**Cloud e Cibersegurança**

Projeto  
-  
Programação Aplicada à Cibersegurança

Tomé Vitorino

Bruno Português

Setúbal, 19 de Janeiro de 2025

**Cloud e Cibersegurança**

**App “WireWhale”**

Tomé Andrade Vitorino

Bruno Português

Setúbal, 19 de Janeiro de 2025

# Índice

[**Índice 3**](#_1fob9te)

[**Índice de Imagens 4**](#_3znysh7)

[**Introdução 5**](#_2et92p0)

[**Aplicação - WireWhale 6**](#_1t3h5sf)

[Ferramentas utilizadas 6](#_17dp8vu)

[Estrutura dos ficheiros 7](#_3rdcrjn)

[Apresentação Visual 7](#_26in1rg)

[**Explicação do código - dependencies.py 9**](#_2lwamvv)

[**Explicação do código - main.py 10**](#_jryljdn7tt4l)

[Função \_\_init\_\_ (self, root) 10](#_b96k2ir4aagt)

[Função get\_default\_interface(self) 10](#_ppx49nv4gm3h)

[Função create\_widgets(self) 10](#_u2f5d0hfomnp)

[Função start\_capture(self) 11](#_uauwz6pwm4od)

[Função sniff\_packets(self) 11](#_af07oz1fv2or)

[Função stop\_capture(self) 11](#_5j3mphn8t8ce)

[Função proccess\_packet(self, pkt) 11](#_dbhsv8kf9ene)

[Função get\_protocol(self, pkt) 12](#_gpd242twe4wh)

[Função get\_service\_info(self, source\_port, dest\_port) 12](#_17ckpjaayn20)

[Função apply\_filter(self) 12](#_924w6y53bjrs)

[Função reset\_filters(self) 12](#_uiit60ix6wbu)

[Função update\_treeview(self) 12](#_ubxieatj7247)

[Função clear\_packets(self) 13](#_m1pgofrwxw9h)

[Função save\_packets(self) 13](#_r6r6cy1ruq7i)

[Função main 13](#_53uqewv5ayp2)

[**Conclusão 14**](#_111kx3o)

[**Webgrafia 15**](#_3l18frh)

# Índice de Imagens

[Fig.1 - Aplicação WireWhale. 8](#_hj5pgidulk3a)

[Fig.2 - Demonstração de captura de pacotes. 8](#_tonp26y3d3ir)

# Introdução

Este relatório vai mostrar o desenvolvimento da nossa aplicação de análise de pacotes, *WireWhale*, paralela à já existente “*Wireshark*”. O principal objetivo da *WireWhale* é oferecer uma forma fácil e eficaz de capturar e analisar pacotes de rede com o propósito de estabelecer padrões de tráfego, investigar problemas de conectividade e, mais importante, detectar possíveis ameaças ou ataques maliciosos contra redes de computadores.

Este projeto foi escolhido devido ao nosso interesse na área da cibersegurança e à utilidade prática de ferramentas de análise. A análise de tráfego de rede é indispensável para a segurança e integridade de sistemas e dados.

A aplicação está escrita em *Python*; utiliza a biblioteca *Scapy* para a captura e manipulação de pacotes, enquanto outras bibliotecas e frameworks, como a *Tkinter*, estão integradas para a tornar mais amigável e intuitiva com uma interface gráfica. Foi desenvolvida de forma a funcionar plenamente tanto em *Windows* como em *macOS*.

Este projeto ajudou-nos a compreender melhor a linguagem de programação *Python* e ajudou-nos também a melhorar a nossa forma de programar.

# Aplicação- *WireWhale*

## Ferramentas utilizadas

* *Software:*
  + *Visual Studio Code*;
  + *GitHub Desktop.*
* Linguagens de programação:
  + *Python.*
* Bibliotecas:
  + Externas:
    - *Scapy;*
    - *Pandas;*
  + Internas:
    - *Tkinter;*
    - *Threading;*
    - *Datetime;*
    - *Platform;*
    - *Subprocess;*
    - *Sys.*

## 

## Estrutura dos ficheiros

## 

## 

## 

## Apresentação Visual

### Fig.1 - Aplicação *WireWhale.*

### Fig.2 - Demonstração de captura de pacotes.

# Explicação do código - dependencies.py

### Fig.3 - dependencies.py

Neste *script*, importamos a biblioteca *subprocess* e *sys* para poder executar um *script* que instala as dependências explícitas no ficheiro “requirements.txt”. O código abre e lê esse ficheiro de texto e de seguida executa o comando “pip install” para instalar as dependências encontradas.

# Explicação do código - main.py

## Função \_\_init\_\_ (self, root)

Esta função inicializa a aplicação e define as suas propriedades principais e a interface do utilizador. Ela define o título da janela como “WireWhale”, o tamanho para 950x550 e aplica uma cor de fundo escura. Inicializa também variáveis essenciais como ***packet\_data*** (para armazenar os pacotes capturados), ***filtered\_data*** (para armazenar os pacotes filtrados), e flags como ***capture\_active*** (para controlar o sniffing de pacotes) e ***auto\_scroll\_enabled*** (para gerenciar o comportamento da UI). A interface de rede padrão é detectada chamando a função ***get\_default\_interface()***, e os nomes das colunas para a tabela de pacotes são definidos. Finalmente, a função ***create\_widgets()*** é chamada para criar a interface gráfica do utilizador.

## Função get\_default\_interface(self)

Esta função determina a interface de rede predefinida a ser utilizada para o sniffing de pacotes com base no sistema operativo. Utiliza a ***platform.system()*** do Python para identificar o SO. Para o macOS, a interface predefinida é definida como “en0” e, para o Windows é definida como “Ethernet”. Se o SO for Linux ou outro tipo, devolve ***None***, indicando que a interface deve ser especificada manualmente.

## Função create\_widgets(self)

Esta função cria e organiza a interface gráfica do utilizador para a aplicação. Primeiro, cria um ***canvas*** para alojar os botões com as ações correspondentes. Um segundo ***canvas*** é criado para conter campos de entrada de filtros para *IP*, protocolo e porta, juntamente com um botão para aplicar esses filtros. Também cria uma *treeview* para exibir os detalhes dos pacotes capturados. A tabela possui uma *scrollbar* para melhorar a navegação. Uma label na parte inferior da janela mostra o número de pacotes capturados.

## Função start\_capture(self)

Esta função inicia o processo de captura de pacotes. Ela verifica se a captura já está ativa e, se não estiver, define a *flag* ***capture\_active*** como *True* e cria uma nova thread para executar a função ***sniff\_packets()***. Isto permite que o *sniffing* de pacotes ocorra em segundo plano sem congelar a interface gráfica.

## Função sniff\_packets(self)

Esta função lida com o *sniffing* de pacotes usando a função ***sniff()*** do *Scapy*. Ela especifica o parâmetro ***prn*** para processar cada pacote capturado com a função ***process\_packet()***. O parâmetro ***store*** é definido como 0 para evitar o armazenamento de pacotes na memória do *Scapy*, e ***iface*** é definido como a interface padrão detectada anteriormente. O parâmetro ***stop\_filter*** para o sniffing quando a *flag* ***capture\_active*** é definida como *False*.

## Função stop\_capture(self)

Esta função determina se o sniffing de pacotes deve parar. Ela verifica a *flag* ***capture\_active*** e retorna *True* se a captura não estiver mais ativa, efetivamente parando o processo de *sniffing*.

## Função proccess\_packet(self, pkt)

Esta função processa cada pacote capturado e extrai informações como o registo de data e hora, endereços *IP* de origem e destino, protocolo, portas de origem e destino e informações adicionais sobre o serviço. Ela formata estes dados num tuplo e anexa-os à ***packet\_data***. O novo pacote também é adicionado à tabela na interface do *user*, e a *label* do contador de pacotes é atualizada para refletir o número total de pacotes capturados.

## Função get\_protocol(self, pkt)

Esta função identifica o protocolo de um determinado pacote. Verifica se o pacote contém uma camada *IP* e, em seguida, determina se é *TCP, UDP, ICMP* ou outro tipo. Se o pacote contém uma camada *ARP*, o protocolo é definido como “*ARP*”. Se nenhuma destas camadas estiver presente, devolve “Other”.

## Função get\_service\_info(self, source\_port, dest\_port)

Esta função fornece informações adicionais sobre as portas num pacote mapeando números de portas bem conhecidos para seus serviços correspondentes (por exemplo, a porta 80 é “*HTTP*”). Ela verifica as portas de origem e destino para identificar um serviço correspondente. Se nenhum serviço conhecido for encontrado, ela retorna “Other”.

## Função apply\_filter(self)

Esta função filtra os pacotes capturados com base nos critérios especificados pelo utilizador para endereços *IP*, protocolo e porta. Ela recupera os valores de filtro dos campos de entrada e aplica-os à lista ***packet\_data***. Os pacotes filtrados são armazenados na variável ***filtered\_data*** e exibidos na tabela, substituindo os dados não filtrados.

## Função reset\_filters(self)

Esta função limpa todos os filtros aplicados, redefinindo os campos de entrada e a lista ***filtered\_data***. Ela atualiza a tabela para exibir todos os pacotes capturados, removendo efetivamente qualquer filtragem.

## Função update\_treeview(self)

Esta função atualiza a tabela de visualização de pacotes. Limpa a tabela e volta a preenchê-la com ***filtered\_data*** (se forem aplicados filtros) ou ***packet\_data*** (se não estiverem activos filtros).

## Função clear\_packets(self)

Esta função limpa todos os pacotes capturados limpando tanto a ***packet\_data*** quanto a ***filtered\_data***. Atualiza a tabela para remover todas as entradas e repõe a etiqueta do contador de pacotes para “0”.

## Função save\_packets(self)

Esta função salva os pacotes capturados num arquivo *CSV* chamado ***captured\_packets.csv***. Ela usa a biblioteca *pandas* para criar um *DataFrame* a partir de ***packet\_data*** e depois o exporta. Se não houver pacotes disponíveis, ele imprime uma mensagem indicando que não há nada para salvar.

## Função main

Este é o ponto de entrada do programa. Ele cria um objeto raiz *Tk*, inicializa a classe ***WiresharkApp***, e inicia o *loop* de eventos principal da aplicação com ***root.mainloop()***.

### 

# Conclusão

Este projeto proporcionou-nos uma experiência prática muito valiosa no que respeita à cibersegurança e à utilização de ferramentas de análise de pacotes de rede. O projeto cumpriu o objetivo principal de fornecer um mecanismo de fácil utilização para capturar e analisar o tráfego de rede num dispositivo como forma de identificar padrões, diagnosticar problemas de conectividade ou detetar potenciais ameaças e ataques maliciosos.

A integração de bibliotecas como o *Scapy* para a captura de pacotes e *Tkinter* para a *GUI* mostrou a versatilidade da linguagem *Python* na construção de aplicações fáceis de construir e de fácil utilização.

Este projeto também nos permitiu melhorar a nossa programação e a forma como desenvolvemos uma aplicação ou um script.

# Webgrafia

* *Tkinter Documentation*:

<https://docs.python.org/3/library/tk.html>

* *Scapy* *Documentation*:

<https://scapy.readthedocs.io/en/latest/>

* *Pandas* *Documentation*:

<https://pandas.pydata.org/docs/index.html>