**📄 Plantilla: Informe de Verificación de Seguridad de Software**

**🎯 Propósito**

Esta plantilla tiene como propósito brindar una herramienta ágil y confiable para documentar y gestionar las actividades de verificación de seguridad en proyectos de desarrollo de software.

Está diseñada para ser utilizada en proyectos reales, permitiendo a los equipos aplicar los mecanismos de verificación, utilizar herramientas de autoevaluación y refinar sus estrategias de pruebas de seguridad antes de la liberación del software.

**🚀 Guía de Uso**

1. **Rellena las Secciones:** Completa cada sección con la información específica de tu proyecto. Los corchetes **[ ]** indican dónde debes insertar tu contenido.
2. **Sé Detallado:** Aunque es una plantilla ágil, la calidad de la información que ingreses es clave para su utilidad. Sé específico en tus hallazgos, contramedidas y planes.
3. **Adapta según Necesidad:** Si una sección no aplica o necesitas añadir más detalle, siéntete libre de modificar la plantilla.

**📝 Información General del Proyecto**

* **Nombre del Proyecto:** [Ej: SecureConnect - Plataforma de Colaboración Segura]
* **Fecha del Informe:** [Ej: 02 de Julio de 2025]
* **Versión del Software / Módulo Analizado:** [Ej: v1.0.0 - Módulo de Intercambio de Documentos]
* **Equipo Responsable:** [Ej: Equipo Alfa - Ingeniería de Seguridad]
* **Metodología de Desarrollo:** [Ej: Ágil (Scrum/Kanban)]

**🛠️ Fase 1: Diseño y Requisitos Seguros - Modelado de Amenazas (Threat Modeling)**

**1.1. Componentes Analizados 🧠**

* [Describe los componentes clave analizados en esta fase. Ej: Módulo de subida de documentos, Base de datos de usuarios, API de cifrado.]

**1.2. Amenazas Identificadas (STRIDE) y Contramedidas Propuestas 🛡️**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Amenaza (STRIDE) | Descripción de la Amenaza | Componente/Flujo Afectado | Impacto Potencial | Contramedidas Propuestas |
| Spoofing | [Ej: Suplantación de identidad de usuario al cargar un documento.] | [Ej: Módulo de autenticación.] | [Ej: Acceso no autorizado a documentos.] | [Ej: Implementar MFA, verificar tokens de sesión.] |
| Tampering | [Ej: Modificación de metadatos de un documento en tránsito.] | [Ej: Flujo de carga/descarga de documentos.] | [Ej: Integridad de datos comprometida.] | [Ej: Usar firmas digitales, cifrado con autenticación.] |
| Repudiation | [Ej: Usuario niega haber subido un documento específico.] | [Ej: Módulo de auditoría de acciones.] | [Ej: Falta de trazabilidad, problemas legales.] | [Ej: Logs inmutables, firmas digitales.] |
| Information Disclosure | [Ej: Exposición de claves de cifrado en logs.] | [Ej: Sistema de logging.] | [Ej: Confidencialidad de datos comprometida.] | [Ej: Redacción de información sensible en logs, gestión segura de claves.] |
| Denial of Service | [Ej: Ataque de inundación en la API de subida de documentos.] | [Ej: API de subida de documentos.] | [Ej: Indisponibilidad del servicio.] | [Ej: Rate limiting, WAF, balanceo de carga.] |
| Elevation of Privilege | [Ej: Usuario normal obtiene permisos de administrador.] | [Ej: Módulo de control de acceso.] | [Ej: Control total del sistema por un atacante.] | [Ej: Principio de menor privilegio, validación estricta de roles.] |
| [Añadir más si es necesario] |  |  |  |  |

**1.3. Diagrama Conceptual (Descripción/Enlace) 🧩**

* [Describe brevemente el diagrama conceptual (DFD, arquitectura) utilizado para el modelado de amenazas. Si es un archivo externo, proporciona un enlace o referencia. Ej: "Diagrama de flujo de datos del módulo de documentos, mostrando actores, procesos y almacenes de datos, con puntos de amenaza marcados."]

**👨‍💻 Fase 2: Codificación Segura y Análisis Estático/Composición**

**2.1. Fragmento de Código (Vulnerable y Seguro) 💻**

* **Descripción de la Funcionalidad:** [Ej: Función de procesamiento de entrada de usuario para el nombre de un documento.]
* **Código Vulnerable:**

# Código Python vulnerable (ej. sin saneamiento de entrada)  
**def** **process\_document\_name\_vulnerable**(name):  
 **return** f"Documento: {name}"

* **Vulnerabilidad Esperada (SAST):** [Ej: Inyección de código (XSS si es web, Command Injection si es shell).]
* **Código Seguro (Corregido):**

# Código Python seguro (ej. con saneamiento de entrada)  
**import** html  
**def** **process\_document\_name\_secure**(name):  
 sanitized\_name = html.escape(name) # Saneamiento de entrada  
 **return** f"Documento: {sanitized\_name}"

**2.2. Resultados SAST (Análisis Estático) ⚙️**

* **Herramienta Utilizada (Simulada):** [Ej: SonarQube]
* **Hallazgos Clave (en el código vulnerable):** [Ej: "Code Smell: Insecure use of user input in string formatting", "Vulnerability: Potential XSS via unsanitized input".]
* **Guía de Corrección de la Herramienta:** [Ej: "Sugiere usar funciones de saneamiento como html.escape() o librerías de validación de esquemas."]
* **Estado de Cumplimiento de Política:** [Ej: "Incumplimiento de la política 'Todas las entradas de usuario deben ser saneadas y validadas'."]

**2.3. Resultados SCA (Análisis de Composición) 📦**

* **Herramienta Utilizada (Simulada):** [Ej: Snyk]
* **Dependencia Vulnerable Identificada (si aplica):** [Ej: requests versión 2.20.0 (con vulnerabilidad CVE-202X-XXXX)]
* **Severidad de la Vulnerabilidad:** [Ej: Alta/Crítica]
* **Acciones Sugeridas por la Herramienta:** [Ej: "Actualizar a requests versión 2.28.1", "Considerar alternativa httpx."]
* **Estado de Cumplimiento de Política:** [Ej: "Incumplimiento de la política 'No se permiten dependencias con vulnerabilidades de severidad alta o crítica'."]

**🧪 Fase 3: Pruebas de Seguridad - Análisis Dinámico y Pruebas de Penetración**

**3.1. Plan de Prueba DAST (Análisis Dinámico) 🕵️‍♀️**

* **Funcionalidad Probada:** [Ej: Autenticación de usuarios - Página de login.]
* **Herramienta Utilizada (Simulada):** [Ej: OWASP ZAP]
* **Funcionalidades de ZAP Relevantes y Uso:**
  + **Spider:** [Ej: Para mapear todas las URLs y puntos de entrada de la aplicación de login.]
  + **Active Scan:** [Ej: Para inyectar payloads automáticamente y buscar vulnerabilidades como Inyección SQL, XSS, etc., en los campos de usuario/contraseña.]
  + **Fuzzer:** [Ej: Para enviar grandes volúmenes de datos o caracteres especiales a los campos de entrada para buscar fallos o comportamientos inesperados (ej. denegación de servicio, enumeración de usuarios).]
* **Vulnerabilidades Buscadas (Relacionadas con la Política):** [Ej: Mensajes de error que revelan si el usuario existe (enumeración de usuarios), falta de bloqueo de cuentas tras múltiples intentos fallidos (fuerza bruta), bypass de CAPTCHA.]

**3.2. Hallazgos de Pentesting Manual (Simulados) 🔍**

* **Técnicas Manuales Aplicadas:** [Ej: Manipulación de tokens de sesión (cambiando un ID de usuario en el token), bypass de lógica de negocio (intentar acceder a recursos de administrador cambiando un parámetro en la URL sin autenticación adecuada).]
* **Hallazgo Potencial:** [Ej: "Se logró enumerar usuarios válidos al observar diferencias en los mensajes de error de la página de login ('Usuario no encontrado' vs. 'Contraseña incorrecta')."]
* **Relación con la Política:** [Ej: "Incumplimiento de la política 'Todos los mecanismos de autenticación deben ser robustos contra ataques de fuerza bruta y enumeración de usuarios'."]
* **Recomendación de Corrección:** [Ej: "Unificar mensajes de error de autenticación para evitar la enumeración de usuarios. Implementar bloqueo de cuentas o CAPTCHA progresivo tras N intentos fallidos."]

**☁️ Fase 4: Despliegue y Monitoreo - Configuración Segura y Protección en Tiempo Real**

**4.1. Aplicación de Configuración Segura 🔒**

* **Entorno/Servidor Afectado:** [Ej: Servidor de aplicaciones en AWS EC2.]
* **Herramienta de Automatización (Simulada):** [Ej: Ansible Security]
* **Elementos de Configuración Verificados/Aplicados:**
  + [Ej: Cierre de puertos no esenciales (SSH solo desde IP restringidas, no 80/443 abiertos innecesariamente).]
  + [Ej: Deshabilitación de servicios innecesarios (ej. FTP, Telnet).]
  + [Ej: Permisos de archivos y directorios (ej. /var/www/html con permisos 755, archivos de configuración sensibles con 600).]
  + [Ej: Configuración de firewall (ej. solo permitir tráfico a puertos 80/443 desde el balanceador de carga).]
* **Proceso de Verificación/Aplicación:** [Ej: "Se ejecutó un playbook de Ansible que audita la configuración del servidor contra una línea base de seguridad y aplica automáticamente las correcciones necesarias. El reporte de Ansible confirma el cumplimiento."]

**4.2. Uso de RASP (Runtime Application Self-Protection) 🛡️**

* **Concepto de RASP:** [Ej: RASP es una tecnología que se integra directamente en la aplicación o en su entorno de ejecución para monitorear el comportamiento en tiempo real y bloquear ataques antes de que lleguen al código vulnerable.]
* **Cómo RASP Protegería SecureConnect:**
  + **Inyección SQL:** [Ej: RASP detectaría patrones de inyección SQL en las consultas a la base de datos generadas por la aplicación y bloquearía la ejecución de la consulta maliciosa.]
  + **XSS (Cross-Site Scripting):** [Ej: RASP analizaría las respuestas HTTP y bloquearía la inserción de scripts maliciosos en la página antes de que sean renderizados por el navegador del usuario.]
  + **Manipulación de Parámetros:** [Ej: RASP identificaría intentos de alterar parámetros de la URL o del cuerpo de la solicitud de formas no esperadas por la lógica de negocio y los bloquearía.]

**📈 Fase 5: Mejora Continua - Métricas y Reportes de Seguridad**

**5.1. Métricas Clave de Seguridad 📊**

* **Métrica 1:** [Ej: **Número de Vulnerabilidades Críticas Abiertas por Sprint.** Justificación: Indica la cantidad de riesgos más graves que aún no han sido remediados, priorizando la atención del equipo.]
* **Métrica 2:** [Ej: **Tiempo Promedio de Remediación (MTTR) de Vulnerabilidades de Alta Severidad.** Justificación: Mide la eficiencia del equipo en corregir problemas de seguridad importantes, impactando directamente el riesgo acumulado.]
* **Métrica 3:** [Ej: **Porcentaje de Cobertura de Escaneos SAST/DAST en el Pipeline CI/CD.** Justificación: Evalúa la madurez de la integración de seguridad en el proceso de desarrollo, asegurando que el código y la aplicación se están verificando consistentemente.]
* **Métrica 4:** [Ej: **Porcentaje de Cumplimiento de Políticas de Codificación Segura.** Justificación: Refleja la adherencia del equipo a los estándares de codificación segura, reduciendo la introducción de nuevas vulnerabilidades.]

**5.2. Esquema de Reporte Ejecutivo de Seguridad 📈**

* **Título del Reporte:** [Ej: Informe Mensual de Postura de Seguridad - SecureConnect]
* **Audiencia Objetivo:** [Ej: Alta Gerencia, CISO, Líderes de Equipo]
* **Secciones Principales:**
  1. **Resumen Ejecutivo:** [Ej: Estado general de la seguridad, tendencias clave, acciones destacadas y próximos pasos. Lenguaje no técnico.]
  2. **Métricas Clave de Seguridad:** [Ej: Gráficos y tablas de las métricas definidas en 5.1, con explicaciones concisas de su significado y tendencia.]
  3. **Vulnerabilidades Abiertas y Remediadas:** [Ej: Desglose por severidad, equipo responsable, y progreso en la remediación. Enfocado en las más críticas.]
  4. **Cumplimiento de Políticas:** [Ej: Estado de adherencia a las políticas de seguridad clave, con áreas de mejora identificadas.]
  5. **Incidentes de Seguridad (si aplica):** [Ej: Breve resumen de cualquier incidente, impacto y lecciones aprendidas.]
  6. **Próximos Pasos y Recomendaciones:** [Ej: Propuestas de acciones para mejorar la postura de seguridad, justificación de recursos.]
* **Propósito del Reporte:** [Ej: Proporcionar visibilidad estratégica sobre la seguridad del software, facilitar la toma de decisiones basada en datos y justificar la inversión en iniciativas de seguridad.]

**🧪 Conjunto de Pruebas de Seguridad: Mejorando la Precisión de los Análisis Automatizados**

**6.1. Gaps Identificados en la Detección Automatizada 🎯**

* **Gap 1:** [Ej: **Vulnerabilidades de Lógica de Negocio Compleja.** Las herramientas automatizadas luchan por entender flujos de negocio complejos donde la falla de seguridad no es un patrón de código, sino una secuencia de interacciones o una validación de permisos específica.]
* **Gap 2:** [Ej: **Falsos Positivos/Negativos en Configuraciones Específicas.** Las herramientas pueden no interpretar correctamente configuraciones muy personalizadas o entornos de despliegue únicos, llevando a alertas erróneas o a pasar por alto vulnerabilidades reales.]

**6.2. Casos de Prueba Específicos para Mejorar la Precisión 📝**

* **Caso de Prueba 1: Bypass de Lógica de Negocio en Permisos de Documentos**
  + **Descripción:** Un usuario con permisos de "lector" intenta modificar un documento cambiando directamente el ID del documento en la URL o en la carga útil de una solicitud API, sin que la interfaz de usuario lo permita. La aplicación debería validar los permisos a nivel de backend para cada operación, no solo confiar en la interfaz.
  + **No detectable por SAST/DAST estándar porque:** SAST no entiende la lógica de negocio compleja de permisos entre módulos, y DAST podría no generar la secuencia de solicitudes específica para explotar esta falla sin un conocimiento previo.
  + **Cómo mejora la precisión:** Permite identificar fallas críticas en la autorización que son específicas de la lógica de negocio de SecureConnect, que las herramientas genéricas no captarían.
  + **Integración en CI/CD (Automatización Parcial):** Se puede implementar como una **prueba de integración de seguridad** utilizando un framework de pruebas (ej. Pytest con requests para Python) que simule las solicitudes maliciosas y verifique las respuestas de la API. Se ejecutaría en cada build.
* **Caso de Prueba 2: Inyección de Encabezados HTTP Personalizados para Acceso Elevado**
  + **Descripción:** Un atacante intenta inyectar encabezados HTTP personalizados (ej. X-Admin-Role: true o X-Forwarded-For: 127.0.0.1) para engañar a la aplicación o a un proxy/WAF y obtener acceso privilegiado o bypassar restricciones de IP.
  + **No detectable por SAST/DAST estándar porque:** SAST no analiza el comportamiento de la aplicación con encabezados personalizados, y DAST podría no tener la inteligencia para inferir qué encabezados específicos podrían ser explotables sin un conocimiento profundo de la arquitectura.
  + **Cómo mejora la precisión:** Dirige la verificación a un vector de ataque específico que podría ser explotado por una mala configuración o un error en el manejo de proxies/balanceadores de carga.
  + **Integración en CI/CD (Automatización Parcial):** Se puede crear un **script de prueba personalizado** (ej. en Python con la librería requests) que envíe solicitudes con estos encabezados manipulados a un entorno de prueba, verificando las respuestas y los logs de la aplicación. Esto se ejecutaría como parte de las pruebas de despliegue.

**6.3. Ideas de Integración en CI/CD para Pruebas Complejas ⚙️**

* **Pruebas Unitarias de Seguridad:** Escribir pruebas unitarias que validen funciones críticas de seguridad (ej. validación de entrada, autenticación, autorización) contra casos de borde y payloads maliciosos.
* **Pruebas de Integración de Seguridad:** Crear suites de pruebas que simulen interacciones entre componentes o con servicios externos, inyectando payloads de seguridad y verificando el comportamiento.
* **Scripts Personalizados de Pruebas de Seguridad:** Desarrollar scripts en lenguajes como Python que automaticen escenarios de ataque complejos que las herramientas estándar no cubren, ejecutándolos como parte del pipeline de despliegue.
* **Contenedores de Pruebas de Seguridad:** Utilizar contenedores Docker para crear entornos de prueba aislados donde se puedan ejecutar herramientas de seguridad específicas o scripts personalizados de forma consistente.