Sửa code

- Đã sửa một vài code cho nó chạy tốt:

1. Sửa code trong `pentest.py` : vì nó bị lỗi click, sửa lại cú pháp cho chạy được thôi
2. Dùng Gemini API vì nó có free plan: sửa lại codebase vài chỗ để nó hỗ trợ Gemini
3. Viết thêm file Docker compose để chạy các service.

Tóm tắt các bước thiết lập và chạy VulnBot

1. Chuẩn bị môi trường

```
git clone <repo-url>
cd VulnBot-main
python -m venv venv
source venv/bin/activate  # Linux/macOS: .\venv\Scripts\activate (Windows)
pip install -r requirements.txt
```

2. Dựng services bằng Docker

```
# Tạo docker-compose.yml với Kali, MySQL, Milvus
docker-compose up -d
# Lấy IP container Kali
docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' kali-for-vulnbot
```

3. Cấu hình

Chỉnh sửa các file trong config:

- **basic_config.yaml**: SSH Kali (hostname, port, user, password)
- **db_config.yaml**: MySQL connection
- **kb_config.yaml**: Milvus settings
- **model_config.yaml**: LLM settings (API key, model name)

4. Khởi tạo

```
python cli.py init
```

5. Tạo Knowledge Base (tùy chọn)

```
python startup.py --all
# Truy cập http://localhost:8501 để tạo KB
# Bật enable_rag: true trong basic_config.yaml
```

6. Chạy pentest

```
python pentest.py
# Hoặc: python cli.py vulnbot -m 10
```

Nhập mô tả mục tiêu → VulnBot sẽ tự động lập kế hoạch, thực thi lệnh trên Kali và cập nhật chiến lược dựa trên kết quả.

Kết quả sau một lần chạy thử và những cải thiện nên có

Phân tích toàn bộ luồng hoạt động

Giai đoạn 1: Quét và thu thập thông tin (Thành công)

1. **Kế hoạch ban đầu:** Agent lập kế hoạch quét toàn bộ các cổng trên `172.21.0.4`. Kế hoạch rất cơ bản và hợp lý.
2. **Thực thi:** Chạy `nmap -p- -sV -sC -T5 172.21.0.4`.
3. **Kết quả:** Phát hiện 2 cổng mở: `3306` (MySQL 8.0.42) và `33060` (MySQL X protocol).
4. **Đánh giá:** Agent đánh giá đây là một thành công (`check_success: yes`) và cập nhật kế hoạch.

Giai đoạn 2: Quét lỗ hổng chuyên sâu (Thành công một phần)

1. **Kế hoạch mới:** Agent quyết định đi sâu hơn vào cổng MySQL, lên kế hoạch chạy các script `mysql-vuln*` của Nmap.
2. **Thực thi:** Chạy `nmap --script mysql-vuln* -p 3306 172.21.0.4` .
3. **Kết quả:** Lệnh chạy xong nhưng không tìm thấy lỗ hổng nào.
4. **Đánh giá:** Agent vẫn đánh giá là thành công (`check_success: yes`) vì lệnh đã thực thi mà không có lỗi. Sau đó, nó nhận ra rằng không còn task nào trong kế hoạch nữa, `updated_plan` trả về `[]` (rỗng).

Giai đoạn 3: Chuyển pha và lập kế hoạch tấn công (Rất thông minh)

1. **Hết kế hoạch cũ:** Khi kế hoạch của pha "Scanner" hoàn thành, agent chuyển sang pha "Exploiter" (hoặc pha tiếp theo).
2. **Tổng hợp thông tin:** Agent gọi `plan_summary` , tạo ra một bản tóm tắt rất hay về những gì đã làm được (`Penetration Testing Summary`).
3. **Kế hoạch tấn công mới:** Dựa trên tóm tắt (biết có MySQL), agent lập một kế hoạch tấn công rất logic:
   - Thử mật khẩu rỗng.
   - Thử brute-force.
   - Tìm exploit đã biết bằng `searchsploit` .

Giai đoạn 4: Thất bại và cố gắng phục hồi (Điểm mấu chốt)

1. **Thử mật khẩu rỗng:** Thất bại (không tìm thấy gì), agent bỏ qua và đi tiếp.
2. **Thử brute-force (Lần 1):** Lệnh `nmap ... passdb='/usr/share/wordlists/fasttrack.txt'` thất bại vì file không tồn tại.
3. **Đánh giá & Kế hoạch mới:** Agent nhận ra thất bại, loại bỏ task cũ và lên kế hoạch mới: chạy `searchsploit` và **tạo một task mới để thử lại brute-force với đường dẫn wordlist khác (`rockyou.txt`)**. Đây là một hành vi rất thông minh, nó nhận ra nguyên nhân thất bại (sai đường dẫn) và cố gắng sửa chữa.
4. **Thử `searchsploit` :** Thất bại vì `searchsploit: command not found` .
5. **Đánh giá & Kế hoạch mới:** Agent lại loại bỏ task thất bại, kế hoạch chỉ còn lại việc thử brute-force với `rockyou.txt` .
6. **Thử brute-force (Lần 2):** Lại thất bại vì `rockyou.txt` cũng không tồn tại.
7. **Đánh giá & Kế hoạch mới (Cực kỳ thông minh):** Đến đây, thay vì bỏ cuộc, agent đã suy luận một cách xuất sắc: "Cả hai lần brute-force đều thất bại vì không tìm thấy wordlist. Mình không biết wordlist nằm ở đâu. Vậy, **hãy dùng lệnh `find` để tìm file `rockyou.txt` trên toàn bộ hệ thống**".
8. **Thực thi `find` :** Lệnh `find` chạy và không trả về kết quả nào (vì file thực sự không có). Agent sẽ lại đánh giá là thất bại và có thể sẽ bị "kẹt" ở đây nếu không có hướng đi khác.

Đánh giá: Nó hoạt động như ý hay chưa? Có tiềm năng không?

Nó đã hoạt động VƯỢT XA mong đợi của một hệ thống cơ bản!

- **Tư duy logic:** Agent thể hiện khả năng lập kế hoạch theo từng giai đoạn, từ thu thập thông tin đến tấn công.
- **Khả năng thích ứng:** Khi một task thất bại, nó không bỏ cuộc hoàn toàn mà cố gắng tìm phương án thay thế. Ví dụ điển hình là việc thử `rockyou.txt` sau khi `fasttrack.txt` thất bại.
- **Khả năng suy luận nguyên nhân:** Đỉnh cao là khi nó nhận ra vấn đề không phải là brute-force sai, mà là **thiếu file**, và nó đã chủ động lên kế hoạch đi tìm file đó bằng lệnh `find`. Đây là một hành vi giải quyết vấn đề rất giống con người.

Tiềm năng là RẤT LỚN. Nếu được cung cấp một môi trường đầy đủ công cụ và quyền hạn phù hợp, agent này hoàn toàn có khả năng thực hiện các bài pentest từ đầu đến cuối một cách tự động.

Những điểm có thể cải thiện

1. **Vòng lặp thất bại do thiếu công cụ:** Vấn đề lớn nhất hiện tại là agent liên tục thất bại vì môi trường không được chuẩn bị sẵn (thiếu `nmap`, `wordlists`, `searchsploit`). Đây không phải lỗi của agent, mà là lỗi của "người điều hành" (chính là bạn).
2. **Đánh giá thành công/thất bại còn đơn giản:** Hiện tại `check_success` chủ yếu dựa vào việc có lỗi hay không. Nó chưa hiểu sâu về "kết quả có ý nghĩa". Ví dụ, lệnh `nmap --script mysql-vuln*` chạy không lỗi, agent coi là thành công, nhưng thực tế là nó không tìm thấy gì (kết quả không có ý nghĩa tấn công).
3. **Quản lý State (Trạng thái):** Agent chưa có cơ chế ghi nhớ các đường dẫn file đã thử và thất bại. Nó có thể sẽ thử lại các đường dẫn sai nếu prompt lặp lại.
4. **Phụ thuộc vào Prompt cứng:** Hành vi của agent phụ thuộc rất nhiều vào các quy tắc trong prompt. Ví dụ, nếu bạn không "gợi ý" đường dẫn wordlist, nó có thể bịa ra các đường dẫn không có thật.

Có nên trao thêm quyền cho Agent không?

Đây là câu hỏi cốt lõi và là một sự đánh đổi kinh điển trong thiết kế Agent: **Sự tự chủ (Autonomy) vs. Sự an toàn (Safety)**.

- **Quyền hạn khiêm tốn (Hiện tại):**

    - **Ưu điểm:** An toàn tuyệt đối. Bạn kiểm soát hoàn toàn môi trường. Agent không thể làm gì ngoài ý muốn (cài backdoor, xóa file hệ thống). Kết quả các lần chạy sẽ nhất quán.
    - **Nhược điểm:** "Giòn". Agent dễ bị thất bại vì những lý do không đáng có. Bạn phải tốn công chuẩn bị môi trường rất kỹ lưỡng.
- **Trao thêm quyền (Ví dụ: cho phép `apt-get`, `wget`, `git clone`):**
    - **Ưu điểm:** Tăng khả năng phục hồi và tự chủ. Agent có thể tự giải quyết vấn đề thiếu công cụ. Nó có thể tự tải wordlist từ Github, tự cập nhật `searchsploit`, tự cài một công cụ mới mà nó đọc được từ một bài hướng dẫn. **Đây là hướng đi để tạo ra một agent thực sự mạnh mẽ.**
    - **Nhược điểm:** Rủi ro an ninh. Agent có thể bị "ảo giác" (hallucination) và chạy một lệnh nguy hiểm (`rm -rf /`), hoặc tải về và chạy mã độc từ một nguồn không tin cậy.

Đề xuất của tôi: **Trao quyền một cách có kiểm soát**

Thay vì cho phép hoàn toàn hoặc cấm hoàn toàn, bạn có thể thực hiện một giải pháp ở giữa:

1. **Tạo một danh sách "Lệnh an toàn" (Whitelist):** Thay vì danh sách đen `FORBIDDEN_COMMANDS`, hãy tạo một danh sách trắng các lệnh mà agent được phép dùng để sửa đổi hệ thống.
    - Ví dụ: `SAFE_INSTALL_COMMANDS = {'apt-get install', 'wget', 'git clone'}`.
    - Khi agent muốn chạy một lệnh trong danh sách này, bạn có thể:
        - **Yêu cầu xác nhận:** In ra một prompt hỏi bạn: "Agent muốn chạy lệnh `wget ...`. Bạn có đồng ý không? (y/n)".
        - **Chỉ cho phép các tham số an toàn:** Cho phép `apt-get install nmap` nhưng cấm `apt-get remove ...`.
2. **Tạo một "Toolbox" riêng:**
    - Trong `docker-compose.yml`, tạo một volume để map một thư mục trên máy host của bạn vào container, ví dụ: `./my_tools:/pentest/tools`.
    - Bạn có thể tải sẵn các wordlists, các script exploit vào thư mục `my_tools` đó.
    - Sửa prompt để "hướng dẫn" agent rằng nếu cần công cụ gì, hãy tìm trong `/pentest/tools` trước.

Lời khuyên cuối cùng:

- **Bước trước mắt:** Hãy làm theo các gợi ý trước, **chuẩn bị sẵn môi trường** trong `docker-compose.yml` bằng cách cài `seclists` và `exploitdb` (cung cấp `searchsploit`).

  ```
  apt-get install -y ... seclists exploitdb
  ```

  Điều này sẽ giải quyết ngay lập tức các lỗi `command not found` và `file not found` mà bạn đang gặp.

- **Bước lâu dài:** Sau khi hệ thống chạy ổn định, hãy suy nghĩ về việc "trao quyền có kiểm soát". Bắt đầu bằng việc cho phép agent tự tải file bằng `wget` vào một thư mục được chỉ định. Đây sẽ là một bước tiến lớn cho sự tự chủ của agent.