

# CHECKLIST DE CONFIGURACIÓN SEGURA PARA INFRAESTRUCTURA Y APIs

## *Super App Security Kit – Fintech & Ciberseguridad*

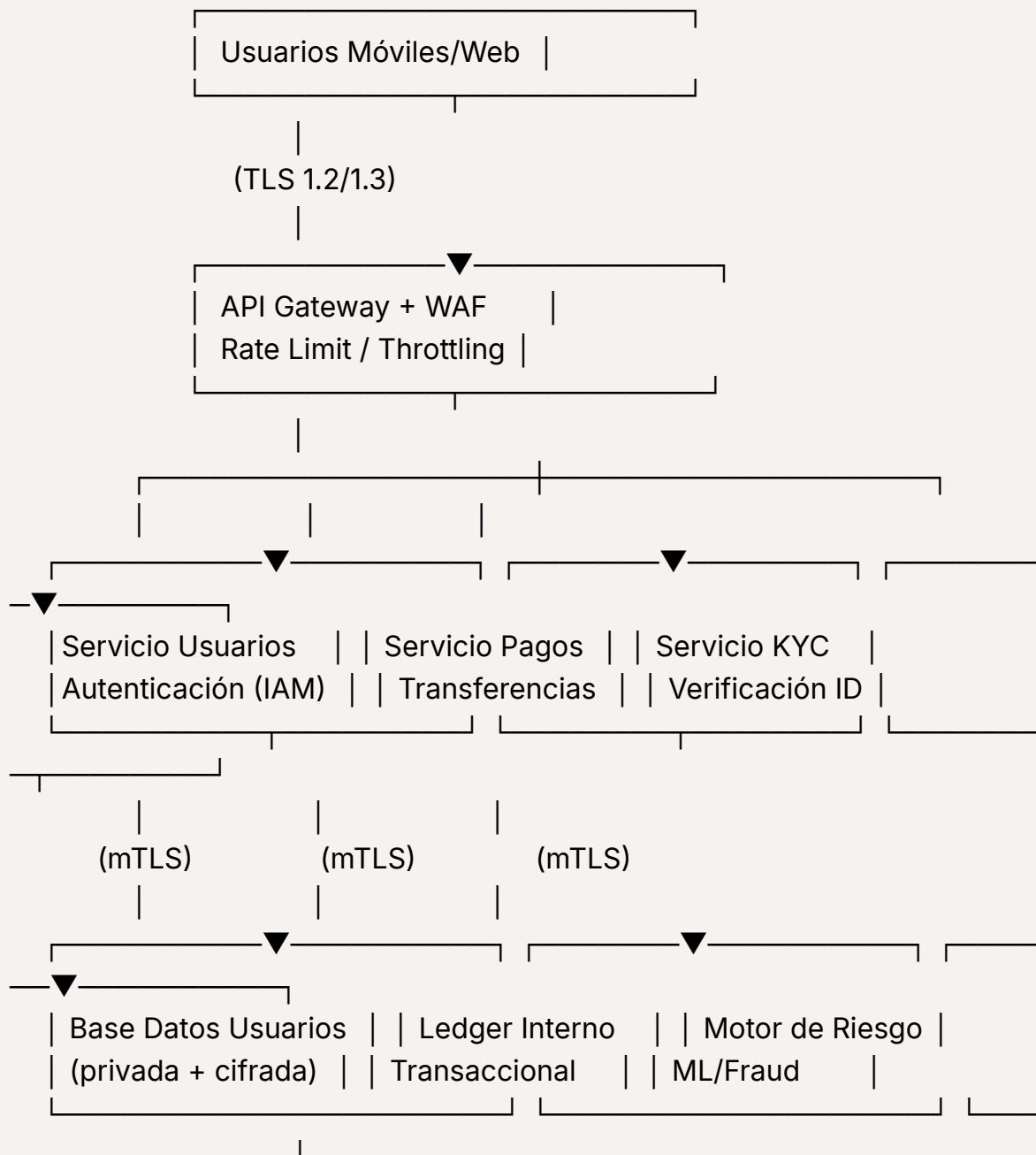
### **Introducción**

En el ecosistema fintech, las super apps se han convertido en plataformas críticas donde convergen pagos, transferencias, créditos, inversiones, identidad digital y servicios de terceros integrados mediante APIs. Este nivel de interconectividad amplía significativamente la superficie de ataque, exponiendo la infraestructura a amenazas como accesos no autorizados, filtraciones de datos, escalamiento de privilegios, ataques a APIs y explotación de vulnerabilidades en microservicios.

Las startups, especialmente durante sus primeras etapas, suelen priorizar el desarrollo rápido de funcionalidades sobre la implementación de controles de seguridad sólidos. Esto deja huecos en la arquitectura que pueden ser explotados por atacantes. Una mala configuración de infraestructura, credenciales expuestas, políticas de permisos demasiado amplios o APIs sin mecanismos de autenticación robustos pueden comprometer no solo información sensible, sino la continuidad del negocio y la confianza de los usuarios.

Por esto, este checklist reúne las prácticas esenciales para establecer una base sólida de seguridad en la infraestructura cloud, contenedores, redes, bases de datos y APIs. Su objetivo es servir como una guía clara, detallada y accionable que permita evaluar, corregir y mantener configuraciones seguras de manera consistente mientras la super app crece y escala.

### Arquitectura General



Observability Layer: Logs (ELK), Metrics (Prometheus),  
Traces (OpenTelemetry), Dashboards (Grafana)

## Checklist

# 1. Gestión de Infraestructura y Cloud

## Control de acceso y privilegios

- **Usar *Identity & Access Management (IAM)* con privilegios mínimos (principio de *Least Privilege*).**

Esto implica que cada usuario, servicio o componente debe tener únicamente los permisos estrictamente necesarios para desempeñar su función. Reducir los permisos limita el impacto en caso de que una cuenta sea comprometida y evita accesos accidentales a recursos críticos.

- **Implementar roles granulares y revisar permisos cada 30 días.**

Las políticas deben segmentarse por áreas (desarrollo, operaciones, análisis, seguridad) y definirse en roles independientes. Revisar permisos de forma mensual permite eliminar accesos innecesarios que se acumulan con el tiempo y que representan un riesgo. También se deben aplicar mecanismos como *permissions boundaries* para evitar elevaciones de privilegios no autorizadas.

- **Habilitar autenticación multifactor (MFA) para todas las cuentas con acceso al cloud.**

MFA añade un nivel extra de seguridad en caso de que una contraseña sea filtrada. Es indispensable para cuentas con privilegios administrativos, pero debe aplicarse a todos los usuarios que gestionen infraestructura. Idealmente combinar factores como aplicaciones autenticadoras, llaves FIDO2 o tokens de hardware.

- **Prohibir cuentas compartidas; cada miembro debe tener un usuario individual.**

Las cuentas compartidas impiden rastrear acciones, dificultan auditorías y crean huecos de seguridad. Cada colaborador debe tener una identidad única de acceso con permisos específicos según su función. Cuando alguien abandona el equipo, su cuenta se debe desactivar de inmediato, evitando dejar accesos abiertos.

- **Registrar y auditar todos los cambios en IAM y accesos administrativos.**

Es fundamental habilitar logs detallados que registren quién realiza modificaciones, cuándo y desde dónde. Estos registros deben integrarse a una plataforma SIEM para detectar patrones sospechosos como intentos fallidos, elevación de privilegios, creación de usuarios inesperados o eliminación de políticas. Mantener auditorías activas ayuda a cumplir normas de cumplimiento como PCI-DSS, ISO 27001 y SOC 2.

## **Seguridad de redes**

- **Configurar VPCs separadas para producción, desarrollo y pruebas.**

Separar entornos evita que errores o vulnerabilidades en desarrollo afecten sistemas en producción. Cada VPC debe tener su propio control de tráfico, subredes y reglas de seguridad independientes. Esto asegura que los datos sensibles se mantengan aislados y reduce la superficie de ataque.

- **Implementar segmentación por subredes (red pública, privada y de datos).**

La red pública debe alojar únicamente balanceadores o servicios expuestos, mientras que los servidores de aplicación y bases de datos deben permanecer en redes privadas. La subred de datos debe tener restricciones más estrictas, evitando cualquier acceso directo desde internet y limitando la comunicación solo a microservicios autorizados.

- **Limitar puertos abiertos únicamente a lo estrictamente necesario.**

Mantener puertos cerrados por defecto reduce el riesgo de accesos no autorizados. Es esencial realizar escaneos periódicos de puertos expuestos y bloquear cualquier apertura innecesaria. Esto previene ataques comunes como fuerza bruta, exploits en servicios expuestos o intrusiones automatizadas.

- **Habilitar firewalls en capa de aplicación (WAF) contra ataques como XSS y SQL injection.**

Un WAF inspecciona tráfico HTTP y bloquea patrones maliciosos antes de que lleguen a la aplicación. Esto es crítico en super apps financieras, donde formularios, autenticación y APIs reciben datos externos de miles de usuarios cada minuto.

- **Restringir accesos SSH/RDP mediante VPN corporativa o túneles Zero Trust.**

El acceso directo desde internet a servidores es una de las principales causas de ataques exitosos. Se debe exigir acceso mediante VPN con autenticación reforzada o sistemas Zero Trust que validan continuamente identidad, dispositivo y contexto. También se debe habilitar *just-in-time access* y restringir tiempos de conexión.

## **Gestión de máquinas virtuales y contenedores**

- **Usar imágenes base oficiales y verificadas.**

Todas las máquinas virtuales (VMs) y contenedores deben construirse a partir de imágenes confiables, mantenidas por proveedores oficiales o equipos internos de seguridad. Esto reduce el riesgo de utilizar imágenes con malware, configuraciones débiles o paquetes obsoletos. Las imágenes deben incluir únicamente lo necesario para ejecutar el servicio, evitando componentes innecesarios que aumentan la superficie de ataque.

- **Activar escaneo automático de vulnerabilidades en imágenes Docker.**

Antes de desplegar cualquier contenedor, es necesario analizarlo con herramientas de *container scanning* (Trivy, Clair, Snyk). Estos escáneres detectan librerías vulnerables, configuraciones inseguras, puertos abiertos y dependencias desactualizadas. El escaneo debe integrarse dentro del pipeline CI/CD para evitar que imágenes inseguras lleguen a producción.

- **Aplicar parches de seguridad en sistemas operativos y kernels.**

Mantener los sistemas operativos actualizados es clave para prevenir ataques basados en vulnerabilidades conocidas (como escalamiento de privilegios o ejecución remota). Debe habilitarse un calendario de actualizaciones periódicas y mecanismos automatizados para aplicar *security patches* críticos. En ambientes productivos, los parches deben aplicarse con técnicas de despliegue seguro como rolling updates.

- **Asegurar contenedores con capacidades limitadas: sin root, sin acceso a host namespace.**

Los contenedores deben ejecutarse con usuarios no privilegiados, sin permisos de root ni capacidades ampliadas. También es importante evitar el uso de `--privileged` y no montar directorios críticos del host. Limitar las capacidades del contenedor evita que un atacante pueda escapar al sistema host o ejecutar comandos peligrosos.

- **Implementar políticas como *Pod Security Standards* (en Kubernetes).**

Cuando se usa Kubernetes, las políticas de seguridad deben garantizar que los pods se ejecuten bajo condiciones restringidas:

- sin escalamiento de privilegios
- sin acceso a sistema de archivos del host
- sin permisos de root
- con límites de CPU y memoria

Estas políticas evitan configuraciones inseguras y aseguran que cada microservicio opere en un entorno aislado y controlado.

## 2. Seguridad de APIs

### Autenticación & Autorización

- Requerir tokens firmados (OAuth 2.0, JWT seguros y con expiración corta).

Las APIs deben aceptar únicamente solicitudes autenticadas mediante tokens generados por un proveedor confiable. Los JWT deben firmarse con algoritmos fuertes (RS256 o ES256), nunca con HS256 si existen múltiples consumidores. Además, los tokens deben tener tiempos de expiración cortos para reducir el riesgo de uso indebido si se filtran.

- Aplicar validación de permisos por endpoint (*Role-Based Access Control* y *Attribute-Based Access Control*).

No es suficiente validar la identidad del usuario: cada operación debe verificar sus permisos y atributos. RBAC garantiza que roles como "cliente", "comercio", "agente" o "administrador" tengan accesos

diferenciados. ABAC permite reglas más dinámicas basadas en contexto, como ubicación, tipo de dispositivo o nivel de riesgo.

- Asegurar flujos de autenticación con PKCE en apps móviles.

El estándar OAuth 2.0 + PKCE protege los flujos de autorización contra interceptación de códigos. Esto es indispensable para apps fintech en dispositivos móviles, donde el usuario es más vulnerable a ataques MITM, malware o redes WiFi inseguras.

## Control de tráfico y protección contra abuso

- Habilitar *rate limiting* (límites por IP, usuario, token).

Esto evita ataques como **fuerza bruta**, **scraping**, intentos automáticos de transferencias o creación de cuentas. Los límites deben variar según el tipo de endpoint: algunos críticos (como login) requieren límites más estrictos.

- Implementar *API throttling* para evitar automatización maliciosa.

El throttling evita que usuarios o bots generen tráfico excesivo que degrade el servicio o prepare un ataque DDoS. Las super apps deben combinar throttling con *dynamic rate limiting* y sistemas de reputación.

- Usar *request validation* estricta: tipos de datos, tamaños máximos, estructuras permitidas.

Toda entrada debe validarse antes de llegar al backend. Esto previene ataques como deserialización insegura, payloads excesivos o inyección de tipos. Se deben rechazar formatos incorrectos con mensajes de error genéricos y seguros.

## Cifrado

- Exigir HTTPS en toda comunicación (TLS 1.2 o superior).

Ninguna API debe aceptar solicitudes HTTP planas. Es obligatorio usar TLS moderno, con certificados válidos, transparencia de certificados y configuraciones seguras del servidor.

- Deshabilitar TLS débiles o inseguros (SSL, TLS 1.0, 1.1).

Estos protocolos tienen vulnerabilidades documentadas (POODLE, BEAST, Heartbleed). Su presencia expone al sistema a ataques MITM, robo de sesiones o descifrado del tráfico.

- Cifrar datos sensibles dentro del payload cuando corresponda (PII o financieros).

Aunque TLS protege el transporte, la información crítica debe cifrarse también internamente para protegerla en logs, bases de datos o servicios intermedios.

## **Gestión de claves y secretos**

- No guardar claves dentro del código ni repositorios.

Es una de las principales causas de brechas en startups. Las claves nunca deben aparecer en GitHub, config.js, .env distribuidos o imágenes Docker.

- Utilizar herramientas como AWS Secrets Manager, GCP Secret Manager o Vault.

Estas herramientas permiten almacenar, rotar, encriptar y gestionar secretos de forma centralizada, protegidos por políticas de acceso estrictas.

- Rotar claves y tokens periódicamente.

La rotación reduce el riesgo de uso prolongado de credenciales filtradas. Las claves deben tener un ciclo de vida definido y automático.

- Implementar detección automática de secretos expuestos.

El pipeline de CI/CD debe incluir escaneos SAST o herramientas como Gitleaks, TruffleHog o GitGuardian para prevenir filtraciones.

## **Monitoreo y auditoría**

- Registrar todas las llamadas a APIs: parámetros, origen, usuario, tipo de error.

Un registro detallado permite detectar patrones maliciosos, intentos de fraude o uso anormal del sistema. En fintech esto es obligatorio para cumplir auditorías regulatorias.

- Analizar patrones anómalos como intentos excesivos de login.

Deben usarse sistemas automáticos de detección de anomalías para identificar bots, scraping o comportamientos sospechosos antes de que causen daño.

- Integrar alertas con SIEM para detectar comportamientos sospechosos.

Las APIs deben enviar métricas, logs y eventos a soluciones centralizadas (Splunk, Elastic, IBM QRadar, Datadog). De esta manera, los equipos de seguridad pueden reaccionar rápidamente ante incidentes.

### 3. Seguridad de Bases de Datos

#### **Aplicar cifrado *en reposo* y *en tránsito***

En una super app fintech, los datos que se almacenan suelen incluir información extremadamente sensible: datos personales, historial financiero, tokens de pago, direcciones y actividad transaccional. Por esto, toda base de datos debe cifrar la información tanto cuando está almacenada (*en reposo*) como mientras viaja por la red (*en tránsito*).

- *En reposo*: Usar mecanismos nativos del proveedor (TDE en SQL Server, AES-256 en PostgreSQL, KMS en AWS/GCP/Azure).
- *En tránsito*: Toda conexión debe usar TLS/SSL para evitar ataques de intermediarios.

Esto garantiza que incluso si un atacante accede a los archivos físicos o intercepta tráfico interno, no podrá leer los datos.

#### **Acceso desde subredes privadas solamente**

Las bases de datos nunca deben ser accesibles desde internet.

El acceso debe limitarse únicamente a máquinas o contenedores dentro de subredes privadas, protegidas detrás de firewalls y sin direcciones IP públicas.

Esto evita ataques automatizados, escaneos externos y vulnerabilidades explotadas desde la red pública.

En casos especiales, herramientas de administración deben conectarse a través de:

- VPN corporativa
- túneles SSH
- Zero Trust Access

Nunca conexión directa.

## **Activar auditorías de consulta y modificación de datos**

Las auditorías permiten registrar quién accede, qué consulta, qué modifica y cuándo lo hace. Esto es esencial en contextos financieros.

Una auditoría bien configurada permite detectar:

- accesos indebidos
- extracción masiva de datos
- intentos de manipular transacciones
- conductas sospechosas de usuarios internos (amenazas internas)

También ayuda a cumplir normativas como PCI-DSS, ISO 27001, SOC 2, GDPR y marcos latinoamericanos como Habeas Data.

## **Configurar backups automáticos y pruebas de restauración**

No basta con tener copias de seguridad; deben ser:

- automáticas
- cifradas
- segregadas de la infraestructura principal

- con retención adecuada (mínimo 30 días en fintech)
- probadas regularmente

Las pruebas de restauración garantizan que en caso de falla, ransomware o ataque interno, la super app pueda restaurarse completa y rápidamente sin pérdida de datos.

### **Aplicar reglas estrictas de firewall: solo acceso desde la app, no público**

El firewall de base de datos debe configurarse para aceptar conexiones únicamente desde:

- servidores de backend
- microservicios autorizados
- IPs internas específicas

Nunca desde:

- interfaces web
- dispositivos externos
- paneles de administración sin protección
- IPs públicas

Esto reduce puntos de entrada y evita que atacantes alcancen el servidor de datos incluso si comprometen la superficie externa.

## **4. Seguridad en Microservicios y Arquitecturas Distribuidas**

### **Aislamiento entre servicios con políticas Zero Trust**

En una arquitectura de microservicios, ningún servicio debe confiar automáticamente en otro, aunque esté dentro de la misma red. El modelo Zero Trust exige que **cada solicitud entre microservicios sea autenticada, autorizada y registrada**, sin excepciones.

Esto evita ataques laterales donde un atacante compromete un microservicio y luego se mueve a otros de la red.

Zero Trust garantiza:

- Identidad verificable para cada servicio.
- Acceso permitido solo según necesidad.
- Verificaciones continuas (*continuous authentication*).

Esto es esencial en una super app, donde módulos financieros, comerciales, logísticos y de pagos comparten infraestructura pero manejan datos con sensibilidad distinta.

## **Firmar y validar solicitudes internas entre microservicios**

Incluso dentro del mismo cluster o VPC, los microservicios deben intercambiar mensajes firmados o autenticados.

Esto puede lograrse mediante:

- Tokens JWT de corto tiempo entre servicios.
- Mutual TLS (mTLS) con certificados rotados automáticamente.
- Firmas HMAC para garantizar integridad.

Esta práctica previene ataques como:

- manipulación de payloads
- falsificación de peticiones internas
- inyección de tráfico malicioso por un servicio comprometido

En fintech es obligatorio asegurar la integridad de transacciones y datos sensibles, incluso dentro del backend.

## **Limitar el acceso entre microservicios usando service mesh (Istio, Linkerd)**

Un *service mesh* proporciona una capa de control que gestiona:

- autenticación
- autorización

- encriptación
- políticas de tráfico
- observabilidad
- retries y timeouts seguros

Istio o Linkerd permiten aplicar reglas detalladas como:

- qué servicios pueden comunicarse entre sí
- cantidades máximas de solicitudes permitidas
- validación automática de certificados
- bloqueo de comunicaciones no autorizadas

Además, facilitan un despliegue más seguro sin tener que modificar directamente el código de cada microservicio.

## **Definir políticas de comunicación: quién puede hablar con quién**

Cada microservicio debe tener permisos explícitos sobre qué otro servicio puede consumir o emitir datos.

Esto se expresa mediante:

- NetworkPolicies en Kubernetes
- ACLs internas
- reglas de firewall de microservicios
- controladores RBAC de comunicación

Estas políticas previenen ataques laterales, escaneo interno, manipulación de datos y accesos indebidos.

Ejemplo:

El servicio de pagos no debería poder acceder directamente al servicio de usuarios sin pasar por validaciones apropiadas.

## **Aplicar rate limiting y circuit breakers por servicio**

Cada microservicio debe tener sus propios controles de resiliencia y defensa ante abuso.

Los *rate limits* evitan que un microservicio reciba tráfico excesivo (legítimo o malicioso).

Los *circuit breakers* permiten que un servicio corte temporalmente conexiones hacia otro si detecta:

- errores repetidos
- lentitud extrema
- comportamiento anómalo

Esto:

- evita cascadas de fallas
- disminuye impactos de ataques DDoS internos
- mantiene la aplicación estable incluso ante errores aislados

Arquitecturas financieras deben garantizar disponibilidad constante, especialmente en módulos transaccionales.

## **Usar logs estructurados y trazabilidad distribuida (OpenTelemetry)**

Los microservicios generan una enorme cantidad de interacciones.

Para seguir el flujo de una operación (por ejemplo, un pago), se requieren:

- Logs estructurados en JSON
- Identificadores únicos por solicitud (*trace IDs*, *span IDs*)
- Integración con OpenTelemetry
- Sistemas centralizados de análisis (Elastic, Datadog, Prometheus + Tempo)

Esto permite:

- detectar ataques distribuidos
- analizar errores entre múltiples servicios

- seguir el rastro de transacciones sensibles
- cumplir auditorías y regulaciones exigentes

En una super app fintech, esto es crítico para reconstruir eventos en caso de intentos de fraude.

## 5. Seguridad Operacional

### **Monitoreo continuo de logs, métricas y alertas**

Una super app fintech debe operar bajo un monitoreo permanente para detectar actividad anómala, fallos críticos, intentos de acceso indebidos o patrones que sugieran ataques automatizados. Esto implica integrar todas las fuentes de datos (APIs, microservicios, bases de datos, contenedores, firewalls y sistemas cloud) en una plataforma centralizada.

Buenas prácticas:

- recolectar logs estructurados y enviarlos a un SIEM
- monitorear métricas en tiempo real (CPU, memoria, latencia, número de solicitudes, errores por endpoint)
- detectar anomalías mediante análisis de comportamiento y machine learning
- implementar paneles de observabilidad para equipos de seguridad y operaciones

Un monitoreo continuo permite actuar antes de que un incidente ponga en riesgo transacciones, datos o disponibilidad del servicio.

### **Configurar alertas para detectar actividad sospechosa**

Las alertas deben estar diseñadas para identificar comportamientos inusuales tanto internos como externos, por ejemplo:

- múltiples intentos fallidos de acceso (brute force)
- transferencias o transacciones fuera del patrón habitual

- accesos a endpoints administrativos desde ubicaciones no autorizadas
- creación masiva de cuentas desde la misma IP
- escaneo de puertos o rutas API poco comunes

Las super apps, especialmente en el sector fintech, deben ser extremadamente sensibles a cualquier señal de fraude, automatización maliciosa o intrusión. Las alertas deben tener niveles de severidad y procedimientos claros según la gravedad.

## **Integración con SIEM (Security Information and Event Management)**

Un SIEM es esencial para correlacionar eventos provenientes de distintas partes de la arquitectura y así detectar ataques complejos o persistentes.

Los sistemas recomendados incluyen:

- Splunk
- IBM QRadar
- Elastic SIEM
- Microsoft Sentinel
- Wazuh (open source)

Con un SIEM bien configurado, la empresa puede:

- detectar patrones de ataque distribuidos
- identificar comportamientos anómalos a través del tiempo
- correlacionar actividades que parecen inofensivas individualmente
- cumplir requisitos regulatorios (PCI DSS, GDPR, ISO 27001)
- generar reportes automáticos para auditoría

En fintech, el SIEM es un componente obligatorio para proteger datos financieros y evitar fraudes.

## **Planes de respuesta ante incidentes y simulacros periódicos**

Un incidente de seguridad no es cuestión de "si pasa", sino "cuándo pasa".

Para reducir impacto, cada organización debe mantener un plan formal de respuesta que incluya:

- detección
- contención
- erradicación
- recuperación
- análisis post-incidente

El plan debe definir roles y responsabilidades claras: quién comunica, quién investiga, quién decide cortar servicios, quién notifica a reguladores y clientes.

Además, deben realizarse simulacros periódicos como:

- ejercicios de *tabletop*
- simulaciones de ransomware
- pruebas de fuga de datos
- ataques de denegación de servicio

Estos simulacros permiten medir tiempos de reacción, coordinar equipos y corregir fallas en los procedimientos antes de una crisis real.

## **Pruebas regulares de penetración y análisis de vulnerabilidades**

Las super apps son blancos atractivos para atacantes, por eso deben someterse a pruebas constantes:

- *Penetration testing* externo (por empresas especializadas)
- *Ethical hacking* interno
- escaneos automáticos de vulnerabilidades (SAST, DAST, SCA)

- pruebas de seguridad en APIs (fuzzing, validación de entradas, bypass de autenticación)

El objetivo principal es:

- identificar debilidades antes que los atacantes
- verificar que medidas de seguridad funcionan
- evaluar el impacto real de una brecha hipotética
- mantener cumplimiento regulatorio en el sector financiero

Estas pruebas deben realizarse al menos cada 3 o 6 meses, y siempre después de cambios significativos en la infraestructura.

## **6. Monitoreo, Alertas y Respuesta a Incidentes**

El monitoreo continuo es un pilar crítico para la seguridad de una super app, ya que permite detectar actividades anómalas antes de que se conviertan en brechas graves. Una infraestructura fintech debe contar con sistemas de observabilidad, correlación de eventos y protocolos de respuesta listos para activarse en minutos. Esta capa de vigilancia permanente asegura que cualquier desviación en el comportamiento normal del sistema, del tráfico o de los usuarios sea detectada, analizada y atendida sin afectar la disponibilidad del servicio ni la confianza de los clientes.

### **Sistema de monitoreo centralizado**

- Implementar plataformas SIEM (como Splunk, Azure Sentinel, Elastic Security) para consolidar logs de toda la infraestructura.
- Recopilar eventos de aplicaciones, APIs, contenedores, bases de datos, IAM, firewalls y sistemas operativos.
- Establecer dashboards con métricas clave: intentos fallidos de autenticación, errores 5xx, patrones sospechosos de APIs, uso anómalo de CPU/memoria, etc.
- Configurar retención de logs de al menos 6 a 12 meses, según normativas fintech (ej. ISO 27001, PCI-DSS).

- Garantizar que los logs sean inmutables mediante almacenamiento WORM (*Write Once, Read Many*).

## **Alertas inteligentes y detección de amenazas**

- Crear alertas basadas en patrones de abuso: *brute force*, *credential stuffing*, scraping, llamadas masivas a endpoints sensibles.
- Activar detección de comportamientos anómalos mediante análisis de datos y machine learning.
- Integrar sistemas EDR/XDR para detectar actividad maliciosa dentro de servidores y contenedores.
- Asegurar que todas las alertas críticas se notifiquen automáticamente al equipo mediante Slack, email o PagerDuty.
- Clasificar alertas por niveles: informativas, sospechosas, críticas y bloqueantes.

## **Respuesta a incidentes (IR): protocolos y responsabilidades**

- Definir un playbook de respuesta a incidentes con pasos claros: detección, contención, erradicación, recuperación y análisis post-incidente.
- Asignar roles del equipo de IR: responsable técnico, analista de seguridad, comunicaciones y cumplimiento normativo.
- Configurar procedimientos de aislamiento automático para endpoints y cargas de trabajo comprometidas.
- Documentar cada incidente en un registro central, incluyendo evidencia, línea de tiempo, impacto y acciones tomadas.
- Establecer un proceso de comunicación interna y externa para incidentes graves (incluyendo reguladores si aplica).

## **Pruebas y simulaciones periódicas**

- Realizar simulacros de incidentes (tabletop exercises) cada trimestre para evaluar la preparación del equipo.

- Ejecutar simulaciones de ataques reales: ransomware, extracción de datos, escalamiento de privilegios y ataques a APIs.
- Revisar y actualizar el playbook de IR después de cada simulación para corregir brechas del proceso.
- Validar la integración de todos los sistemas SIEM/EDR durante las pruebas para asegurar que no existan puntos ciegos.