Introdução

* O relatório descreve o desenvolvimento de uma aplicação destinada à monitorização e análise de redes de computadores. O sistema foi concebido para identificar dispositivos conectados, detetar colisões, capturar pacotes de dados e oferecer uma visualização da topologia da rede em ambiente gráfico e inclui agora recursos aprimorados como autenticação de usuários, criptografia de dados sensíveis e registros de acessos. Além disso, a aplicação armazena informações sobre dispositivos e eventos numa base de dados, permitindo consultas e auditorias futuras.

Relatório Técnico de Avaliação de Software/Hardware

1. Informações Gerais do Projeto

- Nome do Projeto: Monitoramento do Tráfego de Rede

- Tipo de Projeto: Software: Aplicação

- Data de Início:

- Data de Término:

- Equipe de Desenvolvimento: Ander Bogalho

- Responsável pelo Relatório: Ander Bogalho

- Data do Relatório: 22/12/2024

---

2. Resumo Executivo

O objetivo deste projeto é replicar funcionalidades de sistemas de monitorização de redes empresariais, como o Zabbix, utilizando tecnologias personalizadas para a deteção e resolução de problemas em redes. Este projeto foi desenvolvido como uma solução educacional para monitorizar a rede do laboratório da escola, simulando um ambiente profissional.

**Funcionalidades Desenvolvidas**

As principais funcionalidades da aplicação foram implementadas com os seguintes métodos e bibliotecas:

* **Monitorização de Dispositivos na Rede**:  
  A funcionalidade de scan de dispositivos foi implementada utilizando a biblioteca **Scapy**. Esta função envia pacotes ARP para identificar dispositivos na rede, recolhendo informações como IP, hostname e endereço MAC. O código correspondente encontra-se detalhado no ficheiro network\_scan.py (ver Anexo 6.1).
* **Captura e Análise de Pacotes**:  
  Para capturar pacotes em tempo real e analisar os seus conteúdos, foi utilizada a função sniff da biblioteca **Scapy**. Os dados capturados são processados para identificar protocolos, anomalias e colisões na rede. O código está disponível no ficheiro packet\_capture.py (ver Anexo 6.2).
* **Deteção de Colisões**:  
  A análise de colisões baseia-se na identificação de pacotes SYN sem resposta ou pacotes TCP resetados. Esta funcionalidade foi implementada utilizando uma função personalizada para processar os pacotes capturados. Detalhes desta implementação estão no ficheiro collision\_detection.py (ver Anexo 6.3).
* **Armazenamento de Dados**:  
  Os dados de dispositivos e colisões são armazenados num banco de dados SQLite. Esta funcionalidade garante persistência de informações e suporte para consultas futuras. O modelo das tabelas e os métodos de inserção podem ser consultados no ficheiro database.py (ver Anexo 6.4).
* **Visualização da Topologia da Rede**:  
  A aplicação utiliza a biblioteca **NetworkX** para criar grafos que representam a topologia da rede. Estes grafos são renderizados com **Matplotlib**, permitindo uma visualização gráfica intuitiva. O código encontra-se no ficheiro topology\_visualization.py (ver Anexo 6.5).
* **Interface Gráfica**:  
  Uma interface amigável foi desenvolvida com **Tkinter** para facilitar a interação do utilizador. As funcionalidades incluem iniciar monitorizações, capturar pacotes e exibir resultados. A implementação desta funcionalidade está disponível no ficheiro gui.py (ver Anexo 6.6).
* **Criptografia de Dados**: A biblioteca Cryptography é utilizada para proteger dados sensíveis armazenados no banco de dados. Informações como endereços IP, endereços MAC e nomes de dispositivos são criptografados antes de serem gravados no banco de dados, garantindo a confidencialidade em casos de acesso não autorizado. A implementação também inclui a geração de chaves de criptografia exclusivas e o armazenamento seguro dessas chaves para evitar o comprometimento dos dados. (Ver Anexo 6.7 e 6.9)
* **Segurança e Autenticação**: Foi implementada autenticação de usuários com armazenamento de senhas criptografadas e logs de ações. Cada usuário tem uma conta personalizada, com senhas protegidas por algoritmos de hash robustos, garantindo que informações não sejam comprometidas em caso de violação de dados. Além disso, logs detalhados registram eventos como tentativas de login, registros e ações importantes realizadas pelo usuário, proporcionando um histórico completo para auditorias. (Ver Anexo 6.7 e 6.8)

**Resumo das Funcionalidades**

A aplicação desenvolvida apresenta as seguintes funcionalidades principais:

* **Scan de Rede**: Identifica dispositivos conectados à rede, exibindo IP, hostname, endereço MAC e informações sobre latência.
* **Deteção de Colisões**: Monitoriza pacotes de dados para identificar colisões na rede e sugere soluções para os problemas detetados.
* **Captura de Pacotes**: Analisa pacotes de dados em tempo real, identificando protocolos e possíveis anomalias.
* **Base de Dados SQLite**: Armazena informações sobre dispositivos e colisões detetadas para consulta futura.
* **Visualização da Topologia**: Gera um grafo que representa a topologia da rede, destacando a conexão entre dispositivos e o router.
* **Criptografia de Dados**: Protege dados sensíveis como IPs e MACs no banco de dados, garantindo confidencialidade.
* **Segurança e Autenticação**:Oferece autenticação de usuários com senhas protegidas por algoritmos de hash robustos e registro de logs detalhados de ações como login, registro e atividades importantes
* **Interface Gráfica**: Oferece uma interface intuitiva, criada com Tkinter, para facilitar a interação do utilizador com o sistema.

3. Metodologia de Desenvolvimento

* Foi utilizado o modelo **Ágil** para o desenvolvimento do projeto. As tarefas foram divididas em iterações curtas, permitindo a implementação incremental das funcionalidades e a realização de ajustes baseados em feedback.

Ferramentas e Tecnologias Utilizadas:

* **Ambientes de Desenvolvimento**: Visual Studio Code e PyCharm (Linux);
* **Linguagem de Programação**: Python;
* **Bibliotecas**:
* Scapy: Para monitoramento de rede e captura de pacotes;
* Tkinter: Para criação de interface gráfica;
* SQLite3: Para gerenciamento do banco de dados local;
* NetworkX: Para modelagem e visualização da topologia de rede;
* Cryptography: Para encryptar dados sensíveis;
* Matplotlib: Para renderização gráfica da topologia de rede.

4. Testes Realizados

* **Tipos de Testes**: Foram realizados testes de desempenho em máquinas Linux com menor capacidade de processamento e testes funcionais para verificar se a base de dados guardava os registos ordenados nas tabelas corretamente.

**Listagem dos tipos de testes aplicados**:

* Testes de Integração;
* Testes Funcionais;
* Testes de Stress.

5. Conclusão e Recomendações

**Conclusão Geral**:  
O projeto foi avaliado com base nos critérios de funcionalidade, segurança, desempenho, integração e documentação. De forma geral, o projeto evoluiu significativamente, com melhorias em segurança e experiência do usuário. As funcionalidades originais foram mantidas e aprimoradas.

**Principais Pontos Fortes**:  
Os aspectos mais positivos do projeto incluem:

* A funcionalidade do sistema na captação de dispositivos;
* A deteção de colisões ou pacotes resetados na rede, cumprindo os critérios estabelecidos.
* Segurança aprimorada através da autenticação e criptografia.

**Principais Pontos de Melhoria**:  
Foram identificadas as seguintes áreas para aprimoramento:

* Melhorias na interface gráfica;
* Aprimoramento na deteção de um maior número de tipos de pacotes na rede.

**Recomendações**:  
Para futuras fases do projeto ou atualizações, são sugeridas as seguintes ações:

* Implementar um sistema de segurança robusto para proteger a aplicação;
* Substituir a biblioteca SQLite3 por uma base de dados mais escalável e eficiente;
* Melhorar a interface gráfica para maior usabilidade e estética;
* Otimizar o desempenho da aplicação em máquinas com capacidades reduzidas.

6. Anexos

**Anexo 6.1: Monitorização de Dispositivos na Rede**

Código principal que implementa o scan de dispositivos, utilizando pacotes ARP.

def scan\_network(ip\_range):

"""Escaneia a rede e retorna uma lista de dispositivos encontrados com IPs e nomes de máquina."""

arp\_request = scapy.ARP(pdst=ip\_range)

broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")

arp\_request\_broadcast = broadcast / arp\_request

answered\_list = scapy.srp(arp\_request\_broadcast, timeout=1, verbose=False)[0]

devices = []

for element in answered\_list:

device\_info = {

'ip': element[1].psrc,

'mac': element[1].hwsrc,

'hostname': get\_hostname(element[1].psrc)

}

devices.append(device\_info)

return devices

**Anexo 6.2: Captura e Análise de Pacotes**

Código para captura de pacotes com a função sniff e detecção de colisões.

def packet\_callback(packet):

"""Captura pacotes e analisa colisões."""

if packet.haslayer(IP):

src\_ip = packet[IP].src

dst\_ip = packet[IP].dst

protocol = "TCP" if packet.haslayer(TCP) else "UDP" if packet.haslayer(UDP) else "Outro"

details = ""

# Detectar colisões específicas

if packet.haslayer(TCP):

flags = packet[TCP].flags

if "R" in flags: # Pacote resetado

details = "Pacote resetado"

elif "S" in flags and not packet.ack: # SYN sem ACK

details = "SYN sem resposta"

if details:

save\_collision\_to\_db(src\_ip, dst\_ip, protocol, details)

**Anexo 6.3: Deteção de Colisões**

Análise de status dos dispositivos e sugestão de soluções para problemas identificados.

def analyze\_device\_status(device):

"""Analisa o status do dispositivo e retorna um texto de status."""

ip = device['ip']

is\_reachable, ping\_output = check\_ping(ip)

if is\_reachable:

avg\_latency = None

for line in ping\_output.splitlines():

if "Média" in line:

avg\_latency = float(line.split('=')[1].replace('ms', '').strip())

if avg\_latency and avg\_latency > 100:

return f"{ip} está acessível, mas com latência alta ({avg\_latency} ms).", 'high\_latency'

else:

return f"{ip} está acessível. Latência média: {avg\_latency} ms.", 'normal'

else:

return f"{ip} não está acessível.", 'device\_unreachable'

**Anexo 6.4: Gestão e Armazenamento de Dados**

Criação, leitura e manipulação de tabelas no banco de dados SQLite.

def initialize\_database():

"""Inicializa o banco de dados SQLite e cria as tabelas se não existirem."""

connection = sqlite3.connect("network\_devices.db")

cursor = connection.cursor()

cursor.execute("""

CREATE TABLE IF NOT EXISTS devices (

id INTEGER PRIMARY KEY AUTOINCREMENT,

ip TEXT NOT NULL UNIQUE,

mac TEXT NOT NULL,

hostname TEXT NOT NULL,

last\_seen TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

)

""")

connection.commit()

connection.close()

**Anexo 6.5: Visualização da Topologia da Rede**

Criação e visualização de grafos da topologia da rede.

def create\_network\_topology(devices):

"""Cria o grafo de topologia da rede."""

G = nx.Graph()

G.add\_node("Router", label="Router", color='red')

for device in devices:

device\_name = f"{device['ip']} ({device['mac']})"

G.add\_node(device\_name, label=device\_name, color='skyblue')

G.add\_edge("Router", device\_name)

return G

def draw\_network\_topology():

"""Desenha a topologia da rede."""

devices = scan\_network("10.5.50.254/24")

G = create\_network\_topology(devices)

pos = nx.spring\_layout(G)

node\_colors = [G.nodes[node].get('color', 'skyblue') for node in G.nodes()]

labels = nx.get\_node\_attributes(G, 'label')

plt.figure("Topologia de Rede")

nx.draw(G, pos, labels=labels, with\_labels=True, node\_color=node\_colors, font\_size=8, node\_size=800)

plt.show()

**Anexo 6.6: Interface Gráfica**

Código que implementa a interface gráfica da aplicação.

def create\_main\_app():

"""Cria a interface gráfica principal usando Tkinter."""

window = tk.Tk()

window.title("Monitor de Rede")

frame = tk.Frame(window)

frame.pack(padx=10, pady=10)

ip\_entry = tk.Entry(frame, width=40)

ip\_entry.insert(0, "10.5.50.254/24")

ip\_entry.pack()

output\_text = scrolledtext.ScrolledText(frame, width=80, height=20)

output\_text.pack(pady=10)

tk.Button(frame, text="Iniciar Monitoramento", command=lambda: threading.Thread(target=detect\_collisions, args=(scan\_and\_save\_network(ip\_entry.get()), output\_text)).start()).pack(pady=5)

tk.Button(frame, text="Capturar Pacotes", command=capture\_packets).pack(pady=5)

tk.Button(frame, text="Visualizar Topologia", command=draw\_network\_topology).pack(pady=5)

tk.Button(frame, text="Dispositivos Salvos no BD", command=show\_stored\_devices).pack(pady=5)

tk.Button(frame, text="Mostrar Colisões", command=show\_stored\_collisions).pack(pady=5)

window.geometry("900x600")

window.mainloop()

**Anexo 6.7: Novo Módulo de Segurança**

Código que demonstra a geração e utilização de chaves de criptografia:

# Geração da chave de criptografia

def generate\_and\_save\_key():

with open("encryption\_key.key", "wb") as key\_file:

key = Fernet.generate\_key()

key\_file.write(key)

# Carregar a chave

def load\_key():

with open("encryption\_key.key", "rb") as key\_file:

return key\_file.read()

**Anexo 6.8: Interface de Login e Registro**

# Tela de login

def login():

def attempt\_login():

username = username\_entry.get()

password = password\_entry.get()

if authenticate\_user(username, password):

messagebox.showinfo("Bem-vindo", f"Bem-vindo(a), {username}!")

else:

messagebox.showerror("Erro", "Credenciais inválidas!")

**Anexo 6.9: Atualização no Gerenciamento de Dispositivos**

def save\_device\_to\_db(device):

encrypted\_ip = encrypt\_data(device['ip'])

encrypted\_mac = encrypt\_data(device['mac'])

encrypted\_hostname = encrypt\_data(device['hostname'])

cursor.execute("""

INSERT INTO devices (ip, mac, hostname, last\_seen)

VALUES (?, ?, ?, CURRENT\_TIMESTAMP)

ON CONFLICT(ip) DO UPDATE SET

mac=excluded.mac,

hostname=excluded.hostname,

last\_seen=CURRENT\_TIMESTAMP

""", (encrypted\_ip, encrypted\_mac, encrypted\_hostname))

---