Simple Network Management Protocol (SNMP) EngineID Discovery
draft-schoenw-snmp-discover-01

Jürgen Schönwälder

Jacobs University Bremen
Bremen, Germany

68. IETF March 2007
Applications need to know the contextEngineID in order to access information

Implementations typically use USM’s securityEngineID as a “best guess” for the contextEngineID

TSM does not need a securityEngineID and hence TSM lacks a mechanism to “best guess” the contextEngineID

Since many applications rely on contextEngineID discovery (i.e., they do not maintain a data store with discovered or configured engineIDs), we need to provide a mechanism to discover appropriate engineIDs
Applications need to know the contextEngineID in order to access information.

Implementations typically use USM’s securityEngineID as a “best guess” for the contextEngineID.

TSM does not need a securityEngineID and hence TSM lacks a mechanism to “best guess” the contextEngineID.

Since many applications rely on contextEngineID discovery (i.e., they do not maintain a data store with discovered or configured engineIDs), we need to provide a mechanism to discover appropriate engineIDs.
Problem: Context EngineID Discovery

- Applications need to know the contextEngineID in order to access information.
- Implementations typically use USM’s securityEngineID as a “best guess” for the contextEngineID.
- TSM does not need a securityEngineID and hence TSM lacks a mechanism to “best guess” the contextEngineID.
- Since many applications rely on contextEngineID discovery (i.e., they do not maintain a data store with discovered or configured engineIDs), we need to provide a mechanism to discover appropriate engineIDs.
Applications need to know the contextEngineID in order to access information.

Implementations typically use USM’s securityEngineID as a “best guess” for the contextEngineID.

TSM does not need a securityEngineID and hence TSM lacks a mechanism to “best guess” the contextEngineID.

Since many applications rely on contextEngineID discovery (i.e., they do not maintain a data store with discovered or configured engineIDs), we need to provide a mechanism to discover appropriate engineIDs.
Proposal: Introduce well-known localEngineID

- Introduce a well-known “localEngineID” which can be used to refer to the local engine of an “agent”
- In terms of the SNMP architecture, SNMP applications register themself twice under both the real engineID and the well-known “localEngineID” (see the registerContextEngineID() ASI)
- Applications can use the “localEngineID” to retrieve data local to the remote engine (and in particular the snmpEngineID.0 scalar)
- Essentially, the discovery problem is solved by avoiding it through the introduction of a well-known constant
- Proposal covers all the non-proxy cases (which is believed to be the large majority) and is USM compatible
Proposal: Introduce well-known localEngineID

- Introduce a well-known “localEngineID” which can be used to refer to the local engine of an “agent”
- In terms of the SNMP architecture, SNMP applications register themself twice under both the real engineID and the well-known “localEngineID” (see the registerContextEngineID() ASI)
- Applications can use the “localEngineID” to retrieve data local to the remote engine (and in particular the snmpEngineID.0 scalar)
- Essentially, the discovery problem is solved by avoiding it through the introduction of a well-known constant
- Proposal covers all the non-proxy cases (which is believed to be the large majority) and is USM compatible
Proposal: Introduce well-known localEngineID

- Introduce a well-known “localEngineID” which can be used to refer to the local engine of an “agent”
- In terms of the SNMP architecture, SNMP applications register themself twice under both the real engineID and the well-known “localEngineID” (see the registerContextEngineID() ASI)
- Applications can use the “localEngineID” to retrieve data local to the remote engine (and in particular the snmpEngineID.0 scalar)
- Essentially, the discovery problem is solved by avoiding it through the introduction of a well-known constant
- Proposal covers all the non-proxy cases (which is believed to be the large majority) and is USM compatible
Proposal: Introduce well-known localEngineID

- Introduce a well-known “localEngineID” which can be used to refer to the local engine of an “agent”
- In terms of the SNMP architecture, SNMP applications register themself twice under both the real engineID and the well-known “localEngineID” (see the registerContextEngineID() ASI)
- Applications can use the “localEngineID” to retrieve data local to the remote engine (and in particular the snmpEngineID.0 scalar)
- Essentially, the discovery problem is solved by avoiding it through the introduction of a well-known constant
- Proposal covers all the non-proxy cases (which is believed to be the large majority) and is USM compatible
Need to allocate a value which is consistent with the SnmpEngineID textual convention (RFC3411)

Proposal: Use the variable length format 3) together with the unallocated format value 6 and the enterprise ID 0:

\[
\text{localEngineID OCTET STRING ::= '8000000006'H}
\]

There are no documented rules how to allocate something in the SnmpEngineID number space:

- So what is the procedure to allocate a constant?
- Is '8000000006'H the right value to choose?