Przewodnik dla GitHub Copilot: Budowa Osobistego Systemu Bug Bounty

Kontekst Projektu dla Copilota

Cel: Tworzymy aplikację "Bug Bounty Orchestrator". Jest to osobiste narzędzie do automatyzacji procesu wyszukiwania podatności w aplikacjach webowych. System będzie zarządzał celami, uruchamiał skanowania i agregował wyniki.

Architektura:

- **Rdzeń:** Monolityczna aplikacja backendowa w Pythonie z użyciem frameworka **FastAPI**.
- **Narzędzia Skanujące:** Zewnętrzne narzędzia open-source (subfinder, httpx, nmap, nuclei) uruchamiane w odizolowanych kontenerach **Docker**.
- **Orkiestracja:** Aplikacja FastAPI będzie asynchronicznie wywoływać i zarządzać kontenerami Docker za pomocą biblioteki asyncio.subprocess.
- **Baza Danych: PostgreSQL** do przechowywania celów, zasobów i wyników.
- **Interfejs:** System będzie sterowany przez API (bez interfejsu graficznego w tej fazie).

Konwencje Kodowania:

- Używaj type hints w całym kodzie Python.
- Stosuj modele Pydantic do walidacji danych wejściowych i wyjściowych API.
- Wszystkie operacje I/O (baza danych, wywołania subprocessów) muszą być asynchroniczne (async/await).
- Stosuj czytelne nazwy zmiennych i funkcji.
- Dodawaj docstringi w stylu Google do kluczowych funkcji.

Faza 1: Przygotowanie Środowiska i Projektu

Krok 1.1: Struktura Projektu

Utwórz następującą strukturę folderów i plików:

```
bug-bounty-orchestrator/
├── app/
│   ├── __init__.py
│   ├── main.py           # Główny plik aplikacji FastAPI
│   ├── api/
│   │   ├── __init__.py
│   │   └── v1/
│   │       ├── __init__.py
│   │       └── endpoints/
│   │           ├── __init__.py
│   │           └── scans.py    # Endpointy API do zarządzania skanami
│   ├── core/
│   │   ├── __init__.py
│   │   └── scanner.py      # Logika uruchamiania narzędzi w Dockerze
│   ├── db/
│   │   ├── __init__.py
│   │   ├── base.py         # Konfiguracja połączenia z bazą danych
│   │   └── models.py       # Modele SQLAlchemy
│   └── schemas/
│       ├── __init__.py
│       └── scan.py         # Schematy Pydantic
├──.env                  # Zmienne środowiskowe
├──.gitignore
├── Dockerfile            # Do budowy obrazu aplikacji
├── docker-compose.yml      # Do uruchamiania lokalnego
└── pyproject.toml          # Konfiguracja zależności (Poetry)
```

Krok 1.2: Konfiguracja Zależności

Zainicjuj projekt poetry i dodaj wymagane biblioteki.

Bash

```
poetry init -n
poetry add "fastapi[all]" sqlalchemy "asyncpg[sa]" alembic python-dotenv
```

Krok 1.3: Konfiguracja Bazy Danych (Aiven)

1. Załóż darmowe konto na Aiven.io.[1]
2. Utwórz darmową instancję PostgreSQL.
3. Skopiuj Service URI (ciąg połączeniowy) do pliku .env.

Plik: .env

```
DATABASE_URL="postgres://avnadmin:TWOJE_HASLO@twoj-host-p.aivencloud.com:PORT/defaultdb?sslmode=require"
```

Faza 2: Rdzeń Aplikacji - Backend FastAPI

Krok 2.1: Modele Bazy Danych

Plik: app/db/models.py

▶️ PROMPT DLA COPILOTA:
"Using SQLAlchemy 2.0 declarative mapping with type annotations, create database models for a bug bounty scanning application. I need four tables:

1. Target: should have an id (UUID, primary key), domain_name (string, unique), and

created_at (timestamp).
2. Scan: should have an id (UUID, primary key), a foreign key to Target, a status (string), created_at, and completed_at (nullable timestamp).
3. Asset: should have an id (UUID, primary key), a foreign key to Scan, host (string), ip_address (nullable string), technologies (JSONB), and ports (JSONB).
4. Vulnerability: should have an id (UUID, primary key), a foreign key to Asset, template_id (string), severity (string), description (text), and full_finding (JSONB).
Use sqlalchemy.dialects.postgresql.UUID and JSONB types."

Krok 2.2: Konfiguracja Połączenia z Bazą Danych

Plik: app/db/base.py

▶️ PROMPT DLA COPILOTA:
"Create a database connection setup for a FastAPI application using SQLAlchemy 2.0 async engine.
1. Load the DATABASE_URL from environment variables using python-dotenv.
2. Create an async_engine using create_async_engine.
3. Create an AsyncSessionLocal session factory using async_sessionmaker.
4. Create a Base declarative base class."

Krok 2.3: Schematy Pydantic

Plik: app/schemas/scan.py

▶️ PROMPT DLA COPILOTA:
"Create Pydantic models for my bug bounty API.
1. ScanCreate: A model for creating a new scan, it should only accept a domain (string).
2. ScanResponse: A model for the response after creating a scan, it should include scan_id (UUID) and a message (string).
3. ScanStatus: A model to show the status of a scan, including scan_id (UUID) and status (string).
4. AssetResponse: A model for returning discovered assets, including host (string) and technologies (dict).
5. VulnerabilityResponse: A model for returning found vulnerabilities, including host (string), template_id (string), and severity (string)."

Krok 2.4: Logika Skanera (Rdzeń Orkiestracji)

To jest najważniejsza część. Tutaj zaimplementujemy logikę asynchronicznego uruchamiania kontenerów Docker.

Plik: app/core/scanner.py

▶️ PROMPT DLA COPILOTA:
"Create a Python class ScannerService that runs security tools in Docker asynchronously.

1. The class should have an __init__ method that takes an asyncpg database session.
2. Create an async method _run_command that takes a command as a list of strings. It should use asyncio.create_subprocess_exec to run the command, capture stdout and stderr, and wait for it to complete.[2] It should return the decoded stdout.
3. Create an async method run_subfinder that takes a domain string. It should construct and run the docker run --rm projectdiscovery/subfinder -d {domain} -json command.[3] It should parse the JSON output line by line and return a list of subdomains.
4. Create an async method run_httpx that takes a list of subdomains. It should pass the list to the docker run --rm -i projectdiscovery/httpx -json command via stdin.[6] It should parse the JSON output and return a list of active hosts with their technologies.
5. Create an async method run_nuclei that takes a list of active hosts. It should pass them to docker run --rm -i projectdiscovery/nuclei -json via stdin.[8] It should parse the JSON output and return a list of found vulnerabilities.
6. Create a main orchestrator method start_full_scan that takes a scan_id and domain. This method should sequentially call run_subfinder, run_httpx, and run_nuclei, passing the results from one step to the next. After each step, it should update the scan status and save the results (assets, vulnerabilities) to the database."

Krok 2.5: Endpointy API

Plik: app/api/v1/endpoints/scans.py

▶️ PROMPT DLA COPILOTA:
"Create a FastAPI APIRouter for managing scans.

1. Import necessary dependencies, including APIRouter, Depends, BackgroundTasks, schemas, and the ScannerService.

2. Create a dependency function get_db to provide a database session.
3. Implement a POST / endpoint to create a new scan. It should accept a ScanCreate payload. It should create Target and Scan records in the database, then use BackgroundTasks to add the scanner_service.start_full_scan task to run in the background. It should return a ScanResponse with the new scan_id.
4. Implement a GET /{scan_id}/status endpoint that retrieves and returns the current status of a scan from the database as a ScanStatus object.
5. Implement a GET /{scan_id}/results/vulnerabilities endpoint that retrieves all vulnerabilities associated with a completed scan and returns them as a list of VulnerabilityResponse objects."

Krok 2.6: Główny Plik Aplikacji

Plik: app/main.py

▶️ PROMPT DLA COPILOTA:
"Create the main FastAPI application file.
1. Import FastAPI.
2. Import the scans router from app.api.v1.endpoints.
3. Create the app instance of FastAPI.
4. Include the scans router with a prefix /api/v1/scans.
5. Add a root endpoint GET / that returns {'status': 'ok'}.
6. Add a health check endpoint GET /healthz that returns 200 OK. This will be used to keep the service alive on Render.com.[10]"

Faza 3: Konteneryzacja i Wdrożenie

Krok 3.1: Dockerfile

Plik: Dockerfile

▶️ PROMPT DLA COPILOTA:
"Create a multi-stage Dockerfile for a Python FastAPI application using Poetry.
1. **Builder Stage:** Start from a python:3.11-slim image. Set up a working directory. Install

poetry. Copy pyproject.toml and poetry.lock, then run poetry install without creating a virtual environment and with --no-dev.
2. **Final Stage:** Start from a python:3.11-slim image. Create a non-root user. Copy the installed packages from the builder stage. Copy the application code (./app). Set the CMD to run the application using uvicorn app.main:app --host 0.0.0.0 --port 8000."

Krok 3.2: Docker Compose

Plik: docker-compose.yml

▶️ PROMPT DLA COPILOTA:
"Create a docker-compose.yml file for local development.
It should define one service called api.

1. The api service should build from the local Dockerfile.
2. It should load environment variables from the .env file.
3. It should map port 8000 on the host to port 8000 in the container.
4. It should mount the local ./app directory into /app in the container for live reloading."

Krok 3.3: Wdrożenie na Render.com

1. Utwórz nowe repozytorium na GitHub i wypchnij swój kod.
2. Zaloguj się na(https://render.com) i utwórz nową usługę "Web Service".
3. Połącz swoje repozytorium GitHub.
4. W ustawieniach wdrożenia:
   - **Environment:** Wybierz Docker.
   - **Health Check Path:** Ustaw na /healthz.[10]
   - **Environment Variables:** Dodaj DATABASE_URL i wklej wartość z Twojego pliku .env.
5. Uruchom wdrożenie.

Krok 3.4: Konfiguracja Mechanizmu "Keep-Alive"

Aby zapobiec usypianiu darmowej instancji Render, skonfiguruj zewnętrzny monitoring.[11]

1. Załóż darmowe konto na(https://uptimerobot.com/).

2. Dodaj nowy monitor:
    - **Monitor Type:** HTTP(s)
    - **URL:** Adres URL Twojej wdrożonej aplikacji na Render.com, zakończony ścieżką /healthz (np. https://twoja-aplikacja.onrender.com/healthz).
    - **Monitoring Interval:** Ustaw na 5 minutes.
3. Zapisz monitor. Usługa będzie teraz regularnie odpytywana, co zapobiegnie jej uśpieniu.

Po wykonaniu tych kroków będziesz mieć w pełni działający, zautomatyzowany system do wyszukiwania podatności, zbudowany od podstaw z pomocą GitHub Copilot i działający w chmurze bez żadnych kosztów.

Cytowane prace

1. Create hosted PostgreSQL® database for FREE - Aiven, otwierano: września 24, 2025, https://aiven.io/free-postgresql-database
2. How can I run an external command asynchronously from Python? - Stack Overflow, otwierano: września 24, 2025, https://stackoverflow.com/questions/636561/how-can-i-run-an-external-command-asynchronously-from-python
3. projectdiscovery/subfinder: Fast passive subdomain enumeration tool. - GitHub, otwierano: września 24, 2025, https://github.com/projectdiscovery/subfinder
4. projectdiscovery/subfinder - Docker Image, otwierano: września 24, 2025, https://hub.docker.com/r/projectdiscovery/subfinder
5. Image Layer Details - projectdiscovery/subfinder:v2.4.9 | Docker Hub, otwierano: września 24, 2025, https://hub.docker.com/layers/projectdiscovery/subfinder/v2.4.9/images/sha256-6a5d2ccccccb36d75c10afb801f81fc4404c2a688a2ac8076249a8a39a988dfc
6. A Detailed Guide on httpx. httpx is a fast web application... | by Md Shahriar Atik Shifat | Medium, otwierano: września 24, 2025, https://medium.com/@shahriar.atik/a-detailed-guide-on-httpx-9fec63536be5
7. projectdiscovery/httpx - Docker Image, otwierano: września 24, 2025, https://hub.docker.com/r/projectdiscovery/httpx
8. projectdiscovery/nuclei: Nuclei is a fast, customizable vulnerability scanner powered by the global security community and built on a simple YAML-based DSL, enabling collaboration to tackle trending vulnerabilities on the internet. It helps you find vulnerabilities in your applications, APIs, networks, DNS, and cloud configurations. - GitHub, otwierano: września 24, 2025, https://github.com/projectdiscovery/nuclei
9. Nuclei: Navigating the Digital Core: The Vulnerability Scanner in Action - Medium, otwierano: września 24, 2025, https://medium.com/@digomic_88027/nuclei-navigating-the-digital-core-the-vulnerability-scanner-in-action-71273641d850
10. How to Run a Full-Time App on Render's Free Tier (Without It Sleeping) - Sergei Liski, otwierano: września 24, 2025, https://sergeiliski.medium.com/how-to-run-a-full-time-app-on-renders-free-tier

-without-it-sleeping-bec26776d0b9
11. Deploy for Free – Render Docs, otwierano: września 24, 2025, https://render.com/docs/free
12. Understanding Latency in Free Backend Hosting on Render.com - Medium, otwierano: września 24, 2025, https://medium.com/@python-javascript-php-html-css/understanding-latency-in-free-backend-hosting-on-render-com-d1ce9c2571de