                   Algorithms for Automating Administration
                          in SNMPv2 Managers

Status of this Memo

   This memo provides information for the Internet community. It does
   not specify an Internet standard. Distribution of this memo is
   unlimited.

Table of Contents

1. Introduction

   When a user invokes an SNMPv2 [1] management application, it may be
   desirable for the user to specify the minimum amount of information
   necessary to establish and maintain SNMPv2 communications. This memo
   suggests an approach to achieve this goal.

2. Implementation Model

   In order to discuss the approach outlined in this memo, it is useful
   to have a model of how the various parts of an SNMPv2 manager fit
   together. The model assumed in this memo is depicted in Figure 2.1.
   This model is, of course, merely for expository purposes, and the

approach should be readily adaptable to other models.

```
                          (Human) User
                               *
                               *
                ===========User Interface (UI)==========
                               *
                    +---------------------------+
              ... | Management Application N |
                  +---------------------------+       |
                  | Management Application 2  |-----+
              +---------------------------+       |    *
              | Management Application 1 |----+    *
              +---------------------------+  *      *
                                     *      *      *
                ========Management API====================
                   *                              *
                   *            _____          *
            +------------+    /  Local  \    +---------------+
            | Context    |***/  Party    \***| SNMP protocol |
            | Resolver(s) |  \ Database /    |   engine(s)   |
            +------------+    _____/     +---------------+
                                                    *
                                                    *
                    ==========Transport APIs===========
                                    *
                  +--------------------------------+
                  | Transport Stacks (e.g., UDP/IP) |
                  +--------------------------------+
                                    *
                              Network(s)
```
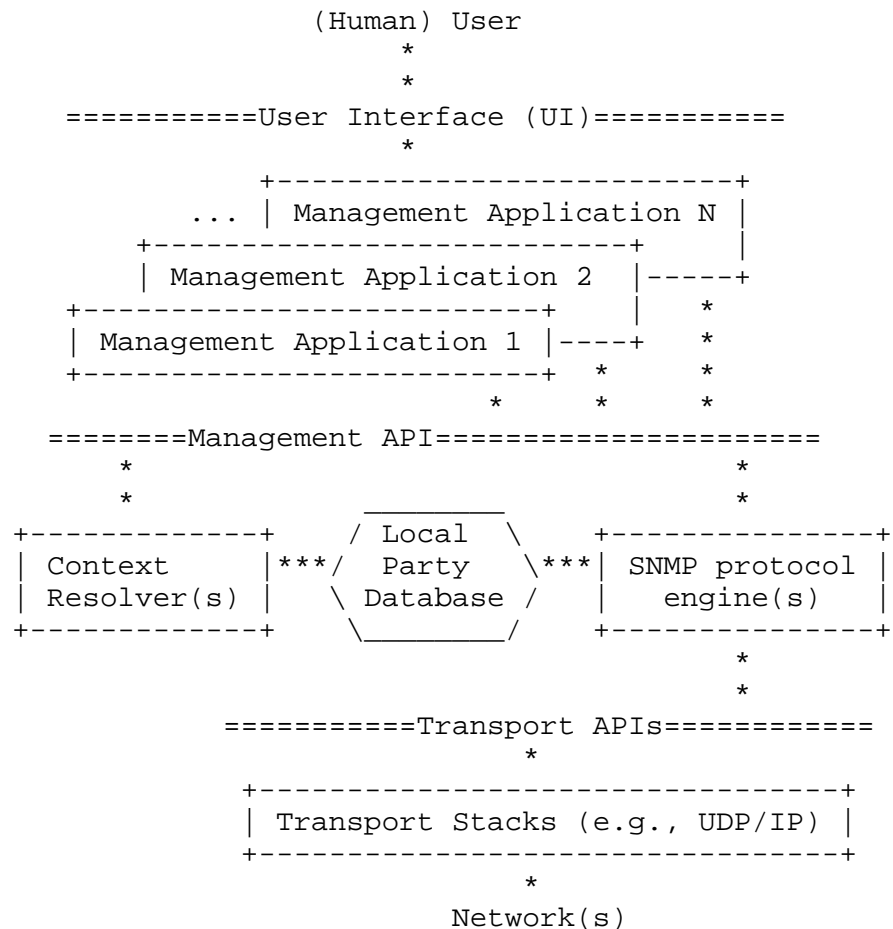
              Figure 2.1 SNMPv2 Manager Implementation Model

   Note that there might be just one SNMP protocol engine and one
   "context resolver" which are accessed by all local management
   applications, or, each management application might have its own SNMP
   protocol engine and its own "context resolver", all of which have
   shared access to the local party database [2].

   In addition to the elements shown in the figure, there would need to
   be an interface for the administrator to access the local party
   database, e.g., for configuring initial information, including
   secrets. There might also be facilities for different users to have
   different access privileges, and/or other reasons for there to be
   multiple (coordinated) subsets of the local party database.

3. Configuration Assumptions

   Now, let's assume that the administrator has already configured a
   local party database for the management application, e.g.,

```
            partyIdentifier:        initialPartyId.a.b.c.d.1
            partyIndex:             1
            partyTAddress:          a.b.c.d:161
            partyLocal:             false
            partyAuthProtocol:      noAuth
            partyPrivProtocol:      noPriv

            partyIdentifier:        initialPartyId.a.b.c.d.2
            partyIndex:             2
            partyTAddress:          local address
            partyLocal:             true
            partyAuthProtocol:      noAuth
            partyPrivProtocol:      noPriv

            partyIdentifier:        initialPartyId.a.b.c.d.3
            partyIndex:             3
            partyTAddress:          a.b.c.d:161
            partyLocal:             false
            partyAuthProtocol:      md5Auth
            partyPrivProtocol:      noPriv

            partyIdentifier:        initialPartyId.a.b.c.d.4
            partyIndex:             4
            partyTAddress:          local address
            partyLocal:             true
            partyAuthProtocol:      md5Auth
            partyPrivProtocol:      noPriv

            contextIdentifier:      initialContextId.a.b.c.d.1
            contextIndex:           1
            contextLocal:           false
            textual handle:         router.xyz.com-public

            contextIdentifier:      initialContextId.a.b.c.d.2
            contextIndex:           2
            contextLocal:           false
            textual handle:         router.xyz.com-all

            aclTarget (dest. party): 1
            aclSubject (src party):  2
            aclResources (context):  1
            aclPrivileges:          get, get-next, get-bulk
```

```
                    aclTarget (dest. party): 3
                    aclSubject (src party):  4
                    aclResources (context): 2
                    aclPrivileges:           get, get-next, get-bulk, set
```

   Note that each context has associated with it a "textual handle".
   This is simply a string chosen by the administrator to aid in
   selecting a context.

4. Normal Operations

   When the user tells the management application to do something, the
   user shouldn't have to specify party or context information.

   One approach to achieve this is as follows: the user provides a
   textual string indicating the managed objects to be manipulated, and
   the management application invokes the "context resolver" to map this
   into a "context handle", and later, when an SNMPv2 operation is
   performed, the "context handle" and a minimal set of security
   requirements are provided to the management API.

4.1. Getting a Context Handle

   A "context handle" is created when the management application
   supplies a textual string, that was probably given to it by the user.
   The "context resolver" performs these steps based on the
   application's input:

        (1) In the local party database, each context has associated
             with it a unique string, termed its "textual handle". If
             a context in the local database has a textual handle
             which exactly matches the textual string, then the
             "context resolver" returns a handle identifying that
             context.

             So, if the application supplies "router.xyz.com-public",
             then the "context resolver" returns a handle to the first
             context; instead, if the application supplies
             "router.xyz.com-all", then the "context resolver" returns
             a handle to the second context.

        (2) Otherwise, if any contexts are present whose textual
             handle is longer than the textual string, and whose
             initial characters exactly match the entire textual
             string, then the "context resolver" returns a handle
             identifying all of those contexts.

             So, if the application supplies "router.xyz.com", then

the "context resolver" returns a handle to both contexts.

(3)   Otherwise, if the textual string specifies an IP address
      or a domain name which resolves to a single IP address,
      then the "context resolver" adds to the local party
      database, a volatile noAuth/noPriv party pair, a volatile
      context, and a volatile access control entry allowing
      interrogation operations, using the "initialPartyId" and
      "initialContextId" conventions. The "context resolver"
      returns a handle identifying the newly created context.

      So, if the application supplies "89.0.0.1", then the
      "context resolver" adds the following information to the
      local party database:

            partyIdentifier: initialPartyId.89.0.0.1.1
            partyIndex: 101
            partyTAddress: 89.0.0.1:161
            partyLocal: false
            partyAuthProtocol: noAuth
            partyPrivProtocol: noPriv
            partyStorageType: volatile

            partyIdentifier: initialPartyId.89.0.0.1.2
            partyIndex: 102
            partyTAddress: local address
            partyLocal: true
            partyAuthProtocol: noAuth
            partyPrivProtocol: noPriv
            partyStorageType: volatile

            contextIdentifier: initialContextId.89.0.0.1.1
            contextIndex: 101
            contextLocal: false
            contextStorageType: volatile
            textual handle: 89.0.0.1

            aclTarget (dest. party): 101
            aclSubject (src party): 102
            aclResources (context): 101
            aclPrivileges: get, get-next, get-bulk
            aclStorageType: volatile

      and the "context resolver" returns a handle to the newly
      created context.

(4)   Otherwise, if the textual string specifies a domain name
      which resolves to multiple IP addresses, then for each

such IP address, the "context resolver" adds to the local
party database, a volatile noAuth/noPriv party pair, a
volatile context, and a volatile access control entry
allowing interrogation operations, using the
"initialPartyId" and "initialContextId" conventions.
Then, the "context resolver" returns a handle identifying
all of those newly created contexts.

(5) Otherwise, if the textual string contains a '/'-
     character, and everything to the left of the first
     occurrence of this character specifies an IP address or a
     domain name which resolves to a single IP address, then
     the "context resolver" adds to the local party database,
     a volatile SNMPv1 party, a volatile context, and a
     volatile access control entry allowing interrogation
     operations. (The SNMPv1 community string consists of any
     characters following the first occurrence of the '/'-
     character in the textual string.) Then, the "context
     resolver" returns a handle identifying the newly created
     context.

     So, if the application supplied "89.0.0.2/public", then
     the "context resolver" adds the following information to
     the local party database:

          partyIdentifier: initialPartyId.89.0.0.2.1
          partyIndex: 201
          partyTDomain: rfc1157Domain
          partyTAddress: 89.0.0.2:161
          partyLocal: false
          partyAuthProtocol: rfc1157noAuth
          partyAuthPrivate: public
          partyPrivProtocol: noPriv
          partyStorageType: volatile

          contextIdentifier: initialContextId.89.0.0.2.1
          contextIndex: 201
          contextLocal: false
          contextStorageType: volatile
          textual handle: 89.0.0.2

          aclTarget (dest. party): 201
          aclSubject (src party):   201
          aclResources (context):   201
          aclPrivileges: get, get-next, get-bulk
          aclStorageType: volatile

     and the "context resolver" returns a handle to the the

newly created context.

      (6) Otherwise, if the textual string contains a '/'-
           character, and everything to the left of the first
           occurrence of this character specifies a domain name
           which resolves to multiple IP addresses, then for each
           such IP address, the "context resolver" adds to the local
           party database, a volatile SNMPv1 party, a volatile
           context, and a volatile access control entry allowing
           interrogation operations. (The SNMPv1 community string
           consists of any characters following the first occurrence
           of the '/'-character in the textual string.) Then, the
           "context resolver" returns a handle identifying all of
           those newly created contexts.

      (7) Otherwise, an error is raised.

4.2. Requesting an Operation

Later, when an SNMPv2 operation is to be performed, the management
application supplies a "context handle" and a minimal set of security
requirements to the management API:

      (1) If the "context handle" refers to a single context, then
           all access control entries having that context as its
           aclResources, allowing the specified operation, having a
           non-local SNMPv2 party as its aclTarget, which satisfies
           the privacy requirements, and having a local party as its
           aclSubject, which satisfies the authentication
           requirements, are identified.

           So, if the application wanted to issue a get-next
           operation, with no security requirements, and supplied a
           "context handle" identifying context #1, then acl #1
           would be identified.

      (2) For each such access control entry, the one which
           minimally meets the security requirements is selected for
           use. If no such entry is identified, and authentication
           requirements are present, then the operation will be not
           performed.

           So, if the application requests a get-next operation,
           with no security requirements, and supplies a "context
           handle" identifying context #1, and step 1 above
           identified acl #1, then because acl #1 satisfies the no-
           security requirements, the operation would be generated
           using acl #1, i.e., using party #1, party #2, and context

#1.

(3) Otherwise, all access control entries having the (single)
     context as its aclResources, allowing the specified
     operation, and having a non-local SNMPv1 party as its
     aclTarget, are identified. If no such entry is
     identified, then the operation will not performed.
     Otherwise, any of the identified access control entries
     may be selected for use.

     The effect of separating out step 3 is to prefer SNMPv2
     communications over SNMPv1 communications.

(4) If the "context handle" refers to more than one context,
     then all access control entries whose aclResources refers
     any one of the contexts, are identified. For each such
     context, step 2 is performed, and any (e.g., the first)
     access control entry identified is selected for use. If
     no access control entry is identified, then step 3 is
     performed for each such context, and any (e.g., the
     first) access control entry identified is selected for
     use.

     So, if the application wanted to issue a get-bulk
     operation, with no security requirements, and supplied a
     "context handle" identifying contexts #1 and #2, then
     acls #1 and #2 would be identified in step 1; and, in
     step 2, party #1, party #2, and context #1 would be
     selected.

     However, if the application wanted to issue an
     authenticated get-bulk operation, and supplied a "context
     handle" identifying contexts #1 and #2, then acls #1 and
     #2 would still be identified in step 1; but, in step 2,
     only acl #2 satisfies the security requirement, and so,
     party #3, party #4, and context #2 would be selected.

(5) If no access control entry is identified, then an error
     is raised.

Note that for steps 1 and 3, an implementation might choose to pre-
compute (i.e., cache) for each context those access control entries
having that context as its aclResources.

5. Determining and Using Maintenance Knowledge

When using authentication services, two "maintenance" tasks may have
to be performed: clock synchronization and secret update. These

tasks should be performed transparently, independent of the
management applications, and without user/administrator intervention.
In order to operate transparently, the SNMP protocol engine must
maintain "maintenance knowledge" (knowledge of which parties and
contexts to use). It is useful for this maintenance knowledge to be
determined at run-time, rather than being directly configured by an
administrator.

One approach to achieve this is as follows: the first time that the
SNMP protocol engine determines that it will be communicating with
another SNMPv2 entity, the SNMP protocol engine first consults its
local party database and then interrogates its peer, before engaging
in the actual communications.

Note that with such an approach, both the clock synchronization
knowledge, and the secret update knowledge, associated with a party,
can each be represented as (a pointer to) an access control entry.
Further note that once an implementation has computed this knowledge,
it might choose to retain this knowledge across restarts.

5.1. Determination of Synchronization Knowledge

To determine maintenance knowledge for clock synchronization:

    (1)   The SNMP protocol engine examines each active, non-local,
          noAuth party.

          So, this would be party #1.

    (2)   For each such party, P, all access control entries having
          that party as its aclTarget, and allowing the get-bulk
          operation, are identified.

          So, for party #1, this would be acl #1.

    (3)   For each such access control entry, A, at least one
          active, non-local, md5Auth party, Q, must be present
          which meets the following criteria:

        - the transport domain and address of P and Q are
           identical;

        - an access control entry, B, exists having either: Q as
           its aclTarget and a local party, R, as its aclSubject,
           or, Q as its aclSubject and a local party, R, as its
           aclTarget; and,

        - no clock synchronization knowledge is known for R.

So, for acl #1, party #3 is identified as having the same
transport domain and address as party #1, and being
present as the aclTarget in acl #2, which has local party
#4 as the aclSubject.

(4)   Whenever such a party, Q, is present, then all instances
of the "partyAuthProtocol" and "partyAuthClock" objects
are retrieved via the get-bulk operator using the parties
and context identified by the access control entry, A.

So, party #1, party #2, and context #1 would be used to
sweep these two columns on the agent.

(5)   Only those instances corresponding to parties in the
local database, which have no clock synchronization
knowledge, and are local mdAuth parties, are examined.

So, only instances corresponding to party #4 are
examined.

(6)   For each instance of "partyAuthProtocol", if the
corresponding value does not match the value in the local
database, then a configuration error is signalled, and
the corresponding party is marked as being unavailable
for maintenance knowledge.

So, we make sure that the manager and the agent agree
that party #4 is an md5Auth party.

(7)   For each instance of "partyAuthClock", if the
corresponding value is greater than the value in the
local database, then the authentication clock of the
party is warped according to the procedures defined in
Section 5.3 of [3]. Regardless, A is recorded as the
clock synchronization knowledge for the corresponding
party.

So, if the column sweep returns information for party #4,
then party #4's authentication clock is advanced if
necessary, and the clock synchronization knowledge for
party #4 is recorded as acl #1.

5.2. Use of Clock Synchronization Knowledge

Whenever a response to an authenticated operation is not received,
the SNMP protocol engine may suspect that a clock synchronization
problem for the source party is the cause [3]. The SNMP protocol
engine may use different criteria when making this determination; for

example: on a retrieval operation, the operation might be retried
using an exponential back-off algorithm; in contrast, on a
modification operation, the operation would not be automatically
retried.

When clock mis-synchronization for a source party, S, is suspected,
if clock synchronization knowledge for S is present, then this
knowledge is used to perform steps 4-7 above, which should retrieve
the instances of the "partyAuthProtocol" and "partyAuthClock" objects
which correspond to S (and perhaps other parties as well). If
information on these objects cannot be determined, then S is marked
as no longer having clock synchronization knowledge. Otherwise, if
the value of the corresponding instance of "partyAuthClock" is
greater than the value in the local database, then the authentication
clock of the party is warped according to the procedures defined in
Section 5.3 of [3], and the original operation is retried, if
appropriate.

So, if traffic from party #4 times out, then a column sweep is
automatically initiated, using acl #1 (party #1, party #2, context
#1).

When clock mis-synchronization for a source party, S, is suspected,
and clock synchronization knowledge for S is not present, then the
full algorithm above can be used. In this case, if clock
synchronization knowledge for S can be determined, and as a result,
"partyAuthClock" value for S in the local database is warped
according to the procedures defined in Section 5.3 of [3], then the
original operation is retried, if appropriate.

5.3. Determination of Secret Update Knowledge

   To determine maintenance knowledge for secret update:

          (1) The SNMP protocol engine examines each active, non-local,
               md5Auth party.

               So, this would be party #3.

          (2) For each such party, P, all access control entries having
               that party as its aclTarget, and allowing the get-bulk
               and set operations, are identified.

               So, for party #3, this would be acl #2.

          (3) For each such access control entry, A, at least one
               active, non-local, md5Auth party, Q, must be present
               which meets the following criteria:

- the transport domain and address of P and Q are
  identical;

- an access control entry, B, exists having either: Q as
  its aclTarget and a local party, R, as its aclSubject,
  or, Q as its aclSubject and a local party, R, as its
  aclTarget; and,

- no secret update knowledge is known for R.

  So, for acl #2, party #3 is (redundantly) identified as
  having the same transport domain and address as party #3,
  and being present as the aclTarget in acl #2, which has
  local party #4 as the aclSubject.

(4) Whenever such a party, Q, is present, then all instances
    of the "partyAuthProtocol", "partyAuthClock", and
    "partyAuthPrivate" objects are retrieved via the get-bulk
    operator using the parties and context identified by the
    access control entry, A.

    So, party #3, party #4, and context #2 would be used to
    sweep these three columns on the agent.

(5) Only those instances corresponding to parties in the
    local database, which have no secret update knowledge,
    and are md5Auth parties, are examined.

    So, only instances corresponding to parties #3 and #4 are
    examined.

(6) For each instance of "partyAuthProtocol", if the
    corresponding value does not match the value in the local
    database, then a configuration error is signalled, and
    this party is marked as being unavailable for maintenance
    knowledge.

    So, we make sure that the manager and the agent agree
    that both party #3 and #4 are md5Auth parties.

(7) For each instance of "partyAuthPrivate", if a
    corresponding instance of "partyAuthClock" was also
    returned, then A is recorded as the secret update
    knowledge for this party.

    So, if the column sweep returned information on party #3,
    then the clock synchronization knowledge for party #3
    would be recorded as acl #2. Further, if the column

sweep returned information on party #4, then the clock
synchronization knowledge for party #4 would be recorded
as acl #2.

5.4. Use of Secret Update Knowledge

   Whenever the SNMP protocol engine determines that the authentication
   clock of a party, S, is approaching an upper limit, and secret update
   knowledge for S is present, then this knowledge is used to modify the
   current secret of S and reset the authentication clock of S,
   according to the procedures defined in Section 5.4 of [3].

   So, whenever the SNMP protocol engine decides to update the secrets
   for party #4, it can automatically use acl #2 (party #3, party #4,
   context #2) for this purpose.

6. Other Kinds and Uses of Maintenance Knowledge

   Readers should note that there are other kinds of maintenance
   knowledge that an SNMPv2 manager could derive and use. In the
   interests of brevity, one example is now considered: when an SNMPv2
   manager first communicates with an agent, it may wish to synchronize
   the maximum-message size values held by itself and the agent.

   For those parties that execute at the agent, the manager retrieves
   the corresponding instances of partyMaxMessageSize (preferrably using
   authentication), and, if need be, adjusts the values held in the
   manager's local party database. Thus, the maintenance knowledge to
   be determined must allow for retrieval of partyMaxMessageSize.

   For those parties that execute at the manager, the manager retrieves
   the corresponding instances of partyMaxMessageSize (using
   authentication), and, if need be, adjusts the values held in the
   agent's local party database using the set operation. Thus, the
   maintenance knowledge to be determined must allow both for retrieval
   and modification of partyMaxMessageSize.

7. Security Considerations

   Security issues are not discussed in this memo.

8. Acknowledgements

   Jeffrey D. Case of SNMP Research and the University of Tennessee, and
   Robert L. Stewart of Xyplex, both provided helpful comments on the
   ideas contained in this document and the presentation of those ideas.

9. References

    [1] Case, J., McCloghrie, K., Rose, M., and S. Waldbusser,
        "Introduction to version 2 of the Internet-standard Network
        Management Framework", RFC 1441, SNMP Research, Inc., Hughes LAN
        Systems, Dover Beach Consulting, Inc., Carnegie Mellon
        University, April 1993.

    [2] McCloghrie, K., and J. Galvin, "Party MIB for version 2 of the
        Simple Network Management Protocol (SNMPv2)", RFC 1447, Hughes
        LAN Systems, Trusted Information Systems, April 1993.

    [3] Galvin, J., and K. McCloghrie, "Security Protocols for version 2
        of the Simple Network Management Protocol (SNMPv2)", RFC 1446,
        Trusted Information Systems, Hughes LAN Systems, April 1993.

10. Authors' Addresses

    Keith McCloghrie
    Hughes LAN Systems
    1225 Charleston Road
    Mountain View, CA 94043
    US

    Phone: +1 415 966 7934
    EMail: kzm@hls.com

    Marshall T. Rose
    Dover Beach Consulting, Inc.
    420 Whisman Court
    Mountain View, CA 94043-2186
    US

    Phone: +1 415 968 1052
    EMail: mrose@dbc.mtview.ca.us