🚀 SENTINEL AI: Kapsamlı Teknik Uygulama ve Detay Planı

Proje Mimarisi: Hibrit AI (Local+Cloud) + PyQt6 GUI + Docker Backend + Linux Target
Takım: Yiğit (System/UI/Security) & Kerem (AI/Data/Backend)

🟦 SPRINT 0: Proje İskeleti ve Altyapı (Temel Atma)

Amaç: Geliştirme ortamının kurulması, klasör hiyerarşisinin sabitlenmesi ve Docker servislerinin ayağa kaldırılması.

📌 Adım 0.1: Klasör Yapısının Kurulması (Ortak)

Aşağıdaki yapıyı birebir oluşturun. Bu yapı tüm import işlemlerinin merkezidir.

- /sentinel-root (Ana proje klasörü)
    - /src (Kaynak kodlar)
        - /core (Backend mantığı, Process Manager, Parserlar)
        - /ui (PyQt arayüz dosyaları, View'lar)
        - /ai (Yapay zeka istemcisi, promptlar)
        - /plugins (Harici araç eklentileri)
        - /tests (Unit testler)
        - main.py (Uygulama giriş noktası)
    - /docker (Docker yapılandırmaları)
        - /llama
        - /api
    - requirements.txt
    - .env.example

📌 Adım 0.2: Docker Servislerinin Kurulumu (Kerem)

Yapay zeka motorunun (Llama 3) çalışacağı konteyner yapısıdır.

- **Dosya:** docker/llama/Dockerfile
    - **İşlem:** Ollama veya uyumlu bir base image kullanın.
    - **Model:** llama3:8b-instruct-q4_K_M (Local kullanım için optimize edilmiş) modelini pull edecek bir script ekleyin .
- **Dosya:** docker-compose.yml

- Service 1: llama-service -> Port 8001:8001. Volume: ./models:/root/.ollama (Modeli her seferinde indirmemek için) .
- Service 2: api-service -> Port 8000:8000. .env dosyasını okuyacak şekilde ayarlayın .

📌 **Adım 0.3: Temel Kütüphaneler (Yiğit)**

- **Dosya:** requirements.txt
  - İçerik:
    - PyQt6 (Arayüz)
    - pydantic (Veri doğrulama)
    - openai (Standart API istemcisi - hem Cloud hem Local Llama için)
    - python-dotenv (Konfigürasyon)
    - defusedxml (Güvenli XML okuma)

🟩 **SPRINT 1: Akıllı Süreç Motoru (The Engine)**

Kritik Kural: Bu Sprint bitmeden, yapay zeka entegrasyonuna (Faz 2) geçilmeyecek. Çünkü AI'nın koşturacağı "at" (Process Manager) henüz hazır değil.

📌 **Adım 1.1: Advanced Process Manager (Yiğit)**

Bu sınıf, UI donmadan terminal komutlarını çalıştıran motordur.

- **Dosya:** src/core/process_manager.py
- **Sınıf:** class AdvancedProcessManager(QObject)
- **Bileşenler:**
  - self._process = QProcess(): Nesne içinde private olarak tanımlayın .
  - **Signals (Sinyaller):**
    - sig_output_stream(text: str, channel: str): UI'ya veri basmak için .
    - sig_process_finished(exit_code: int, log_path: str): İşlem bittiğinde tetiklenir .
- **Metod:** start_process(command: str, args: list)
  - QProcess'in readyReadStandardOutput ve readyReadStandardError sinyallerini dinlemeye başlayın .

- **Teknik Detay (Encoding):** Gelen QByteArray verisini data.data().decode('utf-8', errors='replace') ile string'e çevirin. Bu, Nmap'ten gelen garip karakterlerin programı çökertmesini önler .

📌 Adım 1.2: İnteraktif Girdi Sistemi (Yiğit)

Kullanıcının terminale "y" (Evet) veya "n" (Hayır) diyebilmesi gerekir.

- **Metod:** write_input(self, text: str)
    - Gelen string veriyi (örn: "y") alın.
    - Sonuna mutlaka \n (Enter tuşu) ekleyin .
    - bytes tipine çevirin (text.encode('utf-8')).
    - self._process.write(...) ile gönderin.

📌 Adım 1.3: Loglama ve Session Yönetimi (Yiğit)

- **Mantık:** Her start_process çağrıldığında temp/session_{timestamp}.txt dosyası oluşturun .
- **Akış:** Terminalden akan her satırı hem UI sinyaline (sig_output_stream) gönderin hem de bu dosyaya append modunda yazın.
- **Çıktı:** İşlem bittiğinde dosya yolunu sig_process_finished sinyali ile yayınlayın.

📌 Adım 1.4: Terminal UI (Kerem & Yiğit)

- **Dosya:** src/ui/terminal_view.py
- **Bileşen:** QTextEdit (ReadOnly yapılmalı) .
- **Renklendirme:** sig_output_stream sinyalini dinleyin.
    - Eğer channel == "stderr" ise: Metni Kırmızı (...) basın.
    - Eğer channel == "stdout" ise: Metni Yeşil/Beyaz basın .
- **Butonlar:** Terminal altına "Evet", "Hayır", "Durdur" butonları koyun. Tıklandığında manager.write_input("y") metodunu çağırsın .

🟧 SPRINT 2: Hibrit AI Komut Motoru (The Brain)

Ön Koşul: Sprint 1'deki terminale manuel olarak ping yazıldığında sorunsuz çalışıyor ve durdurulabiliyor olmalı.

📌 Adım 2.1: JSON Şemaları (Kerem)

AI'nın serbest metin yerine "sıkı" (strict) JSON üretmesini sağlayın.

- **Dosya:** src/ai/schemas.py
- **Yapı:** OpenAI response_format uyumlu şema.
    - tool: (String) "nmap", "gobuster" vs.
    - arguments: (List[String]) Örn: ["-sS", "-p-", "192.168.1.1"]. Liste olması Shell Injection riskini azaltır .
    - requires_root: (Boolean) Komut sudo istiyor mu? .
    - risk_level: (String) "low", "medium", "high".

📌 **Adım 2.2: Karar Motoru (Orchestrator) (Kerem)**

- **Dosya:** src/ai/orchestrator.py
- **Mantık:**
    - Kullanıcı girdisini al.
    - Eğer basit bir komutsa (örn: "bana yardım et") -> Local LLM (Docker).
    - Eğer karmaşık bir saldırı senaryosu isteniyorsa -> Cloud (OpenAI GPT-4o-mini).
    - Çıktıyı schemas.py'daki yapıya zorla (strict=True) .

📌 **Adım 2.3: Entegrasyon (Yiğit & Kerem)**

AI'dan gelen veriyi Process Manager'a bağlama anı.

- **Akış:**
    1. Kullanıcı UI'daki input box'a "Ağı tara" yazar.
    2. Orchestrator bunu JSON'a çevirir: {"tool": "nmap", "arguments": ["-sn", "192.168.1.0/24"]}.
    3. Main Controller bu JSON'ı alır.
    4. process_manager.start_process("nmap", ["-sn", "192.168.1.0/24"]) çağrılır.

🟥 **SPRINT 3: Güvenlik, Yetki ve Temizlik**

Amaç: sudo veya root şifresi girmeden, GUI üzerinden güvenli yetki yükseltme.

📌 **Adım 3.1: Pkexec Wrapper (Yiğit)**

- **Dosya:** src/core/process_manager.py (Güncelleme)

- **Mantık:** start_process metoduna requires_root: bool parametresi ekleyin.
- **İşlem:**
    - Eğer requires_root == True ise:
    - Komut listesinin en başına pkexec stringini ekleyin .
    - Örn: ["nmap", "-sS"] -> ["pkexec", "nmap", "-sS"].

📌 **Adım 3.2: Yetki Reddi Yönetimi (Yiğit)**

- **Dinleme:** QProcess çıkış kodunu (exit code) kontrol edin.
- **Kodlar:**
    - 126 veya 127: Kullanıcı parola ekranında "İptal"e bastı veya yanlış girdi demektir .
- **Aksiyon:** Bu kodlar gelirse sig_auth_failed sinyali atın. UI'da "İşlem İptal Edildi: Yetki Verilmedi" uyarısı gösterin.

📌 **Adım 3.3: Secure Cleaner (Yıkıcı) (Yiğit)**

Root yetkisiyle oluşturulan dosyalar (örn: Nmap XML çıktısı) normal yollarla silinemez.

- **Dosya:** src/core/cleaner.py
- **Metod:** force_delete(file_path: str)
- **Algoritma:**
    1. Önce os.remove(path) dene.
    2. PermissionError hatası alırsan (catch):
    3. subprocess.run(["pkexec", "rm", path]) komutunu çalıştır .
    - *Teknik Not:* Pkexec cache süresi (varsayılan 5 dk) sayesinde kullanıcıya ikinci kez şifre sorulmaz .

🟧 **SPRINT 4: Veri Adaptasyonu ve Parsing**

Amaç: Nmap çıktısını (XML) okuyup tabloya basmak. En çok hata potansiyeli olan yer burasıdır.

📌 **Adım 4.1: Pydantic Veri Modeli (Kerem)**

Tüm sistemin konuşacağı ortak dil.

- **Dosya:** src/core/models.py

- **Model:** class StandardFinding(BaseModel)
    - tool: str
    - target: str (IP veya Hostname)
    - type: str ("open_port", "vuln" vb.)
    - detail: str ("80/tcp http Apache 2.4")

📌 **Adım 4.2: XML Repair (Kerem - Kritik Güvenlik Önlemi)**

Nmap yarıda kesilirse XML dosyası bozuk (kapanmamış tag) kalır.

- **Metod:** repair_xml(file_path)
    - Dosyanın son 100 baytını oku.
    - Eğer </nmaprun> ile bitmiyorsa, dosyaya manuel olarak </runstats></nmaprun> stringini append et .
    - Bu sayede iterparse patlamadan okuyabilir.

📌 **Adım 4.3: Nmap Adapter (Kerem)**

- **Dosya:** src/core/adapters/nmap_adapter.py
- **Kütüphane:** defusedxml.ElementTree (Güvenlik için zorunlu) .
- **Parsing:**
    - XML içindeki <host> nodelarını gez.
    - <port> ve <service> alt nodelarını al.
    - Her birini StandardFinding nesnesine çevirip listeye at.

📌 **Adım 4.4: UI Tablo Gösterimi (Yiğit)**

- **Dosya:** src/ui/results_view.py
- **Bileşen:** QTableView + QAbstractTableModel.
- **Veri:** Kerem'in döndüğü List[StandardFinding] listesini tabloya bağlayın. Sütunlar: Hedef | Port | Servis | Detay.

🟫 **SPRINT 5: Öneri Motoru (Recommendation Engine)**

Amaç: Bulunan sonuçlara göre "Sıradaki Adım"ı önermek.

📌 **Adım 5.1: Maskeleme Servisi (Kerem)**

Cloud'a veriyi göndermeden önce anonimleştirme.

- **Dosya:** src/ai/masking.py
- **Regex:**
    - IPv4 Regex: \b(?:\d{1,3}\.){3}\d{1,3}\b -> "[HOST_X]" ile değiştir .
    - Domain Regex: Kurumsal domainleri "[DOMAIN_Y]" ile değiştir.

📌 **Adım 5.2: Öneri Şeması (Kerem)**

- **Dosya:** src/ai/schemas.py (Ekleme)
- **Schema:** SuggestionSchema
    - related_finding_id: Hangi bulguya istinaden?
    - action_title: "Gobuster ile dizin tara"
    - suggested_command_template: "gobuster dir -u {target} ..." .

📌 **Adım 5.3: UI Öneri Paneli (Yiğit)**

- **Tasarım:** Tablonun sağına veya altına açılır bir panel ("Action Center").
- **İşlev:** AI'dan gelen JSON verisini kartlar (Card View) şeklinde göster.
- **Buton:** "Taslağı Hazırla" butonu -> Bu butona basınca komut Sprint 2'deki giriş kutusuna kopyalanır (otomatik çalıştırmak yok!) .

🟪 **SPRINT 6: Plugin Sistemi ve Final Build**

Amaç: Sistemi genişletilebilir kılmak ve Linux binary üretmek.

📌 **Adım 6.1: Plugin Arayüzü (Interface) (Yiğit)**

- **Dosya:** src/core/interfaces.py
- **Abstract Class:** class ToolPlugin(ABC)
- **Zorunlu Metodlar:**
    1. check_dependency() -> bool: Araç sistemde var mı?
    2. get_command_schema() -> dict: AI parametreleri
    3. parse_output(file_path) -> List[StandardFinding]
    4. mask_data(text) -> str: Gizlilik maskelemesi

📌 **Adım 6.2: Plugin Yükleyici (Plugin Loader) (Yiğit)**

- **Dosya:** src/core/plugin_manager.py
- **Teknoloji:** importlib kütüphanesi.
- **Mantık:**
    - plugins/ klasöründeki tüm .py dosyalarını listele.
    - Her birini modül olarak import et.
    - İçindeki sınıf ToolPlugin'den türetilmiş mi kontrol et.
    - check_dependency True dönerse listeye ekle, yoksa pasif yap .

📌 **Adım 6.3: Linux Build (Kerem)**

- **Ortam:** WSL2 veya Ubuntu Sanal Makine.
- **Komut:** pyinstaller --onefile --name sentinel-ai --windowed src/main.py .
- **Sonuç:** dist/sentinel-ai (Tek parça çalıştırılabilir Linux dosyası).

✅ **Son Kontrol: Başarı Kriterleri**

1. **Engine:** UI donmadan ping atabiliyor musunuz?
2. **AI:** "Tarama yap" dediğinizde Nmap komutu oluşuyor mu?
3. **Security:** Root gerektiren işlemde şifre sorulup, iptal edilince program çökmeden duruyor mu?
4. **Data:** Nmap bitince tablo doluyor mu?
5. **Build:** Çıkan tek dosyayı (sentinel-ai) temiz bir Linux'a atınca çalışıyor mu?