Budowa Osobistego Systemu do Automatycznego Wyszukiwania Podatności: Przewodnik Implementacyjny

Niniejszy dokument stanowi kompleksową dokumentację techniczną opisującą proces projektowania, budowy i wdrożenia spersonalizowanego systemu do automatycznego wyszukiwania błędów i podatności w aplikacjach webowych. Celem jest stworzenie efektywnego narzędzia do celów zarobkowych (bug bounty), wykorzystując wyłącznie darmowe, ogólnodostępne narzędzia i platformy chmurowe. Projekt czerpie inspirację z zaawansowanych, komercyjnych rozwiązań, takich jak "Bug Detective" [1], adaptując ich logikę przepływu pracy do realiów projektu indywidualnego, o zerowym budżecie.

Plan Architektoniczny: Od Wizji "Bug Detective" do Osobistego Centrum Dowodzenia

Analiza dostarczonej dokumentacji dla aplikacji "Bug Detective" ujawnia wysoce złożoną, rozproszoną architekturę opartą na mikroserwisach, przeznaczoną dla środowisk korporacyjnych.[1] Taki model, choć potężny, jest nieosiągalny i niepraktyczny dla indywidualnego użytkownika. Zamiast ślepo kopiować tę strukturę, należy wydestylować jej fundamentalną logikę operacyjną:

Skanuj -> Analizuj -> Przechowuj -> Raportuj. To właśnie ten przepływ pracy stanowi podstawę dla naszego, znacznie uproszczonego, ale wciąż wysoce efektywnego systemu.

Dekonstrukcja Inspiracji "Bug Detective"

System "Bug Detective" opiera się na wielowarstwowej architekturze, obejmującej warstwę

klienta (Web Dashboard, Mobile App, CLI), bramę API (API Gateway) oraz dedykowane mikroserwisy do detekcji, analizy i raportowania.[1] Każdy z tych komponentów wymagałby osobnego zespołu deweloperskiego i znaczących zasobów obliczeniowych.[1] Próba replikacji tej architektury w skali 1:1 jest z góry skazana na porażkę. Kluczem jest zrozumienie, że nie budujemy produktu komercyjnego, lecz osobiste narzędzie automatyzujące. W związku z tym, koncepcje takie jak "Detection Service" czy "Analysis Service" zostaną przetłumaczone na konkretne, wykonywalne zadania w ramach jednej, spójnej aplikacji.

Monolit w Kontenerze: Pragmatyczna Architektura dla Darmowych Platform

Proponowana architektura opiera się na podejściu monolitycznym, ale z modularnym wykonaniem. Sercem systemu będzie pojedyncza aplikacja napisana w Pythonie z wykorzystaniem frameworka FastAPI, pełniąca rolę "Rdzenia Orkiestracji". Aplikacja ta nie będzie sama przeprowadzać skanów, lecz będzie zarządzać uruchamianiem wyspecjalizowanych narzędzi open-source, z których każde będzie działać w odizolowanym kontenerze Docker. Cały system będzie zarządzany za pomocą jednego pliku docker-compose.yml.

Takie podejście bezpośrednio adresuje złożoność modelu "Bug Detective", konsolidując logikę wielu mikroserwisów w jednym, łatwym do zarządzania kodzie. Architektura wysokopoziomowa przedstawia się następująco:

- **Użytkownik -> API FastAPI:** Punkt wejścia do zarządzania celami i inicjowania skanów.
- **Orkiestrator FastAPI -> Silnik Docker:** Główna logika aplikacji, gdzie wywołania API skutkują wykonaniem poleceń docker run dla poszczególnych narzędzi skanujących.
- **Narzędzia Skanujące (Kontenery) -> Cel:** Izolowane narzędzia wykonujące zadania rozpoznania i skanowania podatności.
- **Orkiestrator FastAPI <-> Baza Danych PostgreSQL:** Trwałe przechowywanie danych o celach, odkrytych zasobach, statusie skanów i znalezionych podatnościach.

Rdzeń Rozpoznawczy i Skanujący: Zestaw Narzędzi Open-Source

Fundamentem skuteczności systemu jest dobór odpowiednich narzędzi. Wybór padł na zestaw rozwijany przez projectdiscovery.io, który stał się de facto standardem w branży bug

bounty, uzupełniony o klasyczne narzędzie do mapowania sieci. Kluczowym kryterium wyboru była nie tylko efektywność, ale przede wszystkim natywne wsparcie dla strukturyzowanych formatów wyjściowych (JSON), co jest niezbędne dla automatyzacji.

Racjonalizacja Wyboru Narzędzi

Wybrano następujące narzędzia ze względu na ich wydajność (napisane w Go), aktywny rozwój oraz doskonałą integrację w zautomatyzowanych przepływach pracy [2]:

- **Subfinder:** Do szybkiej, pasywnej enumeracji subdomen z wielu źródeł.[2]
- **HTTPX:** Do masowego sondowania HTTP/HTTPS, weryfikacji, które z odkrytych subdomen są aktywne, oraz zbierania dodatkowych informacji (tytuły stron, kody statusu, technologie).[3]
- **Nmap:** Do kompleksowego skanowania portów i identyfikacji usług działających na odkrytych hostach.[4]
- **Nuclei:** Potężny, oparty na szablonach skaner podatności, który pozwala na precyzyjne i szybkie wyszukiwanie tysięcy znanych błędów konfiguracyjnych i podatności.[6]

Konteneryzacja Zestawu Narzędziowego

Uruchamianie każdego narzędzia w dedykowanym kontenerze Docker jest fundamentalną decyzją architektoniczną. Eliminuje to problemy z zależnościami (np. specyficzne wersje Go wymagane przez subfinder czy httpx [8]), zapewnia powtarzalność środowiska i izoluje procesy skanowania od systemu hosta. Dzięki temu orkiestrator FastAPI nie musi zarządzać instalacjami, a jedynie wykonaniem poleceń

docker run.

Poniżej przedstawiono polecenia instalacyjne i przykładowe użycie dla każdego narzędzia w trybie zautomatyzowanym:

- **Enumeracja Subdomen (subfinder):**
  - Instalacja obrazu: docker pull projectdiscovery/subfinder.[10]
  - Użycie: docker run --rm projectdiscovery/subfinder -d example.com -json
  - *Opis: Skanuje domenę example.com i zwraca listę znalezionych subdomen w formacie JSON.*
- **Sondowanie HTTP/HTTPS (httpx):**

    - Instalacja obrazu: docker pull projectdiscovery/httpx.[11]
    - Użycie: cat subdomains.txt | docker run --rm -i projectdiscovery/httpx -json
    - *Opis: Wczytuje listę subdomen ze standardowego wejścia, sprawdza, które z nich odpowiadają na portach HTTP/HTTPS i zwraca szczegółowe informacje w formacie JSON.*
- **Mapowanie Sieci (nmap):**
    - Instalacja obrazu: docker pull instrumentisto/nmap.[4]
    - Użycie: docker run --rm instrumentisto/nmap -sV -T4 -iL targets.txt -oX -
    - *Opis: Przeprowadza szybkie skanowanie z detekcją wersji usług na hostach z pliku targets.txt i zwraca wyniki w formacie XML, który jest łatwy do parsowania.*
- **Skanowanie Podatności (nuclei):**
    - Instalacja obrazu: docker pull projectdiscovery/nuclei:latest.[12]
    - Użycie: docker run --rm projectdiscovery/nuclei -l urls.txt -json-export results.json
    - *Opis: Skanuje listę adresów URL z pliku urls.txt przy użyciu domyślnego zestawu szablonów i zapisuje znalezione podatności do pliku results.json.*

Tabela 1: Zestaw Narzędzi i Konfiguracja Automatyzacji

Nazwa Narzędzia	Cel	Oficjalny Obraz Docker	Przykładowe Polecenie dla Automatyzacji	Kluczowe Flagi Wyjściowe
Subfinder	Enumeracja subdomen	projectdiscovery/subfinder	docker run --rm projectdiscovery/subfinder -d {TARGET} -silent	-json (dla parsowania) lub brak (dla potokowania)
HTTPX	Sondowanie HTTP/HTTPS	projectdiscovery/httpx	cat subdomains.txt | docker run --rm -i projectdiscovery/httpx -silent	-json
Nmap	Skanowanie	instrumentisto/	docker run --rm	-oX - (wyjście

	portów i usług	nmap	instrumentisto/nmap -sV -T4 -iL hosts.txt -oX -	XML na stdout)
Nuclei	Skanowanie podatności	projectdiscovery/nuclei	docker run --rm projectdiscovery/nuclei -l urls.txt -silent	-json-export {FILE} lub -json

Rdzeń Orkiestracji: Budowa Logiki Aplikacji z FastAPI

Centralnym elementem systemu jest aplikacja-orkiestrator, która spaja poszczególne narzędzia, bazę danych i interfejs użytkownika w spójną całość.

Dlaczego FastAPI?

Wybór frameworka FastAPI jest podyktowany jego nowoczesnością, wysoką wydajnością oraz, co najważniejsze, natywnym wsparciem dla operacji asynchronicznych (async/await).[13] Skanowanie bezpieczeństwa to operacja I/O-bound, która może trwać od kilku minut do wielu godzin. Użycie synchronicznego frameworka prowadziłoby do blokowania serwera i timeoutów, czyniąc aplikację bezużyteczną.[15] Architektura asynchroniczna jest zatem nie tyle wyborem, co koniecznością.

Projekt Asynchronicznego Przepływu Pracy

Logika aplikacji musi być zaprojektowana wokół modelu "uruchom i odpytuj". Użytkownik nie może oczekiwać na wyniki w ramach jednego żądania HTTP.

- POST /api/v1/scans: Ten endpoint przyjmuje cel do skanowania (np. {"domain": "example.com"}). Jego zadaniem **nie jest** uruchomienie skanu. Zamiast tego, tworzy on

nowy rekord skanu w bazie danych ze statusem "oczekujący" i natychmiast zwraca odpowiedź 202 Accepted wraz z unikalnym identyfikatorem scan_id.
- GET /api/v1/scans/{scan_id}: Umożliwia cykliczne sprawdzanie statusu skanu (np. "oczekujący", "uruchomiony_subfinder", "uruchomiony_nuclei", "zakończony").
- GET /api/v1/results/{scan_id}: Po zakończeniu skanu, ten endpoint pozwala na pobranie ustrukturyzowanych wyników (subdomen, otwartych portów, podatności) z bazy danych.

Implementacja Asynchronicznego Wykonywania Zadań

Sercem technicznym orkiestratora jest zdolność do uruchamiania zewnętrznych procesów (kontenerów Docker) w sposób nieblokujący. Wykorzystamy do tego wbudowane w Pythona biblioteki asyncio i subprocess.

Funkcja asyncio.create_subprocess_exec jest idealnym narzędziem do tego zadania.[16] Pozwala ona na uruchomienie polecenia

docker run... jako zadania w tle, bez blokowania głównej pętli zdarzeń FastAPI. Aplikacja może w tym czasie obsługiwać inne żądania, np. zapytania o status.

Przykładowy, uproszczony fragment kodu ilustrujący tę koncepcję:

Python

```
import asyncio

async def run_scan_task(scan_id: str, domain: str):
    # Krok 1: Aktualizacja statusu w bazie danych
    await db.update_scan_status(scan_id, "running_subfinder")

    # Krok 2: Uruchomienie subfindera
    proc_subfinder = await asyncio.create_subprocess_exec(
        'docker', 'run', '--rm', 'projectdiscovery/subfinder', '-d', domain, '-json',
        stdout=asyncio.subprocess.PIPE,
        stderr=asyncio.subprocess.PIPE
    )
    stdout, stderr = await proc_subfinder.communicate()
```

```
    if proc_subfinder.returncode == 0:
        # Krok 3: Parsowanie wyników i przekazanie do kolejnego narzędzia
        subdomains = parse_subfinder_json(stdout)
        await db.save_assets(scan_id, subdomains)

        # Krok 4: Uruchomienie kolejnych narzędzi (np. httpx, nuclei) w podobny sposób
        await db.update_scan_status(scan_id, "running_httpx")
        #... logika dla httpx...
    else:
        await db.update_scan_status(scan_id, "failed")
        # Zapisz błędy z stderr do logów

    await db.update_scan_status(scan_id, "completed")
```

Ten kod demonstruje kluczowy łańcuch operacji: uruchomienie zadania, przechwycenie jego standardowego wyjścia (stdout), które zawiera wyniki w formacie JSON [17], a następnie przetworzenie tych danych i zapisanie ich w bazie.

Tabela 2: Definicja Punktów Końcowych API

Ścieżka Endpointu	Metoda HTTP	Opis	Ciało Żądania (Request Body)	Odpowiedź Sukcesu
/api/v1/scans	POST	Inicjuje nowy skan dla podanego celu.	{"domain": "string"}	202 Accepted z {"scan_id": "uuid"}
/api/v1/scans/{scan_id}	GET	Zwraca aktualny status skanu.	Brak	200 OK z {"status": "string"}
/api/v1/results/{scan_id}/assets	GET	Zwraca listę odkrytych zasobów (subdomen).	Brak	200 OK z listą obiektów asset

/api/v1/results/{scan_id}/vulnerabilities	GET	Zwraca listę znalezionych podatności.	Brak	200 OK z listą obiektów vulnerability

Zarządzanie Danymi na Darmowej Platformie

Każdy system analityczny wymaga trwałego magazynu danych. Wybór odpowiedniej, darmowej bazy danych jest kluczowy dla stabilności i funkcjonalności projektu.

Wybór Dostawcy PostgreSQL w Darmowym Planie

Rekomendowanym dostawcą bazy danych jest **Aiven**. Ich darmowy plan dla PostgreSQL jest wyjątkowo hojny i idealnie dopasowany do potrzeb tego projektu. Oferuje on dedykowaną maszynę wirtualną (co eliminuje problemy z "głośnymi sąsiadami"), 1 GB przestrzeni dyskowej, 1 GB RAM oraz, co istotne, automatyczne kopie zapasowe.[19]

Należy być świadomym ograniczeń: limit 1 GB na dane oraz maksymalnie 20 jednoczesnych połączeń.[20] Architektura naszej aplikacji, z jednym orkiestratorem, bez problemu mieści się w tych limitach.

Projekt Schematu Bazy Danych

Inspirując się schematem z "Bug Detective" [1], tworzymy uproszczoną, ale w pełni funkcjonalną strukturę tabel w PostgreSQL:

SQL

```
-- Tabela do przechowywania głównych celów skanowania
CREATE TABLE targets (
```

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    domain_name VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Tabela do śledzenia poszczególnych zadań skanowania
CREATE TABLE scans (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    target_id UUID REFERENCES targets(id),
    status VARCHAR(50) NOT NULL DEFAULT 'pending',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    completed_at TIMESTAMP WITH TIME ZONE
);

-- Tabela do przechowywania odkrytych zasobów (np. subdomen)
CREATE TABLE assets (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    scan_id UUID REFERENCES scans(id),
    host VARCHAR(255) NOT NULL,
    ip_address VARCHAR(45),
    technologies JSONB, -- Przechowuje dane z httpx
    ports JSONB, -- Przechowuje dane z nmap
    discovered_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Tabela do przechowywania znalezionych podatności
CREATE TABLE vulnerabilities (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    asset_id UUID REFERENCES assets(id),
    template_id VARCHAR(255) NOT NULL,
    severity VARCHAR(50) NOT NULL,
    description TEXT,
    full_finding JSONB, -- Przechowuje cały obiekt JSON z Nuclei
    found_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

Logika Przetwarzania Danych

W ramach orkiestratora FastAPI, po otrzymaniu wyników JSON z każdego narzędzia, zostanie zaimplementowana logika do parsowania tych danych i wstawiania ich do odpowiednich tabel

przy użyciu asynchronicznego sterownika bazy danych, takiego jak asyncpg.

Wdrożenie i Utrzymanie na Darmowych Platformach

Ostatnim krokiem jest spakowanie aplikacji i wdrożenie jej na platformie chmurowej, która oferuje darmowy plan. Ten etap wiąże się z pokonaniem specyficznych ograniczeń takich usług.

Pakowanie Aplikacji z Docker Compose

Dostarczony zostanie kompletny plik docker-compose.yml, który definiuje usługę api. Usługa ta będzie budowana na podstawie lokalnego pliku Dockerfile i będzie zawierać naszą aplikację FastAPI. Plik ten nie będzie zawierał definicji bazy danych, ponieważ korzystamy z usługi zarządzanej (Aiven). Zamiast tego, dane dostępowe do bazy Aiven zostaną przekazane do kontenera api jako zmienne środowiskowe.

Wybór Platformy Hostingowej: Render.com

Rekomendowaną platformą do hostowania aplikacji FastAPI jest **Render.com**. Ich darmowy plan wspiera wdrażanie usług webowych bezpośrednio z obrazów Docker, co idealnie pasuje do naszej architektury.[21] Platformy takie jak Vercel, choć popularne, są zoptymalizowane pod kątem aplikacji frontendowych i funkcji serverless, a nie długo działających serwerów backendowych, co czyni je nieodpowiednimi dla naszego orkiestratora.[23]

Krytyczne Wyzwanie: Pokonanie "Usypiającego Serwera"

Najważniejszym aspektem wdrożenia na darmowej platformie jest zrozumienie i obejście jej fundamentalnego ograniczenia: usługi w planie darmowym Render.com są automatycznie "usypiane" (zatrzymywane) po 15 minutach braku aktywności.[22] Dla naszej aplikacji, która musi utrzymywać działające w tle procesy skanowania przez długi czas, jest to problem krytyczny,

który uniemożliwiłby działanie systemu.

Rozwiązanie tego problemu jest proste, ale absolutnie kluczowe dla powodzenia projektu:

1. **Implementacja endpointu /healthz w aplikacji FastAPI.** Będzie to prosty endpoint, który wykonuje szybkie, lekkie sprawdzenie, np. zapytanie do bazy danych (SELECT NOW();), aby potwierdzić, że cała usługa jest w pełni sprawna i ma połączenie ze swoimi zależnościami.[22]
2. **Konfiguracja zewnętrznej usługi monitorującej.** Należy skorzystać z darmowej usługi, takiej jak **UptimeRobot**, i skonfigurować ją tak, aby co 5 minut wysyłała żądanie HTTP GET na adres naszego endpointu /healthz.

Ten mechanizm generuje stały, sztuczny ruch, który sprawia, że z perspektywy platformy Render.com nasza usługa jest ciągle aktywna. Dzięki temu nigdy nie zostanie uśpiona, co pozwala na nieprzerwane działanie długotrwałych zadań skanowania w tle. To nie jest "hack", lecz standardowy wzorzec projektowy dla utrzymywania stanu na platformach z ograniczeniami bezczynności.

Tabela 3: Porównanie Darmowych Platform i Ich Ograniczeń

Platforma	Typ Usługi	Kluczowe Ograniczenia	Przydatność dla Projektu
Render.com	Usługa Webowa	Usypianie po 15 min. bezczynności, limit godzin miesięcznie (750h) [21]	**Wysoka** (problem usypiania jest rozwiązywalny, wsparcie dla Docker)
Aiven.io	Baza Danych (PostgreSQL)	1 GB danych, 1 GB RAM, 20 połączeń, brak SLA [20]	**Wysoka** (limity wystarczające dla projektu osobistego, dedykowana VM)
Vercel	Frontend / Funkcje Serverless	Limit czasu wykonania funkcji, brak wsparcia dla	**Niska** (nieodpowiednia dla backendu

		długo działających procesów [23]	orkiestracji)

Zaawansowane Przepływy Pracy i Personalizacja

Po uruchomieniu podstawowej platformy, jej prawdziwa wartość ujawnia się w możliwości tworzenia zaawansowanych, ukierunkowanych przepływów pracy.

Mistrzostwo w Nuclei: Poza Domyślnymi Skanami

Siła Nuclei tkwi w jego silniku opartym na szablonach. Istnieje ogromna, rozwijana przez społeczność biblioteka szablonów, a także możliwość pisania własnych.[7] Zamiast uruchamiać wszystkie szablony na raz, co generuje dużo szumu informacyjnego, można tworzyć skany ukierunkowane.

Na przykład, po tym jak httpx zidentyfikuje, że na danym hoście działa WordPress, orkiestrator może uruchomić Nuclei, wskazując mu użycie tylko szablonów specyficznych dla WordPressa: docker run --rm projectdiscovery/nuclei -l wordpress-targets.txt -t technologies/wordpress/ Podobnie, można skupić się na konkretnych kategoriach podatności, np. tych z listy OWASP Top 10, korzystając z odpowiednich tagów i szablonów.[26] Taki inteligentny, warstwowy przepływ pracy — od ogólnego rozpoznania do specyficznego skanowania — drastycznie zwiększa stosunek sygnału do szumu i pozwala szybciej znajdować wartościowe podatności.

Budowa Zautomatyzowanego Systemu Powiadomień

Inspirując się funkcjonalnościami "Bug Detective", takimi jak integracja z narzędziami do komunikacji [1], można w prosty sposób zaimplementować system natychmiastowych powiadomień.

Po zakończeniu skanu Nuclei i zapisaniu wyników do bazy danych, orkiestrator może wykonać dodatkowy krok: sprawdzić, czy w ostatnim skanie znaleziono jakiekolwiek podatności o statusie critical lub high. Jeśli tak, aplikacja może sformatować prostą wiadomość i wysłać ją za pomocą żądania HTTP POST na adres URL webhooka w usłudze Discord lub Telegram.

Pozwala to na otrzymywanie powiadomień w czasie rzeczywistym o najważniejszych znaleziskach, bez konieczności ciągłego monitorowania wyników.

Wnioski

Przedstawiona dokumentacja dowodzi, że możliwe jest zbudowanie zaawansowanego, zautomatyzowanego systemu do wyszukiwania podatności przy zerowych kosztach, czerpiąc inspirację z rozwiązań klasy korporacyjnej. Kluczem do sukcesu jest pragmatyczne podejście do architektury i świadome wykorzystanie darmowych narzędzi i platform.

Najważniejsze wnioski i rekomendacje:

1. **Tłumacz, nie kopiuj:** Złożone architektury korporacyjne powinny służyć jako źródło inspiracji dla logiki przepływu pracy, a nie jako sztywny plan do implementacji.
2. **Konteneryzacja jako fundament:** Użycie Dockera dla każdego narzędzia skanującego jest podstawą stabilności, powtarzalności i łatwości zarządzania całym systemem.
3. **Asynchroniczność jest kluczowa:** Wybór asynchronicznego frameworka, takiego jak FastAPI, i zaprojektowanie API w modelu "uruchom i odpytuj" to jedyny realny sposób na obsługę długotrwałych zadań skanowania.
4. **Zarządzaj ograniczeniami:** Darmowe platformy chmurowe oferują ogromne możliwości, ale ich ograniczenia (jak usypianie serwerów) muszą być aktywnie zarządzane poprzez świadome decyzje projektowe, takie jak implementacja endpointów zdrowia i zewnętrznego monitoringu.
5. **Wartość tkwi w przepływie pracy:** Prawdziwa siła systemu nie leży w pojedynczych narzędziach, ale w ich inteligentnym łączeniu w łańcuchy (np. wyniki jednego narzędzia determinują parametry uruchomienia kolejnego) oraz w automatyzacji powiadomień, co przekształca system z pasywnego zbieracza danych w aktywne narzędzie do odkrywania podatności.

Cytowane prace

1. Blueprint Architektury - Bug Detective.pdf
2. projectdiscovery/subfinder: Fast passive subdomain enumeration tool. - GitHub, otwierano: września 24, 2025, https://github.com/projectdiscovery/subfinder
3. 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
4. instrumentisto/nmap - Docker Image, otwierano: września 24, 2025, https://hub.docker.com/r/instrumentisto/nmap
5. How to Install Nmap (Network Scanner) on Linux - Medium, otwierano: września 24, 2025,

https://medium.com/@redswitches/how-to-install-nmap-network-scanner-on-linux-9f6ef23e5dc9
6. 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
7. Testing with Nuclei Templates: Make Your DAST Scans 10x More Accurate, otwierano: września 24, 2025, https://www.appsecengineer.com/blog/testing-with-nuclei-templates-make-your-dast-scans-10x-more-accurate
8. subfinder Tool in Linux - GeeksforGeeks, otwierano: września 24, 2025, https://www.geeksforgeeks.org/linux-unix/subfinder-tool-in-linux/
9. Installing httpx - ProjectDiscovery Documentation, otwierano: września 24, 2025, https://docs.projectdiscovery.io/opensource/httpx/install
10. projectdiscovery/subfinder - Docker Image, otwierano: września 24, 2025, https://hub.docker.com/r/projectdiscovery/subfinder
11. projectdiscovery/httpx - Docker Image, otwierano: września 24, 2025, https://hub.docker.com/r/projectdiscovery/httpx
12. 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
13. Asynchronous Programming with FastAPI: Building Efficient APIs - DEV Community, otwierano: września 24, 2025, https://dev.to/dhrumitdk/asynchronous-programming-with-fastapi-building-efficient-apis-nj1
14. FastAPI CLI, otwierano: września 24, 2025, https://fastapi.tiangolo.com/fastapi-cli/
15. Run external program without hanging server fast-api - The freeCodeCamp Forum, otwierano: września 24, 2025, https://forum.freecodecamp.org/t/run-external-program-without-hanging-server-fast-api/425653
16. 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
17. Retrieving the output of subprocess.call() in Python - GeeksforGeeks, otwierano: września 24, 2025, https://www.geeksforgeeks.org/python/retrieving-the-output-of-subprocesscall-in-python/
18. subprocess — Subprocess management — Python 3.13.7 documentation, otwierano: września 24, 2025, https://docs.python.org/3/library/subprocess.html
19. Create hosted PostgreSQL® database for FREE - Aiven, otwierano: września 24, 2025, https://aiven.io/free-postgresql-database

20. Free plans | Aiven docs, otwierano: września 24, 2025, https://aiven.io/docs/platform/concepts/free-plan
21. Deploy for Free – Render Docs, otwierano: września 24, 2025, https://render.com/docs/free
22. 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
23. Limits - Vercel, otwierano: września 24, 2025, https://vercel.com/docs/limits
24. Fair use Guidelines - Vercel, otwierano: września 24, 2025, https://vercel.com/docs/limits/fair-use-guidelines
25. 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
26. OWASP ASVS Security Evaluation Templates with Nuclei, otwierano: września 24, 2025, https://owasp.org/www-project-asvs-security-evaluation-templates-with-nuclei/
27. OWASP ASVS Security Evaluation Templates with Nuclei, otwierano: września 24, 2025, https://nest.owasp.org/projects/asvs-security-evaluation-templates-with-nuclei