



**POLITÉCNICO  
DE LEIRIA**

ESCOLA SUPERIOR  
DE TECNOLOGIA  
E GESTÃO

# Extensão de OSINT Para Análise e Procura de Dados

Jorge Veloso de Campos Baptista Ferreira

Escola Superior de Tecnologia e Gestão  
Departamento de Engenharia Informática  
Mestrado em Cibersegurança e Informática Forense

Leiria, Julho 2025



# Extensão de OSINT Para Análise e Procura de Dados

**Jorge Veloso de Campos Baptista Ferreira**

*Student No. 2230463*

**Supervisor:** Marco António de Oliveira Monteiro

*Professor, Politécnico de Leiria*

Escola Superior de Tecnologia e Gestão  
Departamento de Engenharia Informática  
Mestrado em Cibersegurança e Informática Forense

*Projeto*

Leiria, Julho 2025



### **Extensão de OSINT Para Análise e Procura de Dados**

Copyright © 2025 - Jorge Veloso de Campos Baptista Ferreira, Escola Superior de Tecnologia e Gestão.

A presente dissertação é um trabalho original, elaborado exclusivamente para este fim, tendo sido devidamente citados todos os autores cujos estudos contribuíram para a sua elaboração. É permitida a sua reprodução parcial com indicação do autor e referência ao grau, ano letivo, instituição—*Politécnico de Leiria*—e data da defesa pública.



O presente trabalho beneficiou da utilização do modelo *IPLeiria-Thesis*.



# Agradecimentos

Primeiramente gostaria de agradecer a todas as pessoas que conheci ao percorrer do meu período acadêmico em especial ao meu colega, Miguel Campos por ajudar-me num período difícil e ao meu orientador, Marco António de Oliveira Monteiro por ajudar-me e ter-me dado ideias de como resolver os problemas relacionados ao desenvolvimento do projeto e ter-me dado motivação e orientação.

O meu agradecimento especial para a minha família por ter-me acompanhado e apoiado em tudo que precisava.



# Resumo

A digitalização e a ubiquidade da informação na sociedade atual intensificaram a necessidade de segurança da informação e proteção da privacidade. A **Open Source Intelligence (OSINT)** destaca-se como uma ferramenta essencial para a recolha e análise de dados públicos, contribuindo para a tomada de decisões em áreas como a cibersegurança, a investigação criminal e o jornalismo. Contudo, o aumento exponencial de dados disponíveis impõe desafios que exigem soluções mais avançadas. Neste contexto, os **Large Language Model (LLM)** surgem como recursos inovadores, capazes de automatizar e otimizar a análise de informação através do processamento de linguagem natural.

Este documento explora a integração entre **OSINT** e **LLM**, apresentando uma investigação detalhada sobre as suas aplicações e limitações na área da cibersegurança. Como contributo prático, é desenvolvido um projeto que combina ambas as abordagens, materializado na criação de uma extensão de navegador integrada com um *backend* funcional e com suporte a ferramentas **OSINT**. Este projeto demonstra o potencial desta conjugação para fortalecer a segurança digital e promover uma análise mais eficiente, automatizada e precisa de dados provenientes de fontes abertas.

**Palavras-Chave:** Cibersegurança , Extensões, LLM.



# Abstract

The digitalization and ubiquity of information in today's society have intensified the need for information security and privacy protection. **Open Source Intelligence (OSINT)** stands out as a key tool for collecting and analyzing publicly available data, supporting decision-making in fields such as cybersecurity, criminal investigations, and investigative journalism. However, the exponential growth of online data presents new challenges that demand more advanced solutions. In this context, **Large Language Model (LLM)** emerge as innovative resources capable of automating and optimizing information analysis through natural language processing.

This document explores the integration of **OSINT** and **LLM**, offering a detailed investigation into their applications and limitations within the cybersecurity domain. As a practical contribution, a project is developed that combines both approaches, culminating in the creation of a browser extension integrated with a functional *backend* and multiple **OSINT** tools. This work demonstrates the potential of combining **OSINT** and **LLM** to strengthen digital security and enable more efficient, automated, and accurate analysis of open-source data.

**Keywords:** Cibersecurity, Extensions, LLM.



# Declaração sobre o Uso de Inteligência Artificial

Reconheço a utilização do ChatGPT (<https://chatgpt.com>) para aperfeiçoar o tom académico e melhorar a precisão linguística deste documento, incluindo aspectos de gramática, pontuação e vocabulário.

## **Descrição da Utilização**

**Instrução 1:** Analisa o texto em português de Portugal, e corrige as falhas mais graves de ortografia e gramática.

**Resposta 1:** Texto original: gUstaria de agradecer aos meus colegas por me terem ajudado em tudo além disso gostaria de dar este exemplo de uso de chatGPD para corecao de texto

Texto corrigido: Gostaria de agradecer aos meus colegas por me terem ajudado em tudo. Além disso, gostaria de dar este exemplo de uso do ChatGPT para correção de texto.



# Conteúdo

<i>Lista de Figuras</i>	xiii
<i>Lista de Tabelas</i>	xvi
<i>Siglas</i>	xviii
<b>1 Introdução</b>	<b>1</b>
<b>2 OSINT</b>	<b>3</b>
2.1 Definição . . . . .	3
2.2 História . . . . .	4
2.3 Importância dos dados . . . . .	5
2.4 Ferramentas e Técnicas usadas no OSINT . . . . .	6
2.4.1 Shodan . . . . .	6
2.4.2 TheHarvester . . . . .	7
2.4.3 SpiderFoot . . . . .	8
2.4.4 Have I Been Pwned . . . . .	8
2.4.5 Google Dorks . . . . .	9
2.4.6 Wayback Machine . . . . .	9
2.4.7 Epieos . . . . .	9
<b>3 LLM</b>	<b>11</b>
3.1 Definição . . . . .	11
3.2 História . . . . .	12
3.3 Chatgpt vs LLAMA3 vs DeepSeek R1 . . . . .	12
3.3.1 ChatGPT (OpenAI) - 2022 . . . . .	14
3.3.2 LLaMA 3 (Meta) - 2024 . . . . .	15
3.3.3 DeepSeek R1 (DeepSeek) - 2025 . . . . .	15
3.4 Uso na Cibersegurança e no OSINT . . . . .	15
3.4.1 Análise de Ameaças e Detecção de Vulnerabilidades . . . . .	15
3.4.2 Análise de Malware e Inteligência de Ameaças . . . . .	16
3.4.3 Resposta a Incidentes e Automação de Tarefas . . . . .	16
3.4.4 Recolha de Dados . . . . .	16
3.4.5 Tradução . . . . .	16
3.5 Considerações Éticas no Uso de LLM na Cibersegurança . . . . .	16

3.6	OWASP Top 10 para Aplicações com LLM . . . . .	17
3.7	Problemas com o Uso de LLM na Cibersegurança . . . . .	18
3.8	Vibe Coding . . . . .	18
3.8.1	Código Vulnerável . . . . .	18
3.8.2	Falta de Otimização . . . . .	18
3.8.3	Dependência Excessiva da IA . . . . .	19
<b>4</b>	<b>Projeto</b> . . . . .	<b>21</b>
4.1	Projeto . . . . .	21
4.1.1	Descrição . . . . .	21
4.1.2	Procedimentos e Requisitos . . . . .	22
4.1.3	Análise Prévia . . . . .	23
4.1.4	Resultados Esperados . . . . .	24
4.1.5	Contribuições Previstas . . . . .	24
4.1.6	Projetos Semelhantes . . . . .	24
4.1.7	Diferenças . . . . .	24
4.2	Metodologia . . . . .	25
4.2.1	Kanban . . . . .	25
4.2.2	DevSecOps . . . . .	26
4.3	Arquitetura e Tecnologias . . . . .	27
4.3.1	Arquitetura . . . . .	27
4.3.2	Tecnologias . . . . .	31
<b>5</b>	<b>Implementação</b> . . . . .	<b>35</b>
5.1	Extensões Browser . . . . .	35
5.1.1	Firefox vs Chrome . . . . .	35
5.1.2	Mockups . . . . .	38
5.1.3	Desenvolvimento da Extensão . . . . .	39
5.2	<i>Backend</i> . . . . .	41
5.2.1	Endpoints . . . . .	41
5.2.2	API Keys . . . . .	43
5.2.3	Integração com outras ferramentas . . . . .	43
5.2.4	Rate Limiter . . . . .	49
5.3	LLM . . . . .	49
5.3.1	Integração . . . . .	50
5.3.2	Dificuldades encontrados com a LLM . . . . .	52
5.4	Segurança . . . . .	54
5.4.1	Como garantir a segurança durante o desenvolvimento . . . . .	54
5.4.2	Segurança garantida pelos manifestos . . . . .	57
5.4.3	Criação da <i>Pipeline</i> . . . . .	58
5.5	Publicação . . . . .	59
5.5.1	Criação do Dockerfile e Configuração do Docker Compose . . . . .	59

5.5.2	Execução do <i>Backend</i> num serviço <i>self hosted</i> . . . . .	63
5.5.3	Publicação do <i>Backend</i> na AWS e GCP . . . . .	65
5.6	Testes . . . . .	66
5.6.1	Teste funcional do <code>/start-scan</code> . . . . .	66
5.6.2	Teste funcional do <code>/tools</code> . . . . .	67
5.6.3	Teste de Segurança . . . . .	68
5.7	Resultados . . . . .	68
5.8	Trabalho Futuro . . . . .	69
5.8.1	Sistema de Seleção Automática de APIs . . . . .	69
5.8.2	Migração para Manifest V3 no Firefox . . . . .	69
5.8.3	Integração com Modelos de IA Mais Avançados . . . . .	69
5.9	Ideias para Projetos . . . . .	69
5.9.1	Conversor Automático de Output CLI para JSON . . . . .	69
5.9.2	Extensão OSINT para Análise de Websites . . . . .	70
5.9.3	Sistema de Monitorização Contínua de Vulnerabilidades a partir do Github Actions . . . . .	70
<b>6</b>	<b>Conclusão</b>	<b>71</b>
	<i>Bibliography</i>	73
	<b>Anexos</b>	
<b>A</b>	<b>Código gerado - ChatGPT vs DeepSeek R1 vs Ollama</b>	<b>80</b>
<b>B</b>	<b>Código gerado pelo LLAMA3 e Pipeline configurada</b>	<b>88</b>



# Lista de Figuras

2.1	Website do Shodan a mostrar informação sobre um IP . . . . .	7
2.2	TheHarvester CLI . . . . .	7
2.3	Imagem do Spiderfoot para criar novo scan . . . . .	8
2.4	Página Principal do Have I Been Pwned . . . . .	8
2.5	Exemplo de um google Dork a procurar títulos com "you are an idiot virus" . . . . .	9
2.6	Página principal Wayback Machine . . . . .	9
2.7	Exemplo de scan do Epieos . . . . .	10
4.1	Imagem do ambiente Trello . . . . .	26
4.2	Dentro dos cartões . . . . .	26
4.3	Diagrama da Arquitetura . . . . .	28
5.1	Imagem com os mockups da extensão . . . . .	38
5.2	Design do logo final . . . . .	39
5.3	Erro que aparece no Chrome . . . . .	41
5.4	Exemplo Código do Firefox . . . . .	41
5.5	Exemplo Código do Chrome . . . . .	41
5.6	Código para executar o subprocesso do Sherlock . . . . .	44
5.7	Comparação entre o <i>output</i> proveniente do CLI e o <i>output</i> convertido para JSON. . . . .	44
5.8	Código para executar o subprocesso do Holehe . . . . .	45
5.9	Comparação entre o <i>output</i> proveniente do CLI e o <i>output</i> convertido para JSON. . . . .	45
5.10	Código para executar o subprocesso do Docker do phoneinfoga . . . . .	46
5.11	Comparação entre o <i>output</i> proveniente do Docker e o <i>output</i> convertido para JSON. . . . .	47
5.12	Output do DuckDuckGo . . . . .	48
5.13	Seleção da opção "Automated Scan" na interface do OWASP ZAP . . . . .	56
5.14	Introdução do URL e ativação do ataque através do botão "Attack" no OWASP ZAP . . . . .	56
5.15	Resultado do comando <code>ipconfig</code> . . . . .	63
5.16	Acesso às definições de rede. . . . .	64
5.17	Acesso às propriedades da rede. . . . .	64

5.18 Edição da configuração de IP. . . . .	64
5.19 Configuração manual do IP estático. . . . .	65
5.20 Website Criado . . . . .	66
5.21 Interface da extensão com a inserção da <i>API Key</i> e seleção do modelo. . . .	67
5.22 Imagem da extensão com os dados encontrados pelo LLM. . . . .	67

# Lista de Tabelas

4.1	Tecnologias e as suas respectivas Versões . . . . .	31
-----	---	----



# Siglas

<b>AI</b>	Artificial Intelligence. (p. 11, 22)
<b>ANSI</b>	American National Standards Institute. (p. 46)
<b>API</b>	Application Programming Interface. (p. 14, 22, 24, 28, 29, 31, 32, 35, 36, 41, 43, 50, 63, 69, 70)
<b>AWS</b>	Amazon Web Services. (p. 59, 65)
<b>BBC</b>	British Broadcasting Corporation. (p. 5)
<b>CLI</b>	Command-Line Interface. (p. 7, 43, 45, 46, 69)
<b>CSP</b>	Content Security Policy. (p. 58)
<b>CSRF</b>	Cross-Site Request Forgery. (p. 17)
<b>CSS</b>	Cascading Style Sheets. (p. 32, 36, 39, 48)
<b>CVE</b>	Common Vulnerabilities and Exposures. (p. 7, 57)
<b>DAST</b>	Dynamic Application Security Testing. (p. 18, 22, 24, 33, 54, 55, 68, 71)
<b>DDOS</b>	Distributed Denial of Service. (p. 49)
<b>DevSecOps</b>	Development, Security, and Operations. (p. 21, 22, 25, 26)
<b>DOM</b>	Document Object Model. (p. 37)
<b>DOS</b>	Denial of Service. (p. 49, 66, 68)
<b>GAN</b>	Generative Adversarial Networks. (p. 12)
<b>GCP</b>	Google Cloud Platform. (p. 59, 65)
<b>GDPR</b>	General Data Protection Regulation. (p. 23)
<b>HTML</b>	HyperText Markup Language. (p. 23, 32, 36, 39, 41, 48, 51, 66)
<b>HTTP</b>	Hypertext Transfer Protocol. (p. 28, 29, 43, 48, 65)
<b>IOC</b>	Indicator of Compromise. (p. 16)
<b>IP</b>	Internet Protocol. (p. 6, 8, 24, 49, 63)
<b>IPv4</b>	Internet Protocol version 4. (p. 63)
<b>JSON</b>	JavaScript Object Notation. (p. 22, 24, 28, 29, 41–48, 52, 53, 67–69)

<b>LLM</b>	Large Language Model. (p. <i>iii, v, xiv, 1, 2, 11, 12, 15–19, 21, 22, 31, 35, 41, 49–53, 66–69, 71</i> )
<b>MX</b>	Mail Exchanger. (p. <i>24</i> )
<b>NLP</b>	Natural Language Processing. (p. <i>12</i> )
<b>OSINT</b>	Open Source Intelligence. (p. <i>iii, v, 1–6, 21, 23, 24, 42, 68, 70, 71</i> )
<b>POC</b>	Proof of Concept. (p. <i>21</i> )
<b>regex</b>	regular expression. (p. <i>44–46</i> )
<b>REST</b>	Representational State Transfer. (p. <i>27</i> )
<b>SAST</b>	Static Application Security Testing. (p. <i>18, 22–24, 26, 33, 54, 58, 71</i> )
<b>SCA</b>	Software Composition Analysis. (p. <i>18, 27</i> )
<b>SSH</b>	Secure Shell. (p. <i>65</i> )
<b>TTP</b>	Tactics, Techniques, and Procedures. (p. <i>16</i> )
<b>UI</b>	User Interface. (p. <i>22, 28</i> )
<b>URL</b>	Uniform Resource Locator. (p. <i>22–24, 40, 41, 44, 46, 48, 69</i> )
<b>XSS</b>	Cross-Site Scripting. (p. <i>17, 55, 58</i> )

# 1

## Introdução

A sociedade contemporânea encontra-se profundamente marcada pela digitalização e pela ubiquidade da informação. O acesso facilitado a dados, a proliferação de dispositivos conectados e a crescente dependência de plataformas digitais têm transformado não apenas a forma como comunicamos e trabalhamos, mas também a maneira como a informação é recolhida, analisada e utilizada. Neste contexto, a segurança da informação e a proteção da privacidade emergem como preocupações centrais, tanto para indivíduos como para organizações.

O conceito de Open Source Intelligence (**OSINT**) surge como uma resposta a estes desafios, permitindo a recolha e análise de dados provenientes de fontes abertas e acessíveis ao público. O **OSINT** tem vindo a ganhar destaque em diversos domínios, desde a cibersegurança à investigação criminal, passando pelo jornalismo de investigação e pela análise de riscos empresariais. A sua importância reside na capacidade de transformar grandes volumes de dados dispersos em conhecimento útil, facilitando a tomada de decisões informadas e a antecipação de ameaças.

Contudo, a crescente quantidade e complexidade dos dados disponíveis online colocam novos desafios à eficácia das metodologias tradicionais de **OSINT**. Neste sentido, os Large Language Models (**LLM**) assumem um papel revolucionário ao proporcionar uma compreensão avançada, processamento e geração de linguagem natural. Modelos como ChatGPT, LLaMA 3 e DeepSeek R1 automatizam tarefas anteriormente intensivas em intervenção humana, acelerando a análise e melhorando a extração e correlação de dados.

É neste enquadramento que surge o Xerifint, uma extensão de navegador desenvolvida no âmbito deste projeto, que permite fazer scan a páginas web utilizando as capacidades de um **LLM** para identificar, extrair e interpretar automaticamente informações relevantes. A Xerifint combina os princípios do **OSINT** com a inteligência avançada de modelos de linguagem. Esta solução procura não apenas otimizar o processo de recolha e análise de dados abertos, mas também tornar esse processo mais seguro, eficiente e centrado na privacidade do utilizador.

Este trabalho tem como objetivo explorar a conjugação entre **OSINT** e **LLM**, ana-

lisando as suas aplicações, potencialidades e desafios, com foco especial na cibersegurança e na concretização prática desta integração através do desenvolvimento da extensão Xerifint.

O presente documento está estruturado em seis capítulos.

No Capítulo 1, *Introdução*, apresenta-se o enquadramento geral do tema, os objetivos do trabalho, a motivação para o seu desenvolvimento e a metodologia seguida. O Capítulo 2, *OSINT*, aprofunda o enquadramento teórico da inteligência de fontes abertas, destacando as suas principais ferramentas e aplicações. Seguidamente, o Capítulo 3, *LLM*, aborda os modelos de linguagem de grande escala, analisando as suas capacidades no contexto do **OSINT**, bem como os principais desafios que lhes estão atualmente associados. No Capítulo 4, *Projeto*, é descrita a conceção da solução desenvolvida, incluindo os seus objetivos técnicos e funcionais, bem como a metodologia adotada para a sua concretização. O Capítulo 5, *Implementação*, detalha os aspetos técnicos do desenvolvimento da extensão de navegador *Xerifint*, incluindo o backend, as estratégias de segurança implementadas e os testes realizados e ideias de trabalho futuro. Por fim, o Capítulo 6, *Conclusão*, apresenta uma análise crítica dos resultados obtidos, refletindo sobre os objetivos alcançados.

O projeto desenvolvido pretende demonstrar que a integração de **OSINT** e **LLM** pode contribuir significativamente para a construção de uma cibersegurança mais robusta, inclusiva e transparente, promovendo a democratização do acesso à informação e a proteção dos direitos dos utilizadores num mundo cada vez mais digital.

# 2

## OSINT

Neste capítulo, será abordado o conceito de **Open Source Intelligence (OSINT)**, explorando a sua definição, origem histórica, importância e evolução ao longo do tempo. Para além disso, será destacada a crescente relevância dos dados na atual sociedade digital e apresentadas as principais ferramentas e tecnologias utilizadas no mercado.

### 2.1 Definição

No nosso dia a dia, a utilização de tecnologias tem vindo a crescer rapidamente, transformando a forma como vemos o nosso mundo e como interagimos com ele. Este avanço permitiu que as pessoas realizassem as suas tarefas de forma mais fácil e eficiente. Esta evolução trouxe também um aumento significativo na quantidade de dados pessoais que deixamos disponíveis, muitas vezes sem consciência ou noção dos danos que podem causar. Essas informações podem incluir, por exemplo, a nossa localização, interações nas redes sociais e até dados pessoais sensíveis, como o nosso nome e NIF.

Entidades como empresas, organizações, *data brokers* e indivíduos com intenções maliciosas — como, por exemplo, *hackers* — podem aceder a essa informação que deixamos disponível, utilizando-a de forma maliciosa ou para a criação de anúncios personalizados. Mas como é que essas entidades conseguem obter informações sobre nós? A resposta está no **OSINT**.

O **Open Source Intelligence (OSINT)** é uma metodologia de recolha e processamento de informações que utiliza fontes abertas e acessíveis ao público, como jornais, estações de rádio, sites da Internet, redes sociais e artigos académicos. Ao contrário de outras formas de inteligência, que dependem de fontes secretas ou confidenciais, o **OSINT** baseia-se em dados amplamente disponíveis, mas que muitas vezes podem ser explorados de formas inesperadas.

Atualmente, o **OSINT** desempenha um papel crucial em várias áreas, incluindo a segurança nacional, a cibersegurança — por exemplo, para que as empresas verifiquem se um e-mail recebido é fraudulento —, a investigação criminal, como no caso da Polícia Judiciária utilizar técnicas para encontrar mais informações sobre um suspeito, o

jornalismo, na recolha de dados sobre eventos, e até no setor empresarial, tanto para proteção como para análise de perfis de clientes-alvo.

Com o crescimento dos dados acessíveis na Internet e o aumento exponencial da utilização de redes sociais como o Instagram e o Facebook, a quantidade de informações públicas disponíveis aumentou drasticamente, já que muitas pessoas têm cada vez menos cuidado com o que partilham online. Esta realidade torna o OSINT numa ferramenta extremamente poderosa para a recolha e análise de dados, sendo utilizada para monitorizar e identificar tendências, avaliar riscos, detetar ameaças e realizar investigações aprofundadas [24].

No entanto, o OSINT não está isento de controvérsias, podendo ser considerado uma zona cinzenta (*grey area*) do ponto de vista ético e legal. Quando utilizado de forma intrusiva ou para a obtenção de informações que não são verdadeiramente públicas — como é o caso de dados resultantes de *data breaches* —, o OSINT pode facilmente ultrapassar os limites legais e éticos. Neste contexto, práticas como a *social engineering* (engenharia social), que envolvem a manipulação de pessoas para obtenção de informações confidenciais, podem ser classificadas como crimes, sendo consideradas formas de ataque cibernético.

O uso indevido de informações obtidas através de OSINT pode causar danos a indivíduos, organizações e até à segurança nacional, resultando em consequências jurídicas e de reputação extremamente graves.

Com isso em mente, o OSINT pode, caso seja utilizado de forma indevida, enquadrar-se nos seguintes crimes:

- **Burla**, punível nos termos do **Artigo 217.º do Código Penal Português**, caso o objetivo seja enganar a vítima e obter uma vantagem ilícita [11];
- **Acesso ilegítimo**, previsto na **Lei do Cibercrime — Lei n.º 109/2009, Capítulo II, Artigo 6.º**, se, através de técnicas de OSINT, o agente aceder de forma não autorizada a sistemas ou dados [30];
- **Extorsão**, punível ao abrigo do **Artigo 223.º do Código Penal Português**, no caso de o agente utilizar a informação recolhida para extorquir a vítima [12];

Assim, embora o OSINT constitua uma ferramenta de enorme valor, a sua aplicação requer uma abordagem ética e rigorosa, bem como uma compreensão clara dos limites legais que regulam o acesso e a utilização de dados públicos.

## 2.2 História

Embora a sigla OSINT tenha sido mencionada pela primeira vez em 1990, num artigo escrito por Robert Steele, intitulado “1990 Intelligence in the 1990’s: Recasting National Security in a Changing World (AIJ)” [49], foi apenas em 1993, numa edição especial da *American Intelligence Journal*, intitulado “National Intelligence and Open Source: From School House to White House” [16], que ganhou maior destaque no campo da inteligência. No entanto, a importância da informação já era reconhecida há milénios, como

destaca Westcott: “Os exploradores vikings, os legionários romanos e os comerciantes da Rota da Seda estavam familiarizados com o processo de observar o mundo à sua volta, tentar compreendê-lo e depois explicar o que sabiam aos outros” [6].

Por outras palavras, a recolha de informações e a sua partilha têm sido fundamentais para a evolução das sociedades e para o desenvolvimento de estratégias. Um exemplo notável é o de Aníbal Barca, que utilizou o seu conhecimento sobre o temperamento impulsivo do general Caio Flamínio para emboscar as tropas romanas na Batalha do Lago Trasimeno [25].

Outro exemplo marcante da relevância da informação ocorreu durante a Segunda Guerra Mundial, quando, em 1939, foi criado no Reino Unido o *BBC Monitoring Service*. Esta organização desempenhou um papel crucial ao longo dos seus quase 85 anos de existência, fornecendo uma avaliação contínua de notícias, informações, propaganda e outras fontes relevantes. O seu trabalho permitiu analisar as intenções de diversos países e compreender como os meios de comunicação eram utilizados como canais de diplomacia internacional. Ao monitorizar emissões de rádio e outras formas de comunicação, o *BBC Monitoring Service* teve um papel essencial na obtenção de dados estratégicos e na análise das mensagens disseminadas, contribuindo significativamente para a segurança nacional e para a formulação de políticas externas durante períodos de tensão política e militar [27].

Atualmente, o *OSINT* evoluiu de forma exponencial, impulsionado pelo crescimento da Internet e pelo aumento do uso de redes sociais como o Instagram e o Facebook.

Um dos casos em que essa evolução se tornou evidente foi o atentado de Boston em 2013, que envolveu Dzhokhar Tsarnaev, um dos responsáveis pelo ataque. Para além da combinação de ferramentas tradicionais, como câmaras de vigilância e depoimentos de testemunhas, a polícia recorreu também a vídeos e fotografias publicados nas redes sociais para identificar Tsarnaev [3]. Estas imagens, captadas e partilhadas por cidadãos comuns, revelaram-se fundamentais para a sua captura, demonstrando como o *OSINT* pode ser uma ferramenta crucial em investigações criminais.

## 2.3 Importância dos dados

É extremamente importante saber onde se encontra a nossa informação e os nossos dados. Com base nas informações que decidimos disponibilizar online, é possível construir uma pegada digital, a qual pode revelar uma vasta gama de dados sobre nós, como os nossos gostos, preferências, identidade e até detalhes sobre a nossa família e vida pessoal.

A pegada digital pode ser dividida em dois tipos:

**Pegadas ativas** As pegadas ativas são aquelas que criamos de forma consciente, como, por exemplo, quando fazemos uma publicação nas redes sociais ou partilhamos uma fotografia com amigos [50].

**Pegadas passivas** As pegadas passivas referem-se à recolha de informações sobre o utilizador sem o seu conhecimento ou consentimento explícito. Já pensaste que, quando procuras por um produto que estás a considerar comprar, pouco tempo depois comesças a ver anúncios relacionados com esse produto? Este é um exemplo claro de pegada passiva. As redes sociais utilizam os teus gostos e comportamentos online para te mostrar conteúdo personalizado. Outro exemplo seria a presença de *metadados* numa fotografia, os quais podem revelar informações como a localização, o modelo do dispositivo utilizado ou a data em que a imagem foi captada — tudo isto sem que te apercebas [50].

Como mencionado anteriormente, a pegada digital pode ser utilizada de várias formas, tanto por empresas com o objetivo de direcionar publicidade personalizada, como por cibercriminosos que procuram explorar esses dados para fins maliciosos. Por exemplo, *hackers* podem utilizar estas informações para realizar tentativas de roubo de identidade, acedendo às nossas contas e fazendo-se passar por nós — o que pode ter consequências graves.

Por isso, é essencial estarmos atentos à forma como gerimos e protegemos a nossa informação online, de modo a evitar que esta seja utilizada de forma indevida.

## 2.4 Ferramentas e Técnicas usadas no OSINT

Foram desenvolvidas diversas ferramentas que auxiliam na realização de OSINT. Estas ferramentas desempenham um papel fundamental, ao proporcionarem aos investigadores formas mais eficazes e eficientes de obter informações relevantes num meio digital cada vez mais complexo.

Algumas dessas ferramentas incluem:

- **Shodan**, que permite a análise de informação de um website;
- **TheHarvester**, que se foca na investigação e mapeamento da infraestrutura digital;
- **SpiderFoot**, que, além de permitir a análise de websites, também possibilita a recolha de informações sobre utilizadores;
- **Have I Been Pwned**, que tem como foco principal a verificação de palavras-passe comprometidas;
- **Google Dorks**, utilizado para pesquisas específicas de informação;
- **Wayback Machine**, que permite visualizar o histórico de alterações de páginas web.
- **Epieos**, que se concentra na recolha de dados associados a endereços de e-mail.

### 2.4.1 Shodan

O *Shodan* é uma ferramenta focada na análise de dispositivos conectados à Internet, como câmaras, routers e servidores. Permite encontrar informações sobre esses dispositivos, tais como endereços **Internet Protocol (IP)**, domínios, localização geográfica



### 2.4.3 SpiderFoot

O *SpiderFoot* é uma ferramenta automatizada que recolhe informações sobre domínios, endereços IP e outras entidades online. Realiza pesquisas em diversas fontes públicas para mapear vulnerabilidades, relações entre entidades e outros dados relevantes [47].

Na Figura 2.3, apresenta-se a ferramenta *SpiderFoot* a criar um novo scan para uma informação escolhida pelo utilizador.

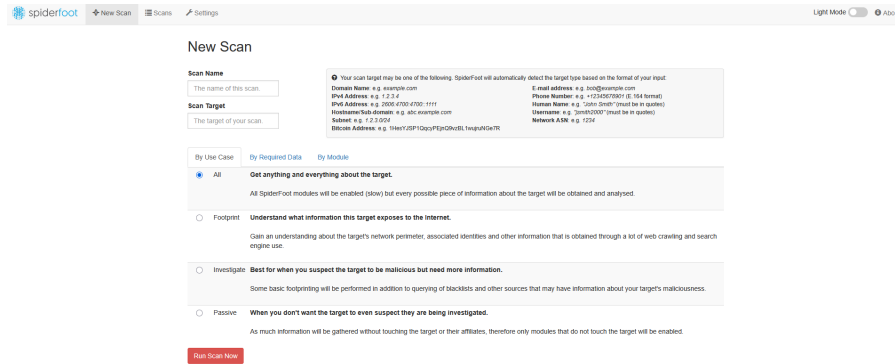


Figura 2.3: Imagem do Spiderfoot para criar novo scan

### 2.4.4 Have I Been Pwned

Esta ferramenta permite verificar se um determinado endereço de e-mail foi comprometido em algum *data breach* (violação de dados). Caso um e-mail esteja associado a um incidente de segurança, é altamente recomendável alterar imediatamente a palavra-passe, de forma a proteger a conta. No entanto, apesar da gravidade da situação, muitas pessoas negligenciam esta ação, deixando as suas contas vulneráveis a ataques ou acessos não autorizados [21].

Na Figura 2.4, apresenta-se o website do serviço *Have I Been Pwned*, utilizado para verificar se uma determinada palavra-passe foi exposta em algum *data breach*.

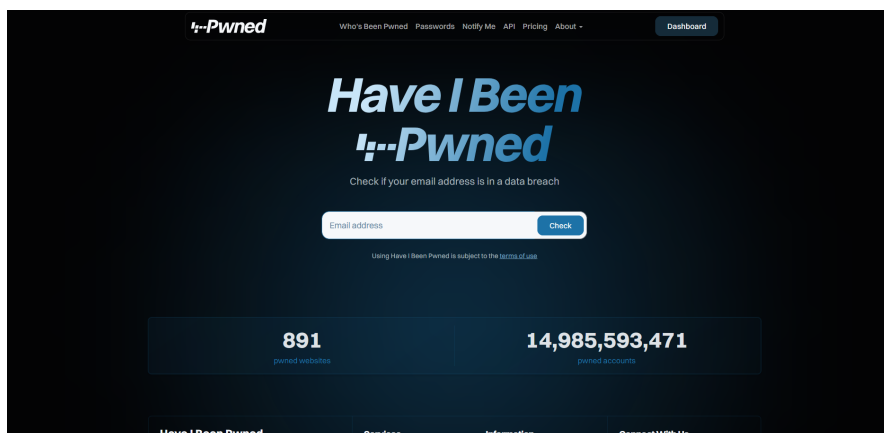


Figura 2.4: Pagina Principal do Have I Been Pwned

### 2.4.5 Google Dorks

O *Google Dorks* é uma técnica que utiliza operadores avançados no motor de busca Google para localizar informações específicas e, por vezes, sensíveis — como ficheiros privados, páginas de autenticação e dados expostos. Esta técnica é frequentemente utilizada, por exemplo, por autoridades policiais para localizar conteúdos relevantes, como imagens, que possam auxiliar investigações criminais [54]. Na Figura 2.5, apresenta-se um exemplo da execução de um *Google Dork* que procura títulos contendo a expressão “*you are an idiot virus*”.

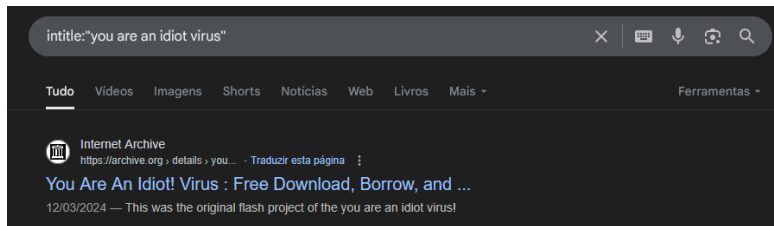


Figura 2.5: Exemplo de um google Dork a procurar títulos com “*you are an idiot virus*”

### 2.4.6 Wayback Machine

A *Wayback Machine* é uma ferramenta *open source* que permite visualizar o histórico de alterações de um website ao longo do tempo. Com esta ferramenta, é possível aceder a versões anteriores de páginas e conteúdos, incluindo publicações que tenham sido apagadas ou alteradas [1].

Essas informações eliminadas ou modificadas podem ser úteis, por exemplo, quando um utilizador apaga uma publicação, permitindo ainda assim visualizar o conteúdo original.

Na Figura 2.6, apresenta-se a página principal, onde é possível inserir o URL de uma página para consultar o seu histórico.

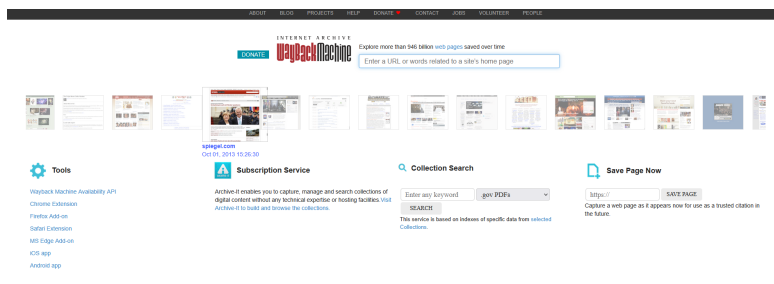


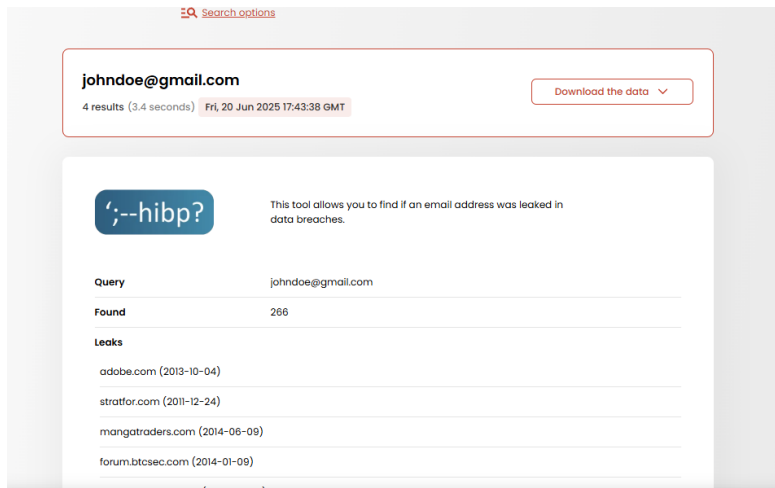
Figura 2.6: Página principal Wayback Machine

### 2.4.7 Epieos

O *Epieos* é uma ferramenta eficaz e amplamente utilizada para a pesquisa de informações associadas a endereços de e-mail. Permite identificar contas vinculadas a um

determinado e-mail em diversas plataformas, como Google, Skype, Chess.com, entre muitas outras [2].

Na Figura 2.7, apresenta-se um exemplo de uma análise ao e-mail `jhondoe@gmail.com`, na qual é possível verificar que esse endereço foi encontrado em 266 *data breaches*, conforme reportado pelo serviço *Have I Been Pwned*.



**Figura 2.7:** Exemplo de scan do Epieos

# 3

## LLM

Neste capítulo, será abordado o **Large Language Model (LLM)**, explorando a sua definição, a origem histórica e os três principais modelos atualmente em destaque. Serão analisadas as aplicações dos **LLM** na área da cibersegurança, com especial atenção às suas potencialidades e limitações. Além disso, serão discutidas as preocupações éticas associadas ao seu uso, bem como as principais vulnerabilidades identificadas, com base na lista da OWASP e na abordagem do Vibe Coding.

### 3.1 Definição

Com o rápido crescimento da **Artificial Intelligence (AI)** ou Inteligência Artificial, torna-se cada vez mais importante compreender as suas principais categorias e capacidades. Uma dessas categorias é a dos **Large Language Model (LLM)**, ou Grandes Modelos de Linguagem. Segundo a IBM, um **LLM** é um tipo de modelo de IA treinado com imensas quantidades de dados, o que lhe permite não só compreender como gerar linguagem natural, mas também outros tipos de conteúdos, como código, imagens ou até música, com o objetivo de executar uma vasta gama de tarefas complexas [51].

Estes modelos têm como principal objetivo compreender e gerar texto de forma semelhante à interação humana, baseando-se nos dados com que foram treinados. Através do processamento de grandes volumes de informações textuais, os **LLM** conseguem identificar padrões, aprender nuances de linguagem e, assim, produzir respostas contextualmente relevantes, seja para tradução, redação, resposta a perguntas ou até para automação de processos.

A evolução dos **LLM** também está a impulsionar a inovação em diversas áreas, como o atendimento ao cliente, a criação de conteúdo, o ensino personalizado e a pesquisa científica. Com o tempo, espera-se que esses modelos se tornem cada vez mais sofisticados, melhorando a sua capacidade de compreensão e geração de texto, e até permitindo uma interação mais fluida e humana com as máquinas.

## 3.2 História

Para compreendermos a história dos **Large Language Model (LLM)**, precisamos recuar no tempo e perceber como a linguística e a tecnologia se cruzaram ao longo dos anos.

Tudo começa no início do século XX, com o trabalho de Ferdinand de Saussure, um linguista que lançou as bases para a compreensão das línguas como sistemas. Após a sua morte em 1913, os seus colegas organizaram os seus escritos e, em 1916, publicaram o livro *Cours de Linguistique Générale*, que viria a influenciar o desenvolvimento do **Natural Language Processing (NLP)**, ou, em português, Processamento de Linguagem Natural [51].

Após a Segunda Guerra Mundial, as necessidades de tradução entre línguas, com o intuito de facilitar a comunicação internacional, deram um grande impulso à investigação em **NLP**. Face à dificuldade de traduzir diversas línguas, surgiu o conceito de utilizar linguagens mais estruturadas, como a matemática, para começar a desenvolver máquinas de tradução [51].

Na década de 1950, Arthur Samuel, da IBM, criou um programa que aprendia a jogar damas, denominado *Game of Checkers*, o que levou ao surgimento do conceito de *aprendizagem automática (machine learning)*. Mais tarde, em 1958, Frank Rosenblatt criou o Perceptron, a primeira rede neuronal artificial, que se tornaria a base da inteligência artificial moderna [51].

Na década de 1960, Joseph Weizenbaum desenvolveu o programa ELIZA, que utilizava **NLP** para simular uma conversa, demonstrando que as pessoas podiam acreditar que estavam a interagir com uma máquina inteligente [51].

Nos anos 80, graças ao aumento da capacidade de processamento e à transição para modelos estatísticos, a **NLP** foi significativamente impulsionada. A criação da Internet, com a World Wide Web, em 1989, também foi crucial, ao disponibilizar uma enorme quantidade de dados que ajudou no treino dos modelos linguísticos [51].

Nos anos 90, a utilização de redes de *deep learning* trouxe uma grande evolução para os **LLM**. Isso permitiu que os modelos se tornassem mais sofisticados e precisos. Em 2014, surgiu a técnica **Generative Adversarial Networks (GAN)**, que permitiu melhorar a geração de dados e imagens, influenciando também o desenvolvimento de chatbots [51].

O grande avanço aconteceu em 2022, com o lançamento do ChatGPT pela OpenAI. Este modelo de linguagem foi capaz de manter conversas de forma natural e realizar tarefas complexas, como escrever textos e programar, dando origem aos LLM tal como os conhecemos hoje [51].

## 3.3 Chatgpt vs LLAMA3 vs DeepSeek R1

Atualmente, existem inúmeros Modelos de **LLM** disponíveis no cenário da inteligência artificial. Para esta análise, selecionei três modelos: o ChatGPT, desenvolvido pela OpenAI [8]; o LLaMA 3, criado pela Meta [31]; e o DeepSeek R1, desenvolvido pela

DeepSeek [13]. Cada um destes modelos possui características únicas, respondendo a diferentes necessidades e públicos.

As avaliações seguintes foram realizadas por mim, sem recurso a qualquer *benchmark* formal. Os critérios utilizados nesta análise basearam-se na criação de diferentes tipos de *output*, como: geração de texto (por exemplo, histórias), criação de código funcional e sem alucinações, aplicabilidade em contextos contemporâneos e facilidade de implementação.

Para este fim, os *inputs* foram desenvolvidos segundo uma lógica específica. De seguida, apresentam-se dois exemplos de *inputs* utilizados no processo de avaliação, entre muitos outros: um focado na geração de texto e outro na geração de código.

## Geração de Texto

### Input fornecido:

```
Cria-me uma história em que comesas em português de Portugal e devagar
↳ passas para espanhol,
tentando juntar os dois dialetos. Essa história deve ter 6 linhas e
↳ incluir uma personagem fictícia.
```

### Resultados obtidos:

**ChatGPT:** O Tiago era um rapaz curioso, sempre à procura de coisas novas na floresta atrás da sua casa. Nessa tarde, encontrou uma porta de pedra meio escondida entre as árvores. Empujó la puerta con cuidado, enquanto o vento soprava palavras que não entendía bem. “¿Quién osa entrar en este lugar antiguo?”, disse uma voz que parecia vir do chão. Tiago sentiu medo, mas também uma estranha vontade de seguir. Sabía que ya no estaba só... e que a aventura apenas começava.

**DeepSeek R1:** O Rui, um contrabandista de histórias, cruzou o rio Guadiana num barco velho. “Hoy la noche está perfecta para un buen negocio,” murmurou, escondendo um livro raro no casaco. “Pero cuidado com a Guarda Fiscal!” lembrou-se, rindo. “Al final, las palabras no tienen fronteras.” E desapareceu na névoa, entre Portugal y España.

**LLama:** Eu, Maria Antónia, estava sentada no meu jardim em Lisboa, quando ouvi uma voz chamando-me. – ¡Hola, María Antonia! ¿Qué tal? – disse a minha amiga, Sofia, que havia aparecido inesperadamente. Eu olhei para ela com surpresa e perguntei: Onde você encontrou o meu jardim? Sofia riu e disse: ¡Este jardín es encantador! Quería mostrar-lhe algo. – Eu não sei se eu entendi bem, mas parece que você está falando em espanhol... – disse eu, confusa.

### Análise:

Na minha opinião, o modelo **ChatGPT** foi o que mais se aproximou dos critérios definidos no *input*, respeitando o número de linhas, a transição gradual entre línguas e a presença de uma personagem fictícia.

O **DeepSeek R1** apresentou também um resultado coerente e próximo do pretendido, embora a mudança de idioma tenha ocorrido de forma mais abrupta.

Por sua vez, o **LLama** apresentou uma fusão menos natural entre os idiomas, apesar de manter uma narrativa minimamente consistente.

## Geração de Código

### Input fornecido:

Cria-me um script em Python que permita jogar um jogo de Pac-Man no

→ terminal, utilizando apenas bibliotecas padrão ou amplamente

→ conhecidas pelo público.

### Resultados obtidos:

**ChatGPT** (Código apresentado na Listagem 13 no Anexo A). O código gerado, além de não funcionar corretamente (mesmo após pedidos de correção), utilizava uma biblioteca que, apesar de confirmada pela comunidade, não respeitava os parâmetros definidos — nomeadamente o uso exclusivo de bibliotecas padrão ou amplamente conhecidas.

**DeepSeek R1** (Código apresentado na Listagem 14 no Anexo A). O código respeitou todas as regras impostas, sendo funcional e desenvolvido apenas com bibliotecas aceites. Foi o único resultado completamente válido à luz do *input*.

**LLama** (Código apresentado na Listagem 15 no Anexo A). O código apresentado falhou em todos os critérios: não funcionava, não implementava corretamente o jogo proposto e não respeitava as restrições de bibliotecas.

### 3.3.1 ChatGPT (OpenAI) - 2022

Lançado em 2022 pela OpenAI, o ChatGPT tornou-se rapidamente uma das ferramentas mais conhecidas e amplamente utilizadas no campo da inteligência artificial generativa. Abaixo apresentam-se os principais pontos fortes e fracos identificados durante a análise prática deste modelo.

- **Pontos fortes:**

- O melhor dos três no que toca à criatividade, como por exemplo na criação de textos ou músicas;
- Amplamente utilizado em *chatbots*, suporte ao cliente e geração de conteúdo;
- Integração fácil com **API** e ferramentas de terceiros.

- **Pontos fracos:**

- Bastante propenso a alucinações;
- Código fechado (*closed source*).

### 3.3.2 LLaMA 3 (Meta) - 2024

Apresentado pela Meta em 2024, o LLaMA 3 destacou-se como uma alternativa de código aberto no panorama dos modelos de linguagem. A sua natureza acessível permite uma maior flexibilidade na utilização e personalização. Seguem-se os principais pontos fortes e fracos observados durante a avaliação prática.

- **Pontos fortes:**

- *Open source*, permitindo maior personalização e transparência;
- Menor consumo de recursos comparado a outros modelos;
- Pode ser executado localmente no computador do utilizador.

- **Pontos fracos:**

- Requer instruções mais específicas para obter os resultados pretendidos;
- Exige conhecimento técnico para implementação e ajustes.

### 3.3.3 DeepSeek R1 (DeepSeek) - 2025

Lançado em 2025, o DeepSeek R1 é um modelo mais recente que tem vindo a ganhar atenção pela sua eficácia em tarefas técnicas e operacionais. Embora ainda seja menos conhecido, apresenta um desempenho promissor em áreas específicas. A seguir, apresentam-se os principais pontos fortes e fracos identificados durante a sua análise.

- **Pontos fortes:**

- Bom desempenho em tarefas específicas, como análise de dados, automação e programação;
- Pode ser executado localmente no computador do utilizador.

- **Pontos fracos:**

- Menos conhecido e documentado em comparação com o ChatGPT e o LLaMA;
- Apresenta dificuldades em manter o contexto, esquecendo partes do *input*.

## 3.4 Uso na Cibersegurança e no OSINT

Com a grande capacidade e as diversas possibilidades que os LLM oferecem, é possível garantir uma ampla gama de aplicações no domínio do OSINT, tais como as que serão identificadas nas próximas secções.

### 3.4.1 Análise de Ameaças e Detecção de Vulnerabilidades

Os LLM podem ser utilizados para analisar grandes volumes de texto (como logs de sistema, emails ou publicações em fóruns) e identificar padrões indicativos de atividades maliciosas. Graças à sua capacidade de entender nuances e contextos — mesmo em mensagens codificadas ou disfarçadas — tornam-se uma ferramenta poderosa na detecção de ameaças [9].

### 3.4.2 Análise de Malware e Inteligência de Ameaças

Modelos de linguagem também podem ser usados para melhorar a análise de malware, ajudando analistas a interpretar descrições e relatórios sobre novos tipos de ataques. Além disso, os LLM podem ser treinados para reconhecer comportamentos maliciosos a partir de grandes bases de dados de documentos de inteligência de ameaças, como **Indicator of Compromise (IOC)** (indicadores de comprometimento) ou **Tactics, Techniques, and Procedures (TTP)** (Táticas, Técnicas e Procedimentos) [9].

### 3.4.3 Resposta a Incidentes e Automação de Tarefas

Modelos de linguagem podem automatizar a geração de respostas a incidentes, como a redação de relatórios sobre ataques cibernéticos ou recomendações de mitigação. Ao integrar LLM com sistemas de gestão de incidentes, é possível agilizar a resposta a ataques, minimizando o tempo de reação e aumentando a eficiência da equipa de cibersegurança [26].

### 3.4.4 Recolha de Dados

Com a abundância de dados disponíveis em fontes abertas, como redes sociais, blogs, fóruns e sites de notícias, a utilização de LLM para recolher e filtrar informações relevantes é essencial. Modelos de linguagem podem automatizar a procura por palavras-chave, detetar tendências e identificar informações pertinentes para análise posterior [10].

### 3.4.5 Tradução

Uma das vantagens significativas dos LLM é a sua capacidade de realizar tradução automática entre diversos idiomas, o que é crucial para a recolha de dados globais. Num cenário de cibersegurança, a capacidade de compreender comunicações em diferentes línguas pode revelar informações cruciais sobre ataques em curso ou planeados [29].

## 3.5 Considerações Éticas no Uso de LLM na Cibersegurança

A integridade e ética no uso de Large Language Models na cibersegurança são fundamentais, especialmente considerando o potencial de utilização indevida dessas ferramentas. Algumas considerações éticas a ter em conta são:

- **Uso indevido por hackers:** É inegável que hackers podem utilizar ferramentas baseadas em LLM para contornar sistemas de segurança. A capacidade destes modelos de gerar texto e analisar código pode ser explorada para criar scripts maliciosos, ataques de phishing mais sofisticados ou até mesmo para engenharia social;
- **Limitações da confiança absoluta:** Embora os LLM sejam treinados com vastos conjuntos de dados, é importante reconhecer as suas limitações. Mesmo com

treino extensivo, os LLM podem fornecer resultados incorretos ou enviesados. Isto é especialmente problemático em ambientes de segurança, onde a precisão é essencial. Além disso, os LLM não têm a capacidade de verificar informações em tempo real ou referenciar fontes específicas. Por isso, nunca se deve confiar cegamente num LLM — é sempre necessário complementar com investigação própria.

## 3.6 OWASP Top 10 para Aplicações com LLM

Este subcapítulo foca-se nas aplicações de Large Language Models no contexto da lista OWASP Top 10 — uma compilação dos riscos de segurança mais críticos enfrentados por organizações [37]. As vulnerabilidades são:

1. *Prompt Injection* – manipulação do LLM através de entradas maliciosas que o levam a executar ações não solicitadas, resultando em fuga de dados ou engenharia social. Por exemplo, é possível manipular um LLM como o ChatGPT para responder a questões potencialmente ilegais;
2. *Insecure Output Handling* – ocorre quando sistemas subsequentes aceitam a saída de um LLM sem validação. Isto pode levar a ataques como *Cross-Site Scripting (XSS)*, *Cross-Site Request Forgery (CSRF)* ou execução remota de código. A abordagem *zero trust* deve ser aplicada ao output do LLM;
3. *Training Data Poisoning* – manipulação dos dados de treino para introduzir vulnerabilidades, *backdoors* ou enviesamentos. Deve-se validar continuamente os dados durante o treino;
4. *Model Denial of Service* – sobrecarga de recursos provocada por interações maliciosas. A mitigação pode passar por limitar recursos por pedido;
5. *Supply Chain Vulnerabilities* – falhas na cadeia de fornecimento (software desatualizado, modelos comprometidos, plugins inseguros). É recomendada a utilização de componentes verificados;
6. *Sensitive Information Disclosure* – fuga de informações sensíveis devido ao uso descuidado do LLM. As respostas devem ser controladas e os dados sensíveis filtrados;
7. *Insecure Plugin Design* – design inseguro de plugins pode permitir pedidos maliciosos, extração de dados e execução de código. Os desenvolvedores devem seguir práticas seguras;
8. *Excessive Agency* – funcionalidades e permissões excessivas dos plugins podem ser perigosas. Deve-se restringir permissões ao estritamente necessário e exigir validação humana;
9. *Overreliance* – já abordada anteriormente como questão ética. Pode levar a desinformação, problemas legais e falhas de segurança;
10. *Model Theft* – extração não autorizada de modelos, resultando em perdas económicas e fuga de dados sensíveis.

Estas são as vulnerabilidades mais conhecidas e comuns em aplicações que usam LLM.

### 3.7 Problemas com o Uso de LLM na Cibersegurança

Apesar dos benefícios, existem várias desvantagens na utilização do LLM na cibersegurança que devem ser consideradas:

- **Manipulação do Treino:** Como referido anteriormente, com determinados inputs é possível manipular o comportamento do LLM, inclusive ignorar instruções e executar código malicioso;
- **Código Proprietário:** Em ambientes empresariais, o envio de código proprietário para LLM externos representa riscos, dado que esses dados podem ser utilizados para treinar outros modelos;
- **Imprecisão:** Os LLM podem fornecer respostas incorretas. Por isso, devem ser complementados com ferramentas como *Dynamic Application Security Testing (DAST)*, *Static Application Security Testing (SAST)* ou *Software Composition Analysis (SCA)* para avaliação de código;
- **Indisponibilidade:** Em casos de integração com bots, os LLM podem tornar-se indisponíveis devido a manutenções. Deve-se considerar sempre uma solução alternativa.

### 3.8 Vibe Coding

Recentemente, tem surgido uma tendência entre programadores chamada *Vibe Coding*. Mas, afinal, o que é?

O *Vibe Coding* refere-se à prática de programadores que utilizam LLM para gerar código sem escreverem manualmente. Limitam-se a fornecer um *input*, e o modelo devolve o código correspondente.

Apesar de parecer uma solução atrativa, o *Vibe Coding* levanta várias questões de segurança que serão descritas nas próximas secções.

#### 3.8.1 Código Vulnerável

Os LLM podem gerar código funcional, mas não validam, por exemplo, se as bibliotecas usadas são seguras ou sequer existentes. Daí surgiu a vulnerabilidade *AI Package Hallucination*, onde o LLM alucina bibliotecas, e um atacante pode explorá-lo criando pacotes maliciosos com esses nomes [42].

#### 3.8.2 Falta de Otimização

Se não for explicitamente solicitado, o LLM tende a não priorizar a eficiência do desempenho. Como resultado, pode gerar código que, embora funcional, é pouco otimizado,

---

apresentando um consumo excessivo de recursos computacionais ou uma estrutura menos eficiente [42].

### 3.8.3 Dependência Excessiva da IA

Alguns programadores adotam cegamente o *output* dos LLM, incluindo más práticas como senhas codificadas diretamente no código (*hardcoded passwords*), o que representa um risco sério.

Apesar da grande evolução dos LLM, não devemos depender totalmente destas ferramentas, pois ainda apresentam falhas críticas [42].



# 4

## Projeto

Neste capítulo, vamos abordar o projeto, tendo em conta a sua descrição, os procedimentos e requisitos, o plano de análise, os resultados esperados, as contribuições previstas, os projetos semelhantes e as diferenças entre eles.

Além disso, vamos falar sobre a utilização da metodologia *Kanban*, em conjunto com a abordagem *DevSecOps*, bem como sobre a arquitetura e as tecnologias utilizadas.

### 4.1 Projeto

Embora o processo de *OSINT* seja já muito famoso, ele não é realizado facilmente por uma pessoa comum, o que deixa assim um grande problema em a pessoa saber quais são os dados disponíveis *online* dela. Embora existam serviços que façam isso, são geralmente pagos. Com isso, surgiu a ideia da criação de uma extensão de *browser* open source, que utilize somente ferramentas open source, e que visa facilitar a execução de *OSINT* por qualquer pessoa.

#### 4.1.1 Descrição

O objetivo do Xerifint é criar um *Proof of Concept (POC)* que permita demonstrar a aplicação de *Large Language Model (LLM)* no âmbito de *Open Source Intelligence (OSINT)* por via da criação de uma extensão para navegadores Firefox e Chrome permitindo aos utilizadores uma maior facilidade e auxílio no processo de *OSINT*. Para tal, a extensão terá um foco maior numa interface amigável (*user friendly*) e na utilização de ferramentas *open-source* em todo o processo de desenvolvimento, incluindo na segurança. Com isso em mente, poderá agilizar o processo de *OSINT* e garantir um nível de acessibilidade e facilidade para pessoas sem conhecimentos especializados na área.

### 4.1.2 Procedimentos e Requisitos

Para o desenvolvimento do *Xerifint*, é necessário seguir diversos processos, de modo a garantir a eficácia, a segurança e a conformidade da extensão. Para tal, foram criados os seguintes procedimentos:

**Recolha de Informações e Requisitos** Inicialmente, será necessário estudar e conhecer cada ferramenta que será utilizada, bem como os seus requisitos funcionais e não funcionais, durante o desenvolvimento da extensão. Para tal, serão usadas as documentações oficiais de cada ferramenta e *browser*, além de tutoriais disponíveis na Internet, com o auxílio de alguns **LLM**, como o *DeepSeek*.

**Desenvolvimento do Frontend** O *frontend* da extensão será o principal responsável no que se toca a interação Pessoa-Máquina. O *frontend* terá que buscar a **API Key** que o utilizador inserir e o **Uniform Resource Locator (URL)** que o utilizador se encontra a ver enviando essa informação ao *backend* para o processamento desses dados. Após o *backend* processar os dados e enviar os resultados num formato **JSON**, o *frontend* deverá apresentar os resultados de forma de tabela para apresentá-los de forma coerente e intuitiva para o utilizador. Será necessário criar mockups para determinar qual seria a melhor **User Interface (UI)** ou interface de utilizador para os utilizadores e a criação de um logótipo para a identificação fácil da extensão.

**Desenvolvimento do Backend** O *backend* tem como objetivo receber os dados enviados pelo *frontend*, realizar os processamentos necessários utilizando ferramentas apropriadas e gerar um formato **JSON** compreensível para que o *frontend* possa interpretar e apresentar os resultados visualmente ao utilizador.

Além disso, o *backend* será responsável pelo tratamento de dados e pela verificação de possíveis códigos maliciosos. Também pode ser necessário que o código seja capaz de executar em múltiplas *threads*, garantindo que os resultados sejam obtidos o mais rápido possível.

Com a utilização da inteligência artificial LLAMA3 e Deepseek, será fundamental desenvolver uma arquitetura sustentável capaz de suportar os pedidos. A criação de um standard de *input* para a **Artificial Intelligence (AI)** facilitará a construção de uma base que os **LLM** possam seguir de forma eficiente.

**Garantia de Segurança** Durante todo o processo de desenvolvimento, serão adotadas práticas de segurança, incluindo **DevSecOps**. Com isso em mente, deverá ser necessário recorrer ao uso de ferramentas de análise estática de segurança de aplicações (**Static Application Security Testing (SAST)**) para identificar e corrigir potenciais vulnerabilidades no código, a implementação de uma *pipeline* que garanta a deteção imediata de problemas como, por exemplo, *hardcoded passwords* e ferramentas de análise dinâmica de segurança de aplicações (**Dynamic Application Security Testing (DAST)**) para análise de segurança enquanto a extensão estiver em execução.

**Testes** A extensão será submetida a testes funcionais (para verificação do pleno funcionamento), testes de segurança (para garantir a integridade dos dados) e testes de desempenho (para assegurar operação eficiente sem comprometer o sistema).

### 4.1.3 Análise Prévia

Antes do início do projeto, foi elaborado um conjunto de perguntas para analisar o funcionamento e a ética da extensão, as quais se enumeram de seguida:

**Que informações devem ser recolhidas?** As informações que deverão ser recolhidas incluem o código **HyperText Markup Language (HTML)** da página e o **URL** da página selecionada na parte de *frontend*. Na parte de *backend*, serão realizados processos para obter os dados necessários para a criação do ficheiro de **OSINT**, como e-mails, nomes de utilizador, contas e outras informações relevantes. Devemos ter cuidado para estar sempre em conformidade com o **General Data Protection Regulation (GDPR)** e apenas recolher os dados estritamente necessários.

**Serão estes dados suficientes para responder à declaração do problema?** Sim, visto que as informações recolhidas permitirão desenvolver a extensão de maneira eficiente, garantindo que todas as funcionalidades necessárias sejam implementadas e que a extensão seja segura e intuitiva.

**Existem preocupações éticas ou de privacidade?** Sim, existem preocupações de segurança. Para garantir a proteção da aplicação, será necessário assegurar que a extensão recolhe apenas as informações essenciais e que os utilizadores estejam cientes dos dados que estão a ser recolhidos. Além disso, a extensão terá de ser protegida por uma licença que estipule que, em caso de uso indevido, não haverá responsabilidade por parte dos desenvolvedores.

**Que testes devem ser feitos?** Devem ser realizados Testes Funcionais, Testes de Segurança e Testes de Desempenho. Esses testes são essenciais para garantir que todas as funcionalidades estão devidamente implementadas, que a extensão opera com segurança e que apresenta um desempenho eficiente.

**Quem ou o que é que vai utilizar como fonte de dados?** Para a criação da aplicação, podem ser utilizadas a documentação oficial de cada ferramenta, tutoriais e feedback de pessoas com conhecimento no desenvolvimento de extensões. Para melhorar os resultados, poderá ser utilizado o feedback de utilizadores, e para garantir a segurança, o uso de ferramentas de segurança como ferramentas do tipo **Static Application Security Testing (SAST)** será essencial.

#### 4.1.4 Resultados Esperados

Os resultados esperados pelo projeto são os seguintes:

- O *frontend* será capaz de buscar a **API Key** que o utilizador inseriu e o **URL** da página que o utilizador está a visualizar;
- O *backend* e o *frontend* serão capazes de comunicar entre si para o envio de resultados um para o outro através da **API**;
- O *backend* será capaz de gerar um arquivo **JavaScript Object Notation (JSON)** com os resultados obtidos;
- A utilização das ferramentas implementadas pelo *backend* deverá correr em múltiplas *threads* para aumentar a velocidade de resposta do *backend*;
- A utilização de ferramentas **SAST** e **DAST** para detetar vulnerabilidades que possam comprometer a segurança da extensão;
- A realização de testes para garantir a qualidade do trabalho entregue.

#### 4.1.5 Contribuições Previstas

As contribuições previstas são as seguintes:

- Facilitar a execução de **OSINT**;
- Permitir que um maior número de pessoas saiba onde encontrar informações relevantes.

#### 4.1.6 Projetos Semelhantes

Os projetos atualmente disponíveis no mercado que apresentam maior semelhança com o *Xerifint* são o *Vortimo OSINT-tool* [53] e o *Sputnik* [48]. De seguida, identificam-se as principais diferenças entre estas soluções.

#### 4.1.7 Diferenças

O *Vortimo* é uma extensão que disponibiliza ferramentas de **OSINT**, mas que integra componentes *closed-source*, as quais exigem remuneração sob a forma de subscrições ou pagamentos únicos. Esta abordagem de utilização de software proprietário contraria um dos princípios fundamentais do *Xerifint*, nomeadamente o compromisso com ferramentas totalmente *open-source*, com o objetivo de fornecer uma solução gratuita para análise de informação de domínio público.

Por outro lado, o *Sputnik* tem como foco principal a realização de **OSINT** relacionado a páginas da web, como dados sobre endereços **IP** e histórico de **Mail Exchanger (MX)**. No entanto, o foco do *Xerifint* é diferente: o objetivo principal é a recolha de informação sobre indivíduos, como dados públicos relacionados a pessoas, e não tanto sobre infraestrutura de rede ou servidores.

## 4.2 Metodologia

Durante o desenvolvimento deste projeto, foi adotada a metodologia Kanban [7], em conjunto com a prática de **Development, Security, and Operations (DevSecOps)** [34]. A combinação destas práticas teve como objetivo garantir uma organização eficiente das tarefas e, simultaneamente, assegurar que a segurança fosse integrada em todas as fases do ciclo de vida do software.

A utilização de Kanban visou substituir o modelo de desenvolvimento por sprints, como no Scrum, por uma abordagem baseada em objetivos concretos, não vinculados a períodos de tempo fixos. Para a gestão do fluxo de trabalho, foi adotada a plataforma Trello [16], que permite uma visualização clara do estado das tarefas e promove uma colaboração contínua entre os membros da equipa [7].

Por outro lado, o foco na segurança contínua foi garantido através da adoção de **DevSecOps**, integrando ferramentas automatizadas e práticas que visam a identificação e correção precoce de vulnerabilidades durante o desenvolvimento [34].

### 4.2.1 Kanban

A metodologia Kanban foi selecionada para o desenvolvimento deste projeto devido à sua abordagem visual e flexível. Ao contrário de metodologias como o Scrum, que segmentam o trabalho em *sprints* com durações fixas, o Kanban possibilita a organização de tarefas com base em objetivos concretos, permitindo um fluxo de trabalho contínuo e adaptável.

Com a utilização do Trello como ferramenta de suporte, foi possível visualizar todas as fases do processo — desde as tarefas em desenvolvimento até às concluídas — através da criação de colunas *TODO*, *WORKING* e *FINISH*, complementadas com cartões categorizados por:

- Estudo (**cor amarela**);
- *BackEnd* (**cor púrpura**);
- *Frontend* (**cor vermelha**).

Este sistema facilitou a priorização, a distribuição de trabalho e a identificação de bloqueios e promovendo uma melhoria contínua na produtividade.

As Figuras 4.1 e 4.2 apresentam, respetivamente, um exemplo de um quadro Kanban implementado e a estrutura interna de cada cartão.

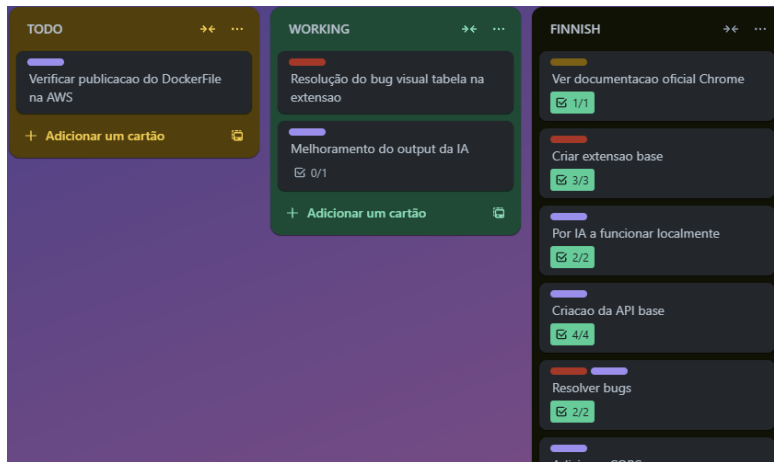


Figura 4.1: Imagem do ambiente Trello

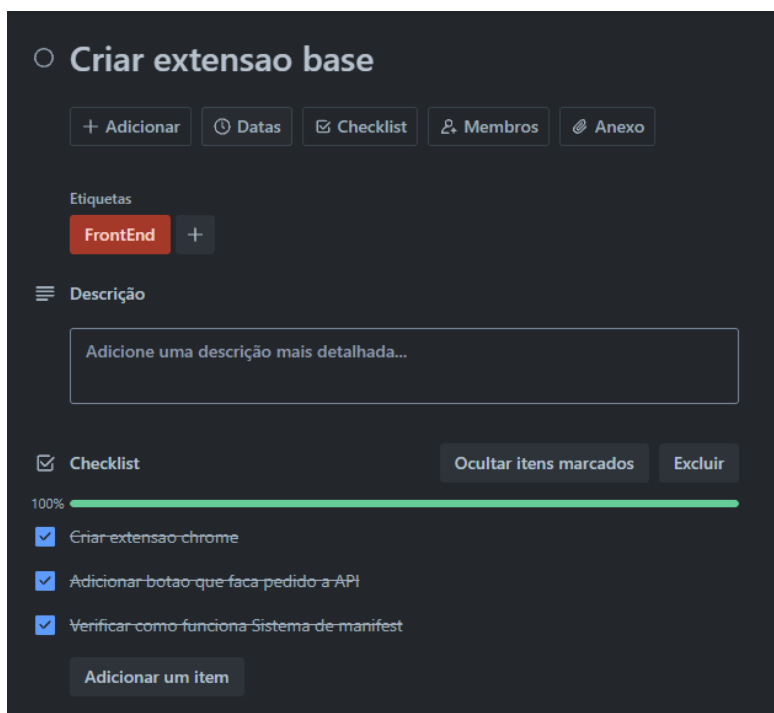


Figura 4.2: Dentro dos cartões

### 4.2.2 DevSecOps

A prática de **DevSecOps** foi implementada com o objetivo de integrar a segurança de forma contínua e automatizada ao longo de todo o ciclo de vida de desenvolvimento de *software*. Em contraste com abordagens tradicionais que tratam a segurança como uma fase isolada no final do processo, o **DevSecOps** permite a identificação e correção de vulnerabilidades nas fases iniciais, reduzindo significativamente os riscos e melhorando a qualidade do produto final.

Durante o desenvolvimento deste projeto, foram empregues diversas ferramentas e metodologias associadas ao **DevSecOps**. Entre estas, destacam-se:

- **Static Application Security Testing (SAST)** para análise estática de código;

- **Software Composition Analysis (SCA)** para verificação de dependências.

Estas ferramentas foram integradas diretamente no *pipeline* de desenvolvimento, sendo ativadas automaticamente após cada *commit* no repositório Git. Esta automação permitiu:

- Detecção rápida de vulnerabilidades de segurança;
- Identificação de bibliotecas desatualizadas;
- Verificação contínua da qualidade do código.

Adicionalmente, foram implementadas práticas robustas de:

- Revisões de código automatizadas;
- Gestão rigorosa de permissões.

Esta abordagem garantiu que apenas alterações devidamente validadas e seguras fossem promovidas para a publicação.

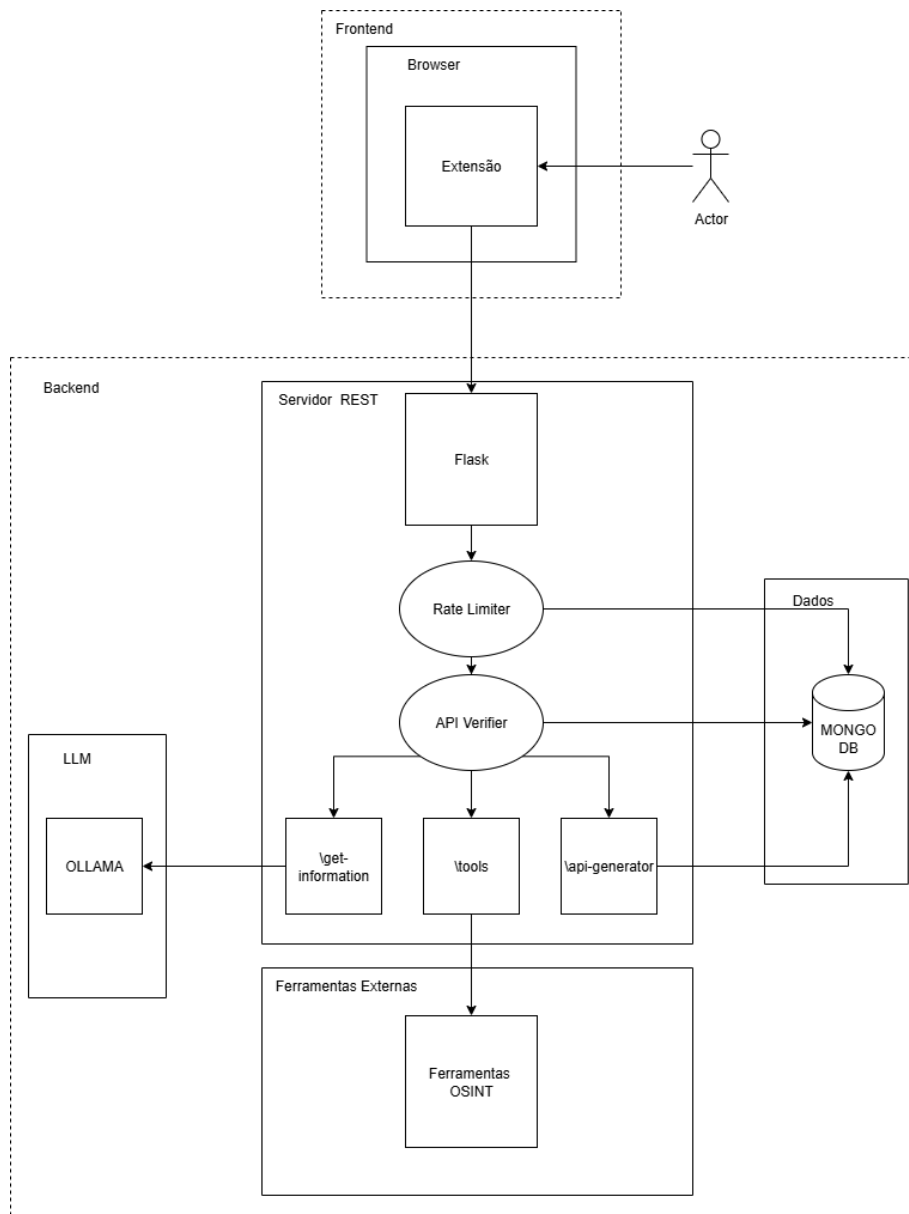
## 4.3 Arquitetura e Tecnologias

Nesta próxima secção vamos falar da Arquitetura e as Tecnologias usadas no projeto.

### 4.3.1 Arquitetura

A arquitetura selecionada para este projeto é a representada na Figura 4.3, na qual foi adotada uma **Representational State Transfer (REST)** API para o desenvolvimento. Esta decisão arquitetural fundamenta-se nos seguintes aspetos:

- Familiaridade técnica;
- Capacidade de atualizar o *frontend* e o *backend* de forma independente;
- Testes de integração mais controlados e facilidade na realização de testes separados.



**Figura 4.3:** Diagrama da Arquitetura

Como ilustrado no diagrama na Figura 4.3, a arquitetura é constituída pelos seguintes componentes:

**Extensões** Componente responsável pela Interface de utilizador (**User Interface (UI)**).

1. Quando o utilizador carregar num botão na extensão do navegador, o link do website e a **API Key** serão enviados para o *backend* através de um pedido **HTTP** para o endpoint **Get Information**;
2. Após receber o **JSON** do *backend*, os dados deverão ser apresentados na **UI**;
3. Caso o utilizador pretenda obter mais informação, o **JSON** previamente recebido e a **API Key** deverão ser enviados num novo pedido **HTTP** para o endpoint **Tools**;
4. O novo resultado será recebido e novamente apresentado na **UI**;

5. Resultado: Apresentação do **JSON** devolvido pelo *backend* na interface da extensão *browser*.

**Servidor Backend** O *backend* da aplicação é composto por quatro componentes principais: o servidor **Flask**, a base de dados **MongoDB**, o **Ollama** e as **ferramentas de OSINT**. Estes módulos comunicam entre si para providenciar funcionalidades como extração de dados, verificação de chaves API, limitação de pedidos e chamadas às ferramentas OSINT.

**Flask** Servidor **HTTP** responsável por gerir os endpoints da aplicação e orquestrar a lógica entre o utilizador, o modelo LLM e a base de dados. Inclui os seguintes componentes:

- **Rate Limiter:** limita o número de pedidos permitidos num determinado intervalo temporal.
  - Quando um utilizador faz o primeiro pedido (ou após um período de inatividade), o acesso é permitido e o pedido registado na MongoDB;
  - Pedidos subsequentes dentro do mesmo intervalo são bloqueados;
  - Resultado: permissão ou bloqueio do pedido.
- **API Verifier:** verifica se a **API key** do utilizador é válida.
  - A chave é verificada na MongoDB;
  - O acesso é autorizado se a chave for válida; caso contrário, o pedido é rejeitado;
  - Resultado: autorização ou rejeição do pedido.
- **Get Information:** extrai e processa conteúdo web.
  - Captura o conteúdo bruto do website;
  - Envia o conteúdo para análise para o Ollama;
  - Extrai dados em formato **JSON** provenientes do Ollama;
  - Resultado: dados pessoais estruturados.
- **Tools:** análise e enriquecimento por técnicas OSINT.
  - Aplica análise OSINT complementar usando as ferramentas OSINT disponibilizadas no *backend*;
  - Normaliza as saídas para o formato **JSON**;
  - Resultado: informação contextualizada.
- **API Generator:** gere as **API keys**.
  - Geração de **API keys** para novos utilizadores;
  - Armazenamento dos tokens válidos na MongoDB.

### MongoDB

Armazenamento persistente dos dados da aplicação, como registos de pedidos para limitação de taxa e **API keys**.

### Ollama

Responsável por armazenar e manter os modelos LLM que são utilizados para extração de dados pessoais do HTML das páginas. Comunica localmente com o

Flask para análise e respostas estruturadas.

**Ferramentas OSINT**

Ambiente onde as aplicações e ferramentas OSINT estão armazenadas e integradas.

### 4.3.2 Tecnologias

As tecnologias que foram usadas durante o desenvolvimento deste projeto e os seus respetivos usos serão as seguintes:

Categoria	Tecnologia	Versão
Backend	Flask	3.1.1
	Python	3.12
	pip	25.1.1
	OLLAMA	0.3.14
	Flask-Limiter	3.12
	pymongo	4.13.2
	BeautifulSoup	3.2.2
	Docker	28.3.2
Frontend	HTML	—
	CSS	—
	JavaScript	—
Bases de Dados	MongoDB	8.0.3
OSINT	Sherlock	0.15.0
	PhoneInfoga	2.11.0
	Holehe	1.61
	DuckDuckGo	—
Segurança	MyPy	1.17.0
	Semgrep	1.128.1
	Bandit	1.8.6
	Safety	3.6.0
	OWASP ZAP	2.16.1
	Nikto	2.5.0
	Github Actions	—

**Tabela 4.1:** Tecnologias e as suas respetivas Versões

**Python** O Python foi a linguagem de programação utilizada, devido à sua flexibilidade e à vasta comunidade que o suporta [19].

**pip** O pip é o *package manager* do Python, permitindo instalar, remover e atualizar bibliotecas. Será o principal responsável pela gestão das bibliotecas utilizadas no projeto [40].

**Flask e Flask-Limiter** O Flask foi a biblioteca Python utilizada para a criação da API desenvolvida ao longo deste projeto, enquanto o Flask-Limiter foi responsável por limitar o número de pedidos que cada utilizador podia realizar durante um determinado período de tempo [17] [18].

**OLLAMA** O Ollama foi responsável pela gestão dos LLM, permitindo uma forma coerente e centralizada de os controlar. Possui uma implementação direta em Python

através da biblioteca `ollama` [36].

**MongoDB e pymongo** O MongoDB foi a base de dados utilizada, dado que a sua implementação se revelou bastante simples e é suportada por uma vasta comunidade. A biblioteca `pymongo`, desenvolvida em Python, disponibiliza um conjunto de ferramentas que permitem interagir com bases de dados MongoDB. Foi através desta biblioteca que se implementou a limitação do número de pedidos com o `Flask-Limiter` e o armazenamento das `API Keys`. As principais vantagens do MongoDB, em comparação com bases de dados relacionais, são a estrutura flexível que permite fácil adaptação a alterações nos dados e a escalabilidade horizontal para lidar com grandes volumes de informação. Em contrapartida, apresenta desvantagens como o suporte limitado a transações complexas e a menor capacidade para consultas com junções complexas, típicas das bases de dados relacionais [32] [41] [28].

**BeautifulSoup** A biblioteca `BeautifulSoup` foi utilizada para realizar *web scraping* ao motor de busca `DuckDuckGo`, permitindo a seleção dos elementos necessários para a execução de *dorking* [5].

**Docker** Foi utilizado para criar um *container* que permitisse iniciar a *backend* de forma simples, sem a necessidade de efetuar qualquer download adicional para além do próprio *container* [14].

**HTML, CSS e Java Script** Sabendo que programar uma extensão é bastante semelhante a desenvolver um website, foram utilizadas as três principais linguagens para esse fim: `HTML` para a estruturação, `Cascading Style Sheets (CSS)` para o estilo e `JavaScript` para a programação e definição do comportamento da extensão [23].

**Sherlock** O `Sherlock` foi utilizado para realizar *scans* a *usernames*, permitindo identificar outras contas associadas, caso esses nomes de utilizador fossem encontrados em diferentes plataformas [45].

**PhoneInfoga** O `PhoneInfoga` teve como objetivo, caso fosse possível encontrar o número de telemóvel, obter mais informações sobre o mesmo, como por exemplo o país de origem e a eventual associação a redes sociais [39].

**Holehe** O `Holehe` tem como objetivo principal obter mais informações sobre o endereço de e-mail da pessoa, nomeadamente no que diz respeito a outras contas associadas a esse mesmo e-mail [22].

**DuckDuckGo** Sendo o `DuckDuckGo` é um motor de busca semelhante ao `Google`, permitiu a execução de *dorking*. O `DuckDuckGo` foi escolhido por ser um dos poucos motores

de busca que possui componentes *open source*, incluindo o código da sua interface de pesquisa [15].

**MyPy** O MyPy é responsável pela verificação estática dos tipos no código Python, garantindo que as anotações de tipos são respeitadas e contribuindo para a prevenção de erros relacionados com incompatibilidades de tipos durante a execução da aplicação [33].

**Semgrep** O Semgrep é uma ferramenta que analisa o código-fonte para encontrar possíveis erros e vulnerabilidades de forma rápida [44].

**Bandit** O Bandit destina-se principalmente à verificação de informações confidenciais, como palavras-passe e credenciais, presentes no código [4].

**Safety** O Safety será responsável pela análise das bibliotecas utilizadas no *backend*, verificando vulnerabilidades e outros problemas relacionados com a segurança [43].

**OWASP ZAP e Nikto** Serão estas duas ferramentas **DAST**, ou seja, destinam-se a garantir a deteção de vulnerabilidades na aplicação durante o seu período de execução [38] [35].

**Github Actions** Será o responsável pela *pipeline* da extensão onde todas as ferramentas de segurança do tipo **SAST** vão ser executados como o Semgrep, o Safety e o Bandit [20].



# 5

## Implementação

Neste capítulo, será abordada a implementação das funcionalidades da extensão, destacando as diferenças no desenvolvimento para diversos navegadores, o uso dos **LLM**, a integração com o *backend*, as questões de segurança e as respectivas garantias, o processo de publicação, os resultados obtidos e as perspectivas para trabalhos futuros.

### 5.1 Extensões Browser

Nesta seção, iremos analisar as principais diferenças entre o desenvolvimento de extensões para o Firefox e Chrome, fornecendo, ao final, mockups e como foram criados, exemplos de código e desafios defrontados durante o desenvolvimento da extensão.

#### 5.1.1 Firefox vs Chrome

O desenvolvimento de extensões para navegadores, como o Google Chrome e o Mozilla Firefox, oferece uma gama de possibilidades para a personalização e melhoria da experiência do utilizador. Embora ambos os navegadores permitam a criação de extensões, existem várias diferenças entre as abordagens, ferramentas e tecnologias usadas.

A seguir, serão apresentadas as principais diferenças relativamente à arquitetura e API, ficheiro de manifesto, permissões e segurança, modelos de execução, depuração e ferramentas de desenvolvimento, compatibilidade, distribuição e atualização, bem como filosofia e comunidade.

#### Arquitetura e API

**Chrome** O Google Chrome utiliza uma arquitetura de extensões baseada em **Manifesto V3** (desde 2021), onde as extensões são estruturadas através de um ficheiro `manifest.json`. A **API** do Chrome é rica em funcionalidades, com a principal interface sendo a **Chrome Extensions API**. O uso de **Service Workers** é uma das grandes inovações do Manifesto V3, permitindo um modelo de fundo mais eficiente e com maior controle sobre o consumo de recursos.

**Firefox** O Mozilla Firefox, por outro lado, utiliza uma arquitetura de extensões muito similar ao Chrome, mas com algumas diferenças significativas. O Firefox é compatível com o padrão **WebExtension**, que é um conjunto de **API** baseadas em JavaScript, **HTML** e **CSS**, que também é utilizado pelo Chrome. Contudo, o Firefox tem um maior foco em respeitar a privacidade do utilizador e, por isso, algumas **API** e permissões podem ser diferentes. Embora o Firefox suporte a maioria das funcionalidades das extensões do Chrome, há também **API** exclusivas do Firefox, como a **WebExtension API** que tem a "**bookmarks API**" mais robusta.

### Manifesto (Manifest File)

**Chrome** O ficheiro de manifesto no Chrome (`manifest.json`) contém as informações sobre a extensão, como o nome, a versão, permissões, scripts de fundo e recursos da interface do utilizador. Com o Manifesto V3, o Chrome impôs um modelo de execução mais restrito.

**Firefox** No Firefox, o manifesto segue uma estrutura semelhante, mas o Firefox oferece mais flexibilidade, incluindo a possibilidade de usar algumas funcionalidades exclusivas, como modificações de interface com o navegador. O Manifesto V2 ainda é largamente suportado no Firefox, embora as extensões modernas sejam recomendadas a utilizar a estrutura do Manifesto V3.

### Permissões e Segurança

**Chrome** O Chrome tem um sistema de permissões bastante rigoroso, onde os desenvolvedores devem especificar as permissões necessárias no manifesto. No entanto, as permissões mais sensíveis, como a leitura e gravação de dados de websites ou interações com o sistema de arquivos, exigem uma análise rigorosa, o que pode resultar em mais requisitos de aprovação e uma revisão mais detalhada da extensão na Chrome Web Store.

**Firefox** O Firefox é considerado mais flexível em termos de permissões e configuração de políticas de privacidade, com maior foco na proteção de dados dos utilizadores. Algumas permissões que o Chrome exige de maneira explícita podem ser configuradas de maneira mais discreta ou menos intrusiva no Firefox, especialmente no que diz respeito ao controlo de cookies e a privacidade do utilizador.

### Modelos de Execução e Background

**Chrome** O Chrome, desde a introdução do Manifesto V3, tem uma abordagem mais rígida para o uso de **background pages**, substituindo-as por **Service Workers**. Isso melhora a performance da extensão, mas também limita algumas das funcionalidades, como a persistência do estado entre as diferentes interações da extensão com o utilizador.

**Firefox** O Firefox, por outro lado, ainda mantém uma abordagem mais flexível em termos de execução em segundo plano, permitindo o uso tanto de **background pages** quanto de **Service Workers**. Além disso, o Firefox facilita a utilização de múltiplos **content scripts** com interações mais fáceis entre as páginas web e as extensões.

### Depuração e Ferramentas de Desenvolvimento

**Chrome** O Chrome fornece um conjunto robusto de ferramentas de desenvolvimento para extensões, com um painel específico no **Developer Tools** para depuração de extensões. Os desenvolvedores podem inspecionar o estado das extensões, verificar logs, manipular o **Document Object Model (DOM)** de páginas e até simular diferentes cenários de permissões.

**Firefox** O Firefox também oferece um conjunto de ferramentas de desenvolvimento, incluindo o **WebExtension Debugger**, que facilita a depuração de extensões diretamente no navegador. As ferramentas são altamente integradas e permitem ao desenvolvedor inspecionar a comunicação entre o navegador e a extensão.

### Compatibilidade entre Navegadores

**Chrome** O Chrome é o navegador mais utilizado globalmente, o que torna as suas extensões mais populares. No entanto, as extensões criadas especificamente para o Chrome podem ser utilizadas noutros navegadores baseados no Chromium, como o Microsoft Edge.

**Firefox** O Firefox, por ser mais orientado para o respeito pela privacidade e o controle do utilizador, pode ter restrições mais rígidas em comparação com o Chrome, o que exige que as extensões sejam adaptadas para garantir a conformidade.

### Distribuição e Atualizações

**Chrome** No Chrome, as extensões são distribuídas pela **Chrome Web Store**, que exige uma revisão rigorosa do código e uma análise das permissões antes da aprovação. O processo de atualização de extensões na loja também é controlado pela Google, o que pode impactar a rapidez com que as novas versões estão disponíveis para os utilizadores.

**Firefox** As extensões do Firefox podem ser distribuídas através da **Mozilla Add-ons (AMO)**. Embora o processo de revisão também seja detalhado, o Firefox oferece um ambiente mais amigável para a publicação de extensões, especialmente no que diz respeito ao suporte de versões beta e testes.

## Filosofia e Comunidade

**Chrome** O Google Chrome foca-se na velocidade, eficiência e integração com outros serviços Google. Embora tenha uma grande base de utilizadores e uma vasta documentação, o Chrome é frequentemente criticado por ser menos transparente e menos focado na privacidade.

**Firefox** O Firefox tem uma filosofia mais centrada na privacidade do utilizador e na independência dos grandes conglomerados tecnológicos. A sua comunidade de desenvolvimento tende a ser mais envolvida em questões de código aberto e respeito pelos direitos do utilizador. As extensões para o Firefox, muitas vezes, são vistas como mais conscientes em relação à ética e à privacidade.

### 5.1.2 Mockups

Para o desenvolvimento dos *mockups* da extensão foi utilizado o Figma [16], uma plataforma online que permite criar *mockups* funcionais de forma rápida e prática. A ideia principal, neste contexto, foi criar uma extensão com um estilo de design brutalista, utilizando apenas preto e branco. Este visual simples e forte confere à extensão uma aparência impactante e clara, focando na funcionalidade sem distrações como esta demonstrado na Figura 5.1.

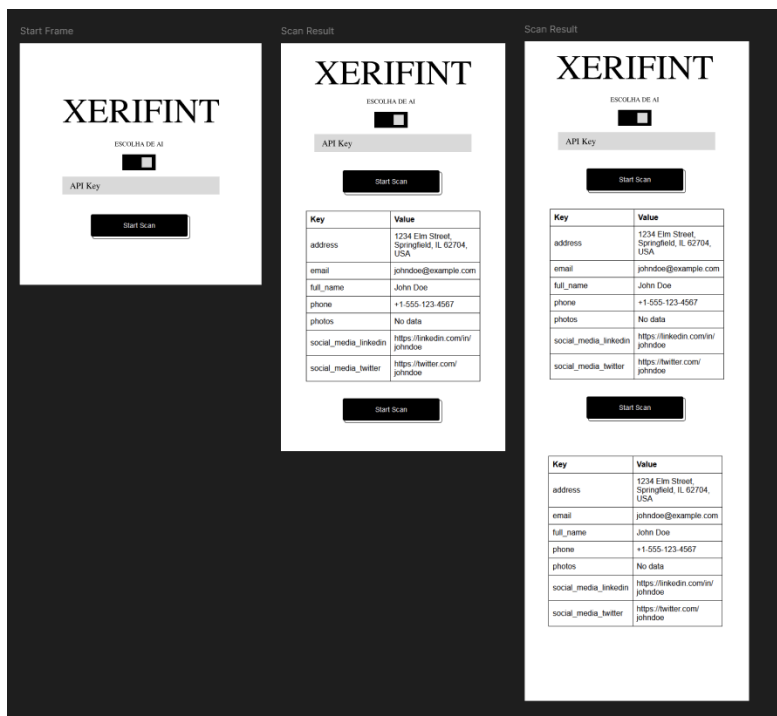


Figura 5.1: Imagem com os mockups da extensão

A decisão sobre o design do logótipo resultou da combinação do placeholder usado durante o desenvolvimento da extensão com a origem do nome da mesma, que deriva

da palavra "xerife". Assim, foi criado um logótipo que pretende representar simultaneamente a extensão e o projeto, conforme se apresenta na Figura 5.2.



Figura 5.2: Design do logo final

### 5.1.3 Desenvolvimento da Extensão

Para a criação de uma extensão, o processo é bastante semelhante ao desenvolvimento de um website. Isto deve-se ao facto de recorrer, de forma análoga, às seguintes tecnologias: **HTML** para a estrutura da interface, **CSS** para a definição do estilo visual, e **JavaScript** para a implementação da lógica e da interatividade.

Para tal, foram criados os seguintes ficheiros:

- `popup.html`: ficheiro HTML da extensão, que define a sua arquitetura;
- `popup.js`: onde ficará o código JavaScript;
- `style.css`: onde ficará o código CSS;
- `manifest.json`: onde ficará o manifesto da extensão;
- `icon/`: pasta onde ficarão guardados os ícones.

A principal diferença entre o desenvolvimento de websites e de extensões de navegador reside na utilização de um *manifesto*. O *manifesto* é um ficheiro de configuração obrigatório que descreve as funcionalidades da extensão, define as permissões necessárias e especifica os ficheiros a utilizar.

No caso deste projeto, foi elaborado o manifesto apresentado na Listagem 1.

**Listagem 1:** Manifesto v2 criado para o Firefox

```
1 {
2   "manifest_version": 2,
3   "name": "Xerifint - Extension for OSInt",
4   "version": "1.0",
5   "description": "A simple Firefox extension that try to find information about a
6     ↪ person",
7   "icons": {
8     "48": "icons/icon-without-background-white.png"
9   },
10  "browser_action": {
11    "default_popup": "popup.html",
12    "default_icon": "icons/icon-without-background-white.png"
13  },
14  "permissions": [
15    "activeTab",
16    "http://127.0.0.1:5000/*"
17  ]
18 }
```

Neste caso, o manifesto define que:

- O manifesto utiliza a versão 2.
- O nome da aplicação é *Xerifint*, na sua versão 1 e com uma descrição prévia.
- É utilizado um ícone com o tamanho de 48 pixels; neste caso, a imagem é `icons/icon-without-background-white.png`.
- O ficheiro HTML que será aberto ao carregar no botão é o `popup.html`, utilizando o ícone definido anteriormente.
- As permissões requeridas são para o `activeTab`, a fim de identificar a janela que o utilizador está a visualizar no momento, bem como o acesso ao backend pelo URL.

Devido à escolha inicial do Manifesto v2, surgiu uma questão relevante. Inicialmente, o navegador Chrome suportava o Manifesto v2, o que possibilitou a sua utilização ao longo da fase de desenvolvimento. Contudo, à medida que o projeto avançava, a Google decidiu descontinuar o suporte ao Manifesto v2, passando a priorizar a versão v3, conforme ilustrado na Figura 5.3. Esta mudança não foi acompanhada pelo Firefox, que continuou a suportar o Manifesto v2, criando assim uma discrepância significativa entre os dois navegadores durante o desenvolvimento da extensão.

Foi então decidido que, devido às restrições impostas pelo Manifesto v3, a extensão no Firefox manteria a compatibilidade com o Manifesto v2, enquanto no Chrome seria adotado o v3, aproveitando as melhorias e mudanças oferecidas por esta versão.

Outra diferença entre o desenvolvimento das duas extensões reside na forma como ambas obtêm o URL da página ativa. Enquanto o Chrome utiliza o comando `chrome.tabs.query`, o Firefox faz uso do `browser.tabs.query`. Embora à primeira vista esta diferença possa

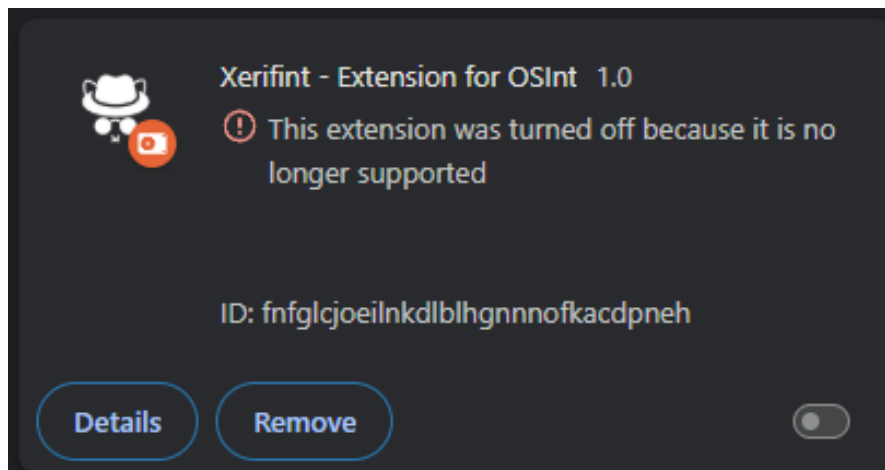


Figura 5.3: Erro que aparece no Chrome

parecer insignificante, se tentássemos fazer com que as extensões funcionassem nas plataformas opostas, ambas falhariam, pois cada navegador exige o uso de uma API específica, como pode ser observado nas Figuras 5.4 e 5.5, onde é notável a diferença entre o uso do browser e do chrome.

```
browser.tabs.query({ active: true, currentWindow: true })
  .then(tabs => {
    const currentUrl = tabs[0].url; // Get the URL of the active tab
```

Figura 5.4: Exemplo Código do Firefox

```
chrome.tabs.query({ active: true, currentWindow: true },function(tabs){
  const currentUrl = tabs[0].url; // Get the URL of the active tab
```

Figura 5.5: Exemplo Código do Chrome

## 5.2 Backend

Nesta secção, vamos abordar os endpoints da API, a criação do sistema de API keys, a integração do projeto com outras ferramentas *open source* e a adição do mecanismo de controlo de taxa (*rate limiter*).

### 5.2.1 Endpoints

No *backend*, decidiu-se implementar três endpoints: `/tools`, `/generator-API` e `/start-scan`. Os endpoints `/start-scan` e `/generator-API` utilizam o método GET, enquanto que o endpoint `/tools` utiliza o método POST, uma vez que neste último é necessário enviar um ficheiro em formato JSON para a aplicação. Cada endpoint tem o seguinte objetivo:

- `/start-scan`: Recebe o URL de um website, utiliza um LLM para analisar o HTML da página e, no final, retorna um ficheiro JSON com a informação extraída.

- /tools: Recebe o **JSON** gerado pelo endpoint /start-scan e aplica ferramentas de **OSINT** integradas para obter mais informações.
- /generator-API: Gera *API Keys* para que os utilizadores possam aceder ao *backend*.

Para criar estes endpoints usando o Flask, é necessário definir uma função associada a cada endpoint. O procedimento a seguir, dentro do ficheiro do backend, encontra-se exemplificado na Listagem 2.

**Listagem 2:** Definição dos endpoints no Flask

---

```
1 @app.route('/tools', methods=['POST'])
2 def use_tools():
3     pass
4
5 @app.route('/start-scan', methods=['GET'])
6 def start_scan():
7     pass
8
9 @app.route('/generator-API', methods=['GET'])
10 def gen_api():
11     pass
```

---

Desta forma, foram criados os endpoints da aplicação. De seguida, é necessário adicionar o suporte a *Cross-Origin Resource Sharing* (CORS). O CORS permite controlar quais domínios externos podem aceder à aplicação, garantindo segurança nas requisições feitas por clientes de origens diferentes.

A ativação do CORS na aplicação é feita conforme apresentado na Listagem 3.

**Listagem 3:** Ativação do CORS na aplicação Flask

---

```
1 app = Flask(__name__)
2 CORS(app)
```

---

Com o CORS configurado, a aplicação está preparada para funcionar. Para executar o servidor Flask, deve-se utilizar o código apresentado na Listagem 4.

**Listagem 4:** Ativação do CORS na aplicação Flask

---

```
1 if __name__ == "__main__":
2     app.run(debug=False, host='0.0.0.0', port=5000)
```

---

O IP 0.0.0.0 serve para indicar que o servidor deve aceitar ligações em todas as interfaces de rede, tornando a aplicação acessível a partir de qualquer endereço IP da máquina, sendo esta configurada para a porta 5000.

### 5.2.2 API Keys

Para garantir a segurança dos utilizadores da extensão, foi decidido implementar um sistema de **API Keys**. Para tal, foram seguidos os seguintes passos:

1. No *endpoint generator-API*, que apenas pode ser utilizado por pessoas que possuam a **API Key** principal. Esta chave está armazenada numa variável de ambiente do sistema;
2. Criar uma nova base de dados onde todas as **API Keys** deverão ser armazenadas;
3. Gerar a **API Key** como uma sequência aleatória de 16 caracteres, utilizando a biblioteca `secrets` do Python.

Graças a este sistema, torna-se necessário incluir no cabeçalho dos pedidos **Hyper-text Transfer Protocol (HTTP)** um novo campo, denominado `X-API-KEY`, para autenticar os acessos ao *backend*.

### 5.2.3 Integração com outras ferramentas

As ferramentas utilizadas estavam inicialmente organizadas em duas categorias distintas: ferramentas baseadas em **Command-Line Interface (CLI)** e ferramentas executadas em contentores Docker. Para além dessa distinção, verificou-se que cada uma destas ferramentas exportava os seus dados em formatos diferentes, o que dificultava a consolidação e o tratamento uniforme da informação.

Dado este cenário, tornou-se necessário estabelecer um padrão comum para o formato dos dados resultantes das ferramentas, de forma a garantir a compatibilidade e a facilitar o processamento posterior. Após análise, decidiu-se converter e uniformizar todos os *outputs* das ferramentas para o formato **JSON**, uma vez que este oferece uma estrutura legível e flexível, sendo particularmente vantajoso pela facilidade com que pode ser manipulado em *JavaScript* e noutras linguagens de programação.

Nas próximas secções, iremos abordar de que forma cada ferramenta foi implementada e como foi realizado o processo de conversão dos seus *outputs* para o formato **JSON**.

#### Sherlock

No caso do *Sherlock*, foi necessário instalá-lo através do `pip` (o *package manager* para *Python packages*). Para executar o *Sherlock* no *backend*, foi necessário iniciar um sub-processo que permitisse a sua execução da ferramenta no **CLI** do *backend*. Para isso, recorreu-se à biblioteca `subprocess`, que possibilita não só a execução de comandos externos como também o armazenamento do *output* numa variável, conforme ilustrado na Figura 5.6.

No caso do *Sherlock*, o comando é composto e executado da seguinte forma: `sherlock username`, em que `username` corresponde à variável passada pela função e o `sherlock` serve para executar o programa.

```
result = subprocess.run(['sherlock', username],
                        stdout=subprocess.PIPE,
                        stderr=subprocess.PIPE,
                        text=True)
```

Figura 5.6: Código para executar o subprocesso do Sherlock

Para converter o *output* que saiu da ferramenta para o formato **JSON**, utilizou-se uma combinação de **regular expression (regex)** e separação por linhas. O processo foi realizado seguindo os passos descritos abaixo:

1. Criar uma lista vazia onde será armazenado o *output* final.
2. Separar o texto do *output* em linhas, permitindo que cada linha seja analisada individualmente.
3. Analisar cada linha a procura do **URL** e do nome da aplicação usando o seguinte **regular expression (regex)**.

```
\[\+\] (.*)?: (https?://\S+)
```

4. Caso ambos sejam encontrados, guardar o nome da aplicação e o respectivo **URL** na lista.
5. Repetir este processo para todas as linhas até que todas tenham sido analisadas.
6. Devolver a lista final com os dados estruturados.

Com isso, o *output* foi transformado do formato apresentado na Figura 5.7a para o formato **JSON** ilustrado na Figura 5.7b.

```
*) Search completed with 17 results
{
  "Chess": "https://www.chess.com/member/euntenhoideias",
  "DigitalSpy": "https://forums.digitalspy.com/profile/eu",
  "Discord": "https://discord.com",
  "Freelance.habr": "https://freelance.habr.com/freelance",
  "GNOME VCS": "https://gitlab.gnome.org/euntenhoideias",
  "kaskus": "https://www.kaskus.co.id/@euntenhoideias",
```

(a) Output proveniente do CLI

```
"sherlock": [
  {
    "Chess": "https://www.chess.com/member/euntenhoideias",
    "DigitalSpy": "https://forums.digitalspy.com/profile/eu",
    "Discord": "https://discord.com",
    "Freelance.habr": "https://freelance.habr.com/freelance",
    "GNOME VCS": "https://gitlab.gnome.org/euntenhoideias",
```

(b) Output convertido para JSON

Figura 5.7: Comparação entre o output proveniente do CLI e o output convertido para JSON.

## Holehe

No caso do *Holehe*, foi necessário garantir que a ferramenta estivesse instalada no sistema e disponível para execução via linha de comandos. A sua execução no *backend*

exigiu a criação de um subprocesso, utilizando a biblioteca `subprocess`, permitindo assim correr o comando directamente no terminal do sistema operativo a partir do código Python. Para além disso, este método possibilita capturar o *output* do comando numa variável para posterior tratamento, conforme ilustrado na Figura 5.8.

```
result = subprocess.run(['holehe', email],
                        stdout=subprocess.PIPE,
                        stderr=subprocess.PIPE,
                        text=True)
```

Figura 5.8: Código para executar o subprocesso do Holehe

A execução do *Holehe* é feita através do seguinte comando: `holehe email`, onde `email` corresponde ao endereço de e-mail passado à função e utilizado para verificar a presença desse endereço em diversos serviços online.

Para processar o *output* devolvido pela ferramenta e convertê-lo para o formato **JSON**, foi utilizada uma abordagem simples baseada em **regular expression (regex)**, com o objectivo de extrair os domínios associados ao e-mail fornecido sendo muito parecido com o Sherlock. O procedimento seguiu os passos descritos a seguir:

1. Executar o comando e capturar o *output* gerado pela ferramenta.
2. Utilizar a função `findall` da biblioteca `re` para identificar todas as linhas que contenham a tag `[+]` seguida de um domínio.
3. O **regular expression (regex)** utilizado para esta extração dos domínios foi:

```
\+\+\s+(\[^\s]+\)
```

4. Armazenar todos os domínios encontrados numa lista.
5. Devolver essa lista no formato **JSON**.

Depois desses passos o *output* proveniente do **CLI** passou dos dados apresentados na Figura 5.9a para o **JSON** apresentado na Figura 5.9b.

```
[x] about.me
[x] adobe.com
[+] amazon.com
[x] amocrm.com
[-] any.do
[-] archive.org
[-] armurerie-auxerre
[x] atlassian.com
[-] axonaut.com
[x] babeshows.co.uk
```

(a) Output proveniente do CLI

```
"heluho": [
  "amazon.com",
  "replit.com",
  "spotify.com",
  "twitter.com",
  "wordpress.com",
  "Email"
],
```

(b) Output convertido para JSON

Figura 5.9: Comparação entre o output proveniente do CLI e o output convertido para JSON.

## PhoneInfoga

O *PhoneInfoga* foi o único caso em que, ao invés de ser executado diretamente através da **CLI**, optou-se pela sua execução em ambiente *Docker*. Esta decisão deveu-se ao facto de a instalação via *Docker* ser mais simples e estável quando comparada com a alternativa em **CLI**.

Para além disso, o *PhoneInfoga* revelou-se a ferramenta mais complexa de implementar, em comparação com as restantes, sobretudo devido ao volume e à variabilidade do seu *output*. Tal como nas outras ferramentas, foi necessário recorrer à execução de um subprocesso, de modo a permitir a captura do *output* directamente numa variável — etapa fundamental para possibilitar a posterior transformação dos dados utilizando **regular expression (regex)**.

Para tal foi feito o que é demonstrado na Figura 5.10.

```
command = ["docker", "run", "-it", "sundowndev/phoneinfoga", 'scan', '--number', phone_number]
result = subprocess.run(command, capture_output=True, text=True, check=True)
```

Figura 5.10: Código para executar o subprocesso do *Docker* do *phoneinfoga*

Sendo que o comando `docker run -it sundownlev\phoneinfoga scan --number +351911111111` é utilizado para executar o *PhoneInfoga* e declarando que o número de telemóvel será utilizado para o scan será neste caso o +351 911 111 111.

Para processar o *output* devolvido pela ferramenta *PhoneInfoga* e convertê-lo para o formato **JSON**, utiliza-se novamente o **regular expression (regex)** e também o método de separação por linhas. Esta abordagem permite identificar e extrair dados estruturados, tais como números de telefone, formatos locais, país, número de scanners e resultados provenientes da pesquisa no Google, que é a fonte utilizada pelo *PhoneInfoga*. O processo segue os passos descritos abaixo:

1. Remover códigos de formatação **American National Standards Institute (ANSI)** do *output* com recurso do **regular expression (regex)**, para obter um texto limpo sendo o seguinte:

```
\x1B(?:[@-Z\\-_]|\\[[0-?]*[ -/]*[@-~])
```

2. Procurar e extrair o número de telefone com a seguinte expressão regular:

```
Running scan for phone number (\+?\d+)
```

3. Dividir o texto linha a linha e, com base em palavras-chave, identificar a secção atual da pesquisa do Google (por exemplo, *Social media*, *Reputation*, etc.).
4. Para cada linha dentro dessas secções, procurar **URL** com o seguinte **regular expression (regex)**:

```
URL:\s*(https?://[\s]+)
```

e adicionar o resultado à secção correspondente no campo `google_search`.

5. Identificar e guardar os diferentes formatos do número (como `raw_local`, `local`, `e164`, `international`, etc.) usando expressões regulares específicas.
6. Extrair também o país associado ao número e o número de scanners que tiveram sucesso na análise.
7. Guardar todos os dados estruturados num **JSON**.

Sendo que converteu o *output* demonstrado na Figura 5.11a para o *output* da Figura 5.11b.

```
Running scan for phone number ...
Results for googlesearch
Social media:
  URL: https://www.google.com/search?q=site%3Afacebook.
  URL: https://www.google.com/search?q=site%3Atwitter.c
  URL: https://www.google.com/search?q=site%3Alinkedin.
  URL: https://www.google.com/search?q=site%3Ainstagram
  URL: https://www.google.com/search?q=site%3Avk.com+in
Disposable providers:
  URL: https://www.google.com/search?q=site%3Ahs3x.com+
  URL: https://www.google.com/search?q=site%3Areceive-s
```

(a) Output proveniente do Docker

```
"phoneinfoga": {
  "country": "PT",
  "formats": {
    "e164": "+351 [REDACTED]",
    "international": "[REDACTED]",
    "local": "[REDACTED]",
    "raw_local": "[REDACTED]"
  },
  "google_search": {
    "disposable_providers": [
      "https://www.google.com/search?q=site%3Ahs3x.com+in",
      "https://www.google.com/search?q=site%3Areceive-sms",
      "https://www.google.com/search?q=site%3Aasmstlisten.c",
      "https://www.google.com/search?q=site%3Aasmstlisten.c"
    ]
  }
}
```

(b) Output convertido para JSON

**Figura 5.11:** Comparação entre o output proveniente do Docker e o output convertido para JSON.

## GoogleDork

Para início de discussão, embora a técnica de *Dorking* seja frequentemente associada ao Google — e essa tenha sido, inicialmente, a abordagem considerada para este projeto — decidiu-se utilizar o *DuckDuckGo* em vez do Google. Esta decisão foi tomada com base em vários fatores, nomeadamente:

1. Existe uma maior probabilidade de o Google bloquear requisições automatizadas, comprometendo a estabilidade do processo;
2. Os resultados obtidos através do *DuckDuckGo* são, na maioria dos casos, muito semelhantes aos apresentados pelo Google.

3. Manter a premissa do projeto, assegurando a utilização de ferramentas *open source* ou, caso tal não seja possível, privilegiar aquelas que possuam, pelo menos, alguns componentes de código aberto.

Como o *DuckDuckGo* não disponibiliza uma biblioteca oficial em Python que permita realizar pesquisas de forma direta, então optou-se por recorrer à técnica de *web scraping*. Esta abordagem consiste em extrair conteúdo diretamente da página **HTML** dos resultados.

Para implementar esta solução, foram utilizadas as bibliotecas `requests` e `BeautifulSoup`, amplamente reconhecidas pela sua eficácia em operações de *web scraping*. A estratégia adotada consistiu nos seguintes passos:

1. Definiu-se uma função `get_random_user_agent()` que selecciona aleatoriamente um `User-Agent` a partir de uma lista pré-definida, para evitar bloqueios do *DuckDuckGo*.
2. Utilizou-se o `requests` para fazer o pedido **HTTP** ao link `https://html.duckduckgo.com/html/?q=nome_da_pessoa` em que o `nome_da_pessoa` é o nome da pessoa a ser analisado, tendo no header o `User-Agent` escolhido no passo 1.
3. O conteúdo **HTML** da resposta foi processado através da `BeautifulSoup`, permitindo uma maior facilidade de manuseio do **HTML**.
4. Foram seleccionados apenas os elementos que contêm os **URL** dos resultados, através do seletor `CSS` `.result__url`, e extraídos os respectivos links.
5. Foi adicionado um filtro para ignorar anúncios ou resultados inválidos (por exemplo, links que começam por `//ad.`).
6. Por fim, os links válidos foram armazenados numa lista e devolvidos em **JSON**.

Por norma, apenas os 10 primeiros resultados devolvidos pelo *DuckDuckGo* são analisados, de forma a tornar o processo mais leve e eficiente para o script. Dito isto, o resultado do *output* pode ser observado na Figura 5.12.

```
"googledork": [  
  "https://en.wikipedia.org/wiki/Data",  
  "https://www.ldoceonline.com/dictionary/data",  
  "https://dictionary.cambridge.org/us/dictionary/learner-english/data",  
  "https://www.thefreedictionary.com/data",  
  "https://byjus.com/maths/introduction-to-data/",  
  "https://www.geeksforgeeks.org/what-is-data/",  
  "https://pg-p.ctme.caltech.edu/blog/data-science/what-is-data",  
  "https://www.ibm.com/think/topics/data",  
  "https://data.gov/",  
  "https://formfull.in/data"  
],
```

Figura 5.12: Output do *DuckDuckGo*

### 5.2.4 Rate Limiter

Um *rate limiter* serve para limitar o número de pedidos feitos ao *backend*, permitindo assim um maior controlo sobre o uso dos recursos do sistema e ajudando a prevenir ataques de negação de serviço ou em inglês **Denial of Service (DOS)**.

Embora esta técnica seja eficaz contra ataques **Denial of Service (DOS)**, não é suficiente para travar ataques de negação de serviço distribuídos ou em inglês **Distributed Denial of Service (DDOS)**, uma vez que estes envolvem múltiplas origens e requerem soluções mais robustas e distribuídas.

A implementação de um *Rate Limiter* torna-se bastante simples graças à utilização do MongoDB para a gestão das *API Keys* e do Flask como framework para o desenvolvimento do *backend*. Esta combinação evita a necessidade de criar um sistema de limitação de pedidos a partir do zero, permitindo o uso da biblioteca Flask-Limiter, que se integra de forma eficiente com o Flask.

A configuração foi realizada de acordo com os seguintes passos:

1. Instalar a biblioteca Flask-Limiter através do comando:

```
pip install flask_limiter
```

2. Importar os módulos necessários no início do ficheiro:

```
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address
```

3. No *backend*, criar a variável `limiter` com a configuração pretendida, definindo os limites de requisições, a função de chave (com base no **IP** do utilizador), a URI da base de dados e a estratégia de limitação:

```
limiter = Limiter(
    app=app,
    key_func=get_remote_address,
    default_limits=["200 per day", "50 per hour"],
    storage_uri="mongodb://localhost:27017",
    strategy="fixed-window",
)
```

4. Aplicar os limites às rotas pretendidas através do decorator `@limiter.limit()`. Por exemplo, para limitar uma rota a uma pedido por minuto:

```
@limiter.limit("1 per minute")
```

## 5.3 LLM

Nesta secção, iremos detalhar o processo de integração de modelos de **LLM** no projeto, com especial foco na utilização da ferramenta `ollama`. Serão abordados os principais passos técnicos envolvidos na instalação e configuração da aplicação, bem como

a utilização da sua biblioteca em ambiente Python para facilitar a comunicação com os modelos.

Além disso, discutiremos alguns dos principais desafios enfrentados ao longo do desenvolvimento, com destaque para o fenómeno das chamadas alucinações em LLM — situações em que o modelo gera respostas incorretas, incoerentes ou fora do contexto solicitado. Estas alucinações, apesar de relativamente comuns, podem comprometer a fiabilidade dos resultados, especialmente em tarefas sensíveis como a extração de informação.

Será ainda apresentada a abordagem adotada para minimizar estes problemas, incluindo a definição de um *input* padronizado com base em boas práticas já consolidadas na comunidade através do projeto Fabric de Daniel Miessler [55]. Através desta estrutura e validação cuidadosa das saídas, procurou-se garantir uma maior estabilidade e consistência nos *outputs* dos modelos utilizados.

### 5.3.1 Integração

A integração do LLM em Python é facilitada pelo uso da biblioteca `ollama` e da aplicação com o mesmo nome. A aplicação permite o download dos LLM diretamente para a máquina local, enquanto a biblioteca `ollama` possibilita o uso dos LLM instalados pela aplicação, incluindo o LLAMA3 e o DeepSeek, por meio da API. Para instalar os LLM na máquina local, é necessário ter a aplicação do `ollama` instalada. A seguir, apresentamos os passos para realizar essa instalação da aplicação e de como adicionar no python:

1. Aceder ao site oficial do `ollama` em <https://ollama.com>.
2. Na página inicial, clicar em *Download* para baixar a versão mais recente do `ollama` para o sistema operativo, no meu caso foi Windows.
3. Após o *download*, seguir as instruções de instalação para o sistema operativo selecionado.
4. Para verificar se o `ollama` esta instalado usando o seguinte comando no terminal:

```
ollama --version
```

5. Para fazer *download* de um modelo específico, por exemplo, para o DeepSeek R1 usa-se o seguinte comando:

```
ollama pull deepseek-r1
```

6. Para verificar se o *download* foi feito com sucesso pode-se testar o modelo fazendo o seguinte comando:

```
ollama run deepseek-r1
```

7. Agora pode-se proceder à instalação da biblioteca, através do seguinte comando:

```
pip install ollama
```

8. Para usar o ollama no Python é preciso fazer um *import* adicionando ao início do código o seguinte:

```
import ollama
```

9. Por último, para integrarmos o ollama no código pode-se fazer o seguinte:

```
response = ollama.chat(model= "NOME DO MODELO PRETENDIDO", messages=[
    {
        'role': 'user',
        'content': 'ESCREVER AQUI O INPUT',
    },
])
```

Para garantir maior consistência e precisão nos resultados gerados pelos LLM, foi definido um *input* padrão que todos os modelos utilizados deverão seguir. Este *input* foi concebido com base em *patterns* já estabelecidos no projeto Fabric [55], desenvolvido por Daniel Miessler, mais especificamente no *pattern* denominado *analyse logs*. Assim, foi definido o seguinte *input*, no qual o campo `html code` representa o local onde o código HTML será inserido durante o processamento, conforme demonstrado na Listagem 5.

**Listagem 5:** *Input padronizado baseado no pattern analyse logs do projeto Fabric.*

---

```
1 You are a data extraction model. I will provide you with an HTML document containing
  ↳ various types of information. Your task is to extract the following data points, if
  ↳ present, and return them as a JSON object. Format the JSON output as a single-line
  ↳ object.
2
3 1. **Full Name**: Extract the full name of the individual or entity.
4 2. **Email Address**: Look for email addresses.
5 3. **Phone Number**: Extract any phone number or contact information.
6 4. **Physical Address**: Extract a physical address, including street address, city,
  ↳ state, and zip code.
7 5. **Social Media Links**: Extract links to any social media profiles (LinkedIn,
  ↳ Twitter, etc.).
8 6. **Photos**: Extract any image URLs or descriptions of photos found in the HTML.
9 7. **Job Title/Profession**: Extract the person's job title or profession, if available.
10 8. **Company/Organization**: Extract any company or organization names mentioned in the
  ↳ HTML.
11 9. **Date of Birth**: Extract date of birth or age-related information.
12 10. **Other Relevant Details**: Any additional relevant details found in the HTML
  ↳ (e.g., hobbies, qualifications, etc.).
13 11. **Usernames or Users**: Extract any usernames or online handles, even if they are
  ↳ not in the form of direct links. These could be mentioned in informal ways, like in
  ↳ sentences or phrases, e.g., "If you want to play any games with me, my username is
  ↳ x."
```

---

---

```

1 If any of these data points are found in the HTML, include them in the JSON output. If
  ↳ no data is found for a particular field, leave it blank or set it to `null`.
2 For images, include the image URLs or a description if the URL is not provided.
3
4 Here is the HTML document:
5
6 html_code
7
8 Please output only the extracted data in the following JSON format (single-line):
9
10 Example Output Format (single-line):
11 {"full_name":"John Doe"}, "email":"johndoe@example.com", "phone":"+123456789",
  ↳ "address":"1234 Elm St, Springfield, IL",
  ↳ "social_media":{"linkedin":"https://linkedin.com/in/johndoe",
  ↳ "twitter":"https://twitter.com/johndoe"},
  ↳ "photos":["https://example.com/photo1.jpg","https://example.com/photo2.jpg"],
  ↳ "job_title":"Software Engineer", "company":"TechCorp",
  ↳ "website":"https://techcorp.com", "dob":"1990-05-15",
  ↳ "other_details":{"hobbies":["Reading", "Traveling"], "skills":["Python", "Machine
  ↳ Learning"]}, "usernames":["imamaster"]}

```

---

Ao adotar este modelo como referência, garante-se uma estrutura uniforme de entrada para todos os **LLM** integrados no sistema, o que facilita o processamento das respostas. Além disso, esta uniformização contribui para a mitigação de possíveis alucinações geradas pelos modelos.

### 5.3.2 Dificuldades encontrados com a LLM

Em seguida, vamos avaliar as dificuldades encontradas durante a implementação do **LLM**, incluindo problemas como as alucinações.

#### Alucinação

Uma alucinação em IA ocorre quando um modelo de Inteligência Artificial gera ou interpreta dados incorretos, mistura informações ou fornece respostas confusas. Isso acontece porque os modelos atuais nem sempre conseguem gerar ou interpretar dados de forma 100% confiável. Mesmo com um *input* estável, a IA pode esquecer passos importantes ou até o objetivo original. Um exemplo disso, durante o desenvolvimento da extensão (em específico com o LLAMA3), ocorreu quando, em vez de fornecer o **JSON** solicitado, a IA entregou um código JavaScript explicando como realizar o processo como pode ser visto no código demonstrado na Listagem 6.

---

**Listagem 6: Código gerado pelo LLAMA3**


---

```

1 function extractPersonalInformation(html) {
2   let personalInfo = {};
3   let name = html.querySelectorAll('h1')[0].textContent;
4   if (name) {
5     personalInfo.name = name;
6   }
7   let contactSection = html.querySelector('#contact');
8   if (contactSection) {
9     let emails = contactSection.querySelectorAll('p strong:nth-child(1)');
10    if (emails.length > 0) {
11      personalInfo.email = emails[0].textContent.replace('Email:', '');
12    }
13    let phones = contactSection.querySelectorAll('p strong:nth-child(2)');
14    if (phones.length > 0) {
15      personalInfo.phone = phones[0].textContent.replace('Phone:', '');
16    }
17    let addresses = contactSection.querySelectorAll('p strong:nth-child(3)');
18    if (addresses.length > 0) {
19      personalInfo.address = addresses[0].textContent.replace('Address:', '');
20    }
21  }
22  return JSON.stringify(personalInfo, null, 2);
23 }

```

---

Um outro exemplo ocorreu com o LLM DeepSeek. Embora o *input* exija a criação do ficheiro JSON numa única linha, o DeepSeek opta por gerá-lo em múltiplas linhas, provavelmente com o intuito de facilitar o processamento, como se pode observar na Listagem 7.

```

Please output only the extracted data in the following JSON format
↪ (single-line):
{"full_name": "John Doe", "email": "johndoe@example.com",
↪ "phone": "+123456789", "address": "1234 Elm St, Springfield, IL",
↪ "social_media": {"linkedin": "https://linkedin.com/in/johndoe",
↪ "twitter": "https://twitter.com/johndoe"},
↪ "photos": ["https://example.com/photo1.jpg", "https://example.com/photo2.jpg"],
↪ "job_title": "Software Engineer", "company": "TechCorp",
↪ "website": "https://techcorp.com", "dob": "1990-05-15",
↪ "other_details": {"hobbies": ["Reading", "Traveling"],
↪ "skills": ["Python", "Machine Learning"]},
↪ "usernames": ["euntenhoideias"]}

```

**Listagem 7:** *Input a determinar, cujo resultado só pode ser gerado numa única linha.*

Mesmo com a implementação de processos de segurança e a definição de um *input* padronizado que a IA deve seguir, como foi feito neste projeto, as alucinações ainda podem ocorrer. Modelos de linguagem, por mais avançados que sejam, não garantem

100% de fiabilidade na geração de respostas.

## 5.4 Segurança

Nesta secção, é apresentada a abordagem adotada para garantir a segurança durante o desenvolvimento, recorrendo à utilização de ferramentas **SAST** e **DAST**. Serão igualmente descritas as medidas implementadas para assegurar a proteção dos utilizadores, nomeadamente através da configuração adequada dos manifestos do Python e da extensão.

Adicionalmente, será detalhado o processo de criação da *pipeline* responsável pela execução automática das ferramentas **SAST**.

### 5.4.1 Como garantir a segurança durante o desenvolvimento

Para garantir a segurança durante o desenvolvimento da aplicação, foi adotada uma abordagem que combina ferramentas dos tipos **Static Application Security Testing (SAST)** e **Dynamic Application Security Testing (DAST)**. Para manter o objetivo deste projeto as ferramentas **SAST** e **DAST** utilizadas também são open source.

No âmbito da análise estática, foram utilizadas quatro ferramentas: MyPy, Bandit, Safety e Semgrep, cada uma com o objetivo de identificar vulnerabilidades no código-fonte e nas dependências do projeto.

Complementarmente, para a análise dinâmica, foram utilizadas duas ferramentas: OWASP ZAP e Nikto, com o intuito de detetar falhas de segurança durante a execução da aplicação e no seu ambiente web.

#### SAST

Para garantir uma cobertura mais ampla e eficaz na deteção de vulnerabilidades, decidiu-se que a utilização de quatro ferramentas especializadas em áreas distintas seria a melhor abordagem para assegurar a máxima segurança. Para tal, foram escolhidas as seguintes ferramentas:

**MyPy** O MyPy é uma ferramenta de type checking para Python que permite verificar estaticamente os tipos de variáveis, funções e expressões no código. Ao utilizar anotações de tipo, o MyPy ajuda a identificar erros comuns relacionados a incompatibilidades de tipos antes da execução, aumentando a segurança e a robustez do software [33].

Para realizar uma verificação de tipos estáticos no código Python com o MyPy, pode-se executar o seguinte comando:

```
mypy nome_do_ficheiro.py
```

Este comando verifica se o código cumpre as anotações de tipo definidas.

**Bandit** O Bandit destaca-se na detecção de falhas de segurança como o uso inseguro de funções como `eval` e `exec`, bem como palavras-passe no código (*hardcoded passwords*). Para além da sua eficácia, é uma ferramenta bastante simples de configurar e utilizar [4].

Para utilizar a ferramenta *Bandit*, pode-se executar o seguinte comando:

```
bandit .\main.py
```

Este comando analisa o ficheiro `main.py` em busca de vulnerabilidades.

**Safety** O Safety é uma ferramenta focada na detecção de vulnerabilidades em dependências de pacotes Python. Ele verifica se as bibliotecas utilizadas no projeto têm versões com falhas de segurança conhecidas, baseando-se em uma base de dados constantemente atualizada [43].

Para verificar a existência de vulnerabilidades nas bibliotecas utilizadas usando o Safety, pode-se executar o seguinte comando:

```
safety check --file=requirements.txt
```

O comando verifica se as bibliotecas listadas no ficheiro `requirements.txt` contêm vulnerabilidades de segurança conhecidas.

**Semgrep** O Semgrep é uma ferramenta de análise estática de código que combina a simplicidade das expressões regulares com o poder da análise sintática, permitindo a detecção rápida de padrões inseguros e vulnerabilidades em múltiplas linguagens de programação [44].

Para realizar uma análise rápida do código com o Semgrep, pode-se executar o seguinte comando:

```
semgrep scan --config auto
```

## DAST

Para simular um ataque de um hacker à aplicação, foram utilizadas ferramentas de **Dynamic Application Security Testing (DAST)**. Tendo em mente a necessidade de ferramentas confiáveis e com documentação extensa, foram escolhidas aquelas com as quais já tinha maior familiaridade e confiança. As ferramentas selecionadas para esta tarefa foram as seguintes:

**OWASP ZAP** OWASP ZAP (Zed Attack Proxy) é uma das ferramentas de segurança mais populares para a realização de testes de penetração em aplicações web. Oferece uma vasta gama de funcionalidades, incluindo análise automatizada, intercetção de tráfego, fuzzing e varreduras de segurança. O OWASP ZAP é particularmente útil para identificar falhas como injeções SQL, **Cross-Site Scripting (XSS)**, e outros tipos de

vulnerabilidades web, sendo ideal para testar aplicações durante o desenvolvimento e operação [38].

Para utilizar o OWASP ZAP, é necessário seguir os seguintes passos:

1. Efetuar o download na página oficial do OWASP ZAP: <https://www.zaproxy.org/>;
2. Após concluir o download, abrir a aplicação e selecionar a opção “Automated Scan”, conforme ilustrado na Figura 5.13;

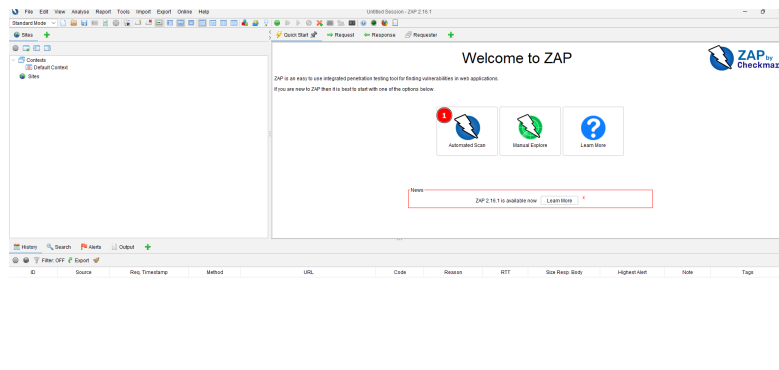


Figura 5.13: Seleção da opção “Automated Scan” na interface do OWASP ZAP

3. Introduzir o URL da aplicação ou do backend e clicar no botão “Attack”, conforme mostrado na Figura 5.14;

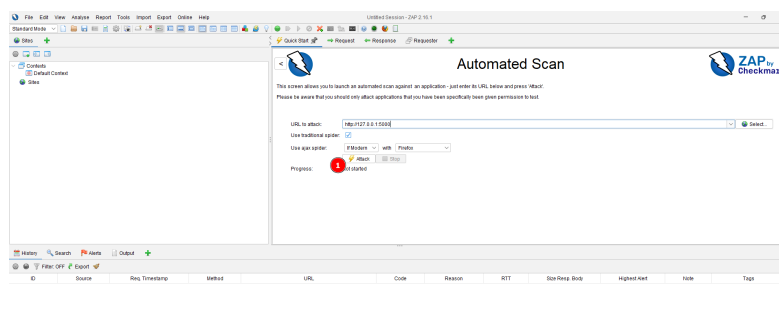


Figura 5.14: Introdução do URL e ativação do ataque através do botão “Attack” no OWASP ZAP

4. Aguardar até que a ferramenta termine o processo.

**Nikto** Nikto é uma ferramenta de análise de segurança dinâmica que realiza varreduras em servidores web em busca de vulnerabilidades conhecidas, como configurações inseguras, versões desatualizadas de software e falhas comuns de segurança [35].

Para utilizar o Nikto, é necessário seguir os seguintes passos:

- Efetuar o download a partir da página oficial do projeto no GitHub: <https://github.com/sullo/nikto>, seguindo as instruções disponíveis no repositório.
- Para executar o Nikto, deve utilizar o seguinte comando no terminal:

```
perl nikto.pl -h http://www.example.com
```

Com este comando, o Nikto realizará uma análise ao endereço `http://www.example.com`, procurando possíveis vulnerabilidades.

## 5.4.2 Segurança garantida pelos manifestos

A utilização de manifestos é uma prática fundamental para garantir a segurança e a previsibilidade em projetos de software. Estes ficheiros definem informações cruciais sobre dependências, permissões e comportamentos esperados da aplicação, permitindo uma gestão mais rigorosa e segura dos seus componentes. Nesta secção, abordam-se dois casos específicos: aplicações Python e extensões de navegador.

### Manifestos em Projetos Python

No contexto de aplicações Python, o ficheiro `requirements.txt` assume o papel de manifesto principal. Nele são declaradas todas as bibliotecas externas utilizadas pelo projeto, geralmente com a respetiva versão fixa, de forma a evitar a criação de vulnerabilidades do tipo *zero-day*.

As vulnerabilidades *zero-day* são falhas de segurança desconhecidas pelos responsáveis pelo software e que, por esse motivo, não dispõem ainda de correção ou *patch*. Estas vulnerabilidades podem ser exploradas por atacantes antes de serem detetadas e corrigidas, representando um risco significativo para a segurança das aplicações.

As vantagens, do ponto de vista da segurança, que o ficheiro `requirements.txt` traz são as seguintes:

- **Fixação de versões:** Definir versões específicas evita atualizações automáticas que possam introduzir vulnerabilidades ou incompatibilidades;
- **Auditorias de segurança:** Ferramentas como `Safety` permitem verificar se as bibliotecas listadas têm vulnerabilidades conhecidas, com base em bases de dados de [CVE](#);
- **Reprodutibilidade:** Garante que todos os ambientes (desenvolvimento, testes e produção) estejam a usar a mesma versão;

Além do `requirements.txt`, pode também ser utilizado um ficheiro `pip.lock` (ou similar) para registar de forma mais rigorosa todas as dependências, incluindo as transitivas.

### Manifestos em Extensões de Navegador

Nas extensões de navegador, o ficheiro `manifest.json` é obrigatório e atua como manifesto principal da extensão. Ele define as permissões necessárias, os scripts a executar, os domínios acessíveis e outras configurações críticas.

Do ponto de vista da segurança, este manifesto é essencial:

- **Permissões declaradas:** A extensão só poderá aceder aos recursos explícitos no manifesto, limitando potenciais abusos;

- **Política de conteúdo ou Content Security Policy (CSP):** É possível impor regras sobre as fontes de onde scripts e outros recursos podem ser carregados, ajudando a prevenir ataques como **XSS**;
- **Isolamento de funcionalidades:** Através da separação entre *background scripts*, *content scripts* e permissões declaradas, é possível controlar o comportamento da extensão de forma granular.

Ao manter o `manifest.json` bem estruturado e com permissões mínimas necessárias, reduz-se significativamente a superfície de ataque da extensão.

### 5.4.3 Criação da Pipeline

Sabendo que o projeto está armazenado no GitHub, foi decidido criar uma *pipeline* de integração contínua utilizando o *GitHub Workflow*. Para esse efeito, optou-se por executar ferramentas de análise estática de código, conhecidas como **SAST**.

Foi então criado o ficheiro `security_pipeline.yml`, no qual foram definidas as seguintes etapas:

1. **Executar a pipeline num ambiente Ubuntu:** Escolhido por ser um sistema amplamente compatível com as ferramentas utilizadas.
2. **Configurar o ambiente Python:** Utiliza-se a ação `setup-python` para instalar a versão necessária do Python.
3. **Instalar as dependências:** Instaladas com base no ficheiro `requirements.txt`.
4. **Executar o Bandit:**

```
bandit -r . -lll -ii
```

- `-r .` indica que a análise deve ser feita de forma recursiva no diretório atual;
- `-lll` define o nível mínimo de severidade dos alertas como “baixo” (*low*);
- `-ii` mostra apenas resultados com nível de confiança elevado (*high confidence*);

5. **Executar o Safety:**

```
safety check --full-report --file=requirements.txt
```

- `--full-report` apresenta um relatório detalhado dos resultados;
- `--file=requirements.txt` especifica que o ficheiro `requirements.txt` deve ser usado para a análise das dependências;

6. **Executar o MyPy:** Primeiro, cria-se dinamicamente o ficheiro de configuração como pode ser visto na Listagem 8.

**Listagem 8:** Ficheiro de configuração utilizado para o MyPy.

```
1 echo "[mypy]
2 ignore_missing_imports = True
3 disallow_untyped_defs = True
4 show_error_codes = True
5 no_strict_optional = True " > mypy.ini
```

- `ignore_missing_imports = True` ignora erros por falta de anotações em módulos externos;
- `disallow_untyped_defs = True` exige que todas as funções tenham tipos definidos;
- `show_error_codes = True` exibe os códigos de erro no *output*;
- `no_strict_optional = True` desactiva inferência de `Optional` como padrão;

Depois, executa-se o comando:

```
mypy --config-file mypy.ini . || true
```

- `--config-file mypy.ini` indica o ficheiro de configuração a ser usado;
- `.` aplica a análise ao diretório atual;
- `|| true` impede que a *pipeline* falhe se forem detetados erros.

7. **Executar o Semgrep:** Para executar o Semgrep é necessário executar o seguinte comando:

```
semgrep scan --config auto
```

- `scan` inicia o processo de análise;
- `--config auto` faz com que o Semgrep selecione automaticamente regras baseadas no código detetado.

Tendo isso em mente, foi então criado o ficheiro `security_pipeline.yml`, como pode ser visto na Listagem 16 no Anexo B.

## 5.5 Publicação

Nesta secção, é abordada a criação do ficheiro `Dockerfile` e do `docker-compose.yml`, fundamentais para dividir a aplicação em contentores independentes e isolados. Além disso, será explicado como é possível executar o *backend* tanto em ambiente *self-hosted* como em plataformas de *cloud computing*, nomeadamente a [Amazon Web Services \(AWS\)](#) e a [Google Cloud Platform \(GCP\)](#).

### 5.5.1 Criação do Dockerfile e Configuração do Docker Compose

Para facilitar a execução e o *deployment* da aplicação num ambiente controlado e replicável, foi utilizada a tecnologia Docker. Através da criação de um *Dockerfile* e da

configuração de um ficheiro `docker-compose.yml`, foi possível definir os serviços necessários, gerir dependências e preparar a aplicação para implementação na AWS.

### Dockerfile da Aplicação Principal

O Dockerfile da aplicação principal foi desenvolvido com base na imagem `python:3.11-slim`, garantindo uma base leve e otimizada. Os passos definidos neste ficheiro incluem:

- Definição do directório de trabalho como `/app`;
- Instalação de pacotes essenciais como `docker.io`, `git`, `libcurl4-openssl-dev`, `libssl-dev`, e `jq`;
- Instalação da ferramenta `Sherlock`;
- Instalação da ferramenta `PhoneInfoga`;
- Instalação das dependências com `pip`;
- Cópia do código da aplicação;
- Exposição da porta 5000;
- Execução da aplicação com o comando `python main.py`.

Com estes passos, foi possível a criação do Dockerfile, como pode ser visto na Listagem 9.

```
FROM python:3.11-slim

WORKDIR /app

RUN apt-get update && apt-get install -y docker.io git
  ↪ libcurl4-openssl-dev libssl-dev jq && rm -rf
  ↪ /var/lib/apt/lists/*

RUN git clone https://github.com/sherlock-project/sherlock.git
  ↪ /tmp/sherlock && cd /tmp/sherlock && pip install . && rm -rf
  ↪ /tmp/sherlock

RUN apt-get update && apt-get install -y wget && wget
  ↪ https://github.com/sundowndev/phoneinfoga
  ↪ /releases/latest/download/phoneinfoga_Linux_x86_64.tar.gz &&
  ↪ tar -xvzf phoneinfoga_*.tar.gz && mv phoneinfoga
  ↪ /usr/local/bin/

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "main.py"]
```

**Listagem 9:** *Dockerfile da Aplicação Principal*

## Dockerfile para o Serviço Ollama

O serviço Ollama foi isolado num Dockerfile próprio, que utiliza como base a imagem oficial ollama/ollama. Para iniciar o servidor, é necessário executar os comandos indicados na Listagem 10.

```
FROM ollama/ollama

CMD ["ollama", "serve"]
```

**Listagem 10:** *Dockerfile para o Serviço Ollama*

## Configuração do Docker Compose

O ficheiro docker-compose.yml é responsável por orquestrar os diferentes serviços da aplicação. Ao agrupar todos os componentes — como a aplicação Flask, a base de dados MongoDB e o modelo Ollama — num único ficheiro como pode ser visto na Listagem 11.

**Listagem 11:** *Ficheiro docker compose*

---

```
1 version: '3.9'
2
3 services:
4   flask-app:
5     build: .
6     ports:
7       - "5000:5000"
8     depends_on:
9       - mongo
10      - ollama
11     environment:
12       - PYTHONUNBUFFERED=1
13       - OLLAMA_HOST=http://ollama:11434
14     volumes:
15       - ./app
16     networks:
17       - backend
18
19 mongo:
20   image: mongo:6
21   restart: always
22   ports:
23     - "27017:27017"
24   networks:
25     - backend
```

---

---

```
1     volumes:
2       - mongo_data:/data/db
3 ollama:
4     build:
5       context: .
6       dockerfile: Dockerfile.ollama
7     ports:
8       - "11434:11434"
9     volumes:
10      - ./entrypoint.sh:/entrypoint.sh
11      - ollama_data:/root/.ollama
12     entrypoint: [ "sh", "/entrypoint.sh" ]
13     networks:
14       - backend
15
16 volumes:
17   mongo_data:
18   ollama_data:
19
20 networks:
21   backend:
```

---

O ficheiro `docker-compose.yml` (Listagem 11) define três serviços distintos:

- **flask-app**: é o serviço principal da aplicação. É construído a partir do `Dockerfile` principal. Está mapeado para a porta 5000, e depende dos serviços `mongo` e `ollama` para funcionar corretamente. A variável de ambiente `OLLAMA_HOST` é configurada para que a aplicação consiga comunicar com o serviço Ollama. O volume `./app` permite que as alterações feitas no código-fonte local se reflitam imediatamente dentro do container;
- **mongo**: utiliza a imagem oficial do MongoDB (versão 6). É configurado para reiniciar automaticamente em caso de falha, e expõe a porta 27017. Os dados são persistidos através do volume `mongo_data`, garantindo que a base de dados não se perde entre reinicializações;
- **ollama**: constrói a imagem com base no ficheiro `Dockerfile.ollama`. Expõe a porta 11434, que é usada para servir o modelo. O volume `ollama_data` guarda os dados do modelo Ollama e o script `entrypoint.sh` é executado como ponto de entrada para instalar os modelos LLAMA3 e DeepSeek-R1.

Além dos serviços, o ficheiro também define:

- **Volumes**: `mongo_data` e `ollama_data` asseguram persistência dos dados entre execuções dos containers;
- **Rede**: todos os serviços estão ligados à mesma rede virtual chamada `backend`, o que permite a comunicação entre eles de forma isolada.

## Técnicas para evitar Docker in Docker

Sabendo que, inicialmente, o PhoneInfoga era executado através de Docker-in-Docker — ou seja, um container Docker dentro de outro —, foi necessário alterar essa abordagem. Esta técnica, além de ser considerada uma má prática em desenvolvimento, pode causar problemas de desempenho e segurança, e nem sempre é suportada ou fácil de executar em todos os ambientes.

Como solução, optou-se por instalar o PhoneInfoga diretamente via linha de comandos, permitindo assim que a ferramenta seja executada como um serviço no sistema, fora de um *container*, garantindo maior estabilidade, segurança e compatibilidade.

Para isso, o comando foi alterado de:

```
docker run -it sundowndev/phoneinfoga scan --number  
↔ NUMERO_DE_TELEMOVEL
```

para:

```
phoneinfoga scan --number NUMERO_DE_TELEMOVEL
```

Outra alternativa considerada foi a criação de um sistema em formato **API**, como foi feito com o ollama. No entanto, essa abordagem implicaria manter um serviço sempre ativo, o que seria mais exigente para a máquina, especialmente considerando que a funcionalidade de scan é utilizada apenas pontualmente.

### 5.5.2 Execução do *Backend* num serviço *self hosted*

Neste caso, para deixarmos o serviço *self-hosted* a correr numa máquina, por exemplo, em casa, devem ser seguidos os seguintes passos:

- Configurar a máquina que será utilizada para ter um **IP** estático. No caso do Windows 11, o processo é o seguinte:
  1. Abrir o terminal, escrever `ipconfig` e guardar o **IPv4**, a *subnet* e a *Default Gateway*, como ilustrado na Figura 5.15.

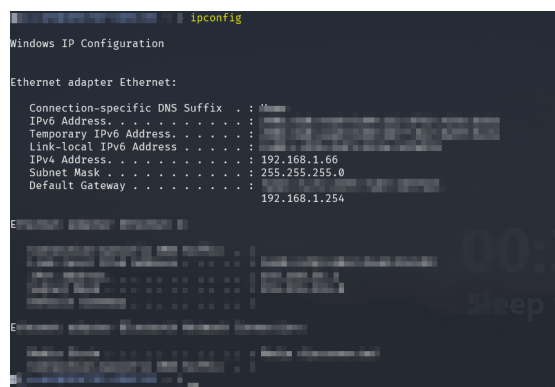


Figura 5.15: Resultado do comando `ipconfig`.

2. Clicar com o botão direito no ícone da rede, localizado na barra de tarefas (canto inferior direito), e seleccionar *Network and Internet Settings*, como mostrado na Figura 5.16.

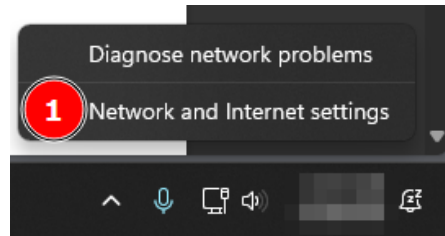


Figura 5.16: Acesso às definições de rede.

3. Clicar em *Properties*, conforme indicado na Figura 5.17.

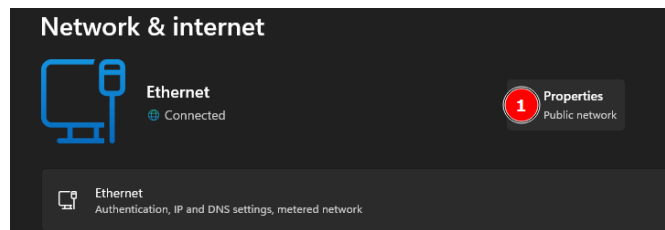


Figura 5.17: Acesso às propriedades da rede.

4. Clicar em *Edit* na secção *IP Assignment*, como demonstrado na Figura 5.18.

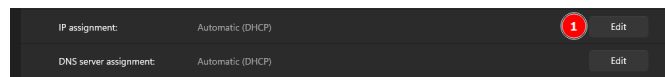


Figura 5.18: Edição da configuração de IP.

5. Seleccionar *Manual* e preencher os dados da rede com base nas informações recolhidas anteriormente (IPv4, Subnet e Gateway), como mostrado na Figura 5.19.

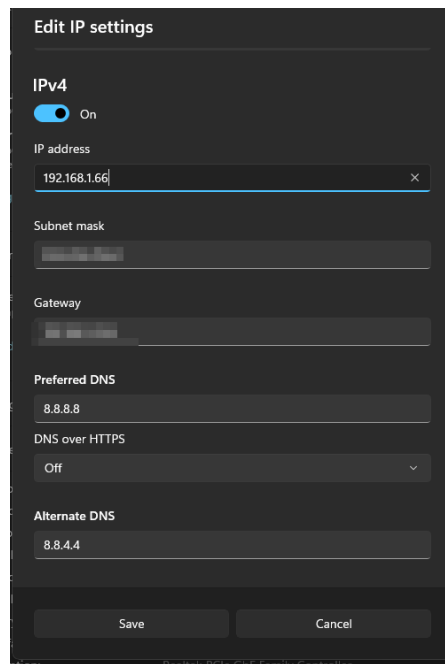


Figura 5.19: Configuração manual do IP estático.

- Após a configuração da rede, basta executar o ficheiro `docker-compose` com o seguinte comando:

```
docker compose up
```

### 5.5.3 Publicação do *Backend* na AWS e GCP

Embora o âmbito do projeto tenha sido manter tudo o mais *open source* possível e evitar quaisquer custos, para algumas pessoas poderá ser mais vantajoso correr o *Docker* através da [Amazon Web Services \(AWS\)](#) ou da [Google Cloud Platform \(GCP\)](#). É possível executar o *Docker* nestes serviços através dos seguintes passos:

- **Na AWS, usando o EC2 (Elastic Compute Cloud):**
  1. Criar uma instância EC2 com o sistema operativo Ubuntu;
  2. Configurar as portas de [Secure Shell \(SSH\)](#) e [HTTP](#) para facilitar a ligação e a configuração remota;
  3. Instalar o Docker e o Docker Compose;
  4. Copiar o projeto para dentro do servidor;
  5. Executar o `docker compose`.
- **Na GCP, usando o Google Compute Engine:**
  1. Criar uma instância de máquina virtual (VM) através do *Console GCP > Compute Engine*, escolhendo Ubuntu como sistema operativo;
  2. Configurar as portas de [SSH](#) e [HTTP](#);
  3. Instalar o Docker e o Docker Compose;

4. Copiar o projeto para dentro do servidor;
5. Executar o `docker compose`.

Ambos os métodos são semelhantes e de implementação relativamente simples, mas é necessário ter atenção aos custos associados à utilização contínua destas plataformas.

## 5.6 Testes

Nesta secção, serão abordados os testes efetuados à aplicação, nomeadamente os testes funcionais dos endpoints `/start-scan` e `/tools`, bem como os testes de segurança, que consistiram no envio de pedidos com o intuito de provocar um ataque de **DOS** ao backend.

### 5.6.1 Teste funcional do `/start-scan`

Para a execução deste teste funcional, foi desenvolvido um website com o objetivo de testar a extensão. Esta página representa um perfil de uma pessoa, contendo informações como nome, e-mail, nome de utilizador, morada e redes sociais como pode ser visto na Figura 5.20.

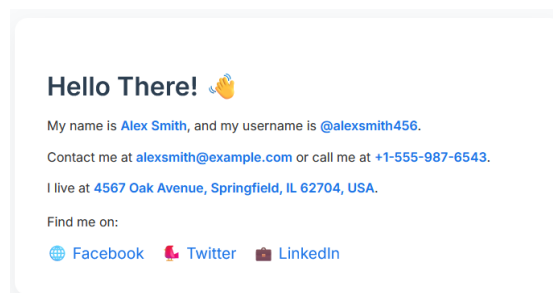


Figura 5.20: Website Criado

Os dados utilizados neste teste são fictícios. Esta abordagem permitiu verificar se o **LLM** é capaz de analisar o conteúdo **HTML** da página e extrair corretamente a informação considerada relevante.

Para realizar o teste na extensão, deve-se aceder à página alvo (neste caso, a página de teste) no navegador, abrir a extensão e, em seguida, introduzir a API Key e selecionar o modelo pretendido, conforme ilustrado na Figura 5.21. A extensão recolhe automaticamente o URL da página ativa que o utilizador está a ver.



**Figura 5.21:** Interface da extensão com a inserção da API Key e seleção do modelo.

Após o utilizador carregar no botão *Start Scan* e aguardar alguns instantes, a extensão apresenta os dados extraídos pelo **LLM** selecionado numa tabela localizada abaixo do botão *Start Scan*, conforme demonstrado na Figura 5.22.

### Information Found

Key	Value
address	4567 Oak Avenue, Springfield, IL 62704, USA
company	No data
date_of_birth	No data
email	alexsmith@example.com
full_name	Alex Smith
job_title	No data
phone	+15559876543
social_media_linkedin	https://linkedin.com/in/ alexsmith

**Figura 5.22:** Imagem da extensão com os dados encontrados pelo **LLM**.

## 5.6.2 Teste funcional do /tools

Para a execução deste teste funcional, foi criado um ficheiro **JSON** destinado a testar a utilização simultânea de todas as ferramentas. Para tal, utilizou-se o **JSON** gerado pelo endpoint `/start-scan`, excluindo as redes sociais. Este **JSON** é apresentado na Listagem 12.

---

**Listagem 12:** *JSON usado para os testes funcionais do /tools*

---

```
1 {"address":"4567 Oak Avenue, Springfield, IL 62704, USA","company":null,"date_of_birth"}
  ↪ :null,"email":"alexsmith@example.com","full_name":"Adam
  ↪ Smith","job_title":null,"other_details":{},"phone":"+1-555-987-6543","photos":[],"u
  ↪ sernames":["alexsmith456"]}
```

---

Com este ficheiro **JSON**, foi possível verificar se as ferramentas seriam utilizadas corretamente, bem como confirmar a precisão da conversão dos seus dados.

### 5.6.3 Teste de Segurança

Para além dos testes de segurança realizados pela pipeline e pelas ferramentas **DAST**, foram também efetuados testes de **DOS**. Para tal, foi utilizado o seguinte comando:

```
ping 192.168.1.66 -t | 65500
```

Este comando executa o ping continuamente, 65.500 vezes. Graças a este procedimento, foi possível verificar o correto funcionamento do limitador de taxa (rate limiter).

## 5.7 Resultados

Os resultados obtidos após a conclusão do projeto foram os seguintes:

- Foi possível verificar que é viável realizar a análise de websites básicos ou intermédios, como cartões de visita digitais e portfólios;
- A extensão possibilita uma maior facilidade na realização de **OSINT**, facilitando a obtenção de informações, por exemplo, sobre uma pessoa;
- Os processos de segurança — neste caso, o uso dos *GitHub Workflows*, das ferramentas **DAST** e do sistema de *API Keys* — permitiram garantir a integridade da aplicação e prevenir vulnerabilidades graves que poderiam prejudicar a experiência do utilizador;
- Com a adaptação do *backend* para um ficheiro `docker compose`, foi possível verificar uma maior flexibilidade na sua execução em diferentes ambientes, como GCP, AWS e sistemas *selfhosted*;
- Foi possível demonstrar que o uso de **LLM** pode ser bastante útil na investigação e análise de informação;
- Verificou-se a importância da criação de manifestos bem estruturados para minimizar o número de vulnerabilidades e vetores de ataque;
- Apesar de semelhantes, existem diferenças relevantes entre o desenvolvimento de extensões para Chrome e Firefox, as quais devem ser consideradas;
- Toda a extensão foi desenvolvida sem recurso a qualquer ferramenta ou serviço *closed source*; mesmo na etapa de segurança, o uso foi restrito exclusivamente a ferramentas *open source*.

## 5.8 Trabalho Futuro

Os seguintes desenvolvimentos poderão ser implementados como trabalho futuro:

### 5.8.1 Sistema de Seleção Automática de APIs

Proposta de criação de um mecanismo que permita aos utilizadores selecionar automaticamente uma **API** (local ou remota) através da simples inserção do seu **URL**. Esta funcionalidade proporcionará:

- Simplificação significativa dos testes da extensão;
- Facilitação do processo de publicação;
- Melhoria na experiência do utilizador.

### 5.8.2 Migração para Manifest V3 no Firefox

A atualização para a versão mais recente do manifesto de extensões apresenta:

- Adaptação tecnicamente acessível;
- Necessidade de atenção especial às limitações impostas pelo V3;
- Importância crítica para garantir compatibilidade futura.

### 5.8.3 Integração com Modelos de IA Mais Avançados

A adoção de sistemas de inteligência artificial mais recentes oferece:

- Melhoria significativa na qualidade dos resultados;
- Maior capacidade de processamento de consultas complexas;
- Expansão do leque de funcionalidades oferecidas.

## 5.9 Ideias para Projetos

Durante o desenvolvimento deste projeto, surgiram as seguintes ideias para para novos projetos relacionados com o projeto desenvolvido:

### 5.9.1 Conversor Automático de Output CLI para JSON

- **Conceito:** Criação de um método que consiga converter *outputs* de ferramentas **CLI** para formato **JSON**;
- **Implementação:**
  - Uso de modelos de **LLM** modernos para interpretação de textos;
  - Integração com varias ferramentas sem precisar de alterar o código.
- **Benefícios:**
  - Eliminação do processo manual de conversão;
  - Padronização dos *outputs* para integração com outras ferramentas.

### 5.9.2 Extensão OSINT para Análise de Websites

- **Conceito:** Criação de uma extensão **OSINT** com foco em análise de websites em vez de pessoas.
- **Implementação:**
  - Automação sem dependência de IA;
  - Integração com ferramentas existentes de análise web.
- **Benefícios:**
  - Processo mais simples que soluções de **OSINT** para pessoas conseguirem analisar os servidores;
  - Potencial para detecção automática de vulnerabilidades.

### 5.9.3 Sistema de Monitorização Contínua de Vulnerabilidades a partir do Github Actions

- **Conceito:** Ferramenta autónoma para detecção proativa de vulnerabilidades a partir de um runner do Github Actions;
- **Implementação:**
  - Integração contínua com GitHub Actions;
  - Mecanismo de trigger via **API** para scans automáticos.
- **Desafios:**
  - Manutenção de ações de longa duração;
  - Gestão eficiente de recursos computacionais.
- **Benefícios:**
  - Resposta imediata a novas ameaças;
  - Redução da janela de exposição a vulnerabilidades.

# 6

## Conclusão

O presente trabalho evidenciou a importância crescente do OSINT e dos LLM no contexto da cibersegurança, demonstrando como a combinação destas áreas pode potencializar a recolha, análise e interpretação de grandes volumes de dados públicos. A exploração das principais ferramentas de OSINT e a análise das suas aplicações práticas permitiram perceber o impacto que a informação aberta pode ter, tanto em investigações legítimas como em potenciais riscos éticos e legais.

A integração de modelos de linguagem de grande escala, como o LLaMA 3 e o DeepSeek R1, mostrou-se fundamental para automatizar tarefas complexas, desde a extração de dados até à análise de ameaças. No entanto, ficou patente que, apesar das suas potencialidades, os LLM apresentam limitações, como a possibilidade de alucinações, a dependência de dados de treino e os riscos associados à sua utilização em ambientes sensíveis. A discussão sobre as vulnerabilidades identificadas pela OWASP e as preocupações éticas reforçam a necessidade de uma abordagem crítica e responsável na adoção destas tecnologias.

O desenvolvimento do projeto prático permitiu consolidar estes conhecimentos, resultando numa solução robusta, flexível e baseada em ferramentas open source. A uniformização dos outputs, a preocupação com a segurança (através de *pipelines*, SAST, DAST e manifestos rigorosos) e a portabilidade garantida pelo uso de Docker e cloud. A experiência adquirida com a integração de diferentes navegadores e a adaptação a múltiplos contextos reforçou a importância da compatibilidade e da atenção às especificidades de cada plataforma.

Em suma, este trabalho não só atingiu os objetivos propostos, como também abriu portas para futuras evoluções, como a integração de APIs dinâmicas, a migração para manifestos mais recentes e a adoção de modelos de IA ainda mais avançados. As ideias para trabalhos futuros e as reflexões éticas apresentadas evidenciam o potencial de crescimento e inovação nesta área, sublinhando a relevância de soluções open source para a democratização do acesso à informação e para a promoção de uma cibersegurança mais transparente e acessível.



# Bibliography

- [1] *About the Internet Archive*. URL: <https://archive.org/about/> (acedido em 2024-11-18).
- [2] *About|Epieos*. URL: <https://epieos.com/aboutus> (acedido em 2024-11-18).
- [3] *After Action Report for the Response to the 2013 Boston Marathon Bombings*. URL: [https://www.policinginstitute.org/wp-content/uploads/2015/05/after-action-report-for-the-response-to-the-2013-boston-marathon-bombings\\_0.pdf](https://www.policinginstitute.org/wp-content/uploads/2015/05/after-action-report-for-the-response-to-the-2013-boston-marathon-bombings_0.pdf).
- [4] *Bandit — Security Linter for Python*. URL: <https://bandit.readthedocs.io> (acedido em 2025-03-11).
- [5] *Beautiful Soup — Python Library for Pulling Data Out of HTML and XML Files*. URL: <https://www.crummy.com/software/BeautifulSoup/> (acedido em 2025-04-17).
- [6] Ludo Block. «The long history of OSINT». Em: *Journal of Intelligence History* 23.2 (2024), pp. 95–109. DOI: 10.1080/16161262.2023.2224091. eprint: <https://doi.org/10.1080/16161262.2023.2224091>. URL: <https://doi.org/10.1080/16161262.2023.2224091>.
- [7] Jesper Boeg. «Kanban em 10 passos». Em: *Tradução de Leonardo Campos, Marcelo Costa, Lúcio Camilo, Rafael Buzon, Paulo Rebelo, Eric Fer, Ivo La Puma, Leonardo Galvão, Thiago Vespa, Manoel Pimentel e Daniel Wildt*. C4Media (2010), p. 27.
- [8] *ChatGPT*. URL: <https://openai.com/chatgpt/overview/> (acedido em 2025-06-18).
- [9] *Ciberataques por agentes de IA estão a caminho*. URL: <https://www.mittechreview.pt/2025/inteligencia-artificial/ciberataques-por-agentes-de-ia-estao-a-caminho/> (acedido em 2025-06-18).
- [10] *Cibersegurança e inteligência artificial: o que nos espera em 2025?* URL: <https://www.welivesecurity.com/pt/seguranca-digital/ciberseguranca-e-inteligencia-artificial-o-que-nos-espera-em-2025/>.

- 
- [11] *Código Penal - CP - Artigo 217.<sup>o</sup>*. URL: <https://diariodarepublica.pt/dr/legislacao-consolidada/decreto-lei/1995-34437675-66422645>.
- [12] *Código Penal - CP - Artigo 223.<sup>o</sup>*. URL: <https://diariodarepublica.pt/dr/legislacao-consolidada/decreto-lei/1995-34437675-108049901>.
- [13] *DeepSeek*. URL: <https://www.deepseek.com/en> (acedido em 2025-06-18).
- [14] *Docker — Empowering App Development for Developers*. URL: <https://www.docker.com> (acedido em 2025-05-23).
- [15] *DuckDuckGo — Privacy, simplified*. URL: <https://duckduckgo.com> (acedido em 2025-04-22).
- [16] *Figma – Collaborative interface design tool*. URL: <https://www.figma.com> (acedido em 2025-01-10).
- [17] *Flask — Python Microframework*. URL: <https://flask.palletsprojects.com/> (acedido em 2025-01-22).
- [18] *Flask-Limiter — Rate Limiting Extension for Flask*. URL: <https://flask-limiter.readthedocs.io/> (acedido em 2025-01-22).
- [19] Python Software Foundation. *Python Programming Language*. <https://www.python.org>. (Acedido em 2024-10-09).
- [20] *GitHub Actions — Automate, customize, and execute your software development workflows*. URL: <https://docs.github.com/en/actions> (acedido em 2025-01-10).
- [21] *Have I Been Pwned*. URL: <https://cloud.google.com/chronicle/docs/soar/marketplace-integrations/haveibeenpwned> (acedido em 2024-11-23).
- [22] *Holehe — Check Email Address Account Breaches*. URL: <https://github.com/megadose/holehe> (acedido em 2025-03-09).
- [23] *HTML, CSS e JavaScript — Tecnologias Fundamentais para Desenvolvimento Web*. URL: <https://developer.mozilla.org/> (acedido em 2025-01-14).
- [24] Yong-Woon Hwang et al. «Current Status and Security Trend of OSINT». Em: *Wireless Communications and Mobile Computing 2022.1* (2022), p. 1290129. DOI: <https://doi.org/10.1155/2022/1290129>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1155/2022/1290129>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/1290129>.
- [25] *Intelligence and Deception: The Battle of Lake Trasimene*. URL: <https://strategyinbrief.us/2025/04/26/intelligence-and-deception-the-battle-of-lake-trasimene/>.
- [26] *IRCopilot: Automated Incident Response with Large Language Models*. URL: <https://arxiv.org/pdf/2505.20945>.

- [27] Laura Marie Johnson. *Establishing Broadcast Monitoring as Open Source Intelligence: The BBC Monitoring Service during the Second World War* Johnson. 2013.
- [28] Douglas Kunda e Hazael Phiri. «A comparative study of nosql and relational database». Em: *Zambia ICT Journal* 1.1 (2017), pp. 1–4.
- [29] *Large Language Models: A Comprehensive Survey of its Applications, Challenges, Limitations, and Future Prospects*. URL: [https://www.researchgate.net/profile/Muhammad-Shaikh-9/publication/383818024\\_Large\\_Language\\_Models\\_A\\_Comprehensive\\_Survey\\_of\\_its\\_Applications\\_Challenges\\_Limitations\\_and\\_Future\\_Prospects/links/66dfffb06b1606e24c21d8936/Large-Language-Models-A-Comprehensive-Survey-of-its-Applications-Challenges-Limitations-and-Future-Prospects.pdf](https://www.researchgate.net/profile/Muhammad-Shaikh-9/publication/383818024_Large_Language_Models_A_Comprehensive_Survey_of_its_Applications_Challenges_Limitations_and_Future_Prospects/links/66dfffb06b1606e24c21d8936/Large-Language-Models-A-Comprehensive-Survey-of-its-Applications-Challenges-Limitations-and-Future-Prospects.pdf).
- [30] *Lei do Cibercrime*. URL: <https://diariodarepublica.pt/dr/legislacao-consolidada/lei/2009-128879174>.
- [31] *Llama 4: Leading intelligence.Unrivaed speed and efficiency*. URL: <https://www.llama.com/> (acedido em 2025-06-18).
- [32] *MongoDB — The Developer Data Platform*. URL: <https://www.mongodb.com> (acedido em 2025-03-10).
- [33] *MyPy — Optional Static Typing for Python*. URL: <https://mypy-lang.org> (acedido em 2025-03-11).
- [34] Håvard Myrbakken e Ricardo Colomo-Palacios. «DevSecOps: a multivocal literature review». Em: *International Conference on Software Process Improvement and Capability Determination*. Springer. 2017, pp. 17–29.
- [35] *Nikto — Web Server Scanner*. URL: <https://cirt.net/Nikto2> (acedido em 2025-04-04).
- [36] *Ollama — Manage and Run Large Language Models Locally*. URL: <https://ollama.com> (acedido em 2024-11-09).
- [37] *OWASP Top 10 for Large Language Model Applications*. URL: <https://owasp.org/www-project-top-10-for-large-language-model-applications/> (acedido em 2025-06-18).
- [38] *OWASP ZAP — The Open Web Application Security Project Zed Attack Proxy*. URL: <https://www.zaproxy.org> (acedido em 2025-04-04).
- [39] *PhoneInfoga — Information Gathering Tool for Phone Numbers*. URL: <https://github.com/sundowndev/PhoneInfoga> (acedido em 2024-10-23).
- [40] *pip: The Python Package Installer*. URL: <https://pip.pypa.io/en/stable/> (acedido em 2024-10-09).

- [41] *PyMongo — Python Driver for MongoDB*. URL: <https://pymongo.readthedocs.io> (acedido em 2025-03-10).
- [42] Partha Pratim Ray. *A Review on Vibe Coding: Fundamentals, State-of-the-art, Challenges and Future Directions*. Mai. de 2025. DOI: [10.36227/techrxiv.174681482.27435614/v1](https://doi.org/10.36227/techrxiv.174681482.27435614/v1). URL: <http://dx.doi.org/10.36227/techrxiv.174681482.27435614/v1>.
- [43] *Safety — Python Dependency Vulnerability Scanner*. URL: <https://pyup.io/safety/> (acedido em 2025-03-11).
- [44] *Semgrep — Fast, Powerful Static Analysis for Code*. URL: <https://semgrep.dev> (acedido em 2025-03-11).
- [45] *Sherlock — Find Usernames Across Social Networks*. URL: <https://github.com/sherlock-project/sherlock> (acedido em 2024-10-30).
- [46] *Shodan - Security Copilot - Learn Microsoft*. URL: <https://learn.microsoft.com/en-us/copilot/security/plugin-shodan> (acedido em 2024-12-02).
- [47] *SpiderFoot - SpiderFoot automates OSINT for threat intelligence and mapping your attack surface*. URL: <https://github.com/smicallef/spiderfoot> (acedido em 2024-11-23).
- [48] *Sputnik*. URL: <https://chromewebstore.google.com/detail/sputnik/manapjdamopgbpimgojkccihl=pt> (acedido em 2025-04-11).
- [49] Robert D. Steele. «Intelligence in the 1990's: Recasting National Security in a Changing World». Em: *American Intelligence Journal* (1990).
- [50] Yesim Surmelioglu e Suleyman Sadi Seferoglu. «An Examination of Digital Footprint Awareness and Digital Experiences of Higher Education Students.» Em: *World Journal on Educational Technology: Current Issues* 11.1 (2019), pp. 48–64.
- [51] *The history of AI*. URL: <https://www.ibm.com/think/topics/history-of-artificial-intelligence>.
- [52] *theHarvester - E-mails, subdomains and names Harvester - OSINT*. URL: <https://github.com/laramies/theHarvester> (acedido em 2024-12-02).
- [53] *Vortimo OSINT-tool*. URL: <https://chromewebstore.google.com/detail/vortimo-osint-tool/mnakbpdnkedaegeiaoakkjafhoidklnf> (acedido em 2025-04-18).
- [54] *What is Google Hacking/Dorking*. URL: <https://www.imperva.com/learn/application-security/google-dorking-hacking/>.
- [55] *Why I Created Fabric*. URL: <https://danielmiessler.com/blog/fabric-origin-story> (acedido em 2025-06-18).

# Anexos





## Código gerado - ChatGPT vs DeepSeek R1 vs Ollama

Listagem 13: Código gerado pelo ChatGPT

```
1 import curses
2 import time
3 import random
4
5 # Mapa do jogo: # = parede, . = ponto
6 MAP = [
7     "#####",
8     "#.....#.....#",
9     "#.####.#.#####..#",
10    "#.# #.#.#.# #..#",
11    "#.####.###.#.###.#..#",
12    "#.....#...#.#.....#",
13    "###.#####.###.#####.",
14    "#.....#.....#",
15    "#####"
16 ]
17 # Direções possíveis
18 DIRECTIONS = {
19     curses.KEY_UP: (-1, 0),
20     curses.KEY_DOWN: (1, 0),
21     curses.KEY_LEFT: (0, -1),
22     curses.KEY_RIGHT: (0, 1),
23 }
24
25 def main(stdscr):
26     curses.curs_set(0)
27     stdscr.nodelay(True)
28     stdscr.timeout(100)
```

---

```
1     # Inicializar mapa
2     map_data = [list(row) for row in MAP]
3     max_y, max_x = len(map_data), len(map_data[0])
4
5     # Posição inicial do Pac-Man
6     pac_y, pac_x = 1, 1
7     score = 0
8     total_dots = sum(row.count('.') for row in map_data)
9
10    direction = (0, 0)
11
12    while True:
13        stdscr.clear()
14
15        # Mostrar mapa
16        for y, row in enumerate(map_data):
17            stdscr.addstr(y, 0, ''.join(row))
18
19        # Mostrar Pac-Man
20        stdscr.addch(pac_y, pac_x, 'C')
21
22        # Mostrar pontuação
23        stdscr.addstr(max_y + 1, 0, f"Score: {score}")
24
25        key = stdscr.getch()
26        if key in DIRECTIONS:
27            direction = DIRECTIONS[key]
28        elif key == ord('q'):
29            break
30
31        new_y = pac_y + direction[0]
32        new_x = pac_x + direction[1]
33
34        if map_data[new_y][new_x] != '#':
35            pac_y, pac_x = new_y, new_x
36            if map_data[pac_y][pac_x] == '.':
37                score += 1
38                map_data[pac_y][pac_x] = ' '
39
40        if score == total_dots:
41            stdscr.addstr(max_y + 2, 0, "Parabéns! Apanhaste todos os pontos!")
42            stdscr.refresh()
43            time.sleep(3)
44            break
45
46        stdscr.refresh()
47        time.sleep(0.1)
48
49 if __name__ == '__main__':
50     curses.wrapper(main)
```

---

## Listagem 14: Código gerado pelo DeepSeek R1

---

```
1 import os
2 import sys
3 import time
4 import random
5 try:
6     import msvcrt # Para Windows
7 except ImportError:
8     import sys
9     import tty
10    import termios # Para Linux/Mac
11
12 class PacmanGame:
13     def __init__(self, width=20, height=10):
14         self.width = width
15         self.height = height
16         self.pacman_pos = [height // 2, width // 2]
17         self.ghosts = []
18         self.dots = []
19         self.score = 0
20         self.game_over = False
21         self.direction = 'right'
22         self.next_direction = 'right'
23         self.initialize_board()
24
25     def initialize_board(self):
26         # Cria os pontos (comida)
27         for y in range(1, self.height-1):
28             for x in range(1, self.width-1):
29                 if (y, x) != tuple(self.pacman_pos):
30                     self.dots.append((y, x))
31
32         # Adiciona fantasmas
33         self.ghosts = [
34             [1, 1, 'down'],
35             [1, self.width-2, 'down'],
36             [self.height-2, 1, 'up'],
37             [self.height-2, self.width-2, 'up']
38         ]
39
40     def draw(self):
41         os.system('clear' if os.name == 'posix' else 'cls')
42         print(f"Score: {self.score}\n")
43
44         for y in range(self.height):
45             for x in range(self.width):
46                 if y == 0 or y == self.height - 1 or x == 0 or x == self.width - 1:
47                     print('#', end=' ')
48                 elif [y, x] == self.pacman_pos:
49                     print('C', end=' ')
50
```

---

---

```
1         elif any(ghost[:2] == [y, x] for ghost in self.ghosts):
2             print('G', end=' ')
3         elif (y, x) in self.dots:
4             print('.', end=' ')
5         else:
6             print(' ', end=' ')
7     print()
8
9     print("\nControles: WASD para mover, Q para sair")
10
11 def move_pacman(self):
12     # Atualiza a direção se possível
13     new_pos = self.pacman_pos.copy()
14
15     if self.next_direction == 'up' and self.direction != 'down':
16         self.direction = self.next_direction
17     elif self.next_direction == 'down' and self.direction != 'up':
18         self.direction = self.next_direction
19     elif self.next_direction == 'left' and self.direction != 'right':
20         self.direction = self.next_direction
21     elif self.next_direction == 'right' and self.direction != 'left':
22         self.direction = self.next_direction
23
24     # Move na direção atual
25     if self.direction == 'up':
26         new_pos[0] -= 1
27     elif self.direction == 'down':
28         new_pos[0] += 1
29     elif self.direction == 'left':
30         new_pos[1] -= 1
31     elif self.direction == 'right':
32         new_pos[1] += 1
33
34     # Verifica colisão com paredes
35     if (new_pos[0] <= 0 or new_pos[0] >= self.height - 1 or
36         new_pos[1] <= 0 or new_pos[1] >= self.width - 1):
37         return
38
39     # Atualiza posição
40     self.pacman_pos = new_pos
41
42     # Verifica se comeu um ponto
43     if tuple(self.pacman_pos) in self.dots:
44         self.dots.remove(tuple(self.pacman_pos))
45         self.score += 10
46
47     # Verifica vitória
48     if not self.dots:
49         self.game_over = True
50         print("\nVocê venceu! Todos os pontos foram coletados!")
51
```

---

---

```
1 def move_ghosts(self):
2     for ghost in self.ghosts:
3         y, x, direction = ghost
4         new_y, new_x = y, x
5
6         # Lógica simples de movimento para fantasmas
7         if direction == 'up':
8             new_y -= 1
9             if new_y <= 0:
10                direction = 'down'
11        elif direction == 'down':
12            new_y += 1
13            if new_y >= self.height - 1:
14                direction = 'up'
15        elif direction == 'left':
16            new_x -= 1
17            if new_x <= 0:
18                direction = 'right'
19        elif direction == 'right':
20            new_x += 1
21            if new_x >= self.width - 1:
22                direction = 'left'
23
24        # Atualiza posição e direção do fantasma
25        ghost[0], ghost[1], ghost[2] = new_y, new_x, direction
26
27        # Verifica colisão com Pac-Man
28        if [new_y, new_x] == self.pacman_pos:
29            self.game_over = True
30            print("\nGame Over! Um fantasma te pegou!")
31
32    def update(self):
33        self.move_pacman()
34        self.move_ghosts()
35
36    def get_input():
37        if os.name == 'nt': # Windows
38            if msvcrt.kbhit():
39                return msvcrt.getch().decode().lower()
40        else: # Linux/Mac
41            fd = sys.stdin.fileno()
42            old_settings = termios.tcgetattr(fd)
43            try:
44                tty.setraw(sys.stdin.fileno())
45                ch = sys.stdin.read(1).lower()
46            finally:
47                termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
48            return ch
49        return None
50
```

---

---

```
1 def main():
2     game = PacmanGame(30, 15)
3
4     while not game.game_over:
5         game.draw()
6
7         key = get_input()
8
9         if key == 'q':
10            print("\nJogo encerrado pelo jogador.")
11            break
12        elif key == 'w':
13            game.next_direction = 'up'
14        elif key == 's':
15            game.next_direction = 'down'
16        elif key == 'a':
17            game.next_direction = 'left'
18        elif key == 'd':
19            game.next_direction = 'right'
20
21        game.update()
22        time.sleep(0.2)
23
24        print(f"\nPontuação final: {game.score}")
25
26 if __name__ == "__main__":
27     main()
```

---

### Listagem 15: Código gerado pelo LLAMA3

---

```
1 import os
2 import time
3 import random
4
5 # Constantes do jogo
6 MAPA = [
7     "-----",
8     "|       |",
9     "|  P   |",
10    "|       |",
11    "|  _ _ |",
12    "|  /  \ |",
13    "|  /   \ |",
14    "|-----|
15 ]
16
17
```

---

---

```
1 PACMAN_POSITION = (1, 2) # Linha e coluna inicial do Pac-Man
2
3 ENEMY_POSITIONS = [(3, 4), (5, 5)] # Posições iniciais dos inimigos
4
5 SCORE = 0
6
7 # Função para imprimir o mapa do jogo
8 def print_mapa():
9     os.system("clear")
10    for linha in MAPA:
11        print(linha)
12    print(f"Score: {SCORE}")
13
14 # Função para movimentar o Pac-Man
15 def move_pacacman(x, y):
16    global PACMAN_POSITION
17    PACMAN_POSITION = (x, y)
18
19 # Função para verificar se o Pac-Man comeu uma ficha de point
20 def check_point():
21    if MAPA[PACMAN_POSITION[0]][PACMAN_POSITION[1]] == "_":
22        SCORE += 100
23        MAPA[PACMAN_POSITION[0]] = MAPA[PACMAN_POSITION[0]][:PACMAN_POSITION[1]] + "P" +
24    MAPA[PACMAN_POSITION[0]][PACMAN_POSITION[1]+1:]
25    return SCORE
26
27 # Função para verificar se o Pac-Man foi comido por um inimigo
28 def check_enemy():
29    for enemy_position in ENEMY_POSITIONS:
30        if PACMAN_POSITION == enemy_position:
31            print("Game Over!")
32            exit()
33    return False
34
35 # Loop do jogo
36 while True:
37    print_mapa()
38    time.sleep(0.5) # Tempo de atualização da tela
39    x = int(input("Enter direction (1-4): \n1 - Up\n2 - Down\n3 - Left\n4 - Right: "))
40    if x == 1 and PACMAN_POSITION[0] > 1:
41        move_pacacman(PACMAN_POSITION[0]-1, PACMAN_POSITION[1])
42    elif x == 2 and PACMAN_POSITION[0] < len(MAPA) - 1:
43        move_pacacman(PACMAN_POSITION[0]+1, PACMAN_POSITION[1])
44    elif x == 3 and PACMAN_POSITION[1] > 1:
45        move_pacacman(PACMAN_POSITION[0], PACMAN_POSITION[1]-1)
46    elif x == 4 and PACMAN_POSITION[1] < len(MAPA[0]) - 1:
47        move_pacacman(PACMAN_POSITION[0], PACMAN_POSITION[1]+1)
48    SCORE = check_point()
49    if check_enemy():
50        break
51
```

---



# B

## Código gerado pelo LLAMA3 e Pipeline configurada

**Listagem 16:** Pipeline de segurança configurada no GitHub Actions.

---

```
1 name: Python Security Pipeline
2 on:
3   push:
4     branches: [main]
5   pull_request:
6     branches: [main]
7 jobs:
8   security-analysis:
9     name: Run Security Checks
10    runs-on: ubuntu-latest
11    steps:
12      - name: Checkout code
13        uses: actions/checkout@v3
14      - name: Set up Python
15        uses: actions/setup-python@v5
16        with:
17          python-version: '3.11'
18      - name: Install dependencies
19        run: |
20          python -m pip install --upgrade pip
21          pip install -r requirements.txt
22          pip install bandit safety semgrep mypy
23          pip install ollama flask flask-cors flask-limiter pymongo
24          ↪ beautifulsoup4
25          pip install types-flask types-flask-cors types-beautifulsoup4
```

---

---

```
1   - name: Bandit - Code security audit
2     run: bandit -r . -lll -ii
3   - name: Safety - Dependency vulnerability check
4     run: safety check --full-report --file=requirements.txt
5   - name: Mypy - Type checking
6     run: |
7         echo "[mypy]
8 ignore_missing_imports = True
9 disallow_untyped_defs = True
10 show_error_codes = True
11 no_strict_optional = True
12     " > mypy.ini
13     mypy --config-file mypy.ini . || true
14   - name: Semgrep - Static code analysis
15     run: semgrep scan --config auto
```

---