SENTINEL AI: Kapsamlı Teknik Denetim, Mimari Analiz ve Sprint 3 Uygulama Stratejisi Raporu

Yönetici Özeti ve Proje Vizyonu

Sentinel AI projesi, siber güvenlik operasyonlarının yürütülmesinde geleneksel komut satırı arayüzlerinin (CLI) sunduğu esnekliği, modern yapay zeka teknolojilerinin (AI) sağladığı analitik derinlik ve kullanıcı dostu grafik arayüzlerle (GUI) birleştirmeyi hedefleyen öncü bir girişimdir. Bu rapor, projenin mevcut gelişim sürecini "Detaylı Fazlandırılmış.pdf" [1] ve ilgili teknik dokümanlar [2] ışığında denetlemek ve projenin en kritik güvenlik bariyeri olan **Sprint 3: Güvenlik, Yetki ve Temizlik** aşaması için derinlemesine bir uygulama yol haritası sunmak amacıyla hazırlanmıştır. Projenin mimari temelleri, hibrit bir yapay zeka modeline (Local Llama + Cloud GPT), asenkron süreç yönetimine (PyQt6 QProcess) ve Linux tabanlı hedef platformların katı yetki hiyerarşisine tam uyum sağlama prensibine dayanmaktadır.

Geliştirme sürecinin şu ana kadar tamamlanan Sprint 0, 1 ve 2 aşamaları, sistemin iskeletini, sinir sistemini (süreç yönetimi) ve beynini (yapay zeka karar mekanizması) oluşturmuştur. Ancak, bir siber güvenlik aracının gerçek dünyadaki başarısı, işletim sistemi seviyesindeki yetki yönetimi (Privilege Escalation) ve veri güvenliği (Secure Data Handling) standartlarına ne kadar uyduğuna bağlıdır. Bu bağlamda, Yiğit Yücel'in sorumluluğunda olan Sprint 3, sadece bir kodlama görevi değil, aynı zamanda projenin güvenlik duruşunu belirleyen stratejik bir dönüm noktasıdır. Rapor, bu kritik aşamayı, modern Linux güvenlik mimarisi, Polkit mekanizmaları ve güvenli veri imhası prensipleri çerçevesinde ele alacak; potansiyel riskleri, mimari darboğazları ve çözüm önerilerini kapsamlı bir şekilde sunacaktır.

BÖLÜM I: Mimari Temeller ve Teknoloji Yığınının Derinlemesine Analizi

Sentinel AI'nın teknik başarısı, seçilen teknoloji yığınının ve mimari desenlerin birbirleriyle olan uyumuna dayanmaktadır. Bu bölümde, projenin neden bu spesifik teknolojiler üzerine inşa edildiği ve bu tercihlerin projenin uzun vadeli sürdürülebilirliği üzerindeki etkileri analiz edilecektir.

1.1. Grafik Arayüz Motoru Olarak PyQt6 Tercihinin Stratejik Önemi

Python ekosisteminde masaüstü uygulaması geliştirmek için Tkinter, Kivy, PyGObject gibi birçok alternatif bulunmasına rağmen, Sentinel AI projesinde PyQt6'nın (veya lisanslama

modeline göre PySide6) tercih edilmesi, projenin doğası gereği bir zorunluluktur. Siber güvenlik araçları, özellikle Nmap veya Gobuster gibi tarayıcılar, işletim sistemi seviyesinde yoğun giriş/çıkış (I/O) işlemleri gerçekleştiren ve çalışma süreleri milisaniyelerden saatlere kadar uzayabilen süreçlerdir.

Geleneksel Tkinter mimarisi, basit ve senkron bir olay döngüsü (Event Loop) üzerinde çalışır. Bu yapı, uzun süren bir işlem başlatıldığında arayüzün tamamen donmasına ve işletim sistemi tarafından "Yanıt Vermiyor" olarak işaretlenmesine neden olur. Threading modülü ile bu sorun aşılmaya çalışılsa da, Python'un Global Interpreter Lock (GIL) mekanizması ve Tkinter'ın "thread-safe" olmayan widget yapısı, geliştirme sürecini kaotik bir hale getirebilir.[4]

Buna karşın PyQt6, C++ ile yazılmış Qt framework'ünün gücünü Python'a taşır. Projenin kalbinde yer alan QProcess sınıfı, işletim sistemi süreçlerini ana uygulamadan bağımsız bir şekilde yönetmek için tasarlanmıştır. Bu yapı, Sentinel AI'nın kullanıcı arayüzünü dondurmadan arka planda birden fazla güvenlik taramasını aynı anda yürütebilmesine olanak tanır. Ayrıca, PyQt'nin Sinyal-Slot (Signal-Slot) mekanizması, projenin benimsediği Model-View-ViewModel (MVVM) mimarisinin uygulanmasını doğal bir şekilde destekler. Bu sayede, arka planda akan Nmap çıktısı (Model/ViewModel), arayüzdeki terminal ekranına (View) doğrudan ve güvenli bir şekilde bağlanabilir.

1.2. Hibrit Yapay Zeka Stratejisi: Gizlilik ve Performans Dengesi

Projenin en yenilikçi yönlerinden biri, Yerel (Local) ve Bulut (Cloud) yapay zeka modellerini bir arada kullanan hibrit mimarisidir.[1] Bu yaklaşım, siber güvenlik operasyonlarındaki iki temel kısıtı aynı anda çözmeyi hedefler: Veri Gizliliği ve Analitik Derinlik.

Yerel model olarak Docker konteynerinde çalışan Llama 3 (8b-instruct-q4_K_M), kullanıcının yerel ağındaki hassas verilerin dışarı çıkmasını engeller. Basit niyet sınıflandırması (Intent Classification), parametre normalizasyonu ve temel komut üretimi gibi düşük gecikme gerektiren işlemler bu katmanda çözülür. Llama 3'ün kuantize edilmiş (Quantized) versiyonunun seçilmesi, standart bir geliştirici dizüstü bilgisayarında bile kabul edilebilir performansla çalışmasını sağlar.

Öte yandan, karmaşık saldırı senaryolarının analizi, zincirleme mantık yürütme (Chain of Thought) ve yaratıcı çözüm önerileri için OpenAI GPT-4o-mini modeli devreye girer. Bu model, yerel modelin kapasitesini aşan durumlarda bir "danışman" rolü üstlenir. Ancak buradaki kritik güvenlik mekanizması, verinin buluta gönderilmeden önce maskelenmesidir (Sprint 5 kapsamında planlanmıştır). Bu ikili yapı, Sentinel AI'yı sadece bir komut çalıştırıcı olmaktan çıkarıp, operasyonel bir güvenlik asistanına dönüştürür.

BÖLÜM II: Mevcut Durum Denetimi (Sprint 0 - Sprint 2)

Projenin şu anki durumu, paylaşılan dokümanlar [1] ve kod yapısı planları üzerinden denetlenmiştir. Bu bölüm, tamamlanan sprintlerin belirlenen hedeflere ne kadar ulaştığını ve tespit edilen potansiyel riskleri içermektedir.

2.1. Sprint 0: Proje İskeleti ve Altyapı Denetimi

Sprint 0, projenin temellerinin atıldığı ve geliştirme ortamının standardize edildiği aşamadır. Yapılan incelemeler, planlanan yapının modern Python proje standartlarına uygun olduğunu göstermektedir.

Klasör Hiyerarşisi ve Modülerlik:
Proje kök dizini olan sentinel-root altında kurgulanan yapı, "Separation of Concerns" ilkesine sıkı sıkıya bağlıdır. Kaynak kodların /src altında toplanması, backend mantığının /core, kullanıcı arayüzünün /ui ve yapay zeka modüllerinin /ai altında izole edilmesi, kodun okunabilirliğini ve bakımını kolaylaştırmaktadır.3 Özellikle main.py dosyasının /src altında konumlandırılması ve tüm modüllerin buradan çağrılması, Python'un import sistemiyle ilgili potansiyel sorunları (circular import vb.) minimize etmektedir.
Docker Orkestrasyonu ve Servis İzolasyonu:
docker-compose.yml dosyasının analizi 3, servislerin mantıksal ayrımının doğru yapıldığını göstermektedir. llama-service ve api-service olarak iki ayrı konteynerin tanımlanması, mikroservis mimarisine geçiş için uygun bir zemin hazırlamaktadır. llama-service için ./models dizininin volume olarak bağlanması, geliştirme sürecinde her konteyner yeniden başlatıldığında büyük boyutlu model dosyalarının tekrar indirilmesini engelleyerek bant genişliği ve zaman tasarrufu sağlamaktadır. Bu konfigürasyon, projenin CI/CD süreçlerine entegrasyonunu da kolaylaştıracaktır.
Kütüphane Seçimleri ve Güvenlik:
requirements.txt dosyasındaki kütüphaneler 3, projenin ihtiyaçlarını karşılayacak şekilde seçilmiştir. Özellikle defusedxml kütüphanesinin varlığı dikkat çekicidir. Standart Python xml kütüphanesi, dışarıdan gelen ve güvenilmeyen XML verilerine karşı savunmasızdır (Billion Laughs saldırısı vb.). Nmap çıktılarının işlenmesi projenin temel fonksiyonlarından biri olduğu için, defusedxml kullanımı proaktif bir güvenlik önlemi olarak değerlendirilmiştir. Ayrıca, pydantic kullanımı, veri doğrulama yükünü geliştiricinin omuzlarından alarak deklaratif bir yapı sunmaktadır.
Eksiklik ve Risk Uyarısı:
Dokümanlarda .env dosyasının yönetimiyle ilgili net bir strateji belirtilmiş olsa da (.env.example), yerel geliştirme ortamındaki .env dosyasının versiyon kontrol sistemine (Git) yanlışlıkla gönderilmemesi için .gitignore dosyasının içeriği kritik önem taşır. OpenAI API anahtarlarının sızması, projede ciddi maliyet ve güvenlik riskleri oluşturabilir.

2.2. Sprint 1: Akıllı Süreç Motoru (The Engine) Denetimi

Sprint 1, projenin "kas gücünü" oluşturan süreç yönetim motorunun geliştirilmesine odaklanmıştır. Bu aşamadaki çalışmalar, uygulamanın stabilitesini doğrudan etkilemektedir.

QProcess ve Asenkron Mimari:

AdvancedProcessManager sınıfının QProcess tabanlı olması, GUI'nin tepkiselliğini korumak için hayati önem taşır.3 Ancak, burada dikkat edilmesi gereken en önemli teknik detay, QProcess nesnesinin yaşam döngüsüdür. Eğer QProcess nesresi bir metodun içinde yerel değişken olarak tanımlanırsa, metodun çalışması bittiğinde Python'un çöp toplayıcısı (Garbage Collector) nesneyi bellekten silebilir ve süreç beklenmedik bir şekilde sonlanabilir. Dokümanlarda self._process olarak sınıf üyesi yapılması gerektiği belirtilmiştir, bu kurala uyulması kritik önem taşır.
Karakter Kodlaması ve Stabilite:
Nmap ve diğer güvenlik araçları, tarama sırasında veya hata durumlarında standart dışı karakterler veya farklı encoding formatları üretebilir. Dokümanda belirtilen data.data().decode('utf-8', errors='replace') stratejisi 1, bu tür durumlarda uygulamanın çökmesini (Crash) önleyen sağlam bir yaklaşımdır. errors='replace' parametresi, çözülemeyen bayt dizileri yerine ? karakteri koyarak akışın devam etmesini sağlar. Bu, siber güvenlik araçlarında sıkça karşılaşılan binary veri sızıntılarına karşı etkili bir çözümdür.
İnteraktif Girdi Yönetimi:
write_input metodu, terminal tabanlı araçların kullanıcı etkileşimi gerektirdiği durumlarda (Örn: "Devam etmek için") arayüzden yanıt gönderilmesini sağlar. Burada gönderilen verinin sonuna \n (New Line) karakterinin eklenmesi, terminalin girdiyi işlemesi (Enter tuşuna basılmış gibi) için teknik bir zorunluluktur. Bu detayın atlanması, sürecin sonsuz döngüde beklemesine (Deadlock) neden olabilir.

2.3. Sprint 2: Hibrit AI Komut Motoru (The Brain) Denetimi

Sprint 2, projenin zeka katmanını oluşturur ve kullanıcı niyetini teknik komutlara dönüştürür.

Deterministik Çıktı ve JSON Şemaları:
Büyük Dil Modelleri (LLM), doğaları gereği olasılıksal çalışır ve "halüsinasyon" görme eğilimindedir. Bir güvenlik aracında yanlış komut üretilmesi (örneğin yanlış IP'ye saldırı veya sistem dosyalarının silinmesi) kabul edilemez. src/ai/schemas.py dosyasında tanımlanan Strict JSON şemaları 3, modelin çıktısını belirli bir kalıba zorlayarak bu riski minimize eder. OpenAI'nın response_format özelliğinin kullanılması, çıktının her zaman makine tarafından okunabilir olmasını garanti eder.
Shell Injection Önlemleri:
Siber güvenlik yazılımlarında en büyük risklerden biri, uygulamanın kendisinin bir saldırı vektörüne dönüşmesidir. AI tarafından üretilen komutların doğrudan bir kabuk (Shell) ortamında çalıştırılması (Python'da shell=True), komut enjeksiyonu saldırılarına kapı aralar. Sentinel AI projesinde, komut argümanlarının tek bir string yerine bir liste (List) olarak tutulması 5, bu riski ortadan kaldırır. QProcess veya subprocess, liste olarak verilen argümanları doğrudan işletim sistemi çekirdeğine (kernel) iletir ve kabuk yorumlamasını (shell expansion) atlar. Bu, güvenlik mimarisi açısından en doğru yaklaşımdır.

BÖLÜM III: Linux Yetki Yönetimi Teorisi ve Güvenlik

Paradigması

Sprint 3'ün uygulama detaylarına geçmeden önce, bu sprintin çözmeye çalıştığı problemin teorik altyapısını anlamak gerekmektedir. Linux işletim sistemlerinde güvenlik, katmanlı bir yetki modeli üzerine kuruludur.

3.1. "GUI'yi Root Olarak Çalıştırma" Yanılgısı

Geliştirme sürecinde karşılaşılan en yaygın hatalardan biri, root yetkisi gerektiren işlemler (örneğin Nmap ile SYN taraması veya OS tespiti) yapmak için tüm grafik arayüz uygulamasını sudo python main.py komutuyla başlatmaktır. Bu yaklaşım, güvenlik literatüründe "Anti-Pattern" olarak kabul edilir.[6]

Bir PyQt uygulamasını root olarak çalıştırdığınızda:

1. **Geniş Saldırı Yüzeyi:** Uygulamanın kullandığı tüm kütüphaneler (Qt, resim işleme kütüphaneleri, font işleyiciler vb.) root yetkisine sahip olur. Bu kütüphanelerdeki en ufak bir zafiyet, saldırganın tüm sistemi ele geçirmesine neden olabilir.
2. **Dosya Sahipliği Sorunları:** Uygulamanın oluşturduğu tüm yapılandırma dosyaları, loglar ve geçici dosyalar root kullanıcısına ait olur. Kullanıcı uygulamayı daha sonra normal yetkiyle açtığında bu dosyalara erişemez veya yazamaz, bu da uygulamanın bozulmasına yol açar.
3. **X11/Wayland İzolasyonu:** Modern Linux masaüstü ortamları (özellikle Wayland), güvenlik nedeniyle root yetkisine sahip uygulamaların grafik arayüz çizmesine kısıtlamalar getirmektedir.

3.2. Polkit ve Pkexec Mimarisi

Sentinel AI projesinde benimsenen çözüm, **Polkit (PolicyKit)** mekanizmasıdır. Polkit, Unix benzeri sistemlerde yetkilendirme süreçlerini yöneten, süreçler arası iletişim (IPC) tabanlı bir framework'tür. pkexec aracı, Polkit'in komut satırı arayüzüdür ve belirli bir komutun başka bir kullanıcı (varsayılan olarak root) yetkisiyle çalıştırılmasını sağlar.[7]

pkexec kullanmanın avantajları:

- **İnce Ayarlı Yetki (Granularity):** Sadece ihtiyaç duyulan alt süreç (Nmap) root olur, ana uygulama (GUI) normal kullanıcı olarak kalır.
- **Kullanıcı Deneyimi (UX):** İşletim sistemi, güvenli bir grafiksel parola penceresi (Authentication Agent) açar. Uygulama, kullanıcının parolasını asla görmez ve işlemez.
- **Oturum Yönetimi:** Polkit, yetkilendirmeyi belirli bir süre (varsayılan 5 dakika) önbellekte tutabilir, böylece kullanıcı arka arkaya yapılan işlemler için sürekli parola girmek zorunda kalmaz.

3.3. CVE-2021-4034 (PwnKit) ve Güvenlik Dersleri

2022 yılında pkexec aracında keşfedilen CVE-2021-4034 (PwnKit) zafiyeti, yerel bir kullanıcının

root yetkisine yükselmesine olanak tanımıştır.[8] Bu zafiyet, pkexec'in komut satırı argümanlarını işleme şeklindeki bir hatadan kaynaklanıyordu. Sentinel AI projesi, bu zafiyetten etkilenmemek için sistemdeki pkexec sürümünün güncel olmasına güvenmekle birlikte, uygulama seviyesinde de önlemler almaktadır. ProcessManager içinde komutların sıkı bir şekilde yapılandırılması ve pkexec'e geçirilen argümanların kontrol edilmesi, bu tür zafiyetlerin uygulama üzerinden tetiklenmesini zorlaştırır. Ayrıca, projenin sadece bilinen ve güvenli araçları (whitelist) çalıştırması ek bir güvenlik katmanı sağlar.

BÖLÜM IV: Sprint 3 Detaylı Uygulama Kılavuzu (Kullanıcı Sorumluluğu)

Bu bölüm, Yiğit Yücel'in sorumluluğundaki Sprint 3 çalışmalarının adım adım nasıl uygulanacağını, teknik detayları, kod örneklerini ve dikkat edilmesi gereken noktaları içermektedir.

Hedef: src/core/process_manager.py ve src/core/cleaner.py dosyalarını güncelleyerek güvenli yetki yükseltme ve temizlik mekanizmalarını entegre etmek.

Adım 1: Pkexec Wrapper ve ExecutionManager'ın Dönüşümü

Mevcut ProcessManager sınıfı, komutları doğrudan çalıştıran basit bir yapıdadır. Bunu, yetki kontrolü yapabilen akıllı bir "ExecutionManager"a dönüştürmemiz gerekmektedir.

Teknik Gereksinimler:

- Sistemde pkexec aracının varlığı dinamik olarak kontrol edilmelidir (shutil.which).
- start_process metodu, requires_root bayrağını kabul etmeli ve buna göre komut zincirini manipüle etmelidir.
- Çevresel değişkenlerin (Environment Variables) yönetimi sağlanmalıdır. pkexec, varsayılan olarak güvenlik amacıyla çevresel değişkenleri temizler.[10] Ancak Nmap gibi araçlar genellikle buna ihtiyaç duymaz. Özel durumlarda pkexec env VAR=VAL cmd yapısı kullanılabilir, ancak şimdilik standart kullanım yeterlidir.

Uygulama Mantığı (Pseudocode ve Açıklama):

Python

```
import shutil
from PyQt6.QtCore import QProcess
```

```
#... Sınıf Tanımı ve Init...

def start_process(self, command: str, args: list, requires_root: bool = False):
    """
    Komutu başlatır. requires_root True ise komutun başına pkexec ekler.
    """
    final_cmd = command
    final_args = args

    # 1. Yetki Yükseltme Kontrolü
    if requires_root:
        # pkexec sistemde yüklü mü?
        if shutil.which("pkexec") is None:
            # Hata sinyali gönder ve çık
            self.sig_output_stream.emit("HATA: 'pkexec' sistemde bulunamadı. Root işlemleri
yapılamaz.\n", "stderr")
            self.sig_process_finished.emit(127, "") # 127: Command not found standardı
            return

        # Komutu sarmala (Wrapping Strategy)
        # Orijinal: nmap -sS 192.168.1.1
        # Dönüşüm: pkexec nmap -sS 192.168.1.1
        # QProcess yapısında: Program="pkexec", Args=
        final_args = [command] + args
        final_cmd = "pkexec"

    # 2. QProcess Başlatma
    self._process.start(final_cmd, final_args)

    # 3. Loglama Başlat (Session Recording)
    #...
```

Bu kod yapısında dikkat edilmesi gereken en önemli nokta, final_args listesinin oluşturulma şeklidir. Python listelerinde + operatörü yeni bir liste oluşturur, bu da orijinal args listesinin bozulmamasını sağlar.

Adım 2: Yetki Hataları ve Kullanıcı Deneyimi Yönetimi

Polkit penceresi açıldığında kullanıcı "İptal" butonuna basabilir veya parolayı yanlış girebilir. Uygulamanın bu durumu zarif bir şekilde (Graceful Degradation) karşılaması gerekir. pkexec bu durumları standart çıkış kodları (Exit Codes) ile bildirir.[11]

Tablo 1: Polkit Çıkış Kodları ve Anlamları

Çıkış Kodu (Exit Code)	Anlamı	Uygulama Tepkisi
0	Başarılı	Sonuçları işle ve cleaner çalıştır.
126	Yetkilendirme Reddedildi (Dismissed)	Kullanıcıya "İşlem İptal Edildi" uyarısı göster.
127	Yetkilendirme Hatası / Komut Yok	"Hata: Yetki alınamadı veya araç eksik" uyarısı göster.
Diğer	Uygulama Hatası (Nmap hatası vb.)	Standart hata logunu göster.

Uygulama Mantığı:
QProcess'in finished sinyali (sig_process_finished olarak yayınlanmadan önce) bir ara katmanda (slot) işlenmelidir.

Python

```
def _on_process_finished(self, exit_code, exit_status):
    """
    Süreç bittiğinde tetiklenir. Polkit kodlarını kontrol eder.
    """
    if exit_code == 126:
        # Kullanıcı parolayı girmedi veya İptal'e bastı
        self.sig_output_stream.emit("UYARI: Yetki verilmedi. İşlem kullanıcı tarafından iptal edildi.\n",
"stderr")
        self.sig_auth_failed.emit() # UI tarafında pop-up veya bildirim için
    elif exit_code == 127:
        # Kritik hata
        self.sig_output_stream.emit("HATA: Yetkilendirme sistemi hatası veya araç bulunamadı.\n",
"stderr")
        self.sig_auth_failed.emit()
    else:
        # İşlem normal bitti (Başarılı veya uygulamanın kendi hatası)
```

```
        # Log dosyasını kapat ve sonucu UI'ya bildir
        pass
```

Adım 3: Secure Cleaner ve Veri İmhası (Garbage Collection)

Sprint 3'ün en teknik ve gözden kaçmaya müsait kısmı burasıdır. Root yetkisiyle çalışan bir araç dosya oluşturduğunda, dosyanın sahibi root:root olur. Örneğin:
pkexec nmap -oX /tmp/scan_results.xml...
Bu komut sonucunda /tmp/scan_results.xml dosyası oluşur. Uygulama (normal kullanıcı) bu dosyayı okuyabilir ancak silemez. Eğer silinmezse, /tmp dizini zamanla şişer ve hassas tarama verileri sistemde kalır.
Silme Stratejisi: rm vs shred vs secure-delete
Veri güvenliği denilince akla ilk gelen komut shred (dosyanın üzerine rastgele veri yazarak silme) olur.13 Ancak bu proje özelinde shred kullanımı önerilmemektedir:

1. **Donanım Kısıtları:** Modern SSD disklerdeki "Wear Leveling" algoritmaları, shred komutunun aynı fiziksel bloğun üzerine yazmasını engeller. Bu da shred'i SSD'lerde etkisiz kılar ve diskin ömrünü boşuna tüketir.[13]
2. **Dosya Sistemi Yapısı:** Geçici dosyalar genellikle /tmp dizininde tutulur. Modern Linux dağıtımlarında /tmp genellikle tmpfs (RAM Disk) olarak mount edilir. RAM üzerindeki veriyi shred ile silmek anlamsızdır çünkü RAM enerjisi kesildiğinde veri zaten yok olur.
3. **Performans:** Büyük XML dosyalarının üzerine defalarca yazmak (default 3 pass) işlem süresini uzatır.

Bu nedenlerle, projenin "Destructor" mekanizması olarak pkexec rm kullanması en doğru ve performanslı yaklaşımdır.

src/core/cleaner.py Uygulama Kodu:

Python

```
import os
import subprocess
import shutil

class SecureCleaner:
    @staticmethod
    def force_delete(file_path: str):
        """
        Dosyayı silmeyi dener. PermissionError alırsa pkexec ile siler.
        """
        if not os.path.exists(file_path):
```

```
        return

    try:
        # 1. Deneme: Nazikçe (normal yetkiyle) silmeyi dene
        os.remove(file_path)
        # Eğer dosya kullanıcıya aitse veya dizin yazılabilirse bu çalışır
    except PermissionError:
        # 2. Deneme: Erişim engellendi, ağır silahları (root) çağır
        if shutil.which("pkexec"):
            try:
                # Polkit Cache sayesinde (genellikle) tekrar şifre sorulmaz
                subprocess.run(
                    ["pkexec", "rm", file_path],
                    check=True,
                    stdout=subprocess.DEVNULL,
                    stderr=subprocess.DEVNULL
                )
            except subprocess.CalledProcessError:
                print(f"KRİTİK HATA: Dosya root yetkisiyle bile silinemedi: {file_path}")
        else:
            print("HATA: pkexec yok, dosya silinemiyor.")
    except Exception as e:
        print(f"Beklenmeyen silme hatası: {e}")
```

Bu sınıf, projenin "Garbage Collector"ı olarak çalışacaktır. Her tarama işlemi bittikten ve veri parse edildikten sonra (Sprint 4), bu metodun çağrılması zorunludur.

BÖLÜM V: Gelecek Entegrasyonlar ve Sonuç

Sprint 3'ün başarıyla tamamlanması, projenin en büyük teknik engeli olan "Linux Yetki Modeli"nin aşılmasını sağlayacaktır. Bu aşamadan sonra gelecek olan **Sprint 4: Veri Adaptasyonu ve Parsing**, tamamen bu altyapıya güvenecektir. Parser modülü, pkexec ile oluşturulmuş ve StandardFinding modellerine dönüştürülmeyi bekleyen XML dosyalarını okuyacak, ardından SecureCleaner bu dosyaları imha edecektir.

Eylem Planı Özeti (Checklist):

1. **Kodlama:** src/core/process_manager.py içinde pkexec sarmalama mantığını yazın.
2. **Hata Yönetimi:** Exit Code 126 ve 127 için sinyal mekanizmasını kurun.
3. **Temizlik:** src/core/cleaner.py sınıfını oluşturun ve entegre edin.
4. **Test:** Sanal makinede root gerektiren bir işlem başlatın, parolayı girin, işlemin bitmesini bekleyin ve ardından /tmp dizininde dosya kalmadığını doğrulayın. Ayrıca parolayı iptal

ederek uygulamanın çökmediğini teyit edin.

Sentinel AI, bu mimari tercihler ve güvenlik önlemleriyle, sadece bir araç olmanın ötesine geçerek, siber güvenlik uzmanlarının güvenebileceği, sağlam ve profesyonel bir platform olma yolunda ilerlemektedir. Yiğit Yücel'in bu sprintteki başarısı, sistemin güvenilirliğini (reliability) doğrudan belirleyecektir.

Alıntılanan çalışmalar

1. Detaylı Fazlandırılmış.pdf
2. FAZ 1: Gelişmiş Süreç Yönetimi ve İnteraktif Term..., https://drive.google.com/open?id=1_-FKJnQsdHTHJ7VROwrOvVc0yEgzVvvT3XxkqYZj2Tg
3. Detaylı Fazlandırılmış.pdf, https://drive.google.com/open?id=16stmugoT0qSWjTEj7DPnnjAHfcmXhkTd
4. Nmap Doğal Dil Arayüzü Geliştirme Yol Haritası, https://drive.google.com/open?id=1guTfspjKmDiLsojG4qASllrxNWmqz8vUIcgG8NtDzPM
5. Nmap Doğal Dil Arayüzü Geliştirme Raporu, https://drive.google.com/open?id=1mNeR70vR4GHWiBDo1Rejwp6A5v7yFdEgjUUm4zp7Rh0
6. Nmap Doğal Dil Arayüzü Geliştirme Raporu.docx, https://drive.google.com/open?id=1vDbRCsx7LuSxXhBZNwNIR3uoqtYZCDCT
7. pkexec: polkit Reference Manual - Freedesktop.org, erişim tarihi Ocak 9, 2026, https://polkit.pages.freedesktop.org/polkit/pkexec.1.html
8. Lab Walkthrough - Exploiting PwnKit (CVE-2021–4034) - INE, erişim tarihi Ocak 9, 2026, https://ine.com/blog/exploiting-pwnkit-cve-20214034
9. Exploiting PwnKit (CVE-2021-4034): Techniqu... - INE, erişim tarihi Ocak 9, 2026, https://ine.com/blog/exploiting-pwnkit-cve-2021-4034-techniques-and-defensive-measures
10. Environment variables missing when executing via pkexec, erişim tarihi Ocak 9, 2026, https://stackoverflow.com/questions/42960140/environment-variables-missing-when-executing-via-pkexec
11. pkexec - Freedesktop.org, erişim tarihi Ocak 9, 2026, https://www.freedesktop.org/software/polkit/docs/0.105/pkexec.1.html
12. scap-workbench-pkexec-oscap.sh - GitHub, erişim tarihi Ocak 9, 2026, https://github.com/OpenSCAP/scap-workbench/blob/v1-2/scap-workbench-pkexec-oscap.sh
13. rm, shred, blkdiscard, and hdparm Secure Erase Explained, erişim tarihi Ocak 9, 2026, https://dev.to/lovestaco/securely-deleting-data-on-linux-rm-shred-blkdiscard-and-hdparm-secure-erase-explained-3ofi
14. How to Securely Erase a Disk and File using the Linux shred ..., erişim tarihi Ocak 9, 2026,

https://www.freecodecamp.org/news/securely-erasing-a-disk-and-file-using-linux-command-shred/