Universidad de Buenos Aires
Facultades de Ciencias Económicas,
Cs. Exactas e Ingeniería

Maestría en Seguridad Informática

Tesis de Maestría

Seguridad en el ciclo de vida del desarrollo de software

Mejores prácticas de Seguridad con DevSecOps

Autor: Agustín Roque Cao
Director de la Tesis: Khalil Alejandro Moriello

Noviembre 2024 - Cohorte 2016

Declaración Jurada de origen de los contenidos

Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Tesis vigente y que se hace responsable que la totalidad de los contenidos del presente documento son originales y de su creación exclusiva, o bien pertenecen a terceros u otras fuentes, que han sido adecuadamente referenciados y cuya inclusión no infringe la legislación Nacional e Internacional de Propiedad Intelectual.

FIRMADO

Agustín Roque Cao

DNI 32.737.125

Resumen

Esta tesis explora la integración de la seguridad en DevSecOps, una metodología ampliamente adoptada en el mercado debido a su capacidad para combinar prácticas de desarrollo, seguridad y operaciones en un ciclo ágil, adaptado a las demandas actuales donde la rapidez de respuesta es crucial.

Se incluye una revisión exhaustiva de fuentes y bibliografía relevante, estructurada para proporcionar al lector un aprendizaje teórico progresivo. Posteriormente, se analiza la aplicación práctica de esta metodología en un caso real, detallando el contexto, el proceso de implementación, los resultados obtenidos, las lecciones aprendidas y los pasos a seguir.

La tesis funciona como una guía teórico-práctica, ofreciendo recomendaciones y herramientas clave para implementar la seguridad en DevSecOps de manera adecuada y proporcionando un marco para la implementación de prácticas de seguridad en un caso de estudio real. Asimismo, subraya la importancia de adoptar esta metodología de seguridad, destacando los beneficios obtenidos en comparación con enfoques tradicionales.

Palabras clave: DevSecOps, Integración continua, Despliegue continuo, Automatización, Ciclo de vida de desarrollo seguro, Gestión de vulnerabilidades, Pruebas de seguridad, Cultura colaborativa.

Índice de contenidos

Introducción

Actualmente, la provisión de plataformas y servicios digitales es un pilar fundamental en el ideal de una organización exitosa. Esto trae como consecuencia un entorno digital que es cada vez más vulnerable a ataques sofisticados, lo que requiere de medidas de seguridad integrales y proactivas para tratar el constante crecimiento de las ciber amenazas.

Por otra parte, la metodología DevSecOps es ampliamente adoptada en el mercado, ya que ofrece la posibilidad de integrar prácticas de desarrollo, seguridad y operaciones en un ciclo ágil que dé respuesta a las exigencias actuales donde la demanda es creciente y la velocidad de respuesta es un factor determinante.

En este contexto, la seguridad se integra como una parte fundamental del ciclo de vida de desarrollo del software. El objetivo es: lograr que las aplicaciones sean seguras por diseño y que las vulnerabilidades sean detectadas y remediadas de manera temprana y continua.

La integración de la seguridad en el desarrollo y las operaciones a través de DevSecOps enfrenta una serie desafíos que deben ser resueltos, destacando: vulnerabilidades en el código fuente, dependencias y librerías inseguras, configuraciones de ambiente inadecuadas, procesos de seguridad manuales, gestión de vulnerabilidades, monitoreo continuo y respuesta rápida a ciber incidentes. Todo esto apoyado en una cultura de trabajo colaborativa y de responsabilidades compartidas por los distintos equipos que participan del proceso.

La integración de la seguridad en DevSecOps no solo busca resolver estos problemas, sino que también mejora la eficiencia del desarrollo y la calidad del software ofreciendo un enfoque innovador en la materia.

La temática seleccionada es de gran relevancia tanto en el ámbito académico como profesional, además de ser una competencia muy demandada en el mercado actual. En este sentido, esta tesis tiene como objetivo servir como

guía práctica, ofreciendo recomendaciones y herramientas clave para abordar el desafío de implementar la seguridad en DevSecOps de manera adecuada.

Además, este trabajo propondrá un marco que facilite la adopción y la implementación de prácticas de seguridad en DevSecOps, tomando como base un caso real de transformación digital en una organización de envergadura, en el cual se ha aplicado esta metodología.

Para ello, se buscará identificar y evaluar prácticas de seguridad que se integren de manera eficiente en los flujos de trabajo DevOps, desarrollar recomendaciones que ayuden a superar las barreras comunes en la implementación de DevSecOps (desde una perspectiva tanto técnica como cultural), e identificar herramientas y casos de uso prácticos asociados a cada una de las prácticas recomendadas.

La tesis se estructurará en siete capítulos que abordarán los conceptos más relevantes en la materia. Los primeros cuatro capítulos presentarán una revisión exhaustiva de fuentes y bibliografía relevante, estructurada para ofrecer al lector un aprendizaje progresivo. Este recorrido abarcará desde una introducción a la metodología DevOps y el desarrollo de la seguridad en los últimos años, hasta la transformación digital y los principios, actividades, y herramientas de seguridad aplicables en los ciclos de vida de DevSecOps.

Los últimos tres capítulos estarán dedicados a la aplicación de esta metodología en el caso real seleccionado. En esta sección, se proporcionarán detalles sobre el contexto actual del proyecto, la organización, la tecnología y los equipos involucrados, seguido de una descripción del proceso de implementación de la metodología. Finalmente, se analizarán los resultados obtenidos, las lecciones aprendidas y los próximos pasos para continuar con la evolución y mejora continua.

Este trabajo aspira a demostrar que integrar la seguridad en DevSecOps no solo protege aplicaciones y datos, sino que también mejora la eficiencia del desarrollo y reduce los costos de remediación de vulnerabilidades en etapas posteriores. Asimismo, se pretende enfatizar que lograr estos objetivos requiere un cambio cultural hacia la colaboración y la responsabilidad compartida, una

definición clara de prácticas de seguridad recomendadas con sus herramientas asociadas, y una capacitación profesional adecuada para el equipo.

1. DevSECOps: Introducción, seguridad y actualidad

1.1. La metodología DevOps

1.1.1. Introducción

DevOps es una metodología que surge de la necesidad de alinear las funciones de los equipos responsables del desarrollo de software (Dev) y los de operaciones de TI (Ops). Esta unión busca eliminar los silos tradicionales entre estos dos departamentos, promoviendo una cultura de colaboración y comunicación continua. Los ideales que persigue DevOps son: acelerar el ciclo de vida del desarrollo de software, mejorar la calidad del producto entregado y aumentar la frecuencia de entregas, para esto se apoya mayormente en la automatización y la integración continua.

Las personas, los procesos y la tecnología de DevOps mejoran la forma en que los ingenieros construyen, implementan y administran los sistemas, al cerrar la brecha entre los equipos de desarrollo y operaciones, para llevar los productos al mercado rápidamente, al mismo tiempo que abordan los requisitos no funcionales, como la estabilidad y la escalabilidad. DevOps es un conjunto de principios de apoyo y colaboración para brindar valor a los clientes. Si bien muchas personas piensan en DevOps como una tecnología o un conjunto de tecnologías, estas son en realidad un medio para un fin. Es decir, son simplemente herramientas que se utilizan para aplicar mejor los principios de DevOps [1].

Si se entiende que DevOps es, en definitiva, una cultura o un conjunto de principios que se centran en la colaboración, entonces se centra en la necesidad mejorar la interacción o colaboración entre el desarrollo, las operaciones y el control de calidad.

1.1.2. Agile vs DevOps

Agile y DevOps son dos metodologías que comparten varios puntos en común, pero se destaca la prioridad sobre el principio de colaboración. Agile se centra en los procesos de desarrollo y control de calidad buscando incrementar el valor al cliente.

Como los objetivos de Agile y DevOps se encuentran alineados, no es sorprendente encontrar una superposición significativa en las prácticas que los rodean. En muchos sentidos, la intersección de DevOps y Agile se relaciona con una cultura de colaboración, prácticas y procesos técnicos modernos que surgen de esa cultura. Procesos como las pruebas continuas y la implementación en lotes pequeños ayudan a garantizar la entrega rápida de productos funcionales al cliente. DevOps toma los conceptos de Agile y los extiende más allá de la compilación para que pueda amplificar Agile mediante la implementación de prácticas de DevOps [2].

Agile y DevOps, aunque tienen enfoques y orígenes diferentes, comparten una filosofía común centrada en la entrega rápida de valor, la colaboración entre equipos, la automatización y la adaptabilidad. En un resumen muy simplificado, se podría decir que DevOps extiende Agile incorporando los principios de entrega rápida y continua, y, con este fin, al equipo de operaciones de TI (Ops). Cabe destacar que esto no es del todo justo ya que Agile considera y enumera principios y prácticas recomendadas para el equipo de operaciones, e inclusive el de seguridad.

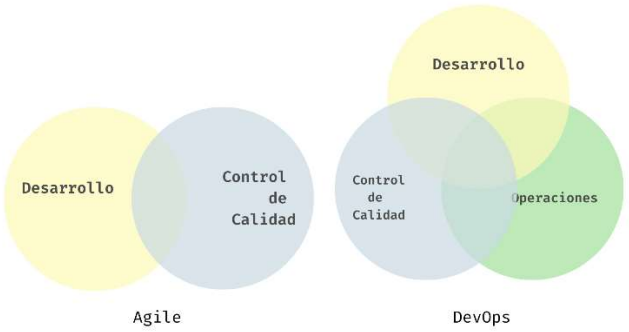

Imagen 1 - Agile vs DevOps - Fuente: Elaboración propia, [4].

1.1.3. El modelo “Three Ways”

El modelo de las tres formas (*three ways*) es probablemente el más conocido en lo que a materia de DevOps refiere. Fue propuesto por *Gene Kim*, *Kevin Behr* y *George Spafford* en *“The Phoenix Project”* [5]. Busca identificar los principios fundamentales para guiar la adopción de la metodología en las organizaciones proporcionando un marco para su implementación. [3,4,5]

1.1.3.1. Flujo

La primera forma se centra en el flujo de trabajo. Tiene por objetivo romper con los silos de desarrollo y operaciones para permitir un flujo completo que abarque desde la idea hasta la creación final de valor agregado para el cliente, o, dicho de otra forma, desde el desarrollo hasta la puesta en producción.

La meta es maximizar la eficiencia y minimizar las interrupciones o bloqueos hasta en el proceso de entrega. Esto se logra mediante la implementación de prácticas de desarrollo ágil, y la automatización del flujo de trabajo mediante la integración continua (Continuous Integration o CI) y la entrega continua (Continuous Delivery o CD).

A continuación, se citan algunos ejemplos [3,4]:

- Metas compartidas: logrando que los equipos tengan metas compartidas, se puede pasar de: desarrollar nueva funcionalidad (equipo de desarrollo) y tener ambientes estables (equipo de operaciones) a: que ambos equipos hagan énfasis en entregar mayor valor al cliente
- Automatización las pruebas: es un elemento clave para esta forma ya que reduce la intervención manual y por lo tanto los cuellos de botella en el flujo proveyéndolo de agilidad y continuidad.

1.1.3.2. Retroalimentación

La segunda forma se basa en la creación de bucles de retroalimentación que permitan a los equipos obtener información sobre la calidad y el rendimiento de las aplicaciones de forma temprana y continua. La retroalimentación rápida es

clave para detectar problemas, ajustarse a los cambios y aplicar la mejora continua.

A continuación, se citan algunos ejemplos [3,4]:

- Indicadores de funcionalidades: a través de flags se pueden activar o desactivar apagar funcionalidades específicas del producto. Esto permite enviarlas a los ambientes productivos sin ponerlas a disposición de los clientes y luego activarlas (en forma segmentada o no) sin cambios adicionales en el código. Esta funcionalidad también se utiliza para test de hipótesis.
- Monitoreo en tiempo real: que permita a los desarrolladores obtener información sobre cómo se comporta su código en producción.

1.1.3.3. Aprendizaje continuo y experimentación

La tercera forma promueve una cultura de aprendizaje constante y experimentación. Esto incluye un cambio de enfoque en cuanto a tratar los errores como fallos, viéndolos como oportunidades de aprendizaje y mejora. Esta forma aboga por pequeñas iteraciones y pruebas controladas para validar nuevas ideas y tecnologías.

A continuación, se citan algunos ejemplos [3,4]:

- 20% del tiempo: es una práctica que consiste en reservar el 20% del tiempo para trabajar únicamente en el desarrollo de nuevas ideas y su experimentación, por ejemplo: un día a la semana o una semana al mes.

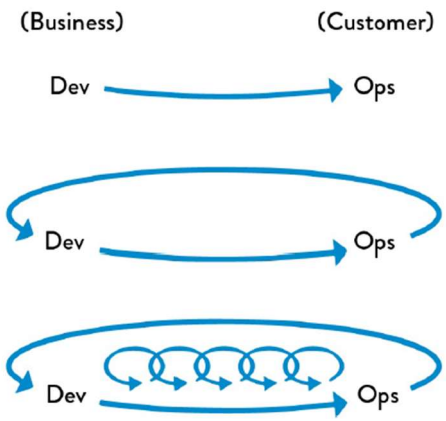

Imagen 2 - Las tres formas - Fuente: [6].

1.1.4. El modelo “CALMS”

El modelo CALMS, que refiere al acrónimo de Cultura (*Culture*), Automatización *(Automation)*, *Lean*, Medición *(Measurement)* y Compartir *(Sharing)* fue propuesto por John Willis y Damon Edwards [7] y es otra aproximación de los principios base de DevOps.

En detalle, los términos adoptados refieren a:

- Cultura: se centra en la colaboración y en cómo las personas trabajan en conjunto por una meta en común.
- Automatización: es uno de los pilares de DevOps. Básicamente la idea es automatizar todo lo que se pueda utilizando métodos como la entrega continua o la infraestructura como código para garantizar la entrega de valor al cliente. La automatización es crítica ya que a mayor cantidad de pasos que requieran intervención manual, mayores cuellos de botella pueden producirse en el flujo [4].
- Lean: si bien no existe una traducción exacta al idioma español, el enfoque Lean refiere a quitar lo que no aporta valor al cliente, por ejemplo: código y funcionalidades innecesarias, requisitos poco claros, burocracia, entre otros.
- Medición: refiere a extraer información de importancia, medir y analizar el rendimiento y la eficiencia en pos de aplicar la mejora continua.

- Compartir: refiere a una comunicación abierta, transparente y colaborativa en todas las etapas y equipos del proyecto.

1.1.5. Otros modelos

El modelo de los cinco ideales fue propuesto por Gene Kim [8] y complementa el de las tres formas centrándose en principios que mejoran la productividad y el nivel de satisfacción de los desarrolladores.

- Localidad y simplicidad: Facilitar que los desarrolladores trabajen de manera independiente en las distintas partes del software.
- Flujo y aumento de la retroalimentación: Optimizar el flujo de trabajo y asegurar la retroalimentación continua.
- Mejora continua y experimentación: Fomentar la innovación y la mejora constante a través de la experimentación.
- Seguridad psicológica: Crear un entorno en el que los empleados se sientan seguros para asumir riesgos y cometer errores sin temor a las consecuencias.
- Focalización en el cliente: Mantener al cliente como el centro de todas las actividades, asegurando que el trabajo realizado le brinde valor.

Existen otros modelos que ofrecen guías conceptuales y prácticas recomendadas para la implementación de DevOps, como el ciclo de vida de DevOps (Atlassian) [9], el modelo CAMS (John Willis) [7], el modelo CAMSSA (ThoughtWorks), entre otros. Cada uno aporta perspectivas valiosas sobre cómo mejorar la colaboración, la automatización, y la entrega continua de software, adaptándose a las necesidades específicas de cada organización.

1.2. La evolución hasta DevSecOps

La seguridad informática ha evolucionado significativamente a lo largo de los años, adaptándose a las nuevas amenazas y tecnologías y acompañando los cambios de metodologías de trabajo y de los ecosistemas organizacionales.

A continuación, se busca describir brevemente esta evolución y relacionar cada enfoque con su aplicabilidad en el campo de DevSecOps.

1.2.1. Seguridad perimetral

La seguridad perimetral se centra en proteger los activos de información de la red interna de una organización mediante la implementación de perímetros alrededor de ella. Para hacerlo se apoya en la utilización de tecnologías como *firewalls*, sistemas de detección de intrusos (IDS) y sistemas de prevención de intrusos (IPS), entre otros. Este enfoque asume que las ciberamenazas provienen del exterior y, por lo tanto, que el interior de la red es seguro.

La seguridad perimetral, si bien se sigue utilizando en la actualidad, con la evolución de las tecnologías y los ecosistemas organizacionales híbridos, sumado a la incorporación de dispositivos móviles y la proliferación de aplicaciones e infraestructura en la nube, hace insuficiente este tipo de protección.

Su aplicabilidad en metodologías DevSecOps es prácticamente nula ya que está basada en la protección de ambientes productivos (al final del ciclo de vida de desarrollo del *software*) y no pregona los principios de cultura de colaboración y responsabilidad compartida de DevSecOps siendo que el responsable de la protección de los activos de información es casi exclusivamente el equipo de ciberseguridad.

1.2.2. Defensa en profundidad

Como respuesta a las limitaciones del modelo anterior, surge el concepto de “defensa en profundidad”. Este enfoque implementa múltiples capas de seguridad a lo largo de la infraestructura, buscando que, si un atacante logra vulnerar una capa, tenga otras barreras de seguridad por delante. Un caso de ejemplo puede ser compuesto por las capas: perímetro, red, aplicación, datos, cada una con su protección de seguridad.

Algunas de las medidas en las que se sustenta este modelo incluyen: la segmentación de redes, la implementación de mecanismos de autenticación y

autorización fuertes, el cifrado de datos en cada capa y políticas de acceso controlado, entre otros.

Nuevamente, su aplicabilidad en metodologías DevSecOps tiene limitaciones similares al enfoque anterior, ya que, conserva el concepto del trabajo en silos en detrimento de una cultura colaborativa y de responsabilidad compartida. Por otra parte, no integra la seguridad desde una etapa temprana del ciclo de vida.

1.2.3. Seguridad Zero Trust

De la misma forma que el enfoque de defensa en profundidad surge el concepto de seguridad *Zero Trust* centrado en la creencia de que las organizaciones no deben confiar automáticamente en nada ni nadie dentro o fuera de sus perímetros y, en cambio, deben verificar todo lo que intente conectarse a sus sistemas antes de otorgar acceso [10].

Zero Trust puede basarse en estrategias de defensa en profundidad, pero ambas no son lo mismo. Mientras que la defensa en profundidad se centra en brindar seguridad en cada capa, *Zero Trust* se centra en el supuesto de que nunca se debe dar por sentado que se debe confiar [4]. Por lo tanto, es muy posible que una arquitectura tecnológica tenga muchas capas de defensa pero que, aun así, utilice un único método de autenticación para demostrar la identidad (por ejemplo, algo que se conoce como una contraseña) que luego se acepta en todas las capas y, por lo tanto, no se adhiera a los principios básicos de *Zero Trust*.

Su aplicabilidad en el campo de DevSecOps tiene limitaciones similares al enfoque anterior: trabajo en silos y en etapas tardías del ciclo de vida.

1.2.4. Desplazar la seguridad: Shift Left

Shift Left es un concepto que consiste, básicamente, en realizar las tareas “antes” en el ciclo de vida de desarrollo del software. Si bien originalmente el enfoque fue concebido para realizar pruebas o controles de calidad, es fácilmente extensible al ámbito de la ciberseguridad.

Los beneficios del enfoque están a simple vista, cuanto antes se realicen los controles, antes se detectan los desvíos y antes pueden solucionarse, lo cual implica beneficios en tiempo y costos asociados.

La siguiente imagen busca graficar los costos asociados a la resolución de desvíos en cada etapa del ciclo de vida de desarrollo del software. Para analizar la misma se debe tener en cuenta que los valores absolutos se deben tomar únicamente como guía ya que los mismos pueden modificarse de acuerdo con la bibliografía de consulta, pero, de todas formas, sirven para sustentar los beneficios de adoptar el enfoque de *Shift Left*.

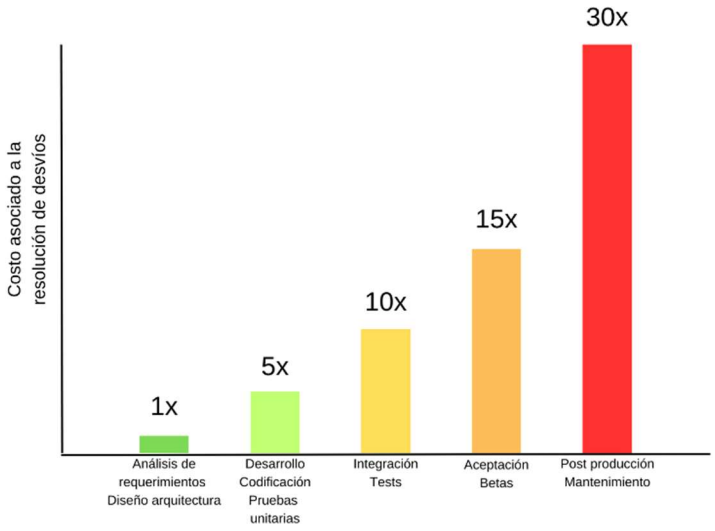

Imagen 3 - Costo por etapa SDLC - Fuente: Elaboración propia.

Al contrario de los enfoques anteriormente descritos, *Shift Left* es un concepto clave en la aplicación de los principios de DevSecOps: la seguridad es responsabilidad de todos. Aplicar medidas de seguridad en etapas tempranas, permite a los equipos de desarrollo involucrarse con las prácticas de seguridad y sentirse parte clave en el proceso, lo cual deriva en un cambio cultural hacia la responsabilidad compartida.

Si se analiza bajo el modelo de las tres formas [5], presentado algunas secciones atrás en este trabajo, las prácticas de *Shift Left* son claves en la aplicación de las dos primeras formas: flujo y retroalimentación.

1.3. Introducción a DevSecOps

1.3.1. Seguridad en DevOps

Tomando una visión simplista, DevSecOps es nada más y nada menos que DevOps + ciberseguridad:

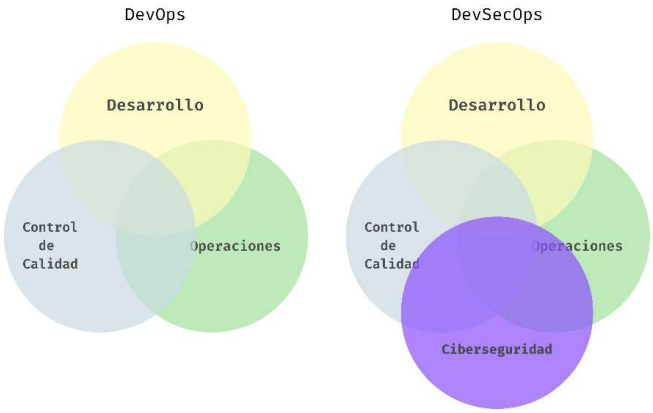

Imagen 4 - DevOps y DevSecOps - Fuente: Elaboración propia, [4].

La práctica de DevSecOps consiste en integrar la seguridad en un flujo de entrega e implementación continuas. Al aplicar seguridad a los principios de DevOps, la verificación de la seguridad se convierte en una parte activa e integrada del proceso de desarrollo.

Al igual que DevOps, DevSecOps es una metodología técnica y organizativa que combina flujos de trabajo de gestión de proyectos con herramientas y tecnologías automatizadas. Para ello, integra auditorías de seguridad activas y pruebas de seguridad en flujos de trabajo de DevOps y de desarrollo ágiles, a fin de integrar la seguridad en el producto, en lugar de aplicarla a un producto terminado [9].

Por último, y antes de pasar al detalle más fino sobre los desafíos, metas, adopción en el mercado y beneficios de DevSecOps, es importante notar que la primera afirmación de esta sección: DevOps + ciberseguridad = DevSecOps, no es del todo cierta. Mientras que la primera afirmación toma como partes separadas a DevOps y a ciberseguridad implementando flujos de desarrollo y operación continuos, pero aplicando medidas de seguridad únicamente en

ambientes productivos (seguridad perimetral, en capas, entre otros); el término DevSecOps integra la seguridad durante todo el proceso y las prácticas de DevOps (desplazando la seguridad a la izquierda). Visto de otra forma, DevSecOps es la intersección de los conjuntos de Desarrollo, Operaciones, Calidad y Ciberseguridad de la “Imagen 4”.

DevSecOps permite que la seguridad sea ágil, adaptativa y coherente con los ritmos rápidos del desarrollo moderno, sin comprometer la velocidad ni la eficiencia.

1.3.2. Desafíos y metas de DevSecOps

A continuación, se listan algunos de los principales desafíos que se deben abordar con DevSecOps [11]:

- La adopción de DevOps está aumentando como una alternativa a las metodologías tradicionales de desarrollo ágil y en cascada, pero la seguridad y el cumplimiento normativo suelen quedar en segundo plano.
- Las prácticas de DevOps fomentan la automatización para lograr escala, pero la seguridad ha sido tradicionalmente manual, basada en procesos y controlada por puertas: la antítesis de la automatización, la transparencia y la velocidad.
- La mayoría de los desarrolladores no tienen conocimientos de codificación segura, incluidos aquellos con conocimientos en metodologías ágiles y DevOps.
- Los enfoques tradicionales de pruebas de seguridad de aplicaciones no fueron diseñados para la velocidad y la transparencia.
- Para algunas aplicaciones en industrias específicas, las nuevas versiones deben volver a ser certificadas después de cada actualización de producción, lo que hace que el cambio rápido sea un problema.

En consecuencia, se mencionan algunas de las metas de DevSecOps [9]:

- Introducir mecanismos de seguridad a lo largo de todo el ciclo de vida del desarrollo del software.
- Garantizar que los equipos de desarrolladores y de operaciones compartan la responsabilidad de seguir las prácticas recomendadas de seguridad. Es más, es vital que se sientan parte esencial del proceso de securización.
- Posibilitar los chequeos y pruebas de seguridad automatizadas en cada etapa de la entrega de software mediante la integración de controles, herramientas y procesos de seguridad.

1.3.3. Componentes de DevSecOps

Para implementar correctamente la práctica de DevSecOps, es necesario tener en cuenta los siguientes componentes [12]:

- Análisis de código: inspeccionar el código fuente de una aplicación en busca de vulnerabilidades, corrigiendo y garantizando que se ajuste a las mejores prácticas de seguridad.
- Administración de cambios: utilizar herramientas de administración de cambios para realizar un seguimiento de los cambios y requisitos asociados. De este modo, se evitan vulnerabilidades de seguridad involuntarias ocasionadas por cambios imprevistos o no registrados.
- Administración del cumplimiento: garantizar que las aplicaciones cumplan con los requisitos regulatorios en materia de seguridad.
- Modelado de amenazas: investigar los problemas de seguridad que puedan surgir antes y después de desplegar la aplicación.
- Capacitación en materia de seguridad: capacitar a los desarrolladores y a los equipos de operaciones. De este modo, pueden tomar decisiones de seguridad independientes a la hora de crear y desplegar la aplicación.

1.3.4. Beneficios y adopción en el mercado

DevSecOps se utiliza en las organizaciones modernas debido a una serie de beneficios que abordan directamente los desafíos contemporáneos. A continuación, se explican algunas de las principales razones por las que las organizaciones adoptan ampliamente esta metodología:

- Apoyo a la transformación digital: DevSecOps es un facilitador clave para la transformación digital, permitiendo a las organizaciones adoptar nuevas tecnologías, optimizar procesos y mejorar la experiencia del cliente de manera ágil y efectiva.
- Reducción del tiempo de puesta en mercado (Time to Market): Las organizaciones pueden llevar soluciones y características al mercado más rápidamente, lo cual es esencial en un contexto actual donde la rapidez es, en la mayoría de los casos, una ventaja competitiva.
- Mayor satisfacción del cliente: Al acortar el ciclo de entrega de nuevas funcionalidades, las organizaciones pueden responder más rápidamente a las necesidades y expectativas de los clientes, lo que se traduce en una mayor satisfacción.
- Menor tasa de errores: La integración continua y las pruebas automatizadas permiten detectar errores y problemas en una etapa temprana del ciclo de desarrollo, lo que reduce la cantidad de desvíos en producción y mejora la calidad general del software.
- Retroalimentación más ágil: Los equipos reciben retroalimentación continua sobre el rendimiento del software en entornos de prueba y producción, lo que les permite realizar ajustes y mejoras rápidamente.
- Automatización de procesos repetitivos: La automatización reduce la carga de trabajo manual, lo que libera a los equipos para centrarse en tareas de mayor valor como la innovación y la resolución de problemas complejos.

- Escalabilidad y adaptabilidad: Las prácticas de DevSecOps combinadas con tecnologías de microservicios y contenedores, permiten a las organizaciones escalar sus aplicaciones de manera flexible adaptándose a cargas de trabajo más exigentes.
- Seguridad integrada: DevSecOps, como se analizó en la sección anterior, integra prácticas de seguridad en todas las fases del ciclo de vida del desarrollo. Esto colabora en la premisa de asegurar que las aplicaciones sean seguras desde su concepción, reduciendo los riesgos de seguridad y las vulnerabilidades en producción.

En resumen, DevSecOps se utiliza en las organizaciones actuales ya que proporciona una ventaja competitiva al acelerar la entrega de software, mejorar su calidad, aumentar la eficiencia operativa y de los procesos, fomentar la colaboración, adaptarse rápidamente a las necesidades del mercado y brindar seguridad en todo el proceso. Estas capacidades son esenciales para las organizaciones que buscan mantenerse relevantes y competitivas en un entorno tecnológico en constante cambio.

2. DevSECOps: Transformación digital

2.1. La cultura

2.1.1. Los cuatro pilares

[13] Existen cuatro pilares clave que se deben tener en cuenta cuando se busca cambiar la cultura DevSecOps de una organización: Personas, Procesos, Tecnologías y Gobernanza

Los principios de DevSecOps se basan en estas cuatro partes interrelacionadas, eliminando los silos y creando un enfoque colectivo. Este entorno de responsabilidad compartida y empatía mutua requiere derribar barreras entre los equipos. En consecuencia, las personas son el punto de partida y la base de cualquier implementación de DevSecOps. La reestructuración de los equipos de DevOps y Seguridad para establecer una cooperación eficiente entre ellos, así como ofrecer una capacitación de buena calidad y específica a toda la organización, garantiza que la seguridad se convierta en un estado de ánimo en lugar de un obstáculo.

El siguiente paso es introducir procesos de apoyo, con el objetivo de mejorar aún más la colaboración entre las personas, así como lograr procesos de desarrollo más seguros en su conjunto. Estos cambios de procesos están diseñados para abarcar las tres áreas funcionales de desarrollo, seguridad y operaciones, proporcionando cohesión y uniformidad entre ellas.

Además, el enfoque DevSecOps requiere contar con las tecnologías adecuadas para permitir que los equipos ejecuten estos procesos y los automaticen. Esto, en última instancia, reduce la superficie de ataque de la organización y permite una gestión eficaz de la deuda de seguridad técnica. La tecnología y las herramientas que respaldan un flujo de trabajo DevSecOps, es a menudo el área en la que la mayoría de las organizaciones piensan primero.

Y, por último, uno de los elementos menos considerados de una verdadera cultura DevSecOps es la gobernanza. Si bien las personas, los procesos y las

tecnologías se unen para apoyarse mutuamente, la gobernanza también desempeña un papel clave. Mide el rendimiento de los demás elementos y puede señalar dónde se necesita más atención para garantizar que todas las partes de la cultura se integren.

2.1.2. El modelo de responsabilidad compartida

[4] Como DevSecOps se centra en la colaboración, debe crear un modelo de responsabilidades compartidas en el que todos sean responsables de la seguridad. La idea es que todo aquel que esté desarrollando un producto, implementando una herramienta o diseñando una arquitectura, sea también responsable de su seguridad.

En DevSecOps ya no es aceptable simplemente asumir que el equipo de seguridad se ocupará exclusivamente de la seguridad. Para que DevSecOps tenga éxito, se deben olvidar los patrones que intenten arrojar la responsabilidad a otro equipo.

2.1.2.1. La matriz RACI

[14] Una herramienta que ayuda a identificar responsabilidades es la matriz RACI. RACI es un acrónimo en inglés (*Responsible*, *Accountable*, *Consulted*, *Informed*) que ayuda a los equipos a brindar claridad con respecto a la asignación de roles en un proyecto y determinar quién es el responsable de una tarea específica.

- Responsable (Responsible): Es quien está a cargo del trabajo de forma directa y quien realmente realiza el trabajo. Solo debe haber un responsable por tarea para que todos sepan a quién acudir si hay preguntas o actualizaciones.
- Aprobador (Accountable): El aprobador es quien está a cargo de supervisar la finalización general de la tarea, aunque es posible que no sea la persona que en realidad realiza el trabajo. El aprobador suele ser un ejecutivo, líder o, incluso, el responsable del proyecto.

En cualquier caso, este rol es el de una persona que está a cargo de aprobar el trabajo antes de que se considere finalizado.

- Consultado (Consulted): Es quien o quienes deben revisar y dar el visto bueno al trabajo antes de entregarlo. Puede haber varios consultados para cada tarea, logro del proyecto o entrega. Se trata de actores con suficiente conocimiento y expertise como para emitir opiniones sobre el trabajo realizado por el responsable de proyecto.
- Informado (Informed): Es quien debe estar informado del progreso y la finalización del trabajo. Probablemente no estén involucradas en ningún otro aspecto de la entrega. De hecho, a estas personas no se les pedirá opinión sobre el trabajo realizado, solo necesitan una actualización de estado de la tarea.

Para crear e implementar una matriz RACI se deben seguir una serie de pasos:

1. Identificar los roles en el proyecto.
2. Detallar las tareas y entregables.
3. Asignar los roles.
4. Compartir la matriz de responsabilidades con todo el equipo.
5. Compartir la matriz de con el resto de los interesados.

2.2. Las personas

2.2.1. Cooperación y empatía

[13] La necesidad de entendimiento entre desarrolladores y profesionales de seguridad ha sido un tema de conversación durante décadas. A menudo es un desafío ayudar a los equipos de seguridad a ser más conscientes de los desafíos que enfrentan los desarrolladores y viceversa. Sin embargo, el valor que crea dicha comprensión en términos de implementación de mejoras exitosas no se puede subestimar. En un modelo DevSecOps, esto es de particular importancia.

La implementación de herramientas de seguridad que generan una tensión innecesaria para los desarrolladores puede ser perjudicial para el proceso. Por otra parte, los desarrolladores que introducen tecnologías que crean riesgos de seguridad cruciales pueden causar problemas igualmente devastadores. Para lograr un balance adecuado existen algunas técnicas y programas que ayudan a romper estas barreras cambiando de a poco la cultura de seguridad, por ejemplo: los campeones de seguridad y las recompensas por hallazgos de errores.

2.2.2. Programa de campeones de seguridad

[15] Un programa de campeones de seguridad (*Security Champions*) difunde la conciencia sobre las mejores prácticas al influir en el comportamiento organizacional hacia mejores hábitos para reducir el riesgo general de seguridad.

Los campeones de seguridad son voluntarios que no trabajan en el ámbito de la seguridad (por ejemplo, desarrolladores) y que reciben capacitación e incentivos adicionales para representar a la seguridad en sus equipos. Actúan como enlaces o puntos de contacto en ambas direcciones: de la seguridad hacia su equipo y viceversa.

Muchos de los métodos que se implementan inicialmente en un buen programa de campeones de seguridad utilizan recompensas extrínsecas y otros incentivos superficiales. Estos sirven para entrenar a la organización a practicar nuevos hábitos hasta que, con el tiempo, la cultura se reconfigura para seguir los principios básicos modificados.

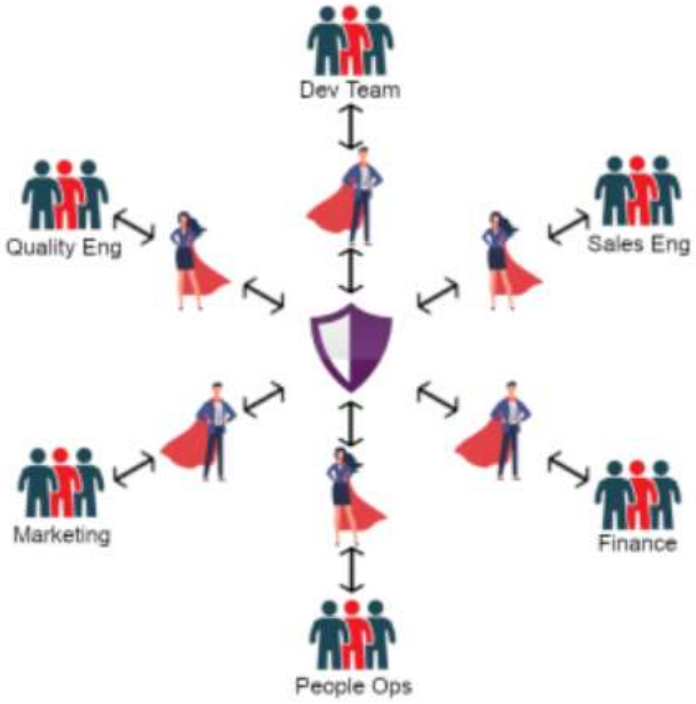

Imagen 5 - Modelo de programa de campeones de seguridad - Fuente: [15].

Algunos de los beneficios que pueden obtener los campeones son:

- Obtener conocimientos, habilidades y experiencia reales en el importante campo de la ciberseguridad
- Aprender a protegerse mejor a sí mismos, a su equipo y a su organización de los incidentes de seguridad
- Destacarse por el enfoque en la calidad e influir en la cultura de su organización y ganar reconocimiento
- Ser un aliado de la seguridad es actualmente un rol reconocido en el mercado.

2.2.3. Programa de recompensa por hallazgo de errores interno

[4, 16, 17] Los programas de recompensas por errores de seguridad internos ofrecen un premio por identificar vulnerabilidades de seguridad en aplicaciones o servicios proporcionados por la organización.

Estos programas ofrecen una recompensa a cualquiera que pueda identificar vulnerabilidades de seguridad en productos existentes. Ayudan a

generar conciencia sobre la seguridad al tiempo que ayudan a garantizar la seguridad de los productos. Los incentivos pueden presentarse en forma de reconocimiento o incluso compensación monetaria.

Estos programas ayudan a generar una cultura de seguridad en toda la organización al alentar a todos a participar en la identificación de vulnerabilidades de seguridad.

Las recompensas por errores relacionados con la seguridad no solo ayudan a identificar posibles problemas de seguridad dentro de los productos de una organización, sino que también alientan a los empleados a comprender mejor las vías que un adversario podría explotar y, por lo tanto, a adquirir más conocimientos sobre los obstáculos que se deben evitar.

2.3. Los procesos

2.3.1. Gestión de vulnerabilidades

[18] Las vulnerabilidades son similares a *back doors* que un atacante puede utilizar para entrar en los sistemas de una organización y robar datos confidenciales, interrumpir las operaciones o causar otros tipos de daños. Al gestionar las vulnerabilidades, una organización puede mitigar el riesgo de violaciones de seguridad y proteger sus datos, reputación y recursos financieros.

El proceso de gestión de vulnerabilidades es un enfoque sistemático para identificar y mitigar las vulnerabilidades en una organización. Su propósito es minimizar el riesgo de una violación de seguridad al reducir la superficie de ataque y abordar las vulnerabilidades antes de que puedan explotarse, protegiendo así los activos, los datos y la reputación de la organización. Los pasos de este proceso generalmente incluyen:

- Descubrimiento: identificar vulnerabilidades en los activos tecnológicos y de información.
- Evaluación: detectar vulnerabilidades mediante herramientas automatizadas o pruebas manuales.

- Priorización: clasificar las vulnerabilidades según su gravedad e impacto potencial.
- Remediación: implementar estrategias multifacéticas, como administración de parches, configuraciones seguras y medidas de control de acceso, para mitigar o eliminar las vulnerabilidades identificadas.
- Verificación: confirmar la resolución adecuada de las vulnerabilidades.
- Monitoreo: detectar y gestionar continuamente nuevas vulnerabilidades.

Algunos de los métodos para descubrir vulnerabilidades son:

- Análisis de vulnerabilidades: las herramientas de análisis se pueden utilizar para analizar periódicamente redes, sistemas y aplicaciones en busca de vulnerabilidades conocidas.
- Pruebas manuales: las herramientas de análisis pueden pasar por alto algunas vulnerabilidades; por lo tanto, se deben realizar pruebas adicionales.
- Auditorías de seguridad: se pueden realizar auditorías de seguridad para evaluar los sistemas de información y los controles de seguridad de una organización a fin de determinar su eficacia y cumplimiento con las normas, estándares y políticas pertinentes.
- Respuesta a incidentes: si ocurre algún incidente de seguridad, se debe investigar la causa raíz. Esto a menudo conduce al descubrimiento de vulnerabilidades previamente desconocidas.

2.3.2. Gestión de incidentes de seguridad

[19] El enfoque de DevSecOps para la gestión de incidentes no se diferencia demasiado de los pasos tradicionales (modelo ITIL) para gestionar los incidentes de forma eficaz. La gestión de incidentes de DevSecOps incluye un énfasis explícito en implicar a los equipos de desarrolladores desde el principio y

en asignar el trabajo en función de la experiencia y no de la jerarquía dentro de la organización.

El proceso de gestión de incidentes consta básicamente de cinco pasos:

1. Detección: Trabajar en forma colaborativa para planificar las respuestas a posibles incidentes identificando las deficiencias de los sistemas. Configurar herramientas de supervisión, sistemas de alerta y *playbooks/runbooks* que ayuden a todos los miembros a saber con quién ponerse en contacto al detectar un incidente y qué hacer a continuación.
2. Respuesta: Identificar miembros de guardia de todos los equipos intervinientes para que el responsable pueda derivar al equipo que corresponda teniendo rápida respuesta.
3. Resolución: Encontrar la solución con mayor rapidez al aprovechar la ventaja de que los miembros de los equipos están más familiarizados con el código o la programación.
4. Análisis: Establecer mecanismos que incluyan retrospectivas para compartir información, métricas y lecciones aprendidas con el objetivo de la mejora continua.
5. Preparación: Tomar lo aprendido en el proceso de análisis retrospectivo, actualizar los *runbooks* y efectuar todos los ajustes necesarios en las herramientas de supervisión y los sistemas de alerta para los siguientes incidentes.

[13] Responder a los incidentes de seguridad no debe ser una actividad improvisada o no planificada. Es fundamental crear flujos de trabajo, planes de acción, manuales de estrategias y manuales de procedimientos con antelación para garantizar que la respuesta a un incidente sea coherente, repetible y medible. La gestión de incidentes debe hacer uso de los metadatos para ayudar a simplificar este proceso, modificando así las métricas para destacar el tiempo que se tarda en volver a implementar un activo comprometido. A medida que se desarrollan los manuales de estrategias, los procesos introducidos deben incluir todas las formas de incidentes. Así como DevSecOps unifica las distintas

funciones, también deberían hacerlo los procesos que lo implementan. Por lo tanto, los procesos de gestión de incidentes deben tener un nivel de abstracción en el que los incidentes operativos y de seguridad se traten siguiendo el mismo marco. Este es otro paso eficaz para reforzar la cultura de responsabilidad compartida que es crucial para DevSecOps.

A su vez, una vez que se han codificado los manuales de estrategias, se pueden integrar en el flujo de trabajo de CI/CD para automatizarlos. En un mundo DevSecOps, la búsqueda proactiva y preventiva de amenazas, y la detección y respuesta continuas a amenazas y vulnerabilidades implican que haya menos incidentes importantes y más mitigaciones. El uso de pruebas de penetración, equipos de seguridad ofensiva y recompensas por errores proporciona una capa adicional de mitigación contra el riesgo de infracciones. Si bien la detección continua es algo excelente, las organizaciones deben estar atentas a la fatiga de alertas. Esto significa establecer bucles de retroalimentación que impulsen la mejora continua del monitoreo y una mayor automatización en la evaluación y respuesta a las alertas.

2.3.3. Evaluaciones de seguridad proactivas

No importa cuán avanzada sea la metodología DevSecOps de una organización, siempre es necesaria la identificación activa de vulnerabilidades.

2.3.3.1. Pruebas de penetración

[13] Las pruebas de penetración suelen ser la primera etapa de madurez en lo que respecta a la gestión de vulnerabilidades en las aplicaciones. El objetivo de estas pruebas es identificar áreas dentro de una aplicación que puedan ser vulnerables a algún tipo de ataque. El objetivo ideal de estas evaluaciones es proporcionar una visibilidad integral de la postura de seguridad general e intentar remediar las vulnerabilidades riesgosas. Sin embargo, lograr esta visión integral de las vulnerabilidades en la aplicación puede llevar mucho tiempo. Por lo tanto, dentro de DevSecOps, si bien estas pruebas son un requisito básico de la higiene

de la seguridad, generalmente se implementan como una actividad de posproducción.

2.3.3.2. Equipo de seguridad ofensiva (*Red Team*)

[20] Las pruebas del equipo rojo utilizan el hacking ético para identificar brechas de seguridad en los sistemas de una organización mediante técnicas del mundo real como las que utilizan los atacantes.

El trabajo en equipo rojo va más allá de una prueba de penetración, porque enfrenta a un equipo de adversarios (el equipo rojo) contra el equipo de seguridad de una organización (el equipo azul). El equipo rojo suele estar formado por profesionales de seguridad altamente capacitados que comprenden las tácticas del mundo real para comprometer entornos. Las organizaciones pueden utilizar la información de esta simulación para corregir las debilidades y mejorar su postura de seguridad.

Para constituir un *Red Team* eficiente, se requieren las siguientes habilidades:

- Habilidades de desarrollo de software y capacidad para desarrollar herramientas personalizadas para superar los sistemas de seguridad.
- Experiencia en pruebas de penetración y comprensión de cómo funcionan los sistemas de seguridad para evitar ser detectados.
- Habilidades de ingeniería social y comprensión de cómo persuadir a las personas para que compartan información confidencial.

Algunas de las tácticas más utilizadas son:

- Las pruebas de penetración de aplicaciones web buscan debilidades en el diseño y la configuración de las aplicaciones web. Funcionan mediante el uso de una técnica maliciosa como la falsificación de solicitud entre sitios (cross-site request forgery) para obtener un punto de acceso.

- Las pruebas de penetración de red buscan debilidades en la red o sistema. Funcionan buscando puntos de acceso, como puertos abiertos en la red inalámbrica.
- Las pruebas de penetración física buscan debilidades en los controles de seguridad física. Por ejemplo, un miembro del equipo rojo podría intentar seguir a los empleados con credenciales para acceder a áreas de acceso restringido.
- Las tácticas de ingeniería social tienen como objetivo persuadir y manipular a los recursos humanos. Funcionan intentando usar tácticas como el phishing o el soborno para obtener información confidencial, de personas con conocimiento de los sistemas o privilegios elevados dentro de los mismos.

2.3.4. Inteligencia de amenazas

[21] Inteligencia de amenazas (*Threat Intelligence*) hace mención al proceso de recopilar, analizar y aplicar información sobre ciberamenazas para prevenir, identificar y mitigar ataques. Su objetivo es ayudar a las organizaciones a estar informadas sobre posibles atacantes, sus tácticas y vulnerabilidades explotadas, mejorando así su capacidad de defensa y respuesta ante incidentes.

[13] Sin embargo, muchas organizaciones tienen dificultades para identificar sus activos tecnológicos de forma que puedan vincular eficazmente la inteligencia sobre amenazas recibida con los activos de su entorno. Al garantizar la creación de procesos para alimentar metadatos desde el canal de DevSecOps a las capacidades de inteligencia sobre amenazas, la organización puede asegurarse de que se está recopilando la inteligencia correcta y de que se está aplicando y respondiendo a ella de una manera adecuada y priorizada en función del riesgo.

[22] La inteligencia de amenazas se puede clasificar en varios niveles:

- Estratégica: Proporciona información de alto nivel sobre las tendencias de amenazas y es útil para la toma de decisiones a largo plazo. Generalmente se centra en cuestiones como situaciones

geopolíticas, tendencias de ciberamenazas en una industria en particular o cómo o por qué ciertos activos estratégicos de la organización pueden ser atacados.

- Táctica: Se centra en identificar indicadores de compromiso y otros datos técnicos que los equipos de seguridad pueden usar para detectar ataques en tiempo real. Por ejemplo, direcciones IP asociadas con servidores de comando y control, hashes de archivos relacionados con malware conocido y ataques de ransomware o líneas de correo electrónico asociadas con ataques de phishing.
- Operacional o Técnica: Detalla los métodos y tácticas que los atacantes usan actualmente y ayuda a anticipar futuros ataques. Por ejemplo, los vectores de ataque que utilizan los atacantes, las vulnerabilidades que explotan y los activos a los que apuntan.

La siguiente imagen ilustra la integración del proceso de inteligencia de amenazas identificando la información de entrada y salida de este:

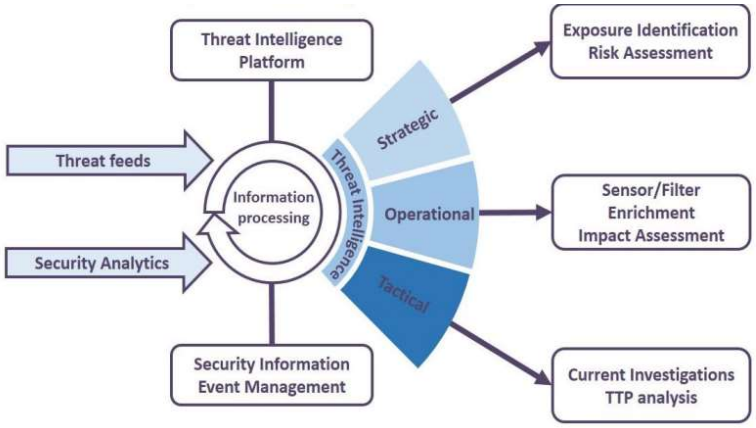

Imagen 6 - Generación de inteligencia de amenazas - Fuente: [21].

2.3.5. Gestión del cumplimiento

[13] La implementación de iniciativas de cumplimiento no tiene por qué ser un ejercicio basado en papel. En cambio, las organizaciones pueden crear metadatos que representen los requisitos de cumplimiento e integrarlos

directamente en los activos. Esto también se puede utilizar para la automatización de políticas de seguridad y para implementar la arquitectura de seguridad deseada. Al codificar los requisitos de cumplimiento en DevSecOps, el proceso de desarrollo se convierte en un facilitador del programa de cumplimiento.

3. DevSECOps: Principios de seguridad en CI/CD

3.1. Integración y despliegue continuos (CI/CD)

[4] La integración y el despliegue continuos (*Continuous Integration and Continuous Deployment/Delivery ó CI/CD*) son la piedra angular de los principios y prácticas de DevOps, ya que permiten la entrega de funcionalidades pequeñas al proporcionar ciclos de retroalimentación cortos y fortalecer la opinión del cliente. El proceso de CI/CD incluye todos los pasos para tomar los cambios en el código del desarrollador y lograr que se compilen e implementen para el cliente. Esto incluye los siguientes pasos:

1. Tomar el código desde el repositorio de código fuente.
2. Integrarlo con el resto de la aplicación.
3. Compilar la aplicación completa.
4. Probar la aplicación.
5. Desplegar la aplicación en ambientes productivos y de forma automatizada.

El proceso completo de integración y despliegue continuos se conoce como *pipeline de CI/CD* y, a diferencia de otras metodologías, rompe los silos de desarrollo, calidad y operaciones, incorporando todas sus tareas en un proceso que contiene una serie de pasos automáticos que las contempla.

La integración continua permite a los desarrolladores integrar su código con el de otros que están trabajando en la misma aplicación, en cualquier momento. Por otra parte, el despliegue continuo toma el código completo, fabrica la aplicación y la deja en un estado de “lista para el despliegue”, de forma que la misma pueda ser instalada en producción en cualquier momento. Es importante destacar que muchas veces se deben elegir los momentos justos para desplegar una aplicación con nuevas funcionalidades a producción, ya que esto tiene un impacto en la experiencia del cliente. Por ello, es importante destacar que el proceso de CI/CD deja lista la aplicación para que sea desplegada cuando llegue

ese momento, habiendo automatizado todas las tareas previas para que la puesta en producción sea prácticamente instantánea.

En la siguiente imagen se pueden visualizar las etapas básicas de un pipeline de CI/CD, que, como se mencionó anteriormente, comprenden desde la planificación y codificación hasta el despliegue y el monitoreo continuo.

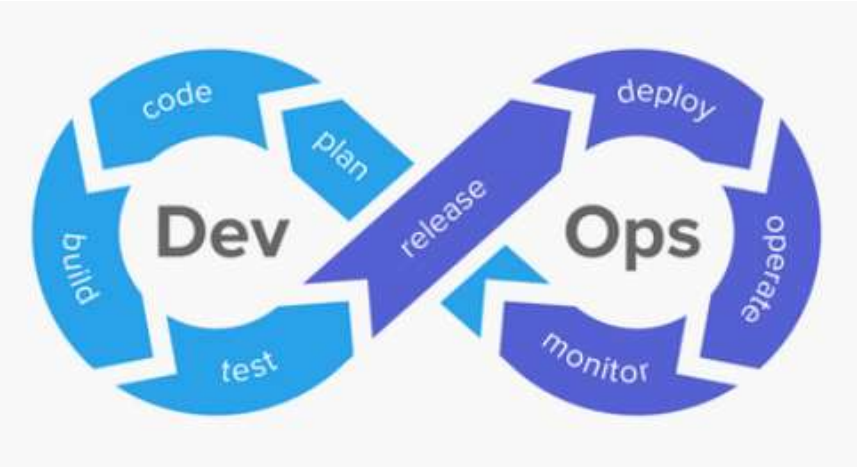

Imagen 7 - Etapas de CI/CD en DevOps - Fuente: [23].

3.2. Seguridad como código

En la era de DevSecOps, "Seguridad como código" se refiere a la práctica de integrar principios y controles de seguridad directamente en los procesos de desarrollo e implementación. Este enfoque trata las políticas y configuraciones de seguridad como código, que se puede controlar por versiones, probar automáticamente e implementar de manera consistente en todos los entornos.

La seguridad como código permite a las organizaciones automatizar las tareas de seguridad, como la gestión de la configuración, el análisis de vulnerabilidades y las comprobaciones de cumplimiento. Al definir estándares de seguridad en el código, los equipos pueden garantizar que las medidas de seguridad se apliquen de manera consistente, lo que reduce el riesgo de error humano y hace que el sistema sea más predecible y sólido. [24]

Para profundizar los conceptos, más adelante se mencionarán herramientas que permiten establecer controles y realizar análisis dinámicos de seguridad sobre la infraestructura como código (*IaC*).

3.3. Perfeccionando el Shift Left

Durante el primer capítulo se mencionaron los fundamentos de y beneficios de la metodología que consiste en “desplazar la seguridad a la izquierda”, sobre todo en metodologías DevSecOps.

Básicamente, la solución es introducir la seguridad en una etapa más temprana del proceso en lugar de hacerlo en los pasos finales. Tener en cuenta la seguridad en el diseño mediante el modelado de amenazas y dividir las pruebas de seguridad enormes en pruebas de seguridad más pequeñas e integrarlas en el proceso de desarrollo. [23]

Para perfeccionar el enfoque *Shift Left*, es fundamental adoptar herramientas de seguridad que se incorporen directamente en los entornos de desarrollo (*IDE*) proporcionando información en tiempo real. Estas soluciones pueden identificar posibles problemas a medida que los desarrolladores escriben código, recomendar prácticas seguras e incluso evitar confirmaciones si se encuentran vulnerabilidades críticas. Además, la integración del modelado de amenazas en las etapas iniciales de la planificación del proyecto ayuda a anticipar y abordar los desafíos de seguridad de manera proactiva. La incorporación de expertos en seguridad en los equipos de desarrollo mejora este proceso, lo que garantiza que se tenga en cuenta la seguridad en cada fase y fortalece la postura de seguridad general de los proyectos. [24]

3.4. Estrategias para pruebas de seguridad

A continuación, se describen dos estrategias para realizar pruebas de seguridad:

- Test Positivo: Es un tipo de testeo que asegura que toda la aplicación se comporta de acuerdo con lo esperado. Está pensado asumiendo que la utilización de la aplicación será únicamente enviando valores válidos, por ejemplo: si la aplicación espera un valor entre 0 y 10, el testeo positivo comprobará que los valores 0,1,2,3,4,5,6,7,8,9 y 10 se comporten de forma correcta.

- Test Negativo: Es un tipo de testeo que busca comprobar si la aplicación se comporta en forma correcta en condiciones inesperadas. Este enfoque permite comprobar la calidad del desarrollo de forma mucho más exhaustiva. En contraposición con la estrategia anterior, si la aplicación espera un valor entre 0 y 10, el testeo negativo comprobará que valores distintos a 0,1,2,3,4,5,6,7,8,9 y 10 se comporten en forma correcta y sean bien manejados por la aplicación.

La siguiente tabla (Tabla 1) establece una breve comparativa entre ambas estrategias presentadas:

Test Positivo	Test Negativo
Se realiza únicamente en condiciones esperadas.	Se realiza únicamente en condiciones inesperadas.
No tiene cobertura para todos los casos posibles.	Tiene cobertura para todos los casos posibles.
No asegura un producto de buena calidad.	Asegura un producto de buena calidad.
Es menos importante que el test negativo.	Es más importante que el test positivo.
Puede ser realizado por un equipo con un nivel de conocimiento bajo.	Debe ser realizado por un equipo con un nivel de conocimiento alto.
Toma menos tiempo.	Toma más tiempo
Se realiza en todas las aplicaciones y todos los casos.	Se realiza únicamente cuando existan posibilidades de condiciones inesperadas.
Asegura que la aplicación es normal.	Asegura que la aplicación no tiene defectos.

Tabla 1 – Estrategias para pruebas de seguridad - Fuente: [25].

3.5. Metodologías para pruebas de seguridad

3.5.1. Metodología estática

La metodología de testeo estático está pensada con el objetivo de prevenir defectos en el código, escaneándolo, pero no ejecutándolo en sí mismo. Las pruebas estáticas se realizan en la etapa inicial del desarrollo para evitar errores, ya que es más fácil encontrar las fuentes de fallas y que éstas se puedan corregir fácilmente.

Por otra parte, las pruebas estáticas se realizan bajo la modalidad de caja blanca, lo que implica que quien las ejecute tiene conocimiento total del código que quiere escanear, por ejemplo, cuando el propio programador verifica cada línea del código antes de hacer la entrega.

Si bien existen diversas técnicas para realizar pruebas estáticas, algunas de las más interesantes desde el punto de vista de la seguridad son las revisiones técnicas: donde se verifican los requerimientos y especificaciones técnicas en busca de discrepancias entre definiciones e implementaciones, por ejemplo la utilización de algoritmos de cifrado débiles; o las revisiones de código: donde se verifican aspectos relacionados a la sintaxis y las buenas prácticas de desarrollo como por ejemplo la presencia de credenciales en el código (*hard coded credentials*), entre otros.

Algunos de los beneficios de la metodología estática incluyen: la detección temprana de defectos, la relación entre costo y beneficio respecto de otras metodologías, la facilidad para encontrar problemas en el código y el incremento de la productividad de los desarrolladores.

En contraposición, sus principales limitaciones están relacionadas a la imposibilidad de ejecutar el código de la aplicación completa y por lo tanto correlacionar sus componentes. En una arquitectura de microservicios, el análisis estático analizará cada microservicio como un componente independiente.

3.5.2. Metodología dinámica

La metodología de testeo dinámico está pensada con el objetivo de comprobar el comportamiento y funcionamiento de una aplicación completa. Las pruebas dinámicas implican ejecutar la aplicación, introducir valores, validar el resultado y compararlo con el resultado esperado. A diferencia de las pruebas estáticas, las dinámicas requieren un mayor conocimiento, por parte del evaluador, del funcionamiento de la solución que se quiere probar. Sin embargo, esta metodología entrega resultados más realistas que las pruebas estáticas.

Básicamente existen dos técnicas para realizas escaneos dinámicos: pruebas de caja blanca y de caja negra. Las primeras están orientadas a

comprobar el comportamiento interno y cómo trabaja el código fuente. Las últimas, están enfocadas en validar la salida de la aplicación sin tomar detalle del funcionamiento interno, es decir, verifica la respuesta de acuerdo con los valores de entrada de la prueba.

Algunos de los beneficios de la metodología dinámica incluyen: revelar errores en tiempo de ejecución, por ejemplo, detectando información sensible que se envíe en texto plano; verificar la correcta integración entre los componentes de la solución, como ser el manejo de sesiones inseguras; y aumentar la confiabilidad de la aplicación completa de acuerdo con los requerimientos y definiciones iniciales.

Por otra parte, las mayores limitaciones de la metodología están relacionadas al mayor tiempo, esfuerzo y complejidad que demandan las pruebas, y a que, en muchos casos, pueden no cubrir el 100% de los escenarios.

3.5.3. Comparación entre estático y dinámico

La siguiente tabla (Tabla 2) establece una breve comparativa entre las metodologías estática y dinámica:

Metodología estática	Metodología dinámica
Evalúa defectos sin ejecutar el código.	Evalúa defectos ejecutando y correlacionando el código.
Su objetivo es prevenir defectos.	Su objetivo es prevenir y subsanar defectos.
Se ejecuta en la etapa más temprana del ciclo de vida de desarrollo.	Se ejecuta en la etapa más tardía del ciclo de vida de desarrollo.
Se ejecuta antes de desplegar el código.	Se ejecuta luego de desplegar el código.
Es menos costoso.	Es más costoso.
Toma menos tiempo.	Toma más tiempo.
Tiene cobertura del 100% del código.	Tiene cobertura mucho menor al 100% del código (en el orden del 50%)
Incluye revisiones informales, tutoriales, revisiones técnicas, revisiones de código e inspecciones.	Incluyen tanto pruebas funcionales como no funcionales.
Es un proceso de verificación.	Es un proceso de validación.

Tabla 2 – Comparación entre metodología estática y dinámica - Fuente: [26].

3.5.4. Metodología interactiva

A diferencia de las metodologías estática y dinámica, el análisis interactivo combina aspectos de ambas integrándose directamente con la aplicación en tiempo de ejecución y monitoreando su comportamiento en búsqueda de fallos. A través de la utilización de sensores que monitorean la aplicación y sus interacciones con la infraestructura, código y otros componentes, recopila datos y los analiza para identificar vulnerabilidades de seguridad.

En lo que se refiere a la etapa en la que se ejecutan este tipo de pruebas, es más cercana a la metodología dinámica, inclusive luego de su ejecución ya que se requiere la interacción con el entorno de ejecución verdadero de la aplicación.

Algunos de los beneficios adicionales de este enfoque son: su menor tasa de falsos positivos en comparación con las otras pruebas y la posibilidad de comprobar el comportamiento de la aplicación y su interacción con el sistema.

4. DevSECOps: Actividades y herramientas de seguridad

Durante el capítulo anterior se ha presentado en la “Imagen 7”, como una representación gráfica del ciclo completo de CI/CD de DevOps, sin aplicar medidas de seguridad sino únicamente centrados en las etapas de desarrollo y operación.

Luego de explicar los principios de seguridad para diseñar un proceso de CI/CD seguro, se propone un nuevo flujo DevSecOps incorporando prácticas y controles de seguridad:

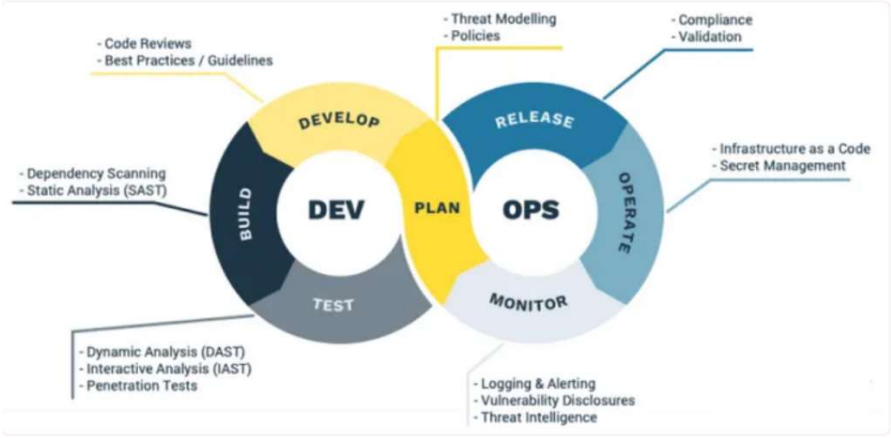

Imagen 8 - Actividades de seguridad en CI/CD en DevSecOps - Fuente: [30].

4.1. Etapa de planificación (*Plan*)

4.1.1. Modelado de amenazas

[23] Se trata de una lista sistemática de todas las formas posibles en las que se puede atacar una aplicación. Por lo tanto, el modelado de amenazas es un proceso para analizar los ataques de forma activa, obteniendo como resultado, una lista de amenazas o escenarios probables. Además, el enfoque debe ser holístico para considerar todas las amenazas y no una parte específica de una aplicación.

Por otro lado, el modelado de amenazas es un proceso colaborativo y repetible.

Los términos que se manejan en el modelado de amenazas son:

- Debilidad: representa un defecto o bug de la aplicación (por ej: la falta de validación del mail).
- Vulnerabilidad: representa una debilidad que puede ser explotada (por ej: la posibilidad de completar el campo del mail con una consulta SQL).
- Ataque: tiene un objetivo, un actor o atacante y un vector de ataque que representa la ruta que debe seguir el atacante para realizarlo.
- Superficie de ataque: representa lo que puede utilizar el atacante para perpetuar el ataque.
- Riesgo: probabilidad e impacto de producirse el riesgo identificado.

El modelado de amenazas debe realizarse en la etapa más temprana posible del ciclo de vida de desarrollo, de forma de identificarlas lo más temprano posible y que el costo de su resolución sea el más bajo.

Existen al menos tres enfoques:

- Centrado en los activos: Crear una lista de activos, jerarquizarlos y luego modelar las amenazas posibles sobre cada activo.
- Centrado en el atacante: Identificar motivaciones, objetivos y oportunidades de un atacante y a partir de ello modelar las amenazas.
- Centrado en la aplicación: Dibujar un diagrama de componentes de la aplicación, identificar que modelos pueden aplicar (por ej: OWASP Top 10) y luego clasificar y ordenar las amenazas.

4.2. Etapa de desarrollo (*Develop*)

4.2.1. Codificación segura

[24] Las prácticas de codificación segura son esenciales para reducir las vulnerabilidades y protegerse contra las amenazas. Establecer y seguir pautas de codificación segura ayuda a los desarrolladores a escribir código más seguro desde el principio. Estas pautas suelen incluir principios como:

- Validación de entrada: asegurar que todos los datos de entrada estén validados para evitar ataques de inyección.
- Autenticación y autorización: implementar una autenticación sólida y asegurarse que los usuarios tengan los permisos adecuados.
- Manejo de errores: desarrollar prácticas seguras de manejo de errores que no expongan información confidencial.
- Cifrado: utilizar cifrado robusto para proteger los datos en tránsito y en reposo.

Educar a los desarrolladores sobre estas pautas a través de capacitación y revisiones de código periódicas fomenta una cultura consciente de la seguridad.

Para aprovechar las prácticas de codificación segura establecidas, es fundamental integrar herramientas de seguridad en los entornos de desarrollo (*IDE*). Estas herramientas pueden aplicar automáticamente los estándares de codificación e identificar posibles problemas de seguridad en la etapa de escritura del código, escanear y analizar el código a medida que el desarrollador lo escribe, notificándolo sobre posibles debilidades y sugiriéndole soluciones [27].

Por otra parte, la incorporación de sesiones de programación entre pares centradas en la seguridad puede mejorar en gran medida la comprensión y el cumplimiento de estas pautas. La programación entre pares no solo mejora la calidad del código, sino que también difunde la conciencia de seguridad entre los miembros del equipo, lo que la convierte en un enfoque de doble propósito para la codificación segura.

4.3. Etapa de implementación (*Build*)

4.3.1. Análisis estático de seguridad de aplicaciones (SAST)

[23] El análisis de código estático (también conocido como análisis de código fuente) se realiza generalmente como parte de una revisión de código (también conocida como prueba de caja blanca) y se lleva a cabo en la fase de construcción. El análisis de código estático se refiere comúnmente a la ejecución de herramientas de análisis de código estático (SAST) que intentan resaltar posibles vulnerabilidades dentro del código fuente "estático" (que no se ejecuta) mediante el uso de técnicas como el análisis de contaminación y el análisis de flujo de datos.

[4] Las herramientas SAST se pueden integrar en el proceso de construcción y ejecutarse cada vez que se incorpora un nuevo código. La herramienta que se encargue de la orquestación del ciclo de integración y despliegue continuo (CI/CD) se puede configurar para rechazar o marcar cualquier problema potencial. Al detectar y rechazar estos problemas en una etapa temprana del proceso CI/CD, los desarrolladores pueden asegurarse de que nunca se introduzcan problemas de seguridad en el código fuente. Además, al proporcionar una respuesta inmediata al desarrollador, el costo de solucionar estos problemas es menor. Identificar los problemas en una etapa temprana del ciclo de vida del desarrollo de software es un gran ejemplo del concepto Shift Left.

Los analizadores de código fuente pueden ejecutarse en código no compilado: para comprobar defectos como errores numéricos, validación de entrada, punteros y referencias, entre otros; o en código binario y de byte: que hacen básicamente lo mismo, pero en el código compilado. Algunas herramientas se ejecutan únicamente en código fuente, algunas solo en código compilado y otras en ambas formas.

[23] El escaneo estático es una buena forma de detectar problemas de codificación como: problemas de sintaxis, vulnerabilidades de seguridad, errores

de programación, desvíos respecto de los estándares de codificación y valores indefinidos.

Algunas de las herramientas SAST más destacadas del mercado son: SonarQube, Veracode, HCL AppScan, Checkmarx y Fortify.

En la siguiente imagen se muestra el funcionamiento de una herramienta SAST:

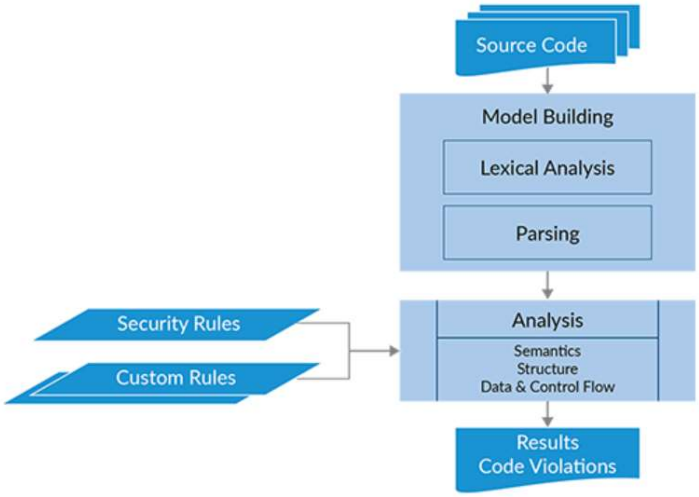

Imagen 9 - Funcionamiento de herramientas SAST - Fuente: [31].

4.3.2. Análisis de seguridad de composición de aplicaciones (SCA)

[4] Durante la fase de compilación de la aplicación, se puede utilizar el análisis de composición de aplicaciones (SCA) para comprobar si existen vulnerabilidades en las dependencias o bibliotecas.

Las vulnerabilidades de terceros se están convirtiendo cada vez más en un área de riesgo importante para las organizaciones, un ejemplo ampliamente conocido fue el de la vulnerabilidad en “Apache log4j” [28]. SCA analiza todo el código fuente para determinar posibles vulnerabilidades conocidas en bibliotecas vinculadas y referencias de código de código abierto.

Las principales herramientas de SCA incluyen GitHub dependabot, Snyk y FOSSA.

Al escanear en la fase de compilación, puede asegurarse que el código que se envía sea seguro y que las dependencias externas, como las bibliotecas de código abierto, también lo sean.

En la siguiente imagen se muestran los componentes de una aplicación modelo donde se realiza una división entre el código generado por los desarrolladores y el software de terceras partes que se incluye como parte del sistema. Sobre este último es que se realiza el análisis de vulnerabilidades SCA.:

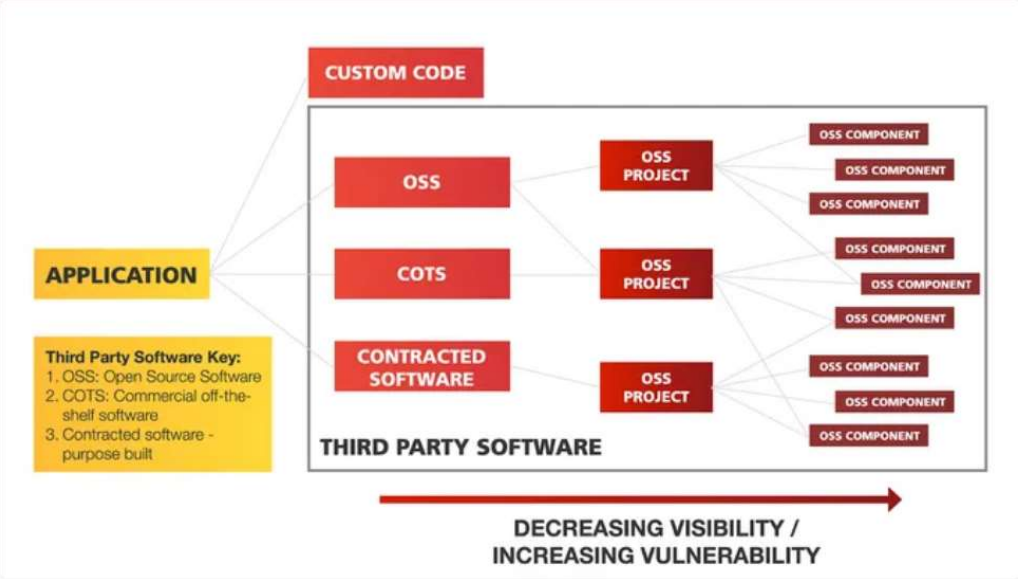

Imagen 10 - Composición de una aplicación – Análisis SCA - Fuente: [30].

4.3.3. Análisis de vulnerabilidades en contenedores

[24] Los contenedores se han convertido en un elemento básico en la implementación de aplicaciones modernas, pero también presentan nuevos desafíos de seguridad. Una gestión eficaz de las vulnerabilidades de los contenedores implica:

- Escaneo periódico: las herramientas automatizadas deben escanear las imágenes de los contenedores en busca de vulnerabilidades conocidas durante el proceso de compilación y antes de la implementación.
- Seguimiento de dependencias: realizar un seguimiento de las bibliotecas y los paquetes utilizados en los contenedores para asegurarse que estén actualizados y libres de vulnerabilidades.

- Gestión de la configuración: proteger las configuraciones de los contenedores siguiendo las mejores prácticas para minimizar las superficies de ataque.

Las imágenes y los registros de contenedores (*registry*) son componentes centrales de un entorno contenerizado y deben protegerse adecuadamente:

- Garantía del origen de la imagen: utilizar únicamente imágenes base confiables y realizar escaneos continuos para detectar vulnerabilidades y configuraciones incorrectas.
- Seguridad de registro: proteger los registros de contenedores con controles de acceso sólidos, cifrado y utilizar imágenes firmadas para verificar su integridad.

La implementación de estas medidas ayuda a prevenir el acceso no autorizado y garantiza que los contenedores estén libres de vulnerabilidades, lo que reduce el riesgo de violaciones de seguridad.

[23] Algunas de las herramientas más destacadas para análisis de vulnerabilidades en contenedores son: Clair, Trivy, Falco y Harbor.

En la siguiente imagen se puede observar cómo el análisis de vulnerabilidades en contenedores se aplica en las distintas etapas del CI/CD. En la etapa de desarrollo (*Develop*) se chequean las vulnerabilidades en las imágenes del código y del registro de imágenes. En la etapa de despliegue (*Deploy*) se define una política que debe ser respetada al momento de desplegar una imagen en ambientes productivos, permitiendo o no avanzar con el mismo. Por último, en la etapa de ejecución (run) se chequea el cumplimiento continuo y la seguridad aplicada al *runtime* del contenedor.

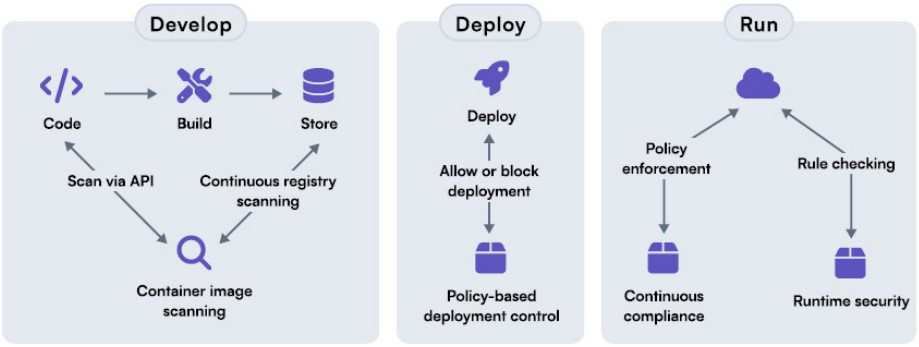

Imagen 11 - Análisis de vulnerabilidades en contenedores en las etapas de CI/CD - Fuente: [24].

4.3.4. Manejo seguro de secretos

[30] Las herramientas de detección de secretos desempeñan un papel fundamental en la seguridad de las aplicaciones, ya que identifican y administran información confidencial, como contraseñas, claves para el consumo de APIs, tokens secretos y otras credenciales que podrían exponerse inadvertidamente en el código fuente o en los archivos de configuración.

Las herramientas de detección de secretos buscan patrones en el código que indiquen un secreto y lo marcan para que los desarrolladores puedan investigar.

Algunas de las principales ventajas de las herramientas de detección de secretos son:

- Prevenir rutas de ataque comunes: el método de infiltración más común para acceder a datos confidenciales es aprovechar los secretos y las claves de acceso expuestas.
- Cumplir con los estándares regulatorios: debido al papel de los secretos en tantos ataques, la mayoría de los marcos de cumplimiento regulatorio requieren capacidades de escaneo y auditoría de secretos para rastrear la exposición de secretos.
- Integración con ciclo de vida de desarrollo: las herramientas de detección de secretos se integran bien con los entornos de desarrollo (IDE) y los orquestadores de procesos CI/CD para

escanear iterativamente el código a medida que se fusiona, lo que facilita su automatización y escalabilidad.

En contraposición, se mencionan algunas desventajas de las herramientas de detección de secretos:

- Falsos positivos: al igual que con muchas otras herramientas de análisis, existe un problema recurrente de falsos positivos/negativos con las herramientas de escaneo de secretos automatizado. Los patrones diseñados para reconocer secretos pueden marcar cadenas de caracteres inofensivas.
- Falsos negativos: la efectividad de la detección de secretos depende de los algoritmos y patrones que la herramienta está configurada para reconocer. Es posible que no se detecten patrones nuevos o inusuales de secretos si la herramienta no se actualiza regularmente, lo que puede generar una falsa sensación de seguridad. Si bien estas herramientas son efectivas para detectar muchos tipos de secretos, no son infalibles y deben ser parte de una estrategia de seguridad más amplia que incluya cifrado, controles de acceso y otras medidas de seguridad.

En la imagen que se muestra a continuación se puede ver un ejemplo de cómo, en cada integración realizada desde el desarrollo hasta el despliegue, se exponen distintos tipos de secretos:

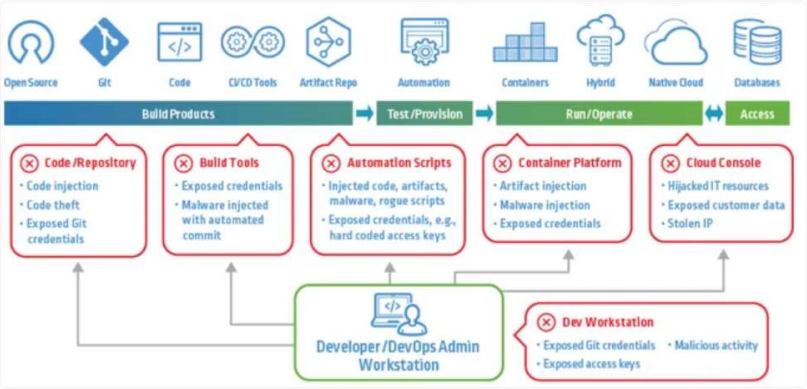

Imagen 12 - Exposición de secretos - Fuente: [30].

4.4. Etapa de pruebas (*Test*)

4.4.1. Análisis dinámico de seguridad de aplicaciones (DAST)

[29] Una herramienta de análisis dinámico de seguridad de aplicaciones (DAST) ejecuta pruebas de penetración automatizadas para encontrar vulnerabilidades en las aplicaciones, mientras éstas se ejecutan.

DAST automatiza el enfoque de un hacker y simula ataques del mundo real para amenazas críticas como *cross-site scripting* (XSS), *SQL injection* (SQLi) o *cross-site request forgery* (CSRF) para descubrir vulnerabilidades y configuraciones incorrectas que otras herramientas de seguridad no pueden detectar.

DAST es completamente independiente del lenguaje y examina la aplicación desde afuera hacia adentro. Con una aplicación en ejecución en un entorno de prueba, los análisis DAST se pueden incorporar/automatizar en un proceso de CI/CD o ejecutar de forma independiente a demanda.

El uso de herramientas de análisis dinámico de seguridad de aplicaciones durante el ciclo de vida del desarrollo de software permite a los equipos descubrir vulnerabilidades antes de que sus aplicaciones estén en producción.

[23] Algunas de las herramientas DAST más destacadas del mercado son: Acunetix, Veracode Dynamic Analysis, HCL AppScan, BurpSuite y Fortify.

4.4.2. Análisis interactivo de seguridad de aplicaciones (IAST)

[30] Las herramientas IAST combinan aspectos de análisis estático y dinámico. Se implementan como agentes dentro de la aplicación o el entorno de prueba y monitorean la aplicación desde adentro mientras ejecuta pruebas en busca de vulnerabilidades de seguridad.

DAST envía solicitudes y observa las respuestas sin comprender los cambios de estado internos ni los cálculos que producen estas respuestas. IAST mejora esto al integrar un componente de monitoreo adicional dentro del entorno de prueba. Este componente monitorea activamente el comportamiento interno de la aplicación durante las pruebas, incluidas las operaciones normales y los

ataques simulados por DAST. Esto permite a IAST observar operaciones internas como conexiones de bases de datos, acceso al sistema de archivos y la ejecución de funciones específicas dentro del código de la aplicación.

IAST puede detectar el uso de funciones criptográficas inseguras y proporciona un análisis profundo, muy similar a una resonancia magnética, de lo que sucede dentro de la aplicación, lo que brinda información sobre problemas sutiles que podrían no alterar el comportamiento observado desde el exterior, pero podrían ser indicativos de fallas de seguridad graves.

Algunas de las ventajas de utilizar herramientas IAST son:

- Respuesta rápida sobre seguridad: las herramientas IAST brindan una respuesta inmediata al analizar las aplicaciones en tiempo real mientras se ejecutan. Esto permite una identificación y solución más rápida de los problemas de seguridad.
- Bajo nivel de falsos positivos: al trabajar dentro de la aplicación mientras se ejecuta, las herramientas IAST pueden lograr un alto nivel de precisión en la identificación de vulnerabilidades, lo que reduce la cantidad de falsos positivos y negativos.
- Amplia gama de detección de vulnerabilidades: IAST puede detectar una variedad de problemas de seguridad, incluidos los relacionados con la configuración, el flujo de datos y las vulnerabilidades tanto del lado del cliente como del lado del servidor.

4.4.3. Análisis de vulnerabilidades en infraestructura

[23] DevOps hace un gran trabajo en la automatización del proceso de desarrollo e implementación, pero como todas las partes móviles (contenedores y bibliotecas, entre otras) se actualizan con frecuencia, es importante asegurarse que la infraestructura donde se implementa el código fuente sea segura.

La mejor forma de hacerlo es incorporar un análisis de vulnerabilidades en el flujo de trabajo.

Un escáner de vulnerabilidades es una aplicación diseñada para evaluar arquitecturas con computadoras, redes o aplicaciones en busca de debilidades conocidas. Estas herramientas se utilizan en la identificación y detección de vulnerabilidades que surgen de configuraciones incorrectas o programación defectuosa dentro de un activo tecnológico como un firewall, un *router*, un servidor web o un servidor de aplicaciones.

Los escáneres modernos suelen estar disponibles como servicio en la nube y, a menudo, tienen la capacidad de personalizar los informes de vulnerabilidades, así como el software instalado, los puertos habilitados, los certificados asociados y otra información relevante.

Existen dos tipos de análisis:

- Los análisis autenticados: que permiten que el escáner acceda directamente a los activos tecnológicos mediante protocolos administrativos remotos como SSH o RDP, y se autentique mediante las credenciales válidas para el sistema. Esto permite que el escáner de vulnerabilidades acceda a datos de bajo nivel, como servicios específicos y detalles de configuración del sistema operativo y que luego, pueda proporcionar información detallada y precisa sobre el sistema operativo y el software instalado, incluidos los problemas de configuración y los parches de seguridad faltantes.
- Los análisis no autenticados: que son un método que puede generar una gran cantidad de falsos positivos y no puede proporcionar información detallada sobre el sistema operativo de los activos y el software instalado. Pero es un método que se utiliza para intentar determinar la postura de seguridad de los activos a los que se puede acceder desde el exterior.

4.4.4. Evaluaciones de seguridad manuales (penetration test)

[27] Se trata de una prueba de penetración que utiliza un conjunto de herramientas y procedimientos para evaluar la seguridad del sistema mediante la inyección de ciberataques simulados autorizados en la aplicación.

El orquestador de CI/CD no automatiza este tipo de pruebas, pero los resultados de la prueba pueden ser un punto de control en el proceso.

A menudo las evaluaciones de seguridad manuales combinan algunas herramientas y métodos similares a SAST, DAST, IAST, análisis de vulnerabilidades en infraestructura, manejo de secretos, buenas prácticas de seguridad en la codificación, entre otras, pero con el diferencial del análisis propio y personal del experto que lo realice.

4.5. Etapa de versionado (*Release*)

4.5.1. Validación de integridad del código

El proceso de validación de la integridad del código puede dividirse en dos etapas:

- Firma de artefactos: utilizar la firma digital para firmar los componentes generados (imágenes, contenedores, binarios, entre otros) para asegurar que no hayan sido manipulados por fuera del circuito definido y esperado.
- Validación por hash: luego de firmados los componentes, se debe validar que el hash resultante coincida al momento de desplegar esos componentes al ambiente productivo.

4.5.2. Validación de cumplimiento normativo

Es el proceso que asegura que la aplicación generada cumple con las regulaciones y normativas vigentes para la organización (por ejemplo: PCI-DSS, BCRA, GDPR, ISO).

Adicionalmente se puede volver a efectuar un escaneo de licencias para asegurar que las dependencias y bibliotecas externas utilizadas cumplan con las licencias de código abierto permitidas por la organización.

4.6. Etapa de operación (*Operate*)

4.6.2. Infraestructura como código (IaC)

[30] La infraestructura como código (IaC) ofrece un enfoque innovador para el aprovisionamiento y la gestión de la infraestructura en la nube a través de código, en lugar de hacerlo mediante procesos manuales.

Este cambio fundamental no solo acelera los ciclos de desarrollo, sino que también introduce nuevas dimensiones de riesgo que deben gestionarse con cuidado.

Existe una amplia variedad de configuraciones de seguridad de IaC incorrectas que pueden exponer las aplicaciones a posibles vulnerabilidades:

- Secretos codificados de forma rígida: Algunas vulnerabilidades surgen de secretos, como claves API, claves de cifrado, contraseñas y otros, que se incrustan directamente en scripts de IaC. La mitigación comienza con el escaneo de los archivos de IaC y la identificación de secretos de todo tipo (similar a lo visto en 4.3.4. Manejo seguro de secretos).
- Privilegios excesivos: Muchas veces, a los usuarios, roles y otras identidades se les otorgan más privilegios de los que realmente necesitan. La mitigación implica revisar rigurosamente los privilegios otorgados para garantizar que sigan el principio del mínimo privilegio.
- Datos sin cifrar: Almacenar datos confidenciales sin cifrar puede provocar una fuga de datos potencialmente peligrosa, lo que enfatiza la necesidad de cifrar en la infraestructura como archivos de configuración de código.

4.7. Etapa de monitoreo (*Monitor*)

4.7.1. Monitoreo y observabilidad

[4] El monitoreo es el proceso de recopilar, analizar y mostrar métricas o registros para realizar un seguimiento del rendimiento, la disponibilidad y el estado general de un sistema. Su enfoque es generalmente reactivo. Se monitorean métricas predefinidas (como el uso de la CPU, la memoria o las tasas de error) y se establecen alertas cuando algo sale mal.

El monitoreo es útil para identificar “problemas conocidos” y realizar un seguimiento de métricas o eventos específicos que se han predeterminado como importantes.

La observabilidad es un concepto más amplio que se centra en lo bien que se puede comprender e inferir el estado interno de un sistema complejo en función de los resultados (métricas, registros y rastros) que genera. Su enfoque es más proactivo y exploratorio. La observabilidad permite hacer nuevas preguntas sobre el sistema sin conocerlas de antemano, centrándose en problemas desconocidos o inesperados.

La observabilidad está diseñada para ayudar a diagnosticar “por qué” algo salió mal al proporcionar datos más completos que permiten obtener información más profunda sobre el comportamiento del sistema.

La observabilidad a menudo se basa en tres tipos de datos principales:

- Métricas: Datos cuantitativos sobre el rendimiento del sistema.
- Registros (eventos, logs): Registros de eventos discretos.
- Rastros (trazas): Información el flujo completo.

4.7.2. Monitoreo continuo

[24] El monitoreo continuo es un proceso vital para mantener la seguridad y el funcionamiento de un proceso CI/CD, habilita a las organizaciones a manejar los incidentes en forma proactiva y dar respuesta rápida a las ciber amenazas. El mismo se compone de:

- Monitoreo en tiempo real: Analizar la información del sistema, aplicación e infraestructura en tiempo real para detectar amenazas y potenciales brechas de seguridad.
- Sistemas de alerta: Diseñar e implementar alertas basadas en comportamientos predefinidos y anómalos para su detección y respuesta temprana.
- Tableros de seguridad: Utilizar tableros de control para identificar la postura de seguridad y ayudar a los equipos involucrados en la toma de decisiones.

Para establecer un buen proceso de monitoreo continuo, es necesario un correcto manejo y análisis de los eventos (logs) de todo el sistema:

- Centralización de eventos: Recolectar eventos de distintas fuentes de datos y centralizarlos para facilitar su análisis.
- Análisis automatizados: Utilizar herramientas que permitan automatizar el análisis de los eventos de forma de identificar patrones, anomalías y potenciales incidentes de seguridad.
- Resguardo y cumplimiento: Asegurar el resguardo de los eventos por un período de tiempo suficiente para asegurar el cumplimiento de las normativas vigentes.

4.7.3. Gestión de eventos e información de seguridad (SIEM)

[4] Debido a la cantidad de información que se puede generar en el ambiente productivo (aplicación, sistemas, infraestructura, entre otros), las herramientas SIEM se pueden utilizar para recopilar todos los datos y ayudar a los equipos de operaciones de seguridad a utilizar los datos en forma centralizada. Las herramientas SIEM recopilan datos de todas las fuentes de información relacionada con la seguridad, incluidos registros, métricas, rastros y eventos, para proporcionar una visión y una gestión unificadas de estos datos.

Estas herramientas proporcionan un análisis en tiempo real de los datos, lo que ayuda a dar sentido a la gran cantidad de información de los sistemas de

monitoreo y registro y ayuda a identificar posibles ataques. Las herramientas SIEM también pueden ayudar en la resolución de problemas y la respuesta ante incidentes al identificar la causa del incidente y ayudar con la automatización.

Las herramientas SIEM pueden detectar un incidente de seguridad y todas las alertas relacionadas con el incidente y presentarlas de una manera que permita tomar medidas con facilidad.

Por ejemplo, durante un ataque de tipo DoS (Denegación De Servicio), puede haber cientos o incluso miles de alertas a medida que los sistemas intentan responder a la avalancha de solicitudes. Las herramientas SIEM pueden correlacionar estas alertas y eliminar los duplicados, lo que facilita el diagnóstico del problema y permite una resolución más rápida.

4.8. Un flujo de CI/CD seguro

[23] El objetivo ideal es “detectar problemas de seguridad (por diseño o vulnerabilidad de la aplicación) lo más rápido posible”.

Como resumen del capítulo se muestra la siguiente imagen que ejemplifica un pipeline de CI/CD que tiene integradas una serie de pruebas, análisis, actividades y herramientas de seguridad (éstas últimas son solo a modo de ejemplo).

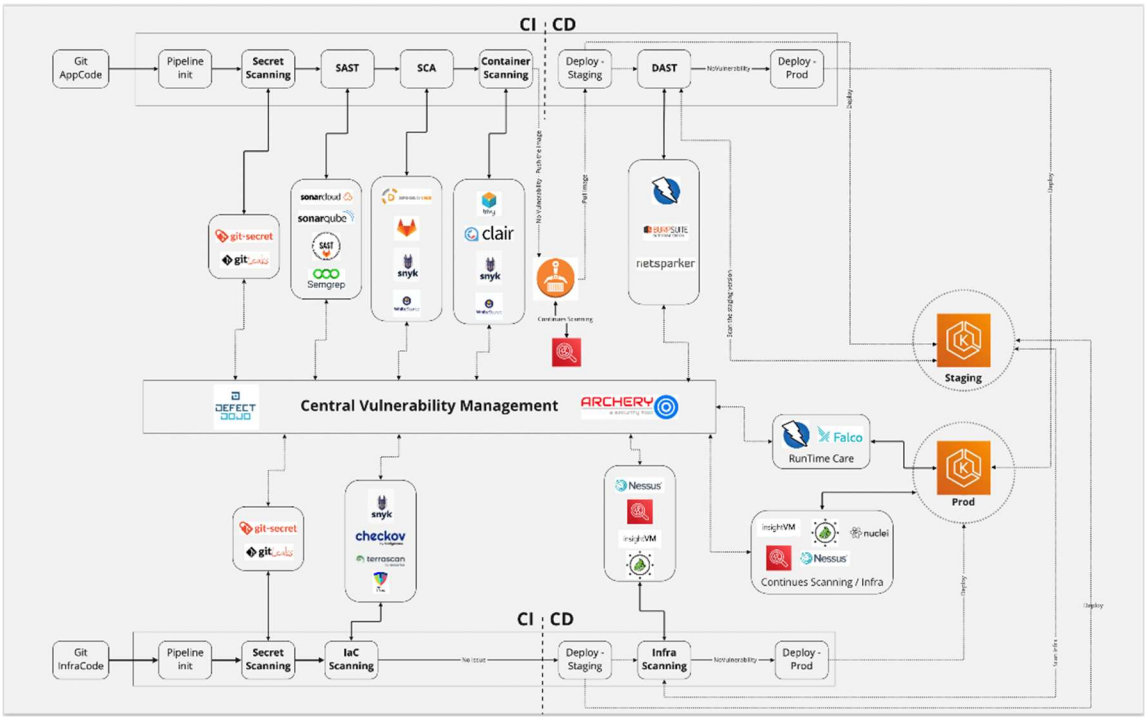

Imagen 13 - Ejemplo de CI/CD con actividades y pruebas de seguridad - Fuente: [23].

En el mismo se puede visualizar que, en la parte superior, el artefacto de entrada del pipeline es el código fuente, mientras que en la inferior es la infraestructura como código.

Por el camino superior, en la etapa de integración continua (CI), se realizan análisis estáticos de seguridad, de composición de aplicaciones, de vulnerabilidades en contenedores y de manejo seguro de secretos. Por el camino inferior, en la misma etapa, se realizan análisis de manejo seguro de secretos y de infraestructura como código.

En la etapa de despliegue continuo (CD), en el camino superior, se realizan análisis dinámicos de seguridad (aunque bien podrían sumarse análisis interactivos y evaluaciones de seguridad manuales). En el inferior se utilizan técnicas de análisis de vulnerabilidades en infraestructura (podrían ser también *penetration tests*).

Posteriormente y ya cuando la aplicación llega a la etapa de “staging” se pueden aplicar las medidas de seguridad descritas para la etapa de versionado (*release*) y, finalmente en la etapa de producción, las descritas en las etapas de operación y monitoreo.

5. Caso de estudio: Contexto actual

5.1. La organización

El *Banco República* es la entidad financiera más importante de su país, existe desde hace más de cien años, liderando en el rubro tanto a nivel local como regional.

Es su prioridad la atención a empresas de todos los tamaños, inclusive en áreas alejadas de las grandes ciudades y de menor relevancia económica. Esto implica que tiene un rol fundamental en la economía y el crecimiento de su país.

El banco pone a disposición una batería de soluciones económicas que incluyen créditos para inversión y capital de trabajo, otorgando, además, financiamiento y garantías a la actividad de comercio exterior.

Por otra parte, también promueve el desarrollo de los individuos mediante el ofrecimiento de una cartera de préstamos personales e hipotecarios para la vivienda personal.

5.2. El proyecto

El *Banco República* es líder en el mercado de servicios financieros ofrecidos a empresas. Sin embargo, la organización es consciente que su plataforma tecnológica se está quedando obsoleta, sumado a que sus procesos no se modernizaron lo suficiente, lo cual le impide tener un tiempo de respuesta lo suficientemente ágil como para atender las constantes demandas y cambios del mercado.

En este sentido, surgió el proyecto de la nueva *Banca Digital*, teniendo como premisa que esta solución permita ofrecer servicios omnicanales a todos sus clientes: empresas y personas. La nueva plataforma tiene como premisa ser integral y prestar servicios tanto en el canal web como en el móvil de forma transparente para el cliente.

La nueva plataforma para empresas toma protagonismo como el proyecto más importante dentro de la cartera del banco, siendo que representa la

verdadera transformación digital que hace tanto tiempo se busca en la organización.

5.3. La cultura y la forma de trabajo

El Banco se rige bajo un organigrama tradicional reflejado en una estructura jerárquica en la que los empleados y departamentos están organizados en niveles de acuerdo con sus funciones y responsabilidades.

La cultura de la empresa, en general, tiene características de trabajo en silos, lo cual conlleva algunos impactos negativos en la modernización:

- Innovación limitada: La falta de intercambio de ideas entre equipos frena la innovación y la creatividad.
- Toma de decisiones lenta: Debido a la falta de información compartida, las decisiones pueden tardar más tiempo en ejecutarse o estar basadas en datos incompletos.
- Falta de cohesión organizacional: Los equipos pueden no estar alineados con la visión y los objetivos estratégicos de la empresa.

Por otra parte, se utilizan muy pocas herramientas colaborativas lo que dificulta la comunicación entre los distintos equipos, las reuniones conjuntas y el seguimiento de tareas que requieren la participación de más de un departamento.

Como metodología de trabajo es ampliamente utilizado el desarrollo en cascada, lineal y secuencial, donde cada fase del proyecto debe completarse antes de pasar a la siguiente. Este enfoque le sienta cómodo al Banco ya que se ajusta a su cultura organizacional y de trabajo en silos.

En contraposición, si bien los equipos han trabajado alguna vez con metodologías ágiles (nunca DevOps o DevSecOps), se percibe una dificultad en la integración, la cultura colaborativa y la responsabilidad compartida.

5.4. La tecnología y las operaciones

El *Banco República* tiene vasta experiencia en el desarrollo, despliegue y securización de aplicaciones con arquitecturas monolíticas, es decir, donde todos los componentes y funcionalidades de la solución están integrados y ejecutados como una única unidad indivisible [32]. Por el contrario, la experiencia de los equipos es casi nula en lo que a implementación de microservicios se refiere.

Por otra parte, posee un gran *core* bancario sobre el que registra todas sus operaciones. Si bien es antiguo, es confiable y dispone de interfaces para que los otros sistemas interactúen con él.

En lo que respecta al tipo de *software* y *hardware* utilizado, es en su inmensa mayoría bajo la modalidad *on-premise*, es decir, que la organización mantiene sus propios servidores y *hardware*, y compra (o desarrolla) el *software* que implementa. Por el contrario, la experiencia en soluciones de tipo *cloud*, es decir, donde el *software* y/o *hardware* lo otorga un proveedor, prácticamente no existe. [33]

5.5. La ciberseguridad

5.5.1. El equipo

El equipo de seguridad de la información se encuentra organizado jerárquicamente como el resto del Banco. En la siguiente imagen se muestra una estructura parcial que involucra a los equipos que tienen mayor participación en el caso de estudio:

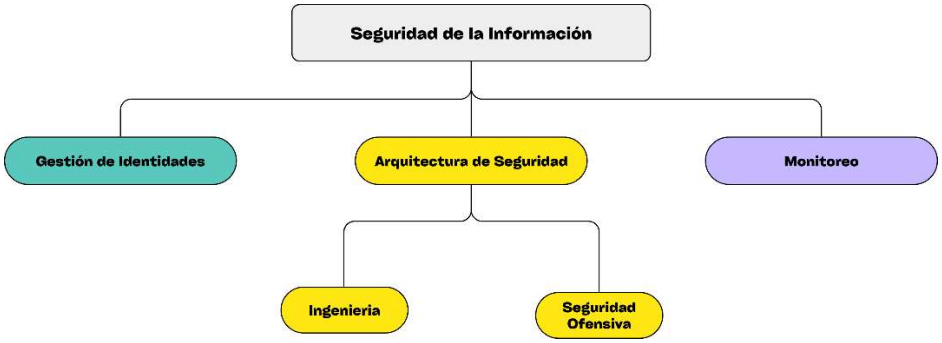

Imagen 14 - Caso de estudio: Organización del equipo de Seguridad de la Información - Fuente: Elaboración propia.

- El equipo de Gestión de Identidades es el brazo operativo del área. Entre sus principales funciones se encuentran el manejo de las identidades del Banco, como ser altas, bajas y modificaciones de usuarios, grupos y roles.
- El equipo de Monitoreo dirige las actividades de monitoreo de eventos de seguridad y el control de desvíos en los estándares implementados, gestionando los incidentes de ciberseguridad en la infraestructura tecnológica.
- El equipo de Ingeniería define las medidas de seguridad sobre la arquitectura tecnológica con el objeto de garantizar la correcta protección de la infraestructura y los servicios digitales.
- El equipo de Seguridad Ofensiva trabaja en las evaluaciones de seguridad con el objetivo de prevenir, detectar y eliminar las amenazas y brechas de seguridad, mediante la implementación de soluciones de propósito específico.

5.5.2. El rol en la organización

Para entender el rol de la ciberseguridad en la organización, es necesario remitirse a una afirmación realizada durante la introducción de este trabajo: “el crecimiento de las plataformas y servicios digitales trae como consecuencia un entorno digital que es cada vez más vulnerable a ataques sofisticados, lo que requiere de medidas de seguridad integrales y proactivas para tratar la constante evolución de las ciber amenazas”. En este sentido, el rol del área ha tomado una

mayor relevancia en los últimos años, crecimiento que se vio reflejado en el aumento de su dotación permanente, incorporando recursos con habilidades valiosas en el mercado actual, destacando arquitectos de ciberseguridad en entornos híbridos y analistas de seguridad web y *mobile* con experiencia en evaluaciones de seguridad.

Por otra parte, y como sucede en la mayoría de las organizaciones que operan en el mercado financiero, la ciberseguridad se ha transformado en un pilar en la toma de decisiones. Esto permite que el equipo participe activamente de todos los proyectos tecnológicos y de negocio del banco definiendo e implementando medidas de seguridad para la protección de los activos de la organización y alineándose con los objetivos de negocio.

5.5.3. El modelo de madurez

Para dar cuenta del nivel de madurez del área de seguridad de la información, se analizan los tópicos descritos en el primer capítulo sobre la evolución de la ciberseguridad, y se evalúa su madurez de acuerdo con los estadios propuestos en el modelo CMMI [34] (Inicial, Repetible, Definido, Gestionado, Optimizado).

- Seguridad perimetral: Su nivel de madurez está entre gestionado y optimizado. Las actividades de seguridad perimetral son proactivas, con métricas para medir su efectividad, y enfoques basados en riesgos. Se monitorean activamente las amenazas y vulnerabilidades, aplicando técnicas para el aprendizaje y la mejora continua.
- Defensa en profundidad: Su nivel de madurez oscila entre definido y gestionado. Los procesos de ciberseguridad están documentados y definidos, con políticas y procedimientos formales que cubren diferentes áreas. Se abordan los riesgos de manera más sistemática y las actividades de seguridad son proactivas.

- Zero Trust: Su nivel de madurez en promedio corresponde al estadio de definido, en muchos casos es repetible y en otros gestionado.
- Shift Left y seguridad en SDLC: Su nivel de madurez está entre inicial y repetible. Los procesos de ciberseguridad están adquiriendo mayor madurez, buscando ser estandarizados y planificados, con una estrategia que defina responsables, tareas y controles.
- DevSecOps: El nivel de madurez es inicial y la experiencia del equipo es prácticamente nula.

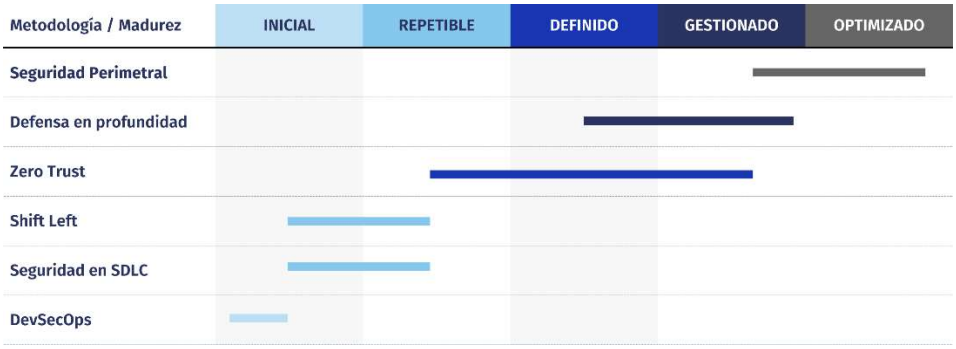

Imagen 15 - Caso de estudio: Metodologías de seguridad y modelo de madurez - Fuente: Elaboración propia.

5.6. La nueva plataforma

5.6.1. La metodología

La metodología adoptada para el desarrollo, despliegue y securización de la nueva solución será DevSecOps.

Si bien es cierto que los equipos del Banco no poseen la experiencia suficiente, desde la Dirección se priorizaron algunos de los beneficios de esta metodología (1.3.4. Beneficios y adopción en el mercado), decidiendo que es el momento justo para comenzar el camino y que, la importancia del proyecto a nivel institucional permitirá a los equipos tomar el aprendizaje con mayor responsabilidad.

5.6.2. La arquitectura

La solución implementará una arquitectura *Cloud Native* [35]. A continuación, se describen brevemente las características de este enfoque que permite fácil escalabilidad, resiliencia y mantenimiento en entornos híbridos (*cloud* y *on-premise*):

- Microservicios: Este tipo de aplicaciones se componen de módulos independientes conectados entre sí, denominados microservicios. Se trata de un patrón utilizado para expandir el proceso de desarrollo y entrega de software, evitando las estrategias lentas y riesgos de los desarrollos en cascada y monolíticos. Con microservicios se desarrolla la aplicación en pequeños servicios de manera totalmente independiente, inclusive en distintos lenguajes de programación y liderados por distintos equipos.
- Contenedores: Los contenedores son una especie de sistema operativo pequeño que contiene lo mínimo indispensable para correr una aplicación, en la que el código fuente viene empaquetado junto con sus librerías y dependencias, y se puede correr casi en cualquier lugar (servidor, escritorio, nube). Una de las ventajas más grandes de los contenedores es su portabilidad y comportamiento predecible en diferentes ambientes. Visto desde una perspectiva de la arquitectura tradicional: un contenedor es similar una máquina virtual, pero sin la carga e ineficiencia de montar un sistema operativo completo.
- Entrega continua: Es la habilidad que permite desplegar cualquier tipo de cambio (una nueva configuración, la solución a un error, entre otros) de una forma segura y rápida. Este concepto fue descrito anteriormente en este trabajo y se abordará con mayor profundidad más adelante.
- DevSecOps: Si, aplicar metodologías DevOps, o DevSecOps si se logra una integración de la seguridad, es un pilar fundamental de una arquitectura Cloud Native.

Por otra parte, pasando a las integraciones de la solución, se sabe que la nueva plataforma para empresas deberá integrarse con servicios internos del Banco (entre ellos el *core*), pero también otros provistos por entes externos (por ejemplo, para la validación biométrica o fe de vida de un cliente o bien para la utilización de múltiples factores de autenticación).

Este punto no es menor, es bien sabido que las interfaces suponen al menos, que los equipos de ciberseguridad presten atención en la definición de canales de comunicación seguros, mecanismos de autenticación y autorización, principios de mínimo privilegio, cifrado de información en reposo y en tránsito, entre otros.

5.6.3. La tecnología

A continuación, se describen algunas de las tecnologías y productos utilizados en la nueva plataforma, sin embargo, cabe aclarar, que todavía no se mencionan herramientas o soluciones de propósito específico utilizadas para la gestión de la ciberseguridad:

- Orquestador de contenedores: Red Hat Openshift [36] en sus versiones on-premise y nube (ARO).
- Entorno: Será híbrido, donde los ambientes de Producción y Homologación serán desplegados on-premise y el ambiente de Desarrollo será desplegado en la nube de Microsoft Azure [37].
- Repositorio de código: Se utilizará GitHub [38] como repositorio de código y también para implementar la infraestructura como código (IaC) a través de gitops.
- Herramientas colaborativas: Se utilizará Slack [39] y Microsoft Teams [40] para la comunicación de los equipos, y Confluence y Jira [41] para documentar los acuerdos y hacer el seguimiento de tareas y tickets.

6. Caso de estudio: Proceso de implementación

6.1. Definir los objetivos de seguridad

A continuación, se detallan los objetivos principales del área de seguridad para la implementación del caso de estudio mediante la aplicación de medidas de seguridad en DevSecOps:

- Definir roles y responsabilidades para desarrollo, operaciones y seguridad.
- Crear conciencia y habilidades en el equipo para mantener prácticas de DevSecOps.
- Realizar capacitaciones en seguridad y en el uso de herramientas de DevSecOps.
- Promover la cultura de “Seguridad como código” y la colaboración entre equipos de desarrollo y seguridad.
- Crear un plan de seguridad que integre DevSecOps dentro del pipeline de CI/CD.
- Implementar herramientas de análisis de seguridad que se ejecuten automáticamente en el pipeline CI/CD.
- Asegurar que cada cambio de código pase por revisiones de seguridad automatizadas en el pipeline.
- Establecer condiciones de aprobación de código según los resultados de las pruebas de seguridad.
- Implementar alertas y reportes automáticos que notifiquen a los equipos de desarrollo de cualquier vulnerabilidad detectada.
- Implementar monitoreo de seguridad en producción para detectar amenazas en tiempo real.

Cabe destacar que en el transcurso de este capítulo se detallarán las medidas de seguridad que contribuyen al cumplimiento de los objetivos propuestos, bajo el alcance de DevSecOps. Sin embargo, no se debe perder de

vista que la seguridad integral de la solución también debe considerar la aplicación de los principios de seguridad perimetral, defensa en profundidad y *zero trust*, entre otros. Es decir, una seguridad óptima debe incluir todos los principios de seguridad descritos en la sección 1.2. de este trabajo, más allá que para el desarrollo de este capítulo y los consiguientes se haga foco exclusivamente en lo relacionado a DevSecOps.

6.2. Promover el cambio cultural

6.2.1. Modelo de responsabilidad compartida

Con el objetivo de implementar un modelo de responsabilidad compartida (sección 2.1.2. El modelo de responsabilidad compartida) se confecciona una matriz RACI que identifique responsables, aprobadores, consultados e informados y que respete las siguientes premisas:

- Otorgar un rol al equipo de seguridad sobre actividades que no son propias de seguridad y promover que el involucramiento sea en etapas tempranas de planificación y definición, por ej: diseño de funcionalidades, priorización de requerimientos funcionales y técnicos, retrospectivas.
- Involucrar equipos de desarrollo, operaciones y hasta negocio en actividades propias de seguridad, otorgándoles un rol clave en cumplimiento de estas, por ej: requerimientos regulatorios de seguridad, evaluaciones de seguridad, remediación de vulnerabilidades.

6.2.2. Capacitación, concientización y seguridad como código

A fin de aplicar principios que permitan avanzar en el camino de la codificación segura, se implementan las siguientes iniciativas:

- Capacitaciones que comuniquen los principios para el desarrollo seguro, contemplando:

- Buenas prácticas en la codificación: Control de accesos, validaciones en cliente y servidor, uso de dependencias, manejo de errores, registros de actividad, entre otros.
- Principios del desarrollo seguro: principio de mínimo privilegio, reducción de la superficie de ataque, segregación de roles, manejo de información sensible, transmisión segura de los datos, entre otros.
- Metodologías de trabajo del área de seguridad: requerimientos, red flags, diseño de arquitecturas y documentos de línea base, entre otros.
- Evaluaciones de seguridad: flujo del proceso, información, objetivos, metodología.

Elaboración y distribución de documentos de línea base de seguridad para el desarrollo seguro de servicios web, aplicaciones web, aplicaciones móviles, software en general, contemplando:

- Principios de seguridad en el diseño de aplicaciones: seguridad por defecto, principios de mínimo privilegio, reducción de la superficie de ataque, defensa en profundidad, DevSecOps.
- Principios de seguridad para la codificación de aplicaciones: validación de entradas, codificación de las salidas, autenticación, autorización, manejo de sesiones, manejo de errores, manejo de datos sensibles, cifrado y hashing, ofuscación de código, revisión de dependencias, accesos a bases de datos.
- Principios de seguridad para las pruebas de aplicaciones: mediante la realización de evaluaciones de seguridad.
- Principios de seguridad para el despliegue y el mantenimiento de las soluciones tecnológicas: gestión de vulnerabilidades, procedimientos de despliegue seguro, controles de configuraciones de ambiente.

- Explicación de los métodos de autenticación y autorización de servicios web disponibles en el mercado, recomendando esquemas de seguridad de acuerdo con la criticidad de la información y la arquitectura de la solución. Por ej: la utilización de API Keys, OAuth, basic authentication, entre otros.
- Utilización de encabezados de seguridad, utilización de cookies seguras, configuraciones de servidores web y seguridad en las interfaces para aplicaciones web.
- Utilización del almacenamiento del dispositivo móvil, utilización de los permisos, uso de redes móviles y wifi, comunicación entre procesos del sistema operativo, seguridad en los modos de ejecución de un dispositivo móvil (detección de root, debugger, hooking, emulator).

6.3. Establecer criterios para vulnerabilidades y riesgos de seguridad

6.3.1. Clasificación de riesgos

Para la clasificación de riesgos se utiliza la metodología de CVSS (*Common Vulnerability Score System*). [42] Se trata de un sistema de puntaje diseñado para proveer un método abierto y estándar que permite estimar el impacto derivado de vulnerabilidades identificadas en Tecnologías de Información, es decir, contribuye a cuantificar la severidad que pueden representar dichas vulnerabilidades. Resulta común identificar el uso de CVSS en bases de datos de vulnerabilidades públicamente conocidas como [43] *National Vulnerability Database* (NVDB) y [44] *Common Vulnerabilities and Exposures* (CVE).

6.3.2. Tipificación de riesgos

Todos los riesgos, a excepción de los defectos leves y medios, deben ser reevaluados de acuerdo con la metodología de reevaluación de riesgos. En función a su resultado, se definen las siguientes categorías:

- Defectos bloqueantes: Aquellas vulnerabilidades cuyo riesgo residual esté comprendido entre 9.0 y 10.0 de acuerdo con la categorización de riesgo CVSS.
- Defectos críticos: Aquellas vulnerabilidades cuyo riesgo residual esté comprendido entre 7.0 y 8.9 de acuerdo con la categorización de riesgo CVSS.
- Defectos medios: Aquellas vulnerabilidades cuyo riesgo residual esté comprendido entre 4.0 y 6.9 de acuerdo con la categorización de riesgo CVSS.
- Defectos leves: Aquellas vulnerabilidades cuyo riesgo residual esté comprendido entre 0.0 y 3.9 de acuerdo con la categorización de riesgo CVSS.

6.3.3. Metodología de reevaluación de riesgos

La evaluación de riesgo de una vulnerabilidad (CVSS) se basa en el peor escenario posible, es decir, el score asignado en forma genérica no tiene en cuenta el contexto particular de cada organización y las protecciones aplicadas sobre dicho activo de información.

Como parte del cumplimiento de los criterios de seguridad y la tipificación de los riesgos, es importante elaborar una evaluación del riesgo específico en el contexto particular del Banco. Es decir, tener en cuenta características de la arquitectura tecnológica como, por ejemplo, la utilización de políticas de seguridad, de firewalls, firewalls de aplicaciones (WAF), aislamiento de redes, información que intercambia el componente donde se detecta la vulnerabilidad, entre otros. Debido a esto, el riesgo residual de muchas vulnerabilidades cambia según el uso específico y el contexto de ejecución.

Por ejemplo, supongamos que se encuentra un CVE de riesgo 9.4 (Critical) en el microservicio de auditoría, con el siguiente vector CVSS [45] AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:H:

- Attack Vector (AV): Network
- Attack Complexity (AC): Low
- Privileges Required (PR): None
- User Interaction (UI): None
- Scope (S): Unchanged
- Confidentiality Impact (C): High
- Integrity Impact (I): Low
- Availability Impact (A): Low

En el contexto particular del microservicio de auditoría, un atacante debe estar al menos en la misma red local del banco para acceder a él, por lo que el Vector de Ataque (AV) se modifica a "Red adyacente" (A). Pero para estar en la red local del Banco, necesitará un conjunto mínimo de privilegios, por lo que Privilegios Requeridos (PR) se modifica al menos a "Bajo" (L). Además, la información dentro del componente son mensajes para mostrar y no tiene información de cuentas, clientes ni ningún otro tipo de información confidencial, por lo que el Impacto de Confidencialidad (C) se puede mover a “Ninguno” (N).

En base a esto, el nuevo vector es AV:A/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:L y la nueva puntuación de riesgo 6.3 (Medio).

Como se muestra en el ejemplo, en muchos casos el riesgo real de una vulnerabilidad en el contexto de ejecución específico es bastante diferente al informado originalmente en el CVE público. Por eso, en base a los resultados de las herramientas de seguridad utilizadas para la detección de vulnerabilidades, se realiza una evaluación de riesgo personalizada para cada CVE para comprender el impacto real en la plataforma y el potencial riesgo en el banco.

6.3.4. Criterios de aceptación para la puesta en producción

Para la liberación de nuevas funcionalidades o actualizaciones sobre la solución, no deben existir defectos bloqueantes o críticos (sección 6.3.2. Tipificación de riesgos), a saber:

- No deben existir vulnerabilidades de riesgo High o Critical de acuerdo con el Common Vulnerability Security Score (CVSS >= 7) detectadas en evaluaciones de seguridad.
- No deben existir vulnerabilidades del OWASP Top 10 con un riesgo residual Major, Critical o Blocker en la herramienta utilizada para análisis estático (SAST).
- No deben existir vulnerabilidades de riesgo residual High o Critical de acuerdo con el Common Vulnerability Security Score (CVSS >= 7) en la herramienta utilizada para análisis de dependencias (SCA).
- No deben existir vulnerabilidades de riesgo residual High o Critical de acuerdo con el Common Vulnerability Security Score (CVSS >= 7) en la herramienta utilizada para análisis de vulnerabilidades en contenedores.
- Se debe acordar una fecha límite de remediación para todas las vulnerabilidades que hayan requerido una reevaluación de riesgo, más allá que no implique un bloqueante para la puesta en producción.

6.3.5. Seguimiento y remediación de vulnerabilidades

Una vez dada la conformidad a la documentación para la puesta en producción por parte del equipo del proyecto (incluyendo al equipo de seguridad), se deben acordar los plazos y prioridades de remediación, teniendo en cuenta el impacto y esfuerzo de la regularización de los hallazgos. Para el seguimiento se utilizarán las herramientas colaborativas de Confluence y Jira.

El proceso de remediación implica:

- Una estimación de esfuerzo de la actualización en conjunto con los equipos de desarrollo y operaciones.

- La creación de una tarea de remediación que indique la prioridad de esta, categorizada en:
- Blocker: para los casos de riesgo reevaluado critical o high.
- Critical: para los casos de riesgo base critical y riesgo reevaluado medium o low.
- Major: para los casos de riesgo base high y riesgo reevaluado medium o low; o bien para los casos de riesgo base medium o low.
- El seguimiento de las tareas de remediación se realiza de acuerdo con los siguientes estados (labels):
- Sin_remediar: la vulnerabilidad todavía no fue tratada.
- Estimacion_de_esfuerzo: hay una asignación a un miembro del equipo de desarrollo/operaciones para realizar un análisis de esfuerzo de corrección.
- Estimado: se documenta la complejidad para la solución y el esfuerzo en horas. Se asigna al equipo de seguridad para su aprobación.
- En_proceso: la tarea se asigna a un miembro del equipo de desarrollo/operaciones para realizar el trabajo, se agrega la versión objetivo con la corrección.
- Remediado: la vulnerabilidad fue remediada.
- Riesgo_aceptado: el riesgo de la vulnerabilidad es aceptado sin necesidad de tomar acciones de remediación.

6.4. Integrar controles de seguridad en el flujo de CI/CD

6.4.1. Implementación del flujo CI/CD

A continuación, se describe el flujo de CI/CD elaborado por los equipos del proyecto para la integración y despliegue del versionado de la nueva plataforma. En la siguiente imagen se puede ver el proceso completo y luego la descripción de los pasos identificados:

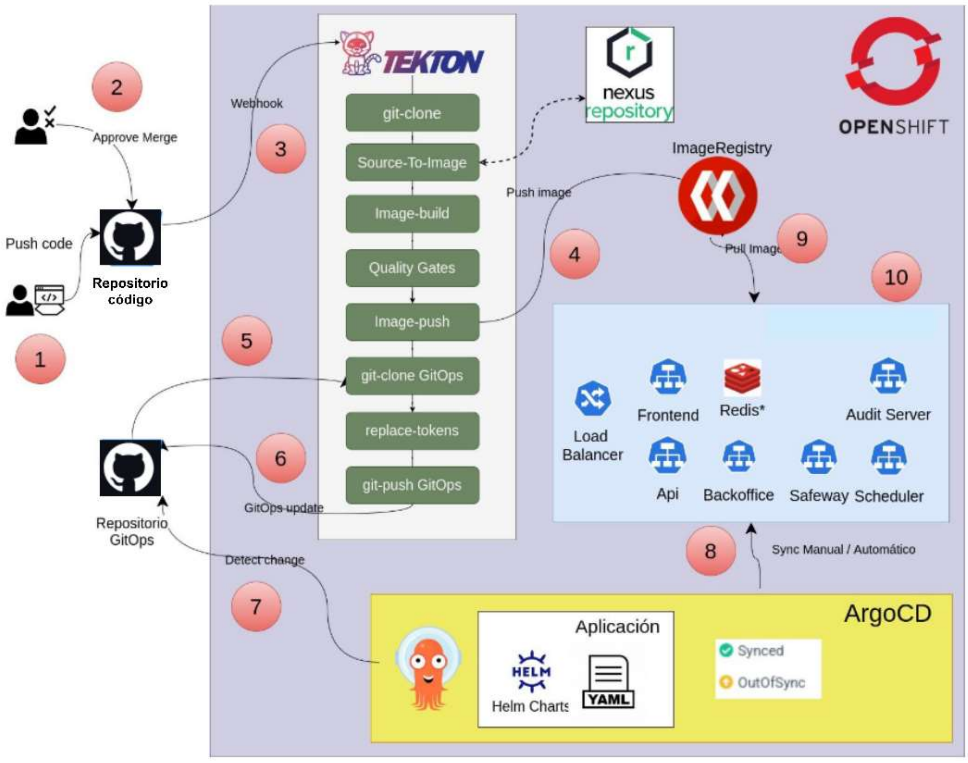

Imagen 16 - Caso de estudio: Flujo CI/CD - Fuente: Elaboración propia.

Descripción del proceso de CI / CD:

1. Un desarrollador realiza un *push* de código e inicia un *merge/pull request* para ser sujeto a revisión.
2. Una vez que los cambios fueron validados se integra el código a la rama del ambiente.
3. *Tekton* [46] (orquestador del proceso de CI) recibe una notificación de un evento en el repositorio de código e inicia el proceso de *build* del pipeline.
4. Se ejecutan las "*Quality Gates*" o pruebas de seguridad definidas. Si el código supera los criterios establecidos (sección 6.3.4. Criterios de aceptación para la puesta en producción) el proceso de pipeline pasa a hacer el *push* de la imagen generada al repositorio de imágenes del clúster de Openshift.
5. Luego de disponibilizar la imagen generada, el proceso de pipeline actualiza el repositorio de *GitOps* para reflejar el cambio.

6. Se actualiza el repositorio de *GitOps* una vez que los valores se actualizaron.

7. *ArgoCD* [47] (orquestador del proceso de CD) monitorea el repositorio de *GitOps* y una vez detectados estos cambios se muestra el estado "*Out of Sync*" hasta que pueda reflejar estas actualizaciones.

8. El sincronismo puede estar automático o manual dependiendo de la configuración.

9. Una vez desplegadas las nuevas configuraciones se procede a descargar las imágenes del repositorio correspondiente a cada componente.

10. *ArgoCD* se actualiza mostrando como nuevo estado "Synced".

6.4.2. Implementación del flujo CI/CD mobile

La siguiente imagen muestra los pasos para la implementación del flujo CI/CD de la aplicación móvil.

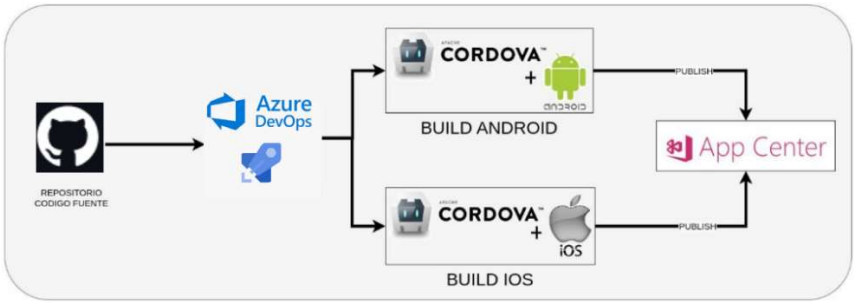

Imagen 17 - Caso de estudio: Flujo CI/CD Mobile - Fuente: Elaboración propia.

Descripción del proceso:

1. El repositorio de origen de código es el mismo que para el flujo anterior, es decir donde se encuentra el código fuente de la solución.

2. Luego se ejecuta, mediante *Azure Pipelines* de *MS Azure DevOps* [48] un pipeline para construir la versión móvil de la plataforma correspondiente.

3. El instalador de la aplicación que surge como salida del punto anterior, se publica en un repositorio de *Microsoft AppCenter* [49].

4. Una vez que se completa el circuito de homologación, se publican las aplicaciones en los *stores* oficiales de cada plataforma.

6.4.3. Despliegue mediante GitOps

Cada componente de la solución cuenta con un chart de *Helm* [50] alojado en el repositorio de *GitOps*, que incluye diversas plantillas dependiendo las necesidades de cada uno. En general existen plantillas de deployment, service, configMap, secret, volumen, entre otros. Cada uno de estos objetos válidos para un ambiente de contenedores, es configurable con los valores obtenidos de los archivos *YAML* de cada ambiente.

Todos estos objetos definen, entre otras cosas, las variables de entorno, cómo se expone el servicio en el clúster, cómo son los *pods*, las configuraciones del despliegue o la ruta de la imagen que se utiliza para esto.

La siguiente imagen es a los efectos de brindar un diagrama ilustrativo para mostrar la interacción descrita anteriormente:

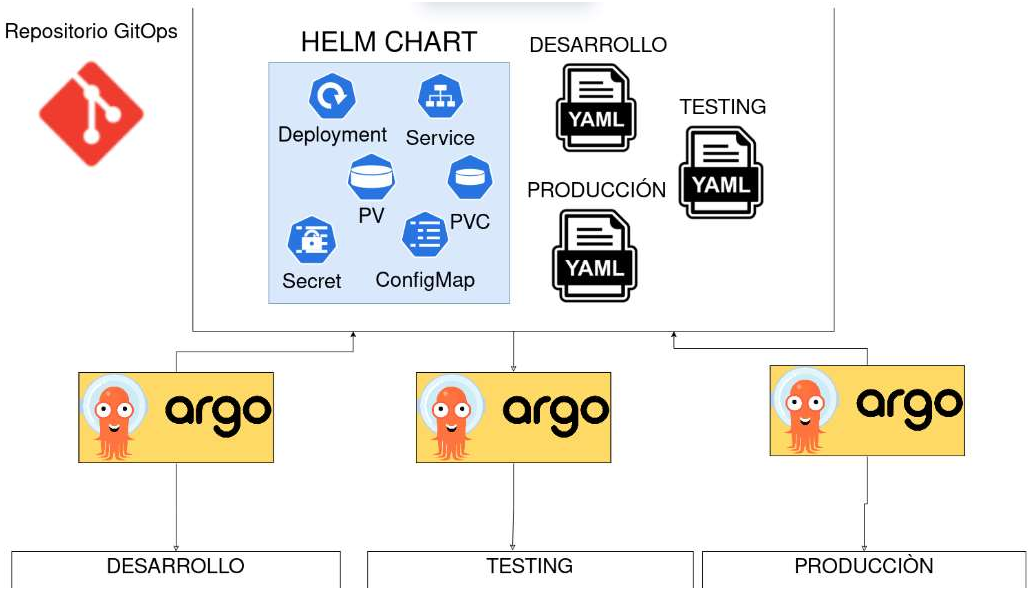

Imagen 18 - Caso de estudio: Despliegue de GitOps - Fuente: Elaboración propia.

6.5. Implementar herramientas de seguridad

6.5.1. Herramientas de seguridad en CI/CD

A partir de la “ Imagen 16” se puede visualizar la integración de herramientas de análisis automáticos de seguridad. A continuación, se muestran las pruebas incluidas en el paso 4 del flujo antes mencionado. Las mismas incluyen un análisis de tipo SAST (sección 4.3.1. Análisis estático de seguridad de aplicaciones (SAST)), SCA (sección 4.3.2. Análisis de seguridad de composición de aplicaciones (SCA)) y de vulnerabilidades en contenedores (sección 4.3.3. Análisis de vulnerabilidades en contenedores).

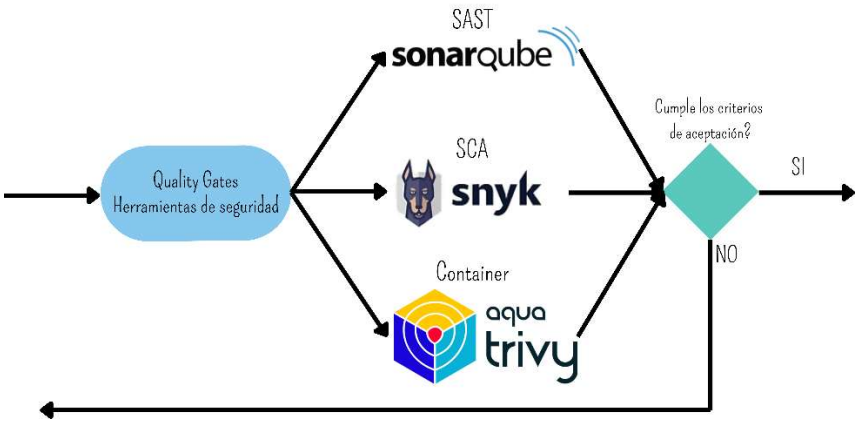

Imagen 19 - Caso de estudio: Herramientas de seguridad en CI/CD - Fuente: Elaboración propia.

En todos los casos, si alguno de los análisis realizados por las herramientas mencionadas, no cumple los criterios de aceptación para la puesta en producción (sección 6.3.4. Criterios de aceptación para la puesta en producción), el flujo no avanza y se deben corregir las vulnerabilidades detectadas para ejecutar nuevamente el proceso.

6.5.2. Herramienta de análisis SAST

Como herramienta de análisis estático de aplicaciones (sección 4.3.1. Análisis estático de seguridad de aplicaciones (SAST)) se utiliza SonarCloud.

[51] SonarCloud es una herramienta de análisis de código brindada bajo la modalidad de software como servicio (SaaS), diseñada para detectar problemas de codificación en más de 30 lenguajes, marcos y plataformas de infraestructura como código (IaC). Al integrarse directamente con el flujo de CI, el código se verifica con un amplio conjunto de reglas que cubren distintas necesidades como: problemas de mantenimiento, confiabilidad y seguridad.

A continuación, se muestra una lista de las reglas de seguridad (de tipo *security hotspot*) implementadas en los análisis [52]:

Regla	Categorías
Accesing Android external storage is security sensitive	Android, cwe, owasp, sans-top25
Allowing both safe and unsafe HTTP methods is security sensitive	cwe, owasp, sans-top25
Allowing deserialization of LDAP objects is security sensitive	cwe, owasp
Allowing requests with excessive content lenght is security sensitive	cert, cwe, owasp
Allowing user enumeration is security sensitive	cwe, owasp, spring
Authorizing non-authenticated users to use keys in the Android keystore is security sensitive	Android, cwe, owasp
Broadcasting intents is security sensitive	Android, cwe, owasp
Constructing arguments of system commands from user input is security sensitive	cwe, owasp, sans-top25
Creating cookies without the “HttpOnly” flag is security sensitive	cwe, owasp, privacy
Creating cookies without the “secure” flag is security sensitive	cwe, owasp, privacy
Delivering code in production with debug features activated is security sensitive	cwe, debug, error-handling, owasp
Disabling auto-escaping in template engines is security sensitive	cwe, owasp
Disclosing fingerprints from web application technologies is security sensitive	cwe, owasp
Disabling CSRF protections is security sensitive	cwe, owasp, sans-top25
Enabling JavaScript support for WebViews is security sensitive	cwe, owasp
Expanding archive files without controlling resource consumption is security sensitive	cert, cwe, owasp, sans-top25
Formatting SQL queries is security sensitive	bad-practice, cert, cwe, owasp, sans-top25
Hard-coded passwords are security sensitive	cert, cwe, owasp

Hard-coded secrets are security sensitive	cert, cwe
Having a permissive Cross-Origin Resource Sharing policy is security sensitive	cwe, owasp, sans-top25
Receiving intents is security sensitive	Android, cwe, owasp
Searching OS commands in path is security sensitive	cwe, owasp
Setting JavaBean properties is security sensitive	cert, cwe, owasp
Using clear-text protocols is security sensitive	cwe, owasp
Using hard-coded IP addresses is security sensitive	cert, owasp
Using long-term access keys is security sensitive	aws
Using non-standard cryptographic algorithms is security sensitive	cwe, owasp, sans-top25
Using pseudorandom number generators (PRNGs) is security sensitive	cwe, owasp
Using publicly writable directories is security sensitive	cwe, owasp
Using slow regular expressions is security sensitive	cwe, owasp, regex
Using unencrypted databases in mobile applications is security sensitive	Android, cwe, owasp
Using weak hashing algorithms is security sensitive	cwe, owasp

Tabla 3 – Caso de estudio: Reglas para análisis SAST - Fuente: [Elaboración propia].

Como se puede visualizar, las reglas están implementadas para detectar brechas de seguridad asociadas a: utilizar algoritmos de cifrado y de hasheado que no sean robustos, utilizar algoritmos de cifrado conocidos, no permitir ataques de *SQL Injection*, *CSRF*, *XSS*, no *hardcodear* contraseñas o secretos en el código, permitir *cookies* únicamente con *flags* de seguridad, no permitir la publicación de *urls* sin cifrado https, etc.

6.5.3. Herramienta de análisis SCA

Como herramienta de análisis de vulnerabilidades en dependencias y bibliotecas de terceros (sección 4.3.2. Análisis de seguridad de composición de aplicaciones (SCA)) se utiliza Snyk [53].

Snyk se integra con el flujo CI/CD como se muestra en la “Imagen 19”, pero adicionalmente y dentro del proceso de gestión y detección de vulnerabilidades continuo, se establecen escaneos proactivos con una periodicidad semanal en búsqueda de nuevas vulnerabilidades.

6.5.4. Herramienta de análisis de vulnerabilidades en contenedores

Como herramienta de análisis de vulnerabilidades en contenedores (sección 4.3.3. Análisis de vulnerabilidades en contenedores) se utiliza *Aqua Trivy* [54].

El orquestador de contenedores *RedHat Openshift* [36] ofrece únicamente imágenes de contenedores sanitizadas, sin riesgos de seguridad y actualizadas constantemente, adicionalmente, implementa un esquema que no permite que los *pods* ejecuten con el usuario *root* sino que lo hacen con usuarios de servicio con privilegios limitados.

Sin embargo, se define utilizar una herramienta Trivy ya que existen algunos componentes de la solución que son entregados directamente por el proveedor, es decir, no son desarrollados por el equipo del Banco. Esto trae como consecuencia que el proveedor entregue dichos componentes como un empaquetado con imagen incluida, motivo por el cual es recomendable efectuar un análisis de seguridad para detectar vulnerabilidades en dicha imagen, que puede o no estar sanitizada.

6.5.5. Herramienta para protección de secretos

Los secretos en el contexto de *Kubernetes* representan objetos que almacenan información confidencial, como contraseñas, tokens OAuth, y claves SSH. Constituyen una forma flexible de almacenar este tipo de información en comparación a hacerlo en la definición de un pod y de una imagen de un contenedor.

En el contexto de RedHat Openshift los secretos se encuentran almacenados con una codificación Base64, la cual no representa un cifrado ni brinda un almacenamiento seguro en reposo. Esta situación se agrava aún más,

cuando esos secretos se encuentran fuera del contexto de ejecución de la solución, es decir, en el repositorio de GitOps.

Con el objetivo de proteger los secretos (sección 4.3.4. Manejo seguro de secretos) fuera del contexto del orquestador de contenedores, se utiliza la herramienta *Sealed Secrets* [55].

Sealed Secrets se compone de dos partes: un controlador en el que se instala en el clúster de *Openshift* y una herramienta cliente llamada *kubeseal*. Esta última utiliza algoritmos de criptografía asimétrica para cifrar el secret, y "sellarlo" con la clave pública. De esta forma se asegura que el secret que queda almacenado en el repositorio de GitOps se encuentra cifrado y no en plano o codificado. Por otra parte, el controlador, que tiene la clave privada, puede descifrar y utilizar el secreto.

6.6. Realizar evaluaciones de seguridad

6.6.1. Alcance de las evaluaciones

Se realizan análisis de aplicaciones web, de aplicaciones móviles y de vulnerabilidades en infraestructura.

El alcance de las evaluaciones contempla, al menos, los siguientes puntos:

- Identificar servicios no autorizados o no homologados.
- Identificar configuraciones débiles, erróneas o faltantes.
- Identificar software o componentes vulnerables.
- Evadir mecanismos de autenticación y autorización utilizando métodos de bypass.
- Evadir mecanismos de seguridad utilizando métodos de inyección sin explotación.
- Detectar y enumerar formularios web.

Para ninguna de las pruebas de intrusión se utilizan metodologías de denegación de servicios.

6.6.2. Modalidades de las evaluaciones

Se realizan análisis de seguridad de tipo “caja blanca” a fin de evaluar los mecanismos de seguridad implementados y determinar el nivel de seguridad de la nueva plataforma realizado por el equipo de ciberseguridad del banco (secciones 2.3.3. Evaluaciones de seguridad proactivas y 4.4.4. Evaluaciones de seguridad manuales (penetration test)).

Por otra parte, para garantizar el cumplimiento normativo y alineado con la política de seguridad del Banco, se realizan evaluaciones de seguridad de tipo “caja negra” o “caja gris” contratadas como servicio y realizadas por un tercero independiente.

Las evaluaciones de seguridad se realizan proactivamente y pueden suceder con versiones de la solución que se encuentren en etapa de pruebas, release o producción, según sea la necesidad que se identifique.

6.6.3. Metodologías de las evaluaciones

Las metodologías utilizadas incluyen análisis de seguridad de tipo estático (secciones 4.3.1. Análisis estático de seguridad de aplicaciones (SAST) y 4.3.2. Análisis de seguridad de composición de aplicaciones (SCA)), dinámico (sección 4.4.1. Análisis dinámico de seguridad de aplicaciones (DAST)), interactivo (sección 4.4.2. Análisis interactivo de seguridad de aplicaciones (IAST)), de infraestructura (sección 4.4.3. Análisis de vulnerabilidades en infraestructura) y manual (sección 4.4.4. Evaluaciones de seguridad manuales (penetration test)).

Para las evaluaciones sobre aplicaciones móviles, se utilizan como base los siguientes componentes activos del OWASP Mobile Security Project [56]:

- MASVS: Consiste en un estándar de verificación de seguridad de aplicaciones móviles.
- MSTG: Consiste en una guía de pruebas de seguridad móvil. Describe los procedimientos técnicos e incluye una lista de casos de prueba para verificar los requerimientos listados en el MASVS.
- La lista de verificación de seguridad móvil.

Para las evaluaciones sobre aplicaciones y sitios web el enfoque utilizado se basa en el Manual de Metodología de Pruebas de Seguridad de Código Abierto (*OSSTMM*) [57], e incluye una combinación de pruebas automatizadas (cuando es posible) y de inspección manual (cuando sea necesario) de los servicios expuestos. El OSSTMM es un estándar ampliamente reconocido y utilizado en la industria de la seguridad informática, adicionalmente, proporciona un marco completo y estructurado para llevar a cabo pruebas de penetración y evaluación de la seguridad en sistemas, redes y aplicaciones.

OSSTMM brinda una serie de escenarios de prueba, técnicas y herramientas para realizar evaluaciones exhaustivas de seguridad.

6.6.4. Herramientas de las evaluaciones

Algunas de las herramientas utilizadas para realizar las evaluaciones de son:

- Burp Suite: Es un conjunto de herramientas para pruebas de seguridad de aplicaciones web, incluyendo escaneo de vulnerabilidades y manipulación de solicitudes y respuestas.
- Metasploit: Es un framework para pruebas de penetración que proporciona módulos de explotación y post-explotación para diversas vulnerabilidades.
- Nmap: Es una herramienta de escaneo de red utilizada para descubrir hosts y servicios, así como para evaluar la seguridad y configuración de los dispositivos conectados.
- Nessus: Es una plataforma de escaneo de seguridad que analiza vulnerabilidades en dispositivos, aplicaciones, sistemas operativos, servicios en la nube y otros recursos de red
- SQLMap: Es una herramienta para explotar y detectar vulnerabilidades de inyección SQL en aplicaciones web y bases de datos.
- Android Studio: Es un entorno de desarrollo integrado (IDE) y se utiliza para el desarrollo de aplicaciones de Android.

- Frida: Es un conjunto de herramientas que permite en tiempo de ejecución conectarse a la aplicación para realizar ingeniería inversa.
- Hopper: Es una herramienta de ingeniería inversa que permite desmontar, descompilar y depurar aplicaciones iOS
- apktool: Es una herramienta que permite realizar ingeniería inversa de archivos instaladores de Android.
- Kali: Es una distribución de Linux pensada y diseñada para la auditoría de seguridad informática, enfocada en pruebas de intrusión.
- Metasploit: Es un framework para la ejecución de scripts de explotación sobre vulnerabilidades conocidas.
- adb: Es una herramienta que permite comunicarse con un dispositivo Android a través de la línea de comandos

6.6.5. Informes de las evaluaciones

Los informes de las evaluaciones de seguridad contemplan dos tipos: técnico y ejecutivo.

El informe técnico debe incluir, al menos, la siguiente información:

- Objetivo y alcance.
- El método de clasificación de riesgos y vulnerabilidades utilizado (típicamente el descrito en la sección 6.3.1.).
- Un resumen con la enumeración de riesgos identificados y su clasificación (sección 6.3.2.).
- Los hallazgos identificados, su nivel de riesgo base, el nivel de riesgo luego de su reevaluación en el contexto del Banco y la evidencia que permita verificar la vulnerabilidad detectada.

El informe ejecutivo debe incluir, al menos, la siguiente información:

- Objetivo y alcance.

- El método de clasificación de riesgos y vulnerabilidades utilizado (típicamente el descrito en la sección 6.3.1.).
- Un resumen con la enumeración de riesgos identificados y su clasificación (sección 6.3.2.).
- El nivel de seguridad general del sistema, aplicación, infraestructura, según corresponda. Para esto se utilizan escalas que contemplen los valores:
- Satisfactorio: No hay hallazgos de riesgo crítico ni alto.
- Adecuado: No hay hallazgos de riesgo crítico, pero pueden haber de riesgo medio y/o bajo.
- Ajustado: No hay hallazgos de riesgo crítico, pero pueden haber de riesgo alto y/o medio.
- Inadecuado: Hay hallazgos de riesgo crítico y/o alto.

6.7. Monitorear el ambiente productivo

6.7.1. Centralización de eventos y alertas

Con el objetivo de centralizar los eventos de seguridad generados por la nueva plataforma, se utiliza una solución de tipo *SIEM/SOAR* (sección 4.7.3.), en particular, el producto IBM QRadar [58].

En el *SIEM* se definen reglas de control para detectar eventos anómalos y procedimientos para su tratamiento de acuerdo con la criticidad de la actividad detectada, por ejemplo: modificaciones en configuraciones de seguridad de la plataforma que se encuentren por fuera del alcance de los estándares y líneas base de seguridad definidos.

El equipo de Monitoreo y Control (sección 5.5.1.) es el encargado de recibir las notificaciones cuando se dispara una alerta y sigue el procedimiento establecido para su resolución. Este procedimiento, en general involucra otras áreas como desarrollo y operaciones, con lo cual la documentación se comparte con dichos equipos, lo que también contribuye en su involucramiento con los incidentes de seguridad.

Por último, y con el objetivo de garantizar el monitoreo de seguridad *7x24*, se contrata un servicio de *SOC* (*Security Operations Center*).

6.7.2. Proceso de gestión de vulnerabilidades

Se define un proceso de gestión de vulnerabilidades (sección 2.3.1.) que tiene tres entradas:

- La detección de vulnerabilidades durante el proceso de CI/CD, a través de las herramientas y evaluaciones de seguridad, cada vez que la plataforma implementa una nueva versión.
- La implementación de un monitoreo semanal mediante la ejecución de chequeos a demanda con las herramientas de seguridad (SAST, SCA, vulnerabilidades en contenedores) en búsqueda de nuevas vulnerabilidades.
- La implementación del monitoreo constante de fuentes que recopilan nuevas vulnerabilidades y sus características [42,43,44].

En la siguiente imagen se puede visualizar el diagrama de flujo para el descubrimiento, clasificación, evaluación y remediación de vulnerabilidades.

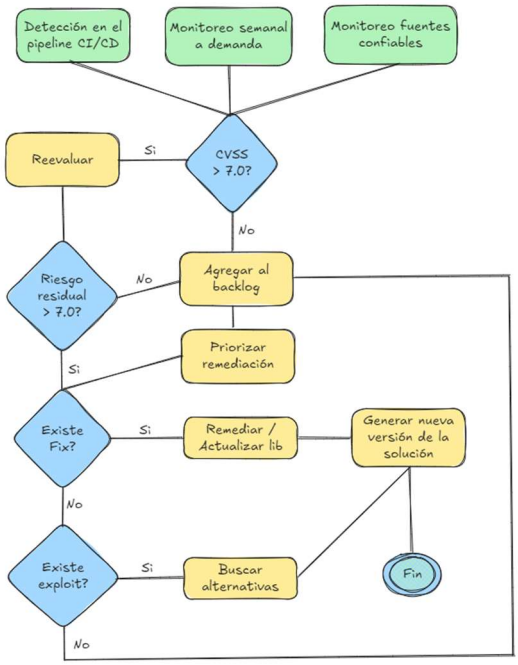

Imagen 20 - Caso de estudio: Proceso de gestión de vulnerabilidades - Fuente: Elaboración propia.

7. Caso de estudio: Resultados, lecciones aprendidas y próximos pasos

7.1. Reducción de vulnerabilidades críticas

7.1.1. Resultados

La definición de los criterios de aceptación para la puesta en producción (sección 6.3.4.) permitió garantizar que el código que llega al ambiente productivo no posea vulnerabilidades de riesgo alto o crítico (CVSS > 7.0).

Al automatizar los análisis de seguridad del código, los desarrolladores pudieron resolver problemas de seguridad en etapas tempranas, lo que, en conjunto con el cumplimiento de los criterios definidos, redujo el número de vulnerabilidades que hubieran llegado a producción.

Por otra parte, la reevaluación de aquellos riesgos cuya clasificación base sea alta o crítica (sección 6.3.3.) permitió contextualizar las vulnerabilidades en el entorno del Banco y comprenderlo con mayor profundidad. También otorgó mayor conocimiento a los equipos de desarrollo sobre la infraestructura de la organización, diferenciándose de un enfoque tradicional en el que el equipo de desarrollo es agnóstico de la infraestructura de los ambientes productivos.

En la siguiente imagen se muestra un ejemplo de detección de una vulnerabilidad bloqueante utilizada en una librería de terceros y detectada mediante la herramienta SCA. La misma a su vez es reevaluada y se recomienda la sustitución del componente ya que no existen remediaciones disponibles a la fecha del análisis.

Dependencia	CVE	Evaluación
xalan:xalan-2.7.2	CVE-2022-34169	Dado que la dependencia se utiliza para el manejo de xmls de configuración, realizamos la siguiente evaluación de riesgo de la vulnerabilidad: **AV:L** Debido a que para realizar el ataque es necesario tener acceso local al contenedor corresponde a una clasificación de Local (AV:L) **AC:L** Mantenemos la evaluación del CVE **PR:H** Se requiere como mínimo un conjunto de accesos para poder ingresar a los ambientes del sistema y permisos de administrador para modificar los archivos de configuración debido a esto la clasificación es Privileges Required High (PR:H) **UI:N** Mantenemos la evaluación del CVE **S:U** Mantenemos la evaluación del CVE **C:N** Mantenemos la evaluación del CVE **I:H** Mantenemos la evaluación del CVE **A:N** Mantenemos la evaluación del CVE A partir del nuevo vector definido (CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:H/A:H) la criticidad resultante de la vulnerabilidad en el contexto particular de Digital es: 6.7 Medium **Riesgo Residual:** Medium **Acción:** No se disponen de actualizaciones. Evaluar la sustitución del componente xalan:xalan-2.7.2.

Imagen 21 - Caso de estudio: Reevaluación de riesgos - Fuente: Elaboración propia.

7.1.2. Lecciones aprendidas

Es importante definir los criterios para vulnerabilidades y riesgos de seguridad lo más temprano posible. Esto permite dar un marco y un contexto concreto sobre el cual los equipos pueden comenzar a gestionar la seguridad de la solución. Configura expectativas, criterios, metodologías y por otra parte contribuye a generar conceptos comunes a los equipos y a dar visibilidad de las metodologías de evaluación del nivel de seguridad de una solución al resto de los equipos del proyecto (desarrollo, operaciones, negocio, entre otros).

7.1.3. Próximos pasos

Continuar trabajando en el seguimiento y hacer mayor foco en la remediación de vulnerabilidades (sección 6.3.5.).

Los próximos pasos consisten en:

- Elaborar tableros que permitan mejorar el seguimiento y tratamiento de vulnerabilidades: para dar mayor visibilidad a los equipos del proyecto.
- Mejorar los procesos de evaluación del costo de remediación y la priorización de vulnerabilidades: en general la resolución de vulnerabilidades de riesgos medio y bajo suele tener baja prioridad frente a requerimientos de negocio, e inclusive a la remediación de riesgos mayores.

- Definir tiempos de resolución: a fin de establecer deadlines para la resolución de vulnerabilidades de riesgos medios y bajos.
- Resaltar la importancia de dar tratamiento: a los riesgos medios y bajos ya que su explotación también puede comprometer los activos de información del Banco.

7.2. Reducción de tiempos de análisis de seguridad

7.2.1. Resultados

La implementación de herramientas de análisis de seguridad permitió realizar pruebas automatizadas en minutos, lo que aceleró notablemente el ciclo de vida del desarrollo del software hasta la puesta en producción.

La incorporación de herramientas SAST, SCA y contenedores en el pipeline de CI/CD permitió además realizar estas pruebas en forma automática y sin intervención manual del equipo de seguridad.

En comparación con el enfoque tradicional, esto aceleró notablemente el ciclo de pruebas de seguridad y además permitió tratarlas en una etapa más temprana.

7.2.2. Lecciones aprendidas

Es importante sumar las herramientas de seguridad al pipeline CI/CD y configurarlas con los criterios correspondientes. Pero es igual de importante la comprensión de los resultados por parte de todos los equipos: desarrollo, operaciones y seguridad.

Las herramientas de seguridad deben incorporarse progresivamente y no todas a la vez. Es tentador incorporar SAST, SCA, contenedores, DAST, IAST, MAST, etc. Sin embargo, la implementación de herramientas requiere de un proceso de madurez que se va adquiriendo con el uso y la familiarización de los equipos con la nueva tecnología.

7.2.3. Próximos pasos

Continuar el proceso de adopción de nuevas herramientas de seguridad:

- Incorporar herramientas de tipo DAST y IAST en la etapa de Pruebas del proceso CI/CD (secciones 4.4.1. Análisis dinámico de seguridad de aplicaciones (DAST) y 4.4.2. Análisis interactivo de seguridad de aplicaciones (IAST)): permitiendo su ejecución automática en el flujo y no contratándolo como un servicio.
- Explorar la solución *Advanced Cluster Security* de *Openshift* [36] para implementar políticas de seguridad y controles sobre los despliegues, como por ejemplo para reemplazar la solución de escaneo de vulnerabilidades en contendores (secciones 4.3.3. Análisis de vulnerabilidades en contenedores, 4.4.3. Análisis de vulnerabilidades en infraestructura y 4.5.2. Validación de cumplimiento normativo).

7.3. Aumento de confianza de los clientes

7.3.1. Resultados

La reducción en incidentes de seguridad ayudó a la institución a mejorar la percepción de seguridad y confiabilidad, lo que constituye un factor importante en la industria financiera.

Mediante evaluaciones de seguridad (sección 6.6.) se han detectado algunas vulnerabilidades asociadas a una mala codificación en la implementación del *OTP* (*One Time Password*) de la plataforma, también la construcción de versiones móviles sin controles de root, debug o hooking que podrían haber llegado a producción. Cabe destacar que muchos de estos defectos tienen que ver con un gap entre los requerimientos de negocio, la implementación de desarrollo y lo esperado por seguridad, con lo cual no se detectaron con las herramientas de la etapa de build sino con evaluaciones manuales realizadas por los expertos.

Si se compara con un enfoque tradicional, estos defectos se hubieran encontrado en una etapa posterior a la puesta en producción, y posiblemente, a través de algún incidente de seguridad asociado a un fraude financiero. Por el contrario, con el enfoque DevSecOps fueron detectados en la etapa de Pruebas del flujo CI/CD dando el tiempo suficiente para la resolución y la visibilidad a todos los equipos del proyecto que decidieron de forma unánime remediar estas vulnerabilidades antes de la puesta en producción.

7.3.2. Lecciones aprendidas

"La seguridad aporta valor al cliente y al negocio" es casi una premisa que los equipos de seguridad buscan defender desde hace mucho tiempo y que, con la implementación de esta metodología, logro transmitirse a los equipos de negocio y desarrollo del proyecto.

La demostración mediante casos prácticos, como, por ejemplo, como un atacante puede utilizar tu *2FA* o aprovechar un dispositivo móvil rooteado para robar tu identidad y realizar transferencias a tu nombre, sirvió enormemente para dar visibilidad en un lenguaje común y no técnico a equipos que no son técnicos.

7.3.3. Próximos pasos

Implementar un proceso de mejora continua que contemple:

- Reducir los tiempos de las evaluaciones de seguridad manuales.
- Incorporar nuevas herramientas que contribuyan a la eficiencia de las evaluaciones.
- Incorporar nuevos profesionales especializados en evaluaciones de seguridad.
- Implementar estrategias de Red Team y Blue Team (sección 2.3.3.2.).
- Mejorar los mecanismos de inteligencia de amenazas (sección 2.3.4.), incorporando a todos los equipos del proyecto en el modelado.

7.4. Reducción de costos de remediación

7.4.1. Resultados

La implementación de la seguridad y las herramientas de análisis en etapas tempranas, además del involucramiento en la toma de decisiones desde la planificación y diseño de los requerimientos funcionales (sección 6.2.1.), permitió reducir los costos de remediación de riesgos de seguridad (Imagen 3) evitando, inclusive, posibles multas regulatorias.

Adicionalmente, los desarrolladores ganaron tiempo productivo, ya que los errores críticos se abordaron automáticamente en etapas tempranas, lo cual evitó costosos retrabajos, reanálisis y recodificación.

7.4.2. Lecciones aprendidas

Adoptar principios de *Shift Left* en DevSecOps permite construir software seguro y de alta calidad desde las primeras etapas del desarrollo. Este enfoque ayuda a reducir costos, aumentar la velocidad de desarrollo, y mejorar la colaboración entre los equipos, lo que contribuye a la entrega de productos más seguros y confiables.

7.4.3. Próximos pasos

Continuar perfeccionando el enfoque incorporando herramientas de seguridad en los entornos de desarrollo (IDE) (sección 4.2.1.).

Continuar haciendo hincapié en la necesidad de la participación de los equipos de seguridad desde la etapa de planificación y diseño.

7.5. Reducción de errores de código en producción

7.5.1. Resultados

Al automatizar los análisis de seguridad del código, los desarrolladores pudieron resolver problemas de seguridad en etapas tempranas, lo que, en

conjunto con el cumplimiento de los criterios definidos, redujo el número de vulnerabilidades que hubieran llegado a producción.

Las herramientas de análisis de código permitieron detectar errores y vulnerabilidades que, en enfoques tradicionales, solo se identificaban en producción.

7.5.2. Lecciones aprendidas

Automatizar las pruebas en el pipeline garantiza que cada cambio de código sea probado exhaustivamente antes de ser desplegado. Esto evita que errores introducidos en nuevas versiones pasen desapercibidos.

7.5.3. Próximos pasos

Continuar implementando principios de mejora continua para optimizar procesos permitiendo reducir tiempos, eliminar cuellos de botella, y hacer más eficiente cada paso, mejorando la productividad.

7.6. Aumento de la velocidad de los lanzamientos

7.6.1. Resultados

Gracias a la implementación de metodologías DevSecOps, flujos de CI/CD, automatización de despliegues y controles de seguridad, los equipos del proyecto lograron acelerar significativamente el lanzamiento de nuevas versiones en comparación con un enfoque tradicional. Mientras que en el método tradicional una actualización podía tardar semanas en ser desplegada, homologada y analizada desde el punto de vista de seguridad, la adopción de DevSecOps ha reducido este tiempo a solo minutos.

7.6.2. Lecciones aprendidas

El *Time To Market* (*TTM*), o tiempo de lanzamiento al mercado, es crucial debido a la rapidez con la que evolucionan las demandas de los clientes y la alta competitividad de los mercados (sección 1.3.4.).

7.6.3. Próximos pasos

Continuar implementando principios de mejora continua para optimizar procesos permitiendo reducir tiempos, eliminar cuellos de botella, y hacer más eficiente cada paso, mejorando la productividad.

7.7. Reducción de incidentes de seguridad

7.7.1. Resultados

El monitoreo proactivo y la rápida intervención en los problemas críticos permitieron reducir la cantidad de incidentes de seguridad en producción.

La definición de un proceso de gestión de vulnerabilidades (sección 6.7.2.) que implemente un monitoreo continuo y recopile no sólo hallazgos en el CI/CD sino también de fuentes externas y de análisis a demanda, permitió tener el ambiente del Banco más controlado, lo que derivó en menor cantidad de incidentes de seguridad.

Por otra parte, la implementación de las evaluaciones de seguridad permitió detectar defectos como los mencionados en la sección 7.3.1. lo cual también redujo la cantidad y el impacto de posibles incidentes de seguridad.

7.7.2. Lecciones aprendidas

Un monitoreo proactivo permite identificar vulnerabilidades y prevenir posibles ataques antes de que se conviertan efectivamente en incidentes de seguridad. Esto ayuda a prevenir brechas de seguridad, evitando que los atacantes lleguen a comprometer los activos de información.

Al detectar y remediar amenazas en etapas tempranas, se pueden prevenir daños importantes y reducir los costos asociados a la recuperación. En contraposición, en un modelo reactivo, las brechas de seguridad suelen ser más graves y costosas de reparar contemplando el costo en la imagen del Banco y por lo tanto de la confianza de los clientes.

7.7.3. Próximos pasos

Continuar trabajando en los procesos de resolución de incidentes, ya que, si bien se redujeron a partir del monitoreo proactivo, en caso de suceder, se deben resolver de la forma más eficiente posible.

Implementar estrategias de Red Team y Blue Team (sección 2.3.3.2.) para detectar brechas de seguridad.

Implementar simulacros de ataques e incidentes de seguridad para medir la eficiencia de los equipos y mejorar los tiempos de respuesta.

7.8. Adopción de la cultura de seguridad

7.8.1. Resultados

Las capacitaciones de seguridad para desarrollo cumplieron los objetivos y sirvieron como un primer acercamiento entre las áreas. La posibilidad de que los desarrolladores conozcan como se diseña la seguridad y que se utilice un lenguaje en común, permitió establecer vínculos entre los miembros de ambos equipos.

Los documentos de lineamientos de seguridad, si bien son algo más “fríos” que las charlas de capacitación, sirvieron como fuente de consulta constante ante dudas sobre si un código que está siendo desarrollado puede ser vulnerable.

Se logró que algunos miembros de los equipos de desarrollo se interesen aún más en la seguridad, inclusive realizando consultas o recomendaciones, propuestas de mejora y alentando a sus compañeros a adoptar buenas prácticas.

7.8.2. Lecciones aprendidas

Si bien los equipos de seguridad y desarrollo, comúnmente, tienen intereses contrapuestos, es importante romper los silos y encontrar intereses en común.

Encontrar el equilibrio entre seguridad y agilidad es la clave para que ambos equipos sientan que cumplen sus objetivos. Si esto se realiza mancomunadamente será aún más productivo.

La adopción de la cultura de seguridad es un proceso, y como tal, debe continuar madurando constantemente.

7.8.3. Próximos pasos

Continuar contribuyendo a la madurez del modelo de responsabilidad compartida, la ruptura de los silos y la adopción de la cultura de seguridad implementado:

- Diseñar e implementar un programa de campeones de seguridad (sección 2.2.2.) que permita establecer recompensas a desarrolladores que cumplen un rol de seguridad en sus equipos, por ejemplo, ofreciendo capacitaciones de seguridad gratuitas.
- Analizar la posibilidad de implementar un programa de recompensas por hallazgos de alcance interno (sección 2.2.3.).
- Continuar perfeccionando el enfoque incorporando herramientas de seguridad en los entornos de desarrollo (IDE) (sección 4.2.1.).

8. Aportes personales del presente trabajo

8.1. Aprendizaje de la teoría a la práctica

La transición de la teoría a la práctica suele ser un desafío considerable. En el caso de DevSecOps, la bibliografía disponible, tanto en libros como en recursos en línea, se centra en fundamentos teóricos y ejemplos prácticos, que a menudo son ficticios. Si bien estos recursos son útiles para comprender la metodología, rara vez ofrecen guías detalladas o experiencias reales que “bajen a tierra” o faciliten su implementación en contextos reales.

Este trabajo cierra esa brecha. A través de definiciones, resultados y lecciones aprendidas aplicadas a un caso real, proporciona a los profesionales una base sólida para comenzar a implementar DevSecOps en sus organizaciones. Es decir, que utilicen este aporte como referencia inicial o línea base, evitando que los equipos tengan que partir desde cero y sin experiencia previa, permitiéndoles adoptar las definiciones propuestas o adaptarlas a sus necesidades.

Un ejemplo de este enfoque es la “gestión de vulnerabilidades”. La teoría establece la necesidad de un proceso asociado, pero rara vez detalla cómo llevarlo a cabo. Este trabajo ofrece un modelo práctico y original que incluye la clasificación y tipificación de riesgos, metodologías de reevaluación, criterios de aceptación de vulnerabilidades y un programa de seguimiento y remediación. Además, se presentan detalles concretos, como los estadios propuestos para el seguimiento de vulnerabilidades (sección 6.3.5. Seguimiento y remediación de vulnerabilidades) y el contenido de los documentos utilizados en las capacitaciones a los equipos de desarrollo (sección 6.2.2. Capacitación, concientización y seguridad como código).

De este modo, el trabajo no solo proporciona herramientas prácticas, sino que también ejemplifica cómo traducir la teoría en acciones concretas.

8.2. DevSecOps en organizaciones tradicionales

Las características de la organización y el proyecto seleccionados para el caso práctico tienen un propósito claro. El *Banco República*, como entidad tradicional, opera en silos, con un exceso de burocracia y metodologías rígidas, basadas en procesos manuales. Además, al ser un banco, la seguridad adquiere una relevancia crítica, incluso mayor que en otras industrias. El proyecto seleccionado, de gran trascendencia, representa un cambio significativo en términos tecnológicos, culturales y metodológicos.

Este trabajo presenta un caso real de éxito en la implementación de DevSecOps dentro de un contexto inicialmente adverso. Ofrece una visión integral que combina los conceptos teóricos desarrollados en los primeros cuatro capítulos con su aplicación práctica.

El aporte principal radica en proporcionar un caso completo que integra todos los elementos clave: procesos, cultura, tecnología y personas. A diferencia de los casos aislados que suelen encontrarse en la bibliografía, este estudio conecta de manera coherente todos los conceptos. Por ejemplo, se implementa una matriz de responsabilidades (RACI) y un programa de capacitación para desarrolladores, vinculándolos con el plan de seguimiento y remediación de vulnerabilidades. Esto permite al lector comprender cómo se interrelacionan los diferentes procesos y cómo ciertas decisiones impactan en otros aspectos.

Además, este trabajo demuestra que DevSecOps es aplicable en cualquier tipo de organización, incluso en aquellas tradicionales, siempre que exista el objetivo de lograr agilidad y seguridad. Presenta un caso real de éxito, detallando los pasos seguidos para su implementación, y ofrece modelos prácticos que no solo inspiran a avanzar en la transformación metodológica, incluso en condiciones iniciales desfavorables, sino que también proporcionan herramientas concretas para abordar el cambio de manera efectiva.

8.3. DevSecOps como modelo evolutivo

DevSecOps es un modelo evolutivo, especialmente en contextos como el caso real presentado.

La bibliografía disponible, como *playbooks* y *guidebooks*, suele listar todos los controles de seguridad que “deben” implementarse, pero no siempre considera que no es viable pasar de un estado inicial a uno completamente maduro de forma instantánea. Aunque se diferencian controles esenciales de aquellos deseables, cada uno requiere un nivel de madurez para ser implementado. Además, las organizaciones enfrentan limitaciones técnicas, económicas y culturales que pueden impedir la adopción de ciertos controles esenciales, mientras que otros, considerados menos prioritarios, pueden ser factibles debido a un menor costo o complejidad en su contexto particular.

Este trabajo aporta un ejemplo concreto y un modelo de adopción gradual adaptado a las posibilidades específicas del *Banco República*. Ayuda al lector a contextualizar su propia implementación considerando factores que a menudo no se abordan en la bibliografía tradicional.

Por ejemplo, en el proceso de "seguridad como código", se presentan pasos específicos para implementarlo: charlas iniciales, creación de lineamientos de seguridad, desarrollo de programas de campeones de seguridad, búsqueda de errores y, finalmente, la integración de herramientas automatizadas en el entorno de desarrollo. En el caso de estudio, las limitaciones económicas y de madurez llevaron a implementar solo los dos primeros pasos. Este modelo permite al lector adaptar el enfoque según su propio contexto, avanzando gradualmente en la medida de sus posibilidades.

Un caso similar se observa en la adopción de herramientas automatizadas de seguridad. Este trabajo propone un modelo original que prioriza controles “deseables” como los escaneos de vulnerabilidades en imágenes de contenedores, que resultaron viables por su bajo costo en el contexto del banco. Sin embargo, no se implementaron herramientas DAST, consideradas más prioritarias, debido a restricciones específicas, sugiriendo, en su lugar,

evaluaciones manuales de seguridad, para suplir temporalmente la ausencia de controles automatizados.

En resumen, este enfoque evolutivo ofrece un modelo práctico y adaptable, permitiendo a las organizaciones avanzar en la implementación de DevSecOps según sus capacidades, mientras consideran soluciones intermedias para superar limitaciones específicas.

8.4. ¿Qué hacer y qué no hacer?

En la mayoría de los casos, la bibliografía disponible sobre DevSecOps se enfoca en sugerir pasos para alcanzar el éxito, pero rara vez aborda qué evitar o cómo reaccionar si los resultados no son los esperados.

Este trabajo ofrece una experiencia integral que no solo detalla decisiones tomadas, sino también los resultados obtenidos, incluyendo aquellos que no cumplieron con las expectativas iniciales. Por ejemplo, se aborda cómo las evaluaciones de seguridad manuales, en ocasiones, generaron retrasos significativos en los despliegues a producción.

Además, el trabajo destaca las lecciones aprendidas y las oportunidades de mejora, identificando tanto éxitos como fracasos. Por ejemplo, se expone cómo la metodología inicial para la remediación de vulnerabilidades de riesgo medio y bajo no fue efectiva, lo que resultó en la acumulación de un gran *backlog* de pendientes. También se resalta la necesidad de incorporar herramientas como DAST y IAST para optimizar y acelerar las evaluaciones de seguridad manuales que generaron retrasos.

En resumen, este trabajo proporciona al lector una visión completa, indicando un camino que ha demostrado ser exitoso para el proyecto. A la vez, advierte sobre posibles obstáculos basándose en una experiencia real, permitiendo al lector anticiparse y planificar estrategias para superar esas dificultades.

Conclusiones

Históricamente, la agilidad y la seguridad han sido conceptos en conflicto: el primero, enfocado en el desarrollo rápido y flexible, y el segundo, percibido a menudo como un obstáculo que ralentiza los ciclos de desarrollo. En la introducción se planteó que, en el contexto actual, las organizaciones deben adaptarse a una dinámica de desarrollo y respuesta rápida para mantenerse competitivas. Al mismo tiempo, el crecimiento en la demanda de servicios y plataformas digitales ha aumentado la exposición a vulnerabilidades y ciberataques, exigiendo mayores niveles de seguridad.

La elección de la temática de esta tesis fue acertada, ya que aborda la necesidad contemporánea de proporcionar mayor agilidad y seguridad a las organizaciones y sus servicios. Así, se ha logrado responder una de las principales preguntas iniciales: ¿es posible ofrecer agilidad y seguridad simultáneamente? Una vez concluido el trabajo, la respuesta es: sí, mediante la adopción de DevSecOps.

Es común que los equipos técnicos asocien DevSecOps solo con la “implementación de herramientas de seguridad automatizadas en el flujo de CI/CD”; sin embargo, esta tesis ofrece un enfoque integral, abordando no solo la automatización de herramientas, sino también los principios fundamentales de la metodología, la transformación digital que implica cambios en procesos, personas y cultura, y la evolución de la seguridad hasta la llegada de DevSecOps. Esto proporciona una visión completa de la adopción de la metodología, más allá de la simple inclusión de herramientas en cada fase del ciclo de desarrollo.

Siguiendo el enfoque cíclico de mejora continua PDCA (Planificar, Hacer, Revisar, Actuar), se presenta el caso de estudio que da contexto al lector, permitiéndole planificar e implementar, para luego revisar los resultados y aprendizajes, y planificar los siguientes pasos. Además, se destaca que la implementación de controles y herramientas de seguridad debe ser gradual y evolutiva; no es viable implementar todas las herramientas de una vez sin experiencia previa, como en el caso de estudio, y esto debe ir acompañado de

recursos humanos competentes y comprometidos que evolucionen al ritmo del cambio cultural.

Los resultados expuestos en el caso de estudio respaldan que la adopción de DevSecOps no solo protege los sistemas, sino que también incrementa la eficiencia del desarrollo y reduce los costos asociados a la remediación de vulnerabilidades, entre otros. Así como se presenta como un modelo de transformación que permite que “seguridad” y “agilidad” coexistan y se potencien mutuamente, fomentando un cambio cultural hacia la colaboración y la responsabilidad compartida.

En conclusión, esta tesis ha logrado cumplir con los objetivos planteados, aportando valor al campo de estudio y ofreciendo una guía base que permite al lector llevar los conceptos teóricos de DevSecOps a la práctica mediante la implementación de un caso real en un contexto desfavorable. Además, presenta una propuesta de evolución gradual adaptada a organizaciones tradicionales, basada en la implementación de medidas concretas y el análisis de sus resultados, incluyendo tanto las experiencias exitosas como aquellas que no alcanzaron las expectativas deseadas.

Bibliografía

[1] D. Virtser, «What Is DevOps,» 22 Junio 2014. [En línea]. Available: www.quora.com/Whatis-DevOps/answer/David-Virtser. [Último acceso: 18 Ago 2024].

[2] S. Mack, Innovate ITIL: A DevOps Approach to the ITIL Framework., DZone, 2018.

[3] S. Mack, The DevSecOps Playbook: Deliver Continuous Security at Speed, Wiley, 2024.

[4] G. Kim, J. Humble, P. Debois y J. Willis, The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations, IT Revolution Press, 2016.

[5] G. Kim, K. Behr y G. Spafford, The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win, IT Revolution Press, 2013.

[6] G. Kim, «The Three Ways: The Principles Underpinning DevOps,» 22 Ago 2012. [En línea]. Available: https://itrevolution.com/the-three-ways-principles-underpinning-devops/. [Último acceso: 18 Ago 2024].

[7] J. Willis y D. Edwards, «DevOps Culture (Part 1).,» 1 May 2012. [En línea]. Available: https://itrevolution.com/articles/devops-culture-part-1. [Último acceso: 18 Ago 2024].

[8] G. Kim, The Unicorn Project: A Novel About Developers, Digital Disruption, and Thriving in the Age of Data, IT Revolution Press, 2019.

[9] Atlassian, «DevOps,» Atlassian, [En línea]. Available: https://www.atlassian.com/es/devops. [Último acceso: 25 Ago 2024].

[10] S. Rose, B. Oliver, M. Stu y S. Connelly, «Zero Trust Architecture,» 2020. [En línea]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf. [Último acceso: 25 Ago 2024].

[11] Gartner, «Integrating Security Into the DevSecOps Toolchain,» 12 Dic 2019. [En línea]. Available: https://insider.govtech.com/california/sponsored/integrating-security-into-the-devsecops-toolchain.html. [Último acceso: 25 Ago 2024].

[12] Amazon, «¿Qué es DevSecOps?,» Amazon Web Services, [En línea]. Available: https://aws.amazon.com/es/what-is/devsecops. [Último acceso: 26 Ago 2024].

[13] Snyk, «DevSecOps Overview,» Snyk, [En línea]. Available: https://snyk.io/series/devsecops/. [Último acceso: 26 Ago 2024].

[14] J. Martins, «Matriz Raci: qué es, cómo crearla con ejemplos y alternativas online,» 6 Feb 2024. [En línea]. Available: https://asana.com/es/resources/raci-chart. [Último acceso: 5 Sep 2024].

[15] D. Lehr, «Security Champion Program Success Guide,» [En línea]. Available: https://securitychampionsuccessguide.org/. [Último acceso: 5 Sep 2024].

[16] C. Cilleruelo, «¿Qué es Bug Bounty? [3 beneficios],» 18 Abr 2024. [En línea]. Available: https://keepcoding.io/blog/que-es-bug-bounty-programa/. [Último acceso: 5 Sep 2024].

[17] BugBase, «How To Handle A Bug Bounty Program Internally,» 14 Dic 2022. [En línea]. Available: https://bugbase.ai/blog/how-to-handle-a-bug-bounty-program-internally. [Último acceso: 5 Sep 2024].

[18] T. W. Kwan, «Leveraging Threat Intelligence to Proactively Mitigate Emerging Cybervulnerabilities,» 18 Oct 2023. [En línea]. Available:

https://www.isaca.org/resources/isaca-journal/issues/2023/volume-5/leveraging-threat-intelligence-to-proactively-mitigate-emerging-cybervulnerabilities. [Último acceso: 5 Sep 2024].

[19] Atlassian, «Gestión de incidentes para equipos de alta velocidad,» Atlassian, [En línea]. Available: https://www.atlassian.com/es/incident-management. [Último acceso: 8 Sep 2024].

[20] J. Cranford, «RedTeaming: How Red Team testing prepares you for cyberattacks,» 7 Jul 2023. [En línea]. Available: https://www.crowdstrike.com/en-us/cybersecurity-101/advisory-services/red-teaming/. [Último acceso: 8 Sep 2024].

[21] CERT-UK, «An introduction to threat intelligence,» 2015. [En línea]. Available: https://www.ncsc.gov.uk/files/An-introduction-to-threat-intelligence.pdf. [Último acceso: 8 Sep 2024].

[22] IBM, «What is threat intelligence?,» [En línea]. Available: https://www.ibm.com/topics/threat-intelligence. [Último acceso: 8 Sep 2024].

[23] O. Foundation, «OWASP DevSecOps Guideline,» 2023. [En línea]. Available: https://owasp.org/www-project-devsecops-guideline/. [Último acceso: 25 Sep 2024].

[24] P. DevSecOps, «Integrating Security into CI/CD Pipelines through DevSecOps Approach,» Hysn Technologies Inc., 2024. [En línea]. Available: https://www.practical-devsecops.com/wp-content/uploads/2024/06/eBook-Integrating-Security-into-CI_CD-Pipelines-through-DevSecOps-Approach-1.pdf. [Último acceso: 25 Sep 2024].

[25] G. f. geeks, «Difference between Positive Testing and Negative Testing,» 17 Oct 2020. [En línea]. Available: https://www.geeksforgeeks.org/difference-between-positive-testing-and-negative-testing/. [Último acceso: 25 Sep 2024].

[26] G. f. geeks, «Difference between Static and Dynamic Testing,» 12 Jul 2024. [En línea]. Available: https://www.geeksforgeeks.org/difference-between-static-and-dynamic-testing/ . [Último acceso: 25 Sep 2024].

[27] D. O. D. –. USA, «DevSecOps Fundamentals Guidebook: DevSecOps Tools & Activities,» 1 Mar 2021. [En línea]. Available: https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsTools-ActivitiesGuidebook.pdf. [Último acceso: 29 Sep 2024].

[28] N. –. C. Inteligente, «Alerta de Seguridad Crítica: Apache Log4j Vulnerability,» 14 Dic 2021. [En línea]. Available: https://nextvision.com/apache-log4j-vulnerability/. [Último acceso: 29 Sep 2024].

[29] GitLab, «Dynamic Application Security Testing (DAST),» [En línea]. Available: https://docs.gitlab.com/ee/user/application_security/dast/. [Último acceso: 7 Oct 2024].

[30] Jit, «Application Security Tools – How and when to use them,» [En línea]. Available: https://www.jit.io/resources/appsec-tools/. [Último acceso: 7 Oct 2024].

[31] R. Camacho, «Herramientas SAST: cómo elegir la solución adecuada,» 15 Nov 2023. [En línea]. Available: https://es.parasoft.com/blog/how-to-choose-the-right-static-application-security-testing-sast-solution/. [Último acceso: 7 Oct 2024].

[32] C. Harris, «Comparación entre la arquitectura monolítica y la arquitectura de microservicios,» Atlassian, [En línea]. Available: https://www.atlassian.com/es/microservices/microservices-architecture/microservices-vs-monolith. [Último acceso: 14 Oct 2024].

[33] S. Manjaly, «Cloud vs. On-premise: ¿qué es mejor?,» Invgate, 1 Jul 2022. [En línea]. Available: https://blog.invgate.com/es/cloud-vs-on-premise. [Último acceso: 14 Oct 2024].

[34] I. C. P. Solutions, «What is CMMI?,» [En línea]. Available: https://cmmiinstitute.com/. [Último acceso: 14 Oct 2024].

[35] whitestack, «¿Qué es el Cloud Native? Aplicaciones, características y principios,» 1 Ago 2023. [En línea]. Available: https://whitestack.com/es/blog/cloud-native/. [Último acceso: 15 Oct 2024].

[36] R. Hat, «RedHat Openshift,» [En línea]. Available: https://www.redhat.com/en/technologies/cloud-computing/openshift. [Último acceso: 17 Oct 2024].

[37] Microsoft, «Microsoft Azure,» [En línea]. Available: https://azure.microsoft.com/es-es. [Último acceso: 17 Oct 2024].

[38] GitHub, «GitHub,» [En línea]. Available: https://github.com/ . [Último acceso: 17 Oct 2024].

[39] Salesforce, «Slack,» [En línea]. Available: https://slack.com/intl/es-ar. [Último acceso: 17 Oct 2024].

[40] Microsoft, «Microsoft Teams,» [En línea]. Available: https://www.microsoft.com/es-ar/microsoft-teams/group-chat-software/. [Último acceso: 17 Oct 2024].

[41] Atlassian, «Jira & Confluence,» [En línea]. Available: https://www.atlassian.com/software. [Último acceso: 17 Oct 2024].

[42] M. Á. Mendoza, «Vulnerabilidades: ¿qué es CVSS y cómo utilizarlo?,» WeLiveSecurity by ESET, 4 Ago 2014. [En línea]. Available: https://www.welivesecurity.com/la-es/2014/08/04/vulnerabilidades-que-es-cvss-como-utilizarlo/. [Último acceso: 27 Oct 2024].

[43] NIST, «National Vulnerability Database,» [En línea]. Available: https://nvd.nist.gov/. [Último acceso: 27 Oct 2024].

[44] Mitre, «Common Vulnerabilities and Exposures,» [En línea]. Available: https://cve.mitre.org/. [Último acceso: 27 Oct 2024].

[45] NIST, «Common Vulnerability Scoring System Calculator,» [En línea]. Available: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator. [Último acceso: 27 Oct 2024].

[46] Tekton, «Tekton,» [En línea]. Available: https://tekton.dev/. [Último acceso: 29 Oct 2024].

[47] ArgoCD, «ArgoCD Project,» [En línea]. Available: https://argoproj.github.io/cd/. [Último acceso: 29 Oct 2024].

[48] Microsoft, «Microsoft Azure DevOps,» [En línea]. Available: https://azure.microsoft.com/es-es/products/devops. [Último acceso: 29 Oct 2024].

[49] Microsoft, «Microsoft Visual Studio AppCenter,» [En línea]. Available: https://appcenter.ms/. [Último acceso: 29 Oct 2024].

[50] Helm, «Helm,» [En línea]. Available: https://helm.sh/. [Último acceso: 29 Oct 2024].

[51] SonarCloud, «What is SonarCloud?,» 2024. [En línea]. Available: https://docs.sonarsource.com/sonarcloud/. [Último acceso: 30 Oct 2024].

[52] SonarCloud, «Security Related Rules,» 2024. [En línea]. Available: https://docs.sonarsource.com/sonarcloud/digging-deeper/security-related-rules/. [Último acceso: 30 Oct 2024].

[53] Snyk, «Snyk Open Source,» [En línea]. Available: https://snyk.io/product/open-source-security-management/. [Último acceso: 30 Oct 2024].

[54] Aqua, «Trivy,» [En línea]. Available: https://trivy.dev/. [Último acceso: 1 Nov 2024].

[55] B. labs, «Sealed Secrets,» [En línea]. Available: https://github.com/bitnami-labs/sealed-secrets. [Último acceso: 1 Nov 2024].

[56] O. Foundation, «OWASP Mobile Application Security,» [En línea]. Available: https://mas.owasp.org/. [Último acceso: 1 Nov 2024].

[57] Isecom, «Open Source Security Testing Methodology Manual,» [En línea]. Available: https://isecom.org/research.html#content5-9d. [Último acceso: 1 Nov 2024].

[58] IBM, «IBM QRadar,» [En línea]. Available: https://www.ibm.com/es-es/qradar. [Último acceso: 3 Nov 2024].