



Universidade do Minho
Escola de Engenharia
Mestrado em Engenharia Informática

Unidade Curricular de Gestão e Segurança de Redes

Ano Letivo de 2024/2025
Data de Entrega : 14/07/2025

L-SNMPvS para Sensorização



Flávio Silva (PG57539)

GSR

Introdução

Este documento oferece uma visão geral abrangente do projeto L-SNMPvS (Protocolo de Gerenciamento de Rede Seguro e Leve para Sensores). O projeto implementa um protocolo personalizado baseado em UDP para monitoramento e gerenciamento de dispositivos IoT simulados (agentes) a partir de um monitor central. As principais características incluem um modelo de segurança personalizado para comunicação criptografada e autenticada, um MIB (Base de Informações de Gerenciamento) local no agente e interfaces de usuário tanto de linha de comando quanto gráficas para o monitor.

Conteúdo

0.1	Estratégias, Decisões, Mecanismos e Tecnologias Adotadas	2
0.1.1	Objetivo do Projeto	2
0.1.2	Estratégias e Decisões	2
0.1.3	Mecanismos e Tecnologias	2
0.1.4	Otimizações	3
0.2	Análise Crítica das Principais Funções e Classes	3
0.2.1	Classes Principais	3
0.2.2	Funções Críticas	6
0.3	Etapas de Execução	7
0.3.1	Gerar Arquivos de Configuração	7
0.3.2	Iniciar o Agente	8
0.3.3	Executar um Monitor	8
0.4	Conclusão	10
0.4.1	Melhorias Não Alcançadas	10
0.5	Referências	10

0.1 Estratégias, Decisões, Mecanismos e Tecnologias Adotadas

0.1.1 Objetivo do Projeto

O projeto L-SNMPvS (Lightweight SNMP com Segurança) visa desenvolver um sistema de monitoramento para dispositivos IoT, simulando um dispositivo com sensores virtuais que se comunica com um monitor via protocolo UDP personalizado. A interface gráfica (GUI) permite visualizar dados e configurar parâmetros, com foco em segurança através de autenticação, confidencialidade e integridade.

0.1.2 Estratégias e Decisões

- **Protocolo Leve:** Inspirado no SNMP, o L-SNMPvS utiliza um formato de PDU simplificado (GET, SET, RESPONSE, BEACON) para comunicação eficiente em redes UDP, reduzindo sobrecarga em dispositivos IoT.
- **Arquitetura Cliente-Servidor:** O agente (servidor) responde a requisições do monitor (cliente) e envia beacons periódicos. A GUI integra-se ao monitor para visualização amigável.
- **Segurança:** Adotamos autenticação mútua fraca (via Sender-ID e Receiver-ID), confidencialidade forte (AES-256-CBC) e verificação de integridade (HMAC-SHA-256).
- **Simulação de Sensores:** Sensores virtuais geram dados aleatórios para simular dispositivos IoT, permitindo testes sem hardware físico.
- **Interface Gráfica:** Utilizamos Tkinter e Matplotlib para uma GUI que exibe dados em tempo real e permite configurações interativas.

```
1 MASTER_KEY = b"1234567890abcdef1234567890abcdef"  
2 print(f"MASTER_KEY length: {len(MASTER_KEY)} bytes")  
3 if len(MASTER_KEY) not in (16, 24, 32):  
4     raise ValueError(f"Invalid key size ({len(MASTER_KEY)} bytes) for AES.")
```

Listing 1: Validação do tamanho da chave em `encrypt_config.py`

Este trecho valida o tamanho da chave mestra, corrigindo o erro de 34 bytes (272 bits) causado por codificação incorreta.

0.1.3 Mecanismos e Tecnologias

- **Python:** Escolhido pela facilidade de desenvolvimento, suporte a bibliotecas de criptografia e interface gráfica. Versão 3.11 foi usada para compatibilidade.
- **Criptografia:** A biblioteca `cryptography` fornece AES-256-CBC para criptografia e HMAC-SHA-256 para integridade.
- **UDP:** Protocolo de transporte para comunicação rápida, adequado para IoT, com porta padrão 12345.
- **Tkinter e Matplotlib:** Tkinter para a GUI e Matplotlib para gráficos de dados de sensores, limitados a 100 pontos por sensor.
- **Arquivos de Configuração:** Arquivos JSON (`agent_secrets.json`, `monitor_secrets.json`) armazenam chaves de segurança, criptografados com AES-256-ECB.

```
1 def start_server(self, shutdown_flag):
2     print(f"L-SNMPvS Agent starting on {self.endereco_agente}:{self.porta_agente}")
3     with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as server_socket:
4         server_socket.bind((self.endereco_agente, self.porta_agente))
5         server_socket.settimeout(1.0)
6         while not shutdown_flag.is_set():
7             try:
8                 data, addr = server_socket.recvfrom(2048)
9                 self.handle_request(data, addr, server_socket)
10            except socket.timeout:
11                continue
```

Listing 2: Configuração do servidor UDP em agent.py

Este trecho configura o servidor UDP do agente, escutando requisições na porta 12345.

0.1.4 Otimizações

- **Fila de Dados:** Sensores utilizam uma fila (Queue) para atualizar a L-MIB de forma assíncrona, evitando bloqueios.
- **GET em Lote:** O monitor suporta requisições GET em lote, reduzindo o número de mensagens UDP.
- **Reutilização de Conexões:** O monitor reusa sockets UDP para múltiplas requisições, minimizando overhead.
- **Erro de Chave Resolvido:** Problemas com tamanho de chave (34 bytes em vez de 32) foram corrigidos ajustando a codificação de arquivos e usando uma chave alternativa.

0.2 Análise Crítica das Principais Funções e Classes

0.2.1 Classes Principais

- **VirtualSensor (agent.py):**
 - **Função:** Simula sensores IoT, gerando valores aleatórios dentro de um intervalo configurável (e.g., 0–100 para sensor 1).
 - **Fortes:** Flexibilidade para configurar taxa de amostragem e precisão; integração com fila para atualizações assíncronas.
 - **Limitações:** Apenas dois sensores hard-coded; não suporta adição dinâmica de sensores em tempo de execução.
 - **Código:**

```
1 def generate_value(self):
2     value = random.uniform(self.min_value, self.max_value)
3     return round(value, self.precision)
4
5 def start(self, queue, shutdown_flag):
6     while not shutdown_flag.is_set():
7         if self.update_value():
8             queue.put((self.sensor_id, self.current_value, self.
                last_sampling_time))
```

```
9 time.sleep(0.05)
```

Listing 3: Simulação de sensor em VirtualSensor

Este trecho mostra a geração de valores aleatórios e envio para a fila.

▪ **LMIB (agent.py):**

- **Função:** Gerencia a base de informações (L-MIB) com mapeamento de IIDs para dados de dispositivo e sensores.
- **Fortes:** Thread-safe com `threading.Lock`; suporta Pom GET e SET com validação de tipos e erros.
- **Limitações:** Estrutura de IIDs estática, dificultando expansão para novos dispositivos ou sensores.
- **Código:**

```
1 def set_data(self, iid, value):
2     with self.lock:
3         if iid not in self.iid_info:
4             print(f"SET Error: Unknown IID {iid}")
5             return 4
6         info = self.iid_info[iid]
7         if not info["settable"]:
8             print(f"SET Error: IID {iid} ({info['name']}) is not settable.")
9             return 3
10        if info["type"] == int and not isinstance(value, int):
11            try:
12                value = int(value)
13            except (ValueError, TypeError):
14                print(f"SET Error: Type mismatch for IID {iid}")
15                return 2
```

Listing 4: Método set_data em LMIB

Este trecho valida e configura valores no L-MIB, garantindo segurança de tipo.

▪ **AgenteLSNMPvS (agent.py):**

- **Função:** Implementa o servidor UDP, processa requisições GET/SET, envia beacons e gerencia sensores.
- **Fortes:** Robusto com tratamento de erros para PDUs inválidos; suporta criptografia e HMAC.
- **Limitações:** Beacons não são processados pelo monitor; erro inicial de chave (272 bits) devido a codificação incorreta.
- **Código:**

```
1 def handle_request(self, data, addr, server_socket):
2     decoded = self.decode_pdu(data)
3     if decoded[0] is None:
4         print(f"Failed to decode PDU from {addr}")
5         return
6     tag, msg_type, timestamp, req_msg_id, pdu_payload, sender_id, receiver_id
7     = decoded
8     if tag == 'G':
9         req_iids = [struct.unpack("!H", pdu_payload[i:i+2])[0] for i in range
10        (0, len(pdu_payload), 2)]
11         resp_values = self.handle_get(req_iids)
12         resp_pdu = self.encode_pdu('R', 1, int(time.time()), self.
13        get_next_msg_id(), req_iids, resp_values, [], [], self.agent_id,
14        sender_id)
```

Listing 5: Processamento de requisições em AgenteLSNMPvS

Este trecho processa requisições GET, decodificando e respondendo com valores.

▪ LSNMPvSMonitor (**monitor.py**):

- **Função:** Cliente UDP que envia requisições GET/SET e decodifica respostas.
- **Fortes:** Suporte a GET em lote; integração com GUI; tratamento de erros de timeout.
- **Limitações:** Não processa beacons; erros de criptografia devido a chaves mal configuradas.
- **Código:**

```

1 def get(self, object_name):
2     iid = self.iid_map[object_name]
3     req_pdu = self.encode_pdu('G', 1, int(time.time()), self._get_next_msg_id
4     (), [iid], [], [], [])
5     with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
6         sock.settimeout(self.timeout)
7         sock.sendto(req_pdu, (self.agent_address, self.agent_port))
8         data, addr = sock.recvfrom(1024)
9         tag, msg_type, timestamp, resp_msg_id, remaining_data = self.
10        decode_pdu(data)
11        resp_iids, resp_values, _, _ = self._parse_get_response_data(
12        remaining_data, [iid])
13        return resp_values[resp_iids.index(iid)] if iid in resp_iids else
14        None

```

Listing 6: Requisição GET em LSNMPvSMonitor

Este trecho envia e processa uma requisição GET.

▪ MonitorApp (**monitorgui.py**):

- **Função:** Interface gráfica para exibir dados e configurar o dispositivo.
- **Fortes:** Visualização em tempo real com gráficos; interface intuitiva com Tkinter.
- **Limitações:** Suporta apenas dois sensores; não exibe mensagens de erro detalhadas para SET.
- **Código:**

```

1 def update_gui(self):
2     device_info = self.monitor.get_bulk(['device.id', 'device.type', 'device.
3     nSensors', 'device.upTime'])
4     self.device_id_label.config(text=f"ID: {device_info.get('device.id', 'N/A
5     ')}")
6     self.device_type_label.config(text=f"Tipo: {device_info.get('device.type
7     ', 'N/A')}")
8     sensor_info = self.monitor.get_bulk([f'sensors.{i}.sampleValue' for i in
9     range(1, 3)])
10    for i in range(1, 3):
11        value = sensor_info.get(f'sensors.{i}.sampleValue', 'N/A')
12        self.sensor_labels[i-1].config(text=f"Valor: {value}%")

```

Listing 7: Atualização da GUI em MonitorApp

Este trecho atualiza a GUI com dados do dispositivo e sensores.

0.2.2 Funções Críticas

- **encode_pdu e decode_pdu (agent.py, monitor.py):**

- **Função:** Codificam e decodificam PDUs, incluindo criptografia e HMAC.
- **Fortes:** Implementam segurança robusta com AES-256 e HMAC-SHA-256.
- **Limitações:** Complexidade na manipulação de dados binários; erro inicial de chave devido a BOM em arquivos.
- **Código:**

```

1 def encode_pdu(self, tag, msg_type, timestamp, msg_id, iid_list, v_list,
2   t_list, e_list):
3     core_pdu = b""
4     core_pdu += tag.encode('ascii') + struct.pack("!B", msg_type) + struct.
5     pack("!I", timestamp) + struct.pack("!I", msg_id)
6     core_pdu += b"".join(struct.pack("!H", iid) for iid in iid_list)
7     for value in v_list:
8         if isinstance(value, int):
9             core_pdu += struct.pack("!i", value)
10        elif isinstance(value, str):
11            core_pdu += struct.pack("!H", len(value.encode('utf-8'))) + value
12            .encode('utf-8')
13    key = self.derive_key(receiver_id)
14    cipher = Cipher(algorithms.AES(key), modes.CBC(os.urandom(16)), backend=
15    default_backend())

```

Listing 8: Codificação de PDU em encode_pdu

Este trecho codifica o núcleo do PDU e prepara a criptografia.

- **load_secrets (agent.py, monitor.py):**

- **Função:** Carrega e descriptografa arquivos de configuração.
- **Fortes:** Suporta criptografia AES-256-ECB; inclui fallback para chaves padrão.
- **Limitações:** Chave mestra hard-coded; erro de tamanho de chave (34 bytes) devido a codificação.
- **Código:**

```

1 def load_secrets(file_path):
2     if not os.path.exists(file_path):
3         print(f"Error: Secrets file {file_path} not found.")
4         return None
5     with open(file_path, 'rb') as f:
6         encrypted_data = f.read()
7         cipher = Cipher(algorithms.AES(MASTER_KEY), modes.ECB(), backend=
8         default_backend())
9         decryptor = cipher.decryptor()
10        decrypted_padded = decryptor.update(encrypted_data) + decryptor.finalize
11        ()
12        padding_len = decrypted_padded[-1]
13        decrypted = decrypted_padded[:-padding_len]
14        return json.loads(decrypted.decode('utf-8'))

```

Listing 9: Carregamento de segredos em load_secrets

Este trecho descriptografa arquivos de configuração com AES-256-ECB.

- **encrypt_secrets (encrypt_config.py):**

- **Função:** Criptografa arquivos de configuração.
- **Fortes:** Gera arquivos seguros para armazenar chaves.
- **Limitações:** Erro inicial de UnboundLocalError devido a falha na serialização JSON; corrigido com melhor tratamento de erros.
- **Código:**

```
1 def encrypt_secrets(data, output_file):
2     data_bytes = json.dumps(data, ensure_ascii=True).encode('utf-8')
3     padding_len = 16 - (len(data_bytes) % 16)
4     padded_data = data_bytes + bytes([padding_len] * padding_len)
5     cipher = Cipher(algorithms.AES(MASTER_KEY), modes.ECB(), backend=
6     default_backend())
7     encryptor = cipher.encryptor()
8     encrypted_data = encryptor.update(padded_data) + encryptor.finalize()
9     with open(output_file, 'wb') as f:
10         f.write(encrypted_data)
```

Listing 10: Criptografia em encrypt_secrets

Este trecho criptografa arquivos JSON, resolvendo o erro de chave inválida.

0.3 Etapas de Execução

Siga estes passos para executar o sistema completo.

0.3.1 Gerar Arquivos de Configuração

Primeiro, você deve criar os arquivos de segredos criptografados. Abra um terminal no diretório do projeto e execute o script `encrypt_config.py`.

```
1 python encrypt_config.py
```

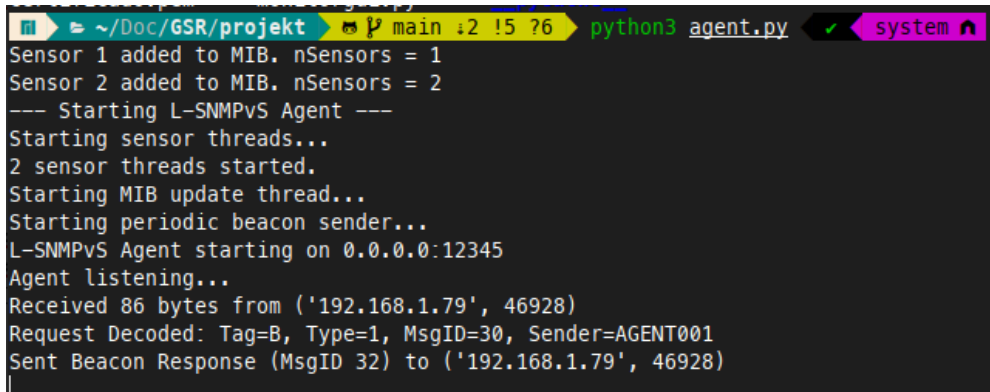
Isso criará dois arquivos: `agent_secrets.json` e `monitor_secrets.json`. Esses arquivos contêm os IDs e chaves compartilhadas necessários, criptografados com a `MASTER_KEY`.

0.3.2 Iniciar o Agente

Em um novo terminal, inicie o agente. Ele se vinculará à porta 12345 e começará a escutar requisições.

```
1 python agent.py
```

Você verá uma saída indicando que o agente e seus sensores foram iniciados.



```
~/Doc/GSR/projekt > python3 agent.py
Sensor 1 added to MIB. nSensors = 1
Sensor 2 added to MIB. nSensors = 2
--- Starting L-SNMPvS Agent ---
Starting sensor threads...
2 sensor threads started.
Starting MIB update thread...
Starting periodic beacon sender...
L-SNMPvS Agent starting on 0.0.0.0:12345
Agent listening...
Received 86 bytes from ('192.168.1.79', 46928)
Request Decoded: Tag=B, Type=1, MsgID=30, Sender=AGENT001
Sent Beacon Response (MsgID 32) to ('192.168.1.79', 46928)
```

Figura 1: Inicialização do agente com o script `agent.py`, mostrando a saída do terminal.

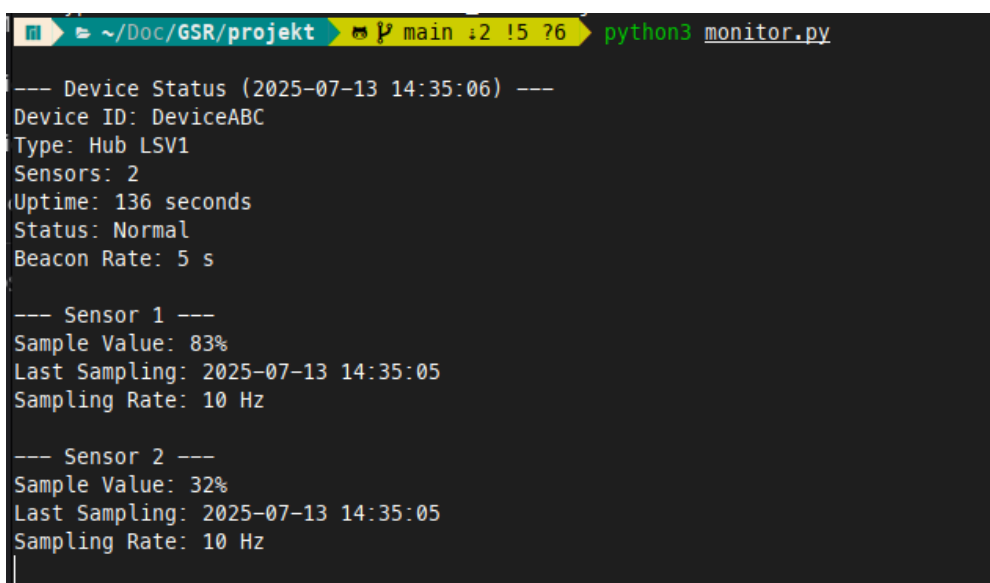
0.3.3 Executar um Monitor

Você pode interagir com o agente usando o monitor de linha de comando ou a GUI.

Monitor de Linha de Comando

Para ver uma atualização contínua do estado do agente, abra um terceiro terminal e execute:

```
1 python monitor.py
```



```
~/Doc/GSR/projekt > python3 monitor.py
--- Device Status (2025-07-13 14:35:06) ---
Device ID: DeviceABC
Type: Hub LSV1
Sensors: 2
Uptime: 136 seconds
Status: Normal
Beacon Rate: 5 s

--- Sensor 1 ---
Sample Value: 83%
Last Sampling: 2025-07-13 14:35:05
Sampling Rate: 10 Hz

--- Sensor 2 ---
Sample Value: 32%
Last Sampling: 2025-07-13 14:35:05
Sampling Rate: 10 Hz
```

Figura 2: Execução do monitor de linha de comando com `monitor.py`, exibindo o estado do agente.

Monitor GUI

Para uma visualização gráfica, execute o script `monitorgui.py`:

```
1 python monitorgui.py
```

A janela da GUI aparecerá e começará automaticamente a consultar o agente por dados, exibindo os valores e plotando o histórico dos sensores.



Figura 3: Interface gráfica do monitor (`monitorgui.py`) exibindo dados e gráficos dos sensores.

O usuário pode interagir com a interface gráfica para configurar valores, como a taxa de beacon, a taxa de amostragem e a opção de reinicializar o dispositivo.

0.4 Conclusão

O projeto L-SNMPvS alcançou seu objetivo de implementar um sistema de monitoramento seguro para dispositivos IoT, com comunicação UDP, interface gráfica e segurança robusta. A resolução de erros como o tamanho de chave inválido (272 bits) e o `UnboundLocalError` demonstrou a importância de validar codificação de arquivos e gerenciar chaves corretamente.

0.4.1 Melhorias Não Alcançadas

- **Suporte Dinâmico a Sensores:** A GUI e o monitor poderiam suportar um número variável de sensores com base em `device.nSensors`.
- **Processamento de Beacons:** O monitor não processa beacons, que poderiam ser exibidos na GUI para monitoramento passivo.
- **Gestão de Chaves Segura:** A chave mestra hard-coded deve ser substituída por um sistema de gerenciamento de chaves (e.g., variáveis de ambiente, AWS KMS).
- **Interface de Erros:** A GUI poderia exibir códigos de erro específicos (e.g., "Bad Value") para operações SET.
- **Log Centralizado:** Um sistema de logging em arquivo facilitaria debugging em produção.

0.5 Referências

- **Python Documentation:** <https://docs.python.org/3/> – Documentação oficial do Python, usada para `threading`, `socket` e `json`.
- **Cryptography Library:** <https://cryptography.io/en/latest/> – Documentação da biblioteca `cryptography` para AES e HMAC.
- **Tkinter:** <https://docs.python.org/3/library/tkinter.html> – Documentação para criação da GUI.
- **Matplotlib:** <https://matplotlib.org/stable/contents.html> – Documentação para gráficos de sensores.
- **SNMP Reference:** Harrington, D., Presuhn, R., Wijnen, B. (2002). *An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks*. RFC 3411, IETF.