**Presentación del Proyecto SentinelCore**

**1. Introducción**

**SentinelCore es una plataforma centralizada de herramientas de seguridad informática diseñada para identificar, analizar y mitigar vulnerabilidades en entornos web y cloud. Desarrollada como proyecto de fin de curso por un equipo de 4 estudiantes, combina una arquitectura modular innovadora con el concepto de “toquen de antesala” para prevenir amenazas antes del almacenamiento. Esta presentación detalla la viabilidad, arquitectura, plan de ejecución, mejoras propuestas y ventajas competitivas, enfocándose en pequeñas empresas y servicios técnicos.**

**Objetivo del Proyecto**

* **Centralizar herramientas de análisis de vulnerabilidades (OWASP ZAP, Wapiti, ClamAV).**
* **Implementar un sistema proactivo de análisis de archivos mediante “toquens de antesala”.**
* **Proporcionar mitigación guiada y automatizada con informes personalizados.**
* **Ofrecer una solución cloud-hosted accesible vía API y dashboard web, con integración de WhatsApp para interacción del usuario.**

**Público Objetivo**

* **Pequeñas empresas: Necesitan soluciones de seguridad accesibles, más allá de un simple antivirus.**
* **Servicios técnicos: Equipos de TI que requieren visibilidad y gestión simplificada de vulnerabilidades.**
* **Entornos educativos: Ideal para formación en ciberseguridad.**

**2. Viabilidad del Proyecto**

**Viabilidad Técnica**

* **Modularidad: Arquitectura basada en módulos independientes (Core, Toquen, Web Scanner, Database, GUI, Mitigation) implementados como contenedores Docker, facilitando desarrollo y escalabilidad.**
* **Integración: Uso de herramientas open-source (OWASP ZAP, Wapiti, ClamAV) con APIs gratuitas o CLI, reduciendo la complejidad para estudiantes.**
* **Innovación: El “toquen de antesala” intercepta archivos antes del almacenamiento, una capa proactiva no común en soluciones tradicionales.**
* **Cloud-Hosted: La aplicación se ejecuta en un servidor central, accesible vía API REST y dashboard web, eliminando la necesidad de instalación local.**
* **Mejora: Contenerización total con Docker para todos los módulos, especialmente los toquens, asegurando aislamiento y portabilidad.**

**Viabilidad Organizativa**

* **Equipo de 4 personas: Tareas divididas por módulos, con roles claros (Líder Técnico, Desarrollador de Escáneres, Desarrollador de Interfaz, Especialista en Base de Datos/Mitigación).**
* **Metodología ágil: Scrum con sprints de 2 semanas, usando GitHub para control de versiones y Trello para gestión de tareas.**
* **Aprendizaje: Proyecto alineado con objetivos educativos, cubriendo integración de APIs, desarrollo seguro y automatización.**

**Viabilidad Económica**

* **Bajo costo: Uso de herramientas open-source y servicios cloud gratuitos (AWS Free Tier, Heroku).**
* **Limitaciones: APIs comerciales (Nessus, Qualys) se evitarán en el MVP para reducir costos.**
* **Mejora: Hospedaje en Netlify/Heroku para el dashboard y API, optimizando recursos.**

**Riesgos y Mitigaciones**

* **Riesgo: Complejidad en la integración de APIs.**
  + **Mitigación: Priorizar herramientas open-source con documentación clara (OWASP ZAP, Wapiti).**
* **Riesgo: Curva de aprendizaje para Docker y APIs.**
  + **Mitigación: Capacitación inicial de 2 semanas en Docker, Flask/FastAPI y APIs de herramientas.**
* **Riesgo: Seguridad de la plataforma.**
  + **Mitigación: Implementar autenticación (HTTP Basic Auth, API keys) y validación de entradas.**

**3. Arquitectura Modular**

**Módulos Funcionales**

1. **Core Module:**
   * **Propósito: Orquesta la comunicación entre módulos y gestiona configuraciones.**
   * **Componentes: main.py, dispatcher.py, config/.**
   * **Tecnologías: Python 3.8+, asyncio, Docker.**
2. **Toquen Module:**
   * **Propósito: Implementa el “toquen de antesala” y toquens especializados para análisis de archivos.**
   * **Componentes: antesala.py, static\_scan.py, pdf\_sanitizer.py.**
   * **Tecnologías: libmagic, ClamAV, asyncio.Queue, Docker.**
3. **Web Scanner Module:**
   * **Propósito: Integra herramientas de escaneo web (OWASP ZAP, Wapiti).**
   * **Componentes: zap\_wrapper.py, results\_parser.py.**
   * **Tecnologías: requests, OWASP ZAP API, Wapiti.**
4. **Database Module:**
   * **Propósito: Almacena resultados y recomendaciones de mitigación.**
   * **Componentes: models.py.**
   * **Tecnologías: SQLite, sqlite3.**
5. **GUI Module:**
   * **Propósito: Proporciona un dashboard web y soporte para interacción vía WhatsApp.**
   * **Componentes: dashboard.py, whatsapp\_api.py.**
   * **Tecnologías: Flask/FastAPI, WhatsApp Business API, passlib.**
6. **Mitigation Module:**
   * **Propósito: Genera recomendaciones y aplica mitigaciones automáticas.**
   * **Componentes: Integrado en results\_parser.py, models.py.**
   * **Tecnologías: reportlab, OWASP/MITRE ATT&CK.**

**Estructura de Carpetas (en evolución)**

**SentinelCore/**

**├── core/**

**│ ├── main.py**

**│ ├── dispatcher.py**

**│ └── config/**

**│ ├── settings.yaml**

**│ └── api\_keys.json**

**├── toquens/**

**│ ├── antesala.py**

**│ ├── static\_scan.py**

**│ ├── pdf\_sanitizer.py**

**│ └── Dockerfile**

**├── webscanner/**

**│ ├── zap\_wrapper.py**

**│ ├── results\_parser.py**

**│ └── Dockerfile**

**├── database/**

**│ ├── models.py**

**│ └── Dockerfile**

**├── gui/**

**│ ├── dashboard.py**

**│ ├── whatsapp\_api.py**

**│ └── Dockerfile**

**├── mitigation/**

**│ ├── mitigation\_engine.py**

**│ └── Dockerfile**

**├── docs/**

**│ ├── README.md**

**│ ├── architecture.md**

**│ └── user\_guide.md**

**├── tests/**

**│ ├── test\_core.py**

**│ ├── test\_toquens.py**

**│ ├── test\_webscanner.py**

**│ ├── test\_database.py**

**│ └── test\_gui.py**

**├── Dockerfile**

**└── docker-compose.yml**

* **Nota: La estructura está en evolución. Futuras iteraciones podrían incluir subcarpetas para nuevos toquens (toquens/image\_analyzer.py) o integraciones cloud (cloud\_connectors/aws.py).**

**Diagrama de Flujo (en evolución)**

**graph TD**

**A[Usuario] -->|Dashboard Web| B[GUI Module]**

**A -->|WhatsApp API| B**

**B -->|REST API| C[Core Module]**

**C -->|Dispatch Task| D[Toquen Module]**

**C -->|Dispatch Task| E[Web Scanner Module]**

**D -->|File Analysis| F[Toquen Antesala]**

**F -->|Delegate| G[Toquens Especializados]**

**E -->|Scan Results| H[Database Module]**

**H -->|Mitigation Data| I[Mitigation Module]**

**I -->|Reports| B**

**I -->|External Services| J[Partner APIs]**

**C -->|Service Mapping| K[Service Manager]**

* **Nota: El diagrama está en evolución. Futuras versiones incluirán conectores para servicios cloud (AWS, Azure) y más toquens especializados.**

**Interfaces y Protocolos de Comunicación**

* **Intra-Module:**
  + **Protocolo: Python function calls, asyncio.Queue (or Redis for scalability).**
  + **Ejemplo: antesala.py envía { "filename": "doc.pdf", "filetype": "pdf" } a pdf\_sanitizer.py via queue.**
* **Inter-Module:**
  + **Protocolo: REST API (Flask/FastAPI) con JSON payloads.**
  + **Ejemplo: POST /scan con { "url": "http://example.com" }.**
  + **WhatsApp API: Usar WhatsApp Business API para comandos (/start\_scan, /view\_results) y notificaciones.**
* **External Services:**
  + **Protocolo: HTTP APIs para herramientas (OWASP ZAP, ClamAV) y partner APIs (e.g., cloud scanners).**
  + **Security: HTTPS, API key authentication, input sanitization.**
* **Mejora: Implementar OpenAPI/Swagger para documentar la API REST y Redis para comunicación robusta.**

**4. Mejoras Propuestas**

**Contenerización con Docker**

* **Implementación: Cada módulo (Core, Toquens, Web Scanner, Database, GUI, Mitigation) se ejecuta en un contenedor Docker independiente, definido en docker-compose.yml.**
* **Beneficio: Aislamiento, portabilidad, y fácil despliegue en entornos cloud.**
* **Ejemplo:**
* **version: '3.8'**
* **services:**
* **core:**
* **build: ./core**
* **depends\_on:**
* **- toquens**
* **- webscanner**
* **toquens:**
* **build: ./toquens**
* **webscanner:**
* **build: ./webscanner**
* **database:**
* **build: ./database**
* **gui:**
* **build: ./gui**
* **ports:**
* **- "5000:5000"**

**Integración con WhatsApp API**

* **Implementación: Usar WhatsApp Business API para:**
  + **Notificar vulnerabilidades detectadas ("Se encontró XSS en http://example.com").**
  + **Ejecutar comandos (/start\_scan http://example.com, /generate\_report scan123).**
* **Componente: whatsapp\_api.py en el GUI Module, usando una biblioteca como python-whatsapp o la API oficial.**
* **Beneficio: Accesibilidad para usuarios no técnicos, ideal para servicios técnicos.**

**Cloud-Hosted API y Dashboard**

* **Implementación: La aplicación se aloja en un servidor cloud (Heroku, AWS), accesible vía API REST y un dashboard web (Flask/FastAPI).**
* **Ventaja Competitiva: No requiere instalación local, facilitando el uso por clientes pequeños y servicios técnicos.**
* **Dashboard Features:**
  + **Visualización de resultados en tiempo real.**
  + **Activación de servicios (escaneos, mitigaciones) desde el dashboard.**
  + **Informes generados por consultores/analistas.**

**Service Mapping y Gestión Externa**

* **Implementación: El Core Module incluye un service\_manager.py para mapear servicios gestionados (web, cloud, endpoints) y activar/desactivar servicios de partners (e.g., AWS Security Hub, VirusTotal).**
* **Beneficio: Centraliza la gestión de servicios externos, manteniendo la herramienta actualizada con soluciones de terceros.**
* **Ejemplo: Activar un escaneo VirusTotal desde el dashboard y mostrar resultados en la misma interfaz.**

**Enfoque en Pequeñas Empresas**

* **Adaptación: Simplificar la interfaz y documentación para usuarios no expertos.**
* **Educación: Incluir guías interactivas y simulaciones de vulnerabilidades en el dashboard.**
* **Soporte Técnico: Permitir que servicios técnicos externos (del grupo) gestionen la herramienta vía dashboard o WhatsApp.**

**5. Plan de Ejecución**

**Equipo y Roles**

* **Líder Técnico (Estudiante A): Core Module, Toquen Module (antesala.py), Service Manager.**
* **Desarrollador de Escáneres (Estudiante B): Web Scanner Module, Toquen Module (static\_scan.py, pdf\_sanitizer.py).**
* **Desarrollador de Interfaz (Estudiante C): GUI Module (dashboard.py, whatsapp\_api.py).**
* **Especialista en Base de Datos/Mitigación (Estudiante D): Database Module, Mitigation Module.**

**Cronograma (6 meses)**

* **Mes 1: Planificación, capacitación (Docker, WhatsApp API, Flask), diseño de arquitectura.**
* **Mes 2-3: Desarrollo de Core, Toquens, Web Scanner (Dockerized).**
* **Mes 4: Implementación de Database, GUI, WhatsApp integration.**
* **Mes 5: Mitigation Module, service mapping, testing.**
* **Mes 6: Pruebas finales, documentación, presentación.**

**Metodología**

* **Scrum: Sprints de 2 semanas, reuniones diarias de 15 minutos.**
* **Herramientas: GitHub (control de versiones), Trello (gestión), Discord (comunicación), pytest (testing).**

**6. Checklist de Tareas**

**Planificación**

* **Definir arquitectura modular y servicios gestionados.**
* **Investigar WhatsApp Business API y Flask/FastAPI.**
* **Configurar repositorio GitHub y Docker Compose.**

**Desarrollo**

* **Implementar Core Module (main.py, dispatcher.py, service\_manager.py).**
* **Desarrollar Toquen Module (antesala.py, static\_scan.py, pdf\_sanitizer.py) en contenedores Docker.**
* **Integrar Web Scanner Module con OWASP ZAP y Wapiti.**
* **Configurar Database Module con SQLite.**
* **Crear GUI Module (dashboard web, WhatsApp API integration).**
* **Implementar Mitigation Module con reportes HTML/PDF.**

**Pruebas**

* **Pruebas unitarias (pytest) para cada módulo.**
* **Pruebas de integración (Core-Toquen-Web Scanner).**
* **Simulación de ataques (XSS, SQLi) para validar detección.**
* **Pruebas de usabilidad (dashboard, WhatsApp).**

**Documentación**

* **Manual de usuario con guías educativas.**
* **Documentación técnica (arquitectura, APIs).**
* **Registro de mejoras y problemas.**

**7. Ventajas Competitivas**

* **Centralización: Consolida herramientas y servicios en un solo dashboard.**
* **Proactividad: “Toquen de antesala” previene amenazas antes del almacenamiento.**
* **Accesibilidad: API cloud-hosted y WhatsApp integration para usuarios no técnicos.**
* **Extensibilidad: Gestión de servicios externos y colaboración con partners.**
* **Educativo: Ideal para pequeñas empresas y formación en ciberseguridad.**

**8. Conclusión**

**SentinelCore es una solución innovadora que combina modularidad, automatización y accesibilidad para pequeñas empresas y servicios técnicos. La contenerización con Docker, la integración con WhatsApp, y el enfoque en gestión de servicios externos lo posicionan como una herramienta versátil y escalable. Este proyecto no solo cumple los objetivos académicos, sino que ofrece un valor real en el mercado de ciberseguridad.**

**Explicaciones Adicionales (Fuera del Artefacto)**

* **Docker: Cada módulo se ejecuta en un contador para garantizar el aislamiento y portabilidad, con docker-compose.yml definido dependencias.**
* **API de WhatsApp: La integración permite a los usuarios recibir alertas y ejecutar comandos, teniendo en cuenta la accesibilidad para servicios técnicos.**
* **Hospedado en la nube: Alojar la aplicación en un servicio central elimina la necesidad de instalación local, ideal para clientes pequeños.**
* **Mapeo de servicios: El service\_manager.py mapea servicios gestionados (web, nube) y permitir activar/desactivar servicios de socios, manteniendo la herramienta actualizada.**
* **Evolución: La estructura de alfombras y el diagrama de flujo son iterativos, con planos para unadir más toquens y conectores nube en el futuro.**