**Informe de Viabilidad, Plan de Ejecución y Checklist para el Proyecto SentinelCore**

**Informe de Viabilidad**

**Factibilidad Técnica**

El proyecto **SentinelCore** es técnicamente viable para un equipo de estudiantes de seguridad informática trabajando en un proyecto de fin de curso, considerando las siguientes razones:

* **Tecnologías accesibles**:
  + Las herramientas propuestas (OWASP ZAP, Nikto, Wapiti, Nmap, etc.) son de código abierto o tienen APIs gratuitas/limitadas (como VirusTotal). SQLite es adecuado para un MVP por su simplicidad.
  + Python, utilizado en la arquitectura modular, es un lenguaje accesible para estudiantes con conocimientos básicos de programación.
  + La arquitectura modular basada en "toquens" permite dividir el desarrollo en componentes independientes, facilitando la implementación por un equipo pequeño.
* **Innovación del "toquen de antesala"**:
  + La idea de interceptar y analizar archivos antes del almacenamiento es innovadora y factible mediante scripts en Python que inspeccionen flujos de datos (por ejemplo, usando bibliotecas como mimetypes o libmagic para identificar tipos de archivo).
  + La colaboración entre "toquens" puede implementarse con un sistema de colas (como multiprocessing.Queue o asyncio) para la comunicación entre módulos.
* **Limitaciones técnicas**:
  + Integrar múltiples herramientas de escaneo (Nessus, Qualys) puede ser complejo debido a restricciones de licencias o APIs. Se recomienda centrarse en herramientas de código abierto para el MVP (OWASP ZAP, Wapiti, ClamAV).
  + La interfaz web/CLI debe ser simple para ajustarse a las capacidades del equipo. Frameworks como Flask o FastAPI son ideales para una interfaz web básica.
  + La escalabilidad de la arquitectura "toquen" dependerá de una correcta gestión de recursos, lo que puede requerir pruebas extensivas.
* **Mejora técnica propuesta**:
  + **Contenerización**: Usar Docker para encapsular cada "toquen" como un microservicio independiente, facilitando la instalación y pruebas en entornos heterogéneos.
  + **Integración con GitHub Actions**: Automatizar pruebas unitarias y de integración para garantizar la estabilidad del sistema.
  + **Uso de Redis o RabbitMQ**: Para la comunicación entre "toquens", en lugar de colas básicas de Python, para mejorar la robustez y escalabilidad.

**Factibilidad Operativa**

* **Equipo de 4 personas**: Un equipo pequeño puede manejar el desarrollo del MVP si se divide el trabajo en módulos específicos (core, toquens, escáneres, interfaz). Sin embargo, la falta de experiencia previa en integración de APIs o desarrollo de sistemas modulares puede requerir una curva de aprendizaje.
* **Tiempo**: Sin límite de tiempo, el proyecto es viable, pero se recomienda un cronograma de 4-6 meses para un MVP funcional, considerando las responsabilidades académicas de los estudiantes.
* **Mejora operativa propuesta**:
  + **Metodología ágil**: Usar Scrum con sprints de 2 semanas para dividir el trabajo y mantener el progreso.
  + **Documentación continua**: Usar herramientas como Notion o Confluence para documentar avances, decisiones técnicas y problemas encontrados.
  + **Capacitación inicial**: Dedicar las primeras 2 semanas a aprender sobre las APIs de OWASP ZAP, Wapiti y SQLite, así como conceptos de microservicios.

**Factibilidad Económica**

* **Costos bajos**:
  + Herramientas de código abierto y APIs gratuitas (como VirusTotal con límites) reducen los costos.
  + Infraestructura de desarrollo: Computadoras personales y servidores locales o en la nube (como AWS Free Tier o Google Cloud para pruebas).
* **Limitaciones económicas**:
  + APIs comerciales (Nessus, Qualys) pueden no ser accesibles para estudiantes. Se recomienda evitarlas en el MVP.
* **Mejora económica propuesta**:
  + Usar servicios gratuitos como GitHub para control de versiones y Netlify/Heroku para hospedar la interfaz web del MVP.
  + Explorar becas o patrocinios académicos para acceder a APIs premium si se desea incluirlas en una versión futura.

**Factibilidad de Mercado**

* **Relevancia**: La centralización de herramientas de seguridad y la mitigación proactiva son necesidades crecientes en empresas con entornos híbridos o multinube.
* **Competencia**: Nessus, OpenVAS y Qualys dominan el mercado, pero carecen de la modularidad del "toquen de antesala" y de mitigación automatizada personalizada.
* **Valor educativo**: El enfoque educativo del proyecto lo hace ideal para entornos académicos y pequeñas empresas que buscan soluciones accesibles.
* **Mejora de mercado propuesta**:
  + **Enfoque en gamificación**: Incluir un módulo de formación interactivo que simule ataques y enseñe a mitigarlos, atractivo para entornos educativos.
  + **Certificaciones**: Alinear el proyecto con estándares como OWASP Top 10 o MITRE ATT&CK para aumentar su credibilidad.

**Riesgos y Mitigaciones**

* **Riesgo**: Complejidad de integración de múltiples herramientas.
  + **Mitigación**: Priorizar herramientas de código abierto con buena documentación (OWASP ZAP, Wapiti) y usar wrappers en Python para simplificar la integración.
* **Riesgo**: Curva de aprendizaje para estudiantes sin experiencia en microservicios o APIs.
  + **Mitigación**: Dividir el proyecto en tareas claras y asignar roles específicos (ver plan de ejecución).
* **Riesgo**: Escalabilidad limitada en el MVP.
  + **Mitigación**: Diseñar la arquitectura con contenedores desde el inicio para facilitar la escalabilidad futura.

**Conclusión de Viabilidad**

El proyecto es **viable** para un equipo de 4 estudiantes, siempre que se enfoque en un MVP con herramientas de código abierto, una arquitectura modular simple y una interfaz básica. Las mejoras propuestas (Docker, metodología ágil, gamificación) aumentarán la calidad del producto y su atractivo educativo.

**Plan de Ejecución**

**Equipo**

* **4 personas**:
  + **Líder Técnico (Desarrollador Core)**: Responsable de la arquitectura, el "toquen de antesala" y la integración de módulos.
  + **Desarrollador de Escáneres**: Enfocado en integrar herramientas como OWASP ZAP y Wapiti.
  + **Desarrollador de Interfaz**: Encargado de la CLI o interfaz web (Flask/FastAPI).
  + **Especialista en Base de Datos y Mitigación**: Gestiona la base de datos (SQLite) y las sugerencias de mitigación.

**Cronograma (6 meses, sin límite de tiempo explícito)**

* **Mes 1: Planificación y Capacitación**
  + Semana 1-2: Investigación y aprendizaje (OWASP ZAP, Wapiti, SQLite, Python asyncio).
  + Semana 3-4: Definición de la arquitectura modular, diseño de la base de datos y prototipo del "toquen de antesala".
* **Mes 2-3: Desarrollo del Core y Toquens**
  + Implementación del "toquen de antesala" (inspección de archivos y redirección).
  + Desarrollo de "toquens especializados" (escaneo estático, sanitización de PDFs).
  + Integración inicial con OWASP ZAP y Wapiti.
* **Mes 4: Integración y Base de Datos**
  + Configuración de SQLite para almacenar resultados.
  + Desarrollo de scripts para generar sugerencias de mitigación basadas en OWASP/MITRE ATT&CK.
  + Pruebas unitarias para cada módulo.
* **Mes 5: Interfaz y Automatización**
  + Desarrollo de la interfaz (CLI o web con Flask).
  + Implementación de automatización para escaneos continuos.
  + Pruebas de integración entre módulos.
* **Mes 6: Pruebas, Documentación y Entrega**
  + Pruebas de usabilidad y seguridad.
  + Documentación técnica y de usuario.
  + Presentación del MVP y demostración.

**Metodología**

* **Scrum**: Sprints de 2 semanas con reuniones diarias de 15 minutos.
* **Herramientas**:
  + **Control de versiones**: GitHub.
  + **Gestión de tareas**: Trello o Notion.
  + **Comunicación**: Discord o Slack.
  + **Pruebas**: pytest para Python, GitHub Actions para CI/CD.

**Mejoras Propuestas**

* **Contenerización**: Usar Docker para cada "toquen" y el core, facilitando la portabilidad.
* **Autenticación para subdominios**: Implementar un módulo de autenticación básica para proteger la interfaz web, siguiendo las recomendaciones del documento.
* **Gamificación**: Añadir un modo educativo con simulaciones de vulnerabilidades para aprendizaje interactivo.
* **Soporte multi-idioma**: Incluir documentación y sugerencias de mitigación en inglés y español para mayor alcance.

**Checklist de Trabajos a Realizar**

**1. Planificación y Diseño**

* Investigar APIs y documentación de OWASP ZAP, Wapiti, Nmap y ClamAV.
* Definir la arquitectura modular (core, toquens, escáneres, interfaz, base de datos).
* Diseñar el esquema de la base de datos (SQLite) para almacenar resultados y sugerencias.
* Crear un diagrama de flujo detallado para el "toquen de antesala" y su interacción con "toquens especializados".
* Configurar el repositorio en GitHub y establecer GitHub Actions para CI/CD.

**2. Desarrollo del Core**

* Implementar main.py y dispatcher.py para coordinar módulos.
* Desarrollar el "toquen de antesala" (antesala.py) con:
  + Identificación de tipos de archivo (mimetypes o libmagic).
  + Redirección a "toquens especializados" usando colas (asyncio o Redis).
  + Análisis de inocuidad básico (integración con ClamAV).
* Crear al menos dos "toquens especializados" (static\_scan.py, pdf\_sanitizer.py).

**3. Integración de Escáneres**

* Desarrollar wrappers para OWASP ZAP (zap\_wrapper.py) y Wapiti.
* Configurar escaneos automáticos y parseo de resultados (results\_parser.py).
* Integrar ClamAV para análisis de malware en archivos.

**4. Base de Datos**

* Implementar modelos de datos en models.py para almacenar:
  + Resultados de escaneos.
  + Sugerencias de mitigación (basadas en OWASP/MITRE ATT&CK).
* Configurar SQLite como base de datos ligera.

**5. Interfaz**

* Desarrollar una CLI básica para interactuar con el sistema.
* (Opcional) Implementar una interfaz web con Flask/FastAPI (simple\_web\_gui.py).
* Proteger la interfaz con autenticación básica (.htpasswd o similar).

**6. Mitigación**

* Generar sugerencias de mitigación personalizadas basadas en resultados de escaneo.
* Implementar scripts de mitigación automática para vulnerabilidades simples (con autorización del usuario).
* Exportar informes en HTML/PDF.

**7. Pruebas**

* Escribir pruebas unitarias para cada módulo con pytest.
* Realizar pruebas de integración entre "toquens", escáneres y la base de datos.
* Probar la interfaz (CLI/web) para usabilidad.
* Simular ataques básicos (inyección SQL, XSS) para validar la detección.

**8. Documentación**

* Redactar documentación técnica (arquitectura, instalación, uso).
* Crear una guía de usuario con ejemplos de mitigación.
* Incluir un apartado educativo con explicaciones de vulnerabilidades.

**9. Mejoras Adicionales**

* Contenerizar el proyecto con Docker para facilitar la instalación.
* Implementar un módulo de gamificación para formación interactiva.
* Añadir soporte multi-idioma para la documentación y sugerencias.

**Conclusión**

El proyecto **SentinelCore** es una propuesta ambiciosa pero factible para un equipo de 4 estudiantes, especialmente con las mejoras propuestas (Docker, metodología ágil, gamificación). La arquitectura modular y el enfoque en mitigación proactiva lo hacen innovador y relevante. El cronograma de 6 meses y el checklist aseguran un desarrollo estructurado, mientras que las mejoras propuestas aumentan el valor educativo y técnico del producto.