DevSecOps:
guía de seguridad integrada desde el diseño y por defecto

2025

CSC
CYBER SECURITY CENTRE

isms forum

Equipo.

Coordinador:

Enrique Cervantes

Especialista en ciberseguridad y arquitectura IT con más de 15 años de experiencia en banca digital, energía, seguros y consultoría tecnológica. Experto en diseño de estrategias de seguridad, protección en entornos cloud y liderazgo de equipos técnicos. Ha liderado iniciativas de transformación digital, integración de seguridad en ciclos DevSecOps y definición de arquitecturas zero trust. Colabora como docente y director académico en programas de máster vinculados a ciberseguridad e innovación tecnológica.

Participantes:

Álvaro Mora Barrera

Arquitecto de seguridad con más de 15 años de experiencia en ciberseguridad, gestión de vulnerabilidades y protección de infraestructuras críticas en sectores como banca, tecnología, energía y educación. Especializado en diseño de arquitecturas seguras, DevSecOps, seguridad en entornos cloud y modelado de amenazas. Ha liderado estrategias de seguridad desde la fase de diseño, alineado con normativas internacionales y en colaboración con áreas legales, de riesgos y privacidad.

Aníbal Díaz Ginés

Especialista en ciberseguridad con más de 17 años de experiencia liderando proyectos estratégicos en entornos complejos y críticos. Experto en gestión de riesgos, implantación de marcos normativos (ISO, ISMS, ITSM) y dirección de equipos en áreas de defensa, telecomunicaciones y consultoría tecnológica. Ha coordinado iniciativas globales de seguridad, desde la gestión de identidades hasta la protección de infraestructuras, combinando visión técnica y de negocio.

Arturo Navarro Cruz

Profesional con más de 10 años de experiencia en ciberseguridad, liderando equipos y diseñando estrategias alineadas con los objetivos empresariales. Actualmente, como subdirector de Seguridad, supervisa operaciones y proyectos, mejorando la gestión de riesgos y reduciendo incidentes. Con un MBA, combina habilidades estratégicas y tecnológicas para optimizar la protección corporativa.

Eva González de Canales González

Especialista en seguridad informática con más de 10 años de experiencia impulsando prácticas de desarrollo seguro, análisis de vulnerabilidades y mejora continua en el ciclo de vida del software. Ha liderado iniciativas de seguridad en el sector financiero, combinando conocimientos técnicos en criptografía, redes y aplicaciones con una sólida vocación docente.

Fran García Martínez

Es especialista en seguridad en la nube con experiencia internacional en el sector salud. Ingeniero Informático con Máster en Seguridad Informática por el Illinois Institute of Technology de Chicago, lidera programas estratégicos de seguridad, desarrolla soluciones alineadas con el negocio y es Board Member de Cloud Security Alliance de ISMS Forum.

Jorge Pardeiro Sánchez

Profesional con más de 10 años de experiencia en ciberseguridad, especializado en arquitectura de seguridad, identidad digital y protección de datos. Actualmente lidera equipos en el diseño y ejecución de estrategias de ciberseguridad, gestionando proyectos en cloud y DevSecOps. Ha trabajado con equipos grandes y ha supervisado presupuestos, implementando tecnologías avanzadas en autenticación y gestión de accesos. Titulado en Ingeniería Informática y Executive MBA.

José Antonio Gutiérrez González

Profesional de la Ciberseguridad con más de 18 años de experiencia. Inició como Consultor para clientes nacionales de diversos sectores. Desde 2016, lidera el área de Ingeniería de Seguridad en una empresa del sector retail. Es Ingeniero Técnico en Informática de Sistemas y posee un Máster en Seguridad de la Información.

José Manuel Rivera García

Profesional en ciberseguridad con amplia experiencia en el sector financiero, actualmente lidera la estrategia de seguridad de la información como CISO. Es Board Member de Cloud Security Alliance de ISMS Forum. Ha trabajado en proyectos de Zero Trust, arquitectura ZTNA y entornos multicloud. Combina un sólido perfil técnico con formación en desarrollo directivo por la Universidad Pontificia Comillas (ICAI-ICADE)..

Yan Bello

PMP, CDBA; Licenciado en Matemáticas y Máster Oficial en IA. Fundador y CEO con más de 18 años impulsando la innovación organizacional, la inteligencia artificial aplicada y la transformación ágil en múltiples sectores. Experto en estrategia, liderazgo de proyectos globales y facilitación de procesos colaborativos. 30 años de experiencia profesional y de investigación en Inteligencia Artificial. Experto en modelos de madurez, metodologías y proceso software. Ha diseñado diversas metodologías propias. Colabora como docente en másteres y dirección de tesis en varias universidades.

Gestión de Proyecto:

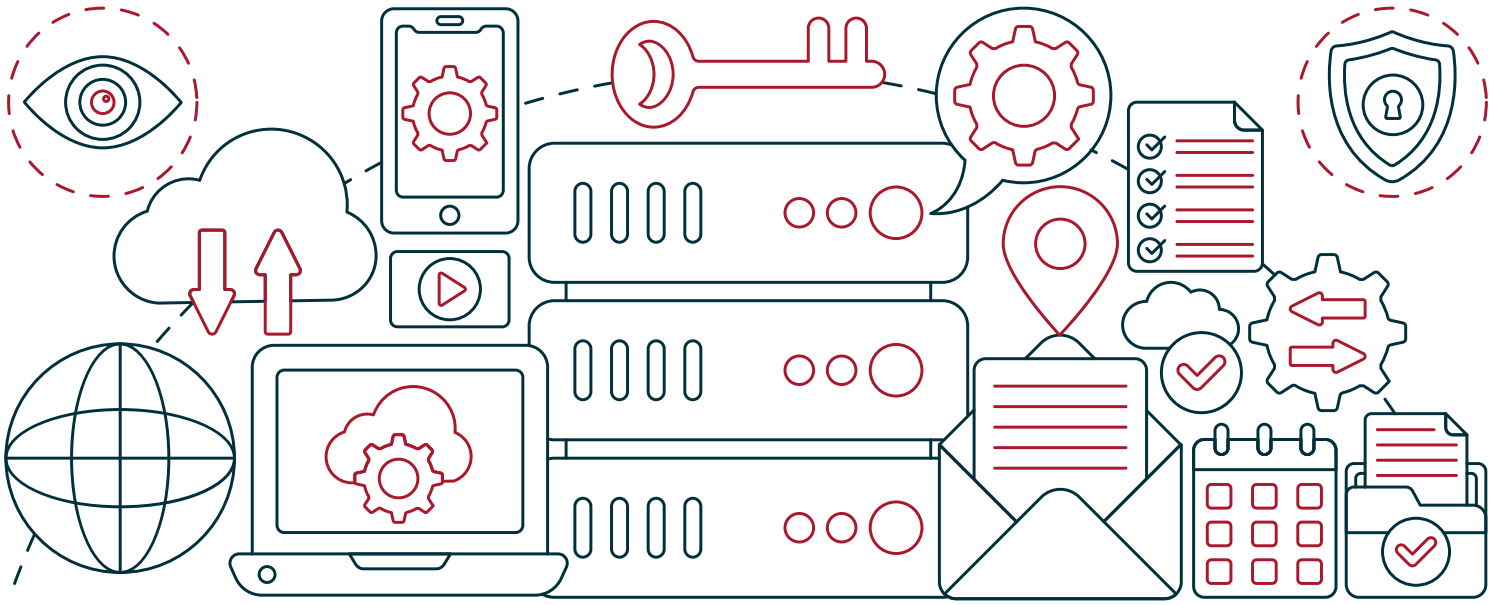
Beatriz García González

Licenciada en Psicología Social por la UCM, desempeña actualmente el cargo de Subdirectora General y Responsable de Proyectos en ISMS Forum. Con una trayectoria profesional que abarca más de 17 años en el ámbito laboral y de recursos humanos, ha consolidado su experiencia como líder en la gestión estratégica de proyectos y el fomento del conocimiento en ciberseguridad.

Diseño y maquetación:

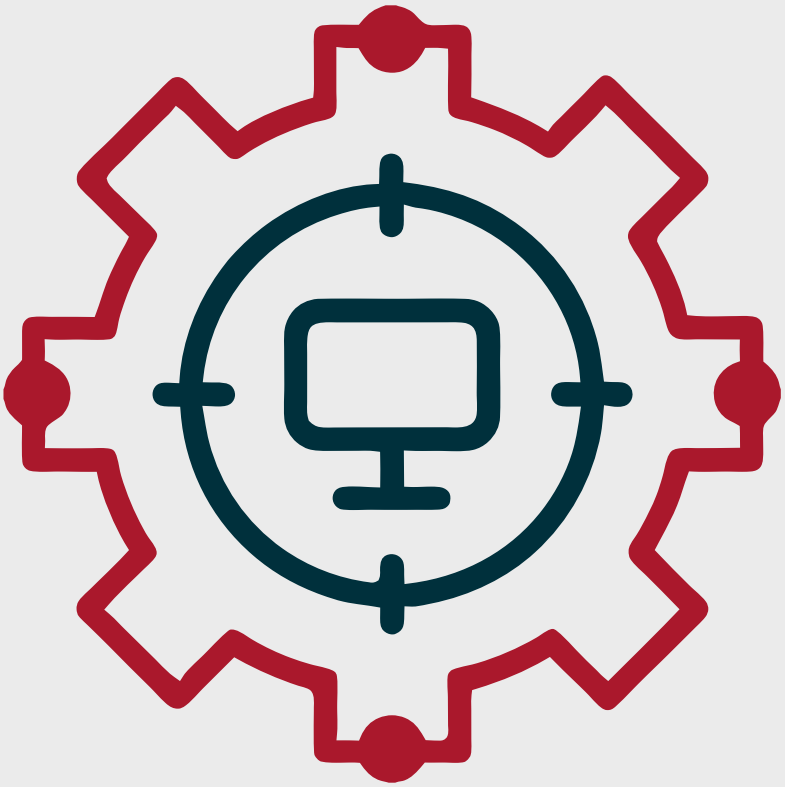
Marta Barroso Cabrera

Licenciada en Periodismo por la UCM, cuenta con una sólida trayectoria de más de 15 años en comunicación externa e interna, así como en la gestión de redes sociales, redacción y edición audiovisual. Actualmente es Communication Manager en ISMS Forum.



Contenido.

1. Introducción.



1.1. Contexto actual de los entornos de desarrollo corporativos

Actualmente las empresas se encuentran en un momento de transformación digital, la agilidad que demanda el mercado, la inclusión de la IA en sus procesos para ser más eficientes hace que los entornos de desarrollo también evolucionen.

En consecuencia, los entornos de desarrollo corporativos están marcados por varias tendencias y desafíos clave:

1. Transformación Digital

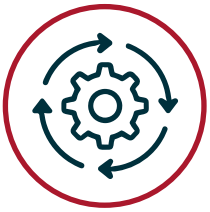
La adopción de tecnologías avanzadas como la inteligencia artificial, el aprendizaje automático y la automatización está revolucionando los procesos de desarrollo. Estas tecnologías permiten una mayor personalización y eficiencia en los programas de capacitación

2. Aprendizaje Continuo

La necesidad de aprendizaje continuo es más crítica que nunca. Las empresas están fomentando una cultura de aprendizaje donde los empleados tienen acceso a recursos educativos de manera constante para mantenerse actualizados y competitivos

3. Trabajo Híbrido

La pandemia ha acelerado la adopción de modelos de trabajo híbridos. Esto ha llevado a un aumento en la demanda de plataformas de capacitación en línea que sean accesibles desde cualquier lugar



4. Gestión del Cambio

La capacidad de gestionar el cambio de manera efectiva se ha convertido en una competencia crucial. Las empresas deben ser ágiles y estar preparadas para adaptarse rápidamente a nuevas circunstancias y tecnologías

5. Diversidad e Inclusión

La promoción de la diversidad y la inclusión en el lugar de trabajo es una prioridad. Los programas de desarrollo están siendo diseñados para ser más inclusivos y para abordar las necesidades de una fuerza laboral diversa

> *Estos factores están configurando un entorno de desarrollo corporativo que es dinámico, centrado en el empleado y altamente dependiente de la tecnología.*

1.2. Introducción a SDLC

El Ciclo de Vida del Desarrollo de Software (SDLC) es un proceso estructurado que define las etapas o fases necesarias para el desarrollo, implementación y mantenimiento de software. Aplicar un SDLC permite llevar a cabo un desarrollo ordenado, facilitando una definición adecuada del alcance del software – a través de requisitos, historias de usuario, casos de uso, etc. para capturar las expectativas, funcionalidades deseadas, condicionantes, entre otros; y permitir una identificación temprana de riesgos cuya gestión proactiva puede mejorar la calidad y seguridad del software resultante.

Un SDLC bien definido, aporta un marco metodológico que descompone el desarrollo de software en fases claramente definidas, permitiendo gestionar eficientemente el proyecto, reducir errores y aumentar la seguridad desde etapas iniciales.

Fases generales de un SDLC

A continuación, se describen las fases más frecuentes de un SDLC, haciendo mención de aspectos de seguridad. Si bien estas fases comprenden actividades que, en función del contexto organizativo, enfoque concreto de cada proyecto o equipo, etc. se pueden realizar de forma secuencial, iterativa, incremental, "solapadas", etc., por lo general estas están presentes en prácticamente todo SDLC en mayor o menor medida. Cabe destacar que no existe una única forma de organizar estos trabajos.

- **Planificación:**
  - Identificación y definición de requisitos de negocio.
  - Análisis preliminar de riesgos y establecimiento de estrategias de seguridad iniciales.
- **Análisis:**
  - Definición detallada de requisitos funcionales y no funcionales.
  - Identificación temprana de requisitos específicos de seguridad.
- **Diseño:**
  - Creación de la arquitectura del software enfocada en seguridad.
  - Implementación del modelado de amenazas ("Threat Modeling"). Ver sección 3.2.
  - Definición clara de controles y contramedidas de seguridad.
- **Desarrollo:**
  - Aplicación de prácticas seguras de codificación (OWASP Guidelines)[1].
  - Integración de herramientas automatizadas desde el desarrollo para el análisis estático – pruebas estáticas de seguridad de aplicaciones (SAST: Static Application Security Testing) y pruebas dinámicas (DAST: Dynamic Application Security Testing).
- **Pruebas:**
  - Ejecución de pruebas funcionales y pruebas específicas de seguridad como Penetration Testing. Realización de las pruebas (SAST/DAST)
  - Validación sistemática del cumplimiento de estándares de seguridad.
  - Realización de pruebas específicas para sistemas que incorporan IA (genIA, LLMs, etc.), considerando los riesgos y amenazas que estos contemplan.
- **Implementación y Despliegue Automático:**
  - Uso de CI/CD (Integración Continua y Entrega Continua) para automatizar despliegues.
  - Estrategias robustas para la monitorización post-despliegue y detección/gestión temprana de incidentes.
  - Implementación de pipelines seguros mediante herramientas apropiadas, tales como: Jenkins, GitHub Actions, GitLab CI/CD o Azure DevOps, etc.
  - Uso de firmas digitales y validaciones automáticas para garantizar integridad y seguridad en los pipelines.
- **Mantenimiento:**
  - Gestión continua de vulnerabilidades y aplicación de parches de seguridad.
  - Actualización regular y monitoreo constante en producción.

1 "Why Adding Security in Dev Teams is Important"

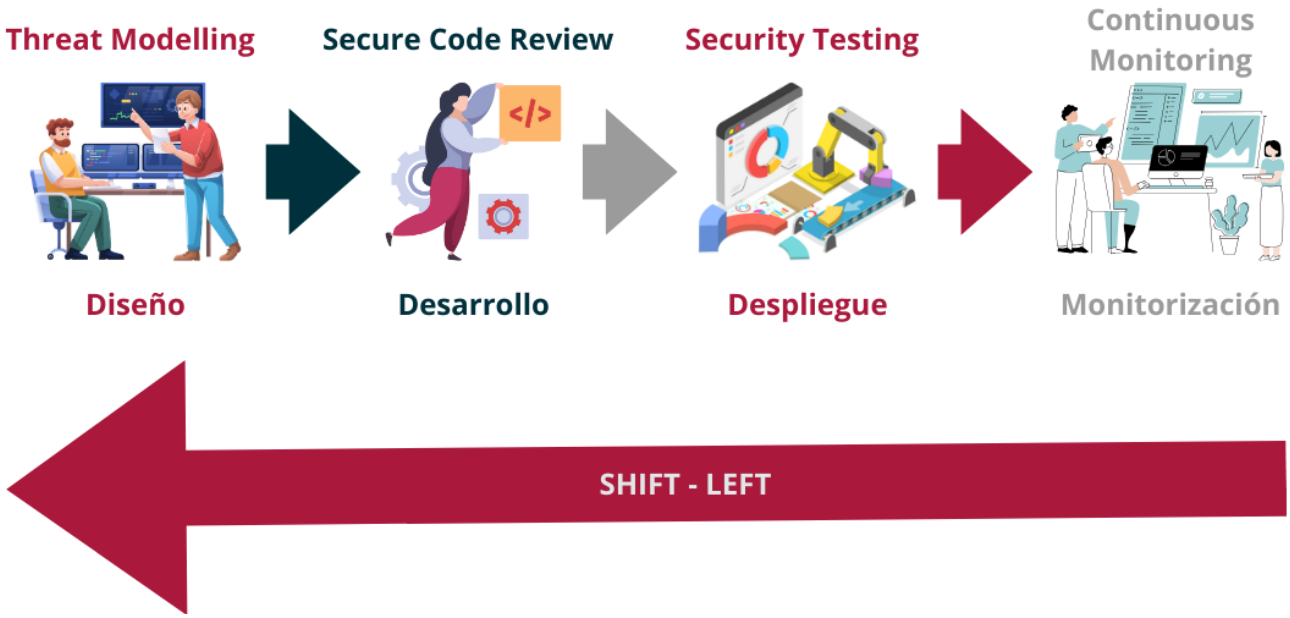


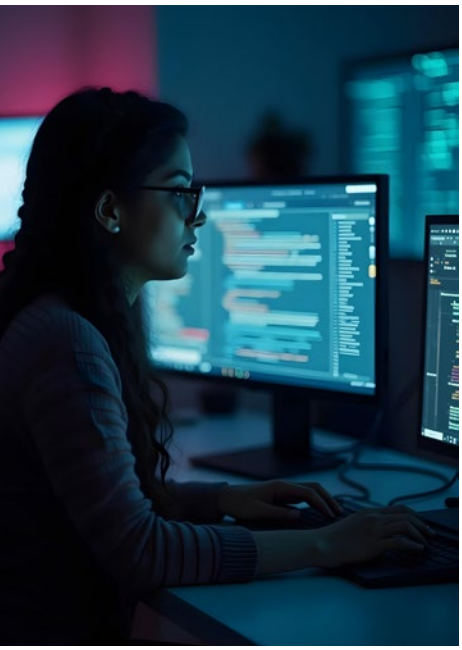
Imagen 1. ***Actividades de Seguridad por fase del SDLC***

1.3. Qué es DevSecOps – *Shift-Left*

Hoy en día, DevOps permite a cualquier organización desplegar cambios en los entornos de producción a un ritmo vertiginoso. Uno de los grandes problemas que suceden en este contexto es la integración de medidas relacionadas con la seguridad en todo nuestro proceso de DevOps para realizar algunas pruebas automatizadas. Así que considerar la cultura DevSecOps o DevOps seguro nos ayuda a promover la estrategia de seguridad *Shift-Left* en nuestra empresa, al menos en el departamento técnico.

Antes de proporcionar una definición de lo que es DevsecOps tenemos que entender que en los entornos de desarrollo corporativos DevOps aúna el desarrollo y las operaciones de software con el objetivo de acortar los ciclos de desarrollo, permitiendo que las organizaciones sean ágiles y mantengan el ritmo de la innovación alto al tiempo que aprovechan la tecnología y las prácticas nativas de la nube para el desarrollo del software. La industria ha adoptado estas prácticas para desarrollar y desplegar software en entornos operativos, a menudo sin una plena comprensión y consideración de los requisitos de seguridad.

¿Qué es la estrategia de seguridad *Shift-Left*?



Como definición simple, la estrategia de seguridad *Shift-Left* es una forma o solución para integrar la seguridad como parte de nuestro proceso de desarrollo y considerar la seguridad desde los pasos iniciales del diseño de aplicaciones o sistemas. En otras palabras, la seguridad es responsabilidad de todos los que trabajan en el proceso de desarrollo y explotación del software. Por supuesto, la seguridad es una profesión y necesitamos personas altamente cualificadas para desempeñar funciones relacionadas con la seguridad; pero en este enfoque, diseñadores, arquitectos de software, desarrolladores, ingenieros DevOps, tienen corresponsabilidad necesaria con el equipo de ciberseguridad sobre la calidad de los productos en términos de seguridad.

El concepto de DevSecOps nos ayuda a que la seguridad se aborde como parte a integrar en todas las prácticas que se emplean en DevOps mediante la adopción de buenas prácticas de seguridad en el proceso de diseño, desarrollo de software, en la generación, el empaquetado, la distribución y el despliegue. DevSecOps es reconocido desde hace años como un facilitador y "acelerador" del desarrollo de capacidades de IA y ML. (Ver: Ref. SEI/2020 [6])[2]

> *El objetivo de esta guía será indicar con directrices prácticas la integración de las prácticas de seguridad en las metodologías de desarrollo. En ella, compañías tanto Administración pública como empresa privada, podrán encontrar la información básica para poder elegir y aplicar prácticas DevSecOps con el fin de mejorar, no sólo la seguridad de sus productos y servicios, sino, además, la eficiencia en la aplicación de controles y monitorización de aspectos clave de ciberseguridad.*

1.4. Beneficios de la integración de seguridad en DevOps

La implementación de DevSecOps en el Ciclo de Vida de Desarrollo de Software (SDLC) ofrece numerosos beneficios para las organizaciones y sus interesados clave (clientes, desarrolladores, usuarios, etc.). Podemos destacar:

Mejora de la seguridad del software

- **Detección temprana de vulnerabilidades:** Los problemas de seguridad se identifican y abordan en fases iniciales del desarrollo, reduciendo riesgos y ayudando no sólo a detectar más o mejor con análisis continuos, sino a parchear de una manera mucho más rápida, pues los entornos están preparados para despliegues ágiles.
- **Mejoras no sólo en desarrollo sino en infraestructura:** La seguridad general se mejora al identificar y reducir vulnerabilidades de forma continua. Aprovechar aspectos como Infraestructura como Código hace que el inventariado y monitorización sea mucho más eficiente y continuo.

2 https://insights.sei.cmu.edu/annual-reviews/2020-year-in-review/devsecops-speeds-artificial-intelligence-and-machine-learning-capability/?utm_source=chatgpt.com

Reducción de costes

- **Menores costes de resolución de problemas e incidentes:** Solucionar problemas de seguridad en etapas tempranas es más económico que hacerlo en producción o en etapas finales.
- **Disminución de costes globales:** La identificación temprana de problemas lleva a una reducción en los costes totales de desarrollo y seguridad.

Aceleración del desarrollo

- **Mayor velocidad de entrega:** La automatización de pruebas de seguridad y la eliminación de cuellos de botella aceleran el proceso de desarrollo.
- **Reducción del tiempo de comercialización ("Time 2 Market"):** La integración de la seguridad desde el inicio permite ciclos de desarrollo más rápidos sin comprometer la seguridad. Enfoques tradicionales donde la seguridad llegaba "al final" hacían que los proyectos se demorasen por resolución de vulnerabilidades o saliesen al mercado asumiendo riesgos que en un entorno DevSecOps son innecesarios.

Mejora de la colaboración

- **Entorno más colaborativo:** Se fomenta una mejor comunicación entre los equipos de desarrollo, operaciones y seguridad.
- **Responsabilidad compartida:** Todos los equipos se involucran en la seguridad, eliminando fricciones y tensiones entre velocidad y seguridad.
- **Programas como Security Champions:** Fomentan aún más el acercamiento de departamentos, en principio, ajenos a ciberseguridad a la realidad del contexto de amenazas y la importancia del área de protección.

Cumplimiento normativo

- **Facilitación del cumplimiento:** Ayuda a cumplir con regulaciones y normas de seguridad y/u otras relacionadas como GDPR, HIPAA, ISO 27001 y ENS.

Automatización y eficiencia

- **Reducción de errores humanos:** La automatización de pruebas de seguridad disminuye los errores manuales.
- **Mejora en la respuesta a incidentes:** Se logra un tiempo de respuesta más corto ante problemas de seguridad.

Valor para el cliente

- **Aumento del valor percibido:** La entrega de software más seguro y de mayor calidad mejora la satisfacción del cliente.
- **Minimización de interrupciones:** Los usuarios experimentan menos problemas una vez que la aplicación está en producción.

Impacto reputacional para la organización

- Una organización con aplicaciones y productos software seguros, genera mayor confianza en sus clientes, usuarios, empleados, proveedores, etc. y tiene mejor imagen y reputación en el mercado.

Adicionalmente se considera que DevSecOps también abarca no solo la seguridad de los desarrollos propios, sino, además, la seguridad en la cadena de suministro de software.

La mayoría del software actual depende de uno o más componentes creados por terceros, pero las organizaciones a menudo tienen poca o ninguna visibilidad y comprensión de cómo estos componentes de software se desarrollan, integran y despliegan. Será necesario, por tanto, conocer las prácticas utilizadas para garantizar la seguridad de los componentes que lo forman. A través de buenas prácticas que se definen y desarrollan con DevSecOps se puede identificar, evaluar y mitigar el riesgo de ciberseguridad para la cadena de suministro de software.

Todos estos beneficios, hace que la adopción de DevSecOps transforma la seguridad de pasar de percibirse como un obstáculo potencial a un componente integral del desarrollo, mejorando la calidad del software, la eficiencia operativa y la postura de seguridad general de la organización. En resumen, convirtiendo a la seguridad del software en un activo de valor para la organización.

1.5. Enfoque del trabajo, hoja de ruta

DevSecOps no es una tecnología, un proceso o un sistema. Es todo un entorno sociotécnico que abarca a las personas que desempeñan ciertos roles, los procesos que cumplen y la tecnología utilizada para proporcionar una capacidad que da como resultado un producto o servicio relevante para satisfacer una necesidad.

> *No existe un proceso único que sirva para todos. Cada canal de DevSecOps debe adaptarse para satisfacer las necesidades de un programa en particular y debe evolucionar a medida que cambian las necesidades de la organización.*

Esta guía pretende dar un enfoque práctico, independiente de fabricantes o herramientas, para que pueda ser de utilidad en cualquier entorno de desarrollo moderno. En ella, se detallan las distintas prácticas y medidas de seguridad que se pueden integrar en cada una de las siguientes fases comunes en metodologías DevOps.

En las siguientes páginas, como si de un proceso de desarrollo se tratase, se podrá viajar por las diferentes etapas de un pipeline *DevSecOps*, estableciendo en cada una los aspectos básicos y ejemplos ilustrativos de lo que debe ser la integración de la ciberseguridad en los ámbitos descritos.

1. **Diseño y planificación**
   a. Fase 0: Un cambio cultural
   b. Fase 1: Seguridad por diseño y por defecto. Threat Modelling y Defensa en profundidad
2. **Desarrollo**
   a. Fase 2: Entornos de desarrollo
   b. Fase 3: Source Code Management (SCM)
3. **Integración y testeo**
   a. Fase 4: Automatización. CI/CD pipelines
   b. Fase 5: Open Source
4. **Despliegue**
   a. Fase 6: Despliegue. Infraestructura como Código y Seguridad
5. **Operación y monitorización continua**
   a. Fase 7: Monitorización. Application Security Posture Management (ASPM)

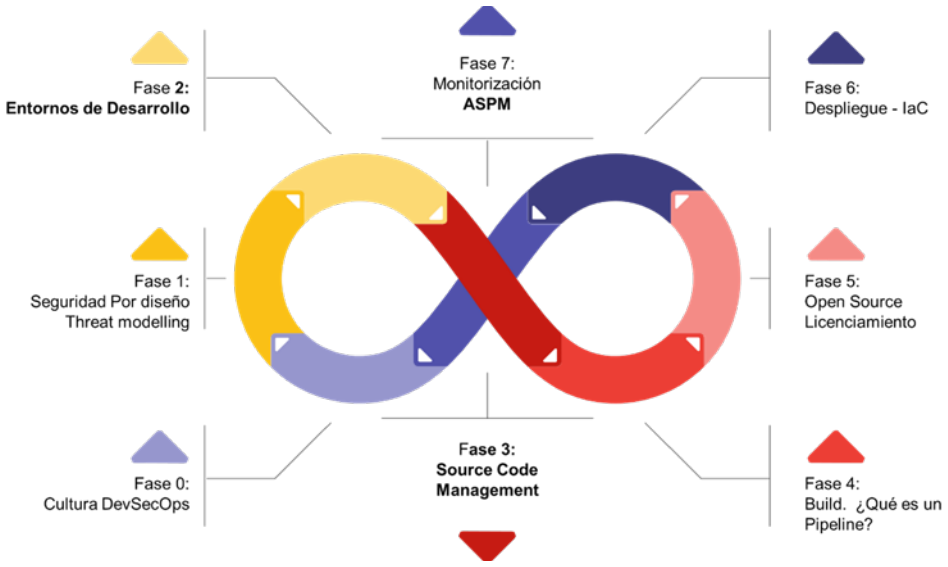
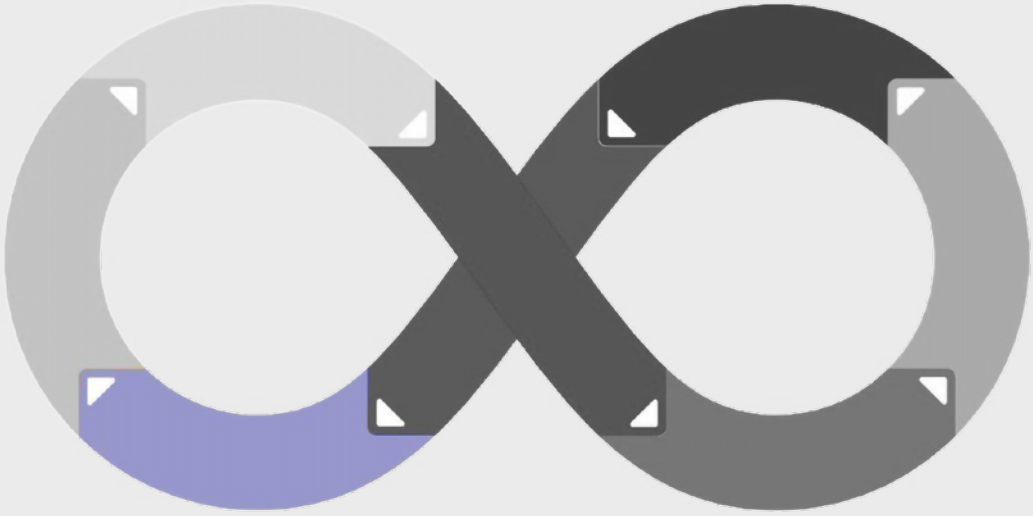


Imagen 2: ***Ilustración del ciclo completo de integración y despliegue continuo***

2. Fase 0: Un cambio cultural.



2.1. Retos organizacionales

En línea con lo presentado, en la edición anterior de esta guía[3], los principales retos organizativos para adoptar DevSecOps siguen siendo por lo general tener capacidad para afrontar los cambios y gestionar/adaptarse a nuevas formas de trabajo y herramientas. A continuación, se destacan varios ámbitos que son relevantes.

Cambio cultural

- Superar la **resistencia al cambio** de los equipos acostumbrados a modelos tradicionales.
- Pasar de un control completo del ámbito de actuación a un modelo basado en **colaboración e intercambio de información.**
- Cambiar la percepción de que las nuevas metodologías son una fiscalización del trabajo, en lugar de una **oportunidad de mejora.**

Aprendizaje y formación continua

- Adquirir nuevos conocimientos y habilidades en herramientas y metodologías ágiles, **flujos de trabajos automatizables y automatizados,** etc.
- Capacitar al personal en el uso de **nuevas tecnologías** como la infraestructura como código, IA para ciberseguridad, etc.

Adaptación de procesos

- Implementar metodologías ágiles como base para DevOps y DevSecOps, y nuevos enfoques o necesidades de integración. Por ejemplo, con la aparición de MLOps y LLMOps, cabe preguntarse o anticiparse de que pronto existirán enfoques como "MLSecOps" y/o "LLMSecOps", o más bien, ¿puede DevSecOps dar respuestas a estas nuevas realidades?
- **Integrar** equipos de desarrollo y operaciones, juntamente con profesionales de seguridad que tradicionalmente trabajaban por separado.
- Adoptar **nuevos mecanismos de aprovisionamiento y automatización.**

Innovación tecnológica constante

- Incorporar plataformas que permitan la automatización de tareas, el uso de IA y agentes, gemelos digitales, soluciones de simulación, etc.
- Implementar herramientas para definir infraestructura como código.
- El acceso a soluciones IA que se integran dentro de otras aplicaciones, flujos de trabajos, o simplemente se usan como apoyo en los dispositivos propios (del BYOD "bring your own device" à al BUYOAI: "bring or use your own AI") plantean nuevos retos de seguridad que implican tanto a desarrolladores como a usuarios.

3 https://master.ismsforum.es/wp-content/uploads/2024/09/guia-devsecops-v41690197585.pdf

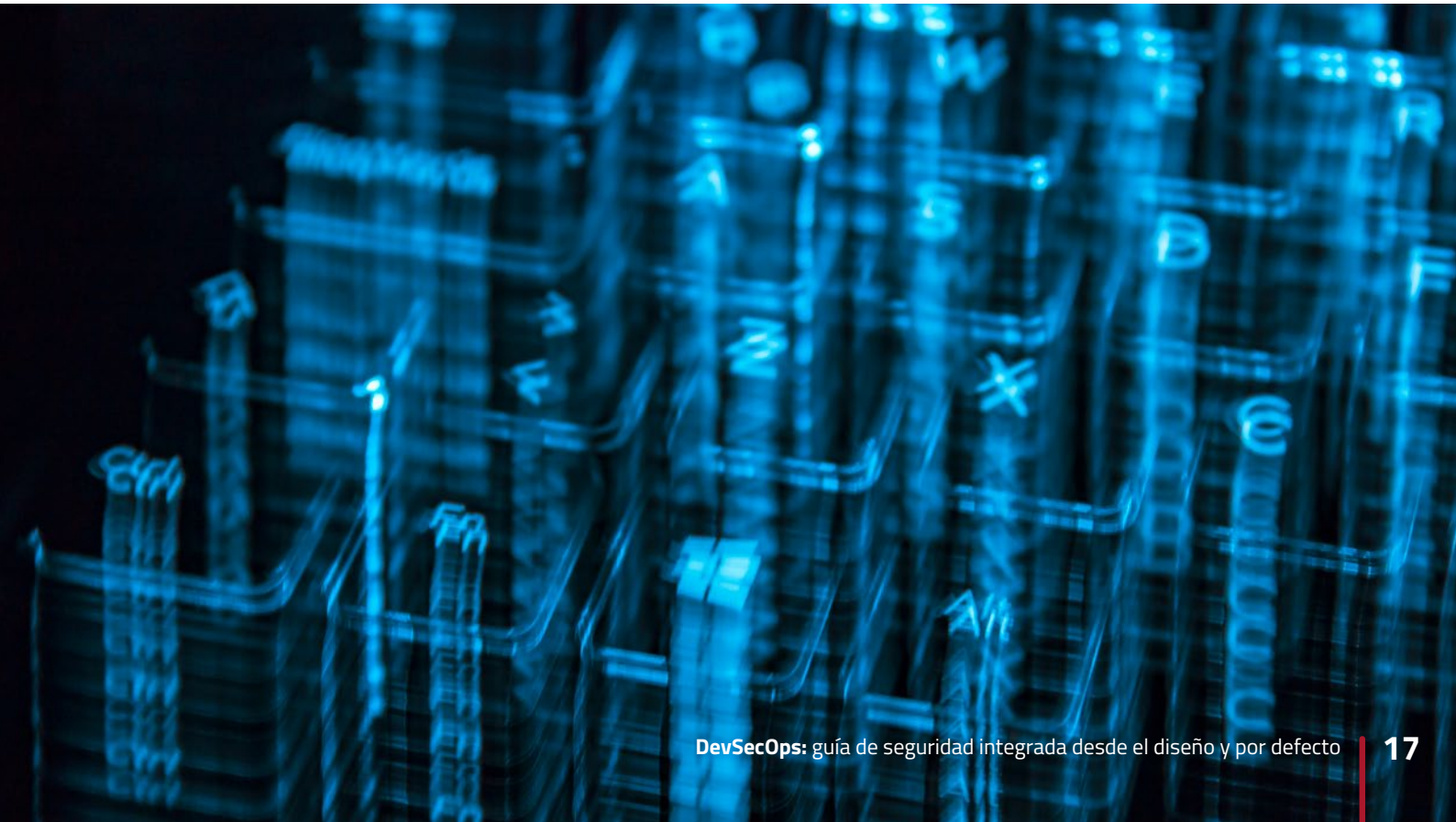
Presión por resultados rápidos

- Responder a la necesidad de mejorar los tiempos de puesta en producción de nuevas soluciones.
- Adaptarse a procesos de digitalización y transformación agresivos que requieren mayor velocidad de respuesta.

Nuevos retos, nuevos roles, ... y otros nuevos retos

- Ante el avance imparable de la tecnología, las normativas y regulaciones relacionadas – que incluyen y/o impactan en requisitos de seguridad, privacidad, etc. las organizaciones se plantean definir, adoptar y desplegar "nuevos" roles. Ejemplo, CAIO (Chief AI Officer), Prompt Engineer, FinOps engineer, etc. pero los nuevos roles también traen con ellos nuevos retos organizativos.
- Durante años el avance y adopción de disciplinas emergentes como el Citizen Development (CD) ha sido relativamente lento, pero con la creciente adopción de la IA como agente *empoderador* de desarrolladores que no son necesariamente programadores profesionales, aparecen nuevos retos de seguridad, arquitectura, etc.
- Otros roles y responsabilidades asociados con nuevas regulaciones (ej. RAI/LAI. Reglamento Europea de IA), traen consigo los retos propios de adoptar un nuevo marco regulatorio, así como la interrelación de dichos roles (sean nuevo o existentes) con otras funciones de la organización.

Es bien conocido que estos cambios no son sencillos de implementar y requieren un esfuerzo significativo y centrado en términos de adaptación organizativa, aprendizaje y superación de la resistencia al cambio. Y son precisamente, estos aspectos los que pueden ayudar a establecer las líneas de actuación para gestionar estos retos.



2.2. Estrategias para abordar los retos

Como organización, para poder hacer una implantación efectiva de un modelo DevSecOps de gestión de los desarrollos y productos, se deben superar los retos descritos con la colaboración activa de todas las áreas y capas del organigrama.

En este sentido, se necesitará:

Cambio cultural y de mentalidad

- **Comunicar claramente los objetivos** y expectativas de la organización respecto a DevSecOps.
- Fomentar una **cultura de concienciación** sobre seguridad en todos los miembros involucrados.
- Promover la **responsabilidad compartida,** donde todo el equipo asume el control de calidad, integración de código y seguridad.

Mejora de la colaboración

- Fomentar la colaboración entre los equipos de desarrollo, operaciones y seguridad para compartir información y adoptar buenas prácticas.
- Implementar una comunicación y colaboración continua para solucionar vulnerabilidades y alinear esfuerzos hacia objetivos comunes.

Capacitación y desarrollo de habilidades

- Incluir expertos en seguridad en el equipo para mejorar la detección de vulnerabilidades y el desarrollo de estrategias.
- Proporcionar formación en buenas prácticas de seguridad – tanto básica como intermedia o avanzada según las necesidades – a los desarrolladores para facilitar una colaboración más eficiente.

Adaptación de procesos y herramientas

- Evaluar los procesos, herramientas y cultura DevOps actuales para identificar áreas de mejora en seguridad.
- Revisar y actualizar las políticas de seguridad para alinearlas con los principios DevSecOps.
- Implementar herramientas orientadas a evaluar la infraestructura del código, realizar pruebas de seguridad y analizar la composición del software.

Enfoque gradual y medición

- Adoptar un enfoque gradual e incremental, comenzando con proyectos piloto que ayuden a demostrar el valor a corto plazo y así, ir ganando apoyos, interés y compromiso.
- Establecer métricas clave para evaluar y mejorar periódicamente la implementación de DevSecOps.

Apoyo de la dirección

- Asegurar el apoyo del equipo líder de la organización para fomentar y liderar el cambio "desde arriba". Este apoyo no sólo está en su participación directa, sino en facilitar los recursos y decisiones para que se puedan implementar acciones por parte de otros miembros de la organización. Por ejemplo, establecer y desplegar el *rol de Security Champion.* Ver sección 2.3.
- Combinar el apoyo de la dirección con un equipo muy involucrado desde otros niveles de la organización, para ejecutar la transformación.

Estas estrategias ayudarán a las organizaciones a superar la resistencia al cambio, mejorar la colaboración entre equipos y establecer una cultura de seguridad integrada en todo el ciclo de vida del desarrollo de software.

Gestión del cambio organizacional participativo

- Involucrar a los empleados en el proceso de cambio desde el principio, solicitando su opinión y sugerencias.
- Crear grupos de trabajo interdepartamentales para diseñar e implementar los cambios necesarios.
- Fomentar la experimentación y el aprendizaje continuo, permitiendo a los equipos probar nuevas ideas y aprender de los errores.

Gestión explícita de la resistencia al cambio

- Identificar y abordar las preocupaciones específicas de los empleados sobre la adopción de DevSecOps.
- Proporcionar apoyo emocional y recursos para ayudar a los empleados a adaptarse al cambio.
- Celebrar los éxitos tempranos y compartir historias de transformación positiva para inspirar confianza en el proceso. Ver punto siguiente sobre storytelling

Liderazgo transformacional

- Capacitar a los líderes en habilidades de gestión del cambio, influencia y liderazgo transformacional.
- Fomentar un estilo de liderazgo que inspire y motive a los equipos a adoptar nuevas formas de trabajo.
- Hay que asegurar que los líderes modelan el comportamiento deseado y demuestran compromiso con la transformación hacia DevSecOps.

Creación de una cultura de aprendizaje continuo

- Fomentar la participación en comunidades de práctica internas y externas relacionadas con DevSecOps.
- Reconocer y recompensar el aprendizaje y la mejora continua en todos los niveles de la organización.

Estas estrategias adicionales, combinadas con las mencionadas anteriormente, pueden ayudar a las organizaciones a gestionar de manera más efectiva el cambio cultural y organizacional necesario para adoptar DevSecOps. La clave está en crear un enfoque holístico que aborde tanto los aspectos técnicos como los humanos de la transformación.

2.3. Introducción al rol de los Security Champions

Uno de los principales problemas que encuentran las compañías en su persecución por analizar, detectar y solucionar vulnerabilidades en los desarrollos o la infraestructura es la falta de perfiles de expertos para estas tareas. Una respuesta a la falta de escalabilidad que supone por un lado la necesidad de incremento de esfuerzos en ciberseguridad y por otro la escasez de talento en el área es el concepto de los programas de "Security Champions".

Los Security Champions son integrantes de los equipos de desarrollo que actúan como enlaces entre las áreas de desarrollo y seguridad, promoviendo prácticas seguras, fomentando una cultura de seguridad en la organización y actuando de balizas u ojos con sensibilidad por temas de ciberseguridad integrados en los diferentes equipos, que apoyados en el equipo de ciberseguridad amplifican su capacidad de detección y acción ante posibles inconvenientes.

Definición y responsabilidades de los Security Champions

Las responsabilidades de los Security Champion no tienen por qué ser las mismas para todas las organizaciones, sino que se pueden adaptar según el tipo de organización y la madurez del modelo de desarrollo que se haya implementado.

Las responsabilidades más comunes de los Security Champion son las siguientes:

- Identificar y comunicar riesgos de seguridad en el desarrollo de aplicaciones.
- Orientar al equipo de desarrollo sobre requisitos y buenas prácticas de desarrollo seguro.
- Servir de enlace entre los equipos de desarrollo y seguridad, facilitando la comunicación y colaboración.
- Participar activamente en las formaciones y mantenerse al día sobre las últimas tendencias y amenazas en seguridad.

Importancia de los Security Champions en la cultura DevSecOps

La incorporación de Security Champions es esencial para fomentar una cultura de seguridad en la organización.

Al estar integrados en los equipos de desarrollo, pueden detectar y abordar problemas de seguridad desde las primeras fases del desarrollo, reduciendo costes y esfuerzos posteriores. También deben asegurarse de que las prácticas de seguridad se consideren en cada decisión de diseño y desarrollo y facilitar la adopción de herramientas y procesos de seguridad en los flujos de trabajo existentes.

Implementación de un programa de Security Champions

Para establecer un programa efectivo de Security Champions, la organización puede seguir estos pasos:

1. **Selección de candidatos.** Identificar a personas dentro de los equipos de desarrollo que muestren interés y aptitud en temas de seguridad. Esta decisión debería tomarse de manera consensuada entre el departamento de Seguridad y los departamentos de Tecnología.
2. **Capacitación.** Proporcionar recurrentemente formación especializada en seguridad de aplicaciones y prácticas de desarrollo seguro además de soporte para la resolución de dudas.

3. **Definición de roles y responsabilidades.** Establecer claramente las expectativas y funciones de los Security Champions dentro de los equipos.
4. **Fomento de la colaboración.** Crear espacios y oportunidades para que los Security Champions compartan conocimientos y experiencias, tanto entre sí como con otros equipos. Pueden ser reuniones de trabajo periódicas o grupos de chat.
5. **Reconocimiento y apoyo.** Reconocer el esfuerzo de los Security Champions y proporcionarles los recursos necesarios para desempeñar su papel de manera efectiva.

Beneficios de contar con Security Champions

La integración de Security Champions aporta múltiples ventajas a la organización, entre ellas:

- **Mejora en la calidad del software.** Al incorporar prácticas de seguridad desde el inicio, se desarrollan aplicaciones más robustas y resilientes.
- **Reducción de vulnerabilidades.** La identificación temprana de riesgos permite mitigar posibles amenazas antes de que se conviertan en problemas mayores.
- **Mejora en la comunicación entre equipos.** La colaboración entre las áreas de desarrollo y seguridad se fortalece, agilizando la resolución de problemas.
- **Optimización de tiempos y recursos.** Al abordar la seguridad de manera proactiva, se minimizan los costes asociados a correcciones posteriores, la carga operativa del equipo de seguridad y posibles incidentes de seguridad.

> *Los Security Champions pueden desempeñar un papel clave en la implementación de la cultura DevSecOps dentro de la organización. Su presencia favorece drásticamente a que la seguridad sea una consideración constante, contribuyendo al desarrollo de software más seguro y a la creación de una cultura organizacional que valora y prioriza la protección de sus activos y datos.*

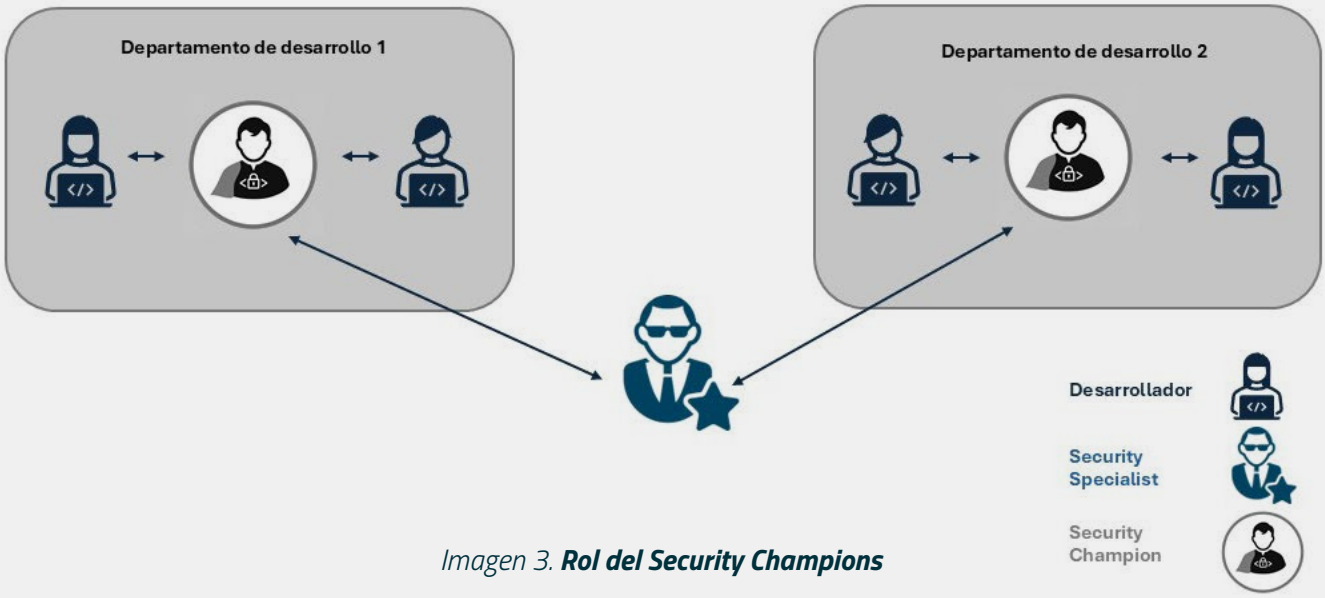
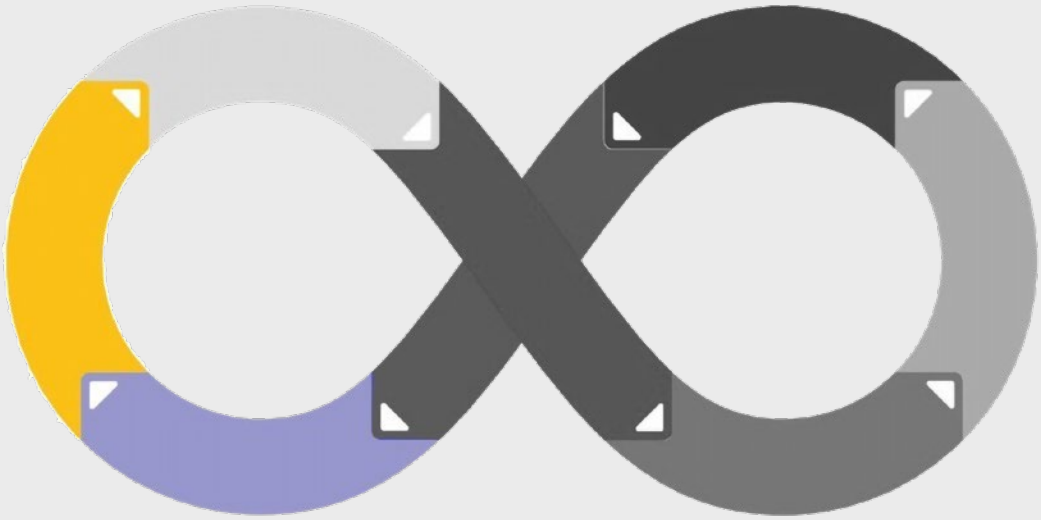
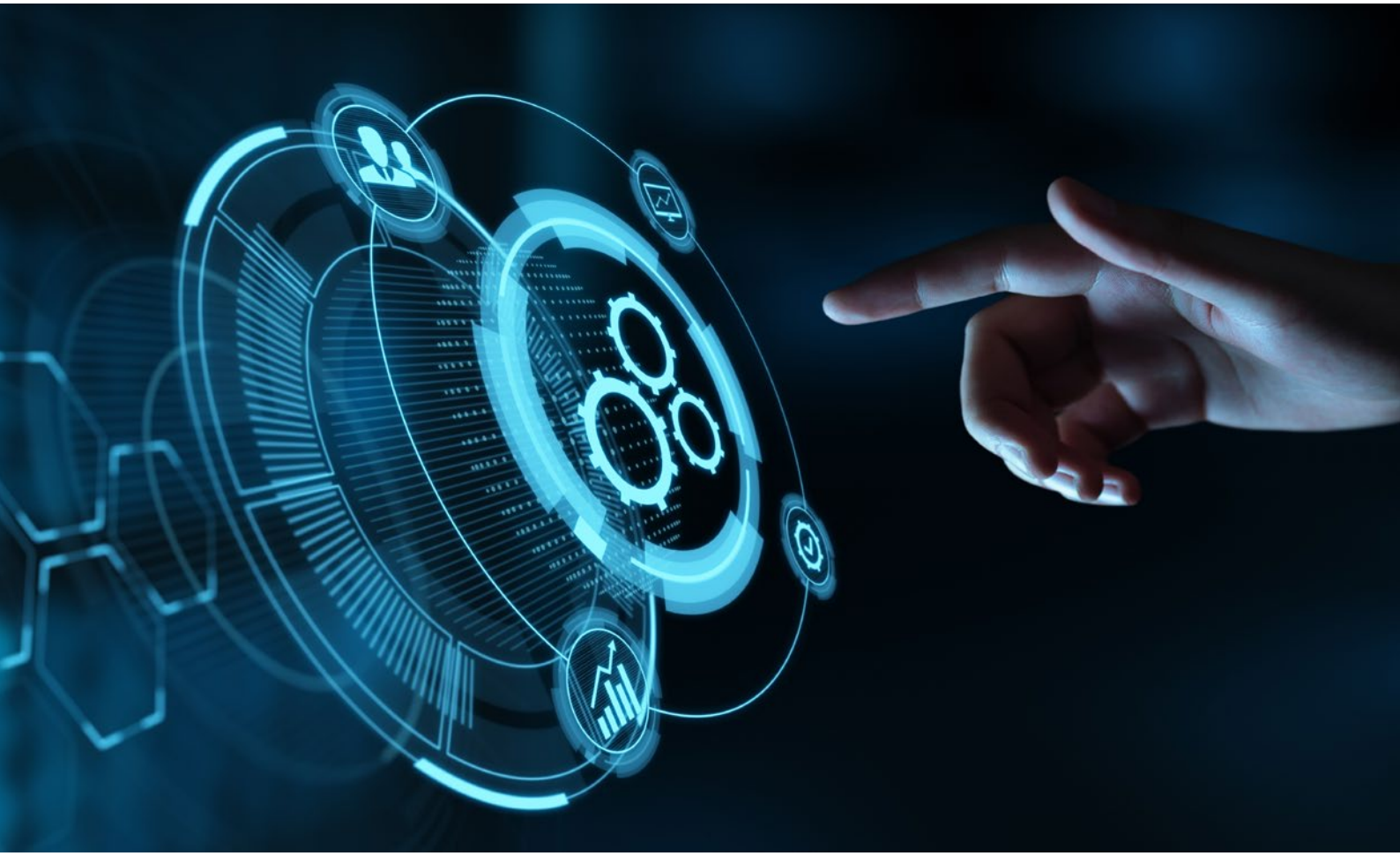


Imagen 3. ***Rol del Security Champions***

3 Fase 1: Seguridad por diseño y por defecto.





Como se ha visto en los puntos anteriores, la seguridad integrada en el ciclo de vida de desarrollo de software no es solo una necesidad técnica, sino una estrategia empresarial fundamental para el éxito sostenible.

El concepto de seguridad por diseño (también conocida como seguridad desde el diseño) y por defecto, o security by design and by default en inglés, busca redefinir el proceso clásico de construcción de productos tecnológicos en general y el desarrollo de aplicaciones de software en particular, en el que la seguridad se trata como un añadido, normalmente incorporado en las etapas finales del mismo. Esta redefinición persigue que la seguridad esté presente de manera intrínseca y por defecto en todas las fases del ciclo de desarrollo, desde la propia concepción y diseño hasta su puesta en producción, para combatir de manera proactiva las distintas ciberamenazas que pueden afectar al producto en cuestión.

La Agencia de Seguridad de Infraestructura y Ciberseguridad de EE.UU. (CISA), junto con otras organizaciones internacionales, subrayó la necesidad de abordar el problema de sistemas "vulnerables por diseño" en su publicación *"Secure-by-Design. Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Secure by Design Software"*[4] . Este tipo de sistemas presentan debilidades que pueden ser explotadas debido a la falta de consideraciones de seguridad durante su desarrollo. Adoptar un enfoque de seguridad por diseño y por defecto permite subsanar o minimizar esas debilidades, reduciendo además una carga de responsabilidad excesiva sobre los usuarios y clientes finales y facilitando la implementación de medidas de protección automáticas y confiables.

De esta manera, un sistema seguro por defecto está diseñado para minimizar los riesgos sin necesidad de que el usuario realice configuraciones adicionales, incluyendo medidas como la eliminación de contraseñas predeterminadas, la activación de medidas de protección por defecto en los procesos de instalación y la reducción de la complejidad en la configuración de los parámetros de seguridad del sistema.

4 https://www.cisa.gov/resources-tools/resources/secure-by-design

Existen muchas referencias elaboradas por entidades u organizaciones de referencia sobre cómo se puede aplicar el concepto de seguridad por diseño y por defecto, desde la CISA citada anteriormente, hasta OWASP[5] o RedHat[6], pasando por otras muchas. De las distintas definiciones y enfoques que hacen cada una de ellas sobre este concepto, podrían enumerarse los siguientes principios como la base para la implementación de la seguridad por diseño y por defecto:

1. **Minimizar la superficie de ataque:** cuanto mayor sea la cantidad de puntos de acceso y funciones expuestas, mayor será el riesgo de que existan debilidades o vulnerabilidades que los atacantes puedan explotar. Por ello, es necesario reducir la superficie de ataque, eliminando elementos innecesarios en la aplicación, restringiendo accesos y minimizando funcionalidades no esenciales.

   Por ejemplo, un sistema que solo necesita exponer un servicio de autenticación no debería exponer endpoints innecesarios para la depuración de errores o servicios internos no documentados.

2. **Establecer una configuración segura por defecto:** los sistemas y aplicaciones deben configurarse de manera segura desde el inicio, sin requerir que los usuarios habiliten opciones de seguridad manualmente.

   Por ejemplo, un gestor de contenidos (CMS) debería tener permisos de usuario restringidos desde la instalación, deshabilitando por defecto opciones que aumentan el riesgo como la ejecución de scripts desde el navegador.

3. **Aplicar el principio de mínimo privilegio:** un usuario o proceso solo debe tener los permisos estrictamente necesarios para cumplir su función, evitando que se puedan producir usos abusivos y reduciendo el impacto de explotación de una posible vulnerabilidad.

   Por ejemplo, un usuario estándar en una aplicación solo debería poder consultar su información, sin tener acceso a funciones administrativas.

4. **Implementar una defensa en profundidad:** la seguridad debe aplicarse en múltiples capas para que si una falla, otras continúen protegiendo el sistema.

   Por ejemplo, un servidor web puede protegerse mediante un firewall que controla el origen de la petición, la aplicación expuesta por el servidor puede implementar un mecanismo de autenticación multifactor, la información de la aplicación puede estar cifrada y además puede existir una monitorización que alerte de la existencia de actividad anómala en la aplicación o servidor.

5. **Fallar de manera segura (fail securely):** cuando un sistema encuentra un error o falla, debe hacerlo de manera controlada, sin exponer información sensible o comprometer la seguridad de este.

   Por ejemplo, si un intento de autenticación en un servicio falla, el mensaje de error devuelto debería ser genérico ("Credenciales incorrectas"), en lugar de indicar específicamente si el usuario no existe o la contraseña es incorrecta.

6. **Validar las entradas de la aplicación por defecto:** nunca se debe asumir que la información recibida externamente por la aplicación es segura por defecto, debiendo siempre validar y sanitizar los datos recibidos.

   Por ejemplo, una aplicación que recibe datos de un servicio externo debe verificar que la información no contenga código malicioso o datos manipulados antes de procesarla.

5 https://owasp.org/www-project-developer-guide/release/
6 https://www.redhat.com/en/blog/security-design-security-principles-and-threat-modeling

7. **Separación de funciones:** las funciones críticas del sistema deben dividirse entre diferentes usuarios o procesos para evitar que una sola entidad tenga control total.

> *Por ejemplo, en una aplicación empresarial, el mismo usuario no debería poder aprobar pagos y modificar cuentas bancarias en el sistema de facturación.*

8. **Evitar la seguridad por oscuridad:** la seguridad no debe basarse únicamente en el supuesto desconocimiento para el usuario externo de cómo puede funcionar un sistema internamente, ya que dicho funcionamiento podría ser filtrado u obtenido mediante diversas técnicas, por ejemplo, a través del reversing[7]. Es preferible utilizar estándares abiertos probados en lugar de algoritmos o métodos propietarios no auditados.

> *Por ejemplo, usar cifrado AES-256 en lugar de un algoritmo de cifrado creado internamente sin revisión de seguridad.*

9. **Simplificar la seguridad (Keep security simple):** cuanto más complejo es un sistema, más propenso es a la aparición de errores o debilidades no controladas y desconocidas. Manteniendo una arquitectura y configuraciones de seguridad sencillas, mejor será la implementación y administración de esta.

> *Por ejemplo, una política de contraseñas que obliga al usuario a renovarla cada semana puede hacer que los usuarios terminen anotándolas en lugares inseguros. Una alternativa más simple y segura es implementar autenticación multifactor (MFA).*

10. **Corregir los problemas de seguridad adecuadamente:** cuando se descubre una vulnerabilidad, la solución de esta debe abordar la causa raíz y no ir enfocada únicamente a la aplicación de un parche temporal.

> *Por ejemplo, si se detecta una vulnerabilidad de inyección SQL en una aplicación, la solución correcta no sería bloquear únicamente ciertos caracteres, sino que deberían implementarse consultas parametrizadas para evitar la inyección utilizando otros mecanismos.*

La integración de una estrategia de seguridad por diseño y por defecto basada en estos principios de seguridad y aplicada en cada etapa del ciclo de vida del desarrollo de la aplicación constituye un paso esencial para fortalecer la ciberseguridad, minimizando el riesgo de ataques y reduciendo los costos asociados a la corrección de fallos de seguridad en etapas más tardías del ciclo de vida o incluso tras la implementación productiva.

Además, permite a las organizaciones cumplir con normativas y regulaciones aplicables, ya que pueden integrar esos requerimientos en todas las etapas del ciclo de vida del desarrollo, surgiendo conceptos relacionados como la privacidad por defecto.

7 Reversing: técnica empleada para analizar y descomponer el software o el hardware de un sistema informático para comprender su funcionamiento y encontrar posibles brechas de seguridad o vulnerabilidades.

3.1. Threat Modeling

Según el Threat Modeling Manifesto[8], el modelado de amenazas consiste en analizar las representaciones de un sistema para poner de relieve los problemas relacionados con sus características de seguridad y privacidad. A alto nivel, cuando se lleva a cabo un modelado de amenazas, deben plantearse cuatro preguntas clave:

- **¿En qué estamos trabajando?**
- **¿Qué puede ir mal?**
- **¿Qué vamos a hacer al respecto?**
- **¿Hemos hecho un trabajo suficientemente bueno?**

Ampliando la definición del modelado de amenazas hacia una definición más formal, se puede representar como un proceso que parte del análisis de la arquitectura, diseño y funcionalidad de una aplicación, con el objetivo de identificar, enumerar y priorizar posibles amenazas potenciales a la seguridad de un sistema, permitiendo implementar contramedidas para la mitigación de las mismas desde la propia fase de diseño de la aplicación.

> *Este proceso permite a las organizaciones aplicar un enfoque de seguridad por diseño y por defecto en el desarrollo de sus aplicaciones, garantizando que la integridad, confidencialidad y disponibilidad son características intrínsecas de las mismas.*

La implementación efectiva de un modelado de amenazas dentro del ciclo de vida de una aplicación se puede realizar siguiendo alguna de las diferentes metodologías que se recogen en este apartado, o incluso desarrollar una propia según la naturaleza de la organización, pero todas ellas coinciden en la ejecución de una serie de pasos estructurados:

1. **Definición de Objetivos y Alcance:** clarificar qué sistemas o aplicaciones serán evaluados y cuáles son los objetivos de seguridad.
2. **Creación de Diagramas de Arquitectura:** representar visualmente la estructura del sistema, incluyendo componentes, flujos de datos y límites de confianza.
3. **Identificación de Amenazas:** utilizar una o varias metodologías para detectar posibles amenazas en el sistema.
4. **Evaluación de Riesgos:** analizar la probabilidad e impacto de cada amenaza identificada.
5. **Desarrollo de Estrategias de Mitigación:** proponer e implementar medidas para reducir o eliminar los riesgos.
6. **Revisión y Actualización Continua:** mantener el modelo de amenazas actualizado frente a cambios en el sistema o en el panorama de amenazas.

8 https://www.threatmodelingmanifesto.org/

Metodologías principales para la identificación de amenazas

Existen diversas metodologías para realizar el modelado de amenazas, cada una con enfoques y aplicaciones específicas. A continuación, se describen las principales características de las metodologías más reconocidas:

1. STRIDE

La metodología para el modelado de amenazas STRIDE (S.T.R.I.D.E.), fue desarrollada inicialmente por Microsoft, basándose en la asignación de amenazas pertenecientes a seis categorías, cuyas iniciales conforman el nombre de la metodología:

- **Spoofing (Suplantación de identidad):** acceso no autorizado mediante la falsificación de identidad.
- **Tampering (Manipulación de datos):** alteración no autorizada de datos.
- **Repudiation (Repudio):** negación de acciones realizadas.
- **Information Disclosure (Fuga de información):** exposición no autorizada de información.
- **Denial of Service (Denegación de servicio):** interrupción de la disponibilidad de servicios.
- **Elevation of Privilege (Elevación de privilegios):** obtención de permisos superiores de forma ilegítima.

¿Cuándo usar la metodología STRIDE?

- Cuando se desarrolla una aplicación web o móvil y se necesita un análisis estructurado de amenazas.
- Para productos de software que siguen un ciclo de desarrollo ágil y requieren una evaluación rápida de amenazas comunes.
- Cuando se trabaja con equipos de desarrollo que no tienen una amplia experiencia en ciberseguridad.

2. PASTA (Process for Attack Simulation and Threat Analysis)

PASTA es una metodología orientada al negocio que busca alinear las amenazas identificadas con los objetivos empresariales, proporcionando una visión integral que combina aspectos técnicos y estratégicos en la gestión de amenazas. Consta de siete etapas:

1. **Definición de Objetivos:** establecer metas de seguridad y alcance del análisis.
2. **Alcance Técnico:** delimitar el entorno tecnológico a evaluar.
3. **Descomposición de la Aplicación:** analizar detalladamente la arquitectura y componentes del sistema.
4. **Análisis de Amenazas:** identificar posibles amenazas y vectores de ataque.
5. **Análisis de Vulnerabilidades:** detectar debilidades explotables en el sistema.
6. **Modelado de Ataques y Simulación:** simular posibles ataques para evaluar su viabilidad.
7. **Análisis de Riesgo e Impacto:** evaluar el riesgo asociado a cada amenaza y su impacto potencial en el negocio.

¿Cuándo usar la metodología PASTA?

- Cuando una organización necesita alinear la seguridad con sus objetivos comerciales y regulaciones.
- Para grandes corporaciones con requisitos de cumplimiento normativo (ej. GDPR, ISO 27001, NIST).
- Para identificar amenazas avanzadas que podrían tener un impacto directo en el negocio.

3. VAST (Visual, Agile, and Simple Threat)

VAST es una metodología que busca integrar el modelado de amenazas en entornos ágiles de desarrollo. Se basa en la creación de modelos visuales que facilitan la identificación y comunicación de amenazas. VAST se enfoca en:

- **Escalabilidad:** adaptarse a proyectos de diferentes tamaños y complejidades.
- **Integración Ágil:** encajar en ciclos de desarrollo ágiles sin interrumpir el flujo de trabajo.
- **Simplicidad Visual:** utilizar diagramas claros que faciliten la comprensión y colaboración entre equipos.

¿Cuándo usar la metodología VAST?

- Cuando se trabaja en ciclos de desarrollo rápido con metodologías **ágiles** o **DevOps.**
- Para empresas que necesitan integrar el modelado de amenazas en su flujo de trabajo sin interrumpir la producción.
- Cuando se requiere una representación visual clara de los riesgos para facilitar la comunicación entre equipos de seguridad y desarrollo.

4. OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation)

Desarrollada por el Software Engineering Institute de la Universidad Carnegie Mellon, OCTAVE es una metodología centrada en activos que evalúa riesgos organizacionales más que tecnológicos, siendo útil para organizaciones que buscan una visión holística de sus riesgos de seguridad, alineando la gestión de amenazas con los objetivos estratégicos. Consta de tres fases:

1. **Construcción de Perfiles de Amenazas:** identificar activos críticos y posibles amenazas.
2. **Identificación de Vulnerabilidades de la Infraestructura:** evaluar debilidades en la infraestructura tecnológica.
3. **Desarrollo de Planes de Seguridad:** diseñar estrategias para mitigar riesgos identificados.

¿Cuándo usar la metodología OCTAVE?

- Cuando se desea analizar la seguridad a nivel organizacional, no solo en un sistema específico.
- Para instituciones gubernamentales, hospitales o empresas de infraestructura crítica que manejan datos sensibles.
- Cuando se quiere adoptar una estrategia de seguridad a largo plazo basada en riesgos.

5. CORAS

CORAS es una metodología europea que facilita el análisis de riesgos mediante modelos visuales basados en el Lenguaje Unificado de Modelado (UML). La fortaleza de CORAS radica en su enfoque visual, que facilita la comprensión y comunicación de riesgos entre diferentes stakeholders. Sus etapas incluyen:

1. **Definición del Alcance:** establecer límites y objetivos del análisis.
2. **Identificación de Riesgos:** utilizar diagramas para representar amenazas y vulnerabilidades.
3. **Estimación y Evaluación de Riesgos:** analizar la probabilidad e impacto de cada riesgo.
4. **Tratamiento de Riesgos:** proponer medidas para mitigar o eliminar riesgos.

¿Cuándo usar la metodología CORAS?

- Cuando se requiere una herramienta visual para comunicar riesgos de seguridad a directivos y partes no técnicas.
- Para proyectos con múltiples dependencias y sistemas interconectados.
- En sectores como telecomunicaciones, defensa o energía, donde la seguridad debe estar alineada con normas y regulaciones.

Las metodologías recogidas en este apartado son una referencia que puede ayudar a la organización a enfrentar la tarea de realizar un modelado de amenazas, pero es la propia organización la que debe hacer el ejercicio de analizar cuál es el enfoque que mejor encaja con sus procesos, cultura y gestión de la seguridad y el riesgo, pudiendo elegir una metodología, combinar varias o desarrollar una propia adaptada a las particularidades de la organización.

Ejemplo de realización de un modelado de amenazas

Para ver cómo se realiza un modelo de amenazas desde un enfoque más práctico, se muestra la aplicación de la metodología **STRIDE** al siguiente escenario: una aplicación web expuesta a internet cuyos datos se alimentan de una base de datos. Para ello, deben ejecutarse las siguientes fases:

1. Elaboración del DFD (Diagrama de Flujo de Datos)

El punto de partida para poder llevar a cabo el modelado de amenazas es elaborar el DFD, que incluirá los elementos involucrados y los datos procesados, almacenados e intercambiados por cada uno de ellos.

Este DFD debe diseccionar el flujograma de la aplicación o sistema en los siguientes tipos de elementos:

- **Actor:** usuario del sistema. Típicamente se referirá al usuario humano que utiliza el sistema o aplicación, pero puede referirse también a clientes como navegadores o dispositivos que hacen uso del sistema de una manera autónoma.
- **Data Store (Almacenamiento de datos):** sistemas cuya función es el almacenamiento de los datos manejados en la aplicación. Ejemplos: Bases de datos, File systems, LDAP, Cookies, Memory-Cache, etc.
- **Data Flow (Flujo de datos):** flujos que representan el intercambio de información entre los distintos elementos del DFD y la tecnología o protocolos utilizados para ello, como por ejemplo HTTPS, IPSEC o RPC.
- **Process (Proceso):** los procesos se refieren a todo lo que ejecute algún tipo de código. Ejemplos: Aplicación/Servicio Web, procesos del Sistema Operativo, VM/Host/Server, etc.

Además, cada elemento deberá identificarse dentro de distintas zonas de "confianza" para poder evaluar los riesgos y amenazas a los que se está expuesto, de manera que se localice el elemento dentro de Internet, Intranet, Cloud Pública, Cloud Privada, etc.

Para el escenario que se desarrolla en este ejemplo, un DFD genérico puede componerse con los siguientes elementos:

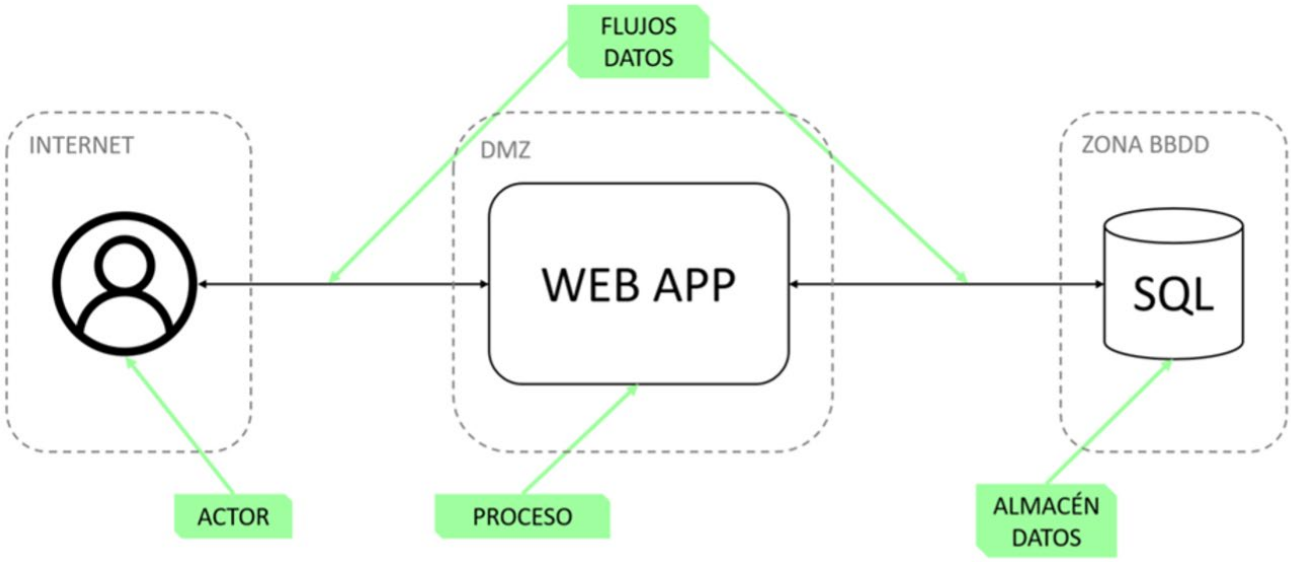


Imagen 4: ***Diagrama de flujos de datos de una web app***

Para poder realizar correctamente el modelado de amenazas, deben identificarse cada uno de los elementos y datos que intervienen y elaborar el DFD con toda la información que se desea modelar.

2. *Modelado de las amenazas*

Una vez elaborado el DFD e identificado cada *Actor, Data store, Data flow* y *Process,* se procede a realizar el modelado de amenazas de la iniciativa basado en la metodología elegida. Para la metodología STRIDE ilustrada en este ejemplo, se evalúan de manera independiente cada uno de los elementos y flujos de información descritos en el DFD contra los siguientes vectores potenciales de amenazas:

Tipología de la amenaza	Definición	Ejemplos de amenaza
Spoofing (Suplantación)	Hacerse pasar por algo o por otra persona	Hacerse pasar por un usuario o servidor válido
Tampering (Manipulación)	Modificación de datos/código no autorizada	Modificación de código (o biblioteca) en un sistema
Repudiation (Repudio)	Afirmar no haber realizado una acción	Eliminar el registro de modificación de un archivo
Information Disclosure (Fuga de información)	Exponer información a alguien no autorizado	Recopilación de información sensible de archivos de registro
Denial of service (Denegación de servicio)	Denegar o degradar el servicio a usuarios legítimos	Bloqueo de un sitio web
Elevation of privilege (Elevación de privilegios)	Obtener capacidades o privilegios sin la debida autorización	Permitir al usuario remoto ejecutar comandos, cambiar de un usuario limitado a admin

No todas las tipologías de amenazas afectan a todos los tipos de elementos identificados, ya que, por ejemplo, un actor/usuario no podría ser sometido a una elevación de privilegios, pero sí sería susceptible de ser suplantado. Como referencia de esta afectación, el tipo de amenaza que debe evaluarse para cada uno de los elementos recogidos en el DFD es la siguiente:

	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of service	Elevation of privilege
Actor	X		X			
Data store		X	X	X	X	
Data flow		X		X	X	
Process	X	X	X	X	X	X

A la hora de realizar la evaluación de amenazas de las distintas tipologías para cada uno de los elementos, ese ejercicio puede apoyarse en la utilización de marcos de referencia como el Common Attack Pattern Enumerations and Classifications (CAPEC™) de MITRE[9], que define una serie de patrones de ataques que deben ser tenidos en cuenta a la hora de modelar las amenazas potenciales o catálogos propios de la organización en base a su experiencia.

9 https://capec.mitre.org/

3. *Modelado de las amenazas*

Una vez modeladas las potenciales amenazas y riesgos de los distintos elementos participantes en la iniciativa, la última fase consiste en la definición de las medidas y controles de seguridad para la mitigación de las amenazas identificadas.

Para la definición de dichas medidas de mitigación, la organización se puede apoyar nuevamente en estándares de referencia como el NIST Cybersecurity Framework (CSF)[10] o el OWASP Application Security Verification Standard (ASVS)[11], así como un catálogo propio que desarrolle internamente la organización. Volviendo al ejemplo citado anteriormente, algunas medidas para la mitigación de las amenazas podrían ser:

Tipología de la amenaza	Ejemplos de amenaza	Control para la mitigación	Ejemplo de mitigación
Spoofing (Suplantación)	Hacerse pasar por un usuario o servidor válido	**Autenticación**	Aplicar autenticación robusta, como MFA para usuarios o certificados para servicios/APIs
Tampering (Manipulación)	Modificación de código (o biblioteca) en un sistema	**Integridad**	Imponer cifrados / hashing robustos
Repudiation (Repudio)	Eliminar el registro de modificación de un archivo	**No repudio**	Registro de eventos clave de interés. Utilizar firmas digitales
Information Disclosure (Fuga de información)	Recopilación de información sensible de archivos de registro	**Confidencialidad**	Imponer un cifrado sólido y controles de acceso
Denial of service (Denegación de servicio)	Bloqueo de un sitio web	**Disponibilidad**	Controlar el uso de recursos to control y construir resiliencia a nivel de servidor
Elevation of privilege (Elevación de privilegios)	Permitir al usuario remoto ejecutar comandos, cambiar de un usuario limitado a admin	**Autorización**	Aplicar el principio de mínimo privilegio

De esta manera, las potenciales amenazas de la aplicación estarían controladas desde la fase de diseño, pudiendo implementar los requerimientos a lo largo de todo el ciclo de vida y resultando en productos y sistemas construidos en base a principios de seguridad por diseño y por defecto.

10 https://www.nist.gov/cyberframework
11 https://owasp.org/www-project-application-security-verification-standard/

3.2. Principios de defensa en profundidad

La defensa en profundidad, frecuente referida con su término en inglés "defense in depth", es una estrategia cuyo objetivo es proteger los sistemas y datos mediante la implementación de múltiples capas de seguridad. Este enfoque reconoce que ningún mecanismo de seguridad es infalible por sí solo, por lo que al superponer diversas medidas defensivas se incrementa la resiliencia ante posibles ataques.

Aplicar una defensa en profundidad ayuda a las organizaciones en los siguientes aspectos:

1. **Reducción de riesgos:** al implementar múltiples capas de seguridad, se disminuye la probabilidad de que una amenaza pueda comprometer completamente un sistema. Si una capa es vulnerada, las siguientes actúan como barreras adicionales.
2. **Detección temprana:** la presencia de diversas medidas de seguridad aumenta las posibilidades de detectar actividades maliciosas en etapas tempranas, permitiendo una respuesta más rápida y efectiva.
3. **Mitigación de impacto:** en caso de que ocurra una brecha de seguridad, las capas adicionales pueden contener y limitar el alcance del daño, protegiendo activos críticos y reduciendo las consecuencias negativas.
4. **Adaptabilidad:** la defensa en profundidad permite a las organizaciones adaptarse a un panorama de amenazas en constante evolución, incorporando nuevas medidas de seguridad conforme surgen nuevas vulnerabilidades y técnicas de ataque.

Las múltiples capas de seguridad de esta estrategia de defensa en profundidad pueden agruparse en tres categorías principales:

1. **Controles Físicos:** estas medidas buscan proteger el acceso físico a los sistemas y datos. Incluyen elementos como cerraduras, sistemas de vigilancia, control de acceso a instalaciones y seguridad perimetral.
2. **Controles Técnicos:** se refieren a las medidas implementadas en el hardware y software para proteger los sistemas informáticos. Dentro de esta categoría se encuentran algunos controles como:
    a. **Firewalls:** dispositivos o programas que filtran el tráfico de red no autorizado.
    b. **Sistemas de Detección y Prevención de Intrusos (IDS/IPS):** herramientas que monitorean y analizan el tráfico de red en busca de actividades sospechosas.
    c. **Software Antivirus y Antimalware:** aplicaciones para la detectar y eliminar software malicioso.
    d. **Cifrado de Datos:** técnicas que protegen la información mediante su transformación en formatos ilegibles para usuarios no autorizados.
3. **Controles Administrativos:** políticas y procedimientos establecidos para gestionar la seguridad dentro de una organización. Incluyen:
    a. **Políticas de Seguridad:** directrices que definen cómo se debe proteger la información y los sistemas.
    b. **Procedimientos Operativos Estándar:** instrucciones detalladas sobre cómo realizar tareas específicas de manera segura.
    c. **Capacitación y Concienciación:** programas educativos para empleados sobre prácticas de seguridad y reconocimiento de amenazas.
    d. **Gestión de Incidentes:** procesos para identificar, responder y recuperarse de eventos de seguridad.

Ejemplo de Implementación de Defensa en Profundidad

Para ilustrar la aplicación de la defensa en profundidad, puede considerarse una organización que a través de su plataforma de comercio electrónico maneja transacciones financieras y datos sensibles de clientes, incluyendo información personal y datos de tarjetas de crédito. Dado que es un objetivo atractivo para los ciberdelincuentes, la organización decide implementar un enfoque de defensa en profundidad para proteger su infraestructura, datos y usuarios. Para ello, despliega las siguientes capas:

1. Controles Físicos

- **Acceso restringido al centro de datos:** solo el personal autorizado puede ingresar a las salas de servidores mediante tarjetas de acceso y autenticación biométrica.
- **Videovigilancia:** instalación de cámaras que monitorizan el acceso a los servidores y áreas que manejan información crítica.
- **Sistemas de alimentación auxiliares:** en caso de cortes de energía, el centro de datos cuenta con baterías de respaldo y generadores para evitar interrupciones.

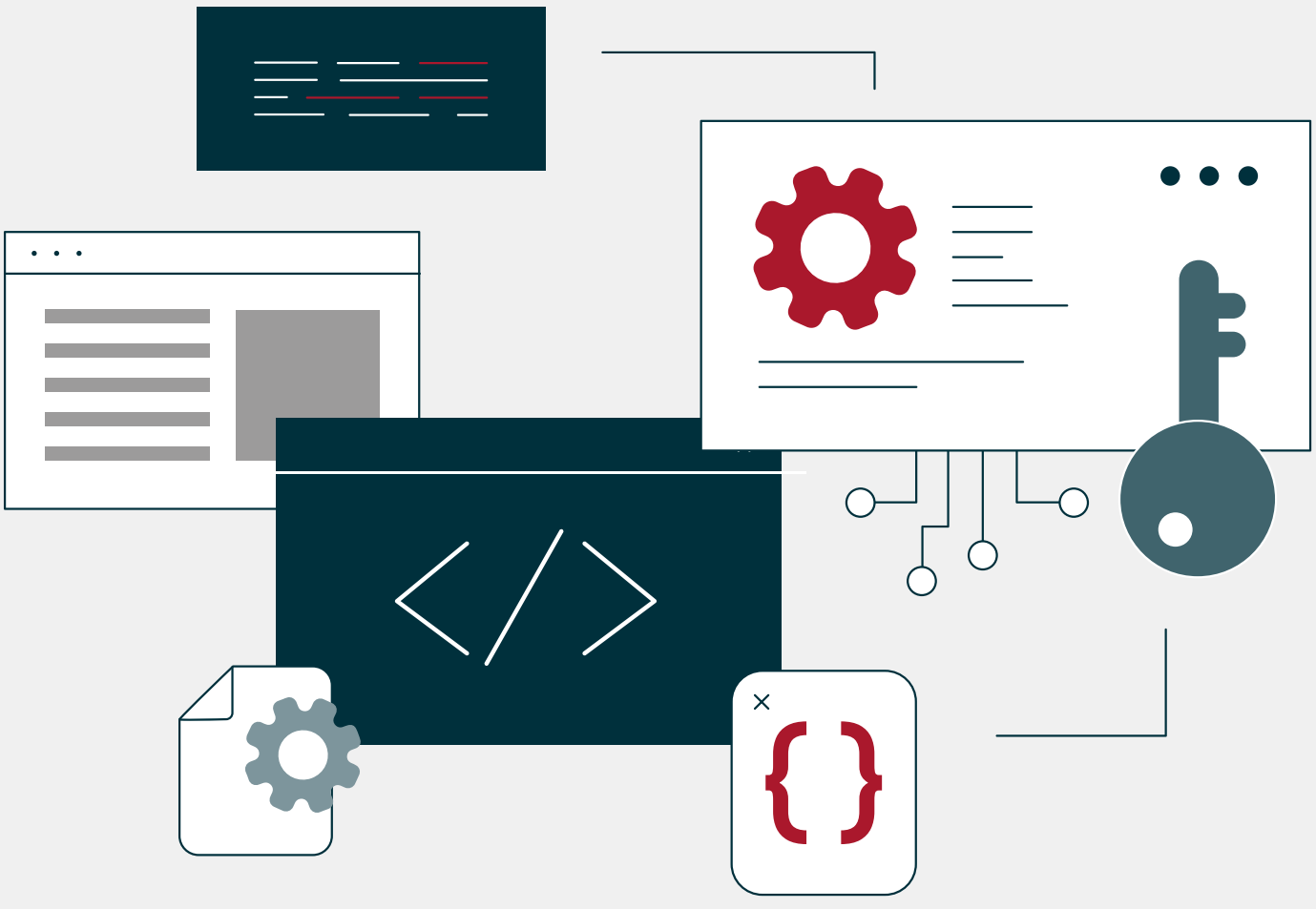
2. Controles Técnicos

- **Firewall perimetral:** un firewall filtra todo el tráfico entrante y saliente, bloqueando conexiones sospechosas.
- **Redes segmentadas:** se separan las bases de datos de los servidores web y redes internas, evitando que un atacante acceda a información crítica desde la interfaz pública.
- **Sistemas de Detección y Prevención de Intrusos (IDS/IPS):** monitorizan y bloquean automáticamente actividades maliciosas en la red.
- **WAF (Web Application Firewall):** protege la tienda en línea contra ataques como inyección SQL y cross-site scripting (XSS).
- **Cifrado de datos:** toda la información sensible (contraseñas, datos financieros) se cifra con AES-256 en reposo y con TLS 1.3 en tránsito.
- **Autenticación Multifactor (MFA):** clientes y empleados deben usar MFA al iniciar sesión en la plataforma.
- **Política de contraseñas seguras:** se exige que los usuarios creen contraseñas robustas y se impide el uso de claves comprometidas en filtraciones de datos.
- **Protección contra ataques de fuerza bruta:** se bloquea temporalmente la cuenta tras múltiples intentos fallidos de inicio de sesión.

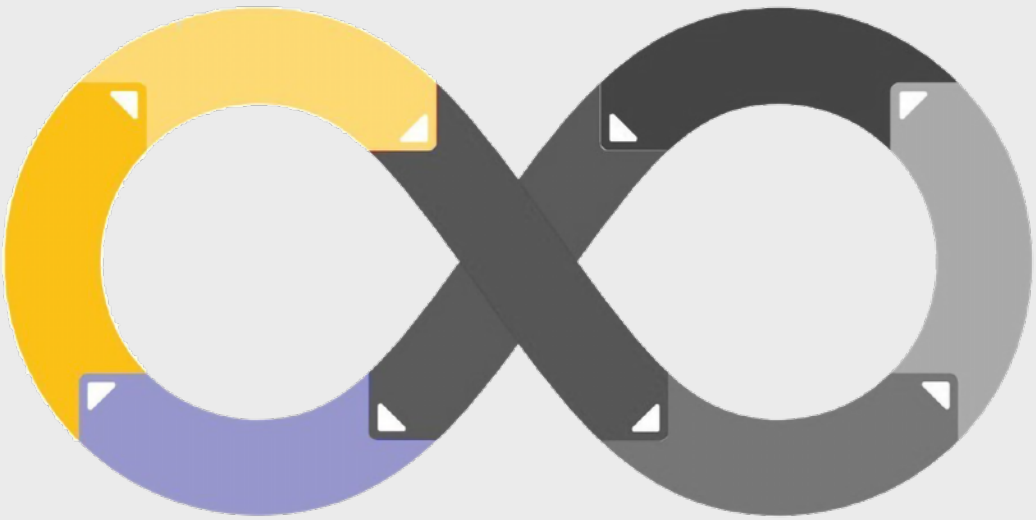
3. Controles Administrativos

- **Concienciación sobre seguridad:** se capacita periódicamente a los empleados sobre buenas prácticas en ciberseguridad, especialmente a los desarrolladores encargados de evolucionar y mantener la plataforma.
- **Plan de respuesta a incidentes:** se establece un protocolo para reaccionar ante brechas de seguridad, incluyendo contacto con las autoridades y notificación a los clientes.
- **Auditorías y pruebas de seguridad:** se realizan pruebas de penetración periódicas para identificar vulnerabilidades antes de que sean explotadas por atacantes.
- **Gestión de parches y actualizaciones:** todo el software (sistema operativo, CMS, plugins, librerías) se mantiene actualizado con los últimos parches de seguridad.

Gracias a este enfoque de defensa en profundidad, incluso si un atacante logra vulnerar una capa de seguridad (por ejemplo, explotar una vulnerabilidad en la web), todavía tendría que superar otras barreras como la segmentación de redes, autenticación multifactor, cifrado y detección de intrusos. De este modo, la plataforma protege los datos de los clientes y minimiza el impacto de cualquier posible brecha de seguridad.



4 Fase 2: Entornos de desarrollo.



En el proceso de desarrollo seguro, la configuración adecuada del entorno de trabajo es una de las primeras barreras de protección contra vulnerabilidades y ataques. Un entorno de desarrollo seguro no solo previene la introducción de errores en el código, sino que también protege los activos digitales, evita filtraciones de información sensible y facilita auditorías de seguridad. Sin un entorno bien gestionado, es posible que credenciales, configuraciones críticas y datos sensibles terminen expuestos en repositorios públicos o comprometidos por errores humanos.

4.1. Plugins IDE

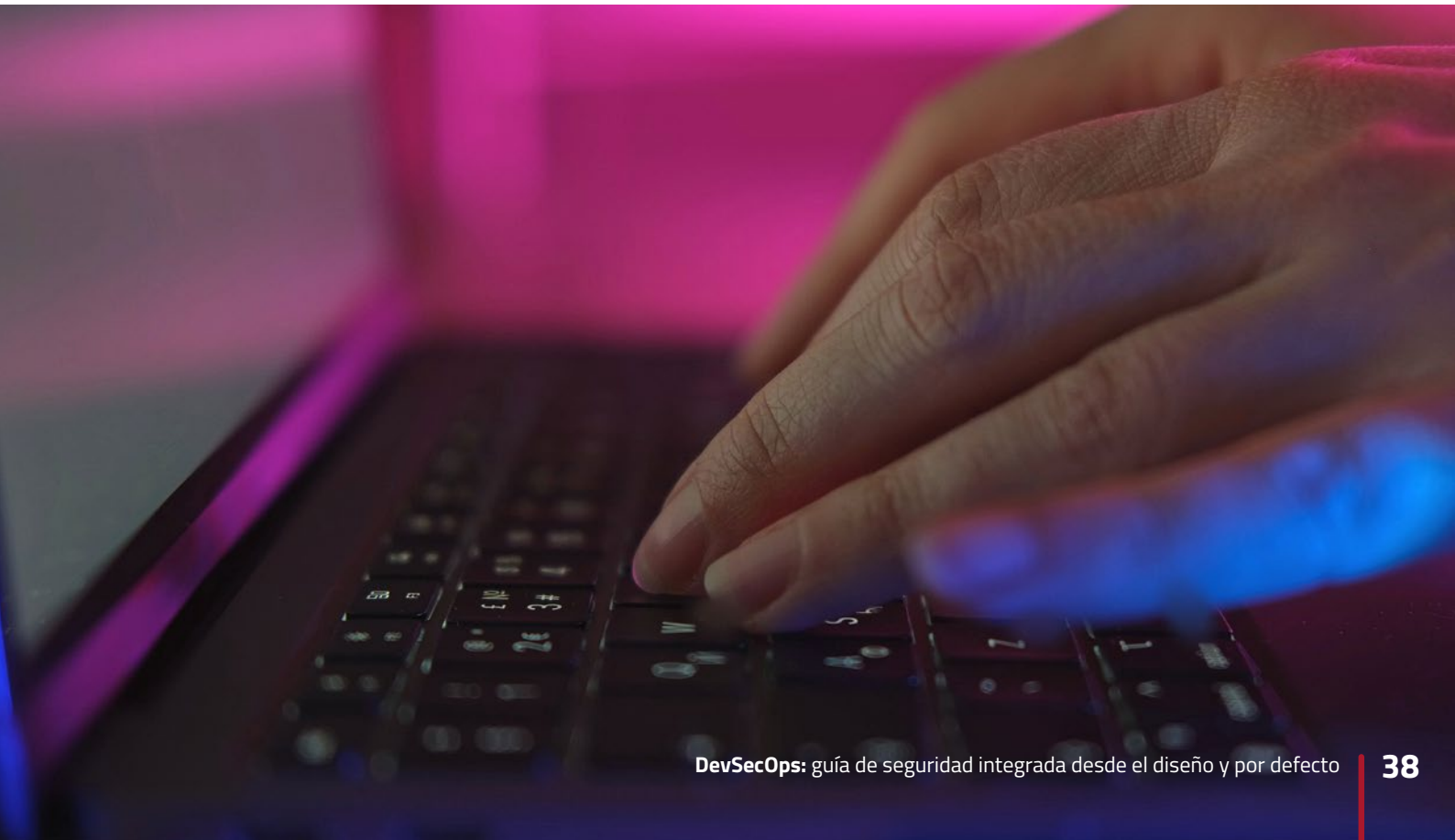
Los plugins pueden ampliar la funcionalidad de los entornos de desarrollo, pero también representar un vector de riesgo si no se gestionan correctamente. Un plugin malicioso o comprometido puede capturar información confidencial, modificar código de manera no autorizada o instalar malware en el sistema del desarrollador. Dado que los IDEs tienen acceso directo al código fuente, un plugin con comportamiento malicioso podría introducir puertas traseras, robar credenciales almacenadas en archivos de configuración o modificar código sin que el desarrollador lo note.

Estrategias para mitigar riesgos:

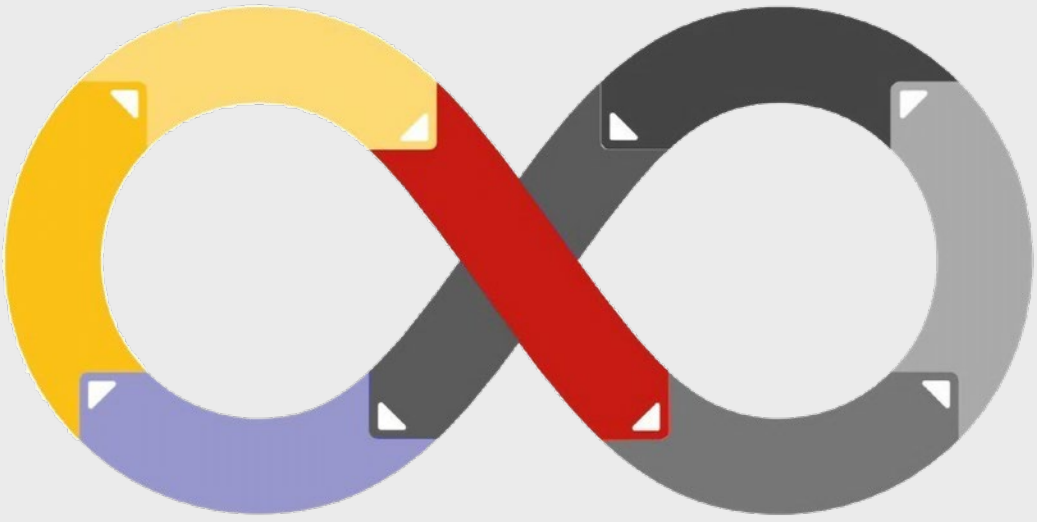
- **Instalar únicamente plugins de fuentes confiables,** como los repositorios oficiales de los IDEs.
- **Mantener los plugins actualizados** para evitar vulnerabilidades explotables.
- **Revisar permisos y acceso a datos sensibles** antes de instalar cualquier extensión. Es recomendable revisar las políticas de privacidad de los plugins y verificar qué tipo de información pueden recolectar.
- **Evitar plugins innecesarios** para reducir la superficie de ataque. Cuantos más plugins instalados, mayor es el riesgo de que alguno contenga vulnerabilidades.
- **Usar listas blancas de plugins aprobados** en entornos empresariales y auditar periódicamente su seguridad.

Los plugins que usan los desarrolladores deberían seguir un control similar a la integración de código abierto en el código. Al tener una visibilidad total del entorno de desarrollo, son el candidato perfecto para la inclusión de malware con el objetivo de exfiltrar datos. Por esta razón, es recomendable implementar herramientas de monitoreo que analicen el comportamiento de los plugins instalados y alerten sobre actividades sospechosas.

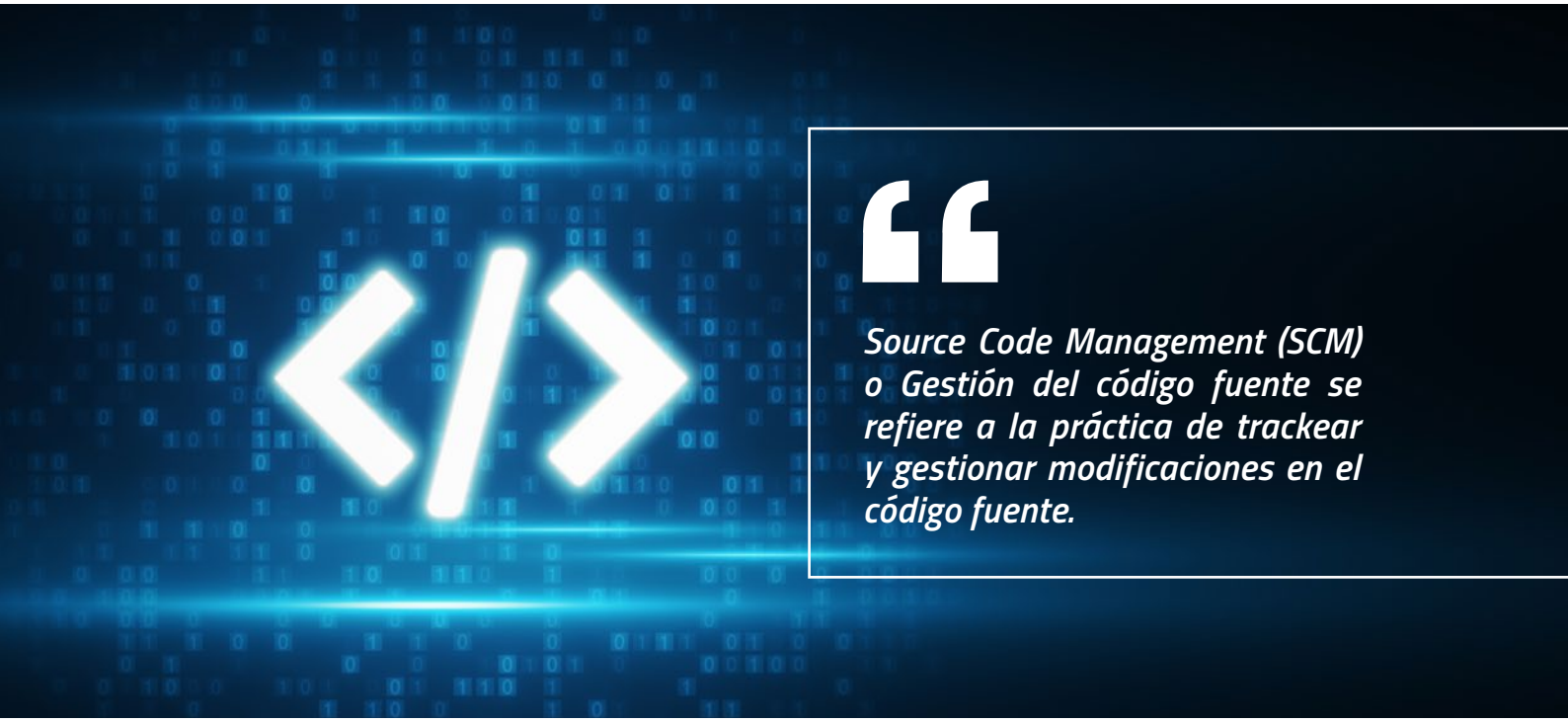
> *Por otro lado, los plugins pueden ser los mejores aliados en la seguridad del software. Cada vez más herramientas de **SAST (Static Application Security Testing) y SCA (Software Composition Analysis), como Snyk,** están añadiendo plugins para el IDE que permiten a los equipos de desarrollo ver en tiempo real las posibles vulnerabilidades sin necesidad de subir el código al repositorio. Esto agiliza notablemente el proceso de desarrollo seguro, ya que los desarrolladores pueden corregir fallos en el código mientras programan, reduciendo la cantidad de vulnerabilidades que llegan a producción.*



5 Fase 3: Source code management (SCM).



5.1. Qué es SCM



> *Source Code Management (SCM) o Gestión del código fuente se refiere a la práctica de trackear y gestionar modificaciones en el código fuente.*

El hecho de llevar un control de los cambios sufridos por el código ayuda a que programadores y desarrolladores se aseguren el estar siempre trabajando con la versión actualizada del código fuente. De esta forma se evitan y resuelven conflictos al integrar código de diferentes fuentes[12].

Como hemos visto en la Introducción al SDLC, el paradigma de desarrollo del software ha evolucionado en las últimas décadas hacia un modelo en el que se hacen actualizaciones y despliegues de código a diario, con múltiples equipos de desarrolladores contribuyendo en paralelo a código fuente compartido. Un desarrollador puede estar trabajando en un fragmento de código y, cuando quiere integrarlo más tarde, se da cuenta de que otro desarrollador ha realizado cambios en el mismo fragmento de código, por lo que entraría en conflicto. Antes de que el SCM se popularizara, apenas existían alternativas para prevenir situaciones como la descrita, y es por ello que el SCM se ha vuelto crucial en entornos de desarrollo de código modernos.

Algunas de las herramientas de SCM más usadas son GitHub[13], Bitbucket[14] o GitLab[15]. Estas herramientas se basan en la centralización y el versionado para resolver los conflictos mencionados anteriormente, además de incluir otras muchas funcionalidades que iremos viendo a lo largo de esta guía, que contribuyen a ofrecer a los equipos de desarrollo de software un entorno mucho más seguro.

Una de las principales ventajas presentadas por las herramientas de Source Code Management es la posibilidad de revertir o deshacer cambios en el código, volviendo a una versión anterior sin errores. Además, se convierten en aliadas esenciales de auditores, pues el mantenimiento de un claro registro de historial de cambios (quién ha hecho qué exactamente y cuándo), ofreciendo integridad, trazabilidad y transparencia. Por último, contribuyen a una mayor autonomía, productividad y velocidad de despliegue de aplicaciones por parte de los desarrolladores, a la vez que a la reducción de costes y la eficacia en la colaboración[16].

12 Splunk. https://www.splunk.com/en_us/blog/learn/source-code-management.html Accedido en 30 de enero, 2025.
13 Github https://github.com/
14 Bitbucket https://bitbucket.org/
15 Gitlab https://about.gitlab.com/es/
16 Atlassian https://www.atlassian.com/es/git/tutorials/source-code-management Accedido en 30 de enero, 2025

La siguiente tabla,[17] aunque no exhaustiva puesto que el objetivo de esta guía no es ser extremadamente técnica, describe algunos de los controles de seguridad que se pueden implementar en la fase de gestión del código fuente.

Nombre control	Descripción	Prioridad	Dificultad de implementación
Sistema de Gestión de Código Fuente	Usar un sistema de control de código fuente (SCM) centralizado, como puede ser GitHub, Bitbucket o GitLab	1	Baja
Roles de usuario	Crear roles de usuarios y de equipos únicos, de tal forma que se proporcione un acceso a medida al código fuente	1	Baja
SSH	Priorizar el uso del protocolo SSH, en lugar de HTTPS, para acceder a los repositorios de código	2	Baja
Clave GPG	Generar una clave GPG (*GNU Privacy Guard*) y añadirla al SCM para garantizar que la persona actualizando el código es quien dice ser	2	Baja
Doble factor de autenticación (MFA)	Imponer el uso de doble factor de autenticación (MFA) en el momento de descargar, acceder o introducir código en el repositorio	1	Baja
"Git hooks" en el servidor	*"Git hooks"* son escaneos automáticos del código, realizados en el servidor en este caso, que se pueden ejecutar tras introducir cambios en el código fuente de forma remota y proporcionar un feedback rápido al desarrollador	3	Media
Colaboración	Promover e integrar el uso de herramientas de colaboración para documentar los cambios introducidos en las aplicaciones software	2	Alta
Pull requests y *peer reviews*	Obligar al uso de *Pull Requests (PR)* o "solicitud de extracción" para asegurar que actualización de código en el repositorio es revisada por otros miembros del equipo (*peer reviews*), sin que se pueda agregar nuevo código fuente directamente	1	Baja
CODEOWNERS	Crear un archivo de *CODEOWNERS* en el repositorio que identifique el equipo o gente responsable o propietaria del código fuente gestionado en dicho repositorio. Configurar los PR para que sean revisados por dicho grupo de *CODEOWNERS*	1	Baja
SECURITY.md	Crear un archivo llamado *SECURITY.md* en la raíz del repositorio que explique a quién y cómo contactar si hay algún problema de seguridad en la aplicación	3	Baja
Repositorio. github	Crear un repositorio llamado. github en el SCM, donde se incluyan los archivos de *SECURITY.md, CONTRIBUTING.md* y demás archivos generales, que definan los procesos y bases a seguir en la organización de forma generalizada cuando no existan dichos archivos en repositorios concretos	3	Baja

*Prioridad del 1 al 3, siendo 1 más prioritario y 3 menos prioritario u opcional.

17 https://github.com/6mile/DevSecOps-Playbook

A mayores de las configuraciones de seguridad, buenas prácticas en la gestión de código fuente incluyen:

- Hacer confirmaciones (*commit*) a menudo, incluyendo información detallada sobre los cambios realizados.
- Asegurarse estar trabajando siempre sobre la versión actualizada del código.
- Usar diferentes ramas (*branches*) de trabajo, independientes de la rama principal de código.
- Estandarizar los flujos de trabajo de los equipos de desarrollo para seguir patrones de colaboración compartidos.

5.2. Uso apropiado de Git

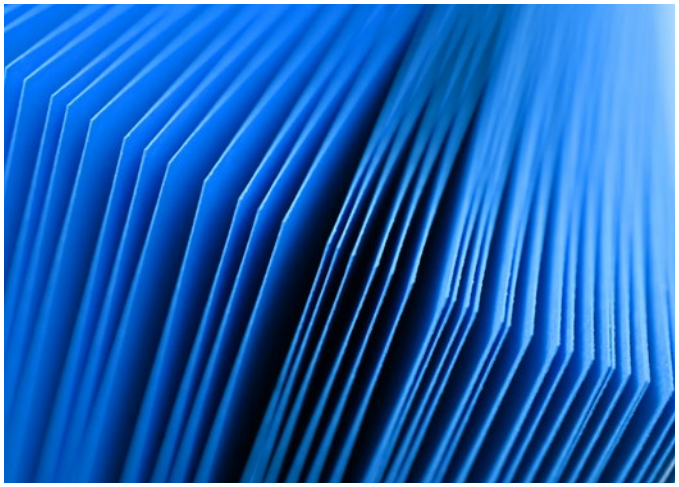
Git es el estándar o sistema de control de versiones distribuido por excelencia. Es un proyecto de código abierto, pero con enorme y creciente aceptación en todo tipo de organizaciones. Git representa una parte central en cualquier caja de herramientas de desarrollo modernas, integrando las prácticas y procesos de DevOps dentro de todo el proceso de desarrollo software, desde la integración continua (CI) hasta el despliegue continuo (CD)[18].

Normalmente, los desarrolladores mantienen una copia del repositorio de código fuente de forma local en su equipo donde se ejecuta el servicio de Git, además de existir el repositorio de forma remota en el servidor de Git remoto (GitHub, Bitbucket o GitLab). El desarrollador trabaja en las modificaciones de código en su copia local y, cuando va avanzando con los cambios, los copia o "empuja" (*push*) al repositorio en el servidor remoto a través de HTTPS o SSH. Dado que el foco de esta guía no es explicar todos los conceptos o acciones de Git, si no explicar cómo se debe utilizar de forma segura.

Se puede obtener más información sobre conceptos básicos de Git en https://git-scm.com/doc

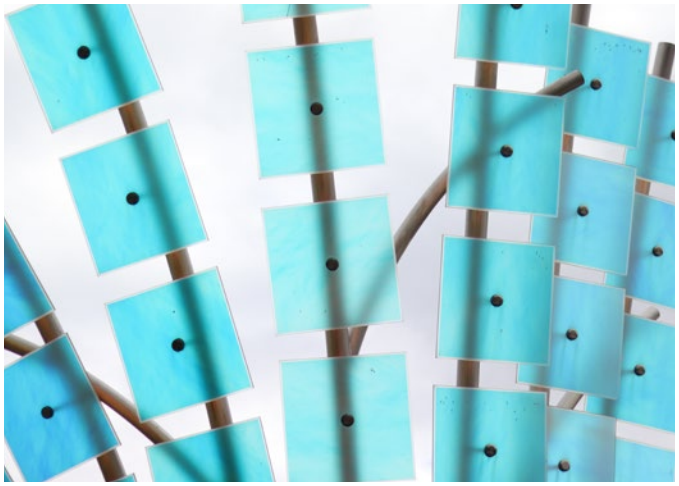
18 IBM. Source code management. Accedido en 3 de febrero, 2025. https://www.ibm.com/docs/en/z-devops-guide?topic=applications-source-code-management

Se recomienda el uso de herramientas como Git, con configuraciones que refuercen la seguridad, tales como:



- **Gestión de accesos y escritura en repositorios.** Limitar los permisos de escritura y lectura en repositorios según la función de cada usuario evita cambios accidentales o maliciosos en el código. Un ejemplo práctico sería únicamente permitir subir código a ramas de desarrollo, con previa aprobación de otro desarrollador del equipo con suficiente experiencia. Además, en entornos empresariales, se recomienda el uso de autenticación multifactor (MFA) para acceder a los repositorios y la restricción de commits directos en la rama principal (main o master).

- **Activación de firmas digitales en los commits.** Permite verificar la autenticidad de cada contribución, reduciendo el riesgo de suplantación de identidad. Sin esta medida, un atacante con acceso al repositorio podría modificar el código y hacer pasar los cambios como si hubieran sido realizados por otro desarrollador legítimo. Además, en proyectos de código abierto o en equipos distribuidos, las firmas digitales garantizan que los commits provienen realmente de los colaboradores esperados.



- **Establecer un flujo de trabajo claro.** Es importante que todo el equipo de desarrollo trabaje con un flujo estandarizado de trabajo. La creación de ramas, los momentos en los que se realizará el merge y cuando se da el código por bueno para ser desplegado son momentos críticos para poder establecer controles. Un flujo bien definido reduce la posibilidad de errores humanos, facilita las revisiones de código y permite detectar vulnerabilidades antes de que el software llegue a producción.

Un flujo de trabajo seguro en Git debería incluir ramas bien definidas (como develop, feature, release y main), revisiones por pares y aprobación de cambios antes de la integración. Hace años se puso de moda el GitFlow, una metodología de trabajo en Git basada en una rama develop y otra main, en la que develop representaba el trabajo de integración continua y en main se integran las releases que se desplegarán en producción. Aunque GitFlow sigue siendo útil en algunos entornos, en la actualidad muchas empresas han adoptado enfoques más ligeros, como GitHub Flow o Trunk-Based Development, que permiten una entrega más rápida y continua del software sin la complejidad de múltiples ramas intermedias.

En cualquier estrategia de versionado, lo más importante es mantener la disciplina en la gestión de ramas, asegurando que cada cambio pase por una revisión estructurada y que las prácticas de seguridad, como el escaneo automático de código y la validación de dependencias, sean parte del proceso de integración.

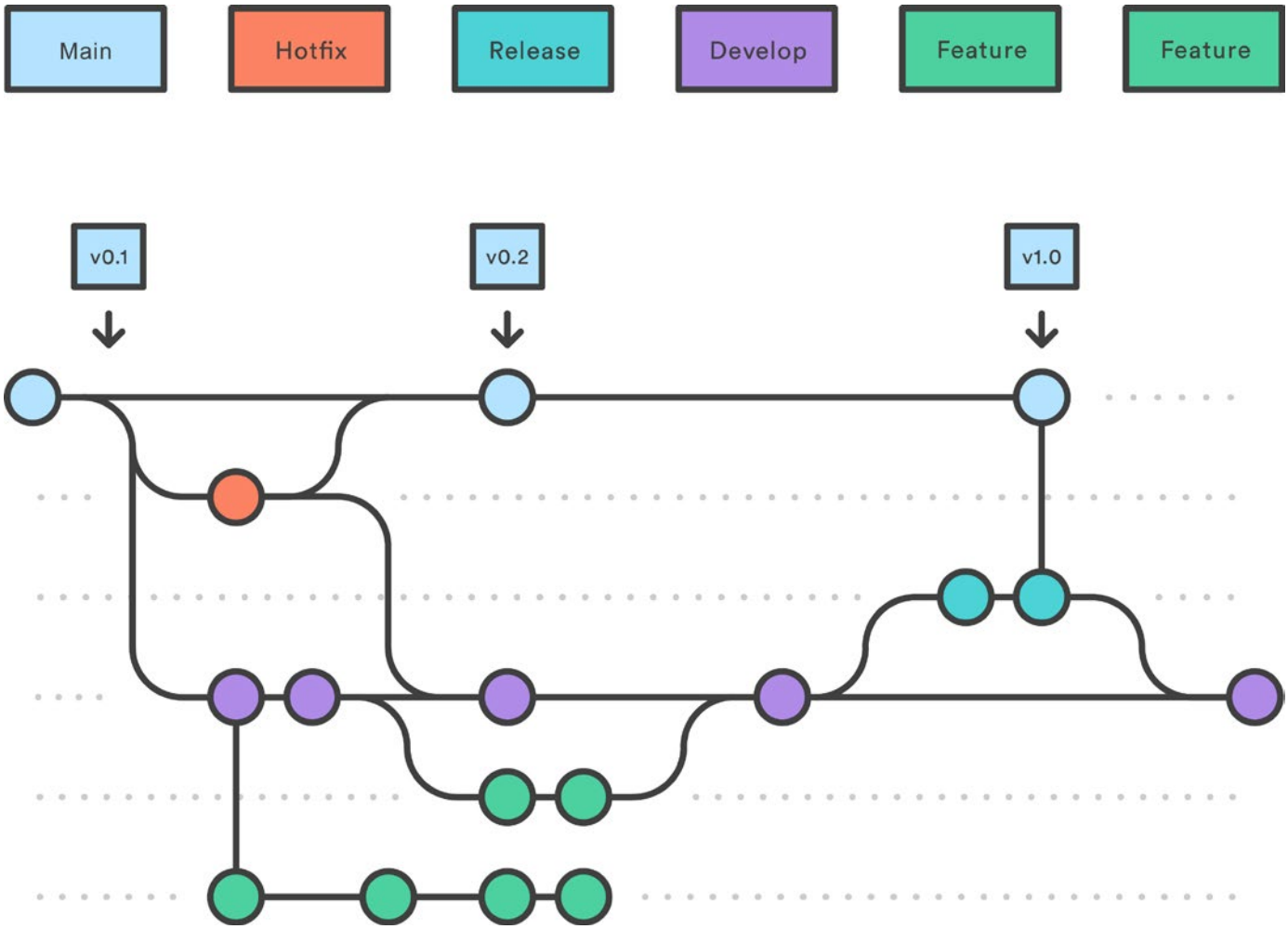
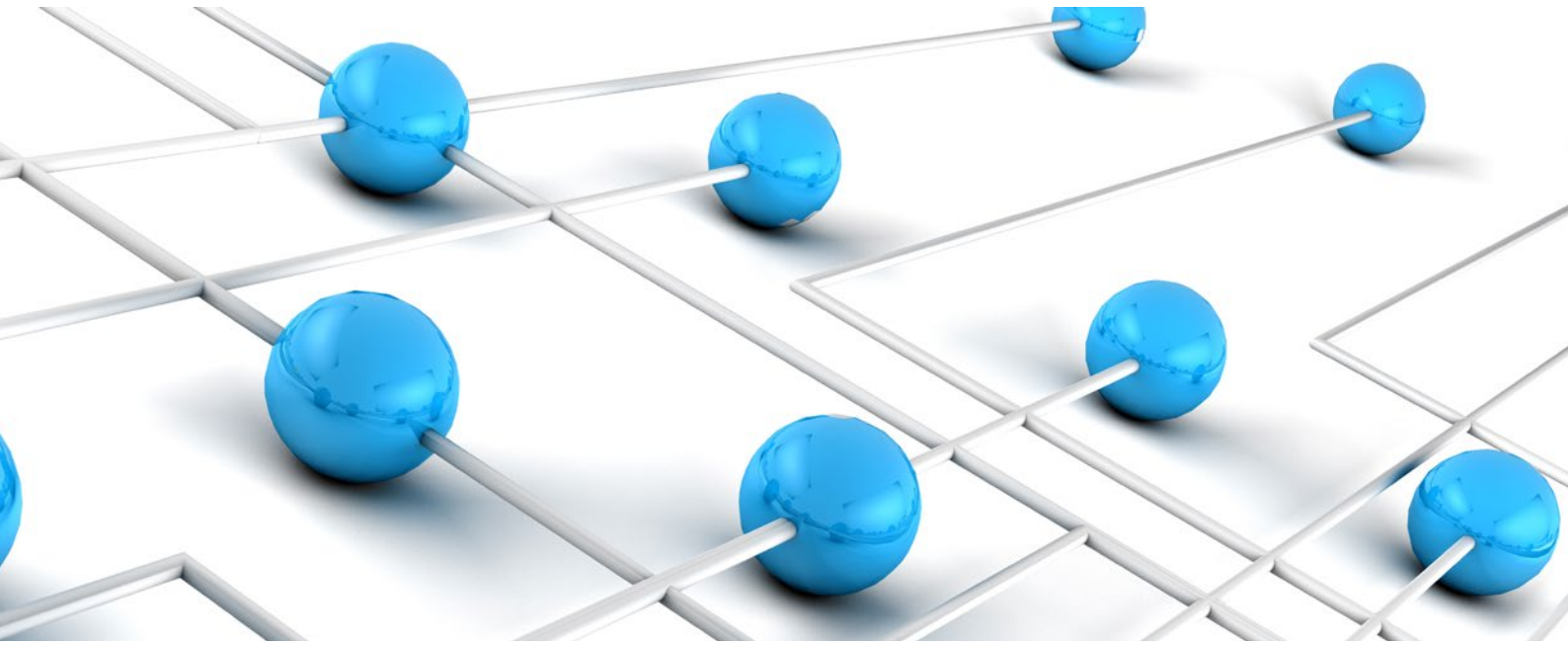


Imagen 5. ***Diagrama de gitflow***[19]



19 https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

5.3. gitignore

> *El archivo **.gitignore** es una herramienta clave para evitar la exposición de archivos sensibles en repositorios de código. Configurarlo adecuadamente previene la inclusión accidental de credenciales, configuraciones locales y archivos temporales, mitigando riesgos de seguridad que pueden derivar en accesos no autorizados, filtraciones de datos y vulnerabilidades explotables.*

El uso incorrecto de **.gitignore** ha causado filtraciones de datos en múltiples ocasiones, donde credenciales de bases de datos, claves API y configuraciones de servidores han sido expuestas en repositorios públicos. Un simple descuido al no ignorar un archivo **.env** puede comprometer aplicaciones enteras. Existen casos documentados donde empresas han sufrido ataques debido a información sensible expuesta en repositorios públicos, facilitando accesos indebidos a bases de datos, servidores y sistemas de autenticación.

Además, los atacantes pueden utilizar herramientas automatizadas para escanear repositorios públicos en busca de información confidencial. Sitios como GitHub han implementado herramientas de escaneo que alertan sobre la exposición de credenciales, pero esto no garantiza la protección total si los desarrolladores no siguen buenas prácticas en la gestión de archivos sensibles.

Algunas buenas prácticas recomendadas son:

Excluir archivos de configuración local como **.env**, **config.json** y **settings.py** para evitar exponer credenciales o configuraciones sensibles. Estos archivos suelen contener claves de acceso a bases de datos, API keys o configuraciones críticas que pueden ser utilizadas por atacantes para comprometer la infraestructura.

Omitir carpetas de dependencias como node_modules o vendor para evitar repositorios innecesariamente pesados y minimizar el riesgo de incluir librerías comprometidas. Incluir dependencias en el repositorio no solo lo hace más difícil de manejar, sino que también puede facilitar la inyección de código malicioso si se descargan paquetes inseguros sin una revisión adecuada.

Evitar la inclusión de binarios y archivos generados como **.log**, **.DS_Store** o archivos de compilación, asegurando que solo el código relevante forme parte del repositorio. Estos archivos pueden contener información sensible de debugging o metadatos que expongan detalles sobre la máquina del desarrollador, como nombres de usuario o rutas internas.

Es recomendable utilizar plantillas de **.gitignore** específicas para cada tecnología, idealmente el desarrollador debe conocer la lógica de negocio de la aplicación para evitar la exfiltración de contraseñas o datos sensibles. En la siguiente URL puede consultarse una versión actualizada y separada por tecnologías populares de plantillas de **.gitignore** de ejemplo: 👉 GitHub Gitignore.

Además, se recomienda realizar auditorías periódicas en los repositorios para verificar que no se han cometido errores en la configuración de **.gitignore**. Herramientas como **git-secrets** o **truffleHog** pueden ayudar a identificar credenciales expuestas de manera accidental y tomar medidas correctivas antes de que la información sea explotada.

5.4. Usuarios/Roles/Accesos

Como ya se describía en la primera edición de la Guía de iniciación en la Seguridad aplicada al DevOps del ISMS Forum[20], en el apartado 5.1. Seguridad desde el diseño y por defecto, una de las principales medidas de seguridad, no sólo en la gestión del código fuente, consiste en definir unos pilares básicos de seguridad por diseño. Varios de estos pilares se centran en implantar el principio de mínimo privilegio y en la gestión segura de usuarios y accesos. El objetivo de este apartado es ampliar la información relativa a la gestión recomendada de usuarios, roles y/o accesos específica a un SCM. Para ello, se proponen las siguientes prácticas:

1. **Implementar y promover el principio de mínimo privilegio,** de modo que tanto los usuarios como los equipos de la herramienta de SCM sólo tengan acceso a los recursos y funcionalidades necesarias para su rol específico. Dado que el principio de mínimo privilegio es ya sobradamente conocido y se ha abordado en numerosos ámbitos, vamos a evitar justificarlo en mayor detalle. Concretamente en un SCM, se recomienda que sean equipos quienes tengan diferentes niveles de acceso (roles) a repositorios, y que los usuarios sean añadidos a equipos para obtener el acceso necesario a los repositorios de código. Un ejemplo de aplicación de este principio sería que un equipo solo tenga permisos de escritura para el repositorio en que necesite modificar código, mientras que por defecto solo tenga permisos de lectura, o ni siquiera visibilidad, sobre otros repositorios de la organización que no sean necesarios para su trabajo.

2. **Definir claramente los distintos roles, a nivel de organización y a nivel de repositorio,** en la herramienta de Source Code Management. En la organización, los usuarios típicamente pueden tener roles de administrador, miembro/colaborador o administrador de seguridad. Sin embargo, a nivel repositorio los posibles roles suelen ser: lectura, escritura (actualizar código), mantenimiento o *maintainers* (jefes de proyecto que necesitan administrar el repositorio sin acceder a acciones confidenciales o administrativas) y administración (acceso total al repositorio, incluidas acciones confidenciales, administrativas y de seguridad)[21]. La práctica recomendada es, como decíamos en el punto 1., asignar estos roles a los equipos, y no a usuarios individualmente. Son estos roles los que proporcionan diferentes permisos dentro del repositorio. Así mismo, se recomienda revisar estos roles y permisos periódicamente, y revocar accesos si fuera necesario.

3. **Considerar y automatizar la gestión de identidades centralizada** a través de la integración de servicios de *LDAP* o de *Active Directory,* de forma que se facilite la asignación y revocación de roles y accesos a la herramienta de SCM. Así, no sería necesaria una gestión adicional de equipos y usuarios en la herramienta, y los usuarios tendrían por defecto ya concedidos los permisos necesarios para sus responsabilidades esperadas a nivel organizacional.

20 Guía de iniciación den la Seguridad aplicada al DevOps, ISMS Forum, 2023. https://www.ismsforum.es/ficheros/descargas/guia-devsecops-v41690197585.pdf

21 Roles de repositorio en GitHub. Accedido en 2 de febrero, 2025. https://docs.github.com/es/organizations/managing-user-access-to-your-organizations-repositories/managing-repository-roles/repository-roles-for-an-organization

4. **Segregar las funciones y responsabilidades dentro del equipo.** Un ejemplo de aplicación de este principio basado en los controles enumerados en el apartado anterior, sería asegurar que un miembro que introduzca una actualización de código en el repositorio no pueda ser quien lo revise y apruebe en un *Pull Request,* ni lo integre con el código de producción.

5. **En la medida posible, es recomendable implementar políticas de control de acceso** que definan, no solo qué roles pueden asumir equipos o usuarios, si no también cuándo y de qué forma lo pueden hacer. Si no se espera que un desarrollador júnior trabaje en horarios de fin de semana ni que tenga que responder ante incidencias, aunque dicho usuario tenga permisos de escritura en un repositorio, se podría bloquear, o alertar, actualizaciones de código por dicho desarrollador en fin de semana. O requerir el doble factor de autenticación y que las actualizaciones de código estén firmadas, de tal forma que un usuario malicioso no pueda actualizar el código si las credenciales del desarrollador son comprometidas.

6. **Implementar controles de acceso durante el proceso de autenticación a la interfaz de la herramienta de SCM,** como puede ser el requerimiento de doble factor de autenticación (MFA) o el bloqueo de la cuenta de usuario de forma temporal tras varios intentos de acceso fallidos en un corto periodo de tiempo.

7. **Por último, como en cualquier área de seguridad de la información, es crucial la formación y concienciación de los usuarios** en la gestión segura de código fuente. Así como realizar auditorías y monitorización de accesos, y configurar alertas de seguridad cuando fuese necesario.

5.5. Commit Signing

La firma de commits añade una capa de autenticación a los cambios realizados en el código fuente. Al firmar digitalmente los commits con GPG o SSH, se garantiza que los cambios han sido realizados por usuarios autorizados y no han sido alterados maliciosamente. Este mecanismo refuerza la confianza en la integridad del código y dificulta la manipulación no autorizada de los repositorios.

El desarrollo de software en equipos grandes o en proyectos de código abierto enfrenta un riesgo importante de manipulación de código. Sin una autenticación sólida, los atacantes podrían suplantar la identidad de desarrolladores legítimos e introducir código malicioso en el repositorio. La firma de commits impide esta suplantación, ya que cada cambio realizado en el código puede ser verificado de manera criptográfica.

Además, en entornos donde se requiere cumplimiento normativo, como en industrias reguladas (banca, salud, gobierno), el firmado de commits facilita la auditoría del código y permite rastrear cada modificación hasta su origen. Esto garantiza la trazabilidad de los cambios y ayuda a cumplir con estándares como ISO 27001, SOC 2 y normativas específicas como el GDPR.

Algunos beneficios de la firma de commits son:

Garantiza la autenticidad del código

Asegura que solo desarrolladores autorizados pueden realizar cambios en el código fuente.

Protege contra la manipulación maliciosa

Los commits firmados no pueden ser alterados sin invalidar la firma, evitando ataques como la inyección de código o la suplantación de identidad.

Facilita auditorías y cumplimiento normativo

Permite verificar la autoría de los cambios, asegurando que el código cumple con regulaciones de seguridad.

Refuerza la confianza en equipos distribuidos y proyectos de código abierto

En entornos colaborativos donde múltiples desarrolladores contribuyen al mismo código base, la firma de commits ayuda a mantener la seguridad e integridad del código.

Cómo habilitar la firma de commits en Git:

Para garantizar que todos los commits sean autenticados, se recomienda configurar Git para utilizar firmas digitales con GPG o SSH. A continuación, se describen los pasos para habilitar esta función:

1. Generar una clave GPG o SSH con el siguiente comando:

```
gpg --full-generate-key
```

Esto creará una clave criptográfica que servirá para firmar los commits.

2. Asociar la clave a la cuenta de Git:

```
git config --global user.signingkey <clave>
```

Donde <clave> es el identificador de la clave GPG generada en el paso anterior.

3. Configurar Git para firmar automáticamente los commits:

```
git config --global commit.gpgsign true
```

Esto asegurará que cada commit realizado se firme automáticamente, evitando que se suban cambios sin autenticación.

4. Verificar la firma de un commit:

```
git log --show-signature
```

Este comando permite verificar qué commits han sido firmados y comprobar la autenticidad de los mismos.

Implementación en entornos empresariales

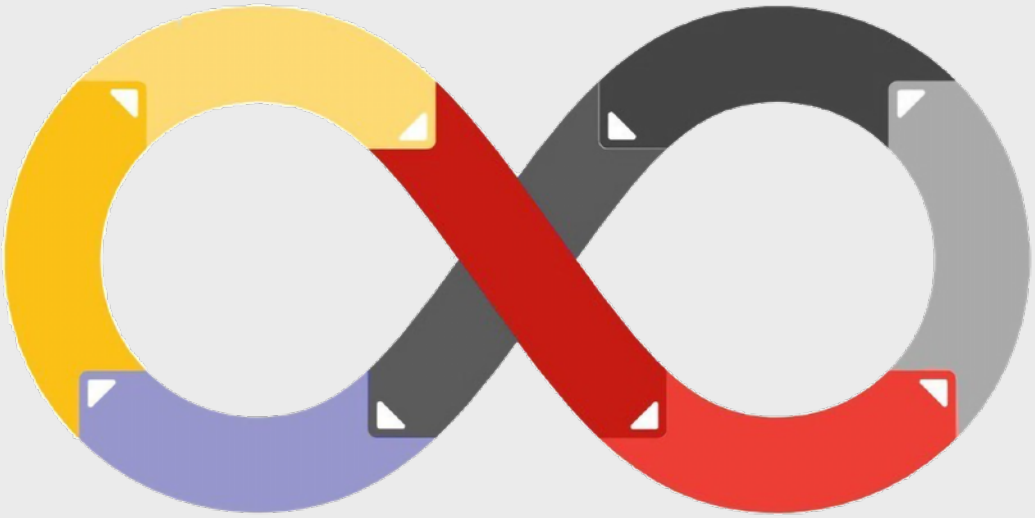
En organizaciones con equipos de desarrollo grandes, es recomendable establecer políticas de seguridad que obliguen el uso de firmas de commits. Para esto, se pueden configurar reglas en plataformas como GitHub, GitLab o Bitbucket para rechazar cualquier commit que no esté firmado. Esto se puede lograr habilitando la opción de "Commits firmados obligatorios" en la configuración del repositorio.

Además, es aconsejable complementar la firma de commits con otras medidas de seguridad, como la integración con un sistema de gestión de identidades (IAM) y el uso de autenticación multifactor (MFA) para reforzar la seguridad del acceso a los repositorios.

Consideraciones y mejores prácticas

- **Almacenar las claves GPG de forma segura.** Se recomienda utilizar gestores de claves seguras, como *gpg-agent*, para evitar que las claves privadas sean comprometidas.
- **Rotar periódicamente las claves.** Para minimizar el impacto en caso de una filtración de claves, se recomienda establecer políticas de rotación de claves criptográficas.
- **Educar a los desarrolladores sobre la importancia del firmado de commits.** Es fundamental concienciar a los equipos de desarrollo sobre la necesidad de esta práctica y proporcionar documentación clara sobre su implementación.
- **Habilitar verificaciones automáticas en los pipelines de CI/CD.** Incluir validaciones en los procesos de integración y despliegue continuo para asegurar que solo se acepten commits firmados y provenientes de fuentes confiables.

6 Fase 4: Automatización CI/CD Pipelines (SDLC).



6.1. Qué es un "Pipeline"

Un pipeline, en el contexto de CI/CD (Integración Continua y Despliegue Continuo), es una secuencia automatizada de procesos que permiten integrar, probar, validar y desplegar software de forma ágil, segura y repetible. El pipeline se concibe como una cadena de etapas donde cada una ejecuta tareas específicas para asegurar que el código del software evoluciona desde su estado inicial hasta su implementación en producción con los más altos estándares de calidad y seguridad.

Fases y componentes clave de un pipeline:

1. Integración Continua (Continuous Integration)

Se centra en la integración frecuente del código fuente proveniente de múltiples desarrolladores en un repositorio común.

Ejecuta automáticamente pruebas básicas para identificar errores rápidamente, tales como pruebas unitarias y análisis estáticos (SAST).

Genera reportes inmediatos que permiten la corrección temprana de problemas.

2. Entrega Continua (Continuous Delivery)

Extiende la fase de integración para preparar automáticamente el software para su despliegue en entornos posteriores.

Proporciona un software listo para ser desplegado en producción en cualquier momento de forma controlada y validada.

3. Despliegue Continuo (Continuous Deployment)

Automáticamente despliega cada cambio que supera todas las fases anteriores en el entorno de producción, sin necesidad de intervención manual.

Implementa políticas robustas de rollback automático en caso de errores detectados tras el despliegue.

Asegura la monitorización activa en producción para identificar posibles problemas en tiempo real.

6.2. Entornos

La gestión de entornos en los pipelines de CI/CD es fundamental para asegurar la calidad, estabilidad y seguridad en el despliegue continuo del software. Cada entorno dentro del pipeline debe ser claramente definido y configurado según sus características y objetivos específicos, asegurando la segregación y protección necesaria.

Principales entornos recomendados en un enfoque adecuado de CI/CD:

1. *Desarrollo (Dev):*

- Destinado a la integración continua del código generado por los desarrolladores.
- Debe ser un entorno altamente dinámico y flexible, permitiendo pruebas rápidas y feed-back inmediato.
- Las medidas de seguridad básicas como el escaneo SAST y DAST preliminar deben implementarse desde este nivel.

2. *Testing (QA):*

- Usado principalmente para pruebas automatizadas y manuales más exhaustivas, incluyendo pruebas funcionales, de rendimiento y seguridad.
- Este entorno debe replicar fielmente las condiciones del entorno de producción.
- Aquí se deben ejecutar análisis de vulnerabilidades más detallados (SAST, DAST y Software Composition Analysis) y validaciones específicas de cumplimiento normativo.

3. *Pre-producción (Staging):*

- Es una réplica exacta del entorno de producción para validar despliegues finales antes del lanzamiento.
- Ideal para realizar pruebas de seguridad completas, análisis de cumplimiento normativo y revisiones manuales exhaustivas de seguridad.
- Deben aplicarse técnicas como pruebas de penetración o pentesting, revisiones exhaustivas de secretos, y validación de controles como la autenticación y autorización robusta.

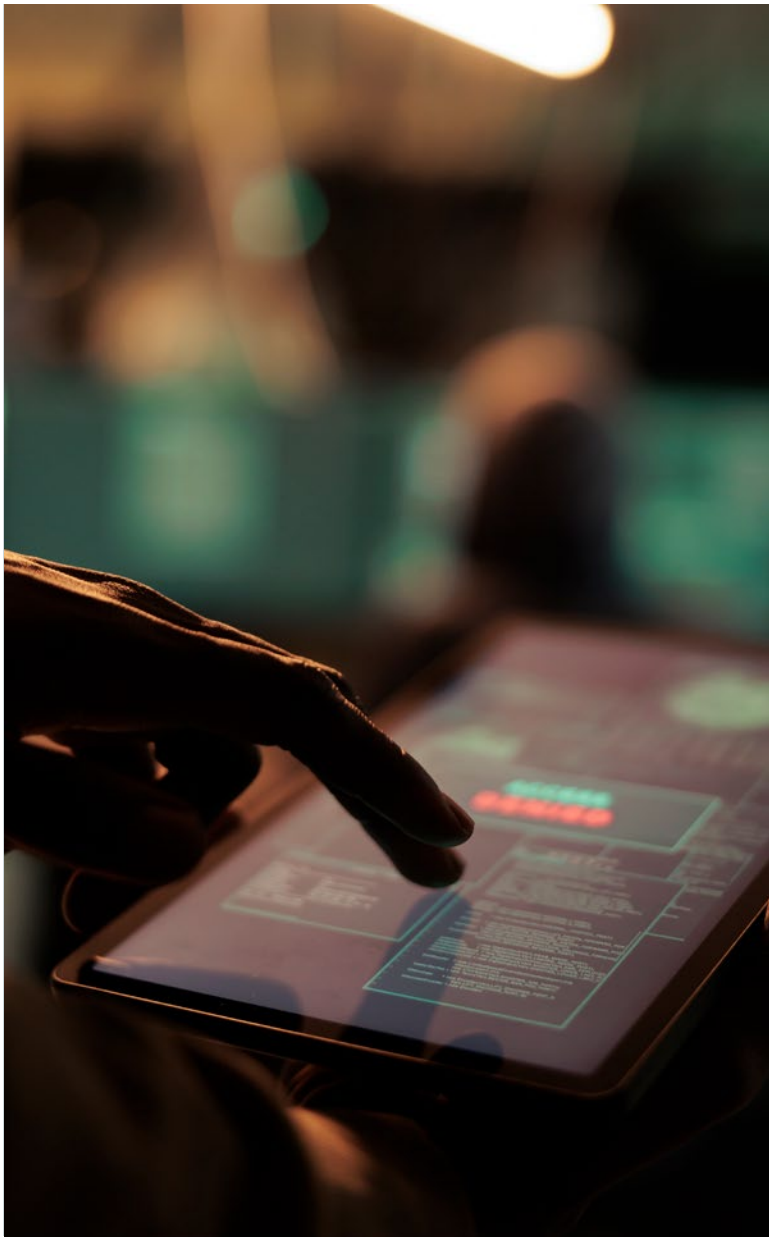
4. *Producción (Prod):*

- El entorno final donde el software queda disponible para el usuario final.
- Debe ser altamente seguro y estable, aplicando estrictamente todos los controles y buenas prácticas identificadas en etapas previas.
- Debe contar con monitorización continua en tiempo real (Application Security Posture Management - ASPM), protección activa (WAF, IDS/IPS), y gestión efectiva de incidentes.

> ***Aislamiento entre entornos.*** *Implementar separación lógica y física entre los distintos entornos para evitar la contaminación cruzada y reducir el riesgo en caso de compromiso. Desde un punto de vista lógico, esta separación implica usar diferentes redes, sistemas de autenticación y autorizaciones específicas para cada entorno. Físicamente, esto podría involucrar el uso de infraestructura dedicada, servidores separados o, en caso de utilizar entornos virtuales y de nube, garantizar configuraciones estrictas y mecanismos de aislamiento robustos como contenedores y máquinas virtuales aisladas. Este aislamiento asegura que un incidente en un entorno, como el desarrollo o testing, no afecte directamente ni comprometa otros entornos más sensibles como pre-producción o producción.*

Gestionar adecuadamente estos entornos, aplicando rigurosamente las prácticas anteriores, es esencial para un pipeline DevSecOps robusto y efectivo, reduciendo significativamente la superficie de ataque y asegurando despliegues continuos seguros y de alta calidad.

6.3. Análisis estáticos de código (SAST)



SAST (Static Application Security Testing), o Pruebas Estáticas de Seguridad de Aplicaciones, es una parte clave en el análisis de seguridad en aplicaciones que se enfoca en analizar el código fuente de una aplicación sin ejecutarlo (analizando el código fuente), con el objetivo de identificar vulnerabilidades y debilidades de seguridad. Este tipo de análisis se lleva a cabo durante la fase de desarrollo de una aplicación, lo más a la izquierda posible (*Shift-Left*), permitiendo que los equipos de desarrollo detecten posibles fallos de seguridad antes de que el software sea lanzado, y, por consiguiente, adelantándose a posibles problemas de cara a producción, donde el coste de cambiar la solución tendría un impacto mayor, tanto en esfuerzo como económico.

¿Por qué es importante el SAST?

La importancia de SAST radica en que permite detectar vulnerabilidades en las primeras fases del ciclo de vida del desarrollo de software. Al identificar problemas de seguridad de manera temprana (*Shift-Left*), se puede reducir significativamente el riesgo de que estos defectos lleguen a la producción y sean explotados por atacantes.

Entre los beneficios del SAST, podemos destacar:

1. **Detección temprana de vulnerabilidades.** Permite identificar errores en el código que podrían ser explotados por atacantes antes de que el software sea desplegado en producción, incluso detectado desde el propio IDLE (entornos integrados de desarrollo de desarrollo), que algunas soluciones tecnológicas proporcionan
2. **Reducción de costos.** Corregir vulnerabilidades en fases tempranas del ciclo de desarrollo es más barato y menos complicado que hacerlo una vez que el software está en producción.
3. **Cumplimiento normativo.** Muchas industrias, como la financiera, la de la salud y la gubernamental, tienen regulaciones estrictas sobre la seguridad de las aplicaciones. SAST ayuda a garantizar que se cumplan los estándares y normas de seguridad.
4. **Mejora de la calidad del código.** El análisis estático también puede ayudar a mejorar la calidad del código, ya que muchos de los errores encontrados también afectan la mantenibilidad y el rendimiento del software.

No obstante, existen diferentes retos a afrontar en el camino del aprovechamiento de esta tecnología:

1. **Falsos positivos.** El análisis estático, puede generar muchos falsos positivos, es decir, identificar vulnerabilidades que en realidad no son un problema. Esto puede generar una sobrecarga de trabajo para los desarrolladores, que deben revisar cada alerta y determinar si realmente se trata de una vulnerabilidad o no. Para poder reducir el impacto de los falsos positivos, es muy importante realizar un triage, y poco a poco ajustar la solución a tu entorno. En muchas ocasiones algunas soluciones de mercado ofrecen la capacidad de triage de manera automática e incluso algunas se apoyan en IA para mejorar en este aspecto.
2. **Cobertura limitada.** No podemos asumir que, por disponer de un SAST, tu aplicación está segura y cubierta ante vulnerabilidades, el SAST realiza una pequeña parte del análisis de seguridad de aplicaciones y controles adicionales en fases posteriores del ciclo de desarrollo tienen que ser implementado.
3. **Requiere experiencia y una operación adecuada.** Para implementar correctamente el SAST y analizar sus resultados de manera efectiva, se necesita personal con experiencia en seguridad de aplicaciones, ya que interpretar los hallazgos puede ser complicado. También es importante apoyarse en las figuras de los Security champions Ver sección 2.3. , ya que en muchas ocasiones las debilidades encontradas no son vulnerabilidades, sino lógica funcional del código, teniendo que tratarlo como falsos positivos.
4. **Costoso en tiempo.** En ciertas ocasiones, dependiendo del volumen de la aplicación, el análisis estático puede llevar tiempo, ya no solo en el propio análisis de la herramienta, sino también en poder ajustar los resultados. No obstante, aunque sea costoso en el tiempo, al realizarse en fases tempranas del desarrollo, todavía estamos a tiempo de ser capaces de reaccionar.

Tipos de análisis en SAST

El SAST puede realizarse utilizando diferentes métodos y enfoques, que pueden variar dependiendo de la herramienta o el marco de trabajo utilizado. Los tipos más comunes de análisis son:

1. **Análisis de código fuente.** Este es el análisis más común en SAST, en el que se revisa el código fuente completo de la aplicación. La herramienta analiza el código para identificar posibles vulnerabilidades relacionadas con la lógica, los errores de sintaxis, las malas prácticas de codificación y las fallas de seguridad.
2. **Análisis de flujo de datos.** Analiza cómo se gestionan y procesan los datos a lo largo del código para identificar posibles riesgos relacionados con la exposición de datos sensibles, la validación insuficiente de entradas o las fugas de información.
3. **Análisis de configuraciones y prácticas de codificación.** Revisa las configuraciones de seguridad y las prácticas de codificación para asegurarse de que los estándares de seguridad se apliquen correctamente, como la correcta gestión de contraseñas, la validación de entradas y el uso adecuado de técnicas de cifrado.

Pruebas de seguridad SAST

A continuación, se detallan algunos ejemplos técnicos de pruebas de seguridad SAST que la mayoría de las herramientas de análisis estático pueden realizar para detectar vulnerabilidades en el código fuente de las aplicaciones:

Desbordamiento de búfer (Buffer Overflow)

- **Descripción:** un desbordamiento de búfer ocurre cuando una aplicación escribe más datos en un búfer (como una variable de tipo char[]) de lo que puede almacenar, lo que puede sobrescribir datos importantes en la memoria y permitir la ejecución de código malicioso.
- **Regla SAST:** verificar el uso de funciones inseguras que no realizan comprobaciones de longitud, como strcpy(), gets(), o sprintf() en C/C++.
- **Ejemplo de código vulnerable:**

```
char buffer[10];
gets(buffer); // Uso inseguro de gets()
```

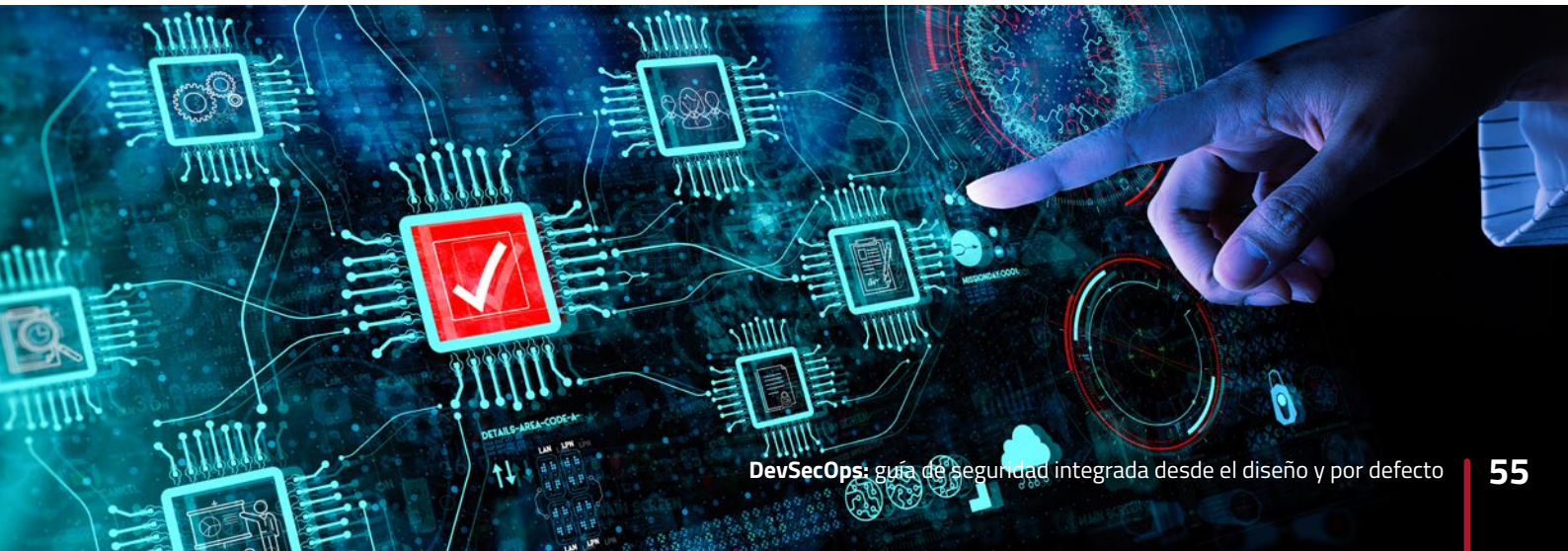
- **Problema:** la función gets() no controla la longitud de la entrada y puede causar un desbordamiento de búfer.

Inyección SQL (SQL Injection)

- **Descripción:** la inyección SQL ocurre cuando un atacante manipula una consulta SQL al insertar datos maliciosos a través de entradas de usuario no validadas. Esto puede permitir a un atacante ejecutar comandos SQL arbitrarios en la base de datos.
- **Regla SAST:** detectar concatenaciones de cadenas que forman consultas SQL sin sanitizar la entrada del usuario.
- **Ejemplo de código vulnerable:**

```
String query = "SELECT * FROM users WHERE username = '" + username + "' AND password = '" + password + "'";
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery(query);
```

- **Problema:** la concatenación directa de los parámetros username y password puede permitir una inyección SQL.



Cross-Site Scripting (XSS)

- **Descripción:** el Cross-Site Scripting (XSS) es una vulnerabilidad donde un atacante inserta scripts maliciosos en páginas web que luego se ejecutan en el navegador de otros usuarios, permitiendo, entre otros, el robo de información sensible.
- **Regla SAST:** detectar la falta de sanitización de entradas de usuario antes de ser mostradas en el HTML.
- **Ejemplo de código vulnerable:**

```
<input type="text" name="username" value="<%= request.getParameter("username") %>">
```

- **Problema:** el parámetro username puede contener código JavaScript malicioso y ejecutarse en el navegador de otros usuarios.

Deserialización insegura

- **Descripción:** la deserialización insegura ocurre cuando una aplicación permite la deserialización de datos no confiables sin validarlos, lo que puede permitir que un atacante ejecute código malicioso o acceda a datos no autorizados.
- **Regla SAST:** detectar la deserialización de objetos sin validación.
- **Ejemplo de código vulnerable (Java):**

```
ObjectInputStream in = new ObjectInputStream(new FileInputStream("userData.ser"));
UserData data = (UserData) in.readObject(); // Deserialización insegura
```

- **Problema:** la deserialización de objetos sin validación de seguridad puede permitir la ejecución de código malicioso.

6.4. Análisis dinámicos de código (DAST)

Dynamic Application Security Testing (DAST), o Pruebas Dinámicas de Seguridad de Aplicaciones, es una técnica de evaluación de seguridad que se enfoca en analizar una aplicación mientras está en ejecución. A diferencia del Static Application Security Testing (SAST), que revisa el código fuente de la aplicación, DAST se centra en la interacción con la aplicación en tiempo real para identificar vulnerabilidades de seguridad que pueden ser explotadas mientras la aplicación está en funcionamiento.

Aunque el DAST ocurre en fases posteriores, algo más a la derecha en el desarrollo que el SAST, siguen siendo pruebas tempranas de seguridad previas a los despliegues productivos, y recomendable ejecutarlas lo más al inicio posible, aunque en muchas situaciones se realizan previo a pases a PRE.

¿Por qué es importante el DAST?

El DAST es crucial para identificar vulnerabilidades en aplicaciones que no pueden ser detectadas por SAST, ya que simula cómo un atacante podría interactuar con la aplicación una vez que está en funcionamiento.

Entre los beneficios del DAST, podemos destacar:

1. **Detección de vulnerabilidades en tiempo de ejecución.** DAST puede identificar vulnerabilidades que solo son reproducibles cuando la aplicación está siendo utilizada, como problemas de configuración en servidores web, validación de entrada deficiente o fallos en la autenticación.
2. **Pruebas sin acceso al código fuente.** A diferencia del SAST, DAST no requiere acceso al código fuente de la aplicación. Esto hace que sea útil en situaciones en las que el código no está disponible o es difícil de analizar (por ejemplo, aplicaciones de terceros o código cerrado).
3. **Simulación de ataques reales.** DAST reproduce el comportamiento de un atacante al interactuar con la aplicación de la misma forma que lo haría un usuario malicioso.
4. **Detección de problemas en entornos reales**. DAST evalúa la seguridad de la aplicación en su entorno real, simulando una ejecución del aplicativo en su entorno, lo que produce un menor número de falsos positivos y una mayor certeza de la seguridad de la aplicación.

Aunque DAST es esencial, existen los siguientes retos:

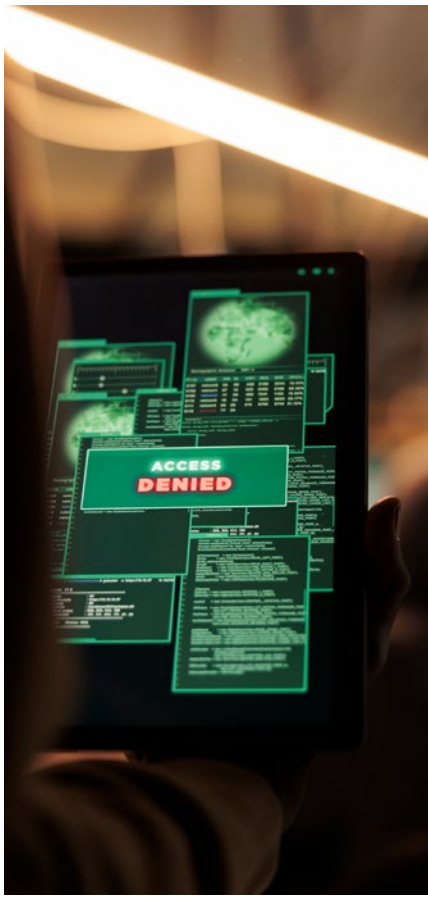
1. **Autenticación y autorización.** Dado que DAST interactúa con la aplicación en ejecución, requiere que en la ejecución de las pruebas se disponga de un set de sesiones válidas con diferentes capacidades de autenticación y autorización para poder explotar al máximo la aplicación
2. **Dispositivos móviles.** Aunque es posible realizar DAST en aplicaciones móviles, debido a la naturaleza de estas, las distintas versiones disponibles y la propia complejidad del dispositivo, hace que en numerosas ocasiones no se utilicen estas herramientas para su validación, y se realicen otro tipo de pruebas específicas en móvil. Como análisis manuales, o pentest.

3. **Dependencia de la configuración y el entorno.** El DAST puede estar limitado por la configuración del entorno de pruebas. Si no se configura correctamente o no se accede a todas las funcionalidades de la aplicación, algunas vulnerabilidades podrían no ser detectadas.
4. **Cobertura de pruebas.** En algunos casos, como es lógico, DAST depende de la batería de pruebas que se ejecute y en algunos casos, debido a esta batería, podría no detectar una vulnerabilidad si no interactúa con la aplicación de manera que revele el problema. Esto puede ocurrir si la aplicación se comporta de manera diferente según las condiciones de prueba o si el flujo de ataque no se reproduce correctamente durante el análisis.

Reglas comunes de análisis DAST

Existen distintos tipos de reglas aplicables a DAST, entre las que podemos destacar las siguientes:

1. **Pruebas de entrada y validación.** Se centran en analizar cómo la aplicación maneja las entradas del usuario y si existe una validación insuficiente de datos. Las pruebas de inyección de SQL, XSS y otros ataques de manipulación de entradas entran en esta categoría.
2. **Análisis de autenticación y autorización.** Estas pruebas se enfocan en evaluar la seguridad de los sistemas de autenticación y autorización de la aplicación, buscando fallos en los mecanismos de login, control de acceso y privilegios. Como indicábamos en los retos, este tipo de pruebas dependen de una adecuada configuración en los sistemas de autenticación y proveer de distintas tipologías de usuarios para poder cubrir todos los casos de uso.
3. **Sesiones y cookies.** Revisa la forma en que la aplicación maneja las sesiones de usuario y las cookies, verificando que se utilicen mecanismos adecuados de seguridad como tokens seguros, expiración de sesiones y protección contra secuestro de sesión.



6.5. Secretos (Gestión y detección de leaks)

La gestión adecuada de secretos (credenciales, claves API, certificados, tokens, etc.) es fundamental para mantener la seguridad en cualquier entorno de desarrollo moderno. Una fuga de secretos puede derivar en incidentes graves de seguridad, comprometiendo la integridad y confidencialidad de los sistemas.

Gestión Segura de Secretos:

- **Almacenamiento centralizado y seguro.** Utilizar herramientas especializadas de gestión de secretos (vaults).
- **Rotación automática.** Implementar políticas que permitan la rotación periódica y automática de secretos, minimizando el riesgo de compromiso prolongado.
- **Control de acceso riguroso.** Aplicar estrictos mecanismos de control de acceso, asegurando que sólo usuarios y procesos autorizados puedan acceder a los secretos específicos que necesitan.
- **Cifrado en reposo y en tránsito.** Garantizar que los secretos se mantengan cifrados tanto en almacenamiento como durante la transmisión para prevenir fugas accidentales.

Detección y Prevención de Leaks:

- **Escaneo proactivo del código.** Integrar herramientas de análisis estático específicas para detectar secretos expuestos en el código fuente antes del commit o push.
- **Monitorización continua.** Realizar escaneos continuos de repositorios para detectar secretos expuestos inadvertidamente.
- **Alertas automáticas y remediación rápida:** Implementar sistemas de alerta que notifiquen inmediatamente a los equipos correspondientes al detectar un secreto comprometido o expuesto, facilitando su rápida revocación y rotación.
- **Concienciación y formación continua.** Capacitar regularmente a los desarrolladores y equipos operativos sobre la importancia de proteger los secretos y cómo evitar su exposición accidental.
- **Especial énfasis en gitignore** **(punto 5.3 de la presente guía).**

La combinación de una gestión robusta y mecanismos proactivos de detección es esencial para mitigar los riesgos asociados a la gestión de secretos en pipelines DevSecOps, asegurando así un entorno de desarrollo seguro y confiable.

6.6. Software Composition Analysis

SCA (Software Composition Analysis) es un análisis que permite identificar vulnerabilidades conocidas (CVEs), identificar el tipo de licencia y controlar la obsolescencia en el software de terceros utilizado por la aplicación.

Integrar herramientas de SCA en los pipelines CI/CD aporta múltiples beneficios.

Uno de ellos sería la identificación temprana de vulnerabilidades, ya que, al analizar automáticamente los componentes durante el proceso de integración, se detectan posibles riesgos antes de que el software llegue a producción.

Otro es el cumplimiento de licencias, que asegura que los componentes utilizados cumplan con las licencias apropiadas, evitando posibles conflictos legales.

También se realiza un mantenimiento proactivo, ya que, al monitorear las dependencias, se facilita la actualización o reemplazo de componentes obsoletos o inseguros.

Criterios para la selección de herramientas de SCA

Para elegir la herramienta de SCA más adecuada para una organización, se deben considerar los siguientes criterios:

- **Compatibilidad tecnológica.** Verificar que la herramienta sea compatible con los lenguajes de programación y frameworks utilizados. En este punto también se puede verificar si la herramienta analiza componentes de repositorios externos, si realiza el análisis de las dependencias transitivas o si identifica si ese componente se utiliza o no dentro de la aplicación y dónde.

- **Capacidad de integración con CI/CD.** Asegurar que la herramienta pueda integrarse correctamente en los pipelines de desarrollo sin penalizar demasiado el tiempo de despliegue. Comprobar si la herramienta soporta la implementación de *Security Gates*[22]. Es importante también que disponga de una API que permita realizar integraciones personalizadas.

- **Frecuencia y actualización.** Es fundamental que la herramienta utilice bases de datos actualizadas con información reciente sobre vulnerabilidades conocidas.

- **Capacidad de generación de reportes y alertas.** Evaluar la facilidad con la que la herramienta presenta resultados y si proporciona alertas o permite exportar los datos para crear cuadros de mando personalizados.

- **Modelo de licenciamiento.** Considerar si la herramienta es de código abierto o comercial, así como el coste asociado a su implementación y mantenimiento.

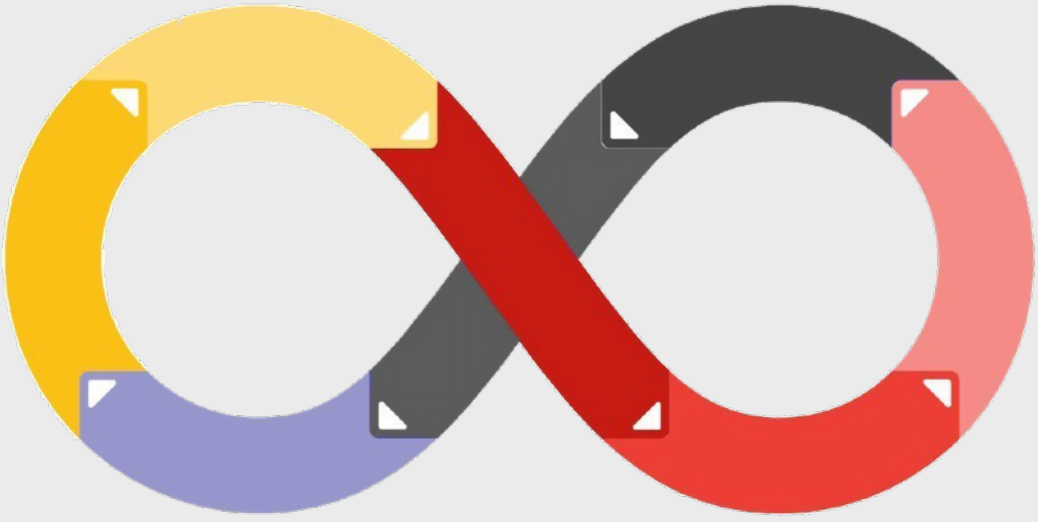
6.7. Otras herramientas

Dependiendo de la organización y sus necesidades, puede ser fundamental considerar otros tipos de análisis que complementen las tecnologías explicadas en este punto y que se pueden integrar, también, en el pipeline:

Interactive Application Security Testing (IAST)	Runtime Application Self-Protection (RASP)	Gestores de secretos y credenciales	Escaneo de infraestructura como código (IaC)
Evalúa la seguridad de una aplicación mientras se ejecuta en un entorno de prueba, proporcionando detección en tiempo real de vulnerabilidades en el código y dependencias.	Implementa mecanismos de seguridad dentro de la propia aplicación en tiempo de ejecución, permitiendo detectar y mitigar amenazas sin intervención externa.	Son herramientas que protegen la información sensible utilizada en los pipelines, reduciendo riesgos de exposición de credenciales.	Detecta configuraciones inseguras en archivos de infraestructura como Terraform y Kubernetes, mejorando la seguridad desde el diseño.

22 Un Security Gate es un punto de control dentro del pipeline CI/CD donde se realizan verificaciones de seguridad antes de permitir que el código avance a la siguiente fase del ciclo de desarrollo. Su función principal es garantizar que solo el software que cumple con los estándares de seguridad definidos se pueda desplegar hacia entornos de producción.

7 Fase 5: Open Source.



> *El uso de software de código abierto (OSS, por sus siglas en inglés) es una práctica extendida en el desarrollo de software, ayudando a estandarizar desarrollos y agilizando el tiempo de entrega. Sin embargo, su integración requiere la implementación de medidas de seguridad y cumplimiento para mitigar riesgos.*

En un análisis realizado por OWASP publicado en febrero de 2024, se establecieron los 10 mayores riesgos de seguridad derivados del uso de OSS. Durante dicho análisis se descubrió que el 89% de los repositorios analizados contienen componentes con versiones de hacía más de 4 años y el 91% de esos repositorios contienen componentes que no han recibido actualizaciones desde hace más de 2 años.

Imagen 6: ***Ranking de riesgos para open source***

Por lo tanto, la atención a la revisión de los componentes que se incluyen desde terceros en el software es clave para mejorar la seguridad de las aplicaciones en producción. La evaluación, monitoreo y licenciamiento adecuado de los componentes OSS son factores clave para garantizar su uso seguro y efectivo.

7.1. Evaluación y monitoreo de componentes OSS

Integrar dependencias de código abierto añade un riesgo al software desarrollado en la empresa, dichos componentes forman parte de nuestro ecosistema no regulado por la compañía y por lo tanto se pierde control sobre ellos. Para gestionar los riesgos derivados de esta integración, la organización debe:

Implementar herramientas de escaneo de vulnerabilidades en dependencias de código abierto

Estas herramientas permiten identificar vulnerabilidades conocidas en librerías y frameworks utilizados en el software. Dichas herramientas deberían estar lo más cerca posible del trabajo diario del desarrollador, para darle visibilidad y fomentar el uso de buenas prácticas.

Establecer políticas de selección y aprobación de componentes OSS

Definir criterios para la adopción de software OSS, asegurando que provenga de fuentes confiables, que tenga mantenimiento activo y una comunidad de soporte. Cuando se implementen dichos componentes, se recomienda referenciar la versión y la firma deseados para evitar una posible alteración no deseada de la librería. Para poder validar su uso es recomendable evaluar la madurez del proyecto OSS y la frecuencia con la que se publican parches de seguridad. Por ejemplo, Tensorflow[23] es una herramienta de machine learning ampliamente reconocida, con una comunidad activa y actualizaciones periódicas, por lo tanto, sería un buen candidato para pasar el filtro.

Realizar auditorías periódicas

La organización debe revisar periódicamente los componentes OSS en uso, asegurándose de que se actualicen con las últimas versiones y que no presenten licencias incompatibles con el proyecto. Se recomienda la adopción de Software Composition Analysis (SCA) para facilitar esta tarea. Sin este tipo de herramientas, la tarea de auditar los componentes se haría imposible dado que muchos de los componentes de OSS suelen tener subdependencias.

Implementar controles de seguridad en el pipeline de CI/CD

La clave de la implementación de todos los puntos anteriores incluye tener un flujo definido de despliegue. Añadir controles bloqueantes en este punto es la forma más efectiva para una vez establecidas las políticas y haber ganado visibilidad sobre los componentes, limita que lleguen a desplegarse OSS que superen el apetito de riesgo definido en producción.

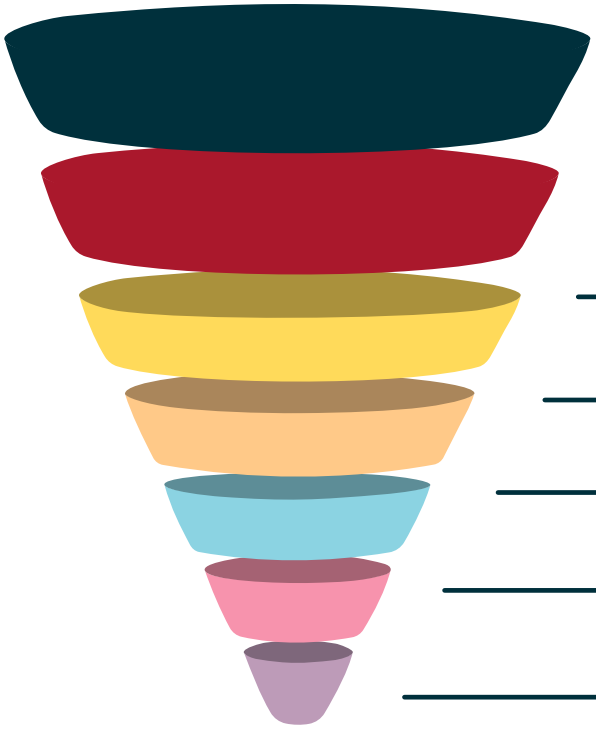
23 https://github.com/tensorflow/tensorflow

7.2. Prácticas para mantener OSS seguro en el SDLC

Para garantizar la seguridad de los componentes OSS a lo largo del ciclo de vida del desarrollo, se recomienda mantener un inventario actualizado de los componentes OSS utilizados. Un Software Bill of Materials (SBOM) permite documentar todas las dependencias de código abierto dentro de un proyecto, facilitando su gestión y monitoreo de seguridad. La NIST SP 800-204D enfatiza la importancia de actualizar el SBOM en cada iteración del desarrollo para reflejar cambios en dependencias.

> *Además, es fundamental aplicar parches y actualizaciones de seguridad de manera regular. La falta de actualizaciones en dependencias OSS es una de las principales causas de vulnerabilidades en aplicaciones. Se recomienda automatizar la gestión de parches con herramientas como Renovate o Dependabot. Según NIST, las organizaciones deben definir estrategias de mitigación en caso de retraso en la aplicación de parches críticos, lo que probablemente ocurra con regularidad.*

Para que la tarea de actualizar las dependencias no sea inabarcable, es necesario filtrar componentes tienen mayor riesgo. Dicho filtro no solo depende de la gravedad de la vulnerabilidad introducida, sino también de la facilidad de explotación de la misma o de si la función que contiene el código inseguro está siendo llamada en algún momento. Un ejemplo de este proceso de filtrado podría ser:



1. **Establecer el SBOM,** incluyendo todos los componentes y realizar el análisis de SCA con sus vulnerabilidades.
2. **Comprobar** cuáles de esas **vulnerabilidades** tienen ya un **parche.**
3. **Establecer el OSS** que forma parte del software que será desplegado en producción.
4. **Analizar si los métodos que contienen la vulnerabilidad** son llamados desde el código de la aplicación.
5. **Definir a partir de que scoring de explotabilidad** considera la organización que es un riesgo aceptable.
6. **Abordar primero** aquellos **desarrollos que se encuentren en activo.**
7. **Finalmente, priorizar aquellos desarrollos** que tengan mayor valor para el negocio.

Imagen 7: ***Embudo de Decisión para la Gestión de Vulnerabilidades OSS***

En el caso de uso de contenedores es una práctica esencial extender este análisis. Se recomienda implementar escaneo de imágenes de contenedores OSS para identificar vulnerabilidades y aplicar controles de seguridad antes del despliegue. De esta manera, la infraestructura como código (IaC) debería seguir el mismo ciclo de vida de análisis que el resto de los componentes de código de la organización.

Por último, fomentar la participación en comunidades OSS contribuye a recibir alertas tempranas sobre vulnerabilidades y participar en la mejora de la seguridad de los proyectos utilizados. La colaboración abierta permite detectar y corregir errores antes de que sean explotados.

7.3. Modelos de licenciamiento Open Source e impacto en los desarrollos

El licenciamiento de software de código abierto puede influir en la forma en que se distribuye y utiliza el software. **Dependiendo del licenciamiento que tenga el proyecto, podrá ser usado o no como parte de un producto comercializable y, por lo tanto, las organizaciones deben considerar:**

Diferencias entre licencias permisivas y restrictivas

Licencias como MIT y Apache 2.0 permiten reutilizar y modificar el código con pocas restricciones, mientras que licencias como GPL y AGPL imponen la obligación de liberar modificaciones bajo la misma licencia, lo que puede limitar el uso del software en productos propietarios.

Compatibilidad entre licencias

Es fundamental analizar la compatibilidad entre diferentes licencias OSS y la estrategia de negocio de la organización para evitar conflictos legales.

Obligación de divulgación del código fuente

Algunas licencias, como la GPL, requieren que cualquier software derivado sea liberado bajo la misma licencia, lo que puede impactar la propiedad intelectual de un producto desarrollado.

Para mayor comprensión, se pueden agrupar los principales modelos de licenciamento de OSS en tres, cada cual con implicaciones diferentes:

Licencias permisivas

Permiten el uso, modificación y distribución del software con pocas restricciones. Ejemplos:

- **MIT:** puede ser usada en proyectos comerciales sin obligación de liberar modificaciones.
- **Apache 2.0:** similar a MIT, pero incluye protección de patentes.
- **BSD:** flexibilidad total para uso privado y comercial.

Licencias copyleft

Obligan a compartir modificaciones bajo la misma licencia. Ejemplos:

- **GPL:** requiere que cualquier software derivado también sea GPL.
- **AGPL:** extiende los requisitos de GPL a software ofrecido como servicio en la nube.
- **LGPL:** permite la vinculación con software propietario, pero modificaciones al código fuente deben ser liberadas.

Licencias de código abierto con restricciones:

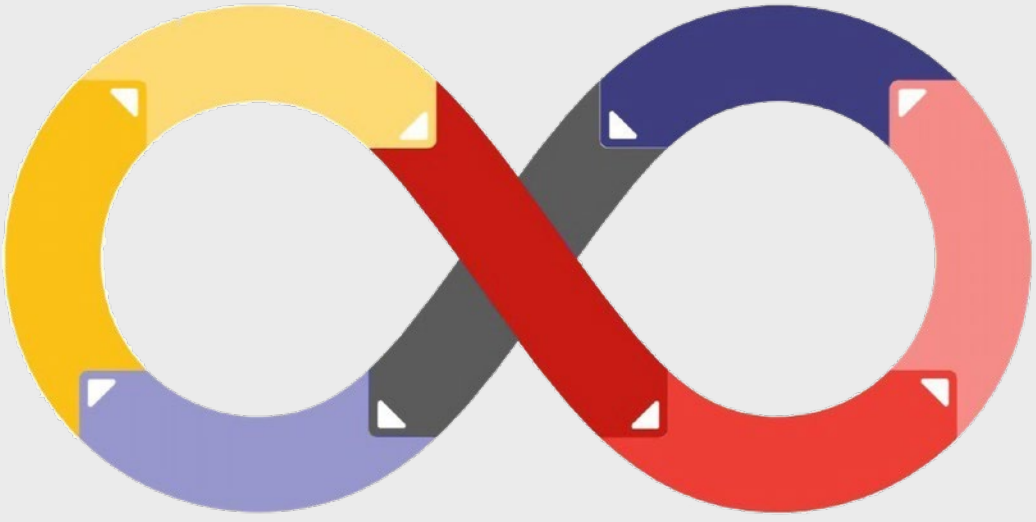
Imponen ciertas condiciones sobre su uso y distribución. Ejemplos:

- **MPL (Mozilla Public License):** permite combinar software propietario con código abierto, pero requiere que los archivos modificados se liberen.
- **EPL (Eclipse Public License):** similar a MPL, pero con restricciones específicas para integraciones comerciales.

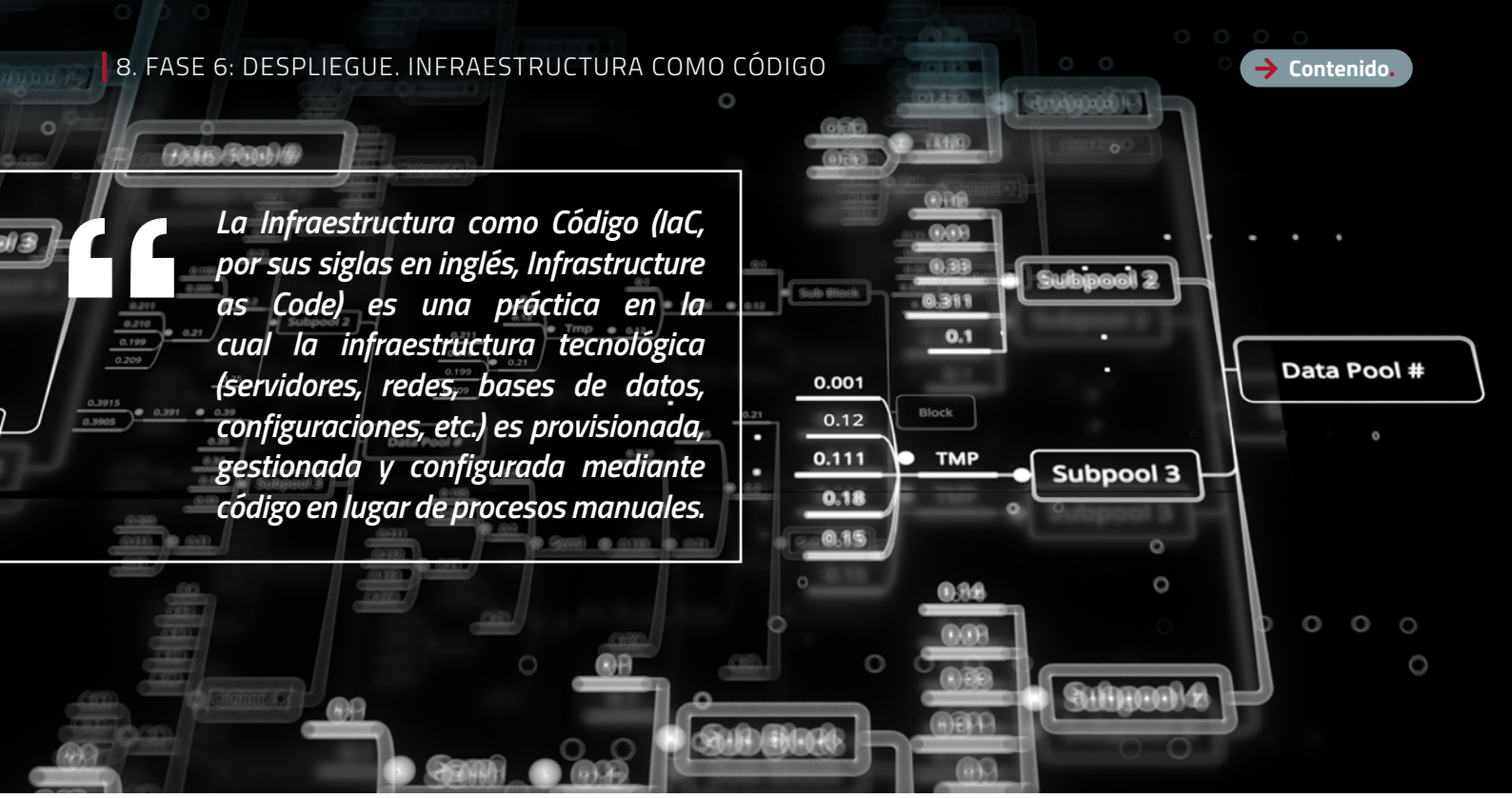
El desconocimiento o la falta de cumplimiento con los modelos de licenciamiento OSS puede exponer a la organización a diversos problemas legales y comerciales. El uso indebido de licencias restrictivas puede obligar a liberar código propietario, afectando la propiedad intelectual y la ventaja competitiva de la empresa. Además, la falta de compatibilidad entre licencias puede generar conflictos que dificulten la distribución del software. En algunos casos, incumplimientos pueden derivar en sanciones legales, litigios y daños a la reputación de la organización, aunque estos últimos puntos son más comunes en el entorno anglosajón. Por ello, resulta fundamental contar con un proceso de revisión y validación de licencias antes de incorporar componentes OSS en proyectos comerciales.

Para gestionar el cumplimiento de licencias de manera eficiente, se recomienda realizar el análisis de licenciamiento de forma automatizada. Este análisis debe integrarse en el proceso de elaboración del SBOM, permitiendo identificar posibles incompatibilidades y evaluar riesgos legales antes de la implementación del software. Herramientas como FOSSology y Black Duck Software pueden facilitar esta tarea al proporcionar una evaluación detallada de las licencias OSS utilizadas en un proyecto.

8 Fase 6: Despliegue. Infraestructura como Código.



Contenido.



> *La Infraestructura como Código (IaC, por sus siglas en inglés, Infrastructure as Code) es una práctica en la cual la infraestructura tecnológica (servidores, redes, bases de datos, configuraciones, etc.) es provisionada, gestionada y configurada mediante código en lugar de procesos manuales.*

Este enfoque permite automatizar la implementación y configuración de recursos de TI a través de scripts, lenguajes descriptivos y herramientas especializadas como Terraform, Ansible, Puppet, Chef y CloudFormation, entre otras.

Beneficios de IaC:

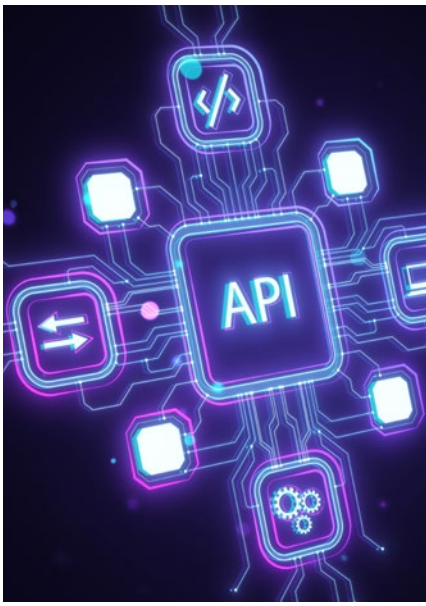
- **Automatización.** Reduce errores humanos asociados con tareas repetitivas o manuales.
- **Consistencia y repetibilidad.** Permite reproducir la infraestructura de forma exacta en múltiples entornos.
- **Velocidad y eficiencia.** Facilita despliegues rápidos y ágiles, permitiendo ciclos más cortos en el desarrollo y entrega de software.
- **Colaboración mejorada.** Facilita la colaboración entre equipos de desarrollo y operaciones, fomentando prácticas DevOps y DevSecOps.
- **Auditoría y trazabilidad.** El código versionado permite una gestión transparente y auditable de cambios en la infraestructura.

8.1. Mejores prácticas de Seguridad en IaC

El uso de la metodología IaC también introduce nuevos desafíos de seguridad que deben abordarse proactivamente. Ignorar las consideraciones de seguridad en las implementaciones de IaC puede exponer a las organizaciones a riesgos significativos, desde configuraciones erróneas hasta la exposición de secretos sensibles y vulnerabilidades explotables.

Este punto se centra en delinear las mejores prácticas de seguridad esenciales que deben integrarse en todo el ciclo de vida de la IaC. Al adoptar estas prácticas, las organizaciones pueden construir infraestructuras robustas, resilientes y seguras desde su concepción, minimizando la superficie de ataque y reduciendo el potencial de incidentes de seguridad costosos y disruptivos. La implementación consciente y consistente de estas directrices es crucial para aprovechar al máximo los beneficios de la IaC al tiempo que se mantiene una postura de seguridad sólida.

Control de Versiones



Como ya se ha explicado es imprescindible almacenar todas las plantillas de IaC en sistemas de control de versiones como Git 1. Esta práctica permite rastrear cada modificación realizada en la infraestructura, facilita la colaboración entre equipos y proporciona la capacidad de revertir a versiones anteriores si es necesario 2. La infraestructura definida como código debe tratarse con el mismo rigor que el código de las aplicaciones 1. El control de versiones no solo ayuda a gestionar los cambios, sino que también actúa como un componente esencial para las auditorías de seguridad y la respuesta a incidentes, ya que ofrece un historial completo de las configuraciones de la infraestructura. En caso de un incidente de seguridad o una configuración errónea, la capacidad de rastrear los cambios y volver a una versión segura anterior es crucial.

Escaneo de IaC en Busca de Vulnerabilidades y Configuraciones Incorrectas

Se debe implementar el escaneo automatizado de plantillas de IaC para detectar configuraciones de seguridad incorrectas, secretos codificados y violaciones de cumplimiento normativo. Para ello, se pueden utilizar herramientas como Terrascan, Checkov, Trivy y KICS 1. Es fundamental integrar estos escaneos en la canalización de integración y entrega continua (CI/CD) para identificar los problemas en las etapas tempranas del desarrollo (*Shift-Left*) 1. La detección temprana de vulnerabilidades en IaC evita el despliegue de infraestructura insegura, lo que reduce significativamente la superficie de ataque y el costo de la remediación. Abordar las vulnerabilidades en la fase de IaC es mucho más económico y menos disruptivo que corregirlas en entornos de producción 14. Al integrar el escaneo de forma temprana, las organizaciones pueden evitar que las configuraciones incorrectas se conviertan en debilidades explotables.

Gestión de Secretos

Es crucial evitar la codificación de información sensible (claves de API, contraseñas) directamente en las plantillas de IaC 1. En su lugar, se deben utilizar herramientas de gestión de secretos dedicadas como HashiCorp Vault, AWS Secrets Manager, Azure Key Vault o Kubernetes Secrets 14. La gestión segura de secretos es primordial, ya que las credenciales expuestas pueden conducir a graves brechas de seguridad, incluido el acceso no autorizado a sistemas y datos críticos. Los secretos codificados en IaC representan un riesgo significativo 10. Los atacantes pueden encontrar fácilmente estos secretos en el control de versiones o en los archivos de configuración. El uso de soluciones de gestión de secretos dedicadas proporciona cifrado, control de acceso y registro de auditoría para la información confidencial.

Principio de Mínimo Privilegio (PoLP)

Se debe otorgar solo el mínimo de permisos necesarios a los usuarios, servicios y recursos definidos en IaC 1. Es recomendable implementar políticas de Identity and Access Management (IAM) detalladas en los entornos de nube 1. Adherirse al principio de mínimo privilegio limita el daño potencial en caso de una brecha de seguridad al restringir el acceso y las capacidades del atacante. Un acceso demasiado permisivo aumenta la superficie de ataque 14. Si una cuenta o recurso se ve comprometido, el potencial de movimiento lateral y exfiltración de datos es mucho mayor. PoLP garantiza que, incluso si ocurre una brecha, el impacto esté contenido.

Revisiones de Código

Es fundamental realizar revisiones de código periódicas por pares del código de IaC para identificar posibles fallos de seguridad y configuraciones incorrectas 1. Esta práctica también promueve el intercambio de conocimientos y refuerza las prácticas de seguridad dentro del equipo 1. La revisión humana complementa el escaneo automatizado al detectar errores lógicos y problemas de seguridad específicos del contexto que las herramientas automatizadas podrían pasar por alto. Si bien las herramientas automatizadas son esenciales, no son infalibles. Las revisiones de código aprovechan el conocimiento y la experiencia colectiva del equipo para identificar vulnerabilidades de seguridad sutiles y garantizar el cumplimiento de las mejores prácticas.

8.2. Ejemplos prácticos de Seguridad para Kubernetes y Docker

8.2.1. Protección de Despliegues de Kubernetes con IaC

Herramientas de IaC para Kubernetes

Los manifiestos YAML de Kubernetes son una forma inherente a IaC. Helm se utiliza para la gestión de paquetes y la automatización de despliegues. Terraform (o equivalente) permite el aprovisionamiento y la gestión de clústeres y recursos de Kubernetes.

Ejemplos de Seguridad

Políticas/Estándares de Seguridad de Pods (PSP/PSS): definir y aplicar contextos de seguridad para pods utilizando IaC (por ejemplo, kubernetes_pod_security_policy de Terraform). Esto evita contenedores con privilegios y obliga a que se ejecuten como usuarios no root. Aplicar PSP/PSS mediante IaC garantiza que todos los pods desplegados dentro del clúster cumplan con las líneas base de seguridad definidas, lo que reduce el riesgo de escape de contenedores y escalada de privilegios. Los contenedores que se ejecutan con privilegios excesivos pueden representar un riesgo de seguridad significativo. Al definir y aplicar políticas de seguridad a nivel de pod a través de IaC, las organizaciones pueden restringir las capacidades de los contenedores y limitar el impacto de una posible vulneración.

Políticas de Red: implementar la segmentación de red y controlar el flujo de tráfico entre pods y namespaces utilizando IaC (por ejemplo, kubernetes_network_policy de Terraform). Esto restringe el tráfico tanto de entrada como de salida en función de las reglas definidas. Definir políticas de red como código garantiza que el aislamiento de la red se aplique de manera consistente en todo el entorno de Kubernetes, lo que limita el radio de explosión de un incidente de seguridad. En un entorno de Kubernetes, controlar la comunicación de red entre diferentes componentes es crucial. IaC permite la definición de políticas de red detalladas que restringen el tráfico solo a lo necesario, evitando el acceso no autorizado y el movimiento lateral.

RBAC (Control de Acceso Basado en Roles): configurar los recursos de RBAC (Roles, ClusterRoles, RoleBindings, ClusterRoleBindings) utilizando IaC para definir permisos detallados para usuarios y cuentas de servicio. Implementar RBAC a través de IaC garantiza que el principio de mínimo privilegio se aplique a todas las interacciones dentro del clúster de Kubernetes, lo que limita el impacto potencial de las credenciales comprometidas. Kubernetes RBAC controla quién puede realizar qué acciones sobre qué recursos. Al definir estos permisos en IaC, las organizaciones pueden automatizar la aplicación del mínimo privilegio, asegurando que los usuarios y servicios solo tengan el acceso necesario.

Gestión de Secretos: utilizar Kubernetes Secrets o integrarse con gestores de secretos externos (por ejemplo, HashiCorp Vault a través de Vault Secrets Operator) utilizando IaC para gestionar de forma segura los datos confidenciales. Gestionar secretos de forma segura dentro de Kubernetes utilizando IaC evita la codificación y garantiza que la información confidencial esté protegida mediante cifrado y controles de acceso. Kubernetes Secrets proporciona un mecanismo para almacenar y gestionar información confidencial. Integrar la gestión de secretos en los flujos de trabajo de IaC garantiza que los secretos se gestionen correctamente durante el despliegue y el tiempo de ejecución.

Límites de Recursos: definir las solicitudes y los límites de recursos (CPU, memoria) para los contenedores en los manifiestos de despliegue (YAML) o Terraform para evitar el agotamiento de recursos y posibles ataques de denegación de servicio. Establecer límites de recursos mediante IaC garantiza que los contenedores no puedan consumir recursos excesivos, lo que contribuye a la estabilidad y seguridad del clúster de Kubernetes. Sin límites de recursos, un solo contenedor con un comportamiento incorrecto podría consumir todos los recursos disponibles, lo que afectaría a otras aplicaciones que se ejecutan en el clúster. IaC permite la definición de estos límites, lo que garantiza una asignación justa de recursos y evita el agotamiento de los mismos.

Evitar el Namespace Default: utilizar IaC para crear y desplegar recursos en namespaces específicos en lugar del namespace default para mejorar el aislamiento y la seguridad. El aislamiento de namespaces, gestionado a través de IaC, ayuda a separar lógicamente diferentes aplicaciones y equipos dentro de un clúster de Kubernetes, lo que mejora la seguridad y la capacidad de gestión. El namespace default en Kubernetes suele ser menos restrictivo. Crear y utilizar namespaces dedicados para diferentes cargas de trabajo proporciona una separación lógica y permite la aplicación de políticas de seguridad más específicas.

Mantener el Host y Contenedor Actualizados: automatizar las actualizaciones del motor de Docker y del sistema operativo del host utilizando IaC (por ejemplo, scripts dentro de UserData de CloudFormation o playbooks de Ansible). Actualizar regularmente Docker y el sistema host mediante la automatización garantiza que las vulnerabilidades conocidas se parchen rápidamente, lo que reduce el riesgo de explotación. El software obsoleto a menudo contiene vulnerabilidades de seguridad. Automatizar el proceso de actualización garantiza que los sistemas ejecuten las últimas versiones con los parches de seguridad necesarios.

Ejecutar Docker en Modo Rootless: configurar Docker para que se ejecute sin privilegios de root utilizando scripts o herramientas de gestión de configuración. Ejecutar Docker en modo rootless limita el impacto de un contenedor comprometido al evitar la escalada de privilegios en el host. Si un proceso de contenedor se ve comprometido y se ejecuta como root, puede obtener el control de todo el sistema host. El modo rootless mitiga este riesgo al ejecutar el demonio de Docker y los contenedores con privilegios de usuario no root.

Utilizar Docker Content Trust: habilitar Docker Content Trust (DCT) para verificar la autenticidad e integridad de las imágenes de contenedores antes del despliegue. Garantizar la autenticidad de las imágenes de contenedores a través de DCT evita el despliegue de software malicioso o comprometido. Los atacantes podrían intentar inyectar código malicioso en las imágenes de contenedores. DCT utiliza firmas digitales para verificar el editor y garantizar que la imagen no ha sido manipulada.

Linting de Dockerfiles: integrar linters como Hadolint en los pipelines de CI/CD para analizar los Dockerfiles en busca de posibles configuraciones incorrectas y problemas de seguridad. El linting de Dockerfiles como parte del proceso de IaC ayuda a identificar y remediar prácticas inseguras antes de que se construyan y despliegen las imágenes de contenedores. Los Dockerfiles definen cómo se construyen las imágenes de contenedores. Las herramientas de linting pueden identificar errores de seguridad comunes en los Dockerfiles, como el uso de imágenes base inseguras o la exposición de puertos innecesarios.

Utilizar Etiquetas Fijas para la Inmutabilidad: especificar etiquetas fijas para las imágenes de contenedores en las configuraciones de despliegue (por ejemplo, manifiestos de Kubernetes, definiciones de tareas de ECS) para garantizar despliegues consistentes y predecibles. Evitar el uso de etiquetas mutables como 'latest'. El uso de etiquetas fijas garantiza que siempre se despliegue la misma versión de una imagen de contenedor, lo que evita actualizaciones o reversiones no deseadas que pudieran introducir vulnerabilidades. Las etiquetas mutables pueden generar despliegues inconsistentes, ya que la imagen a la que apuntan podría cambiar con el tiempo. El uso de etiquetas fijas garantiza que siempre se utilice una versión específica y probada de la imagen.

Escaneo de Imágenes Docker en Busca de Vulnerabilidades: integrar herramientas de escaneo de imágenes de contenedores en el pipeline del CI/CD para identificar vulnerabilidades en las capas de la imagen. El escaneo regular de imágenes Docker en busca de vulnerabilidades conocidas permite la detección y remediación temprana antes del despliegue. Las imágenes de contenedores a menudo incluyen bibliotecas y dependencias de terceros que pueden contener vulnerabilidades. El escaneo de estas imágenes ayuda a identificar y abordar estas debilidades.

8.3. Políticas de seguridad automatizadas

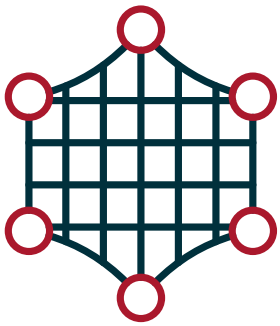
Implementación de Política como Código (PaC)

Es necesario definir las políticas de seguridad en un formato legible por máquina utilizando lenguajes como Rego (OPA), YAML (Checkov) o Python. Implementar políticas de seguridad como código permite una aplicación consistente y automatizada, lo que garantiza que los despliegues de infraestructura cumplan con los estándares de seguridad de la organización. La aplicación manual de políticas es propensa a errores y difícil de escalar. PaC permite a las organizaciones codificar sus requisitos de seguridad y verificar automáticamente el cumplimiento durante el proceso de despliegue.

Integración con CI/CD

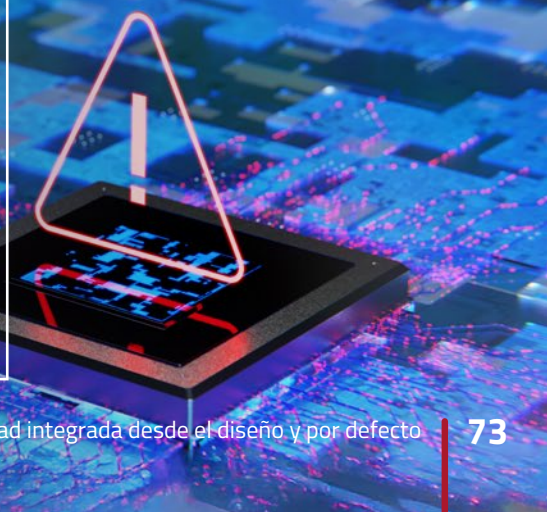
Las herramientas de escaneo y aplicación de políticas deben integrarse en etapas concretas del CI/CD (por ejemplo, pre-commit hooks, etapa de construcción, etapa de despliegue). Se deben automatizar los fallos de construcción o las alertas en función de las violaciones de políticas. Automatizar las comprobaciones de seguridad dentro del CI/CD garantiza una validación de seguridad continua y evita el despliegue de infraestructura no conforme. Al incorporar las comprobaciones de seguridad directamente en el flujo de trabajo de desarrollo y despliegue, las organizaciones pueden detectar y corregir los problemas de forma temprana, lo que reduce el riesgo de desplegar infraestructura vulnerable en producción.

Ejemplos de Políticas de Seguridad Automatizadas



- **Prevención de Recursos Accesibles Públicamente.** Políticas para garantizar que los recursos críticos como bases de datos o buckets de almacenamiento no se expongan inadvertidamente a la internet pública.
- **Aplicación del Cifrado.** Políticas para verificar que los datos sensibles en reposo y en tránsito estén correctamente cifrados.
- **Restricción del Tráfico de Entrada/Salida.** Políticas para aplicar la segmentación de red y controlar el flujo de tráfico hacia y desde los recursos.
- **Obligatoriedad del Etiquetado de Recursos.** Políticas para garantizar que todos los recursos desplegados estén correctamente etiquetados para su identificación, gestión y cumplimiento.
- **Limitación de Permisos IAM.** Políticas para aplicar el principio de mínimo privilegio para los roles y usuarios de IAM creados.

Automatizar la aplicación de estas políticas de seguridad comunes reduce significativamente la probabilidad de errores humanos y garantiza una postura de seguridad coherente en toda la infraestructura. La configuración y verificación manual de estos ajustes de seguridad pueden llevar mucho tiempo y ser propensas a errores. La automatización a través de PaC garantiza que estos controles críticos se apliquen de manera consistente.



Herramientas para Automatizar Políticas de Seguridad

La automatización de políticas de seguridad es un componente clave dentro de las prácticas DevSecOps, facilitando la detección temprana y mitigación efectiva de vulnerabilidades y riesgos de cumplimiento. A continuación, se presentan tipos de herramientas esenciales para automatizar la gestión, implementación y monitorización de estas políticas:

Motores de Políticas Genéricos

Soluciones que permiten definir, gestionar y aplicar políticas de seguridad personalizadas de forma centralizada y estandarizada, ofreciendo flexibilidad y capacidad de adaptación a diferentes contextos y requisitos específicos.

Herramientas de Análisis Estático de Infraestructura

Permiten escanear automáticamente el código y configuraciones de infraestructura en busca de configuraciones inseguras, vulnerabilidades y desviaciones respecto a las políticas predefinidas. Posibilitan la creación de reglas específicas que garantizan una evaluación constante del cumplimiento normativo y seguridad en los recursos desplegados.

Herramientas especializadas en Análisis de Código y Configuraciones IaC

Proveen análisis exhaustivos específicos sobre configuraciones escritas como infraestructura como código, detectando y corrigiendo vulnerabilidades, configuraciones incorrectas y posibles problemas de cumplimiento normativo.

Soluciones Integrales de Escaneo de Vulnerabilidades y Configuración

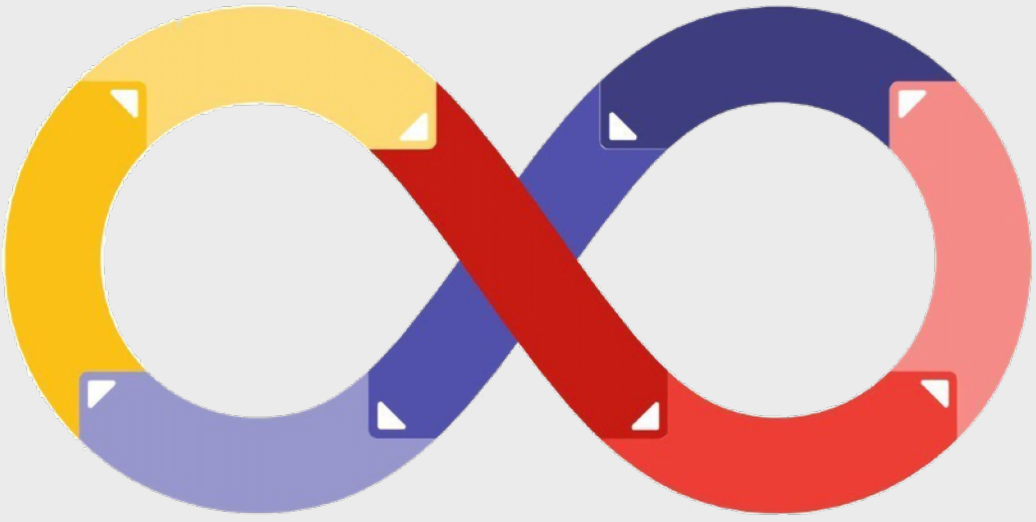
Escáneres capaces de detectar vulnerabilidades conocidas en contenedores, imágenes, dependencias, y simultáneamente identificar malas prácticas y configuraciones erróneas en scripts y archivos de configuración IaC.

Herramientas Nativas de Gestión de Políticas en Entornos Cloud

Plataformas integradas en los entornos de proveedores cloud que permiten aplicar políticas directamente sobre los recursos en la nube, asegurando un cumplimiento constante con estándares internos y regulaciones externas. Facilitan la gobernanza efectiva y continua mediante reglas predefinidas y automatización integrada dentro del ecosistema específico del proveedor.

> *Al aplicar estas categorías de herramientas, las organizaciones pueden establecer mecanismos robustos para automatizar, mantener y supervisar continuamente la conformidad con sus políticas de seguridad, minimizando riesgos y garantizando una postura de seguridad sólida y proactiva en todo momento.*

9 Fase 7: Monitorización. Application Security Posture Management (ASPM).



*La Gestión de la Postura de Seguridad de las Aplicaciones (en adelante, **"ASPM"**), es un concepto relativamente reciente, que ha surgido como una evolución gradual de la seguridad en el ciclo de vida del desarrollo de aplicaciones*

Se trata de un enfoque integral que permite a las organizaciones gestionar la seguridad de las aplicaciones a lo largo de todo su ciclo de vida, y cuyo objetivo principal es proporcionar una visibilidad completa y continua de la postura de seguridad de las aplicaciones, permitiendo a los equipos de desarrollo y seguridad, identificar, evaluar y mitigar riesgos de manera proactiva y eficiente.

Una solución ASPM recopila, unifica y correlaciona información de diferentes orígenes de datos a lo largo del ciclo de vida del desarrollo del software (SDLC), con el objetivo de identificar vulnerabilidades y debilidades, y evaluar el cumplimiento de las políticas de seguridad y las mejores prácticas.

El uso de una solución ASPM ofrece una serie de beneficios significativos para las organizaciones, que impactan directamente en la mejora de su seguridad y la eficiencia de sus procesos de desarrollo. Algunos de los **beneficios** más importantes pueden ser:

Mejora de la visibilidad y el control

Proporciona una visión integral de la postura de seguridad de la aplicación, incluyendo vulnerabilidades, configuraciones incorrectas y riesgos de cumplimiento. Permite la identificación automática de aplicaciones en diversos entornos, tanto locales como en la nube.

Detección y corrección temprana de vulnerabilidades

Permite a los equipos identificar y priorizar riesgos de seguridad de forma temprana en el SDLC, lo que facilita su mitigación antes de que la aplicación se despliegue en producción.

Priorización y gestión de riesgos

Ayuda a las organizaciones a priorizar las vulnerabilidades en función de su gravedad y su impacto potencial, permitiendo una gestión de riesgos más eficaz, centrando los esfuerzos en las áreas de mayor preocupación.

Cumplimiento Normativo

Ayuda a las organizaciones a cumplir con los requisitos de seguridad de diversas normativas y estándares de seguridad. Facilita la generación de informes de cumplimiento, lo que simplifica los procesos de auditoría.

Colaboración entre Equipos

Facilita la comunicación y colaboración entre los equipos de desarrollo y seguridad, promoviendo una cultura de "seguridad desde el diseño".

Reducción de costes

Al detectar y corregir las vulnerabilidades en las primeras etapas del desarrollo, ASPM ayuda a reducir los costes asociados a las correcciones posteriores. Minimiza el riesgo de incidentes de seguridad que son altamente costosos, como las filtraciones de datos.

Automatización y Escalabilidad

ASPM se integra con herramientas y procesos de CI/CD, lo que permite automatizar la evaluación de la seguridad y escalar la gestión de riesgos a medida que la aplicación crece y evoluciona. Orquesta diversas herramientas de seguridad, proporcionando una visión unificada de los resultados.

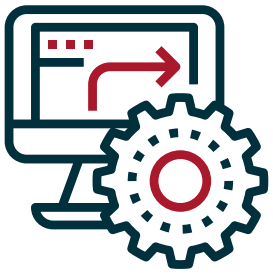
9.1. Integración en entornos CI/CD

Para poder beneficiarnos de todas las ventajas que ofrece un ASPM, es fundamental integrar este modelo en los entornos de Integración Continua y Entrega Continua (**CI/CD**) corporativos.

Esto nos garantiza integrar la seguridad en cada una de las fases del ciclo de vida del desarrollo del software, para no solo identificar y mitigar vulnerabilidades de forma temprana, sino también para fomentar una cultura de seguridad dentro de los equipos de desarrollo.

A continuación, se detallan algunas estrategias clave para integrar ASPM en CI/CD:

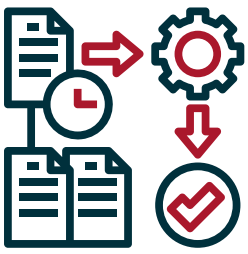
Automatización de Pruebas de Seguridad en las Aplicaciones



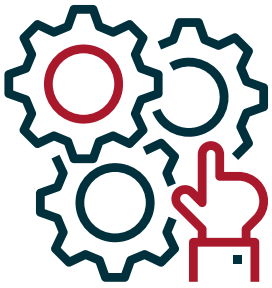
- **Análisis Estático de Código (SAST):** ASPM puede integrar herramientas SAST para escanear el código fuente en busca de vulnerabilidades conocidas, errores de codificación y desviaciones de las normas de seguridad. Esta detección temprana reduce la deuda técnica y evita que las vulnerabilidades se propaguen a las fases posteriores.
- **Análisis Dinámico de Aplicaciones (DAST):** al simular ataques contra la aplicación en ejecución, DAST identifica vulnerabilidades que pueden no ser aparentes en el código estático. ASPM puede orquestar pruebas DAST en diferentes entornos para una cobertura completa.
- **Análisis de Composición de Software (SCA):** mediante el seguimiento de las dependencias de terceros, SCA detecta vulnerabilidades conocidas en bibliotecas y marcos de trabajo. ASPM puede integrar herramientas SCA para alertar sobre componentes vulnerables y sugerir actualizaciones o parches.
- **Pruebas de Infraestructura como Código (IaC):** al escanear plantillas y scripts de IaC, se pueden identificar configuraciones incorrectas y vulnerabilidades en la infraestructura de la nube. ASPM puede incluir pruebas IaC en el pipeline de CI/CD para garantizar la seguridad desde el principio.

Integración con Herramientas de Desarrollo

- **Comentarios en Tiempo Real:** ASPM puede integrarse con IDEs y herramientas de colaboración para proporcionar comentarios instantáneos a los desarrolladores sobre problemas de seguridad en su código. Esto permite correcciones rápidas y reduce el tiempo de remediación.
- **Gestión de Incidencias de Seguridad:** al integrar ASPM con sistemas de seguimiento de incidencias, las vulnerabilidades pueden ser tratadas como cualquier otro error de software, asegurando que sean asignadas, rastreadas y resueltas de manera oportuna.
- **Umbrales de Calidad de Seguridad:** ASPM puede establecer umbrales de calidad en el pipeline de CI/CD que impidan que el código con vulnerabilidades conocidas sea promovido a entornos posteriores. Esto garantiza que solo el código seguro llegue a producción.



Orquestación de Herramientas de Seguridad



- Consolidación de Resultados: ASPM actúa como un centro de intercambio de información, recopilando resultados de diversas herramientas de seguridad y presentándolos en un panel unificado. Esto proporciona una visión integral de la postura de seguridad de la aplicación.
- Correlación de Vulnerabilidades: ASPM puede correlacionar vulnerabilidades de diferentes herramientas para identificar patrones y prioridades de remediación. Esto optimiza los esfuerzos de seguridad y reduce la sobrecarga de alertas.
- Automatización de Flujos de Trabajo: ASPM puede automatizar flujos de trabajo de seguridad, como la activación de escaneos, la generación de informes y la notificación a los equipos pertinentes. Esto agiliza las operaciones de seguridad y libera a los equipos para que se centren en tareas de mayor valor.

Evaluación Continua de la Postura de Seguridad

- Monitoreo en Tiempo Real: ASPM puede monitorear continuamente las aplicaciones en producción para detectar nuevas vulnerabilidades, cambios en la configuración y comportamientos anómalos. Esto permite una respuesta rápida a incidentes y minimiza el impacto de las brechas de seguridad.
- Análisis de Tendencias: Al realizar un seguimiento de las métricas de seguridad a lo largo del tiempo, ASPM puede identificar tendencias y patrones que pueden indicar áreas de mejora o riesgos emergentes. Esto permite una gestión proactiva de la postura de seguridad.
- Pruebas de Regresión de Seguridad: ASPM puede automatizar pruebas de regresión para garantizar que las nuevas características o cambios en el código no introduzcan nuevas vulnerabilidades o rompan controles de seguridad existentes.

Cumplimiento de Políticas de Seguridad

- Automatización de Controles: ASPM puede automatizar la verificación del cumplimiento de políticas de seguridad y estándares de la industria. Esto reduce el esfuerzo manual y garantiza una aplicación coherente de los controles de seguridad.
- Generación de Informes: ASPM puede generar informes de cumplimiento que documenten la postura de seguridad de la aplicación y demuestren la adhesión a los requisitos reglamentarios. Esto facilita los procesos de auditoría para los equipos de GRC corporativos.
- Gestión de Riesgos: ASPM puede ayudar a identificar y evaluar riesgos de seguridad, permitiendo a las organizaciones tomar decisiones informadas sobre la priorización de la remediación y la asignación de recursos.

En resumen, la integración de ASPM en entornos CI/CD no solo mejora la seguridad de las aplicaciones, sino que también agiliza el proceso de desarrollo, fomenta la colaboración entre equipos y garantiza el cumplimiento continuo de las políticas de seguridad. Al adoptar un enfoque proactivo hacia la seguridad, las organizaciones pueden reducir significativamente el riesgo de brechas de seguridad y proteger sus activos más valiosos.

9.2. Monitoreo de riesgos en aplicaciones en tiempo real

El monitoreo de riesgos en aplicaciones en tiempo real es una estrategia fundamental para detectar, priorizar y gestionar amenazas de manera eficaz en entornos dinámicos. La creciente complejidad de los ecosistemas tecnológicos requiere un enfoque integral y automatizado para minimizar brechas de seguridad y reducir el riesgo de ataques.

Para lograrlo, es esencial contar con una visibilidad continua de los activos, detectar amenazas en tiempo real y automatizar la remediación de vulnerabilidades. Este apartado explora los principios del monitoreo de riesgos, la priorización de amenazas y la gestión eficiente de incidentes mediante la integración de **Application Security Posture Management (ASPM).**

Un aspecto clave es el descubrimiento de activos. Para protegerlos, primero es necesario identificarlos y clasificarlos, lo que incluye no solo las aplicaciones en sí, sino también sus dependencias, APIs y bibliotecas externas. Una práctica recomendada es el uso de un **Software Bill of Materials (SBOM),** que permite rastrear los componentes utilizados en el desarrollo y detectar vulnerabilidades en ellos. Además, el monitoreo de la seguridad en los **pipelines de CI/CD** es crucial para evitar que código malicioso se infiltre en las aplicaciones.

Si se ha seguido la implementación prevista en esta guía, a lo largo del **Software Development Life Cycle (SDLC),** se realizan escaneos de seguridad en el código fuente y en las configuraciones de infraestructura en la nube. El uso de herramientas de **Application Security Testing (AST),** como SAST, DAST y SCA, permiten identificar vulnerabilidades. Finalmente, con el uso de modelos como el **Exploit Prediction Scoring System (EPSS)** podemos determinar cuáles son las vulnerabilidades más probables de ser explotadas por atacantes.

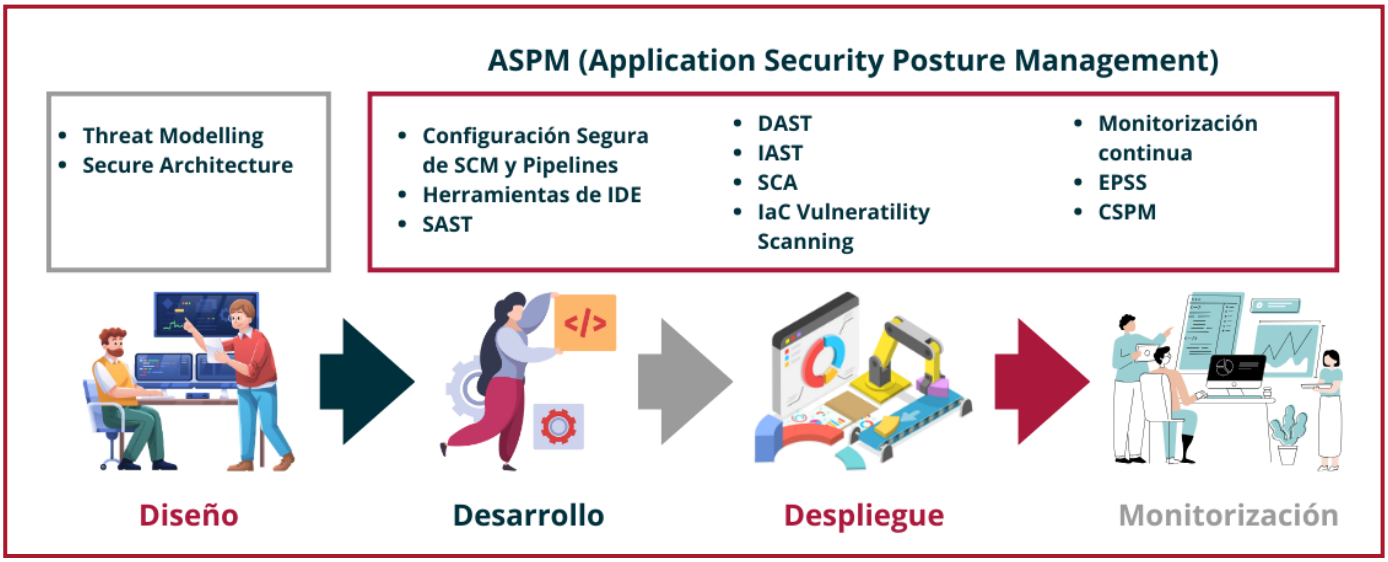


Imagen 8: ***Gestión de la postura de seguridad en aplicaciones (ASPM)***

Es recomendable aplicar respuestas automatizadas para mitigar riesgos sin intervención manual. Un ejemplo es la **rotación automática de credenciales** en caso de que se detecte una filtración de claves API en un repositorio, reduciendo la ventana de exposición.

Otra aplicación clave de la automatización es la **verificación y corrección de configuraciones erróneas en entornos cloud.** Existen herramientas dentro del ámbito de Cloud Security Posture Management (**CSPM**) dedicadas a la detección de desviaciones en las políticas de seguridad (por ejemplo, un bucket de S3 expuesto públicamente) y corregirlas automáticamente aplicando configuraciones seguras predefinidas.

Uno de los principales retos una vez implantado un sistema que da este nivel de visibilidad es la gran cantidad de alertas generadas por los sistemas de monitoreo. Para evitar que los equipos de seguridad se saturen con información irrelevante, es necesario priorizar los riesgos de manera estratégica.

Con el objetivo de facilitar el filtrado de alertas la contextualización es clave. No todas las vulnerabilidades representan la misma amenaza; algunas pueden ser explotadas fácilmente, mientras que otras pueden no tener impacto real en la seguridad. Para determinar la prioridad, se deben analizar tres factores principales: **alcance del riesgo, explotabilidad e impacto potencial.** Si una vulnerabilidad afecta un sistema crítico y es fácilmente explotable, debe ser tratada con urgencia. Se puede resumir una estrategia de gestión en el siguiente cuadro:

Alcance del Riesgo / Explotabilidad	Baja	Media	Alta
Bajo Impacto	Bajo riesgo	Moderado	Importante
Impacto Medio	Moderado	Importante	Crítico
Alto Impacto	Importante	Crítico	Urgente

- **Bajo riesgo:** puede ser monitoreado, pero no requiere atención inmediata.
- **Moderado:** se debe evaluar y programar remediación.
- **Importante:** necesita atención prioritaria en los próximos días.
- **Crítico/Urgente:** acción inmediata requerida para evitar una posible explotación.

Imagen 9: ***Estrategia de gestión de riesgos***

El análisis de alcance y explotabilidad, también conocido como **Reachability Analysis,** permite determinar si una vulnerabilidad identificada en el código es realmente ejecutable en producción. Además, los modelos de inteligencia de amenazas ayudan a identificar ataques en curso, mientras que la automatización reduce los falsos positivos y mejora la precisión de detección. Existen herramientas que cuentan con esta priorización, dependiendo de la organización pueden llegar a ser una pieza más de la security mesh.

Para garantizar la eficacia del monitoreo de riesgos en tiempo real, es necesario adoptar un enfoque estructurado. Esto implica definir métricas de seguridad claras, establecer la frecuencia de evaluación y garantizar la alineación con estándares como **OWASP ASVS.** Idealmente la frecuencia de evaluación debería ser en tiempo real, pero dependerá en gran medida de la capacidad de automatización que haya podido ser implementada.

> *En resumen, para lograr una implementación efectiva de la monitorización de riesgos basada en ASPM, es clave establecer políticas claras de gestión de vulnerabilidades, implementar herramientas de automatización, priorizar en base a modelo de explotabilidad y capacitar a los equipos de desarrollo y seguridad en la adopción de buenas prácticas.*

10. Conclusiones y Referencias.

10.1. Beneficios de un enfoque integral en DevSecOps

Tras haber completado esta guía, esperamos que los lectores hayan comprendido claramente los beneficios con los que iniciaba el ejercicio expuesto en el punto *1.4 "Beneficios de la integración de seguridad en DevOps".*

Específicamente, hemos buscado resaltar cómo un enfoque integral DevSecOps ayuda a mejorar sustancialmente la seguridad del software, la eficiencia operativa, la reducción de costes relacionados con vulnerabilidades y la gestión de incidentes, así como una aceleración significativa en el ciclo de entrega de software. Esperamos que también se reconozca la importancia crucial de la colaboración efectiva entre equipos de desarrollo, operaciones y seguridad para alcanzar objetivos de negocio y tecnológicos de manera segura y eficiente.

A lo largo de esta guía se han abordado diversas fases críticas dentro del ciclo de vida del desarrollo de software desde la perspectiva de DevSecOps. Desde la importancia del cambio cultural inicial hasta la implementación avanzada de automatizaciones y políticas de seguridad, cada etapa resalta la necesidad imperiosa de integrar la seguridad en todos los niveles del proceso.

La gestión adecuada de los distintos entornos (desarrollo, testing, pre-producción y producción) mediante prácticas seguras y robustas, junto con la automatización y el control riguroso de secretos y vulnerabilidades, constituye el núcleo central de un pipeline DevSecOps maduro y confiable.

La Infraestructura como Código (IaC) se presenta como una piedra angular del enfoque DevSecOps, asegurando infraestructuras consistentes, reproducibles y seguras, esenciales para mantener agilidad y seguridad simultáneamente. Por último, la automatización efectiva y la constante monitorización de políticas de seguridad garantizan la proactividad en la gestión de riesgos y cumplimiento normativo.

> *Implementar DevSecOps no solo fortalece la seguridad informática, sino que transforma la seguridad en un componente esencial del valor comercial y tecnológico de la organización. Este enfoque permite mantener un ritmo constante y seguro de innovación tecnológica, potenciando la confianza de clientes, usuarios y stakeholders en general. Generando a la vez, un impacto positivo en la reputación corporativa de las organizaciones que adoptan y promueven DevSecOps.*

10.2. Frameworks relevantes

A continuación, presentamos varios marcos de referencia relevantes en el ámbito de la seguridad del software, el aseguramiento de la calidad y la gestión de la madurez de estas disciplinas en las organizaciones.

OWASP Software Assurance Maturity Model (SAMM): está llamado a convertirse en el modelo de madurez de referencia de aseguramiento de la calidad del software, proporcionando una forma efectiva y medible para analizar y mejorar la postura de seguridad del software de todo tipo de organización. OWASP SAMM da soporte a todo al ciclo de vida completo (SDLC), incluyendo desarrollo y adquisición, y es agnóstico a la tecnología y el proceso concreto. Se ha creado intencionalmente para ser evolutivo y "risk-driven" de forma natural.

El modelo original (v1.0) fue creada por Pravir Chandra en 2009. Y durante más de 10 años, ha demostrado ser un modelo eficaz y ampliamente distribuido para mejorar las prácticas de software seguro en diferentes tipos de organizaciones en todo el mundo. La comunidad ha aportado traducciones y herramientas de apoyo para facilitar la adopción y la alineación. Con la versión 2.0, mejoramos aún más el modelo para abordar algunas de sus limitaciones actuales.

Visión global Modelo SAMM (v2.0):

https://owaspsamm.org/model/

DevSecOps Platform Independent Model (PIM): desarrollado por el Software Engineering Institute (SEI) de la Universidad Carnegie Mellon (USA), para permitir a las organizaciones implementar DevSecOps de una manera segura y sostenible con el fin de aprovechar plenamente los beneficios de la flexibilidad y la velocidad disponibles al implementar los principios, prácticas y herramientas de DevSecOps. Este enfoque está especialmente planteado para las organizaciones que a menudo tienen dificultades para aplicar las prácticas y principios de DevSecOps, particularmente en entornos fuertemente regulados y con restricciones de ciberseguridad, porque carecen de una base consistente para gestionar el desarrollo intensivo de software, la ciberseguridad y las operaciones en un ciclo de vida de alta velocidad. Estas organizaciones necesitan una referencia autorizada para diseñar y ejecutar completamente una estrategia DevSecOps integrada en la que se aborden todas las necesidades de las partes interesadas.

DevSecOps PIM (Version 2.1. 2022):

https://insights.sei.cmu.edu/projects/devsecops-platform-independent-model-pim/

Más información:

https://insights.sei.cmu.edu/blog/modeling-devsecops-to-protect-the-pipeline/

Publicaciones de CISA[24]: numerosos desarrollos de software se basan en la incorporación de fuentes y datos de origen abierto (open source). Aunque son numerosos los beneficios que aportan - innovación, bajo coste, etc. - también existen riesgos asociados, especialmente en el ámbito de la seguridad del software. Por estas razones, el trabajo de CISA es especialmente relevante.

24 https://www.cisa.gov/opensource

El software de código abierto (Open Source Code) se utiliza ampliamente en el gobierno federal de Estados Unidos y en todos los sectores de infraestructura crítica. CISA (la Agencia de Ciberdefensa de Estados Unidos) trabaja para comprender y reducir las ciber-amenazas al gobierno federal y la infraestructura crítica, para lo cual garantizar la seguridad del software de código abierto es un aspecto fundamental. En este sentido destaca la Hoja de Ruta de Seguridad del Software de Código Abierto de CISA, que establece su papel en la protección del software de código abierto, alineándola con su misión. A su vez, los esfuerzos, trabajos y publicaciones de CISA contribuyen a mejorar la seguridad del ecosistema de código abierto en general.

BSIMM[25] (Building Security In Maturity Model): es un framework diseñado para ayudar a las organizaciones a medir y mejorar sus iniciativas de seguridad de software. A diferencia de los modelos prescriptivos que proporcionan una lista de tareas a implementar, BSIMM es descriptivo: se construye estudiando prácticas del mundo real utilizadas por cientos de organizaciones en diversas industrias.

BSIMM se centra en comprender qué están haciendo realmente las organizaciones exitosas para proteger su software. Proporciona un conjunto completo de actividades agrupadas en dominios específicos que sirven como punto de referencia. Al comparar sus propias prácticas de seguridad con estas observaciones, las organizaciones pueden identificar brechas, priorizar mejoras y alinear sus esfuerzos con prácticas probadas de la industria.

ENISA – Ciberseguridad e Inteligencia Artificial: ENISA (La Agencia Europea para la Ciberseguridad) publica numerosos estudios y documentos que ayudan a generar cultura de seguridad (incluidas buenas prácticas de seguridad del software). Por ejemplo, el estudio "Artificial Intelligence and Cybersecurity Research" identifica las necesidades de investigación sobre IA para la ciberseguridad y la protección de la IA, como parte del trabajo de ENISA para cumplir con su mandato en virtud del Artículo 11 de la Ley de Ciberseguridad. Este informe aporta un marco de referencia útil para considerar aspectos relevantes de ciberseguridad en el desarrollo de software, incluyendo valiosos casos de estudio en Telecomunicaciones de última generación, Internet of Things (IOT) e Internet of Everything (IOE), Ciberseguridad en Cyber-Physical Systems (CPS) y en Ciber-Bioseguridad.

25 https://codific.com/bsimm-building-security-in-maturity-model-a-complete-guide/

10.3. Enlaces a recursos

- *6 Mile Security. (2024). DevSecOps playbook. GitHub.*
  https://github.com/6mile/DevSecOps-Playbook
- *Atlassian. (s.f.). Gitflow workflow.*
  https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
- *Atlassian. (s.f.). Tutoriales de Git y gestión del código fuente.*
  https://www.atlassian.com/es/git/tutorials/source-code-management
- *Bitbucket. (s.f.). Repositorio Bitbucket.*
  https://bitbucket.org/
- *Chick, T., Pavetti, S., & Shevchenko, N. (2023). Using model-based systems engineering (MBSE) to assure a DevSecOps pipeline is sufficiently secure. Software Engineering Institute, Carnegie Mellon University.*
  https://insights.sei.cmu.edu/documents/6140/Using_MBSE_to_Assure_DevSecOps_Pipelines.pdf
- *CISA. (2023). Securing the software supply chain: Recommended practices for managing open-source software and software bill of materials.*
  https://www.cisa.gov/sites/default/files/2023-12/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN%20RECOMMENDED%20PRACTICES%20FOR%20MANAGING%20OPEN_SOURCE%20SOFTWARE%20AND%20SOFTWARE_BILL_OF_MATERIALS.pdf
- *CISA. (s.f.). Open source security guidance.*
  https://www.cisa.gov/opensource
- *CISA. (s.f.). Secure by design resources.*
  https://www.cisa.gov/resources-tools/resources/secure-by-design
- *Codific. (s.f.). BSIMM: Building security in maturity model - A complete guide.*
  https://codific.com/bsimm-building-security-in-maturity-model-a-complete-guide/
- *DevOps.com. (2017). DevOps security champion: Who, what and why?*
  https://devops.com/devops-security-champion-who-what-and-why/
- *DevSecOps PIM. (2022). Platform independent model (PIM) (Versión 2.1).*
  https://insights.sei.cmu.edu/projects/devsecops-platform-independent-model-pim/
- *Gennari, J., Lau, S. H., Perl, S., Parish, J., & Sastry, G. (2024). Considerations for evaluating large language models for cybersecurity tasks.*
  https://insights.sei.cmu.edu/documents/5848/Considerations_for_Evaluating_Large_Language_Models_for_Cybersecurity_Tasks.pdf
- *GitHub. (s.f.). Gitignore templates.*
  https://github.com/github/gitignore

- *GitHub. (s.f.). Repositorio GitHub.*
  https://github.com/
- *GitHub. (s.f.). Roles de repositorio en GitHub.*
  https://docs.github.com/es/organizations/managing-user-access-to-your-organizations-repositories/managing-repository-roles/repository-roles-for-an-organization
  (Accedido el 2 de febrero de 2025)
- *GitLab. (s.f.). Acerca de GitLab.*
  https://about.gitlab.com/es/
- *Git-SCM. (s.f.). Git documentation.*
  https://git-scm.com/doc
- *Guía de iniciación en la seguridad aplicada al DevOps. (2023). ISMS Forum.*
  https://www.ismsforum.es/ficheros/descargas/guia-devsecops-v41690197585.pdf
- *IBM. (s.f.). Source code management.*
  https://www.ibm.com/docs/en/z-devops-guide?topic=applications-source-code-management
  (Accedido el 3 de febrero de 2025)
- *MITRE. (s.f.). Common attack pattern enumeration and classification (CAPEC).*
  https://capec.mitre.org/
- *National Institute of Standards and Technology. (2024). Security strategies for open source software components (NIST SP 800-204D).*
  https://csrc.nist.gov/pubs/sp/800/204/d/final
- *NIST. (s.f.). Cybersecurity framework.*
  https://www.nist.gov/cyberframework
- *Ntalampiran, S., Misuraca, G., & Rossel, P. (2023). Artificial intelligence and cybersecurity research (ENISA Research and Innovation Brief). European Union Agency for Cybersecurity (ENISA).*
  https://www.enisa.europa.eu/publications/artificial-intelligence-and-cybersecurity-research
- *OWASP Foundation. (s.f.). Application security verification standard (ASVS).*
  https://owasp.org/www-project-application-security-verification-standard/
- *OWASP Foundation. (s.f.). Developer guide - Release.*
  https://owasp.org/www-project-developer-guide/release/
- *OWASP Foundation. (s.f.). OWASP SAMM - Software assurance maturity model.*
  https://owaspsamm.org/model/
- *OWASP Foundation. (s.f.). Security champions.*
  https://owasp.org/www-project-security-culture/v10/4-Security_Champions/
- *OWASP Foundation. (s.f.). Top 10 risks for open source software.*
  https://owasp.org/www-project-open-source-software-top-10/

- *OWASP Foundation. (s.f.). Why add security in development teams.*
  https://owasp.org/www-project-security-culture/v11/2-Why_Add_Security_In_Development_Teams/
- *Red Hat. (s.f.). Security by design: Principles and threat modeling.*
  https://www.redhat.com/en/blog/security-design-security-principles-and-threat-modeling
- *SEI - Software Engineering Institute. (s.f.). Modeling DevSecOps to protect the pipeline.*
  https://insights.sei.cmu.edu/blog/modeling-devsecops-to-protect-the-pipeline/
- *Splunk. (s.f.). Source code management.*
  https://www.splunk.com/en_us/blog/learn/source-code-management.html
  (Accedido el 30 de enero de 2025)
- *TensorFlow. (s.f.). Repositorio TensorFlow.*
  https://github.com/tensorflow/tensorflow
- *The Threat Modeling Manifesto. (s.f.). Principles for threat modeling.*
  https://www.threatmodelingmanifesto.org/
- *Vergara Cobos, E. (2024). Cybersecurity economics for emerging markets. World Bank.*
  https://openknowledge.worldbank.org/entities/publication/4ec1bf22-3658-4d69-b9d3-43122254bc66?cid=ECR_LI_worldbank_EN_EXT

Si estás interesado/a en colaborar con nosotros o necesitas más información sobre nuestros proyectos, escríbenos a: proyectos@ismsforum.es

Teléfono
(+34) 915 63 50 62

Email
info@ismsforum.es

Web
www.ismsforum.es

RRSS

CSC
CYBER SECURITY CENTRE

isms forum