Universidade do Minho
Departamento de Sistemas de Informática

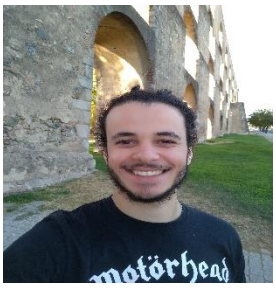
MIETI, UMINHO 2022/2023

Módulo de Gestão Redes

Relatório do Projeto de Grupo

Grupo:

Diogo Cerqueira
A93108



José Gomes
A93083

Índice

1. Introdução

No âmbito do módulo de Gestão de Redes foi proposta a realização de um trabalho prático, que tem como objetivo a implementação de um agente *proxy* SNMPv2c com recurso ao conhecimento que foi adquirido ao longo do semestre.

O presente relatório tem comos objetivos principais a apresentação do problema proposto, a formulação do mesmo, a descrição das decisões efetuadas pelo grupo, e uma reflecção crítica sobre os resultados obtidos.

De forma a realizar o trabalho com sucesso, anteriormente, foi importante e necessário relembrar alguns conceitos e postulados da linguagem de programação *Java*, visto que foi a linguagem adotada pelo grupo para a realização do presente projeto.

2. Requisitos

De forma a garantir o planeamento prévio e adequado da exequibilidade do presente trabalho torna-se essencial identificar claramente os seus requisitos, uma vez que os mesmos servirão de base ao trabalho executado posteriormente.

2.1. Requisitos funcionais

Para que o sistema trabalhe com o funcionamento desejado é necessário que cumpra os seguintes requisitos:

- Possuir um pacote *freeware* instalado com suporte ao protocolo SNMP *(Simple Network Management Protocol)* – o grupo decidiu usufruir do pacote "Net-SNMP".

- Possuir um módulo JDK *(Java Development Kit)* instalado na respetiva máquina (versão 19 ou superior) - o projeto foi construído em linguagem *Java,* logo, é necessário um módulo capaz de compilar o código implementado.

2.2. Requisitos não funcionais

Os próximos requisitos, que serão apresentados nos tópicos seguintes, são não funcionais o que significa que os mesmos não estão ligados diretamente às funcionalidades do sistema, mas sim relacionados com o tempo de execução/resposta e da fiabilidade. Os requisitos são definidos por:

- Código simples e otimizado, que proporcionará um menor tempo de resposta;
- Código bem comentado para facilitar a sua compreensão;
- Repartição do código por vários *packages* (diretorias), que representam uma determinada entidade do problema;

3. Lista de Figuras

4. Lista de Siglas e Acrónimos

SNMP – *Simple Manager Management Protocol*

JDK – *Java Development Kit*

AES - *Advanced Encryption Standard*

MIB – *Management Information Basis*

OID – *Object Identifier*

IP – *Internet Protocol*

IV - *Initialization Vector*

INMF - *Internet Network Management Framework*

5. Implementação

5.1 Linguagem de Programação

De forma a efetuar a realização do código do projeto, foi decidido a utilização da linguagem de programação *Java*, pois, além de ser a linguagem para a qual os membros do grupo estão confortáveis e entendem os conceitos vitais, graças a várias pesquisas que foram realizadas pelo grupo, foi possível descobrir que esta possui bibliotecas nativas, que facilitam a implementação da primitiva SNMPv2c.

As bibliotecas que foram utilizadas pelo grupo são as seguintes:

- ***Crypto*** – *package* exterior importada, que é crucial para a implementação da cifra AES *(Advanced Encryption Standard),* visto que cifra a informação que é comunicada entre o *Agent* e o *Manager.*

- ***Socket*** – classe necessária para o estabelecimento de um canal de comunicação entre o *Agent* e o *Manager*.

- ***Process –*** classe necessária para a execução das primitivas SNMP.

- ***LocalDateTime*** – classe utilizada para funcionalidades inerentes ao domínio do tempo.

- ***Objects*** – classe usada para facilitar a utilização dos objetos do programa.

5.2 MIB

A nossa MIB *(Management Information Basis)* é constituída pela tabela "OperTable" que é constituída por várias entradas.

O diagrama da figura 1 apresenta a estrutura da MIB implementada pelo grupo.

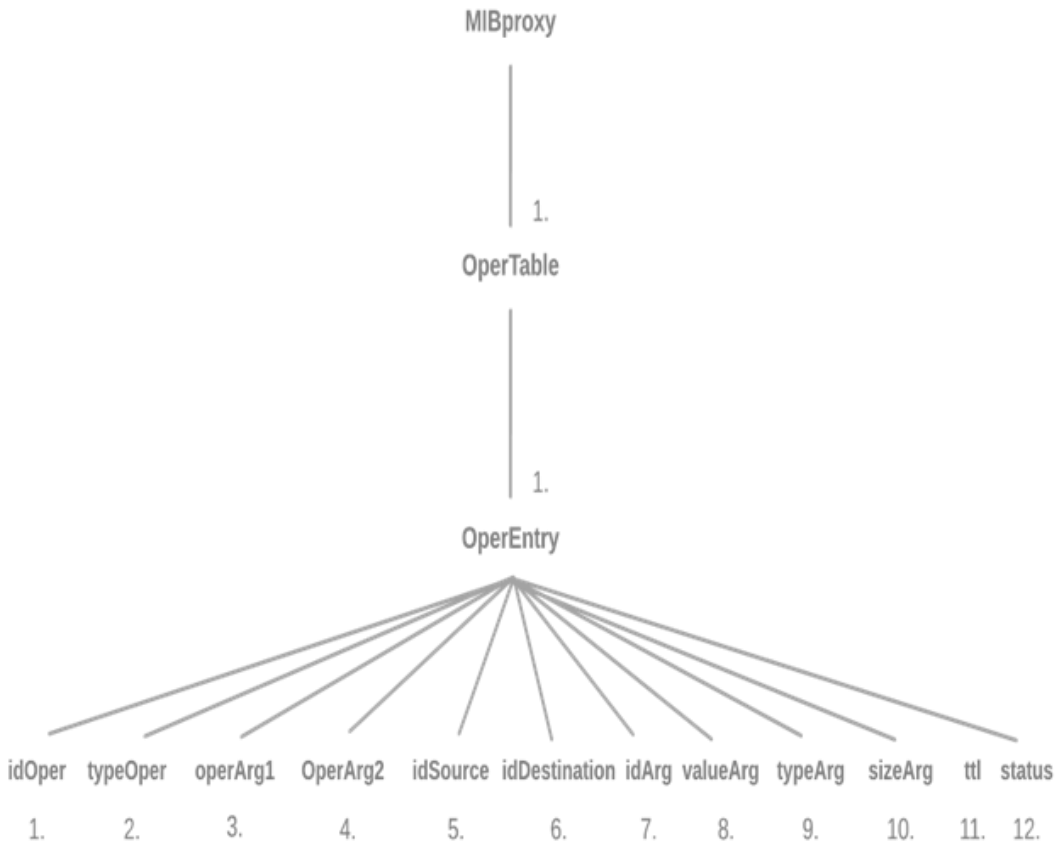


Figura 1 - Esquema que representa a MIB.

Após definida a estrutura acima referida, foi utilizado o software “MIB-Designer” de forma a gerar a MIB num ficheiro de texto (.txt).

```
MIB-PROXY DEFINITIONS ::= BEGIN

IMPORTS
    enterprises,
    MODULE-IDENTITY,
    OBJECT-TYPE,
    Opaque,
    NOTIFICATION-TYPE FROM SNMPv2-SMI
    DisplayString FROM SNMPv2-TC
    OBJECT-GROUP, NOTIFICATION-GROUP FROM SNMPv2-CONF;

mibProxyRegMIB MODULE-IDENTITY
    LAST-UPDATED "202211031042Z"    -- Nov 3, 2022, 10:42:00 AM
    ORGANIZATION "MIETI - Gestao de redes"
    CONTACT-INFO
        ""
    DESCRIPTION
        ""
    REVISION "202211031042Z"    -- Nov 3, 2022, 10:42:00 AM
    DESCRIPTION "Initial version."
    -- 1.3.6.1.4.1.1
    ::= { enterprises 1 }


operTable OBJECT-TYPE
    SYNTAX  SEQUENCE OF OperEntry
    MAX-ACCESS not-accessible
    STATUS  current
    DESCRIPTION "Tabela das operacoes SNMP"
    -- 1.3.6.1.4.1.1.1
    ::= { mibProxyRegMIB 1 }


operEntry OBJECT-TYPE
    SYNTAX  OperEntry
    MAX-ACCESS not-accessible
    STATUS  current
    DESCRIPTION
        "Entrada/linha da tabela com informacao relativa a respetiva operacao."
    INDEX { idOper }
    -- 1.3.6.1.4.1.1.1.1
    ::= { operTable 1 }

```

Figura 2 - Excerto da MIB no ficheiro .txt.

Objetos da MIB:

- **OperTable**: Estrutura de dados que armazena as entradas.
- **OperEntry:** Conjunto que possui os objetos.
- **idOper:** é o identificador da operação recebida pelo agente. É a chave da tabela.
- **typeOper:** mostra o tipo de operação SNMP.
- **operArg1:** 1º argumento de uma determinada operação a ser processada.
- **operArg2:** 2º argumento de uma determinada operação a ser processada.
- **idSource:** identificador da fonte do *request*.
- **idDestination:** código identificador do destino do *request* e onde a operação será executada.
- **idArg:** OID (*Object Identifier*) do objeto da MIB e cujo é o argumento da operação a ser executada pelo agente.
- **valueArg:** valor do objeto referido pelo "idArg" e cujo é o resultado recebido no agente *proxy* vindo do agente SNMP.
- **typeArg:** tipo de dados do "valueArg".
- **sizeArg:** tamanho em *bytes* do "valueArg".
- **ttl:** tempo de vida restante da entrada na tabela. Quando este objeto possuir o valor zero a entrada é retirada e o *Manager* não poderá aceder aos valores.
- **status:** é um número que corresponde ao estado do *request* (*accepted*, *created*, entre outros).

5.3 Árvore de diretorias

Para promover a organização e categorização do código desenvolvido no decorrer do projeto, numa fase inicial o grupo procedeu à construção de uma "árvore" de diretorias/*packages* para distribuir o código desenvolvido.

Cada diretoria da árvore representa um domínio do sistema, onde alberga as classes relativas a esse domínio.

A figura 3 demonstra a árvore de diretorias construída pelo grupo.

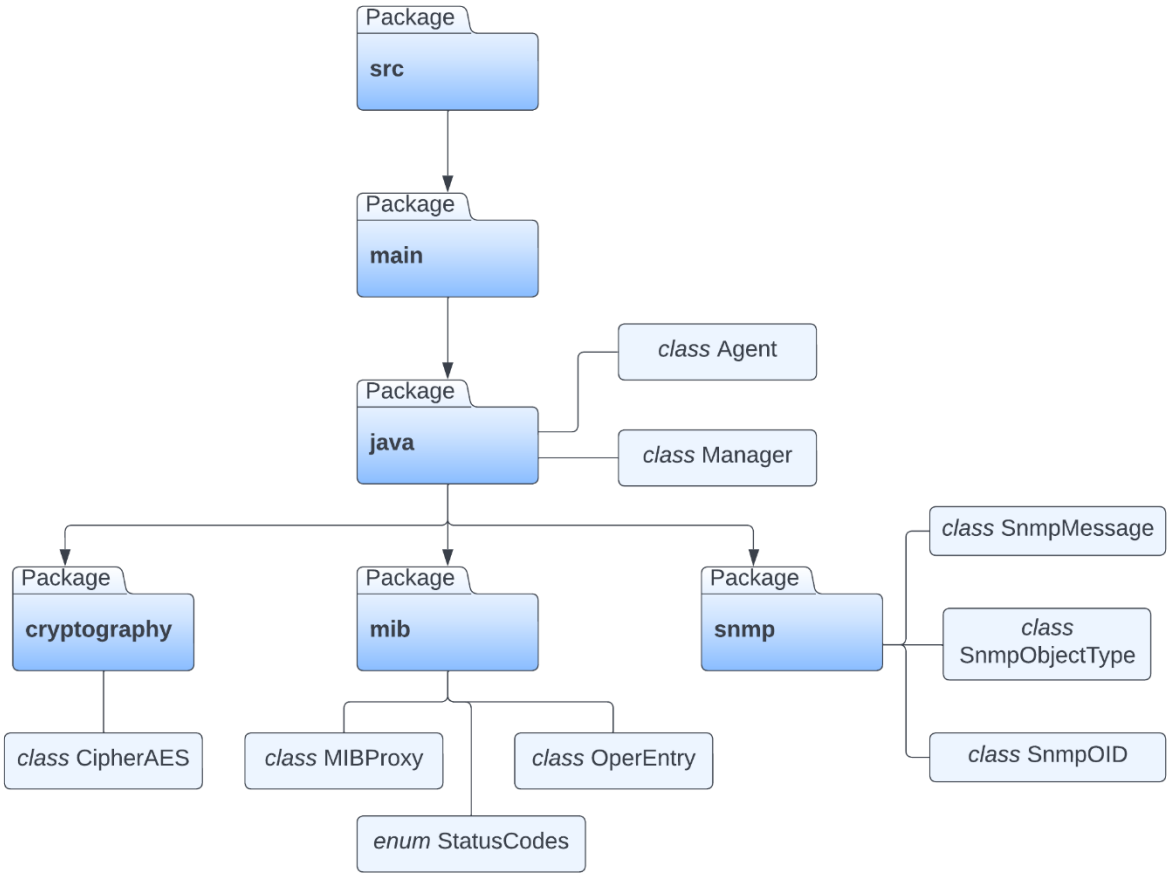


Figura 3 - Árvore de diretorias.

5.4 Classes

No total, o projeto desenvolvido pode ser repartido em nove classes:

5.4.1 Agent

A classe "Agent" é o paradigma do agente *proxy* que implementa a MIB especial de segurança (MIBproxy). Este desempenhará a função de inteface seguro para um agente SNMPv1 ou SNMPv2, normal e não seguro.

A comunicação, entre o agente *prox*y e o *Manager,* assenta no modelo de comunicação cliente-servidor. A classe "Agent" desempenha o papel de servidor, isto é, responde a eventuais pedidos efetuados pelo cliente (classe *Manager*). Posteriormente, a resposta é impressa na *bash.*

Visto que a classe "Agent" atua como servidor, esta é responsável pelo estabelecimento do canal de comunicação com o cliente. O canal de comunicação é definido com recurso à classe “ServerSocket”, da qual é passado como argumento a porta onde se estabelecerá a respetiva comunicação.

Enquanto o cliente não transmitir a mensagem “exit”, o agente *proxy* continuará a atender aos *requests* recebidos.

Após a receção de cada *request*, este passará pela cadeia de processos seguinte:

1. Decifragem do *request*.
2. Execução da primitiva SNMP, encapsulada pelo *request*.
3. Impressão do resultado da execução da primitiva.
4. Atualização do conteúdo da “MIBproxy”.
5. Impressão da “MIBproxy”.

Antes da atualização do conteúdo da “MIBproxy”, é necessário processar o *request*, ou seja, é preciso determinar a natureza da mensagem SNMP, analisar e extrair os parâmetros que compõe essa mensagem.

O método “parseSnmpCommand” da classe "Agent" acomoda os requisitos acima. Numa fase inicial é processada a primitiva SNMP, após o processamento são extraídos os argumentos referentes a essa primitiva e de seguida é feita uma associação entre o tipo de mensagem SNMP com o bloco de instruções associado.

A figura 4 apresenta o algoritmo do método “parseSnmpCommand”.

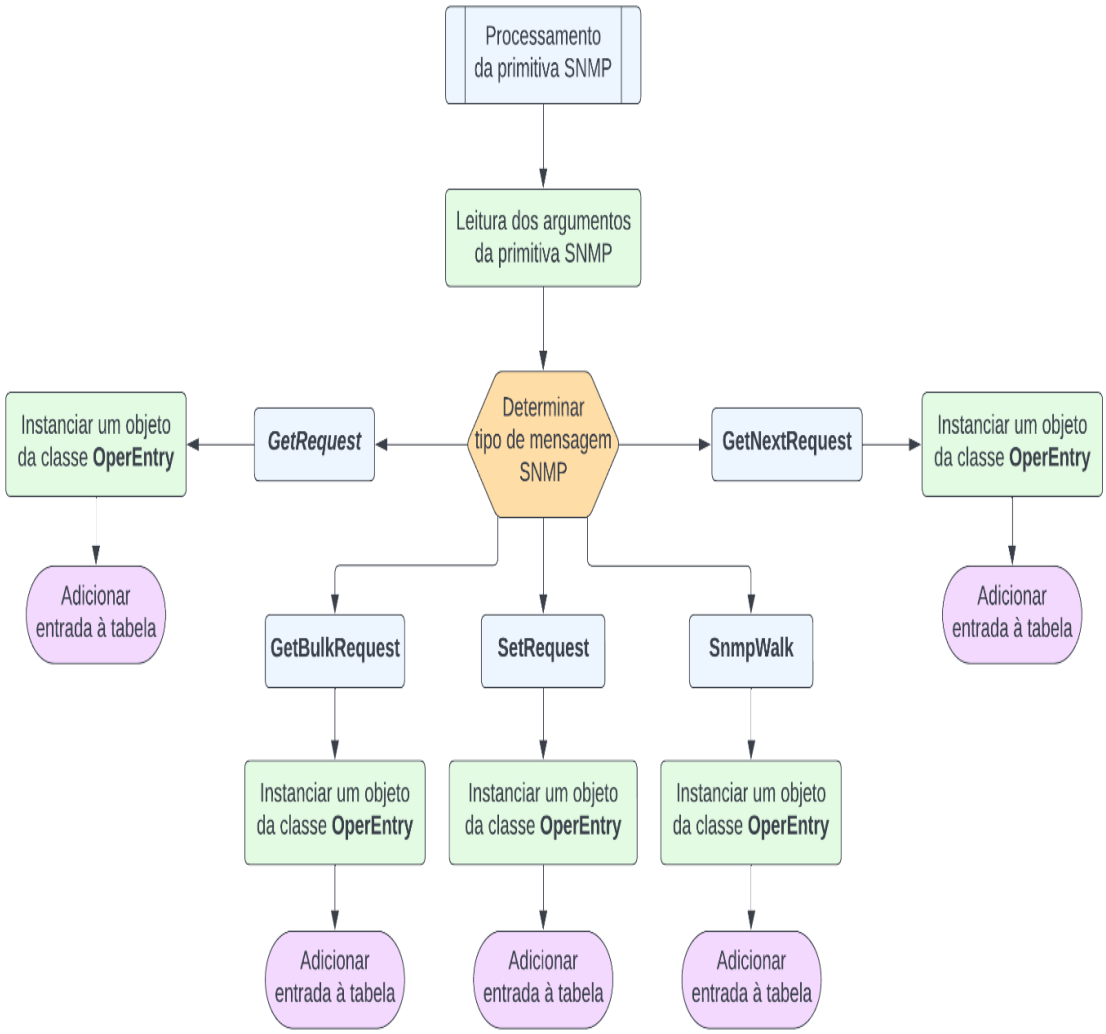


Figura 4 - Fluxograma do algoritmo do método "parseSnmpCommand".

Tal como foi citado no parágrafo anterior, dentro do método “parseSnmpCommand” é estabelecida uma associação entre o tipo de mensagem SNMP com o bloco de instruções apropriado.

A figura seguinte apresenta o bloco de instruções caraterístico da primitiva SNMP “GetRequest”.

```
final var snmpMessage :String = snmpCommandSplitted[0];
switch (snmpMessage) {
    case "snmpget" -> { // GetRequest
        return new OperEntry(
                new SnmpObjectType(idOper.getOID(), counter),
                new SnmpObjectType(typeOper.getOID(), GetRequest.getOperationType()),
                new SnmpObjectType(operArg1.getOID(), encrypt(firstArgument)),
                new SnmpObjectType(operArg2.getOID(), encrypt(secondArgument.toString())),
                new SnmpObjectType(idSource.getOID(), firstArgument),
                new SnmpObjectType(idDestination.getOID(), secondArgument.toString()),
                new SnmpObjectType(oidArg.getOID(), idDestination.getOID()),
                new SnmpObjectType(valueArg.getOID(), snmpCommandOutput),
                new SnmpObjectType(typeArg.getOID(), String.class),
                new SnmpObjectType(sizeArg.getOID(), snmpCommand.getBytes().length),
                new SnmpObjectType(ttl.getOID(), now()),
                new SnmpObjectType(status.getOID(), OK.getCode())
        );
    }
```

Figura 5 - Bloco de funções característico da primitiva GetRequest

5.4.2 Manager

Tal como foi referido na secção da classe “Agent”, a comunicação entre o agente *proxy* e o *Manager* assenta no modelo de comunicação cliente-servidor.

A classe “Manager” desempenha o papel de cliente, ou seja, é responsável por efetuar pedidos SNMP, em conjunto com a cifragem e envio dos mesmos.

O método “main” da classe “Manager” acomoda os requisitos acima. Numa fase inicial o *Manager* conecta-se ao endereço IP *(Internet Protocol)* e à porta do canal de comunicação, que foram estabelecidos pelo agente *proxy*. De seguida é lido o *input* (via teclado) do utilizador, e numa fase terminal esse *input* será cifrado e enviado ao agente *proxy*.

A figura 6 ilustra o algoritmo do método “main” da classe “Manager”.

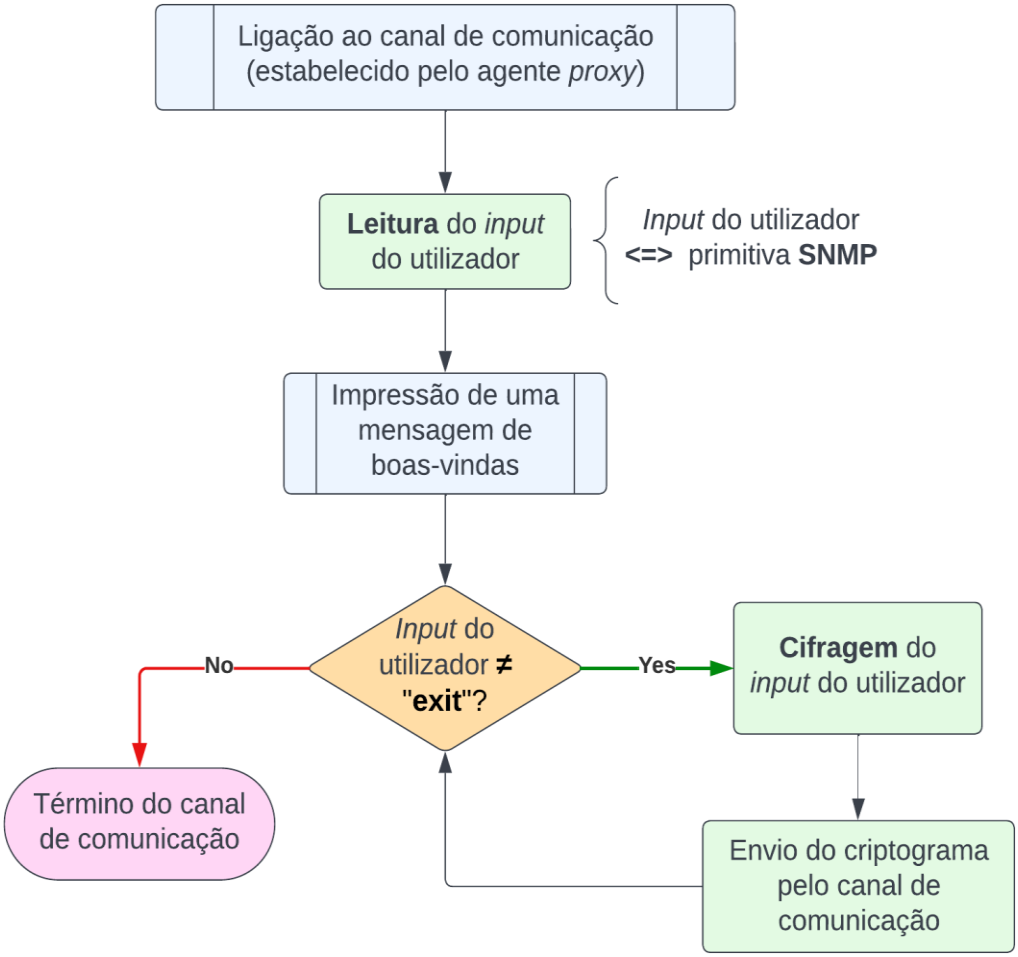


Figura 6 - Fluxograma do método main.

A figura 7 demonstra o envio da primitiva SNMP “GetRequest” e da mensagem especial “exit”, por parte do *Manager*. Na mesma figura também é visível a impressão do resultado dos *requests* recebidos, por parte do *Agent* (terminal da esquerda).

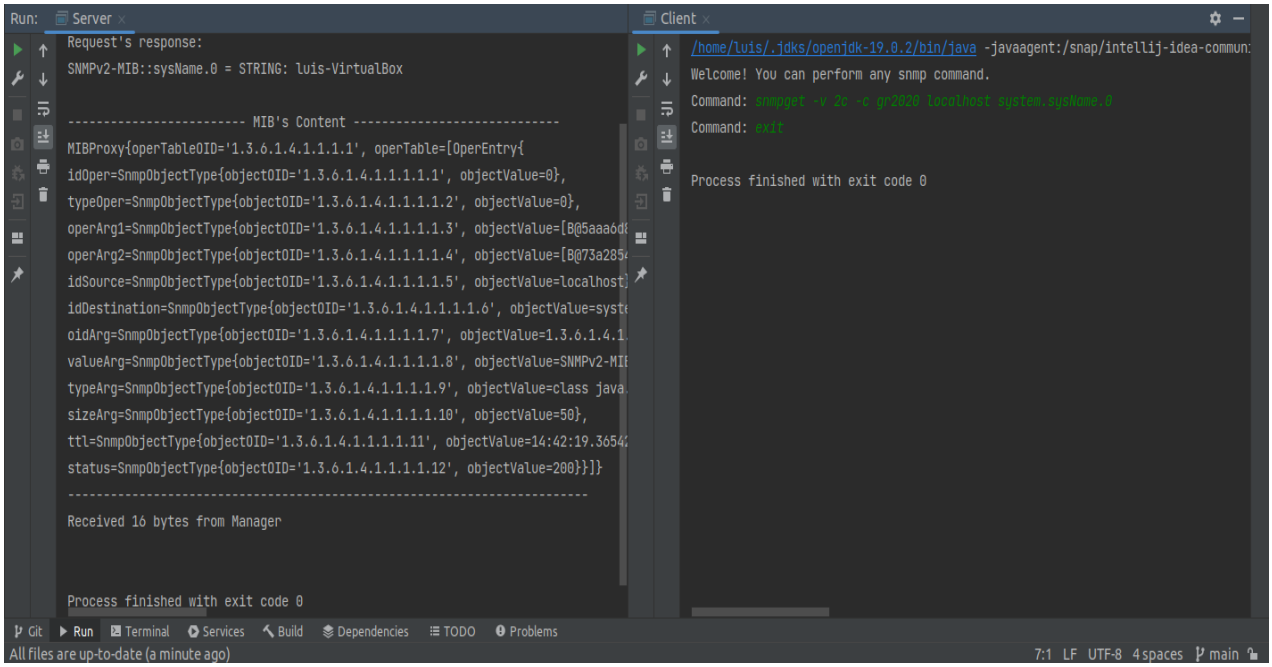


Figura 7 - Envio da primitiva SNMP “GetRequest” e da mensagem especial “exit”, por parte do Manager.

5.4.3 StatusCodes

O enumerado “StatusCodes” define os estados possíveis da qualidade da primitiva SNMP. Consoante a estrutura da primitiva SNMP, nomeadamente a ordem dos parâmetros e a fiabilidade dos mesmos, é atribuído um estado correspondente.

É de salientar que a classe “Agent” tira partido deste enumerado, em particular na instrução de adição de entradas à tabela “operTable”, onde o estado é passado como argumento da instância da classe “OperEntry”.

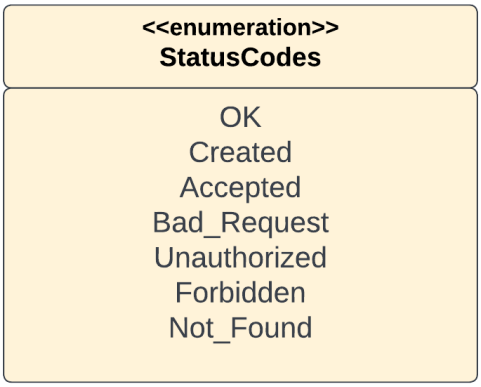


Figura 8 - Diagrama de Classe da StatusCodes.

5.4.4 OperEntry

A classe “OperEntry” representa o tipo de objeto que é armazenado pela tabela “operTable” da classe “MIBProxy”. Esta classe pode ser entendida como um modelo para uma determinada entrada na tabela referida, que contém informação relativa a cada operação SNMP.

A classe “OperEntry” é incorporada e importada em várias classes implementadas pelo grupo, como é o caso das classes “MIBProxy” e “Agent”.

A figura 9 apresenta a sequência de atributos e métodos relativos à classe “OperEntry”.

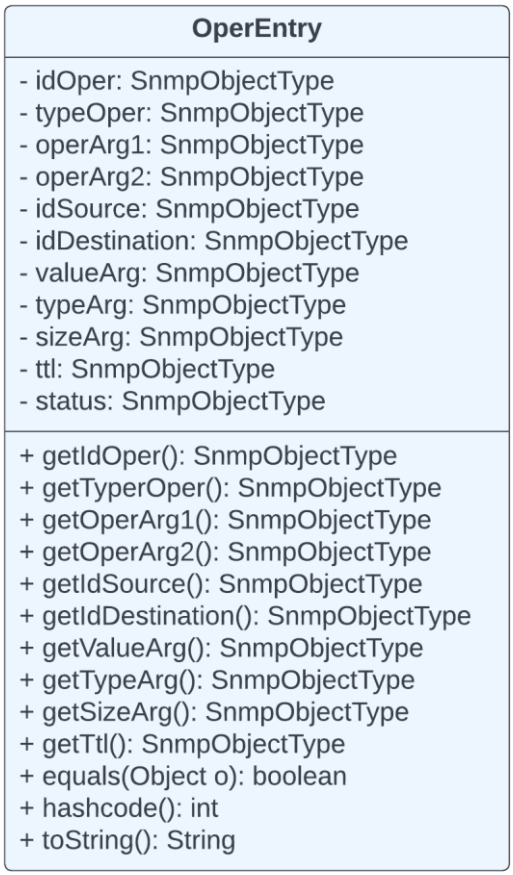


Figura 9 - Diagrama de Classe da OperEntry.

5.4.5 MIBProxy

A classe MIBProxy permite a definição de mecanismos de segurança indiretos que possibilitam uma manipulação fiável e segura dos objetos que são implementados nas MIBs normais dos agentes normais.

A figura 10 ilustra os atributos e métodos da classe MIBProxy.

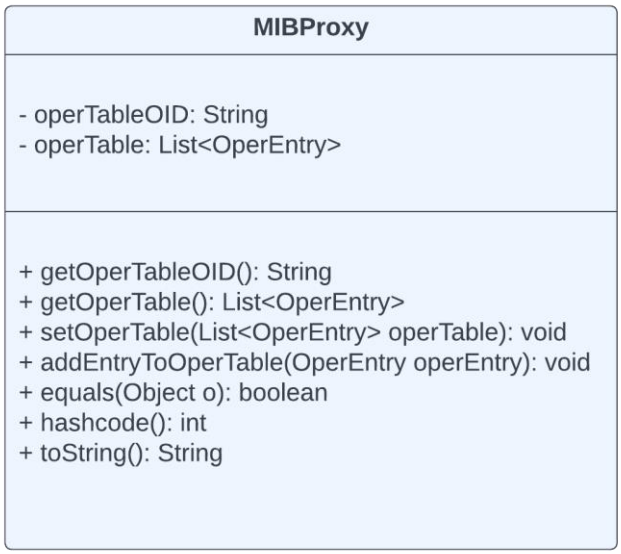


Figura 10 - Diagrama de Classe da MIBProxy.

Através da análise da figura 10 é possível observar que a classe referida é composta por dois atributos, que representam respetivamente, o OID da tabela operTable e a respetiva tabela operTable. Abaixo dos atributos descritos encontram-se listados os métodos da classe MIBProxy que permitem manipular o estado desses atributos.

Os atributos, que representam o estado da classe, descritos anteriormente, têm de ser privados de forma a respeitar os princípios e postulados da programação orientada a objetos, assim, só possível atualizar ou manipular os atributos da classe apenas com recurso aos métodos públicos definidos pelo grupo (representados pelo caracter '+').

O agente *proxy* (representado pela classe "Agent") é responsável pela inicialização e instanciação da classe MIBProxy.

A figura 11 ilustra a inicialização e instanciação da MIBProxy efetuada pelo agente *proxy,* que sucede no método *main* da classe “Agent”.

```
class Agent { // Snmp Server
    1 usage
    private static final int PORT = 5000;

    no usages   joseluisgomes +1
    public static void main(String[] args) throws Exception {
        final MIBProxy mibProxy = new MIBProxy();
```

Figura 11 - Inicialização e instanciação da MIBProxy efetuada pelo agente proxy.

O modificador *final* reforça a estabilidade e segurança da MIBproxy pois inibe qualquer entidade externa ao sistema, que altere ou atualize a referência ao respetivo objeto.

5.4.6 CipherAES

A classe “CipherAES” proporciona a codificação dos objetos da MIBproxy, bem como a cifragem da informação transmitida entre o cliente e o servidor, garantindo a anonimidade e confidencialidade da mesma.

A cifra AES foi implementada com auxílio à API externa “javax.crypto”. A cifra AES é uma variante da cifra por blocos desenhada por Rijmen-Daemen, e é caraterizada por um universo de operações de *XOR* e *lookups* a tabelas.

A figura 12 reflete o esquema da classe “CipherAES”.

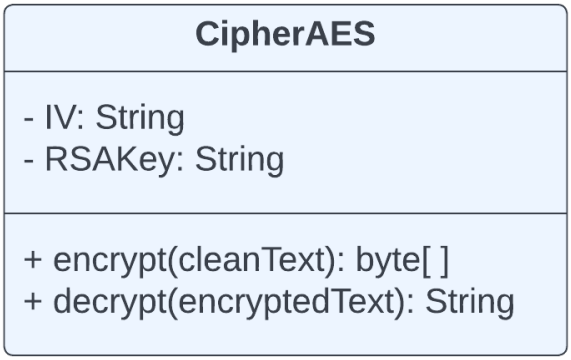


Figura 12 - Diagrama de Classe da CipherAES.

Tal como a figura sugere, a classe encontra-se munida com métodos criptográficos usados respetivamente para encriptar a informação que circula no canal de comunicação, entre o Manager e o Agent, e para desencriptar a mesma.

Os atributos da classe têm de ser obrigatoriamente privados de forma a respeitar os dogmas da programação orientada a objetos, e devido ao facto desses atributos serem órgãos vitais da cifra descrita.

A figura 13 contém os dois métodos responsáveis respetivamente pelas operações de cifra e decifragem. Foi atribuído o modificador *static* para permitir que os métodos referidos possam ser importados em qualquer parte do código do sistema.

```
    public static byte[] encrypt(String cleanText) throws Exception {
        Cipher cipher = Cipher.getInstance( transformation: "AES/CBC/PKCS5Padding",  provider: "SunJCE");
        SecretKeySpec key = new SecretKeySpec(RSAKey.getBytes(UTF_8),  algorithm: "AES");

        cipher.init(ENCRYPT_MODE, key, new IvParameterSpec(IV.getBytes(UTF_8)));
        return cipher.doFinal(cleanText.getBytes(UTF_8));
    }

    1 usage
    public static String decrypt(byte[] encryptedText) throws Exception{
        Cipher cipher = Cipher.getInstance( transformation: "AES/CBC/PKCS5Padding",  provider: "SunJCE");
        SecretKeySpec key = new SecretKeySpec(RSAKey.getBytes(UTF_8),  algorithm: "AES");

        cipher.init(DECRYPT_MODE, key, new IvParameterSpec(IV.getBytes(UTF_8)));
        return new String(cipher.doFinal(encryptedText), UTF_8);
    }
}
```

Figura 13 - Funções de encriptação e desencriptação.

5.4.7 SnmpMessage

O enumerado “SnmpMessage” estabelece as primitivas SNMP suportadas pela aplicação, assim, um determinado *manager* só poderá introduzir uma primitiva que se enquadre nos estados possíveis deste enumerado.

A figura 14 demonstra a estrutura do enumerado “SnmpMessage”.

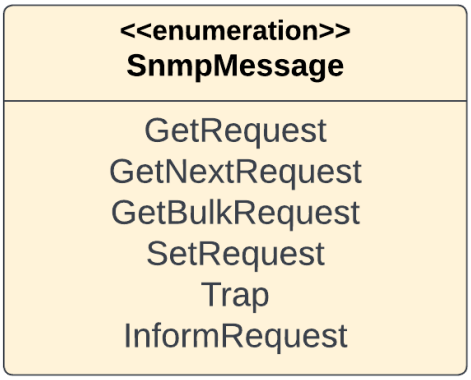


Figura 14 - Diagrama da Classe de SnmpMessage.

Dentro do enumerado referido, cada primitiva SNMP encontra-se associada a um número inteiro, que pode ser interpretado como um código identificador dessa primitiva.

A classe “Agent” tira proveito deste enumerado (dentro do método “parseSnmpCommand”), nomeadamente como argumento da instância da classe “OperEntry”.

5.4.8 SnmpOID

O enumerado “SnmpOID” define os OIDs dos objectos que constituem o objecto “OperEntry”. Em semelhança com o enumerado “SnmpMessage”, a classe “Agent” tira proveito deste enumerado (dentro do método “parseSnmpCommand”), nomeadamente na operação de processamento dos *requests,* em especial com a invocação do método “getOID()”.

A figura 15 demonstra a estrutura do enumerado “SnmpOID”.

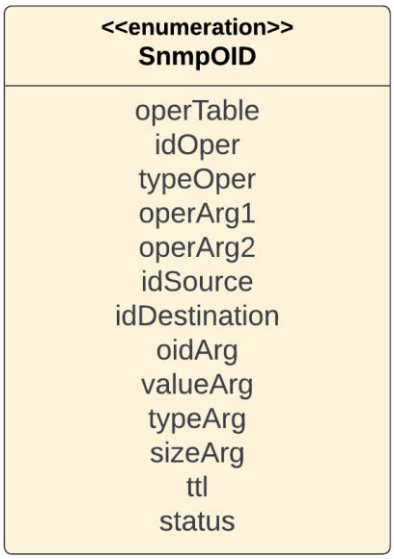


Figura 15 - Diagrama de Classe do SnmpOID.

Dentro do enumerado referido, cada objeto encontra-se associado ao OID respetivo (de acordo com a estrutura da MIB).

5.4.9 SnmpObjectType

A classe “SnmpObjectType” representa o paradigma de um objeto SNMP. Esta é composta pelo OID do objeto e pelo tipo de objeto.

A

figura 16 demonstra a estrutura da classe “SnmpObjectType”.

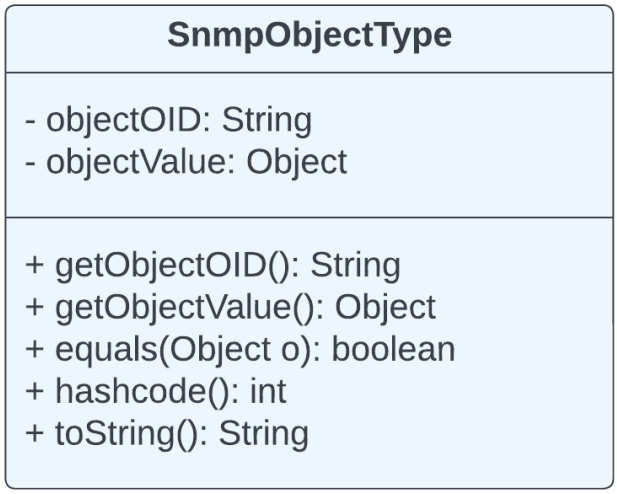


Figura 16 - Diagrama da Classe SnmpObjectType.

5.5 Canal de Comunicação

A comunicação entre o agente *proxy* e o *Manager* realiza-se através de Sockets, com base no modelo de cliente-servidor. No contexto desse modelo, é criada uma *socket* do lado do servidor (classe “Agent”) que espera que uma *socket* do lado do cliente (classe “Manager”) se conecte a esta. A porta usada é a 5000, mas pode ser alterada, desde que a escolhida esteja livre, ou seja, não esteja a ser usada por outro processo.

É garantida a anonimidade e confidencialidade do tráfego, pois os dados são encriptados através da utilização de técnicas criptográficas, mais concretamente, a cifra AES. Esta é uma cifra simétrica (usa a mesma chave para encriptar e desencriptar o tráfego) e possui um vetor aleatório, denominado IV (*Initialization Vector*), que traz aleatoriedade ao criptograma. Decidiu-se escolher esta cifra pois é uma cifra considerada inquebrável e que não traz uma sobrecarga computacional elevada ao processo.

6. Teste, Resultados e Discussão

Os testes seguintes foram realizados no dia 30 de Dezembro de 2022.

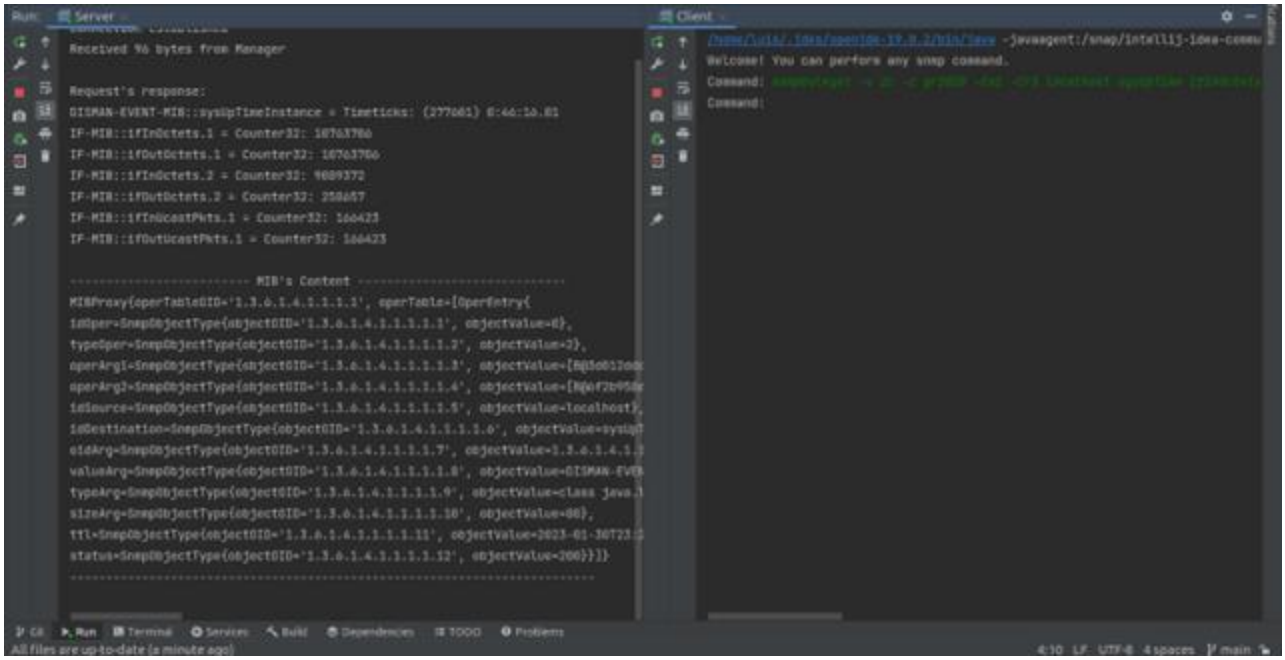


Figura 17 - Resposta à primitiva "snmpbulkget".

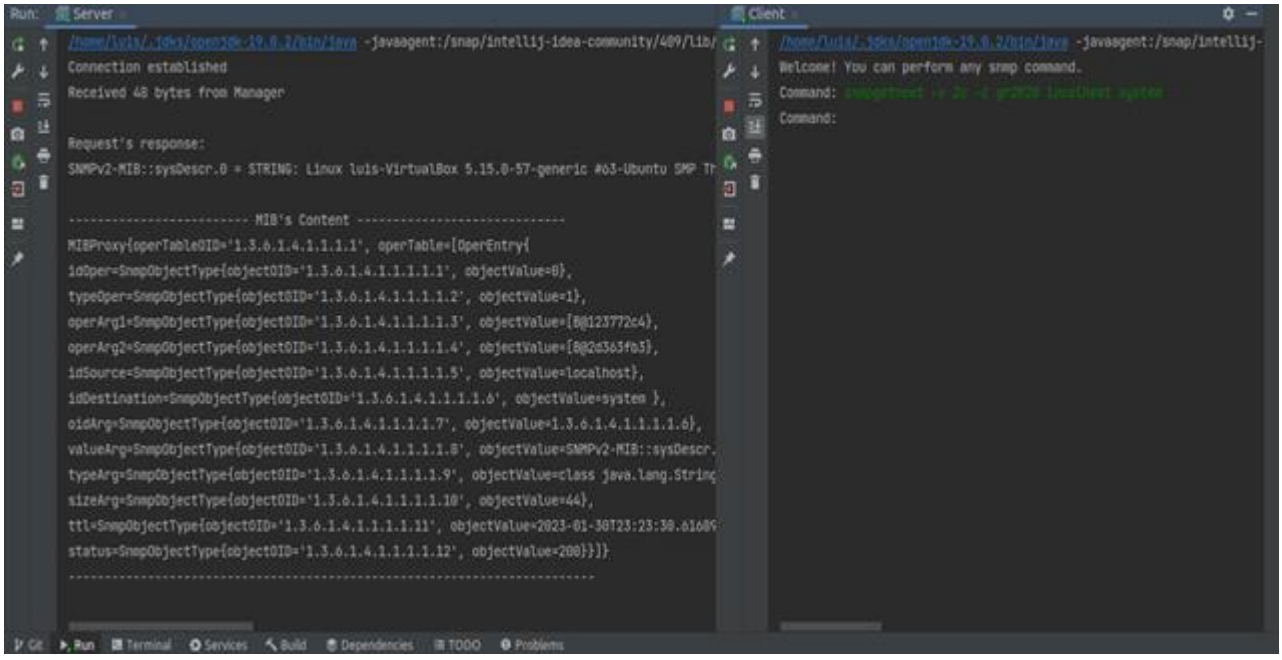


Figura 18 - Resposta à primitiva "snmpgetnext"

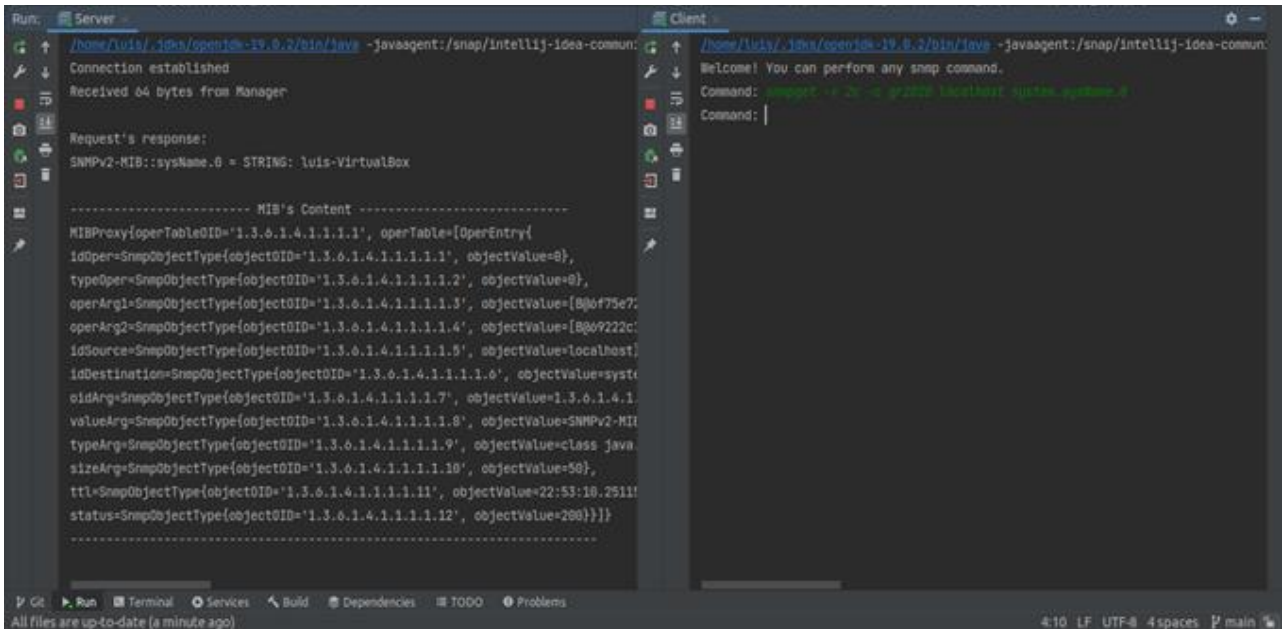


Figura 19 - Resposta à primitiva "snmpget".

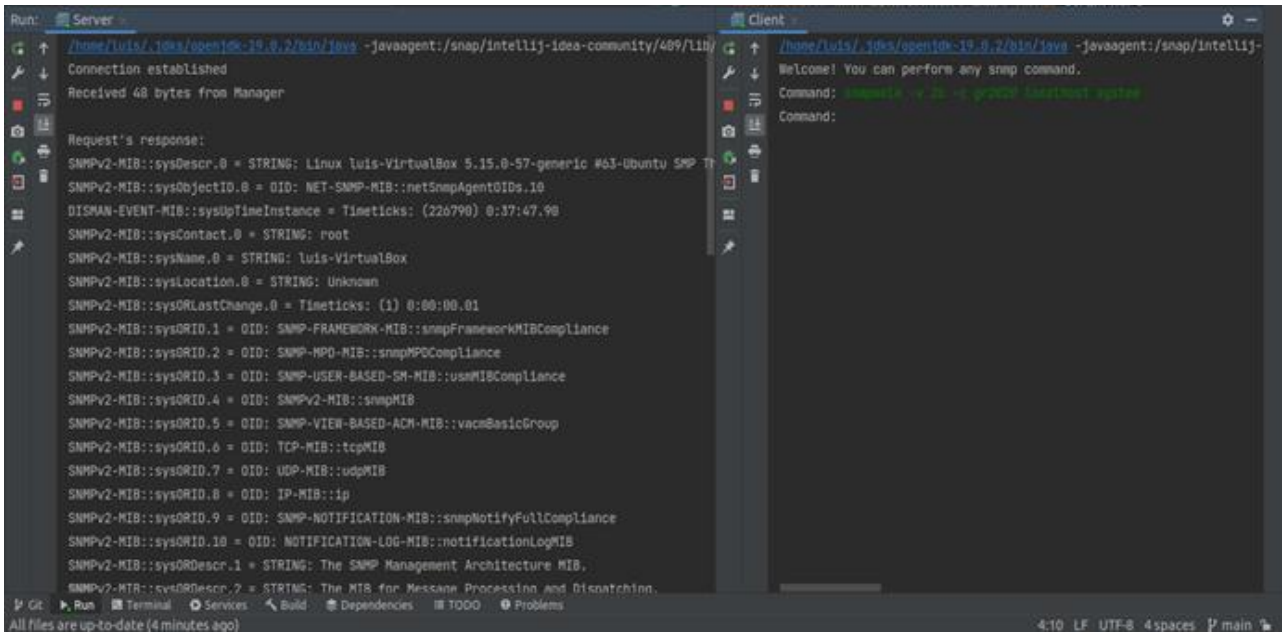


Figura 20 - Resposta à primitiva "snmpwalk

6.1 Discussão de Resultados

O grupo realizou vários testes sobre as primitivas SNMP, tal como é evidenciado pelo conjunto de figuras da secção anterior.

Os testes foram realizados com recurso à documentação do pacote "net-snmp". Através da documentação do pacote referido, foi possível explorar a flexibilidade das primitivas SNMP, nomeadamente na seleção dos parâmetros pretendidos e dos OIDs sujeitos a esses comandos.

Relativamente ao estudo e exploração dos comandos desse pacote, graças às figuras da secção anterior, é visível que o parâmetro "-c" *(community string)* e a versão do protocolo SNMP utilizado, destacado pelo parâmetro "-v" (*SNMP version number*), são respetivamente "gr2020" e "2c". O parâmetro *Agent* é assumido pelo *localhost* e os OIDS explorados são diversificados, como é o caso dos OIDs "system" e "system.sysName.0".

7. Conclusão

Concluído este relatório, é inegável o carácter didático do mesmo.

A realização deste projeto permitiu ao grupo consolidar o conhecimento relativo aos protocolos, mecanismos e filosofias da arquitetura de gestão do INMF e dos conceitos principais sobre ameaças de segurança em aplicações e sobre as estratégias que permitem implementar uma maior segurança ao sistema.

Também nos permitiu aumentar profundamente o conhecimento sobre o protocolo SNMP, através da observação de como este funciona na prática. É de realçar que a execução de testes experimentais, foi fundamental para uma análise crítica dos resultados obtidos.

O grupo acabou por ter algumas dificuldades na encriptação do canal de comunicação, mais em concreto na transformação de byte para texto limpo, mas após um maior estudo sobre este tema foi possível ultrapassar esta dificuldade.

8. Autoavaliação

- Diogo Cerqueira

Neste projeto estive envolvido no desenvolvimento do código e no desenvolvimento do relatório. A maior dificuldade que encontrei foi na utilização conjunta da receção de dados através da socket com a função de desencriptação do tráfego.

- José Gomes

Neste projeto estive envolvido no desenvolvimento do código e no desenvolvimento do relatório. A maior dificuldade que encontrei foi na implementação do código responsável pelo processamento dos *requests* recebidos, por parte do agente *proxy*.

9. Referências

[1] ISO 9595: “Information Processing Systems - Open Systems Interconnection - Common Management Information Service Definition”, Geneva, 1990

[2] Milham D.J., Willetts K.J.: “BT’s Communications Management Architecture”, NorthHolland, 1989

[3] José Bacelar, (2022), “Cifra por Blocos”, [Acedido em 28 de Dezembro de 2022]

[4] All Classes (JavaSE11 &JDK11). (2018, 25 de setembro). https://docs.oracle.com/en/java/javase/11/docs/api/allclasses.html [Acedido em 22 de Dezembro de 2022]

[5] Manage Engine (What is SNMP) https://www.manageengine.com/network-monitoring/what-is-snmp.html (Acedido em 23 de Dezembro de 2022)

[6] NET-SNMP http://www.net-snmp.org/ [Acedido em 19 de Dezembro de 2023]

[7] Man Y. Rhee, Internet Security – Cryptographic Principles, Algorithms and Protocols, Wiley, 2003.

[8] Charles P. Pfleeger, Shari Lawrence Pfleeger, Jonathan Margulies, Security in Computing, Prentice Hall, 2015.

[9] William Stallings, Cryptography and Network Security Principles and Practices, Prentice Hall, 2015.

[10] M. Rose, The Simple Book, Second Edition, Prentice Hall, 1996.

[11] B. Dias, Gestão de Redes, PAPCC, Universidade do Minho, 1996.