The Simplified Configuration Model for SNMPv2

19 March 1995 |

draft-ietf-snmpv2-scm-ds-00.txt |

Steven Waldbusser
Carnegie Mellon University
waldbusser@cmu.edu

Jeffrey D. Case
SNMP Research, Inc.
case@snmp.com

Keith McCloghrie
Cisco Systems, Inc.
kzm@cisco.com

Marshall T. Rose
Dover Beach Consulting, Inc.
mrose@dbc.mtview.ca.us

Status of this Memo

1. Introduction

A network management system contains: several (potentially many) nodes, each with a processing entity, termed an agent, which has access to management instrumentation; at least one management station; and, a management protocol, used to convey management information between the agents and management stations. Operations of the protocol are carried out under an administrative framework which defines both authentication and authorization policies.

Network management stations execute management applications which monitor and control network elements. Network elements are devices such as hosts, routers, terminal servers, etc., which are monitored and controlled through access to their management information.

The Administrative Infrastructure for SNMPv2 [1] defines how the administrative framework is applied to realize effective network management in a variety of configurations and environments. It is the purpose of this document, the Simplified Configuration Model for SNMPv2, to define one such deployment strategy using the administrative framework.

1.1. A Note on Terminology

For the purpose of exposition, the original Internet-standard Network Management Framework, as described in RFCs 1155, 1157, and 1212, is termed the SNMP version 1 framework (SNMPv1). The current framework is termed the SNMP version 2 framework (SNMPv2).

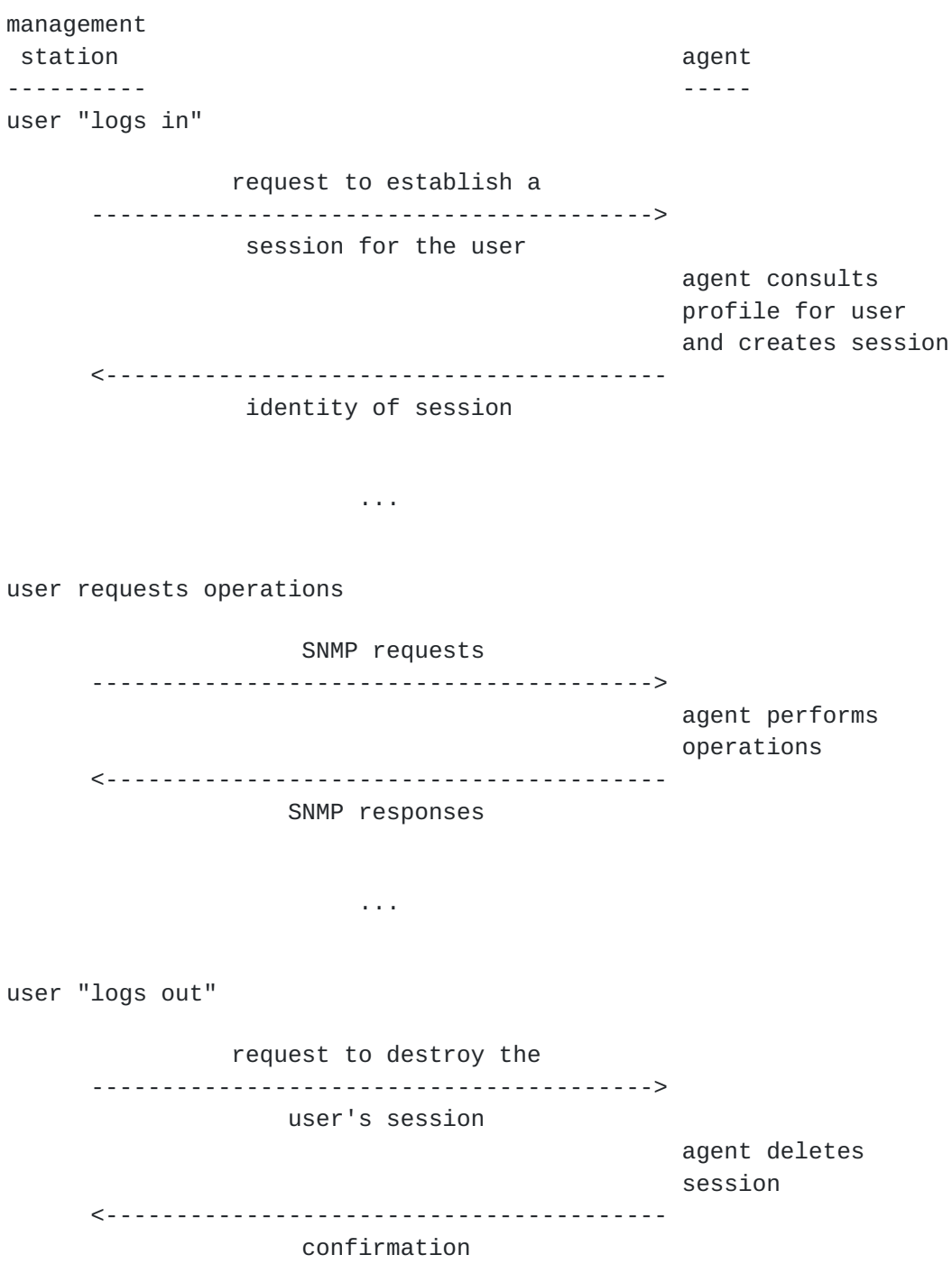
2. Overview

The model describe here is based on the notion of creating transient "management sessions" for use by a management application. Each session is initialized by consulting a user profile, which has been previously configured at the agent. The profile specifies such information as authentication and authorization information, along with a maximum time that a session may be inactive before the agent destroys it.

When a session is created, the SNMPv2 parties and corresponding access control information are dynamically created. These parties are used to perform protocol operations. If the management application completes before the management session expires, it may explicitly destroy the session. Regardless, when a session is destroyed, the corresponding

resources (e.g., parties and ACLs), are deleted by the agent.

Pictorially:



3. User-based Maintenance Functions

Maintenance functions are defined in [1] as a special means for providing controlled access to an SNMPv2 engine in order to perform operations which are not easily accomplished using the administrative infrastructure. The Simplified Configuration Model defines a class of maintenance functions termed "user-based maintenance functions". As with all maintenance functions, the "parties" and "contexts" employed are not accessible to entities which make use of an SNMPv2 engine, nor are they visible through the SNMPv2-PARTY-MIB [2]. These artifacts of SNMPv2's administrative maintenance facility are not actual parties or contexts.

A user-based maintenance function is identified when the context of a management communication has the value userMaintContext, and the source and destination parties identically have the form userMaintParty.<userID>, where <userID> corresponds to an active entry in the scmUserTable, i.e.,

```
dstParty    userMaintParty.<userID>
srcParty    userMaintParty.<userID>
context     userMaintContext
```

Each valid userMaintParty has these characteristics:

```
party           value
-----           -----
AuthProtocol    scmUserAuthProtocol
AuthClock       '7fffffff'H
AuthPrivate     password_to_key(scmUserAuthProtocol,scmUserPassword)
AuthPublic      ''H
PrivProtocol    noPriv
```

To determine the algorithm that maps scmUserPassword to an authentication key, consult the definition of scmUserAuthProtocol in Section 6. Note that since the clock value for these parties is at the maximum, no replay protection is afforded when a user-based maintenance function is performed. Further note that these parties are configured indirectly, by manipulating the scmUserTable -- it is not possible to specify the instances corresponding to a userMaintParty in an SNMP operation.

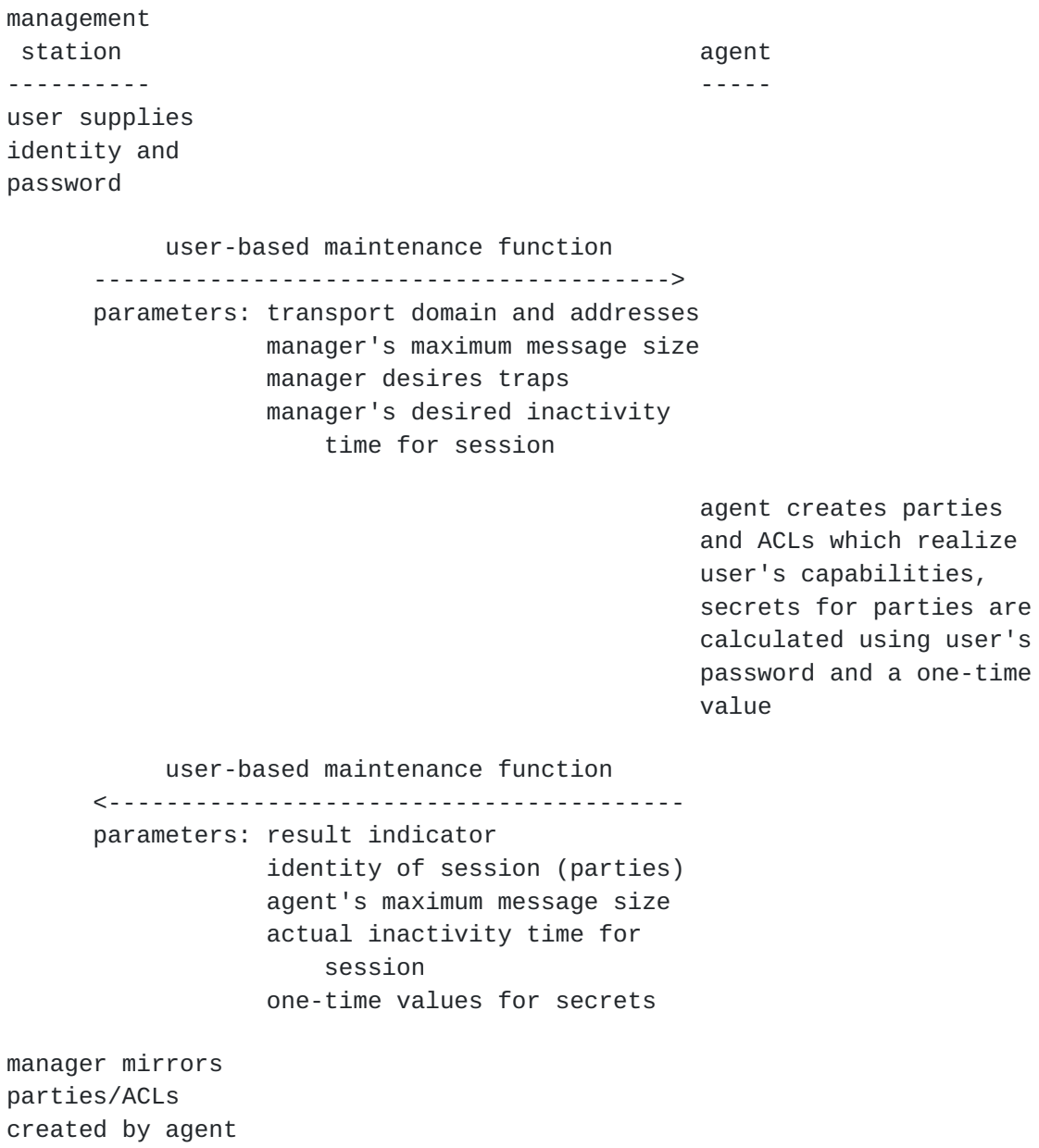
The access allowed to any pairing of userMaintParty.<userID> and userMaintContext is statically defined to be read/write access to all

instances in one (and only one) subtree, userMaintActions.

4. Session Creation Algorithm

Sessions are created when the management station issues a user-based maintenance function, which identifies a user configured at the agent.

Pictorially:



4.1. Step 1: Manager Requests Creation

The manager performs a user-based maintenance function consisting of a getRequest operation containing a variable-binding supplying the parameters of the session to be created. An agent is not required to support such a getRequest hvaing more than one variable-binding. The variable-binding is:

userMaintCreate.<tDomain>.<mAddr>.<aAddr>.<mMMS>.<mTraps>.<mLinger>

where:

<tDomain>
identifies the transport domain to be used for the created parties, encoded as a single sub-identifier, specifically the value of the last sub-identifier of a transport domain defined under snmpDomains [3]:

value	meaning
1	snmpUDPDomain
2	snmpCLNSDomain
3	snmpCONSDomain
4	snmpDDPDomain
5	snmpIPXDomain

<mAddr>
identifies the transport address to be used when the agent sends traps to the manager, encoded as 1+N sub-identifiers, where the first sub-identifier indicates the length of the address, and the remaining sub-identifiers correspond to one octet from that address ([3] defines the address format corresponding to each transport domain). If the manager doesn't desire traps, then this field is encoded as a single sub-identifier having the value zero.

<aAddr>
identifies the transport address that the agent listens to when the manager sends traffic, encoded as 1+N sub-identifiers, where the first sub-identifier indicates the length of the address, and the remaining sub-identifiers correspond to one octet from that address ([3] defines the address format corresponding to each transport domain).

<mMMS>
identifies the maximum message size which the manager can receive,

encoded as 1 sub-identifier in the range 484 to 65507.

<mTraps>
identifies whether the manager wishes to receive traps from the agent, encoded as a single sub-identifier:

value	meaning
0	no traps
1	send traps using without authentication or privacy
2	send traps using authentication
3	send traps using authentication and privacy

<mLinger>
identifies the manager's desire for minimum number of contiguous seconds of inactivity for all parties and access control entries created before they are destroyed by the agent, encoded as 1 sub-identifier in the range 1 to 2147483647.

4.2. Step 2: Agent Analyzes Request

The agent receives the get request from the manager and identifies it as a user-based maintenance function to create a session.

The agent examines the parameter encoded in the instance-identifier of the one (and only) variable-binding of the get operation, and if any are unacceptable, it generates a response to the get operation, containing a single octet value:

value	meaning
1	bad tDomain value
2	bad mAddr value
3	bad aAddr value
4	bad mMMS value
5	bad mTraps value
6	bad mLinger value

Otherwise, the agent retrieves the following information:

(1) the entry in the scmUserTable which corresponds to the user identified in the management communication;

(2) its 12-octet administratively-unique identifier, agentID [2]; and,

(3) the maximum message size which the agent can receive, aMMS.

The agent calculates actualLinger by taking the minimum of mLinger and the corresponding instance of scmUserLinger. If the value of actualLinger is 2147483647, then the agent sets creationType to nonVolatile, otherwise creationType is set to volatile.

The agent computes an integer, sessionID, such that there are no parties known to the agent whose name is any of:

```
scmAgentNoAuthPartyID.<agentID>.<sessionID>
scmManagerNoAuthPartyID.<agentID>.<sessionID>
scmAgentAuthPartyID.<agentID>.<sessionID>
scmManagerAuthPartyID.<agentID>.<sessionID>
scmAgentPrivPartyID.<agentID>.<sessionID>
scmManagerPrivPartyID.<agentID>.<sessionID>
```

where <agentID> identifies the agent's unique identifier, encoded as 12 sub-identifiers.

The agent generates an unpredictable 128-bit quantity, aPad. The agent computes aSecret, based on an algorithm which uses the pairing of the value of partyAuthPrivate for userMaintParty.<userID> and aPad -- consult the definition of scmUserAuthProtocol in Section 6.

If the value of scmUserPrivProtocol is any value other than noPriv, the agent generates a second unpredictable 128-bit quantity, pPad, and the agent computes pSecret, based on an algorithm which uses the pairing of the value of partyAuthPrivate for userMaintParty.<userID> and pPad -- consult the definition of scmUserAuthProtocol in Section 6.

4.3. Step 3: Agent Creates Parties

The agent creates four parties named as:

```
scmAgentNoAuthPartyID.<agentID>.<sessionID>
scmManagerNoAuthPartyID.<agentID>.<sessionID>
scmAgentAuthPartyID.<agentID>.<sessionID>
scmManagerAuthPartyID.<agentID>.<sessionID>
```

where:

<agentID>
identifies the agent's agentID [2], encoded as 12 sub-identifiers.

<sessionID>
identifies this session's sessionID, encoded as 1 sub-identifier.

The parties are created with these values:

party	Agent noAuth	Manager noAuth	Agent Auth	Manager Auth
TDomain	tDomain	tDomain	tDomain	tDomain
TAddress	aAddr	mAddr	aAddr	mAddr
MaxMessageSize	aMMS	mMMS	aMMS	mMMS
Local	true	false	true	false
AuthProtocol	noAuth	noAuth	scmUserAuth	scmUserAuth
AuthClock	0	0	0	0
AuthPrivate	''H	''H	aSecret	aSecret
AuthPublic	''H	''H	''H	''H
PrivProtocol	noPriv	noPriv	noPriv	noPriv
PrivPrivate	''H	''H	''H	''H
PrivPublic	''H	''H	''H	''H
StorageType	creationType	creationType	creationType	creationType
Status	active	active	active	active

If the value of scmUserPrivProtocol is any value other than noPriv, then the agent also creates two more parties named as:

scmAgentPrivPartyID.<agentID>.<sessionID>
scmManagerPrivPartyID.<agentID>.<sessionID>

and having these values:

party	Agent Priv	Manager Priv
TDomain	tDomain	tDomain
TAddress	aAddr	mAddr
MaxMessageSize	aMMS	mMMS
Local	true	false
AuthProtocol	scmUserAuth	scmUserAuth
AuthClock	0	0
AuthPrivate	aSecret	aSecret
AuthPublic	''H	''H
PrivProtocol	scmUserPriv	scmUserPriv
PrivPrivate	pSecret	pSecret
PrivPublic	''H	''H
StorageType	creationType	creationType
Status	active	active

4.4. Step 4: Agent Authorizes Parties

For each entry in the scmCapTable whose value of scmCapIndex equals the value of scmUserCapIndex for the user identified in the management communication, the agent performs Step 4a and 4b.

4.4.1. Step 4a: Agent Checks Contexts

If the context named as:

scmContextID.<agentID>.<localTime>.<localEntity>

where:

<agentID>
identifies the agent's agentID, encoded as 12 sub-identifiers.

<localTime>
identifies the value of scmCapCtxLocalTime, encoded as 1 sub-identifier.

<localEntity>
identifies the value of scmCapCtxLocalEntity, encoded as N (possibly 0) sub-identifiers.

does not exist, then the agent creates it:

context	value
Local	true
View	1
LocalEntity	scmCapCtxLocalEntity
LocalTime	scmCapCtxLocalTime (expressed as the equivalent OBJECT IDENTIFIER)
ProxyDstParty	0.0
ProxySrcParty	0.0
ProxyContext	0.0
StorageType	creationType
Status	active

4.4.2. Step 4b: Agent Creates Access Control Entries

The agent creates 2 entries in the acTable:

ac	value for ACL #1	value for ACL #2
Target	Agent noAuth	Agent Auth
Subject	Manager noAuth	Manager Auth
Context	context from Step 4a	context from Step 4a
Privileges	scmCapNPrivileges	scmCapAPrivileges
ReadViewIndex	scmCapNReadView	scmCapAReadView
WriteViewIndex	scmCapNWriteView	scmCapAWriteView
StorageType	creationType	creationType
Status	active	active

If the value of mTraps is 1,
then 128 is added to the value of acPrivileges for ACL #1; otherwise, if
the value of mTraps is 2,
then 128 is added to the value of acPrivileges for ACL #2.

If the value of scmUserPrivProtocol is any value other than noPriv, then
the agent creates a third entry in the acTable:

ac	value for ACL #3
Target	Agent Priv
Subject	Manager Priv
Context	context from Step 4a
Privileges	scmCapPPrivileges
ReadViewIndex	scmCapPReadView
WriteViewIndex	scmCapPWriteView
StorageType	creationType
Status	active

If the value of mTraps is 3,
then 128 is added to the value of acPrivileges for ACL #3.

When an agent already has many activated user sessions, it is
undesirable for the creation of a new session to be denied due to the
inability of the agent to create the additional parties or access
control entries.
As such, if an agent having many active user sessions is unable to
perform Steps 3 or 4 due to lack of party-related resources, the agent
should begin destroying sessions,
in the order least recently used, until sufficient party-related

resources exist to perform Steps 3 and 4.

4.5. Step 5: Agent Responds

The agent generates a response to the get operation, an octet string having this value:

<result>
a single octet, containing the value 0.

<agentID>
12 octets, containing the agent's 12-octet administratively-unique identifier.

<sessionID>
4 octets, encoded as an unsigned integer using network-byte ordering (big-endian encoding).

<agentMMS>
2 octets, encoded as an unsigned integer using network-byte ordering (big-endian encoding).

<actualLinger>
4 octets, encoded as an unsigned integer using network-byte ordering (big-endian encoding).

<aPad>
16 octets.

<pPad>
16 octets.

If the value of scmUserPrivProtocol is noPriv, then no pPad value is sent (the aPad value completes the response).

4.6. Step 6: Agent Starts Initial Inactivity Timer

Upon sending the response to the get operation, the agent starts a 5 minute timer. If any of the session's 2 or 4 authenticated parties are used before the timer expires, then the timer is cancelled. Otherwise, if the timer expires before their use, then all 4 or 6 of the session's parties and their associated access control entries are immediately deleted.

By use of this timeout, a created session for which the agent-generated response is lost, is deleted after after 5 minutes of non-use.

4.7. Step 7: Manager Analyzes Response +

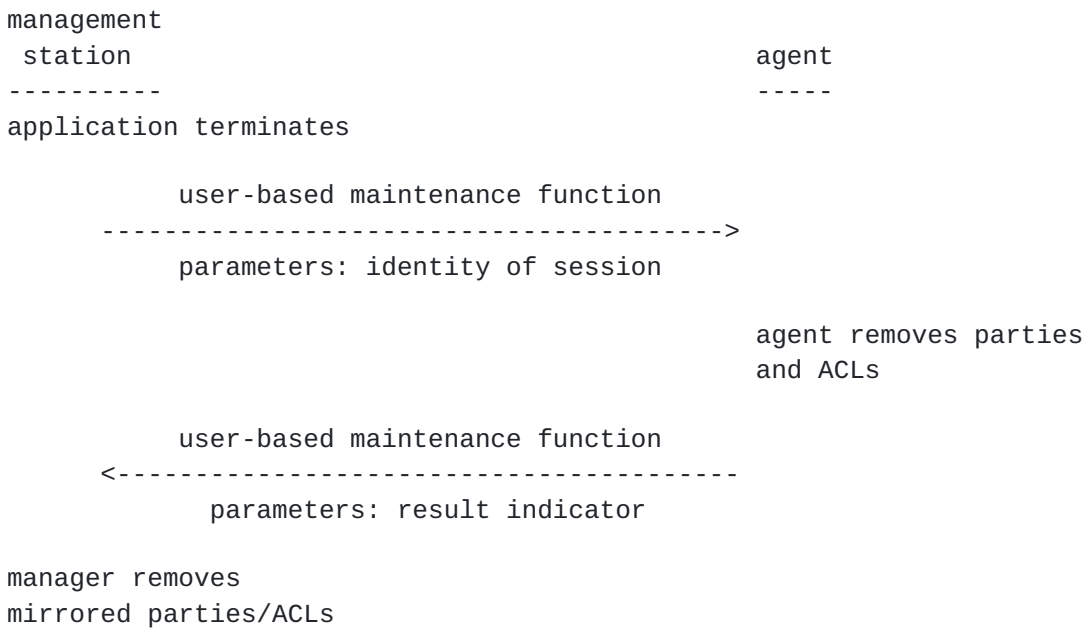
The manager receives the response from the agent and correlates to its earlier request. It then creates mirrors of the parties and access control entries described in Steps 3 and 4b, except that the values of partyLocal are inverted.

The manager should then issue an authenticated request which uses the + created session. This usage serves to confirm that the session has been + successfully created, and to cancel the agent's initial inactivity (5- + minute) timer. +

5. Session Destruction Algorithm

Sessions are destroyed when the management station issues a user-based maintenance function, which identifies a user configured at the agent.

Pictorially:



5.1. Step 1: Manager Requests Destruction

The manager performs a user-based maintenance function consisting of a set operation for

userMaintDestroy.0

setting it to an octet string having this value:

<agentID><sessionID>

where:

<agentID>
12 octets, containing the agent's 12-octet administratively-unique identifier.

<sessionID>
4 octets, encoded as an unsigned integer using network-byte ordering (big-endian encoding).

5.2. Step 2: Agent Analyzes Request and Responds

The agent receives the set request from the manager and identifies it as a user-based maintenance function to destroy a session.

The agent locates those parties whose name is any of:

scmAgentNoAuthPartyID.<agentID>.<sessionID>
scmManagerNoAuthPartyID.<agentID>.<sessionID>
scmAgentAuthPartyID.<agentID>.<sessionID>
scmManagerAuthPartyID.<agentID>.<sessionID>
scmAgentPrivPartyID.<agentID>.<sessionID>
scmManagerPrivPartyID.<agentID>.<sessionID>

If no such parties exist, or if the partyStorageType of any such party isn't volatile, or if the parties weren't created by the user corresponding to this user-based management function, then the agent generates an inconsistentValue response. Otherwise, the agent generates a noError response to the set operation, and deletes all parties and associated access control entries.

5.3. Step 3: Manager Analyzes Response

The manager receives the response from the agent and correlates to its earlier request. It then destroys the mirrors of the parties and access control entries that it created earlier.

6. Definitions

```
SNMPv2-SCM-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, snmpModules
        FROM SNMPv2-SMI
    DisplayString, RowStatus
        FROM SNMPv2-TC
    AccessPrivileges, StorageType, v2md5AuthProtocol, noPriv              |
        FROM SNMPv2-PARTY-MIB;

scmMIB MODULE-IDENTITY
    LAST-UPDATED "9503180000Z"                                            |
    ORGANIZATION "IETF SNMPv2 Working Group"                              |
    CONTACT-INFO
            "        Keith McCloghrie

             Postal: Cisco Systems, Inc.
                     170 West Tasman Drive,
                     San Jose, CA 95134-1706
                     US

                Tel: +1 408 526 5260

             E-mail: kzm@cisco.com"
    DESCRIPTION
            "The MIB module for the Simplified Configuration Model."
    ::= { snmpModules 4 }                                                 |
```

```
-- administrative assignments

scmAdmin        OBJECT IDENTIFIER ::= { scmMIB 1 }

-- parties under these subtrees are created dynamically by the agent
scmPartyID      OBJECT IDENTIFIER ::= { scmAdmin 1 }

scmAgentNoAuthPartyID
                OBJECT IDENTIFIER ::= { scmPartyID 1 }
scmManagerNoAuthPartyID
                OBJECT IDENTIFIER ::= { scmPartyID 2 }
scmAgentAuthPartyID
                OBJECT IDENTIFIER ::= { scmPartyID 3 }
scmManagerAuthPartyID
                OBJECT IDENTIFIER ::= { scmPartyID 4 }
scmAgentPrivPartyID
                OBJECT IDENTIFIER ::= { scmPartyID 5 }
scmManagerPrivPartyID
                OBJECT IDENTIFIER ::= { scmPartyID 6 }


-- context under this subtree might be created by the agent,
-- but normally exist

scmContextID    OBJECT IDENTIFIER ::= { scmAdmin 2 }


-- these are employed by user-based maintenance functions

userMaintParty  OBJECT IDENTIFIER ::= { scmAdmin 3 }

userMaintContext
                OBJECT IDENTIFIER ::= { 0 1 }
```

```
-- object assignments

scmMIBObjects   OBJECT IDENTIFIER ::= { scmMIB 2 }


-- user table

scmUserTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF ScmUserEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "The user table."
    ::= { scmMIBObjects 1 }

scmUserEntry OBJECT-TYPE
    SYNTAX      ScmUserEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "A conceptual row in the user table."
    INDEX       { IMPLIED scmUserID }
    ::= { scmUserTable 1 }

ScmUserEntry ::=
    SEQUENCE {
        scmUserID               DisplayString,
        scmUserPassword         OCTET STRING,
        scmUserAuthProtocol     OBJECT IDENTIFIER,
        scmUserPrivProtocol     OBJECT IDENTIFIER,
        scmUserCapIndex         INTEGER,
        scmUserLinger           INTEGER,
        scmUserStorageType      StorageType,
        scmUserStatus           RowStatus
    }
```

```
scmUserID OBJECT-TYPE
    SYNTAX      DisplayString (SIZE (1..64))
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "The identity of the user corresponding to this conceptual
            row."
    ::= { scmUserEntry 1 }

scmUserPassword OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (8..128))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "The user's password.  On retrieval, this object has the
            value of 8 zero-valued octets."
    ::= { scmUserEntry 2 }
```

```
scmUserAuthProtocol OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "The authentication protocol for this user.  This object may
            never take the value noAuth.

            Once an instance of this object is created, its value can
            not be changed.

            If the value of this object is v2md5AuthProtocol, then two
            algorithms are defined:

              To map scmUserPassword to an authentication key for the
              corresponding userMaintParty: form a string of length
              1,048,576 octets by repeating the value of scmUserPassword
              as often as necessary, truncating accordingly, and use the
              resulting string as the input to the MD5 algorithm.  The
              resulting digest is the authentication key for
              userMaintParty.<userID>.

              To map the pairing of a user's authentication key and the
              aPad quantity to the authentication key for a newly
              created party: append aPad to the value of
              partyAuthPrivate for userMaintParty.<userID>, and use the
              resulting string as the input to the MD5 algorithm.  The
              resulting digest is the authentication key for the newly
              created party."
    DEFVAL      { v2md5AuthProtocol }
    ::= { scmUserEntry 3 }
```

```
scmUserPrivProtocol OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "The privacy protocol for this user.  The value noPriv
            signifies that messages received by the party are not
            protected.

            Once an instance of this object is created, its value can
            not be changed.


            If the value of this object is desPrivProtocol, then one
            algorithm is defined:

              To map the pairing of a user's authentication key and the
              pPad quantity to the privacy key for a newly created
              party: append pPad to the value of partyAuthPrivate for
              userMaintParty.<userID>, and use the resulting string as
              the input to the MD5 algorithm.  The resulting digest is
              the privacy key for the newly created party."
    DEFVAL      { noPriv }
    ::= { scmUserEntry 4 }

scmUserCapIndex OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "The value of an instance of this object identifies one or
            more conceptual rows in the scmCapTable, and has the same
            value as the instance of scmCapIndex for those conceptual
            rows."
    ::= { scmUserEntry 5 }
```

```
scmUserLinger OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "The upper bound on the minimum number of contiguous seconds
            that a dynamically created party may reside in the agent and
            neither generate nor receive management communications,
            before the agent may choose to set the party's status to
            `destroy(6)'."
    ::= { scmUserEntry 7 }

scmUserStorageType OBJECT-TYPE
    SYNTAX      StorageType
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "The storage type for this conceptual row in the
            scmUserTable."
    ::= { scmUserEntry 8 }

scmUserStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "The status of this conceptual row in the scmUserTable.  If
            set to `destroy(6)', then any parties (and associated access
            control entries) having a storage type of `volatile(2)'
            which were earlier created for this user have their status
            set to `destroy(6)'."
    ::= { scmUserEntry 9 }
```

```
-- capabilities table

scmCapNextIndex OBJECT-TYPE
    SYNTAX      INTEGER (0..2147483647)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "The next unassigned value of scmCapIndex.  The value 0
            indicates that no unassigned values are available.

            Reading a non-zero value causes the assignment of the
            retrieved value for use as the scmCapIndex of a future
            capability, and thus causes the value of this object to
            change.                                                  -

            The algorithm for changing scmCapIndex is implementation-  +
            dependent, and the agent may use a subset of values within  +
            1..2147483647, but the agent must guarantee that the value  +
            held by this object is not assigned to any in-use value of  +
            scmCapIndex, e.g., is not pointed to by any other MIB       +
            object.                                                     +

            A management station should create a new MIB view using this
            algorithm: first, issue a management protocol retrieval
            operation to obtain the value of scmCapNextIndex -- this
            value is used as the scmCapIndex of the new capability; and,
            second, issue a management protocol set operation to create
            an instance of the scmCapStatus object setting its value to
            `createAndGo' or `createAndWait' (as specified in the
            description of the RowStatus textual convention)."
    ::= { scmMIBObjects 2 }

scmCapTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF ScmCapEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "The capabilities table."
    ::= { scmMIBObjects 3 }
```

```
scmCapEntry OBJECT-TYPE
    SYNTAX      ScmCapEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "A conceptual row in the capabilities table."
    INDEX       { scmCapIndex, scmCapCtxLocalTime,
                  IMPLIED scmCapCtxLocalEntity }
    ::= { scmCapTable 1 }

ScmCapEntry ::=
    SEQUENCE {
        scmCapIndex             INTEGER,
        scmCapCtxLocalTime      INTEGER,
        scmCapCtxLocalEntity    OCTET STRING,
        scmCapNPrivileges       AccessPrivileges,
        scmCapNReadView         INTEGER,
        scmCapNWriteView        INTEGER,
        scmCapAPrivileges       AccessPrivileges,
        scmCapAReadView         INTEGER,
        scmCapAWriteView        INTEGER,
        scmCapPPrivileges       AccessPrivileges,
        scmCapPReadView         INTEGER,
        scmCapPWriteView        INTEGER,
        scmCapStorageType       StorageType,
        scmCapStatus            RowStatus
    }

scmCapIndex OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "A unique value for each capability.  The value for each
            capability must remain constant at least from one re-
            initialization of the entity's network management system to
            the next re-initialization."
    ::= { scmCapEntry 1 }
```

```
scmCapCtxLocalTime OBJECT-TYPE
    SYNTAX      INTEGER { currentTime(1), restartTime(2) }
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "The temporal context associated with this capability."
    ::= { scmCapEntry 2 }

scmCapCtxLocalEntity OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (0..255))
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "The local entity associated with this capability."
    ::= { scmCapEntry 3 }

scmCapNPrivileges OBJECT-TYPE
    SYNTAX      AccessPrivileges
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "The access privileges which govern the flow of management
            information between the user and the agent when
            communicating using unauthenticated traffic."
    ::= { scmCapEntry 4 }

scmCapNReadView OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "A reference to a MIB view of a locally accessible entity,
            when the user requests the get, get-next, or get-bulk
            operations using unauthenticated traffic; the value of the
            instance identifies the particular MIB view which has the
            same value of viewIndex."
    ::= { scmCapEntry 5 }
```

```
scmCapNWriteView OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "A reference to a MIB view of a locally accessible entity,
            when the user requests the set operation using
            unauthenticated traffic; the value of the instance
            identifies the particular MIB view which has the same value
            of viewIndex."
    ::= { scmCapEntry 6 }

scmCapAPrivileges OBJECT-TYPE
    SYNTAX      AccessPrivileges
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "The access privileges which govern the flow of management
            information between the user and the agent when
            communicating using authenticated, but not private,
            traffic."
    ::= { scmCapEntry 7 }

scmCapAReadView OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "A reference to a MIB view of a locally accessible entity,
            when the user requests the get, get-next, or get-bulk
            operations using authenticated, but not private, traffic;
            the value of the instance identifies the particular MIB view
            which has the same value of viewIndex."
    ::= { scmCapEntry 8 }
```

```
scmCapAWriteView OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "A reference to a MIB view of a locally accessible entity,
            when the user requests the set operation using
            authenticated, but not private, traffic; the value of the
            instance identifies the particular MIB view which has the
            same value of viewIndex."
    ::= { scmCapEntry 9 }

scmCapPPrivileges OBJECT-TYPE
    SYNTAX      AccessPrivileges
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "The access privileges which govern the flow of management
            information between the user and the agent when
            communicating using authenticated and private traffic."
    ::= { scmCapEntry 10 }

scmCapPReadView OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "A reference to a MIB view of a locally accessible entity,
            when the user requests the get, get-next, or get-bulk
            operations using authenticated and private traffic; the
            value of the instance identifies the particular MIB view
            which has the same value of viewIndex."
    ::= { scmCapEntry 11 }
```

```
scmCapPWriteView OBJECT-TYPE
    SYNTAX      INTEGER (1..2147483647)
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "A reference to a MIB view of a locally accessible entity,
            when the user requests the set operation using authenticated
            and private traffic; the value of the instance identifies
            the particular MIB view which has the same value of
            viewIndex."
    ::= { scmCapEntry 12 }

scmCapStorageType OBJECT-TYPE
    SYNTAX      StorageType
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "The storage type for this conceptual row in the
            scmCapTable."
    ::= { scmCapEntry 13 }

scmCapStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
            "The status of this conceptual row in the scmCapTable."
    ::= { scmCapEntry 14 }
```

```
-- maintenance assignments

-- these objects are accessible only to user-based maintenance
-- functions

userMaintActions
                OBJECT IDENTIFIER ::= { scmMIB 3 }


userMaintTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF UserMaintEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "A pseudo-table provided to allow indexing for
            userMaintCreate."
    ::= { userMaintActions 1 }

userMaintEntry OBJECT-TYPE
    SYNTAX      UserMaintEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "Entries in this table are created by the agent dynamically
            when processing a getRequest operation, and are deleted       |
            immediately thereafter.                                       |
            As such, entries are not accessible via other retrieval
            operations."
    INDEX       { IMPLIED userMaintIndex }                                |
    ::= { userMaintTable 1 }

UserMaintEntry ::=
    SEQUENCE {
        userMaintIndex          OCTET STRING,
        userMaintCreate         OCTET STRING
    }
```

```
userMaintIndex OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
            "A pseudo-index provided to allow indexing for                |
            userMaintCreate.  Its value is the BER-encoding of the set    |
            of OBJECT IDENTIFIER sub-identifiers which a manager appends  |
            to userMaintCreate in order to supply the parameters for the  |
            session creation algorithm."                                  |
    ::= { userMaintEntry 1 }

userMaintCreate OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (1|39|55))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "A getRequest operation on this object is used to invoke the  |
            session creation algorithm."                                  |
    ::= { userMaintEntry 2 }


userMaintDestroy OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (16))
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
            "A set operation on this object is used to invoke the
            session destruction algorithm.  On retrieval, this object
            has the value of 16 zero-valued octets."
    ::= { userMaintActions 2 }
```

```
-- conformance information

scmMIBConformance
                OBJECT IDENTIFIER ::= { scmMIB 4 }

scmMIBCompliances
                OBJECT IDENTIFIER ::= { scmMIBConformance 1 }
scmMIBGroups
                OBJECT IDENTIFIER ::= { scmMIBConformance 2 }

-- compliance statements

scmMIBCompliance MODULE-COMPLIANCE
    STATUS      current
    DESCRIPTION
            "The compliance statement for SNMPv2 entities which
            implement the Simplified Configuration Model."
    MODULE      -- this module
        MANDATORY-GROUPS { scmGroup, userMaintGroup }
    ::= { scmMIBCompliances 1 }
```

```
-- units of conformance

scmGroup OBJECT-GROUP
    OBJECTS { scmUserPassword,
              scmUserAuthProtocol, scmUserPrivProtocol,
              scmUserCapIndex, scmUserLinger,
              scmUserStorageType, scmUserStatus,
              scmCapNextIndex,
              scmCapNPrivileges, scmCapNReadView, scmCapNWriteView,
              scmCapAPrivileges, scmCapAReadView, scmCapAWriteView,
              scmCapPPrivileges, scmCapPReadView, scmCapPWriteView,
              scmCapStorageType, scmCapStatus
            }
    STATUS      current
    DESCRIPTION
            "A collection of objects providing support for the
            Simplified Configuration Model."
    ::= { scmMIBGroups 1 }

userMaintGroup OBJECT-GROUP
    OBJECTS { userMaintCreate, userMaintDestroy }
    STATUS      current
    DESCRIPTION
            "A collection of objects providing support for user-based
            maintenance functions."
    ::= { scmMIBGroups 2 }


END
```

7. Appendix A: Password to Key Algorithm

The following code fragment demonstrates the password to key algorithm used when mapping scmUserPassword to an authentication key for a userMaintParty when the value of scmUserAuthProtocol is v2md5AuthProtocol:

```
void v2md5auth_password_to_key(password, passwordlen, key)
    u_char *password;       /* IN */
    u_int   passwordlen;    /* IN */
    u_char *key;            /* OUT - caller supplies pointer to 16
                               octet buffer */ {
    MDstruct    MD;
    u_char      *cp, password_buf[64];
    u_long      password_index = 0;
    u_long      count = 0, i;


    MDbegin(&MD);   /* initialize MD5 */

    /* loop until we've done 1 Megabyte */
    while (count < 1048576) {
        cp = password_buf;
        for(i = 0; i < 64; i++){
            *cp++ = password[ password_index++ % passwordlen ];
            /*
             * Take the next byte of the password, wrapping to the
             * beginning of the password as necessary.
             */
        }

        MDupdate(&MD, password_buf, 64 * 8);
        /*
         * 1048576 is divisible by 64, so the last MDupdate will be
         * aligned as well.
         */
        count += 64;
    }
    MDupdate(&MD, password_buf, 0); /* tell MD5 we're done */
    copy_digest_to_buffer(&MD, key);
    return; }
```

8. Acknowledgements

The Simplified Configuration Model is based on the "Simplified Security Conventions" developed by Steve Waldbusser, and subsequently updated by the SNMPv2 Working Group.

The authors wish to acknowledge in particular the contributions of the following individuals

Dave Arneson (Cabletron),
Uri Blumenthal (IBM),
Doug Book (Chipcom),
Deirdre Kostik (Bellcore),
Dave Harrington (Cabletron),
Jeff Johnson (Cisco Systems),
Brian O'Keefe (Hewlett Packard),
Dave Perkins (Bay Networks),
Randy Presuhn (Peer Networks),
Shawn Routhier (Epilogue),
Bob Stewart (Cisco Systems),
Kaj Tesink (Bellcore).

9. References

[1] Case, J., Galvin, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2)", in progress, SNMP Research, Inc., Trusted Information Systems, Cisco Systems, Dover Beach Consulting, Inc., Carnegie Mellon University, March, 1995.

[2] Case, J., Galvin, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Party MIB for version 2 of the Simple Network Management Protocol (SNMPv2)", in progress, SNMP Research, Inc., Trusted Information Systems, Cisco Systems, Dover Beach Consulting, Inc., Carnegie Mellon University, March, 1995.

[3] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Transport Mappings for version 2 of the Simple Network Management Protocol (SNMPv2)", in progress, SNMP Research Inc., Cisco Systems, Inc., Dover Beach Consulting, Inc., Carnegie Mellon University, March, 1995.

10. Security Considerations

The mechanisms defined in this document allow "users" to be configured, and to activate management sessions for them. How "users" are defined is subject to the security policy of the network administration. For example, users could be individuals (e.g., "joe" or "jane"), or a particular role (e.g., "operator" or "administrator"), or a combination (e.g., "joe-operator", "jane-operator" or "joe-admin"). Furthermore, a "user" may be a logical entity, such as a manager station application or set of manager station applications, acting on behalf of a individual or role, or set of individuals, or set of roles, including combinations. The mechanisms also allow management capabilities to be defined, where one or more users can be authorized for a set of capabilities.

A password is defined for each user, and these passwords will often be generated, remembered, and input by a human. Because human-generated passwords may be less than the 16 octets required by the MD5 authentication protocols, and because brute force attacks can be quite easy on a relatively short ASCII character set, passwords are not used directly, but are instead mapped by the algorithm described in Section 6 and Appendix A. Agent implementations (and agent configuration applications) must ensure that passwords are at least 8 characters in length.

Because these passwords are used (nearly) directly, it is very important that they not be easily guessed. It is suggested that they be composed of mixed-case alphanumeric and punctuation characters that don't form words or phrases that might be found in a dictionary. Longer passwords improve the security of the system. Users may wish to input multiword phrases to make their password string longer while ensuring that it is memorable.

Note that there is security risk in configuring the same "user" on multiple systems where the same password is used on each system, since the compromise of that user's secrets on one system results in the compromise of that user on all other systems having the same password. There is also greater security risk and less accountability in allowing multiple humans to know the password for a given "user".

Note also that the userMaintParty authentication key for a user is the same for all systems on which the user has the same password, and it is necessary to store that authentication key on each such system. As such, an implementation must, to the maximal extent possible, prohibit read-access to these authentication keys under all circumstances except as required to generate and/or validate SNMPv2 messages containing

user-based maintenance functions. |

With respect to the replay-ability of user-based maintenance functions, note that all such operations are effectively idempotent: replaying a request to create a session results in a new session being created, but the session has a new unique set of keys, which can be derived only by an authorized user; similarly, replaying a request to destroy a session results in an inconsistentValue error.

11. Authors' Address

Steven Waldbusser
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
US

Phone: +1 412 268 6628
Email: waldbusser@cmu.edu

Jeffrey D. Case
SNMP Research, Inc.
3001 Kimberlin Heights Rd.
Knoxville, TN 37920-9716
US

Phone: +1 615 573 1434
Email: case@snmp.com

Keith McCloghrie
Cisco Systems, Inc.
170 West Tasman Drive,
San Jose, CA 95134-1706

Phone: +1 408 526 5260
EMail: kzm@cisco.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

Table of Contents