Agente SNMP de Monitorización

Implementación en Python con pysnmp

Ana Lushan Montuenga & Daniel Modrego

Gestión de Red

Arquitectura del Agente

Componentes Principales

- Motor SNMP asíncrono
- Responders personalizados
- Muestreador CPU asíncrono
- Sistema de notificaciones

OIDs Implementados

- MIB-II System
- MIB Empresarial
  - manager, managerEmail
  - cpuUsage, cpuThreshold

Uso de asyncio

Permite monitorizar CPU de forma asíncrona sin bloquear peticiones SNMP

Componentes del Agente SNMP

Clases Principales

- `MibDataStore`
  - Almacenamiento de datos MIB
  - Persistencia en JSON
  - Gestión de OIDs
- `JsonGetCommandResponder`
  - Manejo de peticiones GET
- `JsonGetNextCommandResponder`
  - Manejo de peticiones GETNEXT
- `JsonSetCommandResponder`
  - Manejo de peticiones SET
  - Validación de permisos

Funciones Principales

- `python_to_snmp()`: Conversión Python → SNMP
- `snmp_to_python()`: Conversión SNMP → Python
- `request_observer()`: Captura de securityName
- `send_trap()`: Envío de traps SNMP
- `send_email()`: Alertas por correo
- `cpu_sampler()`: Monitorización de CPU
- `main()`: Inicialización del agente

MibDataStore: Almacén de Datos (1/2)

```
class MibDataStore:
        def __init__(self):
                self.data = {
                        'manager': 'NetworkAdmin',
                        'managerEmail': '[email protected]',
                        'cpuUsage': 0,
                        'cpuThreshold': 80,
                        'sysDescr': 'Mini SNMP Agent',
                        'sysName': socket.gethostname(),
                        'sysLocation': 'Data Center',
                }
                self.above_threshold = False
                self.start_time = time.time()
                self.load_from_json()
```

MibDataStore: Almacén de Datos (2/2)

```
# Relacionar un OID a la clave interna del diccionario
def oid_to_key(self, oid):
        # OIDs personalizados de empresa
        if oid == OID_MANAGER:
                return 'manager'
        elif oid == OID_MANAGER_EMAIL:
                return 'managerEmail'
        elif oid == OID_CPU_USAGE:
                return 'cpuUsage'
        elif oid == OID_CPU_THRESHOLD:
                return 'cpuThreshold'
        # Grupo System de MIB-II
        elif oid == SYS_DESCR:
        return 'sysDescr'
        ...
        return None
```

```
# Otras funciones:
# Cargar valores almacenados desde el JSON
def load_from_json(self):
# Guardar datos persistentes en el JSON
def save_to_json(self):
# Calcular upTime (tiempo desde arranque)
def get_sysuptime(self):
```

Por qué un diccionario centralizado

Ventajas del diseño

- **Simplicidad**: Evita bases de datos
- **Mapeo directo**: OID a valor
- **Extensible**: Fácil añadir atributos

Cliente solicita OID → oid_to_key() traduce → acceso directo → conversión SNMP → respuesta

Responder GET: Lectura de Variables

```
class JsonGetCommandResponder(cmdrsp.GetCommandResponder):
        def handle_management_operation(self, snmpEngine, stateReference, contextName, PDU):
                varBinds = v2c.apiPDU.get_varbinds(PDU)
                rspVarBinds = []
                for idx, (oid, val) in enumerate(varBinds, 1):
                        oid_tuple = tuple(oid)
                        key = mib_store.oid_to_key(oid_tuple)
                        if key is None:
                                rspVarBinds.append((oid, rfc1905.NoSuchObject()))
                        else:
                                if key == 'sysUpTime':
                                        value = mib_store.get_sysuptime()
                                else:
                                        value = mib_store.data[key]
                        snmp_value = python_to_snmp(key, value)
                if errorStatus:
                        rspVarBinds = [(oid, v2c.Null()) for oid, val in varBinds]
                self.send_varbinds(snmpEngine, stateReference, errorStatus, errorIndex, rspVarBinds)
```

Responder GETNEXT: Navegación del Árbol

```
class JsonGetNextCommandResponder(cmdrsp.NextCommandResponder):
        def handle_management_operation(self, snmpEngine, stateReference, contextName, PDU):
                varBinds = v2c.apiPDU.get_varbinds(PDU)
                rspVarBinds = []
                for idx, (oid, val) in enumerate(varBinds, 1):
                        oid_tuple = tuple(oid)
                        # Buscar el siguiente OID servido
                        next_oid = None
                        for candidate in ORDERED_OIDS:
                                if candidate > oid_tuple:
                                        next_oid = candidate
                                        break
                        if next_oid is None:
                                rspVarBinds.append((oid, rfc1905.EndOfMibView()))
                        else:
                                ... # Se envía el valor igual que en el GET
```

ORDERED OIDS: Fundamental para GETNEXT

¿Qué es?

Lista pre-ordenada de todos los OIDs del agente

¿Por qué es necesario?

- SNMP GETNEXT requiere orden lexicográfico estricto
- Permite implementar SNMP WALK correctamente
- Búsqueda eficiente del siguiente OID válido

Responder SET: Validaciones (1/2)

```
class JsonSetCommandResponder(cmdrsp.SetCommandResponder):
        def handle_management_operation(self, ...):
                # NIVEL 1: Verificar permisos de escritura
                if current_security_name != b'private-user':
                        self.send_varbinds(snmpEngine, stateReference, 6, 1, ...)
                        return
                for idx, (oid, val) in enumerate(varBinds, 1):
                        key = mib_store.oid_to_key(tuple(oid))
                        # Proteger variables de solo lectura
                        if key in ['sysDescr', 'sysUpTime', 'cpuUsage']:
                                errorStatus = 17  # notWritable
                                break
                # NIVEL 2: Validar tipo de dato
                if key == 'cpuThreshold':
                        if not isinstance(val, (v2c.Integer, rfc1902.Integer32)):
                        errorStatus = 7  # wrongType
                        ...
```

Responder SET: Validaciones (2/2)

```
# NIVEL 3: Validar rango de valores
python_value = snmp_to_python(key, val)
if key in ['manager', 'managerEmail', 'sysContact']:
        if len(python_value) > 255:
                errorStatus = 10  # wrongValue
elif key == 'cpuThreshold':
        if not (0 <= python_value <= 100):
                errorStatus = 10  # wrongValue
elif key == 'sysServices':
        if not (0 <= python_value <= 127):
                errorStatus = 10; errorIndex = idx; break
# Guardar valor validado
mib_store.data[key] = python_value
# Persistir cambios
if errorStatus == 0:
        mib_store.save_to_json()
```

Monitorización CPU Asíncrona

```
async def cpu_sampler(snmpEngine):
        while True:
                try:
                        cpu_usage = int(psutil.cpu_percent(interval=None))
                        mib_store.data['cpuUsage'] = cpu_usage
                        threshold = mib_store.data['cpuThreshold']
                        if cpu_usage > threshold and not mib_store.above_threshold:
                                mib_store.above_threshold = True
                                # Enviar alarma (TRAP & Email)
                                await send_trap(cpu_usage, threshold)
                                await send_email(cpu_usage, threshold)
                        elif cpu_usage <= threshold and mib_store.above_threshold:
                                mib_store.above_threshold = False
                except asyncio.CancelledError:
                        ...
                await asyncio.sleep(5)
```

SNMP Trap: Notificación

```
async def send_trap(cpu_usage, cpu_threshold):
        trapEngine = engine.SnmpEngine()
        try:
                agent_uptime = mib_store.get_sysuptime()
                SNMP_TRAP_OID = (1, 3, 6, 1, 6, 3, 1, 1, 4, 1, 0)
                TRAP_TYPE_OID = BASE_OID + (2, 1)  # Identificador de evento cpuThresholdExceeded
                errorIndication, errorStatus, errorIndex, varBinds = await send_notification(
                trapEngine, CommunityData('private', mpModel=1),
                await UdpTransportTarget.create((TRAP_HOST, TRAP_PORT)),
                ContextData(), 'trap',
                ObjectType(ObjectIdentity(SYS_UP_TIME), v2c.TimeTicks(agent_uptime)),
                ObjectType(ObjectIdentity(SNMP_TRAP_OID), ObjectIdentifier(TRAP_TYPE_OID)),
                ...
                )
                if errorIndication:
                ...
        finally:
        trapEngine.close_dispatcher()
```

Control de Acceso VACM

```
# Usuarios y comunidades
config.add_v1_system(snmpEngine, 'public-user', 'public')
config.add_v1_system(snmpEngine, 'private-user', 'private')
# Grupos
config.add_vacm_group(snmpEngine, 'public-group', 2, 'public-user')
config.add_vacm_group(snmpEngine, 'private-group', 2, 'private-user')
# Vistas
config.add_vacm_view(snmpEngine, 'read-view', 'included',
(1, 3, 6, 1, 2, 1, 1), '')
config.add_vacm_view(snmpEngine, 'write-view', 'included', BASE_OID, '')
# Accesos: public (RO), private (RW)
config.add_vacm_access(snmpEngine, 'public-group', '', 2,
'noAuthNoPriv', 'exact', 'read-view', '', 'notify-view')
config.add_vacm_access(snmpEngine, 'private-group', '', 2,
'noAuthNoPriv', 'exact', 'read-view', 'write-view', 'notify-view')
```

Captura de securityName: Observer

Funcionamiento:

- Patrón observador en PySNMP
- Intercepta eventos internos del engine
- Punto de ejecución: `rfc3412.receiveMessage:request`
- Captura antes del procesamiento

Ventaja del Observer: No invasivo, acceso limpio a metadatos internos sin modificar lógica de respuesta

Código:

```
current_security_name = None
def request_observer(snmpEngine, execpoint, variables, cbCtx):
        global current_security_name
        if execpoint == 'rfc3412.receiveMessage:request':
                current_security_name = variables.get('securityName', b'')
```

Conclusiones

Logros Principales y Aprendizajes Técnicos

- Implementación completa de un agente SNMP funcional con soporte SNMPv1/v2c
- Integración de MIB-II System y MIB empresarial personalizada (PEN 28308)
- Sistema de monitorización de CPU con alertas: TRAP (SNMP) + Email (SMTP)
- Control de acceso con comunidades public (RO) y private (RW)
- Diseño de objetos escalares con tipos SNMP apropiados
- Manejo de comandos GET, GETNEXT y SET
- Programación asíncrona con asyncio

Uso de IA en el Desarrollo del Proyecto

Aportaciones Positivas

- **Diseño de MIB empresarial**
  - Estructura jerárquica correcta
  - Definición apropiada de OIDs
  - Tipos SNMP bien asignados
- **Arquitectura inicial**
  - Esqueleto del código base
  - Estructura de clases y módulos
  - Patrones de diseño sugeridos
- **Documentación**
  - Explicación de conceptos SNMP
  - Referencias a RFCs relevantes
  - Comentarios de código iniciales

Limitaciones Encontradas

- **Código con errores significativos**
  - Sintaxis incorrecta en varias secciones
  - Incompatibilidades con versiones de pysnmp
  - Imports y dependencias desactualizadas
  - Necesidad de refactorización extensa
- **Lógica incompleta**
  - Gestión incorrecta de VACM
  - Falta de validaciones robustas
  - Manejo inadecuado de casos extremos

Conclusión: La IA fue útil como *punto de partida* y para comprender conceptos complejos, pero requirió **trabajo humano extensivo** para depuración, corrección de errores y adaptación a los requisitos específicos del proyecto.

Prompt inicial

```
Eres un experto en programación de agentes SNMP con Python y pysnmp. Tu tarea es ayudarme a desarrollar un
agente SNMP personalizado siguiendo estas especificaciones exactas:

OBJETIVO: Crear un mini-agente SNMP en Python que:
1. Exponga 4 objetos escalares: manager (RW), managerEmail (RW), cpuUsage (RO), cpuThreshold (RW)
2. Monitorice CPU cada 5 segundos con asyncio y psutil
3. Envíe TRAP + email cuando cpuUsage > cpuThreshold (edge-triggered)
4. Soporte GET, GETNEXT y SET con validaciones completas
5. Persista datos en JSON
6. Implemente VACM con comunidades "public" (RO) y "private" (RW)

REQUISITOS TÉCNICOS:
- MIB personalizada en SMIv2 bajo OID 1.3.6.1.4.1.28308 (Zaragoza Network Management Research Group)
- Tres Command Responders: JsonGet, JsonGetNext, JsonSet
- Validaciones SET: tipo, rango, longitud, acceso (errores: wrongType, wrongValue, notWritable, noAccess)
- Mapa OID → clave y lista ordenada de OIDs para GETNEXT lexicográfico
- Función async cpu_sampler() que actualiza cpuUsage y detecta cruce de umbral
- Envío de TRAP SNMPv2c a 127.0.0.1:162 con varBinds: cpuUsage, cpuThreshold, managerEmail, sysUpTime
- Envío de email via smtplib a localhost:1025

ESTRUCTURA DE ARCHIVOS:
mini_agent.py (main), mib_state.json (persistencia), MYAGENT-MIB.txt (MIB SMIv2), README.md (docs)

CONSIDERACIONES:
- sendPdu() debe llamarse inline para mantener stateReference válido
- NoSuchObject() para GET de OID inexistente, EndOfMibView() para fin de GETNEXT
- DisplayString ↔ v2c.OctetString(utf8), Integer32 ↔ v2c.Integer()
- Cargar JSON al inicio, guardar tras cada SET exitoso
- Edge-triggered notification: solo al cruzar umbral, no mientras está por encima

Ayúdame a desarrollar el código completo, paso a paso, explicando cada componente crítico.
```

Preguntas?