



UNIVERSIDADE
LUSÓFONA

Trabalho Final de Curso

Documentação da API Monitorizador

Miguel Lourenço

Vasco Pereira

27/04/2025

www.ulusofona.pt

Índice

Índice	ii
Lista de Figuras	iii
Resumo	4
Guia de Instalação do Monitorizador	5
Inicialização da API	8
Requests à API	10
Endpoints da API.....	12
Descrição do Código das Funções Principais e Ficheiros	16
Diretório crtsh	24
Diretório knockpy	25
Diretório shodan.....	28
Diretório unused.....	29
Diretório CTI	29
Diretório iocs_feed	32
Diagrama da Estrutura da API.....	37

Lista de Figuras

Figura 1	6
Figura 2	7
Figura 3.....	7
Figura 4	9
Figura 5	10
Figura 6	11
Figura 7	11
Figura 8	12
Figura 9	37

Resumo

O objetivo desta tarefa é a elaboração de uma documentação clara e completa relativa ao desenvolvimento da API realizado ao longo dos últimos meses. Esta documentação visa descrever detalhadamente os principais ficheiros envolvidos no projeto, bem como as funções de maior relevância para o seu funcionamento. Serão ainda apresentados todos os endpoints disponíveis, com a devida explicação dos seus parâmetros, respostas esperadas. Além disso, será incluído um guia prático que descreve, passo a passo, o processo de instalação da API, contemplando os requisitos necessários, configurações e dependências.

Guia de Instalação do Monitorizador

Pré-Requisitos

Para executar o projeto localmente, é necessário garantir que o ambiente cumpre os seguintes requisitos, de modo a permitir a replicação completa deste guia.

Sistemas Operativos:

1. *Linux* (Ubuntu 22.04 ou superior 22.04 ou 24.04 LTS)
2. *macOS*

Todos os sistemas operativos recomendados foram devidamente testados embora seja tecnicamente possível utilizar outra distribuição Linux, não se pode garantir o funcionamento integral do sistema conforme o esperado.

É necessário que a máquina cumpra os seguintes requisitos:

1. *Python* 3.10
2. *Git*
3. Uma IDE à escolha (recomenda-se o *VSCode* ou o *PyCharm*)
4. *Postman* (opcional)
5. *ExploitDB*
6. *Masscan*

Instalação dos principais requisitos

Python

Para instalar o *Python*, é necessário aceder ao terminal da máquina e executar os seguintes comandos:

- `sudo apt update`
- `sudo apt install python3`

Git

Para instalar o *Git*, é necessário aceder ao terminal da máquina e executar o seguinte comando:

- `sudo apt install git`

Exploitdb

Para instalar o *ExploitDB*, recomenda-se consultar a documentação oficial, de forma a garantir uma instalação completa e atualizada. A documentação está disponível em:

<https://www.exploit-db.com/searchsploit>

Masscan

Para instalar o *Masscan*, é necessário aceder ao terminal da máquina e executar os seguintes comandos:

- `sudo apt update`
- `sudo apt install masscan`

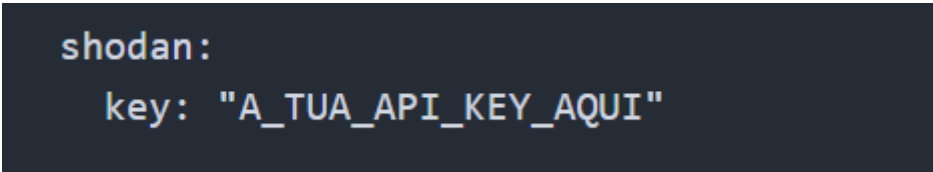
Para além das ferramentas mencionadas acima, é também necessário requerer e configurar as chaves de API, de forma a garantir o funcionamento adequado dos serviços utilizados. Esta configuração deve ser efetuada no ficheiro: `api_keys.yaml`

Keys:

- *Shodan*
- *HackerTarget*
- *SecurityTrails*
- *urlSan.Io*
- *OxSi_f33d*
- *Lookup*
- *Blacklist Checker*
- *Opencti*

Figura 1

Exemplo de como configurar as keys



```
shodan:  
  key: "A_TUA_API_KEY_AQUI"
```

É necessário configurar o ficheiro `configs.yaml` com a interface de rede correta. Para identificar a interface em uso, pode utilizar o seguinte comando no terminal:

- `ip a`

Figura 2

Exemplo de requisição da interface de internet ens33

```
migu@migu-VMware-Virtual-Platform:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 q
    link/loopback 00:00:00:00:00:00 brd 0
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft fo
    inet6 ::1/128 scope host noprefixrout
        valid_lft forever preferred_lft fo
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_U
    link/ether 00:0c:29:84:6b:26 brd ff:f
    altname enp2s1
    inet 192.168.1.24/24 brd 192.168.1.255
        valid_lft 1685sec preferred_lft 16
    inet6 fe80::20c:29ff:fe84:6b26/64 sco
        valid_lft forever preferred_lft fo
```

- masscan_interface: "ens33" (exemplo)

É ainda necessário configurar no mesmo ficheiro o url da máquina que contem o opencti de modo

Figura 3

Exemplo de url Opencti

```
opencti_url: 'http://139.59.163.170:8080/'
```

Após garantir que todas as dependências mencionadas foram corretamente instaladas, podemos proceder com a instalação do projeto.

Inicialização da API

Clonar repositório

Para clonar o repositório para a máquina local, deve-se executar o seguinte comando no terminal:

- `git clone https://github.com/duke-the-1998/TFC-Lusofona-API`

Criar e Ativar um ambiente Virtual

O *Python* utiliza ambientes virtuais, nos quais é possível instalar as bibliotecas necessárias para a execução do projeto. Para criar e ativar o ambiente virtual, devem ser utilizados os seguintes comandos:

- `sudo apt install python3.12-venv`
- `python -m venv venv`
- `source venv/bin/activate`

Instalar dependências

O repositório já inclui as dependências necessárias para o funcionamento do projeto. Para instalá-las, deve-se executar o seguinte comando:

- `pip install -r requirements.txt`

Como correr API

Após concluir todos os procedimentos acima descritos, podemos proceder à execução do projeto. O mesmo deve ser executado a partir do terminal. Como no Linux não é possível utilizar o `sudo` fora do ambiente virtual, e como o `sudo` é necessário para a execução de certos componentes no diretório principal do projeto, deve-se executar o seguinte comando:

- `which python`

Deste modo, será possível obter o caminho até ao *Python* dentro do ambiente virtual. Após isso, podemos executar o seguinte comando:

- `sudo /home/user/PycharmProjects/TFC-Lusofona-API/.venv/bin/python monitorizador.py`

Este comando irá iniciar o servidor *Flask* local na porta 5000, acessível em <http://127.0.0.1:5000/>. No entanto, a porta pode ser alterada no ficheiro `monitorizador.py`.

Figura 4

Porta padrão flask

```
if __name__ == "__main__":  
    app.run(port=5000)
```

Requests à API

Existem três formas principais de interagir com os endpoints da API:

Postman

Pode utilizar o Postman para testar os endpoints expostos pela API.

Exemplo de URL de request:

- <http://127.0.0.1:5000/monitorizador/DOM>

Neste exemplo:

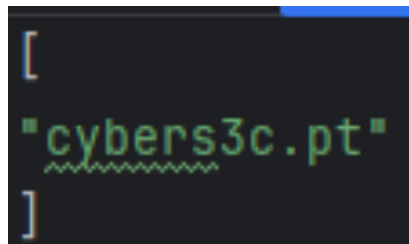
- 5000 que é a porta definida no servidor *Flask*
- monitorizador/DOM representa o *endpoint* que queremos fazer um *request*

O método HTTP a ser utilizado deve ser o POST. No *Postman*, essa opção pode ser selecionada no menu suspenso à esquerda do campo onde é inserido o URL da requisição.

Após definir o método HTTP e o URL, é necessário inserir no corpo da requisição (*body*) uma lista de IPs ou domínios que se deseja consultar, no formato de lista, como exemplificado abaixo:

Figura 5

Exemplo de requisição



```
[  
  "cybers3c.pt"  
]
```

script teste_conexao_api.py

Foi também desenvolvido um script para facilitar a realização de testes automáticos através da linha de comandos.

Exemplo de execução:

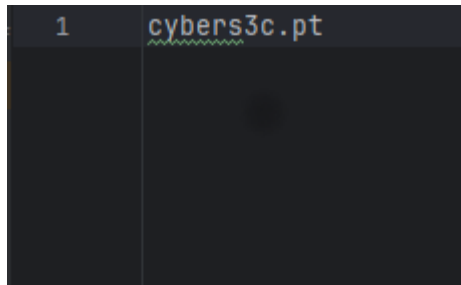
- `python teste_conexao_api.py monitorizador -t DOM -a test.txt`

Neste exemplo:

- DOM é o tipo de scan que se quer realizar
- test.txt é onde vai estar a lista dos alvos que queremos analisar

Figura 6

Exemplo de requisição

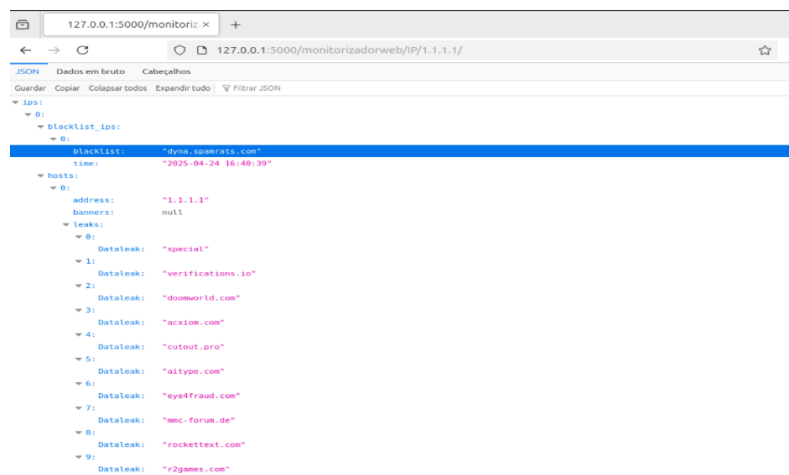


Request browser

É possível requisitar alguns endpoints diretamente através do *browser*; no entanto, essa abordagem está limitada aos endpoints que utilizam o método GET, como é o caso de monitorizadorweb e outros semelhantes.

Figura 7

Exemplo de Requisição



Endpoints da API

Monitorização de Ips

Este *endpoint* retorna informações detalhadas sobre um determinado IP fornecido, bem como sobre os IPs associados. Foi desenvolvido com o método POST, permitindo receber um ficheiro JSON contendo uma lista de IPs.

Parâmetros:

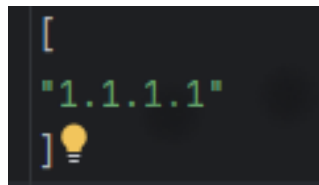
- **Ip's (obrigatório):** Os endereços Ip's a ser analisados. Exemplo: ["8.8.8.8"]

Exemplo de Requisição:

- <http://127.0.0.1:5000/monitorizador/IP>

Figura 8

Exemplo de Requisição Post



Monitorizador de IPS via Web

Este *endpoint* retorna informações detalhadas sobre um determinado IP fornecido e os IPs a ele associados. Ao contrário do anterior, foi desenvolvido com o método GET, sendo, por isso, possível analisar apenas um endereço IP de cada vez.

Parâmetros

- **IP (obrigatório):** IP a ser analisado, como no exemplo: /1.1.1.1/
-

Exemplo de Requisição:

- <http://127.0.0.1:5000/monitorizadorweb/IP/1.1.1.1/>

Monitorização de Domínios

Este *endpoint* retorna informações detalhadas sobre um determinado domínio fornecido, bem como sobre todos os seus subdomínios. Foi desenvolvido com o método POST, permitindo o envio de um ficheiro JSON contendo uma lista de domínios ou subdomínios.

Parâmetros

- **domínios(obrigatório):** Domínios a analisar. Exemplo: ["cybers3c.pt", "teste.pt"]

Exemplo de Requisição:

- <http://127.0.0.1:5000/monitorizador/DOM>

Monitorizador de Domínios via Web

Este *endpoint* retorna informações detalhadas sobre um determinado domínio fornecido, bem como os seus subdomínios associados. Ao contrário do anterior, foi desenvolvido com o método GET, sendo por isso possível analisar apenas um domínio de cada vez.

Parâmetros:

- **domínio (obrigatório):** Domínio a ser analisado, como no exemplo: /**exemplo.com/**

Exemplo de Requisição:

- <http://127.0.0.1:5000/monitorizadorweb/DOM/exemplo.com/>

LOOKUP

Este *endpoint* retorna os *data leaks* associados a um determinado IP, domínio, nome de utilizador (*username*) ou endereço de e-mail. Foi desenvolvido com o método POST, permitindo o envio de um ficheiro JSON com uma lista de IPs, domínios, *usernames* e e-mails para os quais se pretende obter essa informação.

Parâmetros:

- **IP, Domínio, Username ou mail (obrigatório)** a ser analisado

Exemplo de requisição

- <http://127.0.0.1:5000/LOOKUP/>

CVE

Este *endpoint* retorna os scripts e *exploits* associados a um determinado CVE pesquisado. Foi desenvolvido com o método POST, permitindo o envio de um ficheiro JSON com uma lista de CVEs para consulta.

Parâmetros:

- **Cves (Obrigatório):** CVE a ser analisado como no exemplo: ["cve-202154"]

Exemplo de requisição:

- <http://127.0.0.1:5000/CVE/>

pesquisar-cve

Este *endpoint* retorna todos os CVEs associados a uma determinada palavra-chave ou tecnologia. Foi desenvolvido com o método POST, permitindo o envio de um ficheiro JSON contendo uma lista de palavras-chave para procurar os CVEs correspondentes.

Parâmetros:

- **Palavras (Obrigatório):** Palavra a ser analisada como no exemplo: ["Ubuntu "]

Exemplo de requisição:

- <http://127.0.0.1:5000/pesquisa-cve/>

OxSI_feed

Este *endpoint* retorna à consulta do OxSI_f33d da segurança informática em formato JSON. Foi desenvolvido com o método POST.

Exemplo de Requisição:

- http://127.0.0.1:5000/OxSI_feed/

ctipais

Este *endpoint* retorna à consulta sobre o *feed* do OpenCTI relacionado a ataques direcionados a um país durante um período definido.

Parâmetros:

- **/ctipais/<typeScan>/<typeScan2>/<typeScan3>/**

O primeiro parâmetro (<typeScan>) é obrigatório e corresponde ao nome do país.

Exemplo:

- "United Kingdom of Great Britain and Northern Ireland"

O segundo (<typeScan2>) e terceiro parâmetro (<typeScan3>) correspondem à data do período de procura. Ambos devem seguir o formato MM-AAAA.

Exemplo:

- "03-2025"

Exemplos de Requisição:

- <http://127.0.0.1:5000/ctipais/Portugal/03-2025/04-2025/>

- <http://127.0.0.1:5000/ctipais/Portugal/None/03-2025/>
- <http://127.0.0.1:5000/ctipais/Portugal/03-2025/>

ctiTOP10

Este *endpoint* retorna a consulta sobre o *feed* do *OpenCTI* relacionado aos 10 principais ataques direcionados a um determinado país. Foi desenvolvido com o método GET.

Parâmetros:

- **ctiTOP10/<typeScan>/ (Obrigatório)** como no exemplo – Portugal

Exemplos de Requisição:

- <http://127.0.0.1:5000/ctiTOP10/Portugal/>

ctiweb

Este *endpoint* retorna a consulta sobre os *feeds* do *OpenCTI* relacionados aos *hashes*, domínios e IPs do último mês. Foi desenvolvido com o método GET.

Parâmetros:

Este endpoint oferece **três subopções** para retornar o feed de **hashes**, **IPs** ou **domínios**:

- O parâmetro **<typeScan>** é **obrigatório** e define o tipo de feed a ser retornado. Os valores possíveis são:
 - hashes
 - DOM (domínios)
 - IPS

Exemplos de Requisição:

- <http://127.0.0.1:5000/ctiweb/DOM/>
- <http://127.0.0.1:5000/ctiweb/IPS/>
- <http://127.0.0.1:5000/ctiweb/hashes/>

Descrição do Código das Funções Principais e Ficheiros

Diretório Core.py

Dom_checker.py

Este *script* é responsável por realizar o processamento completo das informações de um domínio e dos seus subdomínios para resposta ao *endpoint* /monitorizador/DOM. Este *script* equipare-se ao *ips.py* (que trata os dados associados ao *endpoint* de endereços IP). Ele processa operações como deteção de subdomínios, análise de SSL, verificação de *typosquatting* e listas negras. A seguir estão presentes as funções e a suas respetivas descrições:

is_valid_domain

Esta função verifica se um domínio é válido através de uma expressão regular `r"^((?!-)[A-Za-z0-9-]{1,63}(?!-)\.)+[A-Za-z]{2,6}$"`. Recebe como parâmetro um nome de domínio e retorna verdadeiro se o domínio for válido, ou falso caso contrário. Esta função é bastante importante para a validação da lista de domínios fornecida pelo utilizador, garantindo que os domínios enviados para as próximas etapas do script estejam corretamente formatados.

clear_url

Esta função é responsável por receber uma *string* que contém um domínio ou subdomínio e limpar o URL para obter apenas o nome do domínio. Remove prefixos como `www.` e extrai apenas a parte principal do domínio. Esta função é utilizada para limpar tanto os domínios como os subdomínios, garantindo uma camada adicional de validação e assegurando que todas as entradas estejam padronizadas antes de serem processadas pelas próximas etapas do script, resultando numa resposta mais organizada e limpa.

simplify_list

Esta função tal como o nome indica, simplifica uma lista de listas, removendo os duplicados. É utilizada no processamento de subdomínios, dado um determinado domínio. Como o processamento de subdomínios é realizado através de várias fontes públicas e APIs, é comum existirem duplicados que precisam de ser eliminados.

get_crtsh_subdomains

Esta função obtém os subdomínios de um determinado domínio através da API do serviço `crt.sh`. O processamento da consulta é realizado no ficheiro `crtsh.py`, sendo que esta função invoca o método

`crtshAPI().search()` para recolher os subdomínios associados ao domínio alvo. Os dados são retornados sob a forma de uma lista limpa e padronizada.

get_all_subdomains

Esta função é responsável por obter todos os subdomínios a partir de um determinado domínio. Tal como mencionado anteriormente, estes subdomínios são recolhidos a partir de várias fontes e APIs, como *crt.sh*, *Knockpy*, *Dnsdumpster*, *Shodan* e *SecurityTrails*. Como os subdomínios são obtidos de diversas fontes, a lista necessita de ser validada e retirados os duplicados. A função conta ainda com um mecanismo de *threads* para minimizar o tempo de resposta.

check_reason

Esta função traduz as mensagens de erro para mensagens mais compreensíveis facilitando a identificar a razão de falha no código, assegurando uma resposta mais limpa e organizada.

process_subdomain

Esta função é uma das partes centrais do *script*, sendo responsável por analisar um determinado subdomínio, recolhendo diversas informações como endereço IP, certificados SSL, cabeçalhos de segurança, tecnologias utilizadas e CVEs associados. Estas informações são obtidas a partir de várias fontes e APIs (como *Shodan*, *Netlas*, entre outras), integradas noutras funções e scripts. A função retorna um dicionário com todos os dados relevantes sobre esse subdomínio. Esta função é chamada para cada um dos subdomínios encontrados.

processar_subdominio

Esta função obtém informações de diversas fontes externas (*ONYPHE*, *Netlas* e *Shodan*) para um subdomínio específico. Após recolher esses dados, invoca a função `process_subdomain`, passando-lhe todos os parâmetros necessários.

api_keys

Esta função obtém as chaves das APIs a partir do ficheiro YAML, para que possam ser utilizadas nas várias funções do *script*.

subdomains_finder_dnsdumpster

Esta função obtém os subdomínios de um determinado domínio através da API do *HackerTarget* (*DNS-Dumpster*). A função envia uma requisição à API, utilizando uma chave da API obtida pela função

hackertarget_key(), e processa a resposta que contém pares de subdomínios e IPs. A lista que esta função retorna contém apenas os nomes dos subdomínios.

ssl_version_supported

Esta função verifica quais as versões SSL/TLS que estão a ser utilizadas num determinado domínio. Para isso, estabelece uma conexão com o domínio através da biblioteca padrão *ssl* do *Python*. Após a conexão, chama a função *check_ssl_version()* para identificar as versões suportadas e em uso.

check_ssl_version

Esta função verifica quais as versões SSL/TLS que estão a ser usadas para o hostname. Utiliza a biblioteca padrão *ssl* do *Python* e recebe como parâmetro a conexão SSL realizada pela função *ssl_version_supported*. Esta análise permite avaliar se o serviço está a utilizar versões seguras e atualizadas do protocolo.

db_insert_domain

Esta função insere o domínio na estrutura de dados principal *jsonDominio* e as suas informações detalhadas, como IP, data *leaks* e certificados *securitytrails*. Estas informações são recolhidas a partir de várias fontes externas e APIs, através de chamadas a outras funções.

blacklisted

Esta função verifica se o IP de um domínio está listado em uma ou mais *blacklists* DNS, que são usadas para identificar fontes de spam ou que não sejam confiáveis. A função consulta uma lista de mais de 60 serviços de *blacklist* e utiliza *threads* para fazer essas consultas de forma eficiente.

Insert_headers

Esta função é responsável por verificar e inserir os cabeçalhos de segurança HTTP para um subdomínio. Utiliza a classe *SecurityHeaders* para realizar uma análise dos cabeçalhos de segurança presentes na resposta HTTP do subdomínio. Para cada cabeçalho de segurança, é verificado se ele está definido e qual é o seu conteúdo. Cada resultado recebe um status de acordo com a presença e validade do cabeçalho, sendo classificado como OK (quando está corretamente definido) ou WARN (quando está ausente ou mal configurado). Todos os resultados são organizados numa tabela visual exibida no terminal e armazenados numa lista de dicionários.

typo_squatting_api

Esta função tem como objetivo identificar possíveis domínios de *typo-squatting* relacionados a um domínio. Utiliza a API do serviço DNS *Twister*, que gera variações do domínio original (com erros de digitação comuns) e verifica se esses domínios estão registrados e resolvem para algum endereço IP.

ip_models.py

Neste *script* estão definidas todas as classes referentes aos IPs, como *ModelHost*, *ModelPort* e *ModelInfo*, que serão utilizadas através de funções nos ficheiros *ips.py* e *utils.py*.

Ips.py

Este *script* é responsável por realizar o processamento completo das informações dos ips para resposta ao endpoint */monitorizador/IP*. Este script equipara-se ao *dom_checker.py* (que trata os dados associados ao *endpoint* dos domínios e seus subdomínios). Ele processa operações como masscan, nmap, reverse IP, listas negras, leaks, banners, tecnologias e análise SSL. A seguir estão presentes as funções e as suas respetivas descrições.

validate_ip_address

Esta função verifica se um determinado IP é válido através de uma biblioteca do *Python* chamada *ipaddress* e do *isinstance()*, uma função que verifica se um objeto é instância de uma classe ou não. Assim, através destas duas ferramentas, a função recebe como parâmetro um IP e retorna verdadeiro se for válido, falso caso contrário. Esta função é bastante importante para a validação da lista de IPs que o utilizador envia, garantindo que os IPs que estão a ser enviados para as próximas etapas do script estejam bem formatados.

validate_network

Esta função é responsável por verificar se uma determinada rede IP é válida através da biblioteca do *Python ipaddress* e a função integrada desta biblioteca *ip_network*. Através desta função, se o valor fornecido não for uma rede válida, será lançada uma exceção *ValueError* e o *try-except* captura esse erro, retornando false nesse caso. Esta função é bastante importante para a validação da lista de redes IPs, garantindo que as redes IPs que estão a ser enviadas para as próximas etapas do script estejam bem formatadas.

is_private

Esta função verifica se um determinado IP é privado através da biblioteca do *Python ipaddress* e a função integrada desta biblioteca *ip_address().is_private*. Através desta função, se o valor fornecido for

um IP privado, retorna verdadeiro; caso contrário, retorna falso. Esta função é bastante útil na função *reverse_ip_lookup* para verificar se o IP dado não é privado, permitindo assim executar o comando *nslookup*.

ip_range_cleaner

Esta função é responsável por estender uma gama de endereços IP fornecida em formato de rede, utilizando a biblioteca *ipaddress* do *Python* e a função integrada desta biblioteca *ip_network().hosts()* para gerar todos os endereços IP válidos. Todos os endereços gerados são escritos no ficheiro *cleanIPs.txt*, permitindo acumular os resultados para utilização noutras funções. Esta função é especialmente útil para preparar listas de IPs que serão posteriormente analisadas ou submetidas a varreduras.

Class Importer

Esta classe é responsável por definir a estrutura geral para importação e processamento de informações relacionadas a *hosts* e IPs. Isto inclui informações como: endereço IP, portas abertas (protocolo, estado, descrição e SSL associados), *leaks*, *banners*, tecnologias associadas, sistema operativo e organização associada.

Class NmapXMLImporter

Esta classe é uma subclasse de *Importer* que é responsável por ler e interpretar ficheiros de output do Nmap no formato XML, extraíndo dados dos hosts e das suas respetivas portas.

Configsa

Esta função tem como objetivo obter a configuração da interface através do ficheiro *configs.yaml*. Ela abre o ficheiro localizado em */core/configs.yaml* e obtém uma string com a interface do *masscan*. Esta função é útil para executar o comando *masscan*, pois define qual é a interface de rede que será utilizada durante o processo de varrimento. Caso o ficheiro de configuração não seja encontrado, a função retorna *None*.

ip_scan

Esta função é responsável por executar os comandos *masscan* e *nmap* através da biblioteca do *Python* *subprocess*, que possibilita a execução de comandos diretamente no terminal. A função utiliza o *Masscan* para realizar uma varredura rápida de todas as portas no IP fornecido. Em seguida, as portas detectadas são extraídas e utilizadas para realizar um scan mais detalhado com o *Nmap*, cujos resultados são guardados num ficheiro XML.

blacklistedIP

Esta função verifica se o IP está listado em uma ou mais *blacklists* DNS, que são utilizadas para identificar fontes de spam ou IPs não confiáveis. Ela consulta uma lista de mais de 60 serviços de *blacklist* e utiliza *threads* para realizar essas consultas de forma eficiente.

reverse_ip_lookup

Esta função tem como objetivo realizar uma pesquisa de DNS reverso para um determinado endereço IP público, ou seja, tentar obter o nome de domínio associado a esse IP. A função verifica primeiro se o IP fornecido não pertence a uma rede privada. Caso seja um IP público, executa o comando *nslookup*. Esta operação é útil para identificar o *hostname* de um IP.

utils.py

Este script atua como intermediário para executar os *scripts* responsáveis pelo processamento de informações para os *endpoints* dos domínios e IPs (*dom_checker.py* e *ips.py*).

run_ips

Esta função é responsável por executar as funções relativas ao processamento das informações para o endpoint dos IPs. Antes de executar essas funções, o IP fornecido é validado, garantindo que o código não apresenta erros e que seja mais organizado. As funções chamadas pelo *run_ips* fazem parte do script *ips.py*, que executa operações como varreduras ao *ip*, *lookup* reverso e verificação em *blacklists*.

run_domains

Esta função tem como objetivo executar as funções relativas ao processamento das informações para o *endpoint* dos domínios, equiparando-se à função *run_ips*, mas só que opera para os domínios. Antes de executar, o domínio fornecido é validado, garantindo que o código não apresenta erros e que seja mais organizado. As funções chamadas pelo *run_domains* fazem parte do script *dom_checker.py*, que executa operações como verificação de certificados, subdomínios, tecnologias, cves associados e verificação em *blacklists*.

is_subdomain

Esta função verifica se um subdomínio é válido através de uma expressão regular `r'[0-9a-zA-Z\.-]*\.[0-9a-zA-Z\.-]*\.[w+]`. Recebe como parâmetro um nome de um subdomínio e retorna verdadeiro se o subdomínio for válido, ou falso caso contrário. Esta função é essencial para a validação da lista de

domínios fornecida pelo utilizador, verificando se esta contém algum subdomínio. Esta verificação assegura que as etapas seguintes do script estejam devidamente formatadas e preparadas para serem executadas.

is_main_domain

Esta função verifica se um domínio é válido através de uma expressão regular `"r'^[a-z0-9](?:[a-z0-9-]{0,61}[a-z0-9])?\.[a-z0-9][a-z0-9-]{0,61}[a-z0-9]"`. Recebe como parâmetro um nome de domínio e retorna verdadeiro se o domínio for válido, ou falso caso contrário. Esta função é bastante importante para a validação da lista de domínios fornecida pelo utilizador, garantindo que os domínios enviados para as próximas etapas do *script* estejam corretamente formatados.

get_main_domain

Esta função é responsável por extrair o domínio principal a partir de um subdomínio fornecido como parâmetro. Esta funcionalidade é especialmente útil para processar a lista de domínios fornecida pelo utilizador, permitindo que, caso algum dos itens seja um subdomínio, seja automaticamente convertido no respetivo domínio principal.

treat_domains

Esta função tem como objetivo validar a lista de domínios fornecida pelo utilizador, identificando quais dos elementos são domínios principais e quais são subdomínios. Caso sejam detetados subdomínios, estes são convertidos para os respetivos domínios principais. Esta verificação e validação é realizada com recurso às funções *get_main_domain*, *is_main_domain* e *is_subdomain*.

valid_TLD

Esta função é responsável por devolver uma lista com os domínios válidos, verificados através de uma consulta DNS. Para cada domínio fornecido, tenta resolver o nome através do sistema DNS. Apenas os domínios que obtêm uma resolução com sucesso são considerados válidos e adicionados à lista final.

delete_aux_files

Esta função apaga ficheiros auxiliares temporários, como *cleanIPs.txt*, *scans.txt* e *mscan.json*, caso existam. Informa que todos os ficheiros foram apagados após a execução.

clean_useless_files

Esta função apaga o ficheiro *cleanIPs.txt* se ele existir. Caso contrário, exibe uma mensagem informando que o ficheiro não foi encontrado.

Security_headers.py

Neste ficheiro, estão inseridas, dentro da classe *SecurityHeaders*, algumas funções relacionadas com os cabeçalhos de segurança.

evaluate_warn

Esta função avalia o risco de um cabeçalho HTTP com base no seu conteúdo, definindo um sinalizador de alerta se for considerado inseguro. Retorna um dicionário com o estado de alerta, a presença do cabeçalho e o seu conteúdo.

test_https

Esta função verifica se um site suporta HTTPS e se o seu certificado SSL é válido. Retorna um dicionário com dois campos: *supported* (suporte a HTTPS) e *certvalid* (validação do certificado).

test_http_to_https

Esta função testa se um endereço HTTP redireciona para HTTPS, seguindo até 5 redirecionamentos. Devolve *True* se o redirecionamento ocorrer com sucesso, ou *false* caso contrário.

check_headers

Esta função verifica a presença e o estado de cabeçalhos de segurança importantes. Permite seguir redirecionamentos e identificar falhas na segurança dos cabeçalhos HTTP/HTTPS.

create_json.py

guardar

Esta função tenta abrir e gravar os dados de domínios, previamente recolhidos e armazenados na variável *jsonDominios*, num ficheiro chamado *teste.json*. Caso haja algum erro durante o processo, este será tratado e apresentado.

guardar_json_ips

Esta função tenta abrir e gravar os dados de IPs, previamente recolhidos e armazenados na variável *jsonIps*, num ficheiro chamado *testeIp.json*. Caso haja algum erro durante o processo, este será tratado e apresentado.

clean_json_IPS

Esta função é utilizada para limpar o ficheiro JSON referente aos IPs, de modo a não ficar informação residual para próximas chamadas à API.

clean_json

Esta função é utilizada para limpar o ficheiro JSON referente aos domínios, de modo a não ficar informação residual para próximas chamadas à API.

configs.yaml

Neste ficheiro, estão contidas as configurações referentes à API, neste caso, a interface de rede, especificada como *masscan_interface*.

api_keys.yaml

Neste ficheiro, estão presentes todas as chaves de APIs que terão de ser configuradas para o devido funcionamento da API.

Diretório crtsh

crtshAPI

A classe *crtshAPI* contém a função referente à pesquisa, sendo responsável por realizar consultas à API para obter dados sobre certificados e subdomínios a partir de um domínio fornecido.

Search

Esta função pesquisa o domínio na base de dados crt.sh, permitindo incluir ou excluir certificados expirados. Retorna uma lista de objetos com informações detalhadas sobre os certificados encontrados.

crtsh_cert_info.py

flatten

Esta função recebe listas ou sublistas aninhadas e as expande, retornando uma estrutura plana, sem a necessidade de lidar com listas opcionais aninhadas.

check_expiration_date

Esta função recebe a data de expiração de um certificado SSL e retorna o número de dias restantes até a expiração.

check_cert

Esta função verifica o certificado SSL de um domínio, retornando detalhes como a data de expiração, a data de início e o nome da organização, ou um erro caso a verificação falhe.

check_cert_output

Esta função extrai e formata as informações do certificado SSL, incluindo a data de expiração, a data de início e o nome da organização, além de calcular o tempo restante até a expiração.

Diretório knockpy

config.py

Este ficheiro contém as configurações necessárias para poder executar as funções do *Knockp*.

Dns_socket.py

parse_dns_string

Esta função interpreta uma sequência de bytes DNS codificada, convertendo-a num nome de domínio legível, com suporte a apontadores de reutilização.

class StreamReader

Esta classe conta com diversas funções diversas utilizadas no *knockpy*.

reuse

Esta função converte pós num índice, ajusta-o se necessário e reutiliza dados a partir desse ponto no *buffer* para analisar uma *string* DNS.

make_dns_query_domain

Esta função constrói e codifica uma *string* DNS no formato de consulta, separando o domínio em partes com comprimento prefixado.

make_dns_request_data

Esta função cria uma mensagem de requisição DNS no formato binário, com cabeçalho e dados da consulta especificados.

add_record_to_result

Esta função adiciona ao resultado um registo DNS do tipo A ou CNAME, convertendo os dados conforme o tipo.

parse_dns_response

Esta função processa a resposta a uma consulta DNS, extraindo e organizando registos do tipo A e CNAME num dicionário de resultados.

dns_lookup

Esta função realiza uma consulta DNS para um domínio, enviando a requisição para o servidor especificado por *address*.

_gethostbyname_ex

Esta função resolve um domínio através de DNS, devolvendo o nome, os aliases e os endereços IPv4 associados.

wordlist.txt

Este ficheiro contém todas as palavras utilizadas para conseguir encontrar os subdomínios no processo de análise.

knockpy.py

Este ficheiro contém presentes diversas classes com as respetivas funções utilizadas na busca dos subdomínios através do *Knockpy*.

Class Request

dns

Esta função resolve o nome de domínio alvo para obter o endereço IP. Utiliza um servidor DNS personalizado se definido na configuração (*config["dns"]*), caso contrário, utiliza o DNS padrão do sistema.

https

Esta função tenta estabelecer uma ligação HTTPS com o URL fornecido, utilizando um *user-agent* aleatório e um tempo limite definido na configuração. Retorna uma lista com o código de estado HTTP e o valor do cabeçalho *"Server"*, ou uma lista vazia em caso de erro.

http

Esta função tenta estabelecer uma ligação HTTP com o URL fornecido, utilizando um *user-agent* aleatório e um tempo limite definido na configuração. Retorna uma lista com o código de estado HTTP e o valor do cabeçalho *"Server"*, ou uma lista vazia em caso de erro.

bs4scrape

Esta função tenta extrair subdomínios de uma página HTML, identificando ligações que apontem para subdomínios do alvo fornecido. Devolve uma lista de subdomínios encontrados na resposta HTML, se o estado da resposta for 200.

Class Wordlist

Local

Esta função lê ficheiros locais linha por linha, devolvendo apenas as linhas não vazias.

google

Esta função realiza uma pesquisa no Google para encontrar subdomínios do domínio indicado, utilizando *scraping* com *BeautifulSoup* para extrair os resultados.

duckduckgo

Esta função realiza uma pesquisa no *DuckDuckGo* para encontrar subdomínios do domínio indicado, utilizando *scraping* com *BeautifulSoup* para extrair os resultados.

get

Esta função recolhe palavras de várias fontes (local, *Google* e *DuckDuckGo*), conforme definido na configuração, para formar uma lista de palavras úteis na descoberta de subdomínios.

Class output

progressPrint

Esta função é utilizada para atualizar a linha de comando (ou terminal) com um texto que muda dinamicamente, sem criar linhas.

jsonizeRequestData

Esta função organiza e estrutura informações sobre subdomínios, aliases, IPs, código HTTP e servidor no formato JSON, associando-as ao domínio de destino.

linePrint

Esta função formata e imprime uma linha de dados sobre um endereço IP, subdomínios, código HTTP, servidor e domínio. Ajusta o espaçamento de cada elemento com base no comprimento máximo definido, garantindo uma apresentação organizada e legível.

Classe Start

scan

Esta função realiza a verificação de DNS e HTTP(S) para um determinado domínio e subdomínio e, dependendo dos resultados obtidos, formata e armazena a informação num dicionário de resultados.

knockpy

Esta função executa uma varredura de subdomínios para um determinado domínio, usando uma lista de palavras (*wordlist*) obtida de fontes locais, *Google* e *DuckDuckGo*.

Diretório shodan

Neste diretório, estão presentes as funções referentes ao *Shodan*, uma das componentes principais do monitorizador. Estão presentes funções tanto para a procura de informações referentes a um determinado IP ou domínio, como também para a procura de subdomínios.

API

Esta função devolve a chave necessária para conectar à API. A chave está presente no ficheiro das chaves (*keys*). Em caso de erro, a função retorna *None*.

shodan_subdomains

Esta função procura identificar os subdomínios de um domínio através de uma consulta à API do *Shodan*. Retorna uma lista com os subdomínios encontrados ou uma lista vazia em caso de erro.

search_domain_info

Esta função obtém informações detalhadas sobre um determinado domínio ou subdomínio, realizando uma consulta à API do *Shodan*. Ela retorna um conjunto de listas contendo informações relacionadas aos IPs, subdomínios, cabeçalhos de segurança, tecnologias, portas abertas, versões TLS e serviços, com itens não repetidos. Em caso de erro, retorna listas vazias.

search_ip_info

Esta função recolhe informações detalhadas sobre um IP, realizando uma consulta à API do *Shodan*. Ela retorna um JSON contendo informações sobre o IP, portas abertas, banners, localização, organização, sistema operativo e tecnologias associadas.

Diretório unused

Neste diretório estão presentes todos os scripts não usados

Diretório CTI

Neste diretório estão presentes todos os scripts desenvolvidos para retirar informação e processá-la de diversas fontes externas utilizadas ao longo do projeto.

Leak_lookup.py

valida_tipoScan

Esta função valida se o tipo recebido é um e-mail, IP, domínio ou nome de utilizador.

valida_email

Esta função verifica se o e-mail é válido.

valida_ip

Esta função verifica se um IP é válido.

verify_domain

Esta função verifica se é um domínio válido.

consultarAPiLeakLookup

Esta função faz uma consulta à API do *Lookup*, de modo a retornar os *leaks* para um domínio, IP, email ou nome de utilizador.

guardar_json

Esta função guarda o resultado da pesquisa num ficheiro JSON.

lookup_api

Esta função principal vai solicitar os parâmetros, como a informação a extrair e o formato em que se pretende armazenar, realiza a consulta dos mesmos e processa os resultados.

Consulta_dominio_ip_api

Esta função processa os dados retornados para consultas de domínio ou IP e guarda os resultados numa lista.

Consulta_email_username_api

Esta função processa os dados retornados para consultar email ou nome de utilizador e guarda os resultados numa lista.

black.py

consulta_black

Esta função faz uma pesquisa através do *BlacklistCheckers* para verificar se um domínio ou IP estão presentes em alguma *blacklist* conhecida.

exploidb.py

Neste ficheiro estão presentes as funções referentes ao *Exploid*.

cves

Esta função recolhe *exploits* associados a um determinado CVE e retorna um dicionário com as informações dos mesmos, como *cve*, *exploit_title*, *exploit_id*, *exploit_link*, *date_published*, *date_added*, *date_updated*, *author*, *type*, *platform*, *tags*, *aliases* e *zexploit_code*.

get_exploit_data

Esta função pesquisa *exploits* relacionados a uma determinada CVE na base de dados *Exploit-DB*, utilizando a ferramenta *searchsploit*, e devolve os resultados no formato JSON.

get_exploit_code

Esta função retorna o código-fonte de um *exploit* da base de dados *Exploit-DB*, com base no seu id.

my_cve.py

Neste script estão presentes todas as funções de requisição de informação à API do NVD.

consultar_cveTec

Esta função pesquisa CVEs relacionadas com um termo fornecido, utilizando a API da *NVD*, e devolve uma lista de identificadores CVE encontrados.

consultar_cv

Esta função pesquisa CVEs relacionadas com um termo, usando a API da *NVD*, e devolve as vulnerabilidades encontradas com detalhes como descrição, versão do CVSS, pontuação e gravidade.

guardar_json

Esta função guarda os resultados da pesquisa com a informação de uma CVE no formato JSON.

netlas_domain.py

Neste script estão presentes todas as funções de requisição de informação à API do *Netlas*.

netlas_connection

Esta função estabelece uma ligação com a API do *Netlas*.

netlas_domain

Esta função consulta a API do *Netlas* para obter informações sobre um domínio e devolve a resposta ou *None* em caso de erro.

netlas_lookup

Esta função processa os dados obtidos da API do *Netlas* sobre um domínio, extraíndo e organizando as informações de portas, *software* e DNS.

onyphe_domains.py

conecao

Esta função realiza uma pesquisa na API da *Onyphe* com base num domínio, devolvendo os resultados no formato JSON.

domains

Esta função obtém informações detalhadas sobre um domínio ou subdomínio, incluindo dados de rede, certificados, produtos, protocolos e respostas *HTTP*, *scan_info*, *TLS*, *ports* e *transport*, guardando tudo num dicionário que é depois retornado.

securitytrails.py

obter_dados_certificado

Esta função consulta a API da *SecurityTrails* para obter informações sobre certificados SSL válidos de um domínio, retornando detalhes como datas de validade, entidade emissora, tipo e tamanho da chave, impressões digitais , tudo retornado num dicionário.

obter_sub_dominios

Esta função consulta a API da *SecurityTrails* para obter todos os prefixos dos subdomínios ativos de um domínio fornecido e devolve uma lista desses subdomínios completos ou retorna uma lista vazia em caso de erro.

urlScan.py

consulta_urlScan

Esta função envia um pedido à API do *URLScan* para analisar um domínio ou URL e retorna o link da API com os resultados da análise, se o pedido for bem-sucedido.

obter_resultado

Esta função processa os resultados da análise feita pela API, devolvendo uma lista de servidores identificados e outra com as tecnologias detetadas do domínio, retornando duas listas com as respectivas informações ou *None* em caso de erro.

Diretório iocs_feed

Neste diretório estão presentes todos os *IOCs* para retornar informação através dos *endpoints* da API, contando com informação vinda do *OpenCTI* e respetivos dados.

Api_open_cti_country.py

CTI_pais

Esta função pesquisa os 10 ataques com maior nível de confiança relacionados com um país específico, utilizando a API do *OpenCTI*, e devolve as informações mais relevantes sobre cada ataque, como atacante, tipo de ataque, confiança e TLP.

CTI_pais2

Esta função permite pesquisar os ataques relacionados com um país específico, utilizando a API do *OpenCTI*. A função oferece a possibilidade de filtrar os ataques dentro de um intervalo de datas, com base nos parâmetros `mes_inicio` e `mes_fim`, caso sejam fornecidos. Os dados dos ataques são extraídos e organizados num dicionário, que contém as informações detalhadas sobre o atacante, o tipo de ataque, o alvo, a confiança, TLP (*Traffic Light Protocol*) e a data de criação.

Api_open_cti_ob.py

cti_Ips

Esta função recolhe todos os indicadores de endereços IP (IPv4 e IPv6) criados hoje na plataforma *OpenCTI*. Os dados são armazenados e retornados via dicionário.

cti_domains

Esta função recolhe todos os indicadores de domínios criados hoje na plataforma *OpenCTI*. Os dados são armazenados e retornados via dicionário.

cti_hashes

Esta função recolhe todos os indicadores do tipo *hashes* criados hoje na plataforma *OpenCTI*. Os dados são armazenados e retornados via dicionário.

OxSI_f33d.py

obter_user_api_key

Esta função devolve a chave necessária para conectar à API, caso esta esteja presente no ficheiro das chaves. Em caso de erro, retorna *None*.

obter_password_api_key

Esta função devolve a chave necessária para conectar à API, caso esta esteja presente no ficheiro das chaves. Em caso de erro, retorna *None*.

consultar0xsl_f33d_api

Esta função estabelece ligação à API do OxSI_f33d, recolhe os dados da última semana e retorna-os numa lista de *strings*.

consultar0xsl_f33d

Esta função estabelece ligação à API do 0xSI_f33d, recolhe os dados de um determinado período, e permite opcionalmente indicar um título ou URL a ser pesquisado, retornando-os numa lista de *strings* com os dados encontrados.

guardar_lista

Esta função guarda os dados retornados obtidos num ficheiro.

Monitorizador.py

Este script é o responsável pela API REST desenvolvida em *Flask*, que fornece diversos endpoints, como análise de ameaças, verificação de vulnerabilidades (CVEs), *feed* de inteligência de ameaças, informações relacionadas a IPs, domínios e *dataleaks*. Cada função no monitorizador.py é responsável por um *endpoint* (rota) da API. Abaixo está uma breve descrição de cada função:

run_0xSI_f33d

Esta função consulta o *feed* do projeto 0xSI, desenvolvido e suportado pela SegurancaInformatica.pt. Os dados retornados correspondem, por padrão, à última semana (esse período pode ser ajustado na função `consultar0xsl_f33d_api()`). O *feed* inclui URLs reportadas por utilizadores portugueses, relacionadas com campanhas de *phishing* e *malware*. Os dados são retornados em formato JSON através do endpoint: `/0xSI_feed/[GET]`.

run_cti

Esta função permite consultar dados da plataforma *OpenCTI* conforme o tipo de scan especificado no *endpoint*, aceitando um JSON como entrada para configuração dos parâmetros. Estão disponíveis quatro tipos de scan: TOP10, que é o único que requer um parâmetro de entrada (o país) e retorna os 10 ataques com maior nível de confiança associados a esse local; *hashes*, que recolhe todos os indicadores baseados em *hashes* do dia atual; *domain*, que consulta indicadores do tipo domínio; e *ips*, que consulta indicadores do tipo IP. Todos os resultados são devolvidos em formato JSON através do *endpoint*: `/cti/TypeScan/[POST]`.

Cti

Esta função oferece as mesmas funcionalidades da *run_cti(TypeScan)*, com exceção do tipo de scan TOP10, que não é suportado nesta versão e, portanto, não permite a consulta por país. A principal diferença é que esta função utiliza um *endpoint* do tipo GET, permitindo que seja acedida diretamente

através do navegador. Os resultados mantêm o formato da função *run_cti* e podem ser obtidos através do endpoint: */ctiweb/TypeScan[GET]*.

pais,pais2, pais3 ,pais4

Estas quatro funções partilham a mesma funcionalidade do tipo de scan TOP10 da função *run_cti()*, mas estão disponíveis através de endpoints do tipo GET, permitindo o seu acesso direto via browser. A principal diferença é que estas variantes permitem, opcionalmente, definir um limite temporal: o parâmetro *typeScan2* indica o mês de início e *typeScan3* o mês de fim da consulta. Caso não sejam fornecidos, não é aplicado qualquer limite temporal, sendo retornados todos os resultados disponíveis. Os resultados têm o mesmo formato da função *run_cti()* e podem ser obtidos através dos seguintes endpoints: */ctipais/typeScan/typeScan2/typeScan3/[GET]*, */ctipais/typeScan/typeScan2/[GET]*, */ctipais/[GET]*.

run_cvescript

Esta função permite consultar dados sobre vulnerabilidades CVE através do módulo *exploitdb*. Aceita um JSON com uma lista de identificadores CVE como entrada e devolve os resultados em formato JSON. As consultas são realizadas via o endpoint */CVE/[POST]*.

run_cve

Esta função permite consultar dados sobre vulnerabilidades CVE utilizando a API oficial do NVD (*National Vulnerability Database*). Aceita um JSON contendo uma lista de identificadores CVE como entrada e devolve os resultados em formato JSON. Cada resultado inclui uma descrição resumida da vulnerabilidade, o CVSS (*Common Vulnerability Scoring System*), os valores de severidade ,bem como informação adicional relevante sobre o tipo de falha (fraude, execução remota, etc.). As consultas são efetuadas através do endpoint */pesquisar-cve/[POST]*.

run_lookupScript

Esta função permite consultar informações associadas a emails, nomes de utilizador, domínios e endereços IP através da API do *LeakLookup*. A entrada é feita em formato JSON, indicando o tipo de entidade a consultar (por exemplo: email, username, domínio ou ip) e o respetivo valor. Os resultados, também em formato JSON, são obtidos através do endpoint */LOOKUP/[POST]*.

run_monitorizador

Esta função permite consultar domínios e endereços IP, utilizando dados processados pelos ficheiros *dom_checker.py* e *ips.py*, os quais serão explicados mais à frente neste documento. Suporta dois tipos

de scan: IP ou DOM. A entrada, composta pelos IPs ou domínios a analisar, deve ser fornecida em formato JSON, e os resultados, também em formato JSON, são disponibilizados através do *endpoint* `/monitorizador/TypeScan[POST]`.

Monitorizador

Esta função oferece as mesmas funcionalidades da `run_monitorizador()`, com a principal diferença de que pode ser acedida diretamente através do browser, uma vez que utiliza o método GET. Os resultados são devolvidos em formato JSON e o acesso é feito através do *endpoint*: `/monitorizadorweb/TypeScan/Type_scan_web/[GET]`, onde o TypeScan é o tipo de scan a ser realizado e o *type_scan_web* representa o conteúdo a ser analisado (ips ou domínios).

Teste_conexao_api.py

Este script foi desenvolvido para tornar o teste de todos os endpoints da API o mais fácil possível, com exceção do monitorizadorweb. Desta forma, é possível fazer a requisição dos endpoints através de código, tornando mais fácil realizar as consultas.

Diagrama da Estrutura da API

A baixo encontra um diagrama que representa a estrutura da API, mapeando o conjunto de endpoints disponíveis, os módulos responsáveis pelo processamento da informação para esses endpoints e as tecnologias utilizadas.

Figura 9

Diagrama da Estrutura da API

