Tutorial de NET-SNMP

INDICE

Documento extraído del artículo:
- “Tutorial de NET-SNMP” por Jorge Moreno Carreres

1. Sobre este tutorial

Este tutorial pretende ser una guía rápida de configuración para los demonios incluidos en el paquete de gestión de red NET-SNMP (**snmpd** y **snmptrapd**). No es una guía de utilidades para línea de comandos, ni tampoco pretende entrar en aspectos demasiado teóricos sobre el funcionamiento interno del protocolo de gestión de red SNMP. Solamente es un documento que puede servir tanto para configurar los aspectos más básicos como los más avanzados del agente (**snmpd)** y el demonio receptor de notificaciones (**snmptrapd**).

El escenario de las pruebas ha sido una pequeña red local, constituida por los siguientes hosts:

- Un servidor que actúa como pasarela de red, ejecutando una distribución Debian, cuyo nombre en la red local es **averia**. Este es el host en el que se ejecuta el agente SNMP.

- Un cliente de red ejecutando una distribución Debian, cuyo nombre en la red es **truca**. Aquí se ejecuta el demonio receptor de las notificaciones que manda **averia**.

2. ¿Qué es SNMP?

SNMP son las siglas de *Simple Network Management Protocol* y es un protocolo muy extendido que trabaja en el nivel de aplicación. Mediante SNMP podemos realizar tareas de monitorización de red, configuración de dispositivos, detección de problemas y otras muchas cosas más.

Para gestionar un dispositivo mediante SNMP, éste debe tener un agente que gestione la información contenida en la MIB-II (*Management Information Base*). El protocolo funciona mediante un mecanismo de petición-respuesta, es decir, nosotros mandamos una petición u orden al agente que gestiona un dispositivo concreto y éste nos responde con la información requerida o el resultado de la orden, o con un mensaje de error si nuestra petición fue incorrecta.

Actualmente existen tres versiones de este protocolo: SNMPv1, SNMPv2 y SNMPv3; cada una con sus propias características y limitaciones. Nuestro objetivo principal será abordar las diferencias entre estas tres versiones respecto a la autenticación de las peticiones y órdenes.

2.1. SNMPv1

Cuando se utiliza SNMPv1, el agente emplea un sencillo sistema de autenticación para determinar qué gestor puede acceder a qué variables de la MIB. Este esquema implica la especificación de unas determinadas políticas de acceso, relacionadas con los conceptos de **comunidad**, **modo de acceso** y **vista de la MIB**. Una **comunidad** es un conjunto de hosts relacionados mediante un nombre de comunidad, que debe ser incluido en la petición. El **modo de acceso** especifica los accesos permitidos para una determinada comunidad, que suelen ser **none**, **read-write**, **read-only** o **write-only**. Una **vista de la MIB** define uno o más subárboles de la MIB a los que una determinada comunidad puede acceder.

Cuando el agente recibe una petición, verifica el nombre de comunidad junto con la IP del host desde el que se hizo la petición para determinar que el host realmente pertenezca a esa comunidad. Si esta verificación devuelve un resultado positivo, entonces comprueba que la comunidad tenga acceso a las variables de la MIB solicitadas en la petición. Si esto es correcto, el agente responderá a la petición. Si no, devolverá el mensaje de error correspondiente al host peticionario.

En esta versión también se define la estructura básica que debe tener la MIB, incluyendo los **identificadores de objeto** (tanto numéricos como textuales) de los objetos y tablas de uso más común, como **ifTable**, **tcpConnTable**, **ipAddrTable**, etc. Además, se definen tres operaciones básicas para el acceso a la información de gestión contenida en la MIB: **Get**, **GetNext** y **Set**. Se define también la operación **Trap** para el envío de notificaciones.

2.2. SNMPv2

SNMPv2 es una extensión de SNMPv1. La abstracción de SNMPv2 que permite el uso de los mismos elementos descritos para SNMPv1, es decir, los conceptos de **comunidad**, **vista de la MIB** y **modo de acceso**, es llamada SNMPv2c (*Community-based SNMPv2*), y es la que el agente de NET-SNMP implementa.

El formato de la PDU (mensaje) es el mismo, solo que se usa un nuevo número de versión. En cuanto a las operaciones de acceso a la MIB, además de las definidas en SNMPv1, esta versión define dos nuevas: **GetBulk** e **Inform**. **GetBulk** se usa para obtener de forma eficiente grandes cantidades de datos, e **Inform** se usa para el envío de notificaciones con confirmación por parte del receptor.

En cuanto a la MIB, en esta versión se definen nuevos tipos de objetos para mejorar la semántica.

2.3. SNMPv3

SNMPv3 añade capacidades de seguridad y configuración remota a las versiones previas. Se introduce la arquitectura del **modelo de seguridad basado en usuarios** o USM (*User-based Security Model*), para añadir seguridad a los mensajes, y el **modelo de control de acceso basado en vistas** o VACM (*View-based Access Control Model*) para el control de acceso. También introduce la posibilidad de configurar remotamente el agente dando distintos valores a los objetos que representan la configuración del agente.

La arquitectura soporta el uso simultáneo de distintos modelos de control de acceso, seguridad y proceso de mensajes, entre los cuales hay que resaltar la **autenticación de usuarios mediante protocolos seguros** (como MD5 o SHA) y la **privacidad en la comunicación usando algoritmos de encriptación** como DES.

3. ¿Qué es NET-SNMP?

NET-SNMP es un conjunto de aplicaciones usado para implementar el protocolo SNMP usando IPv4 e IPv6. Incluye:

- Aplicaciones de línea de comandos para:
  - tomar información de dispositivos capaces de manejar el protocolo SNMP, ya sea usando peticiones simples (**snmpget**, **snmpgetnext**) o múltiples (**snmpwalk**, **snmptable**, **snmpdelta**).
  - manipular información sobre la configuración de dispositivos capaces de manejar SNMP (**snmpset**).
  - conseguir un conjunto de informaciones de un dispositivo con SNMP (**snmpdf**, **snmpnetstat**, **snmpstatus**).
  - traducir entre OIDs numéricos y textuales de los objetos de la MIB, y mostrar el contenido y estructura de la MIB (**snmptranslate**).
- Un navegador gráfico de la MIB (**tkmib**), usando Tk/perl.
- Un demonio para recibir notificaciones SNMP (**snmptrapd**). Las notificaciones seleccionadas pueden guardarse en un log (como **syslog** o un archivo de texto plano), ser reenviadas a otro sistema de gestión de SNMP, o ser pasadas a una aplicación externa.

- Un agente configurable para responder a peticiones SNMP para información de gestión (**snmpd**). Incluye soporte para un amplio rango de módulos de información de la MIB, y puede ser extendido usando módulos cargados dinámicamente, scripts externos y comandos.

- Una biblioteca para el desarrollo de nuevas aplicaciones SNMP, con APIs para C y Perl.

Si elegimos instalar el paquete correspondiente en un sistema basado en GNU/Linux, podemos instalar el existente para nuestra distribución. En las distribuciones basadas en Debian corresponde con los paquetes llamados: **snmp** y **snmpd**.

4. El agente SNMP (snmpd)

4.1. ¿Qué es?

snmpd es un agente SNMP que escucha en el puerto 161 de UDP (por defecto), esperando la llegada de peticiones desde algún software de gestión SNMP. Cuando recibe una petición, recopila la información y/o lleva a cabo las operaciones solicitadas, devolviendo la información correspondiente al emisor de la petición.

4.2. ¿Cómo funciona?

El agente **snmpd** se ejecuta como demonio, permaneciendo en segundo plano durante toda su ejecución. Una vez instalado el paquete NET-SNMP, el agente se ejecutará al iniciar nuestro host mediante el sistema de servicios estándar de GNU/Linux. Para iniciarlo, pararlo o recargarlo manualmente usaremos el comando:

 # /etc/init.d/snmpd {start|stop|restart|reload|forcereload}

usando una opción u otra dependiendo de lo que queramos hacer. En su arranque, buscará su archivo de configuración (/**etc**/**snmp**/**snmpd.conf**). Dicho archivo describe el comportamiento del agente durante la ejecución, aunque no es imprescindible para que el agente funcione y responda a peticiones, puesto que dispone de una configuración por defecto.

4.3. Configuración del agente

A pesar de que podemos arrancar el agente incluyendo varias opciones en la línea de comandos (ejecutar **snmpd --help** para una descripción más precisa), llevaremos a cabo la configuración mediante el fichero **snmpd.conf**, puesto que de esta forma las opciones que escojamos quedarán grabadas de forma persistente para posteriores ejecuciones.

4.3.1. Ficheros de configuración

Existe un fichero llamado **snmp.conf** (usualmente en el directorio /**etc**/**snmp**) que contiene la configuración referente al conjunto de programas que forman NET-SNMP. En él podemos especificar aspectos como la localización física de los archivos de la MIB, los puertos por defecto para los demonios **snmpd** y **snmptrapd**, y el directorio donde estos dos demonios guardarán los archivos persistentes que contienen sus datos (usualmente /**var**/**lib**/**snmp**).

El archivo **snmpd.conf** es específico para el agente, y contiene una serie de directivas que describirán su comportamiento en diversos ámbitos, y es el que vamos a construir poco a poco en este punto del tutorial. En los siguientes apartados pasa a explicarse cada uno de los ámbitos posibles de configuración y las directivas correspondientes que pueden usarse.

4.3.2. Especificando información sobre nuestro sistema

Algunas de las siguientes directivas modifican el valor de varios objetos de la MIB, y otras sólo sirven para la propia configuración del agente. Nótese que, en caso de darles un valor en el archivo de configuración del agente, se convertirán en objetos de sólo lectura (obviamente, aquellos que sean de lectura y escritura inicialmente) y, por tanto, no podremos cambiar sus valores mediante el comando **snmpset**.

- **syslocation CADENA**

  Cambia el valor del objeto **system.sysLocation** de la MIB, usado para especificar la localización física del host en el que corre el agente.

  En nuestro ejemplo, incluiremos la siguiente directiva:

  syslocation "Lab. de Práctias de Protocolos y Servicios"

- **syscontact CADENA**

  Especifica la información textual de contacto con la persona responsable del host (objeto **system.sysContact**). Podemos, por ejemplo, incluir lo siguiente en nuestro archivo de configuración:

  syscontact "Profesor de Prácticas de Protocolos y Servicios"

- **sysname CADENA**

  Especifica el nombre del host, usualmente el nombre de dominio completo. El objeto referido por esta directiva es **system.sysName**. Daremos un valor a este objeto con la siguiente directiva:

  sysname "averia.labpracts.dit.ulpgc.es"

- **sysservices NUMERO**

  El objeto **system.sysServices** indica el conjunto de niveles de la arquitectura OSI que el host puede soportar. El valor del objeto es una suma que inicialmente vale cero. Para cada capa L de la arquitectura OSI soportada por el host, sumamos a **sysServices** el valor 2 elevado a L-1. En caso de que el host no soporte OSI (el caso más general), tendremos sólo en cuenta los niveles 1 (físico), 2 (enlace), 3 (red), 4 (transporte) y 7 (aplicación). Para un host, el valor recomendado es 72 (es decir, niveles de transporte y aplicación). Vamos, pues, a incluir este valor en nuestro archivo de configuración:

  sysservices 72

- **sysdescr CADENA**

  Especifica una descripción textual del host (**system.sysDescr**).

  Convencionalmente contiene su nombre completo, una descripción del hardware, del sistema operativo y del software de red.

  La descripción de nuestro host será la siguiente:

  sysdescr " averia.labpracts.dit.ulpgc.es Linux Ubuntu Pentium IV@3200_MHz 1024_MB_RAM"

- **sysobjectid OID**

  El valor de **system.sysObjectID** debe contener una identificación del subsistema de gestión de red que contiene el dispositivo, proporcionada por el fabricante. Provee una forma fácil y no ambigua de saber qué tipo de dispositivo estamos gestionando. Este valor está ubicado dentro del subárbol **enterprises** de la MIB (**1.3.6.1.4.1**). Por ejemplo, si el fabricante "PCs Inc." tiene asignado el subárbol 1.3.6.1.4.1.666, podría asignar el OID 1.3.6.1.4.1.666.1.1 a su router "DeLab 6780".

  En nuestro caso lo dejaremos tal cual está, puesto que en este punto no tenemos ningún subárbol asignado dentro de **enterprises**.

- **agentaddress [<especificadorDeTransporte>:]<direcciónDeTransporte>[,...]**

  Por defecto, **snmpd** escucha en el puerto 161 de UDP de todas las interfaces de red. Mediante esta directiva podemos modificar este comportamiento.

  Por ejemplo, si quisiéramos que el agente escuchara en el puerto 161 de TCP y UDP para todas las interfaces, y en el puerto 9161 en la interfaz de loopback (lo), incluiríamos la siguiente directiva:

  agentaddress 161,tcp:161,localhost:9161

- **agentgroup IDgrupo**

  Esta directiva provoca que el agente cambie su identificador de grupo después de abrir el puerto en el que escucha. **IDgrupo** puede ser el nombre de un grupo o su identificador numérico.

  Si quisiéramos que el agente se ejecutase con el ID de grupo **jordi**, incluiríamos:

  agentgroup jordi

- **agentuser IDusuario**

  Funciona de forma análoga a la directiva anterior, solo que ésta se refiere al identificador de usuario.

  Si queremos que nuestro agente se ejecute con el ID del usuario **jordi**, incluiremos:

  agentuser jordi

- **interface NOMBRE TIPO VELOCIDAD**

  Mediante esta directiva podemos especificar la información referente a aquellas interfaces cuyos parámetros no sean bien detectados por el agente. Si, por ejemplo, disponemos de una interfaz eth0 de tipo Ethernet y con una velocidad de conexión de 100 Mb/s, y por cualquier razón el agente no la detecta correctamente, podemos darle a éste los parámetros de nuestra interfaz:

  interface eth0 6 10000000

- **ignoredisk CADENA**

  Cuando el agente escanea todos los dispositivos de disco disponibles, puede que se bloqueé, pudiendo producir un *timeout* cuando inspeccionemos la tabla de dispositivos mediante **snmpwalk**. Si esto ocurre, podemos usar esta directiva para forzar al agente a que no escaneé ciertos discos, pudiendo incluirla varias veces en el archivo de configuración. El formato de **CADENA** puede seguir la sintaxis del "*bourne shell*".

  Si nuestro disco /**dev**/**hda3** causa problemas al agente durante la detección, podemos obligarle a que no lo escanee:

  ignoredisk /dev/hda3

- **storageUseNFS NUMERO**

  Cuando **NUMERO**=1, esta directiva causa que todos los sistemas de archivos NFS sean marcados como "*Network Disks*" en la tabla **hrStorageTable** (**1.3.6.1.2.1.25.2.3**). Si la directiva no se usa, o **NUMERO**=2, aparecerán como "*Fixed Disks*".

  Por claridad en las consultas, usaremos esta directiva en nuestro archivo:

  storageUseNFS 1

- **override OID TIPO VALOR**

  Esta directiva permite sobreescribir un **OID** con un valor distinto, e incluso con un tipo distinto. Por ejemplo, podríamos sobreescribir el valor y el tipo de **system.sysDescr** mediante la siguiente orden:

  override sysDescr.0 octet_str "PC Lab. Protocolos y Servicios"

  Con esto le estaríamos cambiando el tipo a **octet_str** y su valor a la cadena encerrada entre comillas. Indirectamente, estamos haciendo también que sea modificable mediante **snmpset**, lo cual viola la especificación de la MIB.

4.3.3. Configuración de acceso para SNMPv1 y SNMPv2c

Usando las siguientes directivas podemos configurar los parámetros necesarios para controlar el acceso a nuestra MIB. Para SNMPv1 y SNMPv2c aplicaremos los conceptos de comunidad, vista y modo de acceso explicados en este tutorial. Este modo de acceso a la MIB es el llamado Modelo de Control de Acceso Basado en Vistas o VACM (*View-Based Access Control Model*). Primero, tenemos que definir **nombres de seguridad** y asociarlos con **nombres de comunidad** y con un conjunto de hosts que pertenecerán a dicha comunidad. Después, asociaremos estos nombres de seguridad con **nombres de grupos** para, finalmente, definir **vistas** y asociar cada una de ellas con un grupo de los definidos anteriormente.

- **com2sec NOMBRE FUENTE COMUNIDAD**

  Esta directiva especifica la asociación entre un nombre de seguridad, un host o conjunto de hosts (típicamente definido por un conjunto de direcciones IP) y un nombre de comunidad. **FUENTE** puede ser un nombre de host, la palabra **default** o una dirección de subred junto con los bits dedicados a la máscara de subred.

  Crearemos dos comunidades, una con acceso desde toda la red y otra con acceso sólo desde la red local:

  ```
  com2sec  public_sec     default        public
  com2sec  private_sec    192.168.0.0/24  private
  ```

- **com2sec6 NOMBRE FUENTE COMUNIDAD**

  Esta directiva es la versión de la anterior para direcciones IPv6. Sólo será válida si especificamos UDPIPv6 como dominio de transporte en la compilación.

- **group NOMBRE MODELO SEGURIDAD**

  Esta directiva mapea un nombre de seguridad con un nombre de grupo, asignando a dicho grupo un modelo de acceso que puede ser **v1**, **v2c** o **usm** (*User-based Security Model*, usado para SNMPv3).

  Vamos a crear dos grupos, asociados a los nombres de seguridad que hemos especificado en la directiva anterior. Cada grupo podrá acceder mediante peticiones SNMPv1 o SNMPv2:

  ```
  group   public_grp     v1       public_sec
  group   public_grp     v2c      public_sec
  group   private_grp    v1       private_sec
  group   private_grp    v2c      private_sec
  ```

- **access NOMBRE CONTEXTO MODELO NIVEL PREFIJO LEER ESCRIBIR NOTIFICAR**

  Define los accesos posibles de los grupos definidos anteriormente a las vistas que definiremos con la directiva **view**.

  - **NOMBRE** será el nombre del grupo.
  - **CONTEXTO** representa un nombre de contexto o grupo de nombres de contexto. Para SNMPv1 y SNMPv2c su valor será la cadena vacía.
  - **MODELO** puede contener los mismos valores que en la directiva anterior junto con la palabra **any** (cualquiera).
  - **NIVEL** especifica el nivel de autenticación necesario para este acceso, y puede ser **noauth**, **auth** o **priv**. Para SNMPv1 y SNMPv2c debe valer **noauth**. Se entrará en detalle en el apartado referido a SNMPv3.
  - **PREFIJO** especifica la coincidencia que debe haber entre **CONTEXTO** y el contexto de la PDU (mensaje) entrante. Puede valer **exact** o **prefix**.
  - **LEER** especifica la vista sobre la que se pueden realizar operaciones de lectura en este acceso.
  - **ESCRIBIR** indica la vista de los objetos que podemos escribir mediante este acceso.
  - **NOTIFICAR** define la vista sobre cuyos objetos podremos lanzar notificaciones.

Definiremos los siguientes accesos para los grupos creados anteriormente: el grupo **public** tendrá acceso de sólo lectura a toda la MIB y el grupo **private** tendrá acceso de lectura a toda la MIB, y de escritura a la vista parte, definida en la siguiente directiva.

```
access  public_grp  "" any noauth exact todo none  none
access  private_grp "" any noauth exact todo parte todo
```

– **view NOMBRE TIPO SUBARBOL [MASCARA]**

Esta directiva nos permite definir vistas de la MIB, que no son más que subconjuntos de ésta. Estos subconjuntos pueden estar formados por una o más ramas del árbol de la MIB, e incluso por la MIB entera. A menos que creemos vistas con una sola rama del árbol de la MIB, tendremos que usar varias de estas directivas para crear una nueva vista a nuestro gusto.

- **NOMBRE** será el nombre que queramos asignar a esta vista. Si la vista ya existe, esta directiva la modificará conforme a los parámetros siguientes.
- **TIPO** será la acción que queramos realizar sobre esta vista. Puede valer **included** (incluido) o **excluded** (excluido).
- **SUBARBOL** es el subárbol de la MIB que queramos incluir o excluir de esta vista, dependiendo del valor del parámetro anterior.
- **MASCARA** es una máscara de bits. Nos permite definir de forma sencilla el acceso a un conjunto de objetos en base al OID proporcionado en el campo **SUBARBOL**. Por ejemplo, en la directiva

```
view MiVista included 1.3.6.1.2.1.2.2.1.1.1 ff.a0
```

En este ejemplo usamos la máscara "ff.a0", cuyo equivalente en binario sería "11111111.10100000". El primer cero aparece en la máscara en la posición 10, lo cual permitirá acceder mediante esta vista a todos los objetos cuyo OID se corresponda con "1.3.6.1.2.1.2.2.1.x.2", siendo x un valor cualquiera.

Para nuestro ejemplo, crearemos dos vistas: Una vista **todo** que cubrirá toda la MIB y una vista **parte** que cubrirá toda la MIB, exceptuando el árbol de RMON.

```
view   todo   included   .1     00
view   parte  included   .1     00
view   parte  excluded   .1.3.6.1.2.1.16
```

El siguiente ejemplo genérico nos ilustra sobre cómo conjuntar el uso de los nombres de seguridad, de comunidad y de grupo:

```
#              nombre        fuente          comunidad
com2sec        nombre_sec    localhost       mi_comunidad

#       nombre      modelo   seguridad
group   mi_grupo    v1       nombre_sec

#       nombre  tipo          subarbol   mascara
view    vista   included      .1         80

#                   contexto  modelo nivel     prefijo  leer  escribir notificar
access  mi_grupo    ""        any    noauth    exact    vista vista    vista
```

Disponemos también de una serie de directivas más fáciles de usar, pero que son menos flexibles si pretendemos crear configuraciones personalizadas. Estas directivas son las siguientes:

- **rocommunity COMUNIDAD [FUENTE [OID]]**

  Crea una comunidad con permisos de lectura sobre el subárbol especificado por **OID**, cuyos miembros serán los hosts especificados por **FUENTE**.

  Crearemos una comunidad con acceso de sólo lectura sobre el árbol **interfaces** mediante la siguiente directiva:

  rocommunity cotilla 192.168.0.0/24 .1.3.6.1.2.1.2

- **rwcommunity COMUNIDAD [FUENTE [OID]]**

  Idéntica a la anterior, pero creando una comunidad con permisos de lectura y escritura. Para crear una comunidad con estos derechos de acceso sobre el grupo **system**, y con acceso solamente desde el **localhost**, usaríamos:

  rwcommunity hacker localhost .1.3.6.1.2.1.1

- **rocommunity6 COMUNIDAD [FUENTE [OID]]**
- **rwcommunity6 COMUNIDAD [FUENTE [OID]]**

  Estas dos directivas son las análogas a las dos anteriores para el dominio de transporte UDPIPv6.

4.3.4. Configuración de acceso para SNMPv3

Usando estas directivas podremos crear usuarios que puedan comunicarse con el agente mediante el protocolo SNMPv3 que, a diferencia de las dos anteriores versiones, soporta un modo de acceso basado en usuarios y contraseñas.

- **engineID CADENA**

  Con esta directiva daremos un identificador al agente, llamado **engineID**, que éste usará para poder responder a mensajes SNMPv3. Si no le damos ningún valor, se calculará en base a un número aleatorio junto con la hora actual en segundos. El valor de **CADENA** puede ser cualquier combinación de dígitos hexadecimales, como por ejemplo:

  engineID 001122334455AABBCCDD

- **rouser USUARIO [noauth|auth|priv [OID]]**

  Crea un usuario SNMPv3 con derechos de sólo lectura, especificando el nivel de autenticación requerido (ver la directiva **createUser** para una explicación) y el subárbol de la MIB al cual tendrá acceso. Si el nivel de acceso escogido es **auth** o **priv**, tendremos que asociar contraseñas a los usuarios creados mediante la directiva **createUser**.

Vamos a crear un usuario con acceso de sólo lectura a toda la MIB, sin necesidad de autenticación en el acceso:

rouser manolito noauth .1

NOTA: para activar este usuario, tenemos que incluir una directiva **createUser** en el archivo /**var/lib/snmp/snmpd.conf**, aunque no asignemos ninguna contraseña al usuario, de la siguiente forma:

createUser manolito

- **rwuser USUARIO [noauth|auth|priv [OID]]**

  Idéntica a la anterior, solo que da permisos de lectura y escritura al nuevo usuario.

  Crearemos un usuario con acceso de lectura y escritura sobre el grupo **system**, y con nivel de autenticación **auth**:

  rwuser alfredito auth system

- **createUser [-e ENGINEID] nombre (MD5|SHA) contraseña_auth [DES] [contraseña_priv]**

  Crea un nuevo usuario capaz de comunicarse con el agente mediante SNMPv3. Este nombre de usuario debe haberse definido previamente mediante una directiva **rouser** o **rwuser**.

  Cuando asociemos estos usuarios con nombres de grupo para asociarles vistas, el nombre especificado aquí es el que debe incluirse como nombre de seguridad en la directiva **group** descrita en el apartado anterior. Podemos escoger entre los algoritmos MD5 y SHA para los mensajes autenticados, si bien necesitamos tener instalado el paquete OpenSSL para usar SHA. Para los mensajes encriptados sólo podremos usar DES.

  Disponemos de tres posibles modos de acceso:

  - **noauth**: cuando llegue una petición de este usuario, no se requerirá ninguna contraseña para llevarla a cabo.
  - **auth**: se usa una contraseña solamente para la autenticación del usuario, ya sea mediante el protocolo MD5 o SHA.
  - **priv**: con este modo de acceso, las PDU (mensaje) que viajen por la red irán encriptadas mediante el algoritmo DES. Por ello, se requiere una clave adicional para el cifrado y descifrado.

  Aquí definimos dos contraseñas. La primera es la que se usará cuando el nivel de autenticación sea **auth**, y la segunda cuando sea **priv** (ver la directiva **access**). De todos modos, si la segunda está en blanco se tomará la primera también para el modo de acceso **priv**.

Al usar esta directiva debemos tener en cuenta una cuestión: si la introducimos en nuestro archivo **/etc/snmp/snmpd.conf** cualquier usuario con permiso de lectura sobre este archivo podría leer sin problemas los nombres de usuario y las contraseñas, con lo que esta nueva política de acceso no nos serviría para mucho. En lugar de esto, tenemos que incluirlas en el archivo **/var/lib/snmp/snmpd.conf** y justo después reiniciar el agente. Esto causará que el agente vuelva a leer este archivo, creando los usuarios y borrando las líneas correspondientes (eliminando así las contraseñas del disco duro), para después guardar en este mismo archivo las claves que se derivan de estas contraseñas.

Ejemplo:

Primero, detenemos el agente:

 # /etc/init.d/snmpd stop

Ahora asignamos una contraseña de autenticación al usuario **alfredito** que hemos creado previamente con la directiva **rouser**, escribiendo la siguiente línea en el archivo **/var/lib/snmpd.conf**:

createUser alfredito MD5 pass_alfredito

Ahora, volvemos a ejecutar el agente:

 # /etc/init.d/snmpd start.

Si editamos el archivo **/var/lib/snmpd.conf**, podremos comprobar que la directiva introducida se ha transformado en lo siguiente:

usmUser 1 3
0x80001f8804303031313232333334343535414142424343444 4
0x616c6672656469746f00 0x616c6672656469746f00 NULL .1.3.6.1

A partir de ahora, podremos realizar peticiones SNMPv3 al agente mediante el nombre de usuario y la contraseña que acabamos de crear. Es importante señalar que hay que incluir el modo de acceso mediante la opción **-l**, de la siguiente forma:

 # snmpwalk v 3 u alfredito l authNoPriv a MD5 A pass_alfredito averia system

4.3.5. Envío de notificaciones

Una notificación es un mensaje SNMP no solicitado que envía el agente a un receptor determinado. Generalmente indica un cambio de estado en el dispositivo gestionado por el agente, o una condición de alarma. SNMP define dos tipos de notificaciones: **traps** e **informs**.

Un **trap** es una notificación no confirmada por el receptor. Esto hace que necesitemos un mecanismo para detectar la pérdida de *traps* en la red si las notificaciones que enviamos son de carácter crítico.

Un **inform** es una notificación que requiere confirmación por parte del receptor, lo cual la hace más adecuada para situaciones críticas. Sin embargo, estas notificaciones no fueron incluidas en el protocolo hasta la versión SNMPv2c, con lo que no serán permitidas en agentes que soporten solamente la versión SNMPv1.

- **authtrapenable NUMERO**

  Cuando **NUMERO**=1, provoca el envío de traps ante fallos de autenticación (comunidad incorrecta en la petición, o intento de acceder a un objeto no perteneciente a la vista de la comunidad especificada). El objeto al que se refiere esta directiva es **snmpEnableAuthenTraps**, y el hecho de usarla provocará que sea de sólo lectura.

  Activaremos esta opción en nuestro agente:

  authtrapenable 1

- **trapcommunity CADENA**

  Especifica la comunidad por defecto que será usada al enviar notificaciones. Esta directiva debe aparecer en el archivo de configuración antes que las cuatro siguientes. En nuestra configuración, usaremos la comunidad por defecto **private**:

  trapcommunity private

- **trapsink HOST [COMUNIDAD [PUERTO]]**

  Especifica el host (y opcionalmente el puerto) que recibirá los **traps** SNMPv1 enviados. Podemos especificar también la comunidad a usar en el envío, aunque si no lo hacemos se usará la comunidad por defecto. Si usamos esta directiva, el agente enviará *traps* también en los errores de autenticación. Puede aparecer varias veces para especificar múltiples destinos.

- **trap2sink HOST [COMUNIDAD [PUERTO]]**

  Esta directiva es idéntica a la anterior, solo que trabaja con los **traps** de SNMPv2c.

- **informsink HOST [COMUNIDAD [PUERTO]]**

  Es la equivalente a las dos anteriores para el envío de **informs**.

  Vamos a usar estas tres últimas directivas para mandar todo tipo de notificaciones al host **192.168.0.3**:

  trapsink 192.168.0.3
  trap2sink 192.168.0.3
  informsink 192.168.0.3

- **trapsess [SNMPCMD_ARGS] HOST**

  Es una directiva genérica de configuración que permite especificar cualquier destino con cualquier versión de SNMP.

  A partir de su quinta versión, el agente incorpora soporte DisMan (*Distributed Management Event MIB*) para la MIB. Esta nueva característica permite al agente monitorizar sus propios datos en intervalos regulares, y/o mandar notificaciones cuando se den ciertas condiciones. Estas notificaciones se mandarán la primera vez que se cumplan sus condiciones de envío, y para que vuelvan a enviarse debe ocurrir que las condiciones de envío vuelvan a ser falsas al menos una vez.

Según la página del manual, este soporte no ha sido probado exhaustivamente y, además, se sabe que no está completo. Sin embargo, las directivas aquí explicadas deberían funcionar correctamente.

Si el agente fue compilado con soporte para **DISMAN-EVENT-MIB,** podremos usar las siguientes directivas para monitorizar objetos y mandar notificaciones referentes a ellos.

Enviaremos los **traps** SNMPv3, usando el nombre de usuario **manolito**, con nivel de autenticación **noAuthNoPriv**, haciendo uso de la siguiente directiva:

trapsess v3 e 001122334455AABBCCDD l noAuthNoPriv u manolito 192.168.0.3

Si queremos hacer lo mismo, pero para el envío de **informs**:

trapsess Ci v3 l noAuthNoPriv u manolito 192.168.0.3

- **agentSecName NOMBRE**

El soporte para **DISMAN-EVENT-MIB** requiere un nombre de usuario, mediante el cual escaneará el agente. Éste puede ser especificado mediante esta directiva o usando la opción **-u** de la directiva **monitor** explicada a continuación. El nombre que especifiquemos debe estar declarado mediante una directiva **rouser** o **rwuser**.

El nombre de seguridad de nuestro agente será el siguiente:

agentSecName manolito

- **monitor [OPCIONES] NOMBRE EXPRESION**

Esta directiva manda al agente que se monitorice a sí mismo en busca de problemas basados en la expresión especificada. Ésta puede ser una expresión simple basada en un OID, un operador de comparación (!=, ==, >=, <=, <, >) y un valor entero. El nombre es una cadena arbitraria usada para propósitos administrativos. Pueden especificarse las siguientes opciones:

  - **-t**

    Usar un monitor de intervalo en vez de uno booleano. Esto significa que **EXPRESION** debe contener un valor máximo y otro mínimo. Si el valor del objeto supera el máximo se disparará una alarma, disparándose otra distinta cuando éste sea inferior al mínimo.

  - **-r FRECUENCIA**

    Monitoriza la expresión en intervalos de tiempo cuyo valor será el de **FRECUENCIA**.

  - **-u NOMBRE_SEGURIDAD**

    Usar el nombre de seguridad especificado para las comprobaciones de los objetos. Tenemos que haber dado previamente derechos de acceso a dicho nombre de seguridad mediante las directivas **rouser** o **rwuser**. Si esta opción no se incluye, se utilizará el nombre de seguridad por defecto especificado mediante la directiva **agentSecName**.

- **-o OID**

  Especifica nombres de objetos adicionales que serán incluidos en la notificación, además de los objetos referidos por la directiva. Esto puede ser útil cuando queramos mandar toda una fila de una tabla, aunque monitoricemos sólo una de las posiciones.

- **-e NOMBRE_EVENTO**

  Especifica un nombre de evento que describirá qué hacer cuando la condición se cumpla. Usualmente será el nombre de un **notificationEvent**.

Podríamos, por ejemplo, monitorizar la tabla **hrSWRunPerfTable** en busca de procesos que consuman más de 10 MB de memoria RAM, para devolver su nombre en un **trap**:

monitor o sysUpTime.0 o hrSWRunName "Procesos acaparadores" hrSWRunPerfMem > 10000

Si quisiéramos monitorizar la tabla **ifTable** buscando aquellas interfaces de red que hayan recibido más de diez tramas erróneas, usaríamos:

monitor o sysUpTime.0 e "Errorcito" o ifDescr "Interfaces gafes" ifInErrors > 10

– **notificationEvent NOMBRE NOTIFICACION [[-w] OID ...]**

Esta directiva creará un evento de notificación, que será disparado mediante una directiva **monitor** que se refiera a él con su opción **-e**. **NOTIFICACION** debe ser el **OID** de una notificación, pudiendo incluirse OIDs adicionales, y será el tipo de notificación que se enviará. Por defecto se incluirá al OID el sufijo de la columna u objeto monitorizado, a menos que incluyamos la opción **-w**. Por ejemplo, si monitorizamos la tabla **ifTable** y queremos incluir el valor de **ifDescr** para las filas que produzcan notificaciones, entonces no incluiremos la opción **–w** y se añadirá automáticamente el campo **INDEX**. En cambio, si monitorizamos el objeto **sysContact.0**, ningún sufijo debe ser añadido y entonces sí que usaremos dicha opción.

Asociaremos a la última directiva **monitor** del punto anterior un evento para el envío de un **trap** de tipo **linkDown** mediante la siguiente directiva:

notificationEvent "Errorcito" .1.3.6.1.2.1.11.2

– **linkUpDownNotifications yes**

Esta directiva hará que el agente monitorice la tabla de interfaces (**ifTable**) para determinar cuándo una interfaz es activada o desactivada. Cuando esto pase, se lanzarán notificaciones **linkUp** o **linkDown** respectivamente.

Vamos a incluir esta directiva en nuestra configuración:

linkUpDownNotifications yes

– **defaultMonitors yes**

Activa las siguientes directivas **monitor**:

monitor o prNames o prErrMessage "process table" prErrorFlag != 0
monitor o memErrorName o memSwapErrorMsg "memory"
memSwapError != 0
monitor o extNames o extOutput "extTable" extResult != 0
monitor o dskPath o dskErrorMsg "dskTable" dskErrorFlag != 0
monitor o laNames o laErrMessage "laTable" laErrorFlag != 0
monitor o fileName o fileErrorMsg "fileTable" fileErrorFlag != 0

También la incluiremos en nuestro archivo:

defaultMonitors yes

4.3.6. Configuración de MIB extensible

El agente SNMP obtiene mucha de su información de la sección **1.3.6.1.4.1.2021** (**.iso.org.dod.private.enterprises.ucdavis**) del árbol de la MIB. Cada subsección de esta MIB tiene los siguientes campos:

- **.ID.1 index**

  Es un índice entero. Para escalares, siempre tiene el valor 1. Para tablas, es un índice de filas.

- **.ID.2 name**

  Es el nombre de esta entrada de la tabla.

- **.ID.100 errorFlag**

  Es un flag que devuelve el valor 1 si se detectó un error para esta entrada de la tabla.

- **.ID.101 errMessage**

  Es una cadena que describe el error que provocó que el campo anterior se pusiera a uno.

- **.ID.102 errFix**

  Este objeto actúa como un disparador. Si esta entrada vale 1, se ejecutará un programa pasándole el nombre de la entrada de la tabla como argumento. El programa se especifica en el campo **errFixCmd**.

Este convenio permite al agente examinar fácilmente ciertas circunstancias concretas poniendo el valor de **ID** correspondiente a la sección que se quiera monitorizar, y escaneando el valor de **.ID.100** en busca de errores.

Las siguientes directivas pueden usarse en este apartado:

- **proc NOMBRE [MAX [MIN]]**

  Comprueba que el proceso cuyo nombre es **NOMBRE** esté ejecutándose actualmente. **MAX** y **MIN** son el número máximo y mínimo de instancias de dicho proceso. Si no especificamos ninguno de los dos valores, **MAX** se asume como infinito y **MIN** como uno. Si especificamos **MAX** pero no **MIN**, éste último se asumirá como cero. Éstos se refieren al número máximo y mínimo de instancias del proceso permitidas.

  Podríamos comprobar la existencia de un proceso llamado **amule** en nuestro sistema, y al mismo tiempo que no se esté ejecutando más de una vez, con la siguiente directiva:

  proc "amule" 1 1

- **procfix NOMBRE PROGRAMA ARGUMENTOS**

  Esta directiva registra un comando que sabrá cómo arreglar los errores ocurridos con el proceso especificado por **NOMBRE**, obteniendo la información de éste de la tabla **enterprises.ucdavis.procTable**. Cuando hagamos un **snmpset** al campo **prErrFix** correspondiente, poniéndolo a 1, se ejecutará el comando registrado.

  Podemos, por ejemplo, registrar en un **log** la muerte del proceso **amule**:

  procfix "amule" /bin/sh /usr/local/bin/watchAmule.sh

- **exec [MIBNUM] NOMBRE PROGRAMA ARGUMENTOS**

  Si **MIBNUM** no se incluye en esta directiva, el agente ejecutará **PROGRAMA** con los argumentos especificados, guardando el estado de salida y la primera línea de la salida estándar de dicho programa para futuras consultas a las columnas **1.3.6.1.4.1.2021.8.1.100** y **1.3.6.1.4.1.2021.8.1.101**, respectivamente. El resto de salida del programa se omitirá.

  Si incluimos **MIBNUM** actuará de la misma forma, pero guardará el estado de salida en **MIBNUM.100.0** y la salida completa del programa en **MIBNUM.101**. Cada línea de la salida se guardará en una fila de la tabla especificada (primera línea en **MIBNUM.101.1**, segunda línea en **MIBNUM.101.2**, etc). **MIBNUM** debe ser especificado en formato numérico. Hay que tener en cuenta que el agente guarda en una caché interna el estado y la salida del programa durante unos treinta segundos antes de volcarlos a la MIB, para aumentar la eficiencia y la consistencia de la información en peticiones consecutivas. Puede forzarse el volcado escribiendo un 1 en el objeto **1.3.6.1.4.1.2021.100.VERCLEARCACHE** mediante una orden **snmpset**.

  Probaremos esta característica haciendo que el agente ejecute un script que registre en un **log** la fecha y la hora de su ejecución:

  exec "inicia" /bin/sh /usr/local/bin/pruebaExec.sh

  Podemos extender una rama de la MIB con la salida de un programa o script:

  exec .1.3.6.1.4.1.666 "miComando" /bin/sh /usr/local/bin/miComando.sh

A partir de que reiniciemos el agente, las peticiones de lectura sobre ese OID devolverán lo siguiente:

```
$ snmpwalk v1 c public averia .1.3.6.1.4.1.666
SNMPv2SMI::enterprises.666.1.1 = INTEGER: 1
SNMPv2SMI::enterprises.666.2.1 = STRING: "miComando"
SNMPv2SMI::enterprises.666.3.1 = STRING: "/bin/sh/
usr/local/bin/miComando.sh"
SNMPv2SMI::enterprises.666.100.1 = INTEGER: 0
SNMPv2SMI::enterprises.666.101.1 = STRING: "unlugarMancha"
SNMPv2SMI::enterprises.666.102.1 = INTEGER: 0
SNMPv2SMI::enterprises.666.103.1 = ""
```

- **execfix NOMBRE PROGRAMA ARGUMENTOS**

  Trabaja igual que la directiva **procfix**, solo que en esta ocasión el agente monitorizará los valores de la tabla **enterprises.ucdavis.extTable**, actuando cuando el campo **extErrFix** correspondiente esté a 1.

  Incluiremos la siguiente directiva para notificar los problemas ocurridos en la ejecución de **pruebaExec.sh**:

  ```
  execfix "inicia" /bin/sh /usr/local/bin/fixExec.sh
  ```

- **disk RUTA [ESPACIO_MIN | PORCENT_MIN%]**

  Comprueba el espacio disponible en el disco montado en **RUTA**. Si el espacio es menor que **ESPACIO_MIN** en KB o que **PORCENT_MIN%** en tanto por ciento, la entrada asociada en la tabla **1.3.6.1.4.1.2021.9.1.100** se pondrá a 1 y se escribirá el mensaje de error correspondiente en **1.3.6.1.4.1.2021.9.1.101**.

  Vamos a comprobar que queden, al menos, 100 MB libres en la partición raíz:

  ```
  disk / 100000
  ```

- **includeAllDisks PORCENT_MIN%**

  Añade todos los discos que puedan ser encontrados en el sistema mediante las llamadas al sistema **setmntent** y **getmntent**, o **fopen** y **getmntent**, o **setfsent** y **getfsent**. Si ninguna de estas llamadas se encuentra en el sistema, añadirá la partición raíz usando la llamada **statfs**. Los parámetros especificados mediante la directiva **disk** sobreescriben a los de esta directiva. Hay que tener en cuenta que sólo incluirá los discos que ya estén montados cuando el agente se inicie.

  Monitorizaremos los discos duros para comprobar que les quede, al menos, un 10% de espacio libre:

  ```
  includeAllDisks 10%
  ```

- **load [MAX1 [MAX5 [MAX15]]]**

  Devuelve un 1 en el objeto **1.3.6.1.4.1.2021.10.1.100** y un mensaje de error en **1.3.6.1.4.1.2021.10.1.101** cuando las medias de uno, tres o quince minutos excedan los valores máximos asociados en la llegada de una solicitud **GET** sobre el campo **laLoad** de la tabla **private.enterprises.ucdavis.laTable**.

Vamos a establecer las medias máximas en los siguientes valores:

load 1 2 4

- **file ARCHIVO [TAMAÑO_MAX]**

  Monitoriza el tamaño de **ARCHIVO** comprobando que no crezca más allá del tamaño máximo especificado en KB, pero sin reportar errores al respecto. **TAMAÑO_MAX** es infinito por defecto.

  Comprobaremos que el archivo /**var**/**log**/**messages** no ocupe más de 4 MB:

  file /var/log/messages 4096

4.3.7. Configuración de proxy

Mediante el soporte para proxy del agente podemos hacer que todas las peticiones sobre un OID queden delegadas a un host distinto.

- **proxy [-Cn NOMBRE_CONTEXTO] [ARGS_SNMPCMD] HOST OID [OID_REMOTO]**

  Cede el control del árbol de la MIB especificado por **OID** al **HOST** descrito. Si se incluye **NOMBRE_CONTEXTO**, asignará el árbol a un nombre de contexto particular dentro del agente local. La forma más adecuada de hacer solicitudes a múltiples agentes a través de un sólo proxy sería asignar a cada agente remoto un nombre de contexto. Entonces, podríamos usar **snmpwalk -n nombreContexto1** para preguntar a un agente remoto, y **snmpwalk -n nombreContexto2** para preguntar a otro. En este aspecto sólo se soporta la versión SNMPv3 para las peticiones, puesto que los nombres de contexto no están soportados para las versiones 1 y 2.

  Podemos también mapear el **OID** local a un nuevo OID remoto, especificado por **OID_REMOTO**. Para la autenticación en las consultas, debemos usar los argumentos correctos en **ARGS_SNMPCMD**.

  Ejemplos:

  Podemos mapear el grupo **system** del host **truca** en el árbol experimental de la MIB de **averia**, mediante la siguiente directiva:

  proxy Cn truca1 v1 c public 192.168.0.3 .1.3.6.1.3.11 .1.3.6.1.2.1.1

  Así, haciendo uso del nombre de contexto **truca1**, podemos consultar el subárbol creado de la siguiente manera:

  # snmpwalk v3 u manolito n truca1 averia .1.3.6.1.3.11

  Podemos también hacer que no sea necesario añadir el nombre de contexto a las consultas haciendo uso de una conexión SNMPv3 con **truca**, con la siguiente directiva:

  proxy v3 u manolito 192.168.0.3 .1.3.6.1.3.10 .1.3.6.1.2.1.1

  Con esta directiva, podremos hacer peticiones como la siguiente:

  # snmpwalk v3 u manolito l noAuthNoPriv averia .1.3.6.1.3.10

4.3.8. Pasando el control a programas externos

Con estas directivas podremos hacer que las peticiones **SET** y **GET** a un OID dado se pasen al programa que queramos.

- **pass OID EJECUTABLE**

  Esta directiva es la que usaremos para este fin. El programa especificado por **EJECUTABLE** puede ser llamado de las siguientes formas:

  - **EJECUTABLE -g OID**
  - **EJECUTABLE -n OID**

    Estas dos directivas se corresponden con las llamadas **GET** y **GETNEXT**. El programa especificado debe dar la respuesta esperada a las peticiones efectuadas a través de su salida estándar, de la siguiente forma: la primera línea debe ser el **OID** correspondiente a la petición, la segunda línea debe ser el tipo del valor devuelto (**string**, **integer**, **unsigned**, **objectid**, **timeticks**, **ipaddress**, **counter** o **gauge**), y la tercera será el valor del objeto solicitado. Por ejemplo, si un programa debe devolver el valor 42 al llegar una petición sobre el objeto **.1.3.6.1.4.100**, debería imprimir las siguientes tres líneas por su salida estándar:

    .1.3.6.1.4.100
    integer
    42

    Para indicar que la petición fue errónea, el programa no debe devolver nada por su salida estándar. El agente generará entonces un mensaje de error SNMP correspondiente a **noSuchName**.

  - **EJECUTABLE -s OID TIPO VALOR**

    Esta será la llamada usada para peticiones de tipo **SET**. **TIPO** puede contener los mismos valores que en la llamada anterior, e indica el tipo de objeto del **VALOR** especificado. Si el programa no devuelve nada, se asumirá que la operación **SET** ha salido bien. Si no es así, tendrá que devolver una de las cadenas de error **not-writable** o **wrong-type**, y el agente enviará el correspondiente mensaje de error al gestor que hizo la petición.

  Por defecto, la única comunidad con permiso para este tipo de operaciones será **private**, o la segunda comunidad definida previamente en el archivo de configuración en caso de no existir ésta.

  Para probar esta característica, usaremos un script de ejemplo llamado **passtest**:

  pass .1.3.6.1.4.1.2021.255 /bin/sh /usr/local/bin/passtest

  Ahora podremos hacer peticiones sobre el **OID** que maneja dicho script:

  # snmpwalk v2c c public averia .1.3.6.1.4.1.2021.255

- **pass_persist OID EJECUTABLE**

  Esta directiva es similar a la anterior, solo que en este caso el programa continuará su ejecución tras recibir la primera petición.

Por convenio, en la inicialización se pasará al programa la cadena "PING\n" por su entrada estándar, debiendo éste imprimir en su salida estándar la cadena "PONG\n".

Para peticiones **GET** y **GETNEXT**, el agente pasará al programa dos líneas, una con el comando correspondiente (**get** o **getnext**) y otra con el **OID** correspondiente a la petición. El programa debe dar su salida en el mismo formato que para la directiva anterior. Para indicar que el programa no pudo procesar la petición, deberá devolver la cadena "NONE\n" a su salida estándar.

5. Recepción de notificaciones

Si antes hemos estudiamos las distintas posibilidades que ofrece el agente de NET-SNMP para el envío de notificaciones, en este apartado abordaremos los aspectos de configuración de las herramientas que incluye NET-SNMP para la recepción de éstas notificaciones. Siguiendo la misma filosofía, enfocaremos este estudio desde el punto de vista de la creación de un fichero de configuración persistente, para no tener que acudir a las páginas del manual o a este tutorial cada vez que queramos comenzar un estudio de las notificaciones recibidas en nuestro host.

5.1. Tipos de notificaciones

Este tema ya ha sido explicado en el punto 4.3.5 (envío de notificaciones) de este tutorial. Se recomienda la lectura de dicho punto antes de seguir con los siguientes apartados de esta sección.

5.2. Funcionamiento del mecanismo de notificaciones

Los agentes SNMP tienen la capacidad de monitorizar objetos de la MIB y mandar notificaciones cuando se den unas condiciones determinadas. Para que esto funcione, tiene que haber "alguien" escuchando en el otro extremo, es decir, debe haber una entidad capaz de recibir y procesar dichas notificaciones. Esta entidad generalmente actuará como un servicio más del sistema, y es la que vamos a configurar en esta sección.

El uso de notificaciones mediante los protocolos SNMPv1 y v2c es sencillo, puesto que en este caso el mensaje es mostrado tal cual al receptor. Sin embargo, cuando usamos SNMPv3 la cosa cambia un poco, puesto que en esta versión se hace uso de la autenticación mediante nombre de usuario y contraseña para la comunicación con el proceso receptor de notificaciones. Por esto tendremos que incluir previamente en la base de datos de usuarios a los usuarios que tendrán permiso para enviarnos notificaciones, o el proceso receptor descartará silenciosamente las notificaciones entrantes.

Generalmente, un usuario queda identificado mediante su nombre (**nombre de seguridad**) y el identificador (**engineID)** de la aplicación receptora de la notificación. Cuando usemos el resto de aplicaciones incluidas en el paquete NET-SNMP (**snmpget**, **snmpwalk**, etc), éstas descubren el **engineID** remoto e introducen en la base de datos de usuarios asociada a este **engineID** el nombre de usuario, el **engineID** y el password. Esto facilitará el proceso de comunicación en peticiones futuras.

El mecanismo de informes para SNMPv3 opera con este mismo principio. Cuando enviamos un informe mediante **snmptrap** usamos el **engineID** remoto, y la combinación de este **engineID** con el **nombre de seguridad** que especifiquemos debe existir en la tabla de usuarios remota. El programa **snmptrap** detectará el **engineID** remoto y creará un mensaje SNMPv3 adecuado para el informe que estemos enviando. Entonces, lo único que tendremos que hacer es crear un usuario en la aplicación remota receptora de nuestros informes.

En cuanto al envío de **traps** SNMPv3, la cosa cambia un poco. La diferencia es que los *traps* utilizan el **engineID** de la aplicación local (la que envía el *trap*), en vez de el de la aplicación remota (la receptora del *trap*). Esto quiere decir que, al crear usuarios en la aplicación receptora, tendremos que crear uno para cada **engineID** desde el cual queramos recibir *traps*, añadiendo una directiva de este estilo en **/var/lib/snmp/snmptrapd.conf**:

createUser e <engineID> nombre_usuario ...

5.3. El demonio snmptrapd

La aplicación receptora de **traps** en el paquete NET-SNMP es el demonio **snmptrapd**. Se ejecuta como un servicio más del sistema, pudiendo detenerlo, ejecutarlo o reiniciarlo a nuestro antojo. Por defecto **snmptrapd** escucha en el puerto 162 de UDP.

5.3.1. Ficheros de configuración

Típicamente, el fichero de configuración específico para **snmptrapd** es **/etc/snmp/snmptrapd.conf**. Al igual que el agente, también leerá las directivas de configuración generales para todas las aplicaciones de NET-SNMP (ubicadas en el fichero **snmp.conf**,), y guardará sus datos persistentes (como las directivas **createUser** explicadas más abajo) en **/var/lib/snmp/snmptrapd.conf**. A continuación pasamos a explicar las directivas que podemos incluir en el fichero específico del demonio receptor de traps.

5.3.2. Ejecución de programas para notificaciones entrantes

- **traphandle OID|default PROGRAMA [ARGUMENTOS ...]**

  Esta directiva configura al demonio **snmptrapd** para que lance el programa especificado cada vez que llegue una notificación correspondiente a **OID**, o cualquier notificación cuyo OID no se corresponda con ninguna otra directiva **traphandle** si incluimos la palabra **default** en vez de un OID válido. El programa recibirá los detalles de la notificación entrante a través de su entrada estándar, con una entrada por línea y en el siguiente formato:

  - **HOSTNAME**

    El nombre del host que envió la notificación.

  - **IPADDRESS**

    La dirección IP del host que mandó la notificación.

  - **VARBINDS**

    Una lista de variables que describen la notificación y las variables involucradas en ésta. El primer elemento de la línea (hasta el primer espacio en blanco) será el **OID**, y el resto de la línea será su valor. El primer OID debe ser **system.sysUpTime.0**, el segundo será **snmpTrapOID.0** (que contendrá una cadena describiendo el tipo de notificación recibida), y los siguientes serán las variables asociadas a la notificación. Para **traps** SNMPv1, el último OID será **snmpTrapEnterprise**, con su valor correspondiente.

  Vamos a usar un sencillo script para guardar las notificaciones entrantes en un archivo, mediante la siguiente directiva:

  traphandle default /usr/bin/traptofile file=/var/log/traptofile.log

5.3.3. Formateo de notificaciones

Mediante las siguientes directivas podremos aplicar un formato a las notificaciones entrantes. Puede usarse para especificar los campos de las notificaciones que mostraremos, e incluso el orden en que estos campos se mostrarán.

- **format1 formato**

  Esta directiva se usa para especificar el formato de los **traps** SNMPv1 entrantes.

  Para mostrar la hora de llegada, el tipo de notificación y el host remitente, usaríamos:

  format1 %02.2h:%02.2j TRAP%w.%q SNMPv1 desde %A\n

- **format2 formato**

  Especifica el formato de los **traps** e **informs** SNMPv2 entrantes. Nótese que SNMPv3 usa el mismo formato de notificación que SNMPv2, con lo que también estaremos especificando el formato de las notificaciones para esta versión.

  Para usar el mismo formato que el descrito en la directiva anterior, usaríamos:

  format2 %02.2h:%02.2j TRAP%w.%q SNMPv2 desde %A\n

5.3.4. Guardando notificaciones entrantes para consultas mediante SNMP

En las últimas versiones de NET-SNMP, el demonio **snmptrapd** soporta la **NOTIFICATION-LOG-MIB**. Esto le sirve para registrar las últimas notificaciones entrantes en las tablas **nlmLogTable** y **nlmLogVariableTable**, de forma que podremos acceder a esta información mediante comandos como **snmptable** y **snmpwalk**.

- **dontRetainLogs true**

  El uso de esta directiva significará la desactivación de esta característica en el demonio, y por tanto la no retención de **traps** entrantes en memoria.

5.3.5. Creación de usuarios

La creación de usuarios SNMPv3 en el demonio es idéntica a la explicada para el agente. De igual forma, el uso recomendable de esta característica es incluir las directivas de creación de usuarios en el archivo **/var/lib/snmp/snmptrapd.conf**, con el fin de que no hallan archivos en el sistema que contengan los nombres de usuarios y contraseñas en claro. Los usuarios creados serán aquellos a los que permitiremos enviarnos notificaciones mediante el protocolo SNMPv3.

- **createUser nombre (MD5|SHA) auth_password [DES]**

  El uso de esta directiva es idéntico a su homónima para el fichero de configuración del agente. Por ejemplo, para que **truca** reciba los **traps** SNMPv3 que **averia** le mande con el nombre de usuario **manolito**, hay que incluir esta directiva en su archivo **/var/lib/snmp/snmptrapd.conf**:

  createUser e 001122334455AABBCCDD manolito

  Para la recepción de **informs**, habría que añadir también:

  createUser manolito

5.3.6. Reenvío de notificaciones

El demonio **snmptrapd** es capaz de reenviar ciertas notificaciones a otros hosts, lo cual es útil si queremos distribuir la gestión de estas notificaciones entre varias estaciones gestoras dependiendo de las variables involucradas en ellas.

- **forward OID|default DESTINO**

  Reenvía las notificaciones relacionadas con el **OID** especificado al host **DESTINO**. Si en vez de OID especificamos la palabra **default**, todas las notificaciones se reenviarán a **DESTINO**.

  Si quisiéramos reenviar los **traps** de tipo **linkDown** al host **averia**, usaríamos:

  forward .1.3.6.1.6.3.1.1.5.3 192.168.0.1

5.3.7. Directivas adicionales

- **ignoreAuthFailure (1 | yes | true | 0 | no | false)**

  Ordena al demonio que ignore todas aquellas notificaciones entrantes referentes a errores de autenticación en el agente.

  doNotLogTraps (1 | yes | true | 0 | no | false)

  Especificando **yes**, **true** ó 1, el demonio no guardará las notificaciones entrantes en ningún archivo de **log**.

- **logOption CADENA 4**

  Especifica dónde se deben guardar las notificaciones entrantes. **CADENA** tiene el formato especificado en **LOGGING OPTIONS**.

  Para registrar las notificaciones en la salida de error:

  logOption e

  Para registrar las notificaciones en la salida estándar:

  logOption o

  Para registrar las notificaciones vía **syslog**, como demonio:

  logOption s d

  Para registrar las notificaciones en un archivo:

  logOption f /var/log/traps_format.log

- **outputOption CADENA**

  Especifica las opciones para el formateo de la salida del demonio. El formato de **CADENA** puede especificarse mediante las opciones contenidas en **OUTPUT OPTIONS**.

  Por ejemplo, si sólo queremos que se muestren los nombres de los objetos y sus valores, usaríamos:

  outputOption s

  Antes de usar esta directiva, el formato mostrado es el siguiente:

  HOSTNAME: averia.localdomain
  IP_ADDRESS: 192.168.0.1

  SNMPv2MIB::sysUpTime.0 = 0:0:11:44.56
  SNMPv2MIB::snmpTrapOID.0 = NETSNMPAGENT
  MIB::nsNotifyShutdown
  SNMPCOMMUNITYMIB::snmpTrapAddress.0 = 212.122.104.75
  SNMPCOMMUNITYMIB::snmpTrapCommunity.0 = private

  Al incluirla en el archivo de configuración y reiniciar el demonio, el formato de **trap** quedaría así:

  HOSTNAME: averia.localdomain
  IP_ADDRESS: UDP: [192.168.0.1]:34046

  sysUpTimeInstance = 0:0:04:04.69
  snmpTrapOID.0 = nsNotifyShutdown
  snmpTrapAddress.0 = 212.122.104.75
  snmpTrapCommunity.0 = private

- **printEventNumbers (1 | 0)**

  Especifica si se mostrarán los números de evento en notificaciones de tipo alarma (cuando la alarma se dispare, cuando se desactive, etc).

  printEventNumbers 1

- **doNotFork (1 | 0)**

  Mediante esta directiva podemos ordenar al demonio que permanezca asociado al shell desde el cual se le llamó.

  doNotFork 0

- **pidFile CADENA**

  Especifica el archivo en el que queremos guardar el identificador de proceso para el demonio.

  pidFile /var/lib/snmp/snmptrapd.PID