv2 died due to implementation difficulty. Rather, I feel it died because one had to deploy all the required party information. SNMPv2u is simpler in this respect, requiring less authentication and security information to be deployed. I believe this will be another requirement for any successful authentication and security protocol.

The SNMP Framework

Keith McCloghrie
Cisco Systems, Inc.

Last autumn, the IETF's SNMPv2 Working Group produced a set of updated SNMPv2 documents based on its work during the preceeding 12 months. Despite the controversy surrounding the last few months of that effort, the good news is that major portions of the previous documents (RFCs 1441 through 1452) proved to be "good technology", and the new documents have only minor improvements in those areas. The bad news is that the other portions of those RFCs, specifically, the administrative framework, the security, and the Manager-to-Manager MIB proved to be unworkable in deployment.

The controversy in the working group was concerned with selecting replacement technology for the administrative and security framework, and as a result, the working group could not reach agreement between several competing proposals. As a last-minute compromise, the working group agreed to a fallback position of using SNMPv1's administrative framework to replace the unworkable portions so as to allow the "good technology" to progress on the standardization track.

This combination of using the SNMPv1 administrative framework with the remainder of SNMPv2 is called *Community-based* SNMPv2 or SNMPv2c. At the last IETF meeting in Dallas, the IESG approved the progression of all but the Introduction document to Draft Standard status, with the Introduction being declared an *experimental* (sic) protocol. All of these approved documents are now published as RFCs (see the Standards Summary).

This and succeeding articles of this column will examine the changes to SNMPv2 as represented by those approved documents. In this article, we look at the SNMPv2c administrative framework as well as the changes to the SNMPv2 protocol and transport mappings. Future articles will examine the changes to the rules for defining MIBs (the SMI, Textual Conventions, and Conformance documents), to the SNMPv2 MIB, and to the co-existence between SNMPv1 and SNMPv2.

The primary problem with the original SNMPv2 administrative framework was the use of SNMPv2 parties. RFC 1445 specified that an SNMPv2 message was sent from one party to another party, thereby allowing the security to be based on which parties were communicating. The source and destination parties were identified in the SNMPv2 message wrapper. Inside the wrapper was the SNMPv2 PDU specifying the operation (`get`, `get-next`, `get-bulk`, `set` and so on) to be performed. The SNMPv2c compromise solution is possible because

SNMPv1 also has a message wrapper which contains a version number, a community string, and an (SNMPv1) PDU.

Introducing SNMPv2c

The *Introduction to Community-Based SNMPv2* document specifies that the SNMPv1 message wrapper is used instead of RFC 1445's party-based wrapper, but unlike SNMPv1 (RFC 1157), inside it contains an SNMPv2 PDU. To avoid the potential of having the contained SNMPv2 PDU cause an SNMPv1 system to become confused, SNMPv2c uses a new version number (1 instead of SNMPv1's 0). Two other minor changes are also specified:

- SNMPv2c allows the source transport address (e.g., the IP address and UDP port number) of Response messages to be any address belonging to an agent; and,
- the error status value of `authenticationError` in a SNMPv2 PDU, use of which was specified by RFC 1445, is not used in SNMPv2c.

As a result of this change, SNMPv2 parties are no longer used, nor defined, in SNMPv2c.

Thus, not only does RFC 1445, the administrative framework become obsolete, but so do: RFC 1446, the security specification, which relied heavily on the definition of parties; RFC 1447, the Party MIB, which allowed configuration of parties; and, RFC 1451, the Manager-to-Manager MIB, which also relied on the definition of parties. All of these RFCs are being declared as having *historic* status. It is expected that some IETF working group will produce a replacement for the the Manager-to-Manager MIB at some future date.

Protocol Changes

The *Protocol Operations* document (RFC 1905) defines SNMPv2's PDU. There are only a few changes, mostly concerned with details, in the updated protocol document. Specifically:

- All references to values defined by the administrative framework are removed. This includes: references to a party's maximum message size which is replaced by the maximum message size supported by the appropriate SNMPv2 implementation over the relevant transport protocol; access control for the sending of traps, which is now a local matter; and, the destinations to which `inform` PDUs are to be sent, which now must be specified by the sending application.
- The addition of a `report` PDU. This new PDU was devised by the working group as a way to speed-up the handling of errors relating to the SNMPv2 message wrapper. With SNMPv2c, no usage is made of the `Report` PDU. However, this PDU's definition is retained in the protocol document so that potential future alternate SNMPv2 administrative frameworks can make use of the faster error recovery it can provide.
- The error checking performed by an agent on receiving a `set` PDU is slightly re-ordered. Experience indicated that the majority of implementations perform their initial checking in variable-independent code followed by checking performed by code dependent on the variable being accessed. The re-ordering supports such implementations by performing all the variable-independent checking first.
- The requirement to increment the relevant error counter (defined in the SNMPv2 MIB) on the appropriate occasions is added.
- The processing of a `get-bulk` PDU is allowed to terminate "early" (i.e., before the completion of the specified number of max-repetitions) on one additional condition. This extra condition is if the processing of the `get-bulk` PDU is taking a significantly greater amount of time than that taken by a normal request.
- The definition of SNMPv2's `noSuchInstance` and `noSuchObject` exception conditions is clarified.

Transport Changes

The *Transport Mappings* document (RFC 1906) specifies how SNMPv2 messages are sent over various transport protocols, and also includes the restrictions on the use of ASN.1's Basic Encoding Rules (BER) for encoding SNMPv2 messages. This document has even fewer changes:

- The discussion of proxy between SNMPv2 parties and SNMPv1 communities is deleted.
- All assignments of OBJECT IDENTIFIER values, e.g., an assignment to a particular transport mapping, are now defined using SNMPv2's OBJECT-IDENTITY macro.
- The BER encoding of the new BITS construct is explained.

In the next issue of ***The Simple Times***, we'll look at changes to SNMP's language for describing managed objects.

Frequently Asked Questions

Kaj Tesink
Bell Communications Research

Welcome to the FAQ column. This column features interesting questions, sometimes posted on one of the mailing list, or sometimes found on the grapevine.

Here are two questions from the SNMP list that pointed out some fine thinking, and show that there is sometimes more behind a specification than meets the eye.

Q: MIN-ACCESS, RowStatus

I have a question regarding the SNMPv2c *Conformance Statements* document (RFC 1904).

In section 5.4.3.3, "Mapping of the MIN-ACCESS clause", it indicates that the MIN-ACCESS clause may not be used in a MODULE-COMPLIANCE macro for objects where the level of access is "fixed according to their syntax definition". The example given is an object with the syntax of the textual convention of RowStatus.

Now, I have defined a conceptual table within my MIB that is mandatory and it includes a RowStatus columnar object. The intention of the table is to allow managers to add/delete rows, but I fully expect some agent implementations to provide a limited version of this table that does not allow addition/deletion. I expected that the MODULE-COMPLIANCE macro would allow this, but it appears that it doesn't.

Is this really the intention, or am I missing something?

A: Keith McCloghrie responds:

In the text you quote, the examples given are:

1. conceptual tables and rows since no value other than not-accessible is legal;
2. Counter32 and Counter64 for which read-only is the maximal level of access; and,
3. "certain types of textual conventions", giving RowStatus as an example.

This text is in RFC 1444 and is unchanged in RFC 1904.

But, note that RFC 1573, RFC 1604, RFC 1698, and RFC 1747 all define RowStatus objects for which they have conformance statements with a MIN-ACCESS.

Bottom-line: I think RowStatus needs to be deleted as an example in Section 5.4.3.3.

Q: SetRequest, wrongEncoding

Hi everyone,

In reviewing the latest draft of the proto document, I noticed the following parenthetical statement in Step (5) of first phase SetRequest processing (pp. 22-23):

> "Otherwise, if the variable binding's value field contains an ASN.1 encoding which is inconsistent with that field's ASN.1 tag, then the value of the Response-PDU's error-status field is set to 'wrongEncoding', and the value of its error-index field is set to the index of the failed variable binding. (Note that not all implementation strategies will generate this error.)"

Can someone please explain the reasoning behind that parenthetical caveat? What cases is it supposed to cover...what alternative action (error-code) would one of these implementation strategies report? Or is it intended that an implementation strategy might just silently fix-up the encoding error if it can?

A: Brian O'Keefe responds:

Some implementations ASN.1 decode the entire message before processing it at all. In this case, the implementation will have aborted processing the message long before this step in the proto ops. The resulting behavior would be similar to the case where an encoding error is detected in the message header (i.e., during authentication phase).

Industry Comment

Marshall T. Rose
Dover Beach Consulting, Inc.

The Simple Times is back!

After 18 months of wretched inactivity, we're back in publication. You'll notice two changes: a new format and more content.

A New Format

The ***The Simple Times*** is available in an HTML edition. When publication began in early 1992, Internet hypertext had yet to emerge. So, publication was limited to an ASCII and a PostScript edition. Considering that a substantial portion of ***The Simple Times*** content is resource pointers, supporting HTML provides a powerful combination. Of course, ***The Simple Times*** has its own hypertext index `http://www.simple-times.org/pub/simple-times/issues/`.

For another example, check out the *unofficial* index `http://www.simple-times.org/pub/simple-times/html/` of IETF MIB modules. There you will find every MIB module developed by the IETF, in full hypertext format. Reading a MIB module and need to contact the module's editor or look up an object defined in another module? Just click.

It's a pity we didn't have Internet hypertext in the early days, because the Standards Summary really benefits from it! Speaking of which, here are three other hypertext resources of interest to the SNMP community:

- IETF Home Page
  `http://www.ietf.cnri.reston.va.us/`
- SNMP Testing FAQ
  `http://www.iwl.com/faq.html`
- User-based Security Model (USEC) Resources
  `http://www.simple-times.org/pub/simple-times/usec/`

More Content

The editorial composition of ***The Simple Times*** has changed. This time we're placing more emphasis on outside contributions instead of featured columns.

Previously, only one outside technical article per issue was solicited. The new editorial policy allows each issue to have up to one technical article in each of the following areas:

- *Applications*, in which the SNMP framework is used for networking management;
- *Tools*, such as development environments, testing suites, and so on; and,
- *Operations*, describing how to provision the service, deploy the products, and manage effectively.

Our goal this time is to focus less on pure SNMP technology and more on the use of SNMP for networking management.

Of course, each issue will still contain considerable information about SNMP:

- Keith McCloghrie `<kzm@cisco.com>` discusses the SNMP framework, explaining the three core models: management information, protocol operation, and administrative infrastructure;
- Kaj Tesink `<kaj@mail.bellcore.com>` edits the most frequently asked questions column (be sure to send him your favorite FAQs); and,
- the editor `<st-editorial@simple-times.org>`, gets on a soapbox to skewer the unholy and strike fear in the hearts of the connection-oriented!

But, ***The Simple Times*** needs your help: please consider contributing a technical article to the community! The publication schedule is quarterly, so that's plenty of time for you to do some serious writing.

Standards Summary

SNMPv1 Framework

Consult the latest version of *Internet Official Protocol Standards*. As of this writing, the latest version is RFC 1880.
Full Standards:

- RFC 1155 - Structure of Management Information (SMI);
- RFC 1157 - Simple Network Management Protocol (SNMP);
- RFC 1212 - Concise MIB definitions; and,
- RFC 1213 - Management Information Base (MIB-II).

Proposed Standards:

- RFC 1418 - SNMP over OSI;
- RFC 1419 - SNMP over AppleTalk; and,
- RFC 1420 - SNMP over IPX.

SNMPv2 Framework

Draft Standards:

- RFC 1902 - SMI for SNMPv2;
- RFC 1903 - Textual Conventions for SNMPv2;
- RFC 1904 - Conformance Statements for SNMPv2;
- RFC 1905 - Protocol Operations for SNMPv2;
- RFC 1906 - Transport Mappings for SNMPv2;
- RFC 1907 - MIB for SNMPv2; and,
- RFC 1908 - Coexistence between SNMPv1 and SNMPv2.

(In addition, MIB-II has been split into MIB modules for IP, TCP, and UDP; these documents will be published shortly as draft standards.)

Proposed Standards:

- RFC 1573 - Evolution of the Interfaces Group of MIB-II; and,
- RFC 1354 - IP Forwarding Table MIB.

Experimental:

- RFC 1901 - Introduction to Community-based SNMPv2.

MIB Modules

An *unofficial* index of IETF MIB modules is available.
http://www.simple-times.org/pub/simple-times/html/
Full Standards:

- RFC 1643 - Ether-Like Interface Type (SNMPv1).

Draft Standards:

- RFC 1493 - Bridge MIB;
- RFC 1516 - IEEE 802.3 Repeater MIB;
- RFC 1559 - DECnet phase IV MIB;
- RFC 1657 - BGP version 4 MIB;
- RFC 1658 - Character Device MIB;
- RFC 1659 - RS-232 Interface Type MIB;
- RFC 1660 - Parallel Printer Interface Type MIB;
- RFC 1694 - SMDS Interface Protocol (SIP) Interface Type MIB;
- RFC 1724 - RIP version 2 MIB;
- RFC 1742 - AppleTalk MIB;
- RFC 1748 - IEEE 802.5 Token Ring Interface Type MIB;
- RFC 1757 - Remote Network Monitoring MIB; and,
- RFC 1850 - OSPF version 2 MIB.

Proposed Standards:

- RFC 1285 - FDDI Interface Type (SMT 6.2) MIB;
- RFC 1315 - Frame Relay DTE Interface Type MIB;
- RFC 1381 - X.25 LAPB MIB;
- RFC 1382 - X.25 PLP MIB;
- RFC 1406 - DS1/E1 Interface Type MIB;
- RFC 1407 - DS3/E3 Interface Type MIB;
- RFC 1414 - Identification MIB;
- RFC 1461 - Multiprotocol Interconnect over X.25 MIB;
- RFC 1471 - PPP Link Control Protocol (LCP) MIB;
- RFC 1472 - PPP Security Protocols MIB;
- RFC 1473 - PPP IP Network Control Protocol MIB;
- RFC 1474 - PPP Bridge Network Control Protocol MIB;
- RFC 1512 - FDDI Interface Type (SMT 7.3) MIB;
- RFC 1513 - Token Ring Extensions to RMON MIB;
- RFC 1514 - Host Resources MIB;
- RFC 1515 - IEEE 802.3 Medium Attachment Unit (MAU) MIB;
- RFC 1525 - Source Routing Bridge MIB;
- RFC 1565 - Network Services Monitoring MIB;
- RFC 1566 - Mail Monitoring MIB;
- RFC 1567 - X.500 Directory Monitoring MIB;
- RFC 1595 - SONET/SDH Interface Type MIB;
- RFC 1604 - Frame Relay Service MIB;
- RFC 1611 - DNS Server MIB;
- RFC 1612 - DNS Resolver MIB;
- RFC 1628 - Uninterruptible Power Supply MIB;
- RFC 1650 - Ether-Like Interface Type (SNMPv2);
- RFC 1666 - SNA NAU MIB;
- RFC 1695 - ATM MIB;
- RFC 1696 - Modem MIB;
- RFC 1697 - Relational Database Management System MIB;
- RFC 1747 - SNA DLC MIB;
- RFC 1749 - 802.5 Station Source Routing MIB; and,
- RFC 1759 - Printer MIB.

Experimental:

- RFC 1187 - Bulk table retrieval with the SNMP;
- RFC 1224 - Techniques for managing asynchronously generated alerts;
- RFC 1238 - CLNS MIB; and,
- RFC 1592 - SNMP Distributed Program Interface (SNMP-DPI).

Informational:

- RFC 1215 - A convention for defining traps for use with the SNMP;

- RFC 1270 - SNMP communication services;
- RFC 1303 - A convention for describing SNMP-based agents;
- RFC 1321 - MD5 message-digest algorithm;
- RFC 1470 - A network management tool catalog; and,
- RFC 1503 - Automating Administration in SNMPv2 Managers.

Historic:

- RFC 1156 - Management Information Base (MIB-I) (see RFC 1213);
- RFC 1161 - SNMP over OSI (see RFC 1418);
- RFC 1227 - SNMP MUX protocol and MIB;
- RFC 1228 - SNMP Distributed Program Interface (SNMP-DPI) (see RFC 1592);
- RFC 1229 - Extensions to the generic-interface MIB (see RFC 1573);
- RFC 1230 - IEEE 802.4 Token Bus Interface Type MIB;
- RFC 1231 - IEEE 802.5 Token Ring Interface Type MIB (see RFC 1748);
- RFC 1232 - DS1 Interface Type MIB (see RFC 1406);
- RFC 1233 - DS3 Interface Type MIB (see RFC 1407);
- RFC 1239 - Reassignment of experimental MIBs to standard MIBs;
- RFC 1243 - AppleTalk MIB (see RFC 1742);
- RFC 1252 - OSPF version 2 MIB (see RFC 1253);
- RFC 1253 - OSPF version 2 MIB (see RFC 1850);
- RFC 1269 - BGP version 3 MIB (see RFC 1657);
- RFC 1271 - Remote LAN Monitoring MIB (see RFC 1757);
- RFC 1283 - SNMP over OSI (see RFC 1418);
- RFC 1284 - Ether-Like Interface Type MIB (see RFC 1398);
- RFC 1286 - Bridge MIB (see RFC 1493 and RFC 1525);
- RFC 1289 - DECnet phase IV MIB (see RFC 1559);
- RFC 1298 - SNMP over IPX (see RFC 1420);
- RFC 1304 - SMDS Interface Protocol (SIP) Interface Type MIB (see RFC 1694);
- RFC 1316 - Character Device MIB (see RFC 1658);
- RFC 1317 - RS-232 Interface Type MIB (see RFC 1659);
- RFC 1318 - Parallel Printer Interface Type MIB (see RFC 1660);
- RFC 1351 - SNMP Administrative Model;
- RFC 1352 - SNMP Security Protocols;
- RFC 1353 - SNMP Party MIB;
- RFC 1368 - IEEE 802.3 Repeater MIB (see RFC 1516);
- RFC 1389 - RIPv2 MIB (see RFC 1724);
- RFC 1398 - Ether-Like Interface Type MIB (see RFC 1643);
- RFC 1441 - Introduction to SNMPv2 (see RFC 1901);
- RFC 1442 - SMI for SNMPv2 (see RFC 1902);
- RFC 1443 - Textual Conventions for SNMPv2 (see RFC 1903);
- RFC 1444 - Conformance Statements for SNMPv2 (see RFC 1904);
- RFC 1445 - Administrative Model for SNMPv2;
- RFC 1446 - Security Protocols for SNMPv2;
- RFC 1447 - Party MIB for SNMPv2;
- RFC 1448 - Protocol Operations for SNMPv2 (see RFC 1905);
- RFC 1449 - Transport Mappings for SNMPv2 (see RFC 1906);
- RFC 1450 - MIB for SNMPv2 (see RFC 1907);
- RFC 1451 - Manager-to-Manager MIB;
- RFC 1452 - Coexistence between SNMPv1 and SNMPv2 (see RFC 1908);
- RFC 1596 - Frame Relay Service MIB (see RFC 1604);
- RFC 1623 - Ether-Like Interface Type MIB (see RFC 1643); and,
- RFC 1665 - SNA NAU MIB (see RFC 1666).

Subscribing to SNMP-related Working Groups

- 100VG-AnyLAN MIB Working Group
  <vgmib-request@hprnd.rose.hp.com>
- Application MIB Working Group
  <applmib-request@emi-summit.com>
- AToM MIB Working Group
  <atommib-request@thumper.bellcore.com>
- BGP Working Group
  <iwg-request@ans.net>
- Bridge MIB Working Group
  <bridge-mib-request@nsl.dec.com>
- Character MIB Working Group
  <char-mib-request@decwrl.dec.com>
- Data Link Switching MIB Working Group
  <aiw-dlsw-mib@networking.raleigh.ibm.com>
- DECnet Phase IV MIB Working Group
  <phiv-mib-request@jove.pa.dec.com>
- Entity MIB Working Group
  <entmib-request@cisco.com>
- FDDI MIB Working Group
  <fddi-mib-request@cs.utk.edu>
- Frame Relay Service MIB Working Group
  <frftc-request@nsco.network.com>
- Host Resources MIB Working Group
  <hostmib-request@andrew.cmu.edu>
- IEEE 802.3 Hub MIB Working Group
  <hubmib-request@hprnd.rose.hp.com>
- IDR Working Group
  <bgp@ans.edu>
- Interfaces MIB Working Group
  <if-mib-request@dtl.labs.tek.com>
- IP over AppleTalk Working Group
  <apple-ip-request@cayman.com>
- IPLPDN Working Group
  <iplpdn-request@nri.reston.va.us>
- IPv6 MIB Working Group
  <ip6mib-request@research.ftp.com>
- ISDN MIB Working Group
  <isdn-mib-request@combinet.com>
- IS-IS for IP Internets Working Group
  <isis-request@merit.edu>
- Mail and Directory Management Working Group
  <ietf-madman-request@innosoft.com>
- Modem Management Working Group
  <modemmgt-request@telebit.com>
- NOCtools Working Group
  <noctools-request@merit.edu>
- OSPF IGP Working Group
  <ospf-request@gated.cornell.edu>
- PPP Extensions Working Group
  <ietf-ppp-request@merit.edu>
- RIP Working Group
  <ietf-rip-request@xylogics.com>
- Remote Network Monitoring Working Group
  <rmonmib-request@cisco.com>
- Routing over Large Clouds Working Group
  <rolc-request@nexen.com>
- SNA DLC Services MIB Working Group
  <snadlcmib-request@cisco.com>
- SNA NAU Services MIB Working Group
  <snanaumib-request@cisco.com>
- SNMP Agent Extensibility Working Group
  <agentx-request@fv.com>
- SNMPv2 Working Group
  <snmpv2-request@tis.com>
- TCP Client Identity Protocol
  <ident-request@nri.reston.va.us>
- DS1/DS3 MIB Working Group
  <trunk-mib-request@cisco.com>
- Uninterruptible Power Supply Working Group
  <ups-mib-request@cs.utk.edu>
- X.25 MIB Working Group
  <x25mib-request@dg-rtp.dg.com>

Internet Resources

Automated Services

Automated services are available in the Internet, provided "as is" with no express or implied warranty. Each service accepts a MIB module in the body of a message. MIB module checking:

- Emissary <mib-checker@epilogue.com>
- mosy <mosy@simple-times.org>

MIB module conversion:

- convert SNMPv2 module to SNMPv1
  `<mib-v2tov1@simple-times.org>`
- convert MIB module to HTML
  `<mib-2html@simple-times.org>`

Source Implementations

Source implementations are available in the Internet, provided under various no-fee licensing terms.
Agents:

- Beholder: an RMON agent for UNIX
  `ftp://dnpap.et.tudelft.nl/pub/btng/`
- CMU SNMP: an SNMPv2u agent for UNIX
  `ftp://ftp.cisco.com/ftp/kzm/cmusnmp.tar.gz`
- UT-snmpV2: an SNMPv2 agent for SPARCs
  `http://snmp.cs.utwente.nl/`
- WILMA: an SNMP agent for UNIX
  `http://www.ldv.e-technik.tu-muenchen.de/dist/INDEX.html`

Compilers:

- mosy: a MIB compiler
  `ftp://ftp.cisco.com/ftp/kzm/snmptcl.tar.Z`
- SMIC: a MIB compiler
  `<dperkins@scruznet.com>`
- snacc: an ASN.1 compiler
  `ftp://ftp.cs.ubc.ca/pub/local/src/snacc/`

Platforms:

- NOCOL: a network monitoring package for UNIX
  `ftp://ftp.navya.com/pub/vikas/`
- Scotty: a Tcl-based environment for management applications
  `http://www.cs.tu-bs.de/ibr/projects/nm/`
- snmptcl: a Tcl-based environment for management applications
  `ftp://ftp.cisco.com/ftp/kzm/snmptcl.tar.Z`
- Tricklet: a Perl-based environment for management applications
  `ftp://dnpap.et.tudelft.nl/pub/btng/`
- WILMA: an X-based monitoring package for UNIX
  `http://www.ldv.e-technik.tu-muenchen.de/dist/INDEX.html`

Publication Information

Featured Columnists

Keith McCloghrie	Cisco Systems, Inc.
Marshall T. Rose	Dover Beach Consulting, Inc.
Kaj Tesink	Bell Communications Research

Contact Information

E-mail	st-editorial@simple-times.org
ISSN	1060–6068

Submissions

The Simple Times solicits high-quality articles of technology and comment. Technical articles are refereed to ensure that the content is marketing-free. By definition, commentaries reflect opinion and, as such, are reviewed only to the extent required to ensure commonly-accepted publication norms.

The Simple Times also solicits terse announcements of products and services, publications, and events. These contributions are reviewed only to the extent required to ensure commonly-accepted publication norms.

Submissions are accepted only via electronic mail, and must be formatted in HTML version1.0. Each submission must include the author's full name, title, affiliation, postal and electronic mail addresses, telephone, and fax numbers. Note that by initiating this process, the submitting party agrees to place the contribution into the public domain.

Subscriptions

The Simple Times is available in three editions: HTML, ASCII, and PostScript. For more information, send a message to

```
st-subscriptions@simple-times.org
```

with a `Subject:` line of

```
help
```

Back issues are available via either the Web or FTP, i.e.,

```
http://www.simple-times.org
ftp://ftp.simple-times.org
```

look under /pub/simple-times/issues/. In addition, ***The Simple Times*** has several hard copy distribution outlets. Contact your favorite SNMP vendor and see if they carry it.