UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

FACULTAD DE ÁREA TÉCNICA

CARRERA DE COMPUTACIÓN Y SISTEMAS INFORMÁTICOS

Diseño e implementación de un enfoque de seguridad DevSecOps en el desarrollo de aplicaciones orientadas a microservicios

Trabajo de titulación previo a la obtención del título de:

INGENIERO EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Autor: Quichimbo Guamán Ronald Vicente

Director: Romero González, Karla Alexandra

LOJA

2022

Aprobación del director del Trabajo de Titulación

Loja, día de mes de año

Título académico completo, (no colocar siglas)
Nombres y Apellidos completos del director de la carrera
Director de la carrera de Xxxxxxxxxx
Ciudad.-

De mi consideración:

Me permito comunicar que, en calidad de director del presente Trabajo de Titulación denominado: (nombre del trabajo) realizado por Nombres y Apellidos completos del autor o autores (as) ha sido orientado y revisado durante su ejecución, así mismo ha sido verificado a través de la herramienta de similitud académica institucional, y cuenta con un porcentaje de coincidencia aceptable. En virtud de ello, y por considerar que el mismo cumple con todos los parámetros establecidos por la Universidad, doy mi aprobación a fin de continuar con el proceso académico correspondiente.

Particular que comunico para los fines pertinentes.

Atentamente,

Director: Nombres y Apellidos completos del Director del Trabajo de Titulación y título académico.
C.I.:
Correo electrónico:

Declaración de AUTORÍA y cesión de derechos

Yo, Nombres y Apellidos completos, declaro y acepto en forma expresa lo siguiente:

Ser autor (a) del Trabajo de Titulación denominado: ……, de la carrera de……., específicamente de los contenidos comprendidos en: (se debe colocar los nombres de los capítulos elaborados en el Trabajo de Titulación), siendo (nombres y apellidos completos), director (a) del presente trabajo; también declaro que la presente investigación no vulnera derechos de terceros ni utiliza fraudulentamente obras preexistentes. Además, ratifico que las ideas, criterios, opiniones, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad.

Eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones judiciales o administrativas, en relación a la propiedad intelectual de este trabajo.

Que la presente obra, producto de mis actividades académicas y de investigación, forma parte del patrimonio de la Universidad Técnica Particular de Loja, de conformidad con el artículo 20, literal j), de la Ley Orgánica de Educación Superior; y, artículo 91 del Estatuto Orgánico de la UTPL, que establece: “Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado que se realicen a través, o con el apoyo financiero, académico o institucional (operativo) de la Universidad”, en tal virtud, cedo a favor de la Universidad Técnica Particular de Loja la titularidad de los derechos patrimoniales que me corresponden en calidad de autor/a, de forma incondicional, completa, exclusiva y por todo el tiempo de su vigencia.

La Universidad Técnica Particular de Loja queda facultada para ingresar el presente trabajo al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública, en cumplimiento del artículo 144 de la Ley Orgánica de Educación Superior.

.....

Autor: Nombres y Apellidos completos del autor

C.I.:

Correo electrónico:

Dedicatoria

Xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxxxxxxxx xxxxxxx xxxxxxx xxxxxx xxxxxx xxxxxxxx

Agradecimiento

Xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxxxxxxxx xxxxxxx xxxxxxx xxxxxx xxxxxx xxxxxxxx xxxxxxxxxx xxxxxxxxx x

Índice de contenido

Aquí se debe hacer constar la paginación respectiva de los capítulos, temas y subtemas desarrollados, así como incluir índice de tablas y figuras.

Índice de tablas

Índice de figuras

Resumen

El resumen se presentará en un único párrafo con un máximo de **180 palabras**, sintetiza el aporte que brinda el trabajo realizado. **Obligatoriamente** debe contener las palabras clave **(máximo tres)**.

Ejemplo:

Xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxxxxxxxx xxxxxxx xxxxxxx xxxxxx xxxxxx xxxxxxxx xxxxxxxx xxxxxxxxx xxxxxx xxxxx xxxxx xxxxxxxxxx xxxxxxx xxxxxxxxxxxxxxxx xxxxxxxxxxxxxx.(2009)(2009; et al., 2011)

Palabras clave: xxxxxxxxx, xxxxxxxx, xxxxxx

Abstract

Abstract es el resumen traducido al idioma inglés en donde se incluyen las palabras claves. **Obligatoriamente** debe contener las palabras claves **(máximo tres)**.

Ejemplo:

Xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxxxxxxxx xxxxxxx xxxxxxx xxxxxx xxxxxx xxxxxxxx xxxxxxxxxx

Introducción

Se sugiere presentar en máximo **dos páginas** y considerar los siguientes puntos:

Cómo dio respuesta al problema planteado,

El alcance de los objetivos y su cumplimiento,

Las facilidades u oportunidades, los inconvenientes o limitantes con los que se enfrentó en el desarrollo del trabajo,

La metodología utilizada, (et al., 2019)(1990; et al., 2010, 2015, 2019)

Una breve explicación de los capítulos,

La importancia que tiene la investigación para la institución, empresa o usuarios y la sociedad en general.

Letra Arial N° 11, con sangría (1.27 cm), interlineado doble y justificado

Capítulo uno

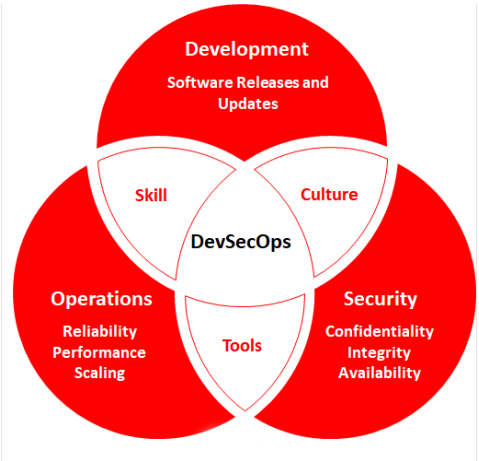
Marco Teórico

1.1 ¿Qué es DevSecOps?

Es un enfoque encargado de implementar la seguridad desde el comienzo de la creación de una aplicación evitando ataques de hacking. De este modo se combina la parte de desarrollo + seguridad + operaciones, en un ciclo de entrega continuo y automatizado, con el objetivo que todos sean responsables de la seguridad. DevSecOps integra la seguridad a través de herramientas automatizas con un conjunto de estándares de comportamientos, conocimientos y hábitos (Shrivastava & Services, 2019). Con llevan a una cultura de seguridad como código (figura 1). El propósito es ayudar a mejorar el tiempo de entrega y enviar automáticamente el código a producción cumpliendo los objetivos de seguridad.

Figura 1

Que es DevSecOps



Nota. Adapto de (Shrivastava & Services, 2019)

El enfoque DevSecOps surge de la principal interrogante que presentaba DevOps, de cómo controlar la presencia de vulnerabilidades en sus equipos evitando afectar la calidad de la seguridad del software. Se debe conocer que DevSecOps no es una idea o conceptos diferentes de DevOps (Santana et al., 2021). Si no una extensión de la filosofía de DevOps integrando la seguridad desde el inicio del ciclo de vida de desarrollo de software (SDLC[1]).

1 https://bit.ly/3NTzju2

Figura 2

Implementación de DevSecOps

Nota. Adaptado de (Sentrio, 2021)

El objetivo principal DevSecOps es automatizar, monitorear y aplicar la seguridad al momento implementar (Sentrio, 2021) en todas las fases del SDLC DevOps. Es decir: planificar, desarrollar, construir, probar, lanzar, entregar, implementar, operar y monitorear (figura 2). En función de lo planteado se utiliza la arquitectura de microservicios, la integración continua y despliegue continuo para poder desarrollar el enfoque apoyado en la forma de un pipeline[2]. Esto permite realizar una variedad de pruebas y validaciones de seguridad automatizadas, sin requerir el trabajo manual de un operador humano. Estas pueden variar de una aplicación a otra según las necesidades de la empresa, el proceso de entrega de software, la madurez y el nivel de automatización.

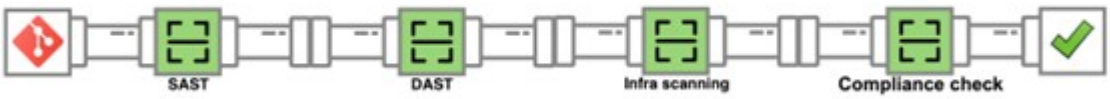
Uno de los problemas más comunes que se puede encontrar para poder implementar DevSecOps, es la velocidad porque es el elemento diferencial de DevOps. En consecuencia la integración de la seguridad no debe afectar al proceso si no acoplarse al ciclo del SDLC. Para añadir la seguridad (figura 2), hay diferentes formas como marcos o guías que recomiendan pasos a realizar como: “La Guía OWASP DevSecOps” (figura 3) que se centra en implementar un pipeline seguro utilizando las mejores prácticas e

2 Es un conjunto de procesos y herramientas automatizados que permite a los desarrolladores y profesionales de operaciones colaborar en la creación e implementación de código en un entorno de producción.(Hall Tom, 2022)

introduciendo herramientas desde la fase de desarrollo(Owasp, 2021). Además, trata de ayudar a promover la cultura de la seguridad en el proceso de desarrollo.

Figura 3

Guía del proceso de implementación de DevSecOps



Nota. Adaptado de (Owasp, 2021).

Estos procesos se realiza de acuerdo al ciclo de vida de desarrollo de software utilizando CI / CD para automatizar el desarrollo, la prueba, y el despliegue de aplicaciones. Sin embargo, cuando se utiliza herramientas de CI / CD para proporcionar automatización. Se debe colocar controles de seguridad en el software de construcción, implementación y automatización.

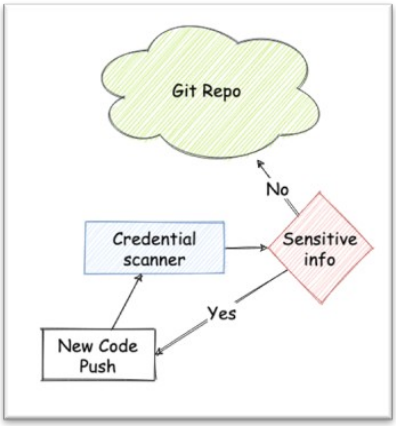
Detección de secretos .

Al momento de desarrollar se debe asegurar que la información confidencial no se envíe a la ubicación donde se almacena todo el proyecto (GitHub, GitLab, etc). Por la razón que se puede ser visible en el historial del repositorio de dicho alojamiento de código ocasionado diversos problemas.

Siguiendo el proceso de detección de secretos (figura 4) se debe escanear los commits y el repositorio para detectar cualquier información sensible como contraseña, clave secreta, confidencial, etc.

Figura 4

Proceso de detección de secretos



Nota. Adaptado de (Owasp, 2021)

Para esto se debe escanear el repositorio en busca de información confidencial y luego eliminarla; tenga en cuenta que cuando se filtra una credencial, ya está comprometida y debe invalidarse. Para detectar los secretos se debe verificar en varios lugares como:

- Detectar secretos existentes mediante la búsqueda en un repositorio de secretos existentes.
- Usar pre-commit[3] para reconocer problemas simples antes de enviarlos a revisión de código.
- Detectando secretos en el pipeline.

Cabe destacar que la mejor ubicación para detectar es pre-commit porque al momento que enviá el código al repositorio se intercepta para realizar sus respectivas pruebas, alertando al desarrollador del posible riesgo si encuentra. Parra implementar este proceso se utiliza herramientas que escanean automáticamente los repositorios en busca de información confidencial implementado directamente en el pipeline con la ventaja de ser repetibles y eficientes cada vez que se hace un commit como:

- Gittyleaks[4] .- Busca información sensible en un repositorio de git.

3 https://bit.ly/3rbILiu

4 https://bit.ly/3D74o7F

- Git-secrets[5].- Impide que se envié secretos y credenciales a los repositorios de git.
- TruffleHog[6]. - Busca secretos en los repositorios de git, profundizando en el historial de confirmaciones y las ramas. Esto es eficaz para encontrar secretos cometidos accidentalmente.
- Repo-supervisor[7].- Es una herramienta que ayuda a detectar secretos y contraseñas en el código.
- Git Hound[8].- Es un complemento de Git donde ayuda a evitar que se confirmen datos confidenciales.

SAST

Las pruebas estáticas de seguridad de las aplicaciones (SAST) comprueban los puntos débiles de seguridad en los pipelines de CI . Su funcionamiento es analizar el código dado en un entorno sin tiempo de ejecución en busca de errores y rutas que conduzcan a debilidades o riesgos de seguridad. Esto permite a los desarrolladores encontrar vulnerabilidades de seguridad en el código fuente de la aplicación en una fase más temprana del ciclo de vida de desarrollo del software. Esto garantiza la conformidad con las normas de codificación antes de ejecutar los códigos (Akujobi, 2021)

Gracias a los hallazgos de vulnerabilidades en una fase temprana, los desarrolladores pueden resolver los problemas de seguridad antes que cause un daño real. El escaneo estático es una buena manera de encontrar problemas de codificación como: violaciones de sintaxis, vulnerabilidades de seguridad, errores de programación, codificación de violaciones estándar y valores indefinidos. Por lo tanto, los desarrolladores utilizan las herramientas SAST en los pipelines como:

5 https://bit.ly/3n163WW
6 https://bit.ly/30efZU5
7 https://bit.ly/3ogwasi
8 https://bit.ly/3wxuq1s

- **SonarQube**[9].- Es una herramienta de revisión automática de código para detectar errores, vulnerabilidades y problemas en el código. Puede integrarse con su flujo de trabajo actual para permitir la inspección continua del código.
- **Clair**[10].- Es un software de código abierto para el análisis estático de vulnerabilidades en contenedores de aplicaciones (actualmente incluye OCI y Docker).
- **Veracode**[11].- Es una herramienta de análisis estático que ofrece una evaluación automatizada bajo demanda del código base de su aplicación. Simplemente envíe su código a su plataforma en línea y recibirá un plan de reparación con resultados detallados de las vulnerabilidades y fallas dentro de la aplicación o dentro del código de terceros que contiene.
- **CodeSweep**[12].- Es una extensión que permite a los desarrolladores verificar su código en busca de vulnerabilidades en cada extracción. La extensión está configurada para ejecutarse como una acción de GitHub y devuelve las vulnerabilidades identificadas en el código modificado.

DAST

Las pruebas de seguridad de aplicaciones dinámicas (DAST) se centran en probar una aplicación en ejecución a través de pruebas exteriores como: inyecciones de SQL o scripts entre sitios (XSS), etc. Con la finalidad de enviar solicitudes maliciosas para verificar su correcto funcionamiento. Las herramientas DAST son especialmente útil para detectar la validación de entrada o salida, problemas de autenticación y errores de configuración del servidor.De esta manera se utiliza herramientas para examinar el software sin necesidad de conocer el código fuente y con el mínimo requerimiento de interaccione del usuario para su configuración como:

9 https://bit.ly/3qCM4jD
10 https://bit.ly/3kwC2MQ
11 https://bit.ly/3oojsHV
12 https://bit.ly/3qznJem

- Zed attack proxy (Owasp zap)[13].- es una herramienta de código abierto que ofrece OWASP para realizar pruebas de seguridad.
- StackHawk[14].- Facilita a los desarrolladores la búsqueda, clasificación y corrección de errores de seguridad de las aplicaciones. Escanea el código de la aplicación en busca de errores, clasificando y corrigiendo, permitiendo acoplarse para automatice su canalización para evitar que errores futuros afecten a producción.

Escaneo de Infraestructura.

La infraestructura es un punto clave al momento de desplegar una aplicación en producción porque contiene todo lo necesario para ejecutar la aplicación. Sin embargo esto puede afectar a la seguridad si no está configurada correctamente así sea que el código este seguro. El software moderno se basa en el uso de la automatización tanto a nivel de desarrollo como de operaciones gracias a la adopción de DevOps. Para impulsa dicha automatización se utiliza la práctica de Infraestructura como Código (IaC) que se encarga de configurar y automatizar la administración de la infraestructura (contenedores, redes, balanceo de cargar, etc), permitiendo gestionar y preparar la infraestructura con código, en vez de hacerlo mediante procesos manuales.

Por lo tanto, para preparar la automatizar de la infraestructura con la IaC involucra a no tener que preparar ni gestionar manualmente los servidores, los sistemas operativos, el almacenamiento ni ningún otro elemento de la infraestructura los desarrolladores cada vez que se crea un nuevo o una corrección de código. De este modo hay herramientas que permiten realizar este proceso como:

- Chef[15] .- Es una herramienta de automatización que convierte la infraestructura en código administrando y configurar múltiples sistemas con facilidad.

13 https://bit.ly/3HhmWEK
14 https://bit.ly/3niUHh0
15 https://bit.ly/3FnXJXC

- Red Hat Ansible Automation Platform[16].- Es una plataforma para diseñar y gestionar la automatización de la TI según sus necesidades.
- Terraform[17].-Es una herramienta que proporciona un flujo de trabajo CLI coherente para gestionar cientos de servicios en la nube.
- AWS CloudFormation[18].- Es un servicio que brinda a desarrolladores una manera sencilla de crear una colección de recursos de AWS y de terceros.

Comprobación del cumplimiento

Devsecops promueve la adopción de la comprobación del cumplimiento o compliance check ayudando a definir los requisitos de cumplimiento de una manera que los humanos y las máquinas puedan leer fácilmente. Esto permite al personal de SecOps desarrollar políticas de cumplimiento como código, sin necesidad de utilizar lenguajes de programación técnicos. Estas políticas de cumplimiento se pueden almacenar en un sistema de control de versiones de código fuente como Git para un monitoreo continuo del cumplimiento durante la fase de desarrollo para poder obtener comentarios en tiempo real sobre el comportamiento del código utilizando herramientas de monitoreo de cumplimiento.

Chef InSpec[19] es una de ellas que convierte los marcos de cumplimiento y las políticas de TI en pruebas automatizadas incorporando en los pipelines. Esto permite a los desarrolladores evaluar de manera proactiva la revisión del código para verificar el funcionamiento de la aplicación o servicio si se está acoplando al cumplimiento de la empresa o políticas de TI (Spiros Psarris, 2020).

Todos los pasos de la guía de Owasp se debe añadir a un ciclo de vida de DevOps (figura 2). Implica fases repetitivas que representan las capacidades, procesos y técnicas cruciales para el desarrollo (Spring et al., 2021). Cada ciclo encuentra una colaboración separada pero constante de equipos y comunicación para garantizar la velocidad, la

16 https://red.ht/3ckdnGY
17 https://bit.ly/3Fe83Bo
18 https://go.aws/3DjzAAK
19 https://bit.ly/3qMKiwk

alineación y la calidad del proceso por ende se indica la incorporando de los procesos de seguridad en cada fase (Soni, 2017; Verona, 2018) .

Plan

Es la fase donde se determina los requerimientos, la arquitectura de software, valores empresariales entre otros más, con el fin de guiar al equipo a obtener los objetivos planteados dividiéndolas en tareas pequeñas para un desarrollo continuo. Cabe destacar que normalmente se utiliza metodologías de desarrollo de software como : scrum, kanban y agile con el fin de agilizar los procesos. Además se utiliza herramientas para realizar el seguimientos de los problemas y llevar de la mano con la gestión de proyectos como: jira, trello, asana y git. A si mismo se debe seleccionar una arquitectura acoplada para mejorar los procesos en DevOps porque se divide en servicios separados. Es por eso que la arquitectura de microservicios está hecha a medida para DevOps con su enfoque basado en servicios que permite a las organizaciones dividir la aplicación en servicios más pequeños.

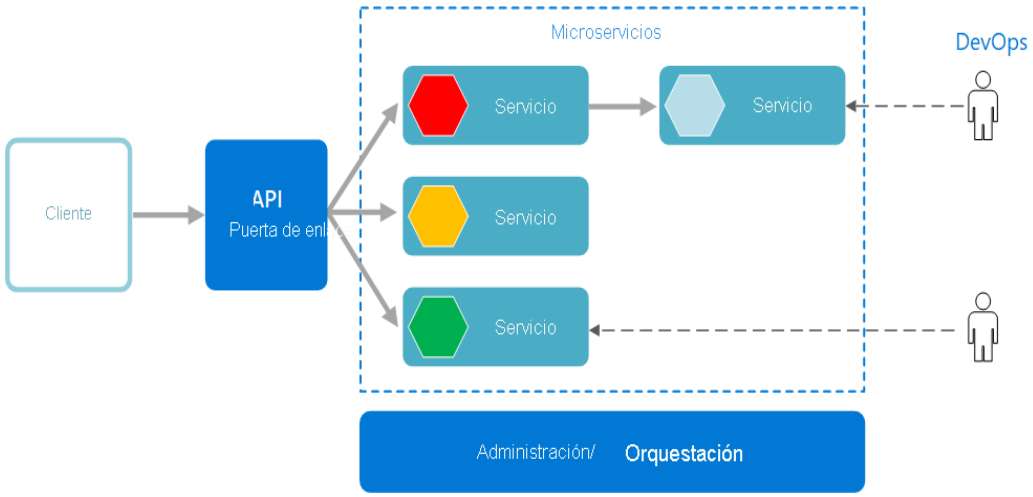
El termino Arquitectura de Microservicios conocido por sus siglas en inglés MSA (Micro Services Architecture) son servicios independientes que cumple una función específica de la función general del sistema (Amazon Web Services, 2020; Rodríguez et al., 2019). Con base en la figura 5 la MSA consta de varios componentes para su creación como: los servicios o llamados contendedores que son sistemas operativos livianos dentro de un sistema host, ejecutando instrucciones nativas del procesador central. Ofrecen ahorros en el consumo de recursos sin la sobrecarga de virtualización generando menos presión en los contenedores que en la estructura tradicional donde se ejecuta como un proceso nativo. Herramientas como Docker son proyectos de código abierto donde permite la creación de contenedores actuando como una forma ligera y portátil para empaquetar aplicaciones (Kratzke, 2017; Moghaddam et al., 2020).

En base a esto cada microservicios brindando su servicio a través de una API, lo cual es el medio de comunicación estableciendo un conjunto de reglas y protocolos para desarrollar e integrar el software. Esto permitiendo que sus servicios se comuniquen con otros, sin necesidad de saber cómo esta implementado(Microsoft, 2019; Readhat, 2017) . Hay diferentes formas de crear APIS como: Soap (Protocolo de Acceso a Objetos Simples) o Rest (Transferencia de Estado Representacional) que incrementan los procesos de transformación reduciendo los costos de mantenimiento y abren nuevas oportunidades.

Conforme crece la complejidad de un sistema con arquitectura de microservicios aumenta también el número de contenedores y APIS que se requieren ejecutar paralelamente, ya sea por la cantidad de microservicios que conforman o por necesidades de replicación (Ward, 2018). Manejar manualmente el ciclo de vida de un alto número es completamente ineficiente. La escalabilidad y alta disponibilidad se convierte en una tarea desafiante en estos escenarios. Razón por la cual se emplean Api Gateway o puerta de enlace que actúa como intermediario gestionando la comunicación entre el cliente y los servicios. De la misma forma herramientas de orquestación de contenedores que se encargan de administrar y manejar el ciclo de vida de los contenedores como Minikube, AWS EKS, Docker Swarm, pero la más utiliza es Kubernetes. Siendo una plataforma desarrollada por Google de código abierto donde automatiza las operaciones de los contenedores eliminando los procesos manuales involucrados en la implementación y escalabilidad.

Figura 5

Componentes de la arquitectura de microservicios



Nota.- Adaptado de (Microsoft, 2019)

Obteniendo la selección de los elementos para la creación del proyecto sean nuevas características o correcciones de errores, traerán consigo un nivel de riesgo de seguridad que debe ser abordado en la sesión de planificación y discutido dentro del equipo antes de proceder a desarrollarlos garantizando el concepto de desplazar la seguridad hacia la izquierda mencionado anteriormente. Hay diferentes procesos que se debe tomar para poder controlar el riesgo como:

- **Requerimientos de seguridad**

Son requisitos no funcionales que capturan y mitigan las amenazas de seguridad en el proyecto para preservar la confiabilidad, integridad, disponibilidad, autenticación y autorización. Hay diferentes formas de implementar este proceso como, las técnicas de Attack Tree y Attack Pattern (Ahmed, 2019). En la figura 6 se muestra un ejemplo de requisitos de seguridad donde deben ser explícitos, completos, concisos, comprensibles e inequívocos .

Figura 6

Ejemplo de requerimiento de seguridad

Requisito de seguridad – Caso de abuso 053	
Título	Como agente de amenaza interno, se intenta introducir en el sistema archivos XML maliciosos, entonces se controla validación de entrada de datos y se produce una devolución de error.
¿Quien?	Agente de amenaza con posibilidad de uso de la funcionalidad de ingesta de datos.
¿Cuándo?	Durante el uso de la funcionalidad de ingesta de datos.
¿Donde?	http://www.contoso.com/es/Ingesta_XML.jsp
Objetivo inmediato	Conseguir introducir datos maliciosos al sistema.
Objetivo final	Producir un ataque de inyección. Por ejemplo un ataque de tipo inyección SQL que permita leer y modificar los datos de la base de datos del entorno de producción.
Contexto/precondiciones	Se presupone que el agente de amenaza ya ha comprometido el sistema, hasta el punto de que puede realizar uso de la funcionalidad de ingesta de datos (en este caso de abuso se abstrae del control de autenticación).
Respuesta/postcondiciones	Esta entrada de datos es validada en cuanto a tamaño y forma. Impidiendo en este control de forma, conjuntos de caracteres especiales que puedan causar un incidente de seguridad.
Resultado esperado	Mensaje de error descriptivo sobre el campo invalido. Valorar el registro de esta actividad como evento de seguridad en determinados casos.
Comentarios	Criminal de la computación. – Alteración no autorizada de los datos – Acto fraudulento (por ejemplo, repetición, personificación, interceptación).
Elementos relacionados	Historia de usuario 048 HU – Ingesta de datos en formato XMLStride – Tampering o manipulación no autorizada
Cumple resultado esperado:	[ ] Sí – [ ] No

Nota.- Adaptado de (auditoriadecodigo, 2021)

Los requisitos de seguridad permite comprender mejor el riesgo que introduce una nueva función indicando explícitamente qué flujos de procesos, datos o servicios se modificarán o alterarán.

- **Clasificación de seguridad**

Es un enfoque para clasificar el nivel de riesgo de seguridad que proviene del hecho de que no todas las funciones o cambios introducirán un alto riesgo. Por lo que es ineficiente realizar todas las pruebas de seguridad y los controles de seguridad para todas las funciones o cambios. Esto ahorrará tiempo y eliminará las discusiones sobre las prioridades. Para poder implementar la clasificación de los riesgos, puede la organización realiza su propia forma de clasificar el riesgo o utilizar diferentes modelos o métodos. Uno de ellos es el modelo de Owasp (Williams, 2022). que se basa en clasificar el riesgo meditan la formula “Riesgo = Probabilidad * Impacto” para cada características o requerimientos del proyecto, esto se clasifica mediante los siguientes pasos :

- Paso 1: Identificación de un riesgo
- Paso 2: Factores para estimar la probabilidad.
- Paso 3: Factores para estimar el impacto
- Paso 4: Determinación de la gravedad del riesgo
- Paso 5: Decidir qué arreglar
- Paso 6: Personalización del modelo de calificación de riesgos

El resultado final es una comprensión completa de las implicaciones de seguridad de cada elemento y qué acciones tomar para mitigar estas implicaciones.

- **Code**

En función de lo propuesto en la planificación esta fase se dedica al comienzo de la creación del código de dicho software según lo requisitos planteados en pequeños procesos (Colliander & Colliander, 2022). Para poder tener un mayor control del código se necesitas un lugar para almacenar y gestionar permitiendo a todo el equipo compartir el mismo almacenamiento central para sus diferentes tipos de código. Hay muchas formas que se pueden implementar como GitHub, GitLab o Bitbucket mejorando el flujo de trabajo, la colaboración y proporcionar una buena disponibilidad. En esta fase se comienza a implementar la guiá de Owasp antes mencionada con la detección de secretos para que los equipos garanticen la seguridad. Se puede aplicar también la validación de entrada y bibliotecas seguras (Colliander & Colliander, 2022).

• **Build**

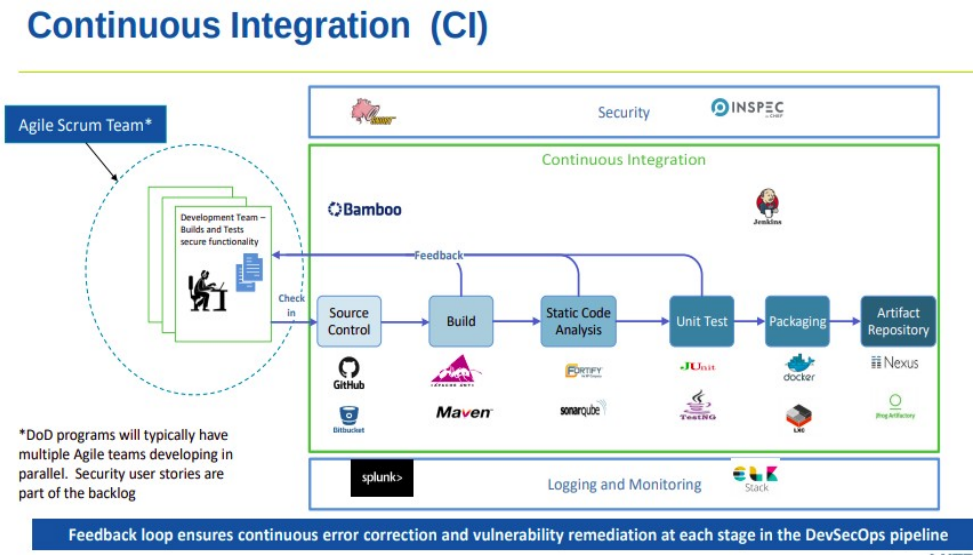
El objetivo de esta fase es construir la aplicación mediante la integración automatizada del código que se va creando en la fase de codificación . DevSecOps es un proceso de desarrollo e implementación ágil y automatizado que utiliza pipelines de integración continua (CI) y despliegue continuo (CD). Son flujos de trabajo que llevan el código fuente del desarrollador a través de varias etapa con la ayuda de herramientas automatizadas desde la fase de construcción, implementación y finalmente a la fase de tiempo de ejecución/producción como: GitHub Actions ,y Gitlab-CI. que permite introducir la integración continua de forma muy sencilla basada en Git. La ejecución de los trabajos se

realizan cada vez que hay un nuevo “commit” mediante la agregación de un único archivo con formato YAML en el proyecto .(Kinsman et al., 2021).

En vista que puede haber muchos desarrolladores trabajando en el código base, cada uno con versiones ligeramente diferentes, es importante ver si todos los cambios diferentes funcionan juntos correctamente. Es ahí donde se incorpora la integración continua (figura 8), con el objetivo de detectar los posibles problemas que pueda tener el software e integrando todo el código del proyecto cuando ya no contenga ningún problema , todo este proceso se hace de forma automática.

Figura 7

Proceso de integración continua CI



Nota. Adaptado por, (Dhawan & Sabetto, 2019)

En esta parte se utiliza las herramientas de seguridad SAS antes mencionadas para realizar prueba y detectar posibles vulnerabilidades de imagen en los contenedores, imágenes de fuentes no confiables, aplicaciones de terceros, inyección Sql, pérdida de autenticación, exposición de datos sensible, configuración incorrecta de seguridad entre otros más (figura 8). De esta manera se garantiza el código seguro si pasa estas pruebas o si se encuentran errores de codificación o vulnerabilidades de seguridad graves tiene la oportunidad el desarrollador de solucionar los problemas más fácilmente porque la revisión

es pequeña pudiendo averiguar dónde radica el problema antes de integrar su código con los demás, esto se hace cada vez que se detecta un cambio en el código base. Además, detectar las vulnerabilidades incluso antes de pasar a la fase de prueba tiene un gran beneficio. Alivia el riesgo de que se encuentren muchas vulnerabilidades graves en la fase de prueba.

Test

En esta fase se ejecutan pruebas para verificar el correcto funcionamiento de todas las características del proyecto tanto automáticas como manuales y otras comprobaciones de seguridad en función de los cambios realizados en el software. Si alguna de las pruebas falla, el código se devuelve a los desarrolladores con un informe adecuado sobre los problemas que deben solucionar. Para estas pruebas se crean datos ficticios, los datos reales no deben utilizarse y se utiliza las herramientas DAST antes mencionadas agregando también las pruebas de escaneo de vulnerabilidades, las pruebas de fuzz y las pruebas de penetración, las dos primeras pueden ser automatizadas incorporando en el pipeline, mientras que la prueba de penetración depende de la profundidad de la prueba y del tipo de aplicación.

- **Pruebas de escaneo de vulnerabilidad**

Estas pruebas se encargar de escanear el software desde el exterior para encontrar posibles vulnerabilidades. Solo se trata de recopilar información y por sí mismo no forma ningún tipo de ataque al sistema pero si el comienzo por dicha averiguación.

Un escaneo consiste en tomar la url o ip del software como entrada. Para después rastrear todos los enlaces en busca de problemas de seguridad. con el propósito de crear un informe para el probador. Existen herramientas que hacen el trabajo solo ingresando varios comandos para obtener la información como: owasp zap, nmap, sqlmap, wfuz, entre otras más. Estas pruebas se pueden automatizar e incorporar en el pipeline de CI/Cd con la ayuda de las acciones de github o git lab ci

Prueba fuzzing

Es un tipo de prueba dinámica (dast) que examina el software en funcionamiento, generando valores de entrada aleatorios. El objetivo es romper el software o revelar una vulnerabilidad de seguridad. Generando la inyección de datos mal formados / semi formados de manera automática. Algunas de las herramientas que se utilizan es el JBroFuzz Sulley, boofuzz y BFuzz.

Pruebas de penetración

Este es el nivel más alto de pruebas de seguridad donde es muy caro y la implementación para una prueba pueden llevar mucho tiempo. Sin embargo, es la mejor manera de encontrar vulnerabilidades más complejas (McDermott, 2000). Las personas que realizan estas pruebas son hackers éticos y se les concede permiso de administrado de la aplicación para realizar tales ataques.

Release

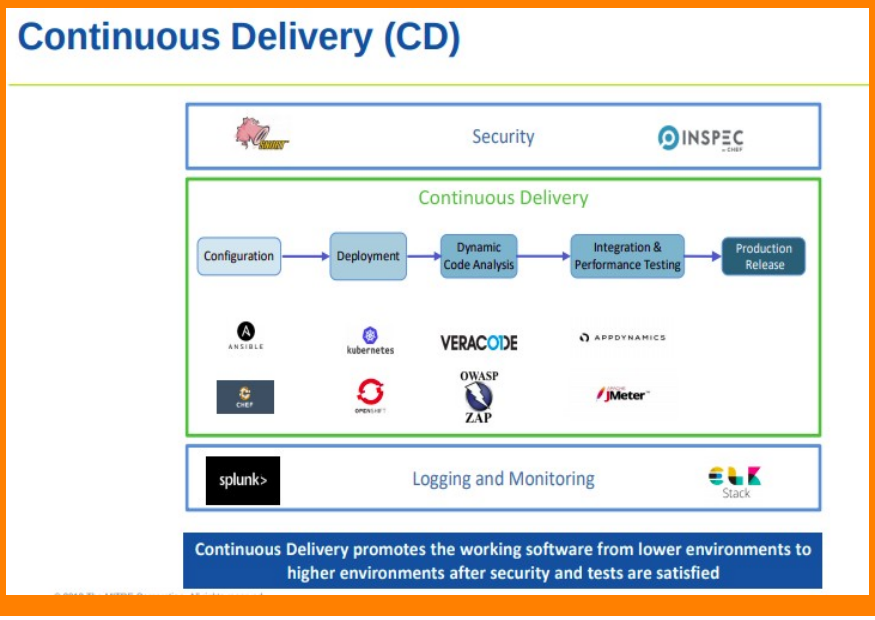
Una vez el código pasó todas las pruebas se puede liberar una versión del software por lo cual el objetivo es indicar que ha pasado con todas las validaciones anteriores. En lo posterior puede ser usada para conocer paso a paso que condujo a ese incidente y como se soluciono. Igual de importante es asegurarse de que el registro se realice en un nivel suficiente y que los registros no se puedan alterar ni perder.

Deploy

Esta fase se encargar de desplegar pequeños códigos aprobados por las pruebas anteriores a producción de forma automática sin ninguna intervención de los seres humanos. Esto se realiza a través de el despliegue continuo (CD) con el objetivo principal de automatizar el proceso de entrega de software en un entorno de producción ejecutando el código a través de múltiples pasos activados automáticamente (figura 8). CD requiere CI porque se basa en el mismo principio subyacente de dividir el trabajo en pequeños incrementos permitiendo liberar rápidamente cambios de código con poco riesgo.

Figura 8

Proceso de entrega continua CD



Nota.Adaptado de(Dhawan & Sabetto, 2019)

Operación

Esta es la fase donde se encuentra desplegado el código y es accesible para los usuarios conteniendo todos los datos reales de la aplicación. Enfrentándose a las amenazas externas que pueden provocar comportamientos raros o inadecuados donde debe contener medidas de seguridad para poder proteger la infraestructura y sus configuraciones. Aplicando y controlando con la herramientas de infraestructura ya mencionado en la guiá de owasp del mismo mono se puede agregar mas formas como el endurecimiento o Hardening y la gestión de configuración automatizada verifica continuamente que las configuraciones coincidan con las configuraciones predefinidas para proteger el entorno. El endurecimiento, a su vez, minimiza los posibles vectores de ataques cibernéticos, por ejemplo, aplicando actualizaciones automáticamente y deshabilitando cualquier servicio no utilizado. Según Walshe y Simpson (2020) “un software recién lanzado tiene, en promedio, 20 nuevas vulnerabilidades de seguridad descubiertas diariamente, disminuyendo con el tiempo a 156 vulnerabilidades por año”.

Monitoreo

Finalmente esta fase se encarga de monitorear continuamente el código implementado verificando que se ejecute correctamente. Se establece medidas para monitorizar y controlar el estado de las aplicaciones y su infraestructura.

Ahora bien para poder implementa este proceso se realiza a través del “comprobación del cumplimiento” que indica la guiá de OWASP antes mencionado. Ademas se puede agregar más herramientas de seguridad como la protección de seguridad de aplicaciones en tiempo de ejecución (RASP) y el firewall de aplicaciones web (WAF) que escanean constantemente el entorno de producción en busca de ataques cibernéticos.

Una de las clave a tener en cuenta es que las herramientas o procesos automatizados juegan un papel importante en DevSecOps.

Capítulo dos

Nombre del capítulo

2.1 Xxxxxxxxxx xxxxxx xxxxxxx xxxx

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.

2.2 Xxxxxxx xxxxxx xxxxxxxxx xxxxxxx

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Capítulo tres

Nombre del capítulo

3.1 Xxxxxxxxx

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxx.

Xxxx.

3.2 Xxxxxxxxx xxxxxxx xxxxxx xxxxxx xxxxxxx

Xxx.

Xxxx.

Conclusiones

Se redactan los puntos más sobresalientes, debilidades o fortalezas del proyecto o investigación, observados o descubiertos durante la ejecución del Trabajo de Titulación, se recomienda redactar por cada conclusión, una recomendación.

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.

Recomendaciones

En esta parte debes sugerir temas para futuras investigaciones y puedan aportar a la academia.

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.

Xxx.

Letra Arial N° 11, las recomendaciones empiezan en una nueva página, cada párrafo inicia con sangría y no colocar viñetas o numeración

Referencias

Akujobi, A. C. (2021). *A Model For Measuring Improvement Of Security In Continuous Integration pipelines*.

Chandramouli, R. (2022). Implementation of DevSecOps for a Microservices-based Application with Service Mesh Implementation of DevSecOps for a Microservices-based Application with Service Mesh. *NIST Special Publication*.

Owasp. (2021, May 1). *DevSecOpsGuideline Secrets Management*. Github. https://github.com/OWASP/DevSecOpsGuideline/blob/master/document/01a-Secrets-Management.md

Quintero, C., & Joaquín, J. (2019). *DevSecOps : integración de herramientas SAST , DAST y de análisis de Dockers en un sistema de integración continua .*

Santana, G., Neto, M., Sapata, F., Muñoz, M., Moraes, A. M. S. P., Morais, T., & Goldfarb, D. L. (2021). DevSecOps in AWS. *AWS Certified Security Study Guide*, 405–441. https://doi.org/10.1002/9781119658856.app3

Sentrio. (2021). *¿Qué es DevSecOps? Integra la seguridad en DevOps*. https://sentrio.io/blog/que-es-devsecops-vs-devops/

Shrivastava, A., & Services, N. G. (2019). DevSecOps What, Why and How Anant Shrivastava NotSoSecure Global Services. *Black Hat*. https://anantshri.info

Spiros Psarris. (2020, September 1). *Using DevSecOps to Facilitate Compliance - Reblaze Blog*. https://www.reblaze.com/blog/using-devsecops-to-facilitate-compliance/

Apéndice

Se incluye de acuerdo al orden citado en el cuerpo del Trabajo de Titulación (TT).

Apéndice A. Xxxxxxx xxxxxx xxxxxx

Xxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx xxxxxxxxx

Arial N° 10, negrita y sin punto final

Arial N° 10, sin negrita, cursiva y sin punto final

xxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxx.

Tabla A1 →

Xxxxxxxxxxx →

Internet	**Urbano**		**Rural**	
	06 a 09	10 a 11	06 a 09	10 a 11
Grado a	10	15	2	4
Grado b	15	18	3	3
Grado c	15	10	3	1

Grado d	10	20	2	1
Total	50	63	10	9

Nota. En esta tabla se observa que los niños del sector urbano tienen mayor acceso al Internet.

Arial N° 10, sin negrita, la palabra *Nota* en cursiva e interlineado doble

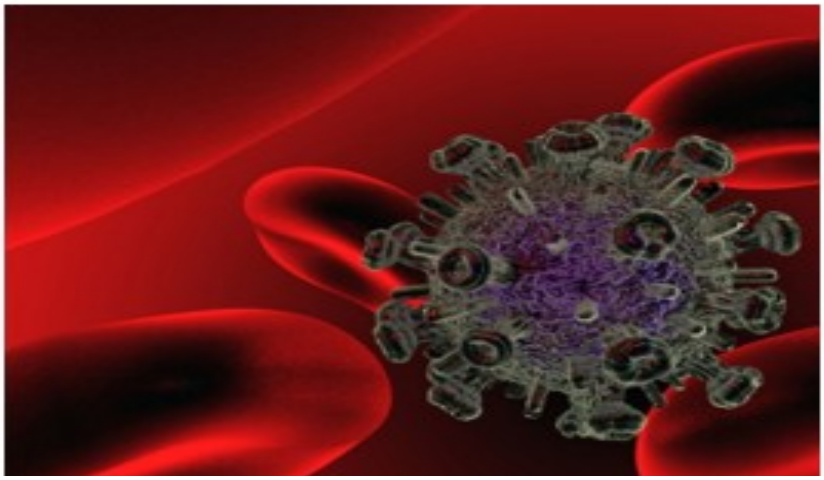
Apéndice B. Xxxxxxx xxxxxx xxxxxx

Xxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxx xxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxx.

Figura B1 → Arial N° 10, en negrita y sin punto final

Imagen del virus VIH → Arial N° 10, sin negrita, cursiva y sin punto

Inte
c



Nota. Adaptado de *Virus VIH* [Fotografia], por Consejo Superior de Investigaciones Científcas, 2011, Flickr (flic.kr/p/aronSf). CC BY 2.0.

Arial N° 10, sin negrita, la palabra *Nota* en cursiva e interlineado doble

Apéndice C. Xxxxxxx xxxxxx xxxxxx

Xxx.

Xxxx xxxxxxxxxxxxxxxxx.

Xxx.

Letra Arial N° 11, interlineado doble y sin cursiva