



**SOICT**

---

# **TECHNICAL REPORT**

## **ALL-ROUNDER OSINT TOOL: Social Network Reconnaissance Program Development**

**Course:** IT3910E – Project I

**Class code:** 750639

**Instructor:** Hoang Viet Dung

**Group Members:**

Nguyen Duy Phuc	20235616	Phuc.ND235616@sis.hust.edu.vn
Nguyen Hung Quang	20235618	Quang.NH235618@sis.hust.edu.vn

## **Abstract**

Open Source Intelligence (OSINT) has become a crucial component in cybersecurity, enabling the collection and analysis of publicly available information for reconnaissance and threat assessment. This technical report presents the design and development of an all-in-one OSINT tool focused on Facebook users reconnaissance. The tool integrates multiple open-source libraries and APIs to automate the collection, correlation, and visualization of user-related data.

Our solution emphasizes usability, modularity, and extensibility, making it suitable for both academic research and practical security assessments. Key functionalities include username enumeration, profile linkage, metadata extraction, and graphical representation of connections. Through testing and evaluation, the tool has demonstrated its effectiveness in gathering social network intelligence efficiently while minimizing manual effort. Future improvements may involve integrating AI-based analysis and expanding platform coverage.

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	OSINT in Cybersecurity: Context and Motivation . . . . .	2
1.2	Existing Tools and Their Limitations . . . . .	2
1.3	Objective and Contribution . . . . .	3
<b>2</b>	<b>System Analysis and Design</b>	<b>4</b>
2.1	System Overview . . . . .	4
2.2	Technology Architecture . . . . .	4
2.3	Data Processing and Normalization . . . . .	5
2.4	Visualization Techniques . . . . .	5
2.5	Ethical and Legal Considerations . . . . .	5
<b>3</b>	<b>Features Implementation</b>	<b>6</b>
3.1	User Friends List Crawling . . . . .	6
3.2	User Profile Data Crawling . . . . .	7
3.3	Check-in Information Collection . . . . .	7
3.4	Graph Generation and Visualization . . . . .	7
<b>4</b>	<b>Demonstration</b>	<b>9</b>
4.1	Environmental Setup . . . . .	9
4.2	Login and Initial Execution . . . . .	9
4.3	Friend List Collection . . . . .	10
4.4	User Profile Scraping . . . . .	11
4.5	Graph Visualization . . . . .	12
4.5.1	Connection Graph . . . . .	12
4.5.2	Location Graph . . . . .	14
4.6	Check-in Data Collection and Visualization . . . . .	16
<b>5</b>	<b>Results Evaluation</b>	<b>18</b>
5.1	Evaluation Metrics . . . . .	18
5.2	Experimental Outcomes . . . . .	18
5.3	Performance Summary . . . . .	19
5.4	Error Handling and Logging . . . . .	19
5.5	Code Quality Analysis . . . . .	19
<b>6</b>	<b>Challenges and Future Work</b>	<b>21</b>
6.1	Challenges Encountered . . . . .	21
6.2	Future Work . . . . .	21

# Chapter 1

## Introduction

### 1.1 OSINT in Cybersecurity: Context and Motivation

Open Source Intelligence (OSINT) refers to the process of collecting, analyzing, and utilizing publicly available information for investigative and analytical purposes. In recent years, OSINT has become a key component in various operational domains, especially in cybersecurity, where it is used for threat intelligence, social engineering analysis, digital footprint tracing, and red teaming exercises.

Among the many sources of OSINT, social networks are particularly valuable due to their rich and high-volume user-generated content. Platforms such as Facebook, Instagram, LinkedIn, and X (formerly Twitter) provide access to personal data, behavioral patterns, relationship graphs, and multimedia that can reveal deep insights into individuals or groups. This makes social media an indispensable resource for OSINT practitioners, especially when conducting reconnaissance in cybersecurity operations.

Despite its value, collecting and processing OSINT data from social networks presents several real-world challenges:

- **Access Limitations:** Many platforms restrict access to their APIs or actively block automated scraping to preserve user privacy.
- **Volatile Content:** Social media content is highly dynamic and frequently removed, requiring timely and efficient extraction methods.
- **Complex Structures:** The semi-structured and ever-changing nature of social media data calls for adaptable and robust parsing mechanisms.

Manual data collection methods are time-consuming, error-prone, and often inefficient. Moreover, many existing OSINT tools are either overly complex, lack adequate support for social media data, or rely exclusively on APIs, which are increasingly limited by platform policies. These factors create a practical gap for analysts and practitioners who need effective, flexible tools to work with social network data.

### 1.2 Existing Tools and Their Limitations

Several OSINT frameworks and tools have been developed to facilitate open data collection. Notable examples include:

- **Maltego:** A powerful tool for visual link analysis with support for many data sources through customizable transforms. However, advanced features require licensing and setup can be complex.
- **SpiderFoot:** An automated OSINT platform capable of scanning across various sources. While it covers a wide range of data types, its support for social media platforms is limited.
- **Recon-ng:** A command-line based reconnaissance framework with modular API integration. It offers great flexibility but can be inaccessible to users without technical expertise.
- **Social-Analyzer:** A newer tool designed to detect user profiles across platforms. While simple to use, it lacks deeper data analysis and visualization capabilities.

While these tools serve specific use cases well, they often suffer from one or more of the following drawbacks:

- High technical complexity or configuration overhead.
- Limited coverage of modern social networks, particularly Facebook.
- Heavy reliance on public APIs, which may no longer provide sufficient access.

## 1.3 Objective and Contribution

This project addresses the current limitations in social network-focused OSINT by proposing the development of a lightweight, extensible, and user-friendly tool tailored for extracting and analyzing public data—initially targeting Facebook.

Key goals of this application include:

- Automating the data collection process from social media platforms through a combination of browser-based interaction and available APIs.
- Reducing the technical barrier for cybersecurity analysts, researchers, and other OSINT users who require actionable intelligence from online platforms.
- Offering intuitive visualization features to support data interpretation and relationship mapping.

The tool is designed with practical usability in mind, making it suitable for real-world OSINT workflows where time efficiency, flexibility, and usability are critical. This chapter lays the foundation for the design, implementation, and evaluation of the system discussed in the following sections.

# Chapter 2

## System Analysis and Design

### 2.1 System Overview

The proposed OSINT tool is designed as a modular, extensible, and user-friendly application for performing reconnaissance on social network platforms, with a primary focus on Facebook. The system integrates both browser-based automation and API-based data collection (where applicable) to maximize data extraction capabilities.

The overall architecture consists of the following components:

- **Command-Line Interface (CLI):** A terminal-based interface that allows users to interact with the tool via text commands. Users can input search targets, specify configuration options (e.g., crawling depth, output format) directly from the terminal.
- **Crawling Engine:** A browser automation module (based on Selenium in Python and Puppeteer in Node.js) that navigates and extracts publicly available data from Facebook profiles, pages, and posts.
- **Data Parser and Cleaner:** Responsible for structuring, filtering, and normalizing the raw data collected from web pages or APIs.
- **Visualization Module:** Uses graph libraries (e.g. SimpleMaps, vis.js) or custom plots to represent relationships, interactions, and metadata in a human-readable form.
- **Storage and Exporter:** Allows saving results in formats such as JSON for further use.

### 2.2 Technology Architecture

Due to limited access to Facebook's official Graph API (which is highly restrictive), the tool primarily uses browser-based crawling to access publicly available information. The crawling process includes:

- Automated login: using dummy accounts to access public content within allowed policy boundaries.
- URL pattern recognition to access user timelines, photos, about sections and friends.
- DOM-based data extraction using XPath or CSS selectors, ensuring adaptability to layout changes.

- Capturing metadata such as timestamps, geolocations.

The tool employs mechanisms to avoid detection or rate-limiting, such as randomized wait times, user-agent rotation, and captcha detection fallback.

## 2.3 Data Processing and Normalization

Once raw data is collected, it is passed to the processing layer which includes:

- **Normalization:** Converting extracted and structured content into a unified schema.
- **Entity Extraction:** Identifying key entities (names, places, links) using simple NLP techniques or regular expressions.
- **Relationship Mapping:** Constructing a graph of connections (e.g., user-to-user, user's location) for visualization.
- **Duplicate Removal:** Ensuring data integrity and reducing noise by filtering redundant or low-value records.

## 2.4 Visualization Techniques

Visual representation is crucial for understanding connections and behaviors. The tool uses:

- **Node-link graphs** to show friend/follow networks or interaction clusters.
- **Timeline views** to display user check-ins over time.
- **Map-based graphs** to show user geographic locations.

These representations help users quickly identify patterns, anomalies, and points of interest in the target's online behavior.

## 2.5 Ethical and Legal Considerations

While the tool targets publicly accessible information, it is developed with respect for ethical guidelines and legal boundaries. It does not bypass platform security or privacy protections and is intended solely for research, education, and authorized investigative use.

Researchers and users are expected to adhere to ethical standards and local regulations regarding data collection and privacy.

# Chapter 3

## Features Implementation

### 3.1 User Friends List Crawling

This feature utilizes Selenium (Python) to simulate user login and navigate to a target Facebook profile. By dynamically loading the friend list via scrolling and parsing the DOM, the system extracts usernames, profile links, and mutual connection data.

The crawled data is then saved in the form of (`user_id.json`) file located at `friends_data_*`, enabling further data analysis or reuse without re-crawling.

The system uses a custom timestamp format to label exported files. For instance, a filename such as (`friends_data_114708062025.json`) encodes both time and date information.

- The first four digits (1147) represent the time in 24-hour format, corresponding to **11:47 AM**.
- The remaining eight digits (08062025) follow the `ddmmyyyy` structure, indicating the date **June 8, 2025**.

This naming convention ensures that all crawled datasets are chronologically ordered and traceable.

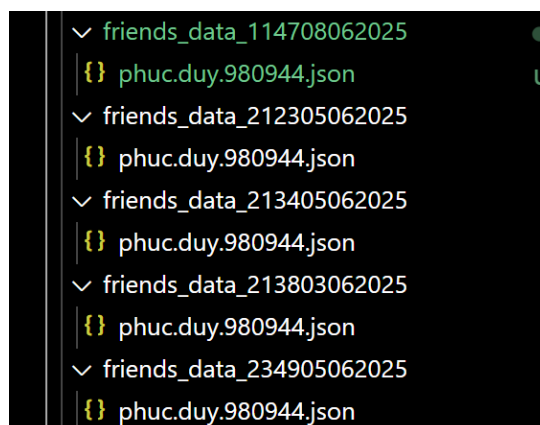
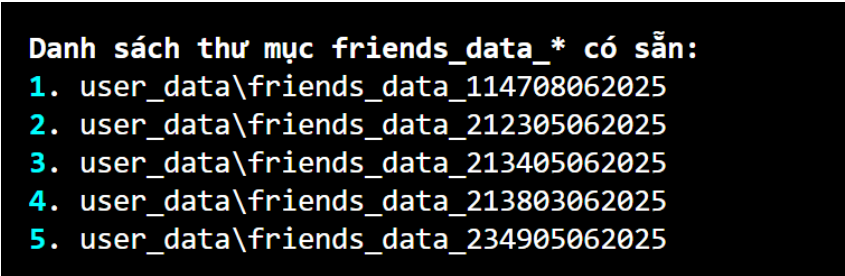


Figure 3.1: Data stored after crawling friends list



## 3.2 User Profile Data Crawling

After collecting the friend list, the tool processes the saved JSON file in (friends\_data\_\*) to extract the list of Facebook profile URLs. Each entry typically contains a unique identifier (user ID or profile link), which is then used to programmatically construct the full URL of the user's Facebook profile.



```
Danh sách thư mục friends_data_* có sẵn:  
1. user_data\friends_data_114708062025  
2. user_data\friends_data_212305062025  
3. user_data\friends_data_213405062025  
4. user_data\friends_data_213803062025  
5. user_data\friends_data_234905062025
```

Figure 3.2: Choose to extract user profile URLs from friend list JSON file

This list of URLs serves as the input for subsequent scraping stages. By iterating through each URL, the system opens the corresponding Facebook profile page, allowing the tool to parse the DOM structure and extract detailed profile information for each user. The tool extracts public profile information such as name, education, current workplace and location. This is handled using Python Selenium automation with page structure parsing techniques tailored to Facebook's dynamic content.

## 3.3 Check-in Information Collection

This is the only feature implemented using Puppeteer (Node.js) due to its superior performance with modern JavaScript-heavy web applications. The crawler simulates user actions, scrolls through the timeline, and extracts posts containing location check-ins. Extracted data includes timestamps, place names, and Facebook post IDs (pfbid).

## 3.4 Graph Generation and Visualization

The final stage of the system focuses on visualizing the collected data through interactive graphs. These visualizations are generated using pre-designed HTML templates and corresponding JavaScript files that extract and render data dynamically. The JavaScript data files are constructed from JSON outputs obtained in previous crawling steps.

Two core libraries are employed for visualization:

- **vis.js**: Used for generating interactive network graphs, including:
  - *Connection Graph*: visualizes friend relationships between users. Nodes represent users, while edges indicate friendships based on the friend list data.
  - *Check-in Graph*: visualizes user check-in locations over time. Each node represents a location, and links depict movement/check-in history.
- **SimpleMaps**: Utilized for plotting the users' residential locations onto an interactive map of Vietnam, providing geographical context to the dataset.

Each graph is constructed by loading a corresponding HTML template which reads structured data from the JavaScript data files (e.g., `location_data.js`, `checkin_data.js`) that are programmatically generated from the raw JSON datasets (e.g., `friends_data_*, (user_id).json`, etc.).

Furthermore, to enrich the user experience and analysis, the graphs include metadata displayed via tooltips on hover. These tooltips present essential details extracted during the scraping phase.

# Chapter 4

## Demonstration

### 4.1 Environmental Setup

To ensure that the tool runs properly, a suitable environment must be prepared beforehand. The tool consists of two main components implemented using different technologies: Python (with Selenium) and Node.js (with Puppeteer).

- **Python Environment Setup:**

- Python 3.10 or higher is required.
- All required Python packages are specified in the `requirements.txt` file.
- To install, navigate to the project root directory and run:

```
pip install -r requirements.txt
```

- **Node.js Environment Setup:**

- Navigate to the subfolder containing the check-in crawler and run:

```
npm i
```

- This installs all necessary dependencies defined in `package.json`, including Puppeteer-core.

### 4.2 Login and Initial Execution

Once all dependencies are installed, the user must log into the Facebook clone account to initialize the session.

- To start the login process, we executed:

```
python main.py login-2-step
```

The process launched a browser window and printed real-time status messages to the CLI interface. During the process, the user was prompted to manually enter a security code to complete authentication.

```

Step 1 of 5 - Closing cookie term modal
Step 2 of 5 - Logging in
🔒 Security code: 887730
Step 3 of 5 - Adding security code
Step 4 of 5 - Saving browser
Step 5 of 5 - Saving cookies
✅ Logging successful after 23.82126212120056 seconds ✅

```

Figure 4.1: CLI output during 2-step verification login

- After a successful login, the main interface of the tool can be started by running:

```
python build_demo.py
```

This command opens a command-line menu interface, allowing the user to select one of the available features provided by the tool.

```

=== Công cụ thu thập thông tin người dùng trên Facebook ===
1. Thu thập danh sách bạn bè người dùng
2. Thu thập dữ liệu hồ sơ người dùng
3. Thu thập thông tin checkin
4. Tạo biểu đồ quan hệ (kết nối, vị trí)
5. Tạo biểu đồ checkin
6. Dừng chương trình
Nhập lựa chọn (1-6):

```

Figure 4.2: Main CLI menu of the Meta-Spy tool

### 4.3 Friend List Collection

To begin the experiment, we selected the **Friend List Collection** feature from the CLI by entering option 1. When prompted, we input the UID of a target Facebook account and required parameters (e.g., number of layer, friends per node):

```

Nhập lựa chọn (1-6): 1
Nhập ID người dùng để thu thập danh sách bạn bè: phuc.duy.980944
Nhập số tầng cần crawl (0 = chỉ lấy bạn bè trực tiếp): 2

Nhập số bạn bè cần thu thập cho mỗi node ở từng tầng:
Tầng 0 (bạn bè trực tiếp): Số bạn bè cần thu thập cho mỗi node: 4
Tầng 1: Số bạn bè cần thu thập cho mỗi node: 2
Tầng 2: Số bạn bè cần thu thập cho mỗi node: 2

```

Figure 4.3: Friend list crawling configuration entered in CLI

The tool logged into the session using the previously saved cookies and navigated to the user's public friend list page. Using automatic scrolling and DOM parsing, it extracted all visible friend entries.

```

✓ Hoàn thành crawl dữ liệu!
📁 Thống kê:
- Tầng 0 (bạn bè trực tiếp): 5 node
- Tầng 1: 5 x 2 = 10 node
- Tầng 2: 10 x 2 = 20 node
- Tổng số tài khoản dự kiến: 35
- Tổng số tài khoản đã crawl: 20
- Thời gian crawl: 023609062025
- Dữ liệu đã được lưu tại: user_data\friends_data_023609062025\phuc.duy.980944.json

```

Figure 4.4: Friend list crawling result shown in CLI

**Observation:** The tool successfully collected **20 friends** over **35 friends** in prediction. The output was saved in the file:

user\_data/friends\_data\_023609062025/phuc.duy.980944.json

This file contains structured data in JSON format with fields such as `uid`, `name`, and `profile_url` for each friend. The data was cleanly formatted and ready for downstream processing.

## 4.4 User Profile Scraping

After collecting the friend list, we proceeded to extract public profile metadata of each user. This is done by selecting option 2 in the CLI interface.

Upon launching the feature, the system begins with initialization steps:

1. Scanning all available friend list directories matching the pattern `friends_data_*`
2. Prompting the user to choose which dataset to process
3. Reading the JSON file(s) containing profile URLs
4. Iterating through each user to scrape public profile fields

```

Nhập lựa chọn (1-6): 2
Bắt đầu quá trình khởi tạo...
Bắt đầu chương trình chính...
Bước 1: Tìm kiếm thư mục friends_data_*
Bước 2: Chọn thư mục để xử lý

Danh sách thư mục friends_data_* có sẵn:
1. user_data\friends_data_023609062025
2. user_data\friends_data_114708062025
3. user_data\friends_data_212305062025
4. user_data\friends_data_213405062025
5. user_data\friends_data_213803062025
6. user_data\friends_data_234905062025

Chọn số thứ tự thư mục (1-6): 1
Đã chọn: user_data\friends_data_023609062025
Bước 3: Tìm kiếm file JSON trong thư mục đã chọn
Bước 4: Đọc dữ liệu từ file JSON
Bước 5: Thu thập thông tin profile từ dữ liệu JSON

Bắt đầu duyệt cây dữ liệu

```

Figure 4.5: CLI interface for selecting the friend list dataset to extract profile information

- For this experiment, we selected the dataset:

user\_data/friends\_data\_023609062025

The program began reading the tree-structured friend list and initialized the profile scraping process for each user node.

**Note:** The scraping pipeline uses Selenium to navigate to each user profile and extract all publicly available fields (e.g., Hometown and current Location, Education, Relation Status, etc.).

```
Đã lưu dữ liệu thành công vào user_data\data_023609062025\phuc.duy.980944.json
Chương trình đã hoàn thành!
```

Figure 4.6: Confirmation of profile data saved after scraping

**Observation:** The profile data is now ready for use in subsequent stages, particularly the graph generation modules (Connection Graph, Location Graph, etc.), where each node will contain metadata from this JSON structure.

## 4.5 Graph Visualization

### 4.5.1 Connection Graph

We then proceeded to visualize the social network graph using option 4 from the main CLI. This module renders an interactive connection graph based on previously scraped profile data.

```
Nhập lựa chọn (1-6): 4
Danh sách thư mục dữ liệu:
1. user_data\data_023609062025
2. user_data\data_114708062025
3. user_data\data_212305062025
4. user_data\data_212305062025_1
5. user_data\data_213405062025
6. user_data\data_213803062025
7. user_data\data_234905062025
Chọn số thứ tự thư mục để xuất dữ liệu: 1

Chọn chức năng:
1. Tạo biểu đồ kết nối bạn bè
2. Tạo biểu đồ vị trí địa lý
Nhập số thứ tự chức năng: 1
✅ Dữ liệu đã được ghi vào: graph-create\friends_data.js
🌐 Đã mở graph-create\connection.html trên trình duyệt mặc định
```

Figure 4.7: CLI interface for selecting dataset and generating friend connection graph

#### Steps taken:

1. The system listed all available directories containing profile data (data\_\*)
2. We selected the target dataset:

user\_data/data\_023609062025

3. When prompted to choose the graph type, we selected:

1. Tạo biểu đồ kết nối bạn bè (Friend Connection Graph)

The program converted all JSON profiles into a JavaScript data format and saved it as:

graph-create/friends\_data.js

This data file was then loaded by the HTML template `connection.html`, which uses the `vis.js` library to render the graph in an interactive browser window.

#### Graph structure:

- Each node represents a Facebook user.
- Edges indicate friendship links between nodes.
- Hovering over a node displays a tooltip containing user information.

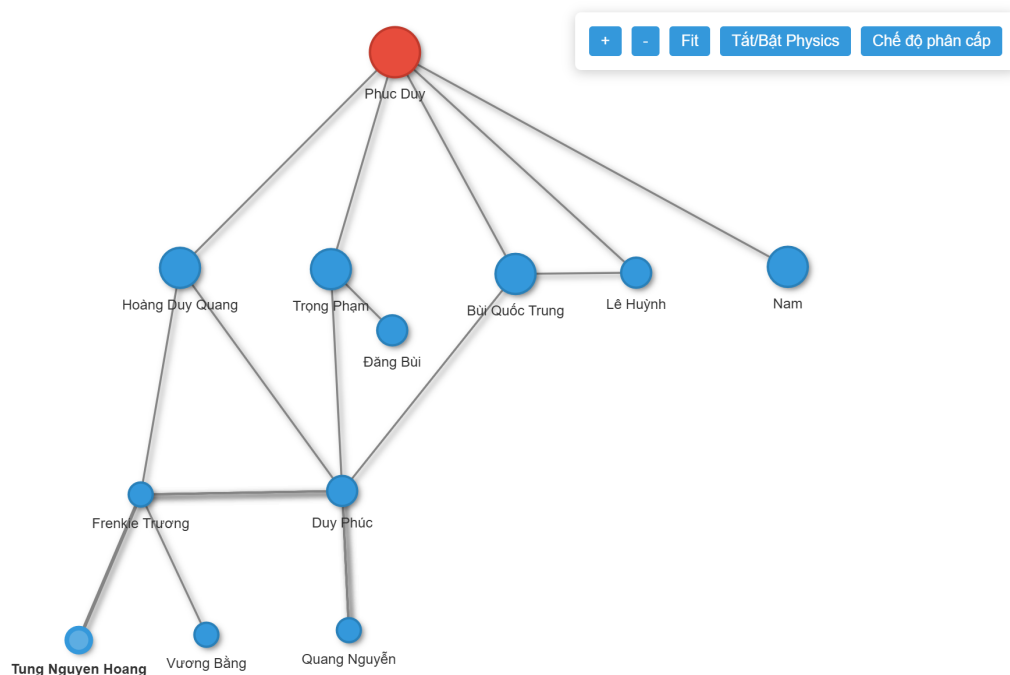


Figure 4.8: Interactive friend connection graph rendered using vis.js

**Observation:** This graph allows analysts to observe the structural properties of a user's social circle, identify central or isolated nodes, and trace connections across multiple layers.

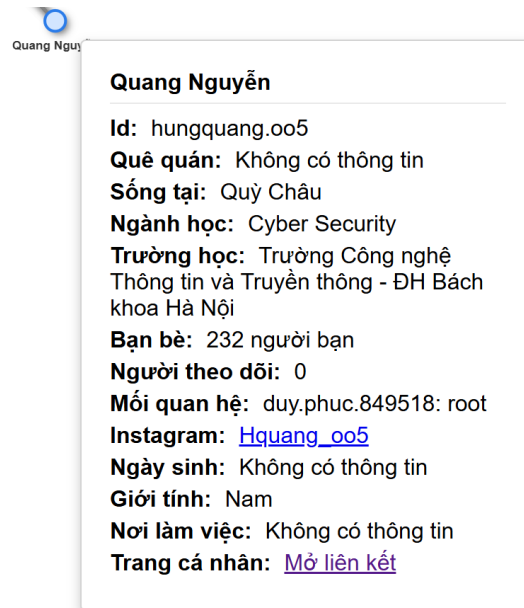


Figure 4.9: Tooltip showing user profile details on graph node hover

## 4.5.2 Location Graph

In addition to analyzing user connections, we visualized the geographic distribution of users based on the hometown or current city information scraped during profile extraction. This feature was accessed by selecting option 4 in the CLI menu, followed by option 2 for geographic visualization.

### Steps performed:

1. The CLI listed all available directories containing profile data (data\_\*)
2. We selected the dataset:

```
user_data/data_023609062025
```

3. We chose the function:

```
2. Tạo biểu đồ vị trí địa lý (Location Graph)
```

4. The system generated a JavaScript file:

```
graph-create/location_data.js
```

5. It then opened the visualization in:

```
graph-create/location.html
```



```
Nhập lựa chọn (1-6): 4
Danh sách thư mục dữ liệu:
1. user_data\data_023609062025
2. user_data\data_114708062025
3. user_data\data_212305062025
4. user_data\data_212305062025_1
5. user_data\data_213405062025
6. user_data\data_213803062025
7. user_data\data_234905062025
Chọn số thứ tự thư mục để xuất dữ liệu: 1

Chọn chức năng:
1. Tạo biểu đồ kết nối bạn bè
2. Tạo biểu đồ vị trí địa lý
Nhập số thứ tự chức năng: 2
✅ Dữ liệu đã được ghi vào: graph-create\location_data.js
🌐 Đã mở graph-create\location.html trên trình duyệt mặc định
```

Figure 4.10: CLI interface for selecting dataset and generating location visualization graph

This location-based graph was rendered using the SimpleMaps JavaScript library, with a focus on Vietnamese geography. The map highlights provinces where at least one user is located, based on profile information.

- Provinces that contain users will be visually highlighted on the map.
- When hovering over a highlighted province, a tooltip appears showing the list of users associated with that location.

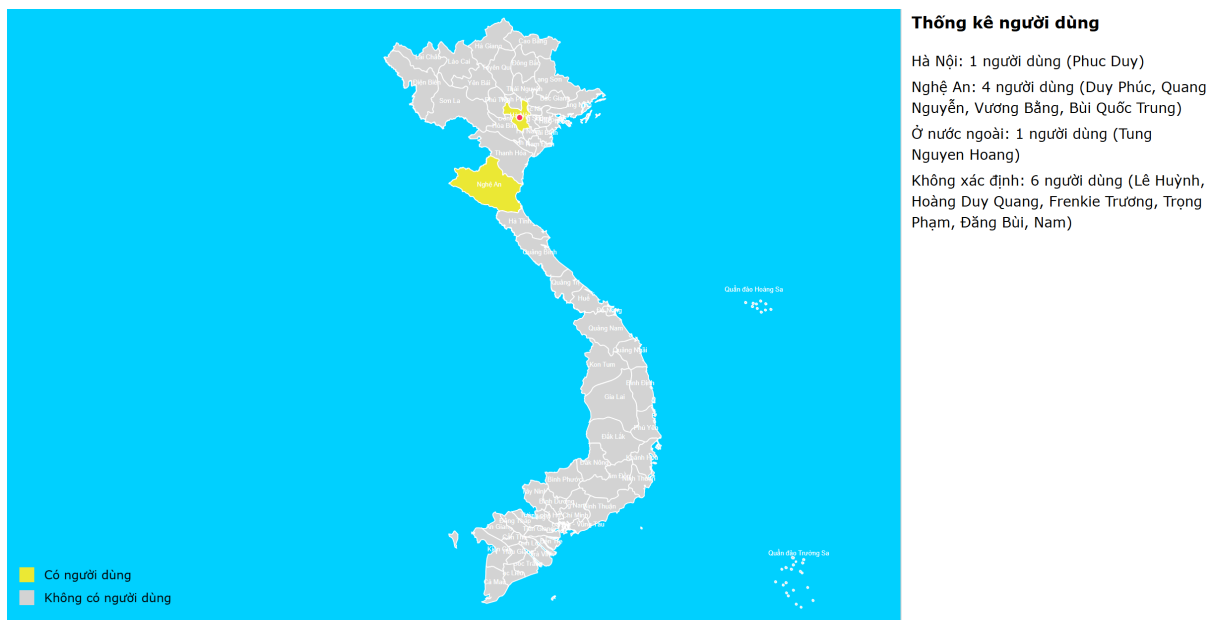


Figure 4.11: Geographic distribution of users on the map of Vietnam (SimpleMaps)

**Observation:** This visualization provides a quick overview of the user's social reach and geographic connections, which can be useful for regional analysis and community behavior studies.

## 4.6 Check-in Data Collection and Visualization

To extract a user's location history on Facebook, we selected option 3 in the CLI menu. This feature activates a Node.js module based on Puppeteer to scroll through timeline posts and identify those containing check-in data.

### Parameters used:

Username/UID: phuc.duy.980944

Number of posts to scroll: 5

Date filter: May 30, 2025

```
Nhập lựa chọn (1-6): 3
> crawlpostfb@1.0.0 start
> node main.js

Nhập username/uid (bắt buộc): phuc.duy.980944
Nhập số bài muốn tìm check-in(cuộn qua n bài): 5
Nhập khoảng năm: 2025
Nhập khoảng tháng: 5
Nhập khoảng ngày: 30
Crawling posts, please wait... (It will take a long time)
Found 5 posts.
Getting pfbid and time for each post, please wait... (It will take a long time)
Found 4 posts with bfbid and time.
Crawling completed.
```

Figure 4.12: Log from CLI showing the check-in crawl process

The crawler successfully identified 5 posts, of which 4 contained valid check-in data with timestamp and Facebook post ID.

- **Resulting data was saved to:**

checkin\_src/user\_data/checkin\_data\_022509062025.json

We then visualized this check-in history using option 5 in the CLI. The system listed all available check-in JSON files and prompted us to select one. It then exported the data to JavaScript format and opened an interactive HTML visualization using the `vis.js` library.

```
=== Công cụ thu thập thông tin người dùng trên Facebook ===
1. Thu thập danh sách bạn bè người dùng
2. Thu thập dữ liệu hồ sơ người dùng
3. Thu thập thông tin checkin
4. Tạo biểu đồ quan hệ (kết nối, vị trí)
5. Tạo biểu đồ checkin
6. Dừng chương trình
Nhập lựa chọn (1-6): 5
Danh sách file checkin_data_*.json:
1. checkin_src\user_data\checkin_data_022509062025.json
2. checkin_src\user_data\checkin_data_030708062025.json
3. checkin_src\user_data\checkin_data_032409062025.json
4. checkin_src\user_data\checkin_data_185307062025.json
5. checkin_src\user_data\checkin_data_203007062025.json
Chọn số thứ tự file để xuất dữ liệu: 1
✅ Dữ liệu đã được ghi vào: checkin_src\checkin_graph\checkin_data.js
🌐 Đã mở checkin_src\checkin_graph\checkin.html trên trình duyệt mặc định
```

Figure 4.13: Selecting check-in dataset to generate graph

The resulting graph shows the user's movement between different check-in locations over time. Each node represents a location, and directed edges represent the temporal order of check-ins. Hovering over a node displays a tooltip with the timestamp and a clickable post link.

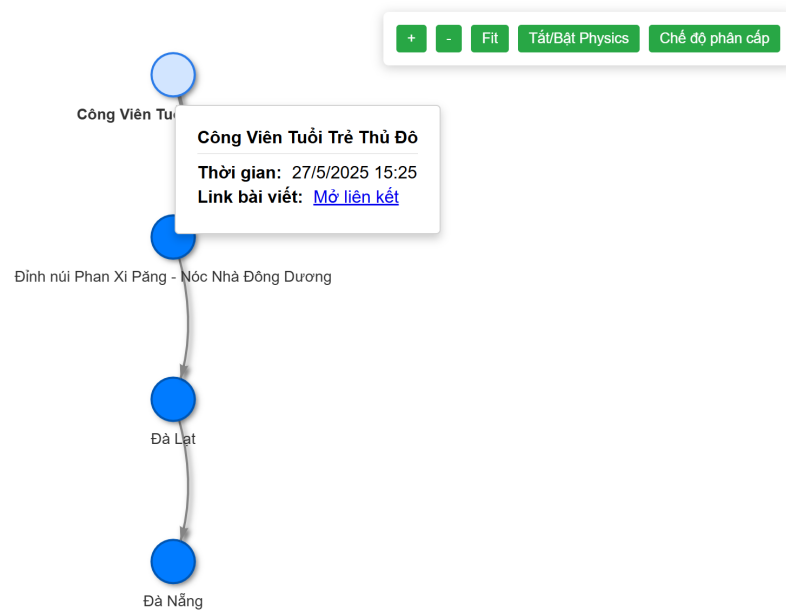


Figure 4.14: Check-in timeline graph and Tooltip showing time and link for each check-in node

**Observation:** The check-in visualization effectively illustrates the sequence and diversity of the user's physical locations. Despite a small dataset, the tool rendered a clean and informative travel graph with minimal user input.

# Chapter 5

## Results Evaluation

### 5.1 Evaluation Metrics

To evaluate the effectiveness of the system, we define four key criteria:

- **Data completeness:** Measures the ratio of successfully collected data over the expected number of user entities and content per module.
- **System reliability:** Assesses the tool’s ability to recover from partial failures such as timeouts, inaccessible profiles, or broken pages.
- **Execution time:** Average duration for each major task (friend list crawling, profile scraping, check-in retrieval, graph rendering).
- **User interaction clarity:** Evaluates how easily users can operate the command-line interface (CLI) and navigate through features.

### 5.2 Experimental Outcomes

The tool was executed on a real Facebook account with a predefined configuration:

- Crawl depth: 2
- Friends per node: {5 (Layer 0), 2 (Layer 1), 2 (Layer 2)}
- Check-in scroll limit: 5 posts

The experiment yielded the following results:

- **Friend list crawling:** A total of 20 accounts were crawled, out of an expected 35 nodes. The shortfall was caused by overlapping friendships—multiple nodes shared common friends, leading to reduced unique counts.
- **User profile scraping:** Publicly accessible profiles were fully scraped. Any restricted or deactivated profiles were skipped with appropriate logging, ensuring continuity.
- **Check-in data collection:** From the specified 5 posts, 4 valid check-in posts were successfully parsed and recorded with timestamps and locations.
- **Graph generation:** All three visualizations—friend connections, geographic location, and check-in trails—were rendered correctly and loaded on the browser.

## 5.3 Performance Summary

Execution times depended heavily on the number of nodes and data richness of each profile. Specifically, friend crawling time increased proportionally with the number of friends requested per user.

The observed time performance metrics are as follows:

- **Friend list crawling:** Average of 20 seconds per node, scaling linearly with the friend count requested.
- **Profile scraping:** Approximately 2 minutes per profile due to dynamic page rendering and multi-tab parsing.
- **Check-in scraping:** 80–100 seconds to scroll and evaluate 5 timeline posts.
- **Graph rendering:** Instant once data files (JavaScript) are prepared; all visuals use static HTML templates with embedded JS data.

## 5.4 Error Handling and Logging

The system applies structured exception handling with fallback mechanisms at each stage. In case of timeouts or missing DOM elements, errors are caught and logged, and the crawler continues with the next item. Logging is encoded in UTF-8 to fully support Vietnamese output, and progress indicators are printed to the CLI for user awareness.

## 5.5 Code Quality Analysis

To evaluate maintainability and code safety, the tool was integrated with **SonarQube** via a Jenkins pipeline. The pipeline included two stages: SCM (source fetch) and static analysis.

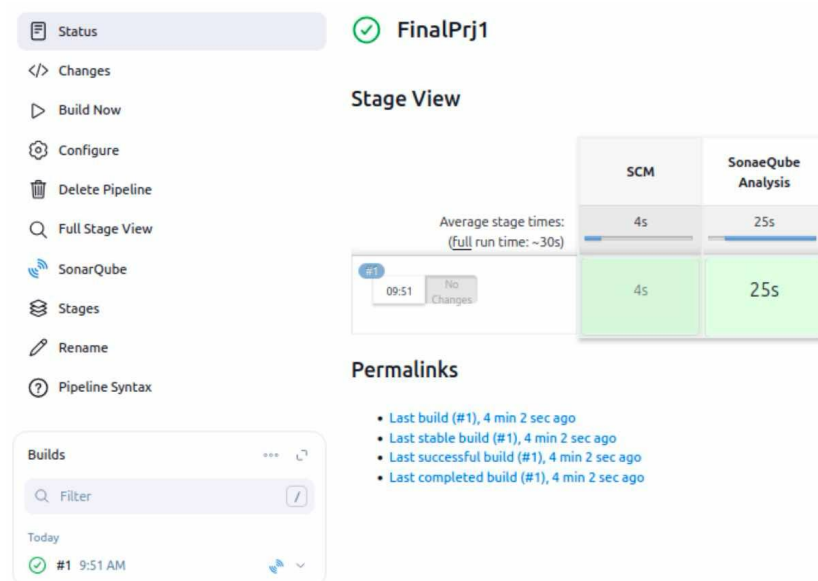


Figure 5.1: Jenkins pipeline execution with SonarQube integration

The SonarQube scan returned clean results:

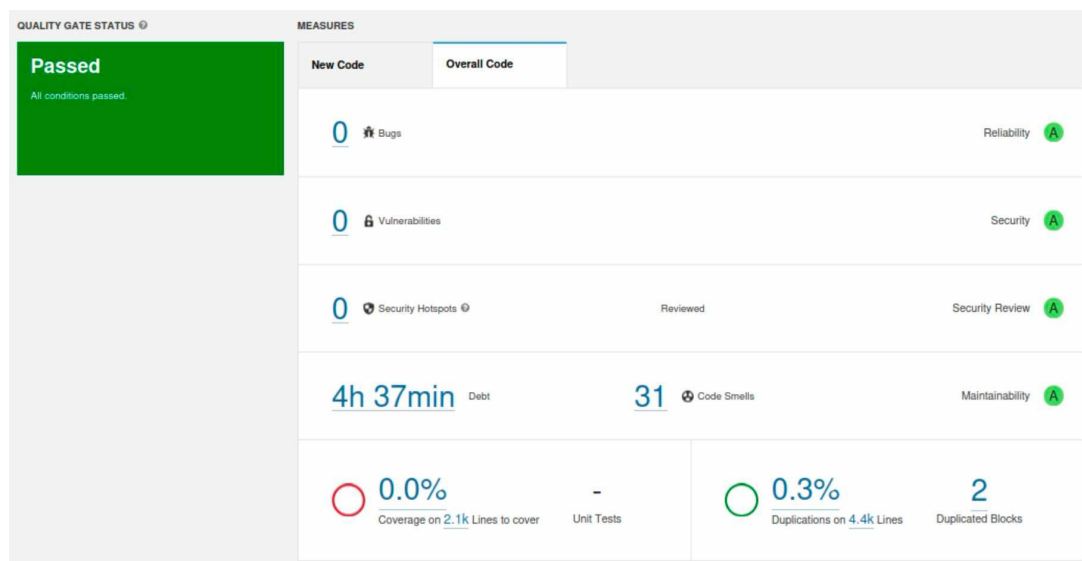


Figure 5.2: SonarQube analysis

Above is the response from SonarQube when we scan our code through Jenkins, which shows 0 bugs, 0 vulnerabilities, 0 security hotspots, but there are some code smells.

After several researches, we concluded that these smells do not alter the code or affect the security of our program.

# Chapter 6

## Challenges and Future Work

### 6.1 Challenges Encountered

During development and testing of the tool, we encountered several challenges that required manual workarounds or limited functionality:

- **Dynamic DOM structure on Facebook:** Facebook frequently changes its front-end layout and obfuscates class names, which leads to broken selectors and scraping failures.
- **Two-factor authentication (2FA):** While login automation works, the 2-step verification code must be entered manually by the user, preventing fully unattended runs.
- **Session handling complexity:** Preserving sessions across Python (Selenium) and Node.js (Puppeteer) modules required custom cookie management and consistent handling of storage formats.
- **Rate limiting and blocking:** Facebook imposes temporary restrictions when it detects suspicious activity (frequent crawling, high request rates), which can halt the crawling process.
- **Limited access to private data:** Only publicly available information can be extracted. If a user restricts visibility of their friends list, timeline, or profile details, the tool cannot retrieve them.
- **Redundant profile scraping:** A single user may appear in multiple branches of the friend tree (e.g., mutual friends). Without deduplication, the tool may repeatedly crawl the same profile, wasting time and resources.

### 6.2 Future Work

To address the limitations above and improve the tool's robustness, scalability, and usability, we propose the following directions for future development:

- **Automatic DOM adaptation:** Integrate a selector auto-matching module using machine learning or CSS pattern discovery to adapt to Facebook UI changes dynamically.
- **Headless login with 2FA bypass:** Explore cookie importing via browser sessions or automation via virtual input to reduce the need for manual intervention in the login phase.

- **Parallel crawling architecture:** Introduce multithreading or asynchronous I/O to enable faster large-scale data collection across multiple accounts.
- **Graph database integration:** Output data to Neo4j or other graph engines to allow advanced queries and analytics on the social graph.
- **Cross-platform extensibility:** Extend the crawling framework to cover other social media platforms (e.g., Instagram, LinkedIn) to build a comprehensive OSINT pipeline.
- **GUI interface for non-technical users:** Replace CLI with a graphical interface to broaden accessibility and ease of use for analysts and investigators.
- **Duplicate detection and skipping:** Add a caching mechanism to keep track of already-crawled user IDs, avoiding redundant requests for users that appear in multiple friend branches.

**Conclusion:** While the current tool proves functional and informative, it remains an evolving prototype. The proposed enhancements aim to improve resilience, automation, and cross-platform capabilities, making it more practical for real-world OSINT and cybersecurity applications.



# Bibliography

- [1] OSINT. *Open-Source Intelligence (OSINT)*: <https://www.imperva.com/learn/application-security/open-source-intelligence-osint/>
- [2] Facebook. *Terms and Policies*: <https://www.facebook.com/policy.php>
- [3] Github Contributors. *MetaSpy*: <https://github.com/DEENUU1/meta-spy>
- [4] Puppeteer Project. *Puppeteer Documentation*: <https://pptr.dev/>
- [5] SeleniumHQ. *Selenium WebDriver Documentation*: <https://www.selenium.dev/documentation/>
- [6] vis.js Team. *vis.js Network Visualization Library*: <https://visjs.org/>
- [7] SimpleMaps. *Interactive Map Library*: <https://simplemaps.com/custom/country>
- [8] *Github of Project*: [https://github.com/DuyPhuc-hust/Project\\_1](https://github.com/DuyPhuc-hust/Project_1)