Universidad de Buenos Aires

Facultades de Ciencias Económicas, Ciencias Exactas y Naturales e Ingeniería

Carrera de Maestría en Seguridad Informática

Título del Trabajo:

Gestión de las Vulnerabilidades de manera efectiva

Autor:

Ing. Sergio Luis Correa Rovira

Tutor de Tesis de Maestría:

Mg. Ing Juan Alejandro Devincenzi

Año de presentación 2025

Cohorte del cursante 2016

Declaración jurada de origen de los contenidos

Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Trabajos Finales vigente y 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.

Sergio Luis Correa Rovira
DNI 32.140.003

Resumen

Gestionar las vulnerabilidades es una tarea crítica dentro de las organizaciones.

Con las nuevas metodologías de entrega continua y despliegue continuo (CI/CD), GitOps y metodologías ágiles, entre otras nuevas disciplinas, se generan cambios constantes en los servicios y aplicaciones, lo que plantea un desafío significativo la gestión de vulnerabilidades y la protección de los activos.

El cumplimiento de normativas sobre el tratamiento de datos, junto con la visión y tolerancia al riesgo de la organización, eleva la importancia de una gestión adecuada de vulnerabilidades.

Para identificar problemas de seguridad se utilizan diversas herramientas como análisis de código estático (SAST), análisis de composición de Software (SCA), escaneo dinámico de las aplicaciones (DAST), escaneo de seguridad en contenedores, modelado de amenazas (threat modelings), Tests de penetración (pentests), programas de recompensas (bug bounty), etc. Sin embargo, estas herramientas detectan una gran cantidad de vulnerabilidades de las cuales muchas veces son irrelevantes, hay muchos duplicados o falsos positivos, lo que complica la identificación de riesgos reales.

Adicionalmente, la existencia de muchas interfaces para cada tipo de herramienta dificulta la interacción con el usuario y la gestión de los hallazgos de seguridad.

Esta tesis aborda estos desafíos al describir los conceptos y condiciones necesarias para implementar un gestor de vulnerabilidades de manera efectiva. Se desarrolla un laboratorio utilizando herramientas de código abierto para lograr una implementación práctica y real. Se trabaja con un orquestador de seguridad llamado Secure Code Box, un gestor de vulnerabilidades llamado DefectDojo y diversas herramientas para la detección de vulnerabilidades

Palabras Clave:

Gestión de las Vulnerabilidades, Vulnerability Manager, DefectDojo, SecureCodeBox, SAST, DAST, DevSecOps

Contenido:

Índice de Figuras

Nómina de abreviaturas

- API, Application Programming Interface
- ATT&CK, Adversarial Tactics, Techniques, and Common Knowledge
- AWS, Amazon Web Services
- CD, Continuous Delivery
- CI, Continuous Integration
- DAST, Dynamic Application Security Testing
- DD, DefectDojo
- DevOps, Metodología que involucra Software Development y Operaciones TI
- DevSecOps, Metodología que involucra Software Development, Operaciones TI y Seguridad
- DVCS, Distributed Version Control System
- GCP, Google Cloud Platform
- GCS, Google Cloud Storage
- GKE, Google Kubernetes Engine
- GDPR, General Data Protection Regulation
- HIPAA, Health Insurance Portability and Accountability Act
- IAC, Infrastructure as Code
- IAST, Interactive Application Security Testing
- K8S, Kubernetes
- LGPD, Brazilian General Data Protection Law
- PCI-DSS, Payment Card Industry Data Security Standard
- REST, Representational State Transfer
- SAST, Static Application Security Testing
- SCA, Software Composition Analysis
- SCB, Secure Code Box
- TDD, Test Driven Development
- VCS, Version Control System

1. Introducción

1.1 Motivación del Trabajo

Según IBM la gestión de vulnerabilidades es:

"*La gestión de vulnerabilidades, un subdominio de la gestión de riesgos de TI, es el descubrimiento, la priorización y la resolución continuos de vulnerabilidades de seguridad en la infraestructura y el software de TI de una organización*"[1]

En otras palabras, la gestión de vulnerabilidades es un proceso crítico para garantizar que los sistemas y datos estén protegidos contra posibles amenazas. Para lograr una protección efectiva, es fundamental contar con un enfoque sistemático que no solo identifique las vulnerabilidades, sino que también priorice su remediación de acuerdo a su nivel de riesgo. Esto implica la necesidad de implementar políticas claras y procedimientos que faciliten la evaluación continua de los sistemas. No obstante, muchas organizaciones se enfrentan al desafío de integrar múltiples herramientas y soluciones en su infraestructura de seguridad.

Sin embargo, debido a la gran variedad de soluciones y herramientas disponibles en el mercado para abordar la gestión de vulnerabilidades, es común que exista una falta de estandarización en las soluciones que se ofrecen. Esto puede resultar en una confusión en la interpretación de los resultados obtenidos por diferentes herramientas, lo que dificulta la identificación precisa de las vulnerabilidades existentes. La inconsistencia en la terminología y los métodos de evaluación puede llevar a evaluar de forma errónea el nivel de riesgo asociado, lo que, a su vez, puede impactar negativamente en la toma de decisiones para la mitigación efectiva de amenazas.

Por lo tanto, es crucial establecer un marco común que racionalice la gestión de estas herramientas y permita una comunicación clara dentro de

los equipos de seguridad. Es necesario que las soluciones y herramientas de gestión de vulnerabilidades sean adaptadas a un lenguaje común que permita una comprensión precisa y efectiva de los resultados obtenidos. De esta manera, se podrá establecer una comunicación clara y efectiva entre los diferentes miembros del equipo de seguridad y se podrán tomar las medidas necesarias para garantizar la seguridad de los sistemas y datos.

Para las organizaciones es crítico poder gestionar las vulnerabilidades porque si alguna de estas es explotada o vulnerada, un atacante podría obtener información confidencial de la empresa, datos financieros, patentes, etc, afectar a la disponibilidad de los servicios y/o la integridad de la información. Un problema de seguridad puede afectar gravemente el negocio.

Un gestor de vulnerabilidades es una solución escalable porque se pueden integrar distintas herramientas de seguridad y visualizarlas desde una misma interfaz gráfica, ayudando a tener una mejor experiencia de usuario y poder trabajar de manera efectiva.

La presente tesis revisará los lineamientos necesarios para poder establecer un marco de trabajo en el cual se pueda establecer un gestor de vulnerabilidades de manera efectiva. Se identificarán los objetivos del negocio, las políticas y estándares de seguridad para establecer un marco metodológico de trabajo, y luego se describirán las componentes necesarias para establecer un gestor de vulnerabilidades.

El trabajo consta también de una implementación práctica, en la cual se describirán las actividades realizadas con herramientas de código abierto para que pueda ser implementado en cualquier organización sin ningún costo de licenciamiento.

1.2 Objetivo del Trabajo

- Realizar un laboratorio sobre la implementación de un gestor de vulnerabilidades para mejorar el nivel de madurez de la organización y dar el marco teórico que justifique el uso de la misma.

1.2.1 Objetivos Específicos

- Establecer los lineamientos necesarios para una gestión de las vulnerabilidades de manera efectiva, esto es:
    - Definir objetivos del negocio con respecto a la seguridad de la información
    - Establecer políticas, procedimientos y definir los requerimientos mínimos para establecer el ciclo de desarrollo de software seguro
- Desarrollar los procesos necesarios para la gestión de vulnerabilidades de manera efectiva
- Orquestar la seguridad de una infraestructura o aplicación de una organización con Secure Code Box
- Gestionar las vulnerabilidades de una infraestructura o aplicación de una organización con DefectDojo
- Utilizar distintas herramientas de seguridad para análisis estático (SAST) y dinámico (DAST) de aplicaciones.
- Crear un laboratorio que integre Secure Code Box, con DefectDojo y distintas herramientas de seguridad.

1.3 Estructura del Trabajo

Capítulo 1: Introducción

Se define los objetivos, objetivos específicos y objetivos del trabajo

Capítulo 2: Estableciendo un Baseline de Seguridad

Se define cual es el estándar mínimo de seguridad en la organización de acuerdo a sus objetivos y necesidades empresariales

Capítulo 3: Herramientas de análisis estático de seguridad en el código (SAST)

Las herramientas de código estático analizan el código sin ejecutarlo y buscando problemas de seguridad. Se explican las principales herramientas open source disponibles actualmente y algunas características

Capítulo 4: Análisis dinámico de seguridad de aplicaciones desplegadas (DAST)

Cuando tenemos servicio web o aplicaciones desplegadas en internet, hay diversas pruebas manuales y automatizadas que ayudan a simular ataques y descubrir vulnerabilidades. Se explican algunas herramientas de código abierto y características de las mismas.

Capítulo 5: Trabajando en un ambiente DevSecOps

Se explica la metodología DevOps y cómo implementar un ambiente DevSecOps. Se explican características y como ayuda en la gestión de vulnerabilidades.

Capítulo 6: Orquestador de seguridad

Se explica la complejidad de manejar múltiples soluciones de seguridad y se plantea un solución con un orquestador de seguridad llamado Secure Code Box (SCB). Se explican las las tecnologías involucradas y las características principales.

Capítulo 7: Interfáz gráfica del usuario

Para solucionar el problema de gestionar las vulnerabilidades reportadas por las distintas soluciones, se explica DefectDojo (DD). Se mencionan sus principales características y cómo se integra con SCB.

Capítulo 8: Implementación de un gestor de vulnerabilidades

Se desarrolla un laboratorio orquestando herramientas de seguridad con SCB y gestionando vulnerabilidades con DD. Se muestra paso a paso desde como instalar las herramientas, cómo se orquestan las herramientas de seguridad y características de SCB y cómo se gestionan las vulnerabilidades con DD.

Capítulo 9: Conclusiones

Las conclusiones del trabajo de tesis de maestría. Se describen los aportes realizados y oportunidades de mejora para futuros trabajos.

Capítulo 10: Bibliografía

Citas bibliográficas en la cual se basó este trabajo.

2. Estableciendo un Baseline de Seguridad

2.1 Contexto Actual de la seguridad

La seguridad en las empresas ha adquirido una relevancia creciente debido al incremento en la sofisticación y frecuencia de los ciberataques [2]. El desarrollo de herramientas basadas en inteligencia artificial, como ChatGPT y Stable Diffusion, se ha utilizado tanto como para mejorar como para realizar nuevos ataques en las organizaciones. Por ejemplo, investigaciones recientes han demostrado que modelos de difusión como Stable Diffusion pueden ser manipulados para generar imágenes alteradas, lo que representa una vulnerabilidad significativa en sistemas de IA generativa [3].

Además, las empresas manejan volúmenes cada vez mayores de datos, lo que incrementa los riesgos relacionados con la protección de la información. Este panorama se complejiza aún más con la promulgación de nuevas legislaciones de protección de datos personales, como el Reglamento General de Protección de Datos (GDPR) en la Unión Europea [4], y marcos regulatorios específicos como la Ley Sarbanes-Oxley (SOX) [5] en Estados Unidos o los estándares de la Industria de Tarjetas de Pago [6]. Estas normativas exigen altos niveles de cumplimiento e imponen sanciones significativas ante posibles violaciones, haciendo de la gestión de la seguridad un desafío multidimensional.

Todos estos problemas de seguridad no pueden ser abordados de la misma manera. Se necesita entender las necesidades del negocio para poder priorizar los riesgos de seguridad. Esto implica que ninguna empresa puede estar ajena a los problemas de seguridad, pero al menos se puede intentar minimizar el riesgo.

Existen muchas formas para poder gestionar este riesgo y muchas perspectivas para poder abordarlo. Una de las áreas en auge en las empresas es el mundo del desarrollo del software.

En el mundo del software existen también algunas organizaciones que se han dedicado a estandarizar y definir las mejores prácticas de seguridad. Tal es el caso de OWASP [7] que se dedica especialmente a la seguridad de aplicaciones web pero ahora también a aplicaciones móviles, servicios web y otros más. OWASP define cada cierto tiempo un ranking de las vulnerabilidades más críticas en determinado tipo de aplicaciones o servicios. El más conocido es OWASP top 10 en el cual se encuentran los problemas de seguridad más críticos en la actualidad.

También se pueden mencionar otros marcos de trabajo como Mitre Att&ck. La Matriz de Mitre ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) [8] es un marco de trabajo de ciberseguridad utilizado para identificar, describir y categorizar las tácticas, técnicas y procedimientos utilizados por los adversarios para llevar a cabo ataques informáticos. La matriz es una herramienta de uso libre que ayuda a los profesionales de seguridad a comprender mejor cómo los adversarios pueden explotar vulnerabilidades en los sistemas informáticos y las redes.

Todos estos marcos de trabajo anteriormente mencionados como también algunos relacionados con el mundo de Cloud como Cloud Security Pillars de Cloud Security Alliance o de algunos proveedores de cloud específicos como AWS - Well architected framework son los que ayudan a establecer un estándar de trabajo.

2.2. Cómo establecer un baseline de seguridad

2.2.1. Definiendo el Apetito de riesgo empresarial

La relevancia del área de seguridad de la información en una empresa puede variar según diferentes factores, como el tamaño de la empresa, el tipo de datos que maneja, la normativa aplicable, la sensibilidad de la información, entre otros.

En general, la seguridad de la información es cada vez más importante para las empresas debido al creciente volumen y valor de los

datos que manejan. Las empresas dependen de la información para tomar decisiones, diseñar estrategias, identificar oportunidades y competir en el mercado. Por lo tanto, cualquier pérdida o compromiso de la información puede tener consecuencias negativas para la empresa, incluyendo pérdida de ingresos, daño a la reputación, sanciones legales y financieras, entre otras.

Además, las empresas están cada vez más sujetas a regulaciones y leyes que establecen requisitos de seguridad y privacidad de la información, como la ley 25.236 de datos personales en Argentina, GDPR Reglamento General de protección de Datos en Europa [4], la Ley Sarbanes-Oxley [5], resoluciones del Banco Central como la BCRA 7724 [9] si es una empresa financiera, entre muchas otras. El no cumplimiento de estas normas puede resultar en sanciones y multas financieras que pueden afectar seriamente el negocio.

Por estas razones, muchas empresas están expuestas a diversos factores que pueden hacer que su negocio fracase. Para prevenir esto, se requiere una definición del apetito de riesgo al que la empresa está dispuesta a asumir.

Es necesario realizar un análisis cuidadoso y sistemático de los riesgos a los que se enfrenta la organización, así como de las oportunidades que pueden surgir de la asunción de ciertos niveles de riesgo. Esto implica evaluar el impacto financiero y reputacional de los riesgos, la capacidad de la empresa para absorber pérdidas, la capacidad de gestión de riesgos existente y la capacidad de innovación y adaptación al cambio.

Una vez que se ha evaluado el apetito de riesgo de una empresa, es posible establecer políticas y procedimientos para la gestión de riesgos que permitan a la empresa operar dentro de los límites de su apetito de riesgo. Esto puede incluir la implementación de controles internos, la identificación y evaluación regular de los riesgos, la asignación de responsabilidades para la

gestión de riesgos y la implementación de estrategias de mitigación de riesgos.

2.2.2. Estableciendo un Políticas, estándar de seguridad y controles internos

Una vez entendido el riesgo al cual la organización está dispuesto a asumir, se pueden establecer políticas, estándares y controles de seguridad de la información. Las políticas son declaraciones de alto nivel que establecen la dirección, el enfoque y los objetivos de la seguridad de la información en una organización.

Los estándares son conjuntos de requisitos específicos que deben ser cumplidos para garantizar la seguridad de la información en una organización.

- Por último, los controles son las medidas técnicas y administrativas implementadas para mitigar los riesgos de seguridad de la información.

Estos 3 tipos de documentos son necesarios para tener el rumbo hacia donde se quiere llegar con la seguridad de la información. Hay compañías en las cuales esto es crítico para el negocio, como pueden ser las empresas financieras o las de la salud. O existen otras que un ataque informático generaría un impacto pero el negocio puede seguir funcionando. De igual manera, en todas las compañías la seguridad tiene relevancia y es necesario establecer un apetito de riesgo en el cual la empresa está dispuesto a asumir o no.

2.2.3. Políticas de Seguridad

Elaborar una política de seguridad de la información puede ser un proceso complejo y requiere una planificación cuidadosa para asegurarse de que se aborden todas las áreas críticas de la organización. A continuación, se describen algunos pasos que pueden ayudar a construir una política de seguridad de la información:

1. *Identificar los objetivos de seguridad*: En función de las prioridades de la organización se deben identificar los activos de información críticos que se deben proteger, las regulaciones que se deben cumplir y las principales amenazas a las que están expuestos.
2. *Definir roles y responsabilidades en cuanto a la seguridad de la información*: Esto incluye a los líderes de la organización, los responsables de seguridad, los empleados y los usuarios finales.
3. *Establecer directrices y procedimientos para la gestión de los activos y protegerlos de posibles amenazas*: Esto incluye directrices sobre la protección de contraseñas, el cifrado de datos, la gestión de parches de seguridad, la gestión de dispositivos móviles, entre otros. Este trabajo se enfoca exclusivamente en la gestión de las vulnerabilidades en el ciclo de vida del software
4. *Revisión y actualización de la política de seguridad de la información* periódicamente para garantizar que siga siendo relevante y efectiva en la protección de los activos de información de la organización.

Hay otros puntos que se pueden mencionar en la creación de políticas de seguridad de la información como por ejemplo la capacitación de los empleados, penalidades por incumplimiento.

2.2.4 Ciclo de Vida de Desarrollo de Software

Las organizaciones que trabajan en el desarrollo del software necesitan establecer lineamientos para realizar un proceso estructurado, que guíe el desarrollo, mantenimiento y retiro de las aplicaciones de software. Este proceso, es conocido comúnmente como Software Development Lifecycle (SDLC), incluye varias fases que aseguran que el software sea desarrollado de manera eficiente, cumpliendo las necesidades del negocio y garantizando la calidad del producto final.

Este proceso suele constar de las siguientes fases:

1. *Planificación y Análisis de requisitos*, que sirven para entender las necesidades de negocios
2. *Diseño del Sistema*, crea los detalles técnicos para cumplir con los requisitos diseñados anteriormente
3. *Desarrollo del Software*, es la etapa en la cual los desarrollo empiezan a hacer realidad el diseño, en esta etapa es cuando se crea el código que va a tener la lógica de la aplicación
4. *Pruebas de software,* para asegurar la calidad del producto
5. *Implementación y despliegue*, que permite que los usuarios finales puedan utilizar el producto
6. Mantenimiento para asegurar el funcionamiento continuo
7. *Retiro*, en el caso de que la aplicación sea discontinuada, se debe planear el retiro

Las metodologías de trabajo, el control de versiones, documentación también puede ser parte del proceso.

2.2.5 Ciclo de Vida de Desarrollo de Software Seguro

Es importante poder integrar la seguridad en el ciclo de vida de desarrollo del software para que el software sea resistentes a ataques, y cumpla con las políticas de seguridad, es por esto que es necesario agregar los siguientes aspectos en la política de desarrollo del software:

1. *Security by Design*: Establecer los principios de seguridad desde la fase de diseño, como el principio de mínimo privilegio, logs, backups, autenticación, etc.
2. *Integración de Pruebas de seguridad*: Es importante realizar distintos análisis para poder identificar potenciales amenazas. Estas herramientas se pueden correr de manera manual o automática en el pipeline[1]. Existen herramientas que se utilizan para hacer análisis de

[1] Un pipeline es una serie automatizada de pasos que transforma el código desde su desarrollo hasta su implementación, asegurando calidad, pruebas y despliegue continuo. En un pipeline se ejecutan tareas en orden, siguiendo reglas definidas

código estático (SAST, Static Application Security Testing), detectar librerías o frameworks desactualizados, también denominado análisis de composición del Software (SCA, Software Composition Analysis), en las aplicaciones desplegadas se puede realizar un análisis de vulnerabilidades dinámico (DAST, Dynamic Application Security Testing), entre otros tipos de herramientas que existen en el mercado. Algunas de estas herramientas se explican en detalle en los siguientes capítulos.

3. *Análisis de potenciales vulnerabilidades*: Hay actividades que se pueden utilizar para detectar en mayor profundidad posibles amenazas. Sesiones de modelado de amenazas (o Threat Modeling) son actividades en las que se juntan distintos equipos que están desarrollando el producto, representantes del equipo de seguridad y gente del negocio. En esta actividad el equipo de seguridad empieza a preguntar a los equipo de desarrollo cómo se protegen de distintas amenazas y se explican potenciales vectores de ataque. También existen actividades como los pentest, en los cuales gente de seguridad especializada simula ser un atacante. Las vulnerabilidades detectadas en estas actividades son luego reportadas
4. *Gestión de Vulnerabilidades*: Establecer procedimientos para la detección, reporte y mitigación de vulnerabilidades a lo largo de todo el ciclo de vida de desarrollo del software.
5. *Monitoreo y respuesta a incidentes*: Integrar capacidades de monitoreo continuo y respuesta a incidentes para poder mitigar ataques en tiempo real.
6. Capacitación continua: Educar a los desarrolladores sobre las mejores prácticas de seguridad y entender las potenciales vulnerabilidades es fundamental para la mejora continua.
7. Existen otras actividades que pueden incluirse como implementar prácticas de código seguro, programas de security champion, exponer la aplicación a un programa de bug bounty, etc

2.2.6 Problemas en el Ciclo de Vida de Desarrollo de Software Seguro

Evidentemente, agregar seguridad al ciclo de vida puede tener complicaciones si no se establecen prioridades. Es fundamental entender el riesgo al cual se exponen las aplicaciones y que es aceptable para el negocio.

Hay herramientas que pueden correr en el código (SAST y SCA) y en aplicaciones desplegadas (DAST) para todos los gustos. Hay muchos proveedores de estos servicios, cada uno con sus fortalezas y debilidades. Muchas de estas soluciones son pagas, y pueden ser costosas dependiendo del presupuesto de la organización. Este trabajo se va a focalizar en herramientas Open Source, las cuales son gratuitas y analizan las aplicaciones con una calidad aceptable.

Las herramientas SAST y DAST entregan un reporte en el cual están todas las potenciales amenazas y vulnerabilidades. Estos reportes muchas veces pueden estar en diferentes formatos. También suelen tener su propio gestor de vulnerabilidades. Tener distintas soluciones que gestionen vulnerabilidades de manera independiente hace difícil tener una visión centralizada e integral de los problemas de seguridad, los desarrolladores que tienen que mitigar las vulnerabilidades tienen que aprender a usar las distintas interfaces que tienen estas herramientas y hace que la dinámica sea tediosa.

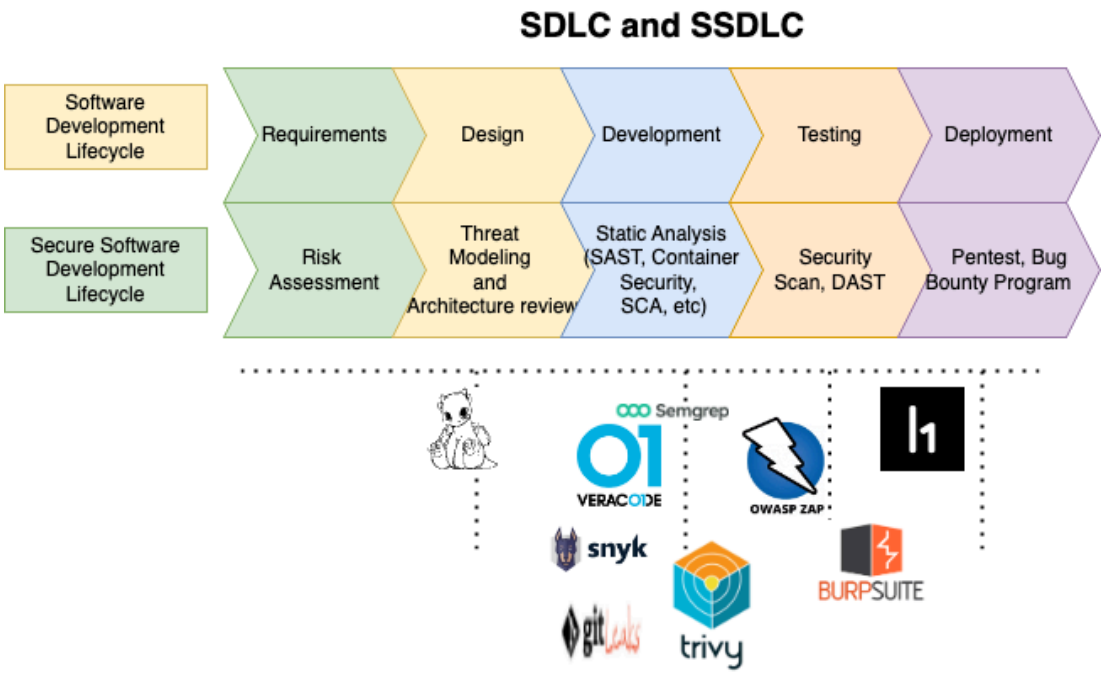

Figura 1 - Herramientas de seguridad en ciclo de Vida del desarrollo del Software

Algunas herramientas tienen reportes y/o entregan resultados de manera estandarizada (el formato SARIF[2] por ejemplo), en muchos otros casos las soluciones entregan resultados de manera customizada a su caso de uso.

[2] El formato SARIF es un estándar definido por Github para reportar vulnerabilidades en formato json, más detalles en el siguiente link: https://docs.github.com/en/code-security/code-scanning/integrating-with-code-scanning/sarif-support-for-code-scanning

3. Herramientas de análisis estático de seguridad en el código (SAST)

3.1. Características de las herramientas de análisis de seguridad en el código

El análisis estático de seguridad en el código es un proceso crucial para identificar y posteriormente remediar las vulnerabilidades antes de que la aplicación sea desplegada. Es una detección temprana que puede ayudar a prevenir ataques que podrían comprometer la aplicación, exponiendo datos sensibles, vulnerando sistemas críticos y hasta incluso la reputación de la organización.

Es por esto, que resulta importante la implementación de análisis de seguridad en el código, y es oportuno incluirlo en el ciclo de vida de desarrollo del software.

Con este tipo de herramientas se puede:

- *Prevenir Vulnerabilidades*: Permiten identificar vulnerabilidades como inyecciones SQL, buffer overflow, acceso remoto por código (RCE, Remote code Execution), etc. que pueden ser explotados por algún agente malicioso.
- *Cumplir con Normativas Vigentes*: Algunas industrias exigen que se realice un análisis de seguridad en el código, algunos ejemplos son PCI-DSS, HIPAA y GDPR.
- *Reducción de Costos*: Detectar y corregir vulnerabilidades en las primeras etapas del ciclo de desarrollo del software reducen significativamente los costos que hacerlo luego cuando la aplicación está desplegada.
- *Automatización*: Las herramientas de seguridad se pueden configurar para ejecutarse desde el entorno gráfico donde los desarrolladores están trabajando o incluso en los sistemas de control de versiones distribuidos como Github, Gitlab o Bitbucket, entre otros cada vez que

suben código al repositorio central. Este último se ejecuta como parte del proceso de integración continua.

- *No requiere un entorno para ejecutarlo*: Como no se ejecuta la aplicación, no es necesario un entorno específico o datos de prueba

Como contrapartida de este tipo de herramienta, tiene algunas limitaciones:

- *Falsos positivos*: Puede generar un alto número de falsos positivos lo que puede llevar a una pérdida de tiempo y recursos al investigar problemas que no son vulnerabilidades reales.
- *Análisis Parcial*: Algunas vulnerabilidades que solo se manifiestan en tiempo de ejecución no son detectadas, como por ejemplo problemas de configuración, interacción con otros sistemas, o relacionadas con la lógica del negocio.
- *Requiere acceso al código fuente*: No se puede realizar análisis cuando no se tiene acceso al código fuente, lo que es inaplicable para aplicaciones de terceros empaquetadas

3.2. Tipos de Herramientas de análisis en el código

Existen distintos tipos de herramientas de análisis de código. El uso de estas dependen de las necesidades y el caso de uso:

- *SAST, Static Application Security Testing*: Estas herramientas hacen análisis del código fuente de una aplicación en busca de vulnerabilidades sin ejecutar el programa. Algunos ejemplos de soluciones pagas son SonarQube, Checkmarx, Fortify. Ejemplos de open source podemos encontrar Semgrep, Bandit, nodejsscan, etc. [10]
- *SCA, Software Composition Analysis*: Estas herramientas analizan las dependencias en librerías de terceros para identificar vulnerabilidades conocidas. Algunos ejemplos son Dependabot, Snyk, WhiteSource

- *IaC Scans, Infrastructure as code scans*: Este tipo de herramientas de análisis de configuración de infraestructura, revisa la configuración de la infraestructura como código para asegurar que la infraestructura esté configurada de manera segura. Un ejemplo es Checkov[3].

3.3. Ejemplos de Herramientas de Seguridad de código estático

Existen múltiples herramientas que se utilizan para realizar análisis de código estático, solo mencionamos ejemplos en los cuales luego realizamos los laboratorios.

3.3.1 Semgrep

Semgrep [11] es una herramienta de análisis estático de código (SAST) diseñada para realizar búsquedas y análisis precisos en código utilizando expresiones regulares para detectar patrones específicos en el código.

Se utiliza un sistema de reglas que combina búsquedas de texto con una comprensión del contexto sintáctico del código, lo que lo hace ideal para detectar problemas específicos y patrones en lenguajes como Python, JavaScript, Java, Go, entre otros.

Según la documentación oficial, Semgrep permite realizar análisis de seguridad en más de 30 lenguajes de programación. Cuenta con una comunidad activa que ha contribuido con más de 2,000 reglas personalizadas, y hasta la fecha, se han ejecutado más de 75 millones de análisis en distintos códigos. La herramienta ofrece tanto una versión gratuita de código abierto como una versión paga [11]; en nuestro caso, nos enfocaremos exclusivamente en la versión Open Source.

[3] https://www.checkov.io/

Para ejecutarlo, sobre una aplicación, solo se debe instalar la aplicación según la documentación oficial[4] y para ejecutar un scan se puede realizar con el siguiente comando:

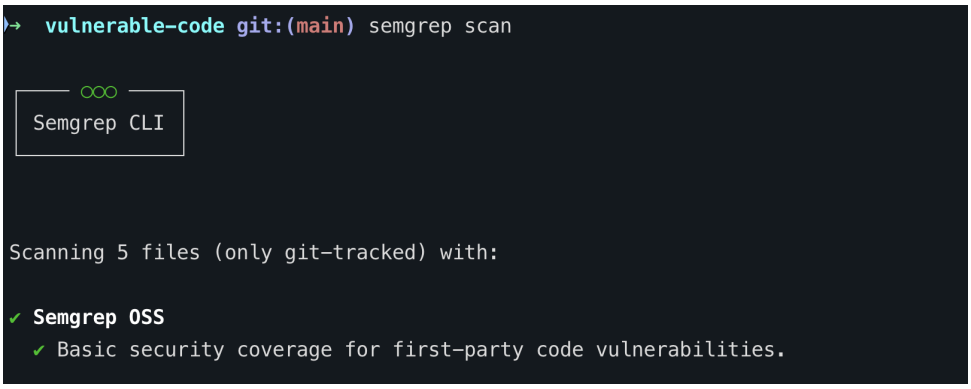

Figura 2 - Scan con Semgrep en repositorio local. Con el comando semgrep scan utiliza las reglas de escaneo por defecto y realiza un scan en el directorio local

Un ejemplo del reporte que entrega es el siguiente:

Figura 3 - Reporte obtenido luego de un scan con semgrep

[4] En la documentación oficial se explica como instalar semgrep según el sistema operativo usado: https://semgrep.dev/docs/getting-started/quickstart

4. Análisis dinámico de seguridad de aplicaciones desplegadas (DAST)

4.1. Ambientes de trabajo

Las aplicaciones atraviesan diversas etapas en el ciclo de desarrollo de software, que incluyen diseño, planificación, codificación, despliegue, monitoreo y mantenimiento, hasta el final de su ciclo de vida. A lo largo de estas fases, la seguridad debe ser una consideración constante, implementando actividades como el Modelado de Amenazas (Threat Modeling), SAST, SCA, DAST y revisiones de arquitectura.

Para garantizar un entorno seguro en el desarrollo de una aplicación, es crucial cumplir con ciertas condiciones y utilizar herramientas de seguridad adecuadas. Durante un primer despliegue, se recomienda realizarlo en un ambiente controlado, que no esté expuesto a Internet y que no contenga información confidencial, o que esta información esté enmascarada u ofuscada. Esto se debe a que en estas fases iniciales, el ambiente puede ser inestable y varios equipos, como desarrolladores, accederán a él para realizar diferentes pruebas.

Los entornos típicos incluyen: Development para las pruebas iniciales, Staging cuando la aplicación se asemeja a la de producción y, finalmente, el ambiente de producción. Dependiendo de las necesidades, pueden crearse otros entornos adicionales para el despliegue.

Es fundamental restringir el acceso a los ambientes que no son de producción. Se recomienda limitar la cantidad de usuarios y, siempre que sea posible, restringir el acceso. Esto es crucial, ya que si una aplicación es expuesta, un atacante podría aprovecharse de vulnerabilidades que aún no han sido mitigadas.

Para llevar a cabo análisis de seguridad en aplicaciones desplegadas, es preferible utilizar entornos no productivos. El ambiente Staging es ideal,

ya que es similar al de producción pero carece de información sensible del cliente.

Si la aplicación utiliza algún tipo de autenticación, puede ser necesario contar con usuarios que tengan distintos roles para escanear la aplicación con diversos niveles de privilegio.

4.2. Análisis de seguridad sobre Aplicaciones desplegadas

Una vez que las aplicaciones se encuentran en condiciones de ser desplegadas, es posible realizar análisis de seguridad adicionales que simulan actividades de un actor malicioso (ciberatacante). Las actividades más comunes que se realizan en las organizaciones son DAST, que suele ser una actividad automatizada y test de Penetración (pentest) que es una actividad manual

4.2.1. Que es DAST (Dynamic Application Security Testing)

DAST es un tipo de escaneo que se realiza en aplicaciones desplegadas en internet. Pueden ser aplicaciones Web, API o móviles. Se centra en buscar vulnerabilidades de la aplicación de manera externa, es decir, no entiende cómo se maneja la aplicación de manera interna y no se tiene acceso al código. DAST analiza la aplicación mientras está corriendo y realiza simulaciones de ataques. La interacción con la aplicación determina si es vulnerable o no a un ataque malicioso real [12].

4.2.2. Características principales de DAST

DAST es una herramienta de seguridad que se utiliza para detectar potenciales vulnerabilidades en las aplicaciones mientras se está ejecutando, las principales características que tiene son [13]:

- *Pruebas en entornos de ejecución*: DAST realiza los escaneos en aplicaciones que se están ejecutando, permitiendo identificar el comportamiento de la aplicación, vulnerabilidades en la lógica de negocio, problemas en la autenticación y autorización.

- *Simulación de ataques*: Estas herramientas explotan vulnerabilidades comunes conocidas, como inyección SQL, cross-site scripting (XSS), ataques de fuerza bruta, y otros tipos de amenazas
- *No requiere acceso al código fuente*. A diferencia de SAST, DAST se ejecuta sobre aplicaciones que se están ejecutando.
- *Identificación de vulnerabilidades en tiempo real.*
- *Menor tasa de falsos positivos*. DAST necesita explotar las vulnerabilidades para detectar nuevas amenazas. Esto reduce la tasa de falsos positivos

4.2.3. Herramientas DAST en el mercado

Existen muchas herramientas que ofrecen estas características, existen algunas que son licenciadas, y otras open source. En el siguiente gráfico se puede ver las distintas herramientas para hacer escaneos en las aplicaciones webs dinámicos:

Figura 4 - Alternativas DAST en el mercado [14]

Entre las aplicaciones comerciales tenemos Synopsys, Tenable, BurpSuite. Entre las aplicaciones Open Source, la más reconocida es OWASP ZAP

4.2.4. OWASP ZAP

OWASP ZAP (Zed Attack Proxy) es una herramienta de OWASP de código abierto que sirve para hacer pruebas de seguridad en las aplicaciones web. Tiene un proxy que intercepta el tráfico de un sitio web y permite identificar vulnerabilidades en las aplicaciones mediante técnicas automatizadas y/o pruebas manuales [15].

Figura 5 - Zap funciona como proxy entre el sitio web y el navegador [15]

Entre sus principales características, se encuentran:

- *Es un Proxy que intercepta el tráfico web*: Permite analizar y modificar solicitudes y respuestas http
- *Permite escaneos automatizados (DAST scans)*: Incluye herramientas para identificar vulnerabilidades conocidas en sitios web
- *Testing Manual*: Permite hacer uso de herramientas manuales para vulnerabilidades que no se pueden identificar de manera automática
- *Plugins*: Tiene una serie de "Add-on" que permite agregar nuevas funcionalidades
- *Contiene una API REST*: Tiene una api que permite realizar integraciones en los pipelines CI/CD para pruebas automatizadas
- *Automation Framework*: Tiene una funcionalidad que permite la integración automatizada de escaneos por medio de archivos de configuración [15]. Especialmente útil en ambientes DevSecOps y utilizada por Secure Code Box.

4.2.4.1. Ejemplo de escaneo con OWASP ZAP

Para mostrar un ejemplo de cómo ejecutar OWASP ZAP, es necesario instalar la aplicación y luego configurar un navegador como proxy.

Una vez definido cual es el objetivo, se puede agregar al target a través de contextos en OWASP ZAP y directamente navegando a través del navegador que está configurado como proxy.

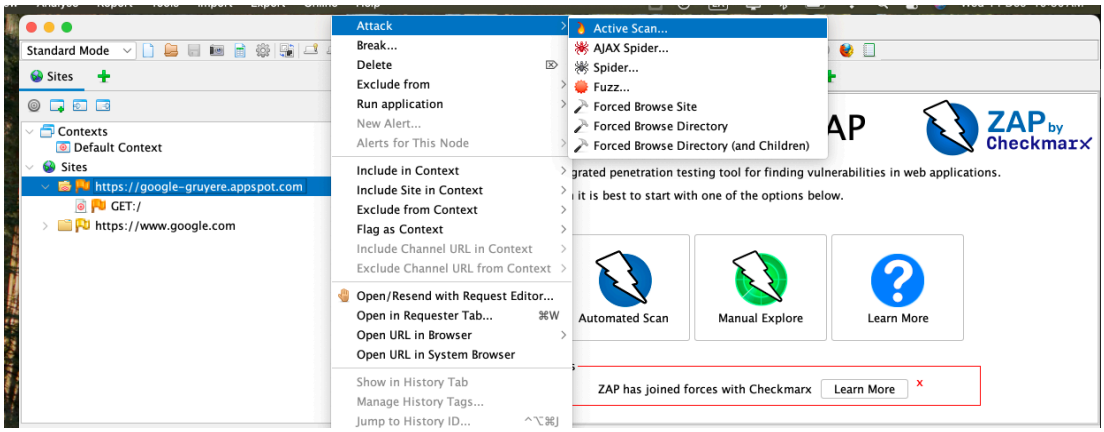

Figura 6 - Como realizar un escaneo DAST con OWASP ZAP mediante la interfaz gráfica

Los resultados se pueden ver en la siguiente figura:

Figura 7 - Resultados de un escaneo automático con OWASP ZAP

Estos resultados pueden exportarse en distintos formatos, desde json, XML, html,, pdf, etc.

4.3. Test de Penetración

4.3.1 Que es un Pentest

Un test de penetración, Penetration test, o simplemente pentest, es una actividad en la cual especialistas de seguridad simulan ataques controlados a una aplicación, sistemas o infraestructuras imitando escenarios del mundo real donde atacantes buscan debilidades en las aplicaciones. Involucra ejecutar ataques reales sobre el objetivo usando herramientas comúnmente usadas por atacantes reales [16]. Además, las pruebas de penetración son útiles para determinar:

- **La tolerancia del sistema frente a ataques reales:** Evaluando cómo las configuraciones actuales enfrentan patrones de ataque que se asemejan a los utilizados en el mundo real.
- **El nivel de sofisticación necesario para comprometer el sistema:** Determinando qué habilidades y recursos requeriría un atacante para explotar las vulnerabilidades identificadas.
- **Contramedidas adicionales:** Identificando medidas de mitigación que puedan reforzar la seguridad contra amenazas detectadas.
- **La capacidad de detección y respuesta del equipo defensor:** Evaluando qué tan bien se detectan los ataques y cómo se responden de manera adecuada [17].

Esta actividad proporciona una visión práctica y detalla vulnerabilidades explotables permitiendo identificar fallas en la aplicaciones y configuración de la red, definir prioridades, cumplir con normativas y estándares de seguridad, etc.

4.3.2 Tipos de Pentest

Para realizar este tipo de actividad, se define un alcance u objetivo, que normalmente puede ser una aplicación web, API, móviles, etc. e incluso infraestructura de red. Esta actividad se puede diseñar de distintas maneras, cada una con distintas características:

- *Pentest de caja blanca (White Box Testing)*, en la cual el equipo de pentesting tiene acceso total a la aplicación, desde el código fuente, configuración y documentación detallada. Esta actividad se utiliza para hacer un análisis profundo y exhaustivo de la aplicación o sistema.
- *Pentest de caja negra (Black Box Testing)*, en el cual el pentester no tiene conocimiento previo del entorno o aplicación. Es el ejemplo más realista de un atacante externo, basándose solo en lo que puede observar desde afuera.
- *Pentest de caja gris (Grey Box testing)*, en este tipo de pentest se tiene acceso parcial a la información interna, como credenciales de usuario o detalles de la arquitectura del sistema, pero no cuenta con acceso completo al código fuente o la infraestructura.

A estos tipos de pentests, se pueden aplicar a distintos tipos de ambientes como por ejemplo aplicaciones Web, Mobile, APIs, arquitectura de redes, de infraestructura, de la nube, etc.

4.3.3 Fases de un Pentest

Existen muchos estándares y procesos en los cuales se definen los pasos de un pentest, entre estos podemos mencionar OWASP Testing

Guide[5], PTES (Penetration Testing Execution Standard)[6], NIST SP 800-115 (Technical Guide to Information Security Testing and Assessment) [17], entre muchos otros. Todos tienen características particulares, pero en líneas generales, se estructuran las fases de un pentest de la siguiente manera:

1. *Reconocimiento y recopilación de información*: El pentester obtiene información sobre la aplicación, y el entorno para conocer su estructura, configuraciones, y posibles puntos de entrada
2. *Escaneo de vulnerabilidades*: Se utilizan herramientas automáticas para detectar vulnerabilidades y se realizan análisis manuales para identificar y explotar posibles vulnerabilidades
3. *Explotación de vulnerabilidades*: Se intentan explotar vulnerabilidades encontradas para confirmar su existencia y entender hasta qué punto se puede comprometer el sistema
4. *Post-explotación y análisis de impacto*: Se evalúa el impacto de la explotación, como si se lograra acceso a información crítica o control de la infraestructura
5. *Reporte*: Se elabora un informe con los hallazgos, incluyendo detalles técnicos, impacto, riesgos y recomendaciones de mitigación
6. *Remediación y Verificación*: El equipo de desarrollo realiza las correcciones necesarias, y se hace una verificación para asegurar que las vulnerabilidades hayan sido mitigadas

Realizando pentests de manera periódica y trabajando en la remediación, ayuda a fortalecer la postura de seguridad de la aplicación, y del ambiente, ayudando a construir aplicaciones más seguras y resilientes, y los desarrolladores mejor entrenados ya que obtienen el conocimiento de cómo evitar estos problemas de seguridad.

[5] https://owasp.org/www-project-web-security-testing-guide/
[6] http://www.pentest-standard.org/index.php/Main_Page

5. Trabajando en un ambiente DevSecOps

5.1. Beneficios de un ambiente DevSecOps

DevOps es una metodología que combina el desarrollo de software con las operaciones de TI para mejorar la eficiencia en la entrega de aplicaciones y servicios. El objetivo es automatizar y optimizar el ciclo de desarrollo de vida del software. Esto es posible a través de una colaboración de equipos de desarrollo y operaciones, el uso de herramientas de automatización y la adopción de prácticas ágiles [18].

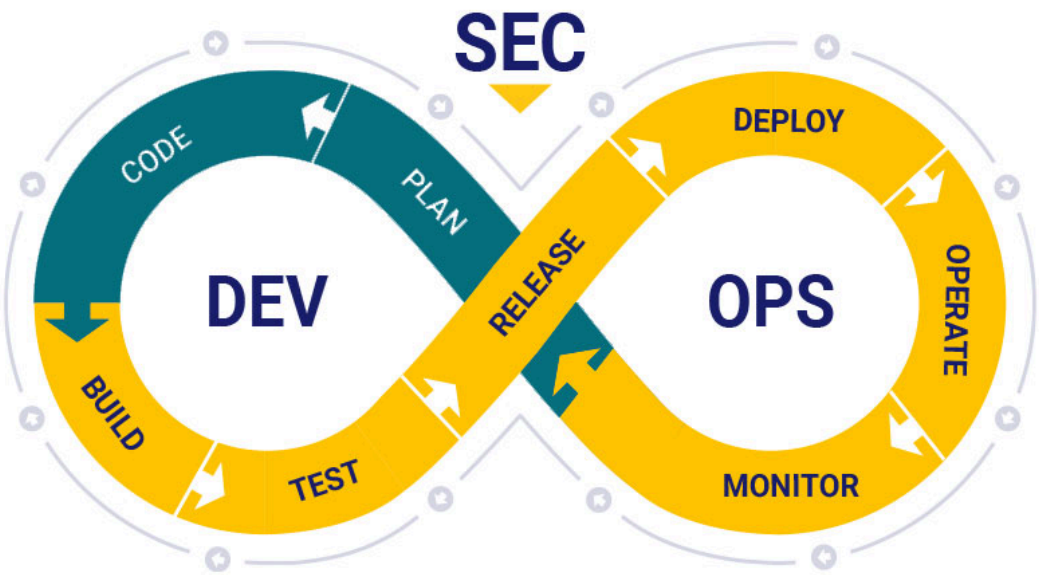

Figura 8 - Ciclo DevSecOps [19]

En los ambientes DevOps los equipos de calidad y seguridad pueden incorporarse con el desarrollo y operaciones a través del ciclo de desarrollo del software. Cuando esto sucede, la metodología suele llamarse DevSecOps.

Según IBM DevSecOps es:

"*Una práctica de desarrollo de software que automatiza la integración de la seguridad y mejores prácticas en cada fase del ciclo de vida de*

desarrollo de software, desde el diseño inicial, hasta la integración, las pruebas, los despliegues y la implementación" [20].

Los principales beneficios que tiene esta metodología son:

1. *Velocidad*: Los equipos se pueden mover a gran velocidad y adaptarse a los cambios para lograr resultados. Esto se puede hacer con microservicios y entrega continua (CD, Continuous Delivery)
2. *Entrega Rápida*: Se puede aumentar la frecuencia de entregas al cliente con prácticas de Integración continua (CI, Continuous integration) y entrega continua (CD) que automatizan los procesos desde el desarrollo hasta el despliegue
3. *Fiabilidad (Reliability)*: Se puede asegurar la calidad implementando controles dentro de la integración continua y entrega continua implementando pruebas automáticas que corroboren el funcionamiento esperado de la aplicación. Se puede implementar metodologías de pruebas de código (Un ejemplo es TDD, Test Driven Development[7])
4. *Escalabilidad*: Consistencia y automación ayuda a gestionar sistemas complejos que cambian constantemente de manera eficiente reduciendo el riesgo. Con técnicas de Infraestructura como código ayudan a gestionar el desarrollo, pruebas y ambientes de producción de manera repetitiva y eficiente.
5. *Colaboración contínua*: Esta metodología hace énfasis en la colaboración de los equipos, tomando ownership y responsabilidad (accountability)
6. *Seguridad*: Herramientas como SAST y DAST ayudan a detectar las posibles amenazas de seguridad. Se pueden implementar también herramientas usando infraestructura como código y políticas como código para mejorar la seguridad de manera escalable [20].

[7] TDD es una metodología de diseño de software basado en pruebas, mas detalle en el siguiente link: https://devops.com/the-benefits-of-test-driven-development/

5.2. GIT en ambientes DevSecOps

Para trabajar en un ambiente DevSecOps y cumplir con las características mencionadas anteriormente, es necesario trabajar en una plataforma ágil, escalable y de colaboración contínua, entre otras características.

En un entorno de desarrollo, se puede utilizar una plataforma de desarrollo colaborativo como Git [21]. Git es un sistema de control distribuido de versiones (DVCS), que permite a los distintos desarrolladores de un proyecto colaborar y gestionar el código de sus proyectos. Puede ser utilizado para trabajar de manera simultánea en un mismo proyecto manteniendo un control detallado de las versiones o modificaciones. Git es implementado por varios proveedores, los más conocidos son Github[8], Gitlab[9] y Bitbucket[10].

Trabajar con un sistema de control distribuido de versiones como Github permite a los desarrolladores focalizarse en las tareas de desarrollo en el proyecto, colaborar en tiempo real, gestionar cambios en el código y automatizar procesos de desarrollo.

Entre las principales características de Github se encuentran:

- *Almacenamiento centralizado*: El código se almacena en repositorios para gestionar proyectos de código
- *Control de versiones*: Usa Git para el control de versiones de código, permitiendo ramas, uniones (del inglés merges), bifurcaciones (del inglés forks) y seguimiento de cambios
- *Ambiente colaborativo*: Con Pull Requests (PRs) se puede colaborar en cambios de código con revisiones antes de ser integrados a la rama principal

[8] Sitio oficial de Github https://docs.github.com/es/get-started/start-your-journey/about-github-and-git
[9] Sitio Oficial de Gitlabhttps://about.gitlab.com/
[10] Sitio oficial de Bitbucket https://bitbucket.org/

- *Automatización de trabajo*: Se pueden automatizar flujos de trabajo como la realización re pruebas, scans de seguridad, despliegue y análisis con Github Actions, una característica nativa de github
- *Herramientas de Seguridad*: Ofrece herramientas de seguridad como dependabot, análisis de código con CodeQL y gestión de secretos.

Dentro del entorno de trabajo desarrollado para la tesis, vamos a trabajar sobre repositorios en github, y vamos a utilizar la característica de github actions para automatizar los flujos de trabajo.

5.3. Github Actions

GitHub Actions es una herramienta de integración y entrega continua (CI/CD) integrada directamente en GitHub. Su principal objetivo es automatizar flujos de trabajo en el desarrollo de software, como la ejecución de pruebas, compilaciones y despliegues [22].

La configuración de GitHub Actions se realiza mediante archivos YAML almacenados en el directorio *.github/workflows* del repositorio. Dentro de estos flujos de trabajo, se definen jobs o tareas específicas, que pueden ejecutarse en paralelo o en serie dependiendo de las dependencias configuradas. Cada job, a su vez, consta de steps o pasos que contienen las acciones individuales, como ejecutar scripts, instalar dependencias o realizar despliegues.

Esta herramienta es especialmente útil para optimizar procesos repetitivos y tediosos, como pruebas y despliegues, reduciendo significativamente el tiempo y esfuerzo requeridos. A continuación, se muestra un ejemplo de un archivo YAML de configuración y cómo se visualiza el flujo resultante directamente en GitHub.

vulnerable-code / .github / workflows / **SAST.yml**

vira-vira Update SAST.yml ✓

Code | Blame | 36 lines (27 loc) · 988 Bytes | Code 55% faster with GitHub Copilot

```yaml
name: Run SAST with Securecodebox

on:
  push:
    branches:
      - main

jobs:
  sast-scan:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - id: 'auth'
        uses: 'google-github-actions/auth@v2'
        with:
          credentials_json: '${{ secrets.GCP_SERVICE_ACCOUNT_KEY }}'

      - name: Set up Google Cloud SDK
        uses: google-github-actions/setup-gcloud@v1

      - name: Install gke-gcloud-auth-plugin
        run: |
          gcloud components install gke-gcloud-auth-plugin
          gcloud components install kubectl

```

Figura 9 - Extracto de código de una github actions - el código está disponible en https://github.com/vira-vira/vulnerable-code/blob/main/.github/workflows/SAST.yml

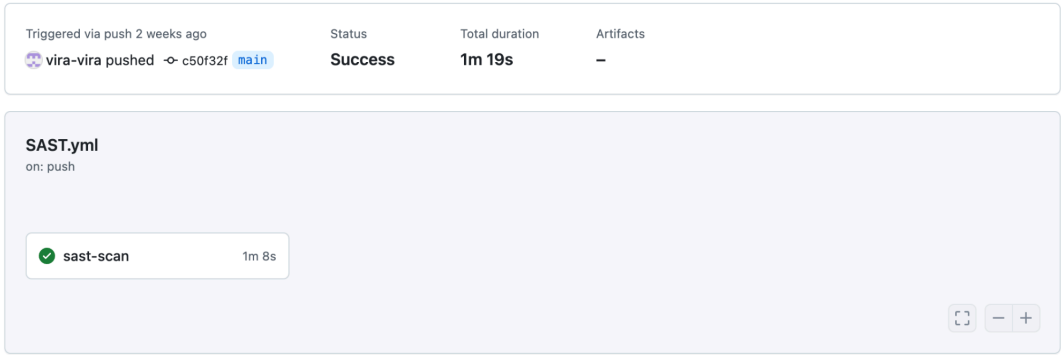

Figura 10 - Ejemplo de como se visualiza un flujo con github actions, el código está disponible en https://github.com/vira-vira/vulnerable-code/actions

6. Orquestador de seguridad

6.1. Dificultades del ambiente DevSecOps

Uno de los principales desafíos que tiene un ingeniero en seguridad en las aplicaciones, es poder realizar un análisis efectivo de las vulnerabilidades que aparecen en los distintos entornos de trabajo de una aplicación.

En el siguiente gráfico, se puede ver las distintas alternativas de herramientas de trabajo para hacer distintas actividades, desde análisis de puertos de red abiertos en una aplicación o infraestructura, enumeración de dominios, búsqueda de librerías vulnerables en el código, análisis de vulnerabilidades en aplicaciones Web, y muchos otros tipos de herramientas. Este gráfico sólo enumera algunas de las herramientas de código abierto que están en el mercado.

Figura 11 - Logos de herramientas que realizan distintas actividades de seguridad. Estas herramientas están contenidas en Secure Code Box

Tantas alternativas generan algunos inconvenientes. Por ejemplo si ejecutamos herramientas que hacen el mismo tipo de scan de seguridad, vamos a tener las mismas vulnerabilidades repetidas en los reportes de distintas herramientas

Gestionar las vulnerabilidades reportadas por estas herramientas es un desafío. Por ejemplo cuando se detectan vulnerabilidades que son falsos positivos, esas vulnerabilidades no tendrían que ser reportadas nuevamente. Otro caso es en el que una vulnerabilidad existe pero tiene un control

compensatorio haciendo que el riesgo disminuya o que la vulnerabilidad no sea explotable.

Cada herramienta tiene una forma de reportar las vulnerabilidades encontradas. Pueden ser reportes en distintos formatos como pdf, html, json, etc. O incluso pueden tener una interfaz gráfica se pueden gestionar las vulnerabilidades. En algunos casos, con herramientas más maduras, tienen un gestor de vulnerabilidades incluído dentro de la herramienta.

Resolver estas vulnerabilidades resulta extremadamente tedioso, porque hay que navegar por distintas plataformas, por distintos reportes, ver las mismas vulnerabilidades que entregan distintas herramientas. Además, muchas de estas vulnerabilidades reportadas suelen ser falsos positivos o son vulnerabilidades que no son explotables.

6.2. Secure Code Box

Por la dificultad de gestionar tantas herramientas de seguridad, es que en OWASP hay un proyecto abierto a la comunidad que se llama Secure Code Box [23] (SCB en adelante). Es una herramienta basada en tecnologías como Kubernetes (k8s) y Docker, montada en una arquitectura de microservicios. Para entender cómo funciona SCB es necesario entender primeramente las tecnologías que lo integran.

Figura 12 - Logo de Secure Code Box - Fuente: https://www.securecodebox.io/

6.2.1. Tecnologías usadas en Secure Code Box

- *Arquitectura de microservicios*: Es un enfoque de diseño de software en el que las aplicaciones se descomponen en pequeños servicios independientes, cada uno con una función específica. Estos servicios se comunican a través de APIs y son escalables, desplegables y desarrollables de forma autónoma, lo que mejora la agilidad y la resiliencia del sistema [24]. Este tipo de arquitectura contrasta con las aplicaciones monolíticas, que son servicios que tienen todos los recursos necesarios para correr un servidor, por ejemplo la interfaz, base de datos, procesamiento, etc. Las aplicaciones actualmente tienden a ser arquitecturas de microservicios, donde Docker y Kubernetes son tecnologías que facilitan este tipo de diseño
- *Contenedor*: Un contenedor es una unidad de software ligera y portátil que incluye todo lo necesario para ejecutar una aplicación: el código, las bibliotecas, las dependencias y las configuraciones. Permiten ejecutar aplicaciones de manera consistente en diferentes entornos, ya que aíslan la aplicación del sistema operativo subyacente. [25].
- *Docker*: Es una tecnología que permite crear, ejecutar y desplegar contenedores [25].
- *Kubernetes*: Es una plataforma de orquestación de contenedores de código abierto que automatiza la implementación, administración y escalado de aplicaciones en contenedores. Permite gestionar clústeres de servidores para garantizar alta disponibilidad y eficiencia en la ejecución de aplicaciones [26].
- *Pod*: en kubernetes, un pod es la unidad mínima de ejecución, dentro de él pueden existir uno o más contenedores, los volúmenes, recursos de red y configuración.
- *Helm*: Es un gestor de paquetes para Kubernetes. Facilita la instalación, configuración y actualización de aplicaciones dentro de un clúster de Kubernetes utilizando "charts", que son plantillas predefinidas para desplegar aplicaciones complejas.[27]

Figura 13 - Tecnologías usadas dentro de Secure Code Box - Fuente: https://siweheee.medium.com/deploy-your-programs-onto-minikube-with-docker-and-helm-a68097e8d545

6.2.2. Qué es Secure Code Box

SCB es una herramienta de orquestación de seguridad diseñada para aprovechar al máximo las capacidades de k8s, como la administración y la orquestación de contenedores [28]. Dentro de estos contenedores, SCB puede seleccionar entre más de 20 herramientas de seguridad y realizar distintos análisis para encontrar vulnerabilidades en el código, en aplicaciones desplegadas e incluso hasta en infraestructuras de red.

SCB se integra profundamente con la API de Kubernetes mediante el uso de Custom Resource Definitions (CRDs) [29] [30] extendiéndose para crear endpoints específicos que permiten gestionar la herramienta.

Para poder utilizar la API de kubernetes con una herramienta de línea de comandos, es posible usar *kubectl*. kubectl gestiona la api de k8s permitiendo ejecutar comandos de manera simple y efectiva. En el siguiente gráfico se muestran algunos ejemplos de uso.

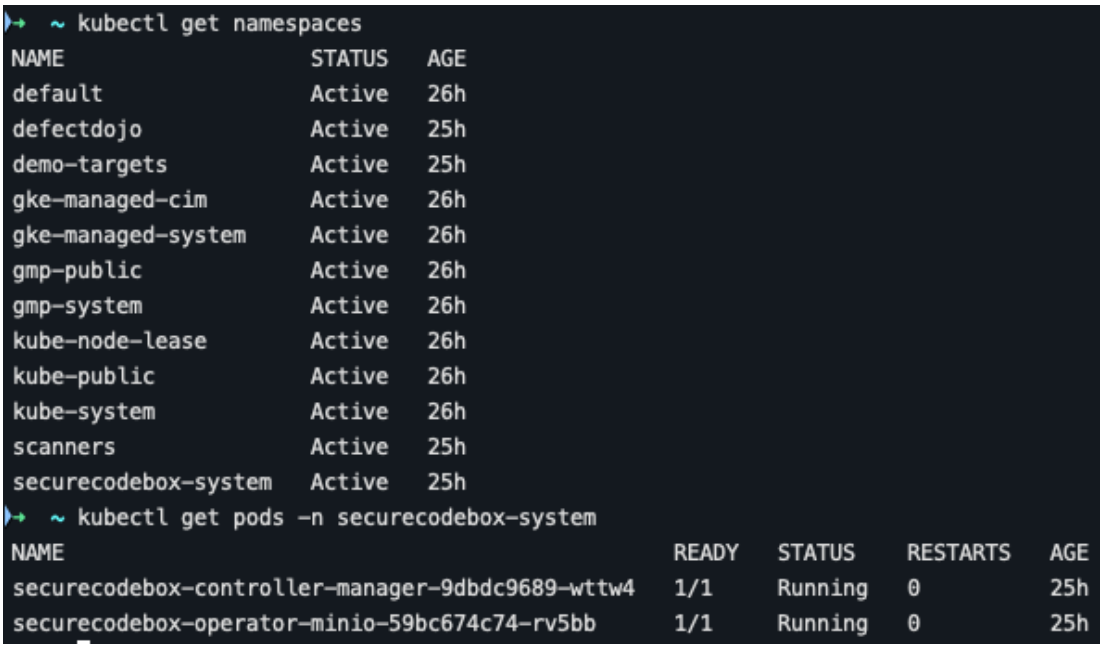

Figura 14 - Enumeración de los namespaces y pods dentro de un cluster de GCP

Mediante el uso de archivos de configuración que están en un formato legible (están basados en formato YAML), se pueden armar estas configuraciones y usando la kubectl, se pueden ejecutar (o aplicar) estos cambios. Mediante el uso de estas herramientas, se facilita la automatización y administración de escaneos de seguridad. En el siguiente gráfico se muestra un archivo básico de configuración para realizar un scan con nmap a un sitio web.

```yaml
apiVersion: "execution.securecodebox.io/v1"
kind: Scan
metadata:
  name: "nmap-scanme.nmap.org"
spec:
  scanType: "nmap"
  parameters:
    - scanme.nmap.org
```

Figura 15 - Archivo de configuración de un scan nmap a el sitio web https://scanme.nmap.org con SCB

6.2.2.1. Características principales de SCB

- **Es un proyecto de OWASP**: Secure Code Box forma parte de la comunidad OWASP, reconocida por promover la seguridad de aplicaciones y software [28].
- **Open Source**: Es una herramienta de código abierto, lo que fomenta su personalización y uso comunitario [28].
- **Basado en Kubernetes**: Se apoya en la infraestructura de Kubernetes para su escalabilidad y orquestación.
- **Amplia integración de herramientas**: Integra más de 20 herramientas de seguridad, cubriendo diversas necesidades de análisis [31].
- **Compatibilidad con gestores de vulnerabilidades**: Permite integrar plataformas como DefectDojo y ElasticSearch para una gestión centralizada de vulnerabilidades [32].
- **Notificaciones personalizables**: Compatible con servicios de mensajería como Slack, correo electrónico y Microsoft Teams [33].

6.2.2.2. Características funcionales de SCB

- **Integración con múltiples herramientas de seguridad**: SCB ejecuta escaneos con herramientas específicas dentro de contenedores, lo que permite personalizar y optimizar los análisis según las necesidades del usuario [31] .

- **Gestión mediante la API de Kubernetes**: La herramienta se administra con kubectl, permitiendo una gestión eficiente y automatizada de escaneos y configuraciones.
- **Configuración basada en YAML**: Los escaneos y parámetros se definen en archivos YAML, lo que facilita su reutilización y control de versiones.
- **Escaneos programados**: SCB soporta la ejecución de escaneos periódicos mediante CronJobs o frecuencias determinadas, asegurando monitoreo continuo [34].
- **Escaneos en cascada**: Es capaz de iniciar escaneos adicionales basados en los resultados de análisis anteriores, optimizando la detección de vulnerabilidades en cadenas complejas [35].
- **Service y container autodiscovery**: Detecta automáticamente servicios y contenedores en el clúster de Kubernetes, facilitando la integración de nuevos elementos [36].
- **Hooks personalizables**: Los hooks permiten integrar SCB con otros sistemas o personalizar la respuesta ante eventos, como el envío de resultados a DefectDojo o plataformas de notificación [37].

Estas características hacen que SBC sea una herramienta con un gran potencial para usarse en ambientes DevSecOps. En esta tesis de maestría, vamos a utilizar SBC para orquestar las distintas herramientas de seguridad.

6.2.2.3. Dificultades en el uso de SCB como orquestador de seguridad

Si bien SCB tiene varias ventajas, pero también hay ciertas desventajas o desafíos que puede tener su uso, algunas de estas son:

- *Curva de aprendizaje*: Si bien el uso una vez conocida la herramienta puede ser simple, para usuarios nuevos, la configuración inicial puede ser compleja, especialmente si no tienen experiencia previa con k8s, Helm, o los principios de DevSecOps.

- *Configuración avanzada*: Adaptar SCB a entornos personalizados, como integraciones con herramientas específicas o flujos de trabajo únicos, puede requerir tiempo y esfuerzo.
- *Dependencia de Kubernetes*: SCB está diseñado para correr en clusters de Kubernetes, esto genera una gran dependencia con está tecnología
- *Consumo de recursos*: Al depender de k8s, SCB puede consumir una cantidad significativa de recursos del sistema, especialmente si se ejecutan múltiples escáneres simultáneamente.
- *Costos asociados*: Si se utiliza un proveedor de nube, los costos de ejecutar y mantener el cluster pueden ser elevados.
- *Cobertura de herramientas*: Aunque SCB soporta múltiples escáneres, puede haber herramientas de escaneo que no estén integradas de manera nativa, requiriendo personalización para incluirlas.
- *Falta de soporte comercial*: Al ser una solución open-source, el soporte puede ser limitado a la comunidad y documentación, lo que podría ser un problema en escenarios críticos.
- *Ruido en los resultados*: Al igual que lo mencionado en los capítulos anteriores, es posible que tantos escáneres de seguridad pueden generar resultados similares, o falsos positivos, lo que significa que SCB podría abrumar a los usuarios con demasiadas alertas si no se configuran adecuadamente los filtros o análisis posteriores.
- *Curva técnica*: Adaptar SCB a necesidades complejas (por ejemplo, añadir escáneres no soportados, integrar workflows personalizados, o extender sus capacidades) puede requerir habilidades avanzadas en programación y DevOps.

7. Interfaz gráfica del usuario

7.1. Como integrar una interfaz gráfica

Una característica que hace interesante el uso de SCB es la integración con herramientas de visualización para la gestión de vulnerabilidades como lo es DefectDojo o Elastic Search con Kibana.

Por medio de un componente llamado Persistent Hook [38] en SCB, es posible mantener una comunicación continua entre los scans y las herramientas de visualización de vulnerabilidades.

Las herramientas principales de visualización es DefectDojo, un gestor de vulnerabilidades y ElasticSearch, que es un motor de búsqueda, que integrado con soluciones como Kibana, pueden utilizarse también como interfaz gráfica.

En este trabajo solo nos vamos a focalizar en el uso de DefectDojo con SCB.

7.2. DefectDojo

Figura 16 - Logo de DefectDojo [39]

DefectDojo (en adelante DD) es una herramienta de código abierto diseñada para ayudar en la gestión y seguimiento de vulnerabilidades. Al igual que SCB, es un proyecto gestionado por la comunidad de OWASP [40], y su objetivo es centralizar y automatizar la gestión de hallazgos de seguridad provenientes de distintas herramientas de seguridad, algunas de las cuales se encuentran integradas con SCB.

Según la documentación oficial, traducida al español:

DD es una plataforma que se utiliza en ambientes DevSecOps. DD funciona como un agregador y una interfaz única para gestionar herramientas de seguridad. DD tiene funciones inteligentes para mejorar y ajustar los resultados de sus herramientas de seguridad, incluida la capacidad de fusionar hallazgos, recordar falsos positivos y filtrar duplicados. [41]

Entre sus características principales se encuentra:

- *Interfaz gráfica de gestión única*: DD tiene una interfaz desarrollada en python que permite la gestión centralizada para todos los hallazgos de seguridad de una sola plataforma de gestión
- *Integración con múltiples herramientas de seguridad*: Permite importar resultados de múltiples herramientas de seguridad. Desde herramientas de código estático como Semgrep, herramientas de análisis dinámico como OWASP Zap o Nikto, y herramientas de escaneo de infraestructura como Nessus, SonarQube, nmap, etc. Son más de 180 herramientas de seguridad que soporta[11]
- *Proporciona un flujo de trabajo para las vulnerabilidades*: Los hallazgos de seguridad, cuando son reportadas en DD, tienen un flujo de trabajo en el cual se pueden gestionar para dar el correcto tratamiento. Desde gestionar los falsos positivos, las prioridades, las excepciones, o incluso la aceptación del riego
- *Gestión de usuarios*: DD tiene una segregación de permisos que puede permite implementar el principio de menor privilegio dentro de la plataforma
- *Gestión de duplicados y hallazgos remediados*: Los hallazgos de seguridad que ya han sido reportados, pueden ser detectados para no ser reportados nuevamente. De la misma manera, en ciertas herramientas, como las herramientas de código estático, si el código con el problema de seguridad ha sido removido o la vulnerabilidad ha

[11] Más detalle sobre las integraciones se puede ver en el siguiente sitio web: https://defectdojo.com/integrations

sido remediada, se puede configurar para automáticamente definir como una vulnerabilidad remediada.

- *Reportes personalizables y métricas*: La plataforma tiene un dashboard en el que se pueden ver las vulnerabilidades y también tiene la posibilidad de crear reportes personalizables.
- *Integraciones con otras herramientas*: DD tiene la posibilidad de ser integrado con herramientas de gestión de proyectos como Jira, de Atlassian o incluso con sistemas de mensajerías como Slack o comunicación por email
- *Extensibilidad*: El uso de DD se puede extender y realizar distintos tipos de automatizaciones mediante el uso de la API que tiene integrada

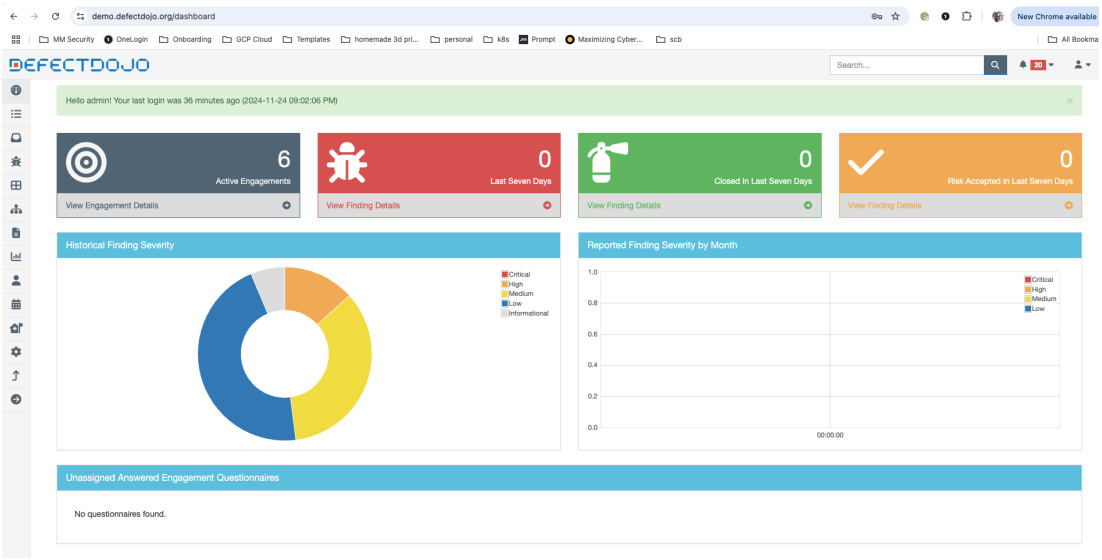

Figura 17 - Captura tomada de https://demo.defectdojo.org . Las credenciales se pueden obtener en https://defectdojo.github.io/django-DefectDojo/getting_started/demo/

7.2.1. Gestión de los hallazgos de seguridad

En DD la gestión de los hallazgos se organiza jerárquicamente y sigue un flujo lógico que permite centralizar todos los aspectos de los análisis de seguridad. A continuación se mencionan los distintos elementos que lo componen:

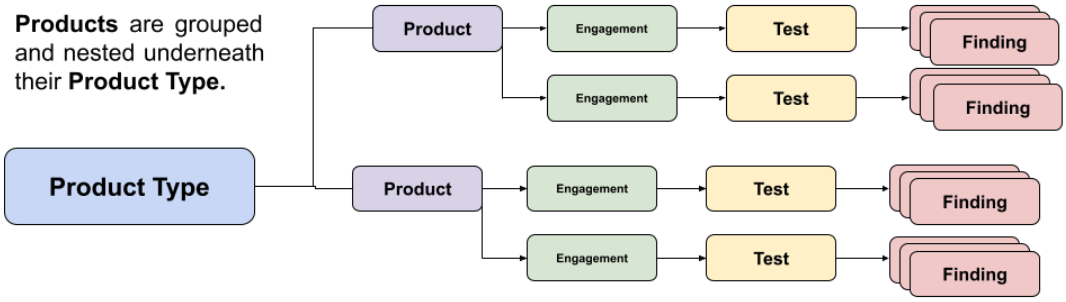

Figura 18 - Estructura de cómo se organizan los hallazgos de seguridad en DefectDojo [42]

- *Product Type (Tipo de producto)*: es el nivel jerárquico que engloba las características generales de un producto. Ejemplos pueden ser Aplicaciones Web, Infraestructura Cloud, Servicios tercerizados. Sirve para agrupar productos por sus características generales [42].
- *Product (Producto)*: Es un elemento dentro del Tipo de Producto. Representa una aplicación, servicio o componente. Un ejemplo puede ser API de Servicio [42].
- *Engagement (Iniciativa)*: Es un esfuerzo temporal para evaluar un producto. Puede corresponder a un sprint, auditoría de seguridad, o un análisis continuo. Suele tener información como fecha de inicio y fin, tipo de análisis, entre otros datos [42].
- *Test (Pruebas)*: Corresponden a una ejecución específica de una o un conjunto de herramientas que está realizando un análisis de seguridad [42].
- *Finding (hallazgos)*: Son los resultados de las pruebas y representan vulnerabilidades o problemas de seguridad. Dentro de los findings podemos tener campos que indican el tipo de vulnerabilidad, la severidad, la descripción, pasos para la remediación, estado, etc. [42].

7.2.2. Gestión de los usuarios, roles y permisos

DD está diseñado para garantizar el acceso a la información a las personas que necesitan según su función o necesidades de su trabajo preservando el principio de mínimo privilegio [43].

DD tiene disponible 4 tipos de permisos:

- *Usuarios* pueden ser designados como miembros de productos o product types
- *Configuration permissions* para acceder a las configuraciones de DDw
- *Global Roles* para acceder a todos los productos o product types
- *SuperUsers* que son los roles administrativos de DD, permiten acceder a los productos, product types y configuraciones

Dentro de cada product o product type, existen distintos permisos que permiten al usuario tener acceso *solo lectura (reader)*, *escritura (writer)*, *mantenimiento (maintainer)*, que tienen los permisos de escritura y pueden modificar un product type pero no pueden borrar la información. Dentro del product o product type también pueden ser *dueños (owners)*, que tienen el mayor nivel de control y por último, pueden tener el permiso *API importer* que tienen acceso limitado a los endpoints de la API.[12]

Los roles globales permiten ver e interactuar con todos los componentes de DD, estos son Product, Product Type, Engagements, Tests y Findings.

Los grupos sirven para agrupar usuarios que necesitan el mismo nivel de permiso, y poder asignarlo a estos.

Los superusers no tienen limitaciones dentro de DD. Pueden cambiar todas las configuraciones y los accesos a los usuarios y grupos dentro de DD.

Los permisos de configuraciones pueden utilizarse para modificar configuraciones, roles de grupos y usuarios.

Con este nivel de granularidad de los permisos, permite que los usuarios o grupos de usuarios vean solo los problemas de seguridad que

[12] Un mayor nivel de detalle de los permisos se puede ver en el siguiente link: https://support.defectdojo.com/en/articles/8955600-user-permission-charts#h_ee05c5f5df

necesitan, de manera que puedan acceder solo a la información que necesitan.

7.2.3. Flujos de Trabajo

7.2.3.1. Findings (Hallazgos) de Seguridad

Las herramientas de seguridad integradas en DD reportan hallazgos de seguridad. Dependiendo de la herramienta, existen distintos tipos de findings, por ejemplo de infraestructura de red, de aplicaciones, de bases de datos, etc. Cada una de estas herramientas provee información que le permite al usuario (que normalmente es la persona que desarrolló la aplicación o en encargado de mantenerla) mitigar la potencial vulnerabilidad. En la siguiente imagen se muestra un ejemplo de un findings de seguridad

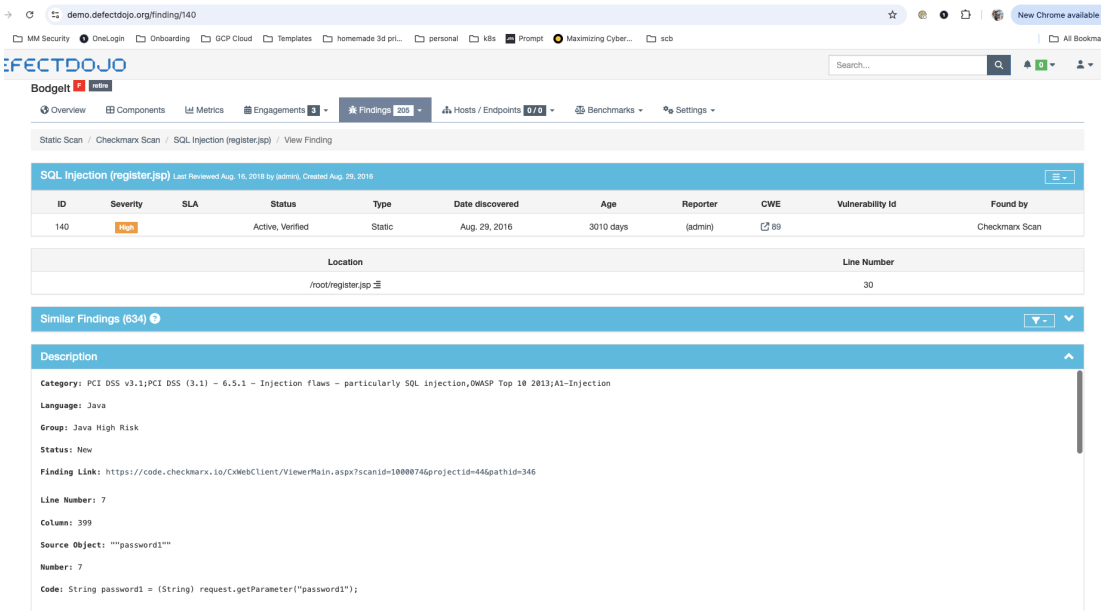

Figura 19 - Estructura de un hallazgo de seguridad

Cada finding puede contener datos que sirvan a la persona asignada para poder dar el correcto tratamiento de la vulnerabilidad.

Algunos de estos campos son el título, la descripción, severidad, el impacto, la mitigación, referencias, notas, etc. Además de los campos requeridos, es posible agregar campos personalizables [44].

7.2.3.2. Flujo de trabajo (Workflow) de un Finding

Descubrimiento del Findings

Los findings de seguridad pueden ser reportados de manera manual o usando la API que DD tiene disponible. Cada finding es posible agruparlas en distintos product types, products, engagements y tests. A su vez, los findings pueden ser asignados a una persona o grupo para que sean correctamente tratados [44].

Triaging de vulnerabilidades

Las vulnerabilidades necesitan ser validadas cuando son descubiertas. A cada vulnerabilidad se le puede aplicar los siguientes estados [44]:

- *False Positive (Falso Positivo)*: Cuando una herramienta detecta una vulnerabilidad pero no es una amenaza activa en el ambiente
- *Out of Scope (Fuera del alcance)*: Es una vulnerabilidad activa, pero irrelevante con respecto al esfuerzo que se requiere para remediar
- *Risk Accepted (Riesgo Aceptado)*: Es una vulnerabilidad activa, pero no es una prioridad remediar y se acepta el riesgo de que un actor malicioso explote este problema de seguridad
- *Under Review (En revisión)*: Puede o no ser una vulnerabilidad activa, está en revisión
- *Mitigated (Mitigado)*: Es una vulnerabilidad que no está activa, está remediada

7.2.4. Integración con herramientas de Seguridad

7.2.4.1. Integración nativa con herramientas de seguridad

Las herramientas de seguridad reportan lo encontrado de distintas maneras. Alguna de estas herramientas, tienen integración nativa con DD mediante conectores. Para utilizar estos conectores es necesario proveer de

información para poder realizar una comunicación efectiva entre la herramienta y DD.

Existe también la posibilidad de utilizar la API que tiene disponible DD. Esta API tiene una gran flexibilidad y permite desde la creación de findings, importar y exportar reportes, crear usuarios, grupos y permisos, product types, products, etc.

7.2.4.2. APIs y parsers de seguridad (analizadores sintácticos)

Un inconveniente que tiene el uso de distintas herramientas de seguridad es que cada una tiene un formato distinto de acuerdo a las necesidad y la información que entrega. Es por esto que resulta complejo agregar campo por campo de un reporte dentro de DD si uno lo quiere hacer de manera manual por medio de la API. Es por esto, que DD tiene una serie de analizadores sintácticos (parsers) que dependiendo de la herramienta que genera el reporte y el tipo de reporte, DD automáticamente lee los campos y puede generar los correspondientes findings de seguridad. Este punto es determinante a la hora de seleccionar un gestor de vulnerabilidades, ya que resulta muy complejo adaptar los reportes de seguridad al formato de DD. Según el sitio oficial de DD, actualmente tiene más de 180 analizadores sintácticos[13] (parser) para poder integrar distintas herramientas de seguridad. En el siguiente gráfico, se muestra una porción de los distintos parser que es posible integrar dentro de DD.

[13] Fuente: https://defectdojo.com/integrations

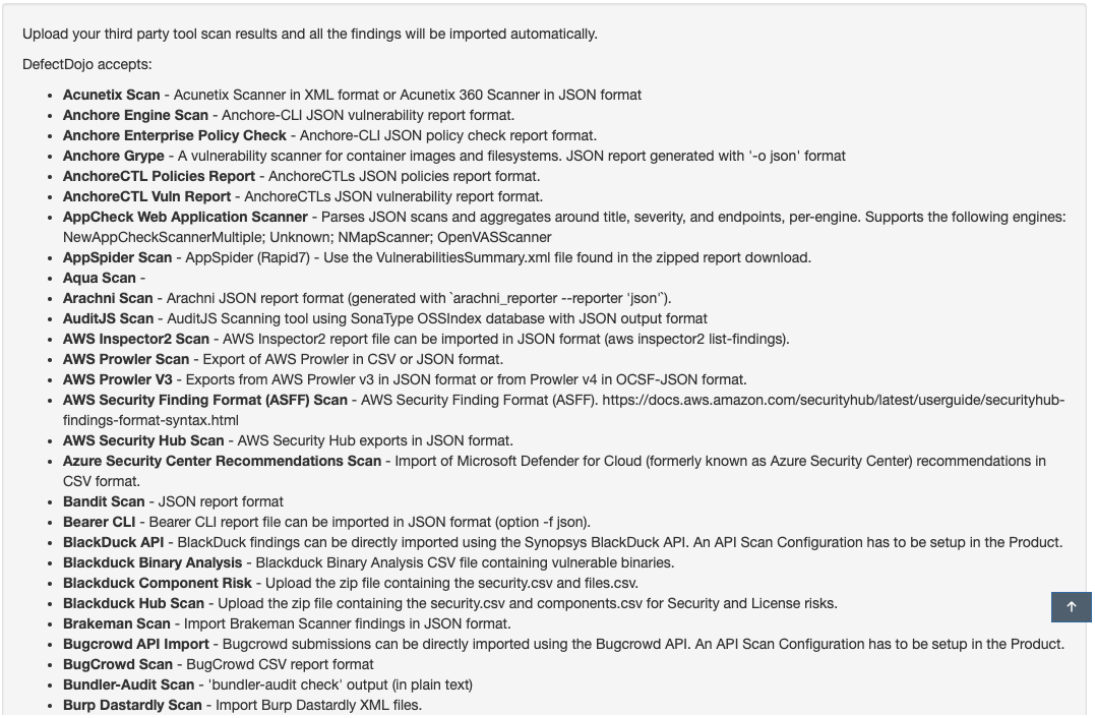

Figura 20 - Dentro de la interfaz de DD, se enumeran algunas de las integraciones posibles con distintas herramientas de seguridad.

De igual manera, existe la posibilidad de que una herramienta tenga un formato que no está identificado dentro de DD, y es posible agregar el/los findings de manera manual o mediante un parser genérico.

7.2.4.3. Detección de Findings duplicados

Otra característica fundamental dentro de DD es la posibilidad de detectar duplicados. Muchas herramientas de seguridad ejecutan análisis de seguridad múltiples veces en la misma aplicación o infraestructura para detectar cambios. Estas herramientas suelen correr de manera periódica o en algunos casos dentro de los pipelines de los repositorios cada vez que un desarrollador pública código dentro de una de las ramas del repositorio. La capacidad de detectar duplicados permite que la vulnerabilidad que ya ha sido reportada, no sea publicada de nuevo y de esta manera no generar ruido de tener múltiples veces el mismo problema de seguridad [45].

7.2.4.4. Remediación automática

Cuando estamos trabajando con herramientas que hacen análisis de código estático, si una vulnerabilidad que ha sido reportada previamente en un repositorio específico, no es detectado cuando se ejecuta el análisis nuevamente, existe la posibilidad de que automáticamente cambie el estado de la vulnerabilidad a remediada, de esta manera se automatiza el flujo de remediar problemas de seguridad.

8. Implementación de un gestor de vulnerabilidades

Para poder mostrar como se puede implementar un gestor de vulnerabilidades en un ambiente DevSecOps se ha creado un laboratorio y se han ejecutado las herramientas anteriormente mencionadas.

Para poder reproducir de manera particular este laboratorio, se enumeran los pasos necesarios para poder realizar el laboratorio:

1. Instalar un cluster de Kubernetes.
2. Instalar la línea de comandos de kubernetes *kubectl*
3. Instalar el gestor de paquetes de kubernetes *Helm*
4. Instalar Secure Code Box
5. Instalar DefectDojo
6. Configurar DefectDojo
7. Instalar los Scanners
8. Instalar el Persistent Hook de DefectDojo
9. Ejecutar SAST sobre un repositorio
10. Desplegar una aplicación para realizar análisis DAST
11. Ejecutar DAST sobre una aplicación desplegada

8.1. Creación del Laboratorio o Prueba de concepto

8.1.1. Instalar un Clúster de Kubernetes

Para poder realizar el laboratorio, es necesario tener un clúster de kubernetes. El cluster puede ser creado de forma local con herramientas como docker, minikube o Kind o bien en cualquiera de las nubes más populares como AWS, Azure, GCP u Oracle, entre otros, se pueden crear clusters de kubernetes. Debido a la cantidad de opciones disponibles, solo se explica como instalar minikube que sería el ambiente adecuado para un entorno de desarrollo y un cluster de kubernetes en Google Cloud Platform (GCP) usando Google Kubernetes Engine (GKE)[14] que sería un ambiente

[14] https://cloud.google.com/kubernetes-engine?hl=es-419

adecuado para un entorno productivo. Para trabajar en GCP, es necesario un paso adicional que es instalar además la línea de comandos de GCP. En los siguientes pasos, solo se muestra la instalación de SecureCodeBox y DefectDojo en GCP pero no cambia en nada si se realiza en otro cluster como minikube. Solo hay que tener en cuenta los requerimientos de CPU, memoria y almacenamiento que pueden ser un limitante a la hora de escalar estos servicios.

8.1.1.1. Opción 1: Instalar minikube

En el sitio web oficial de minikube[15] se indican los requerimientos mínimos para poder instalarlo

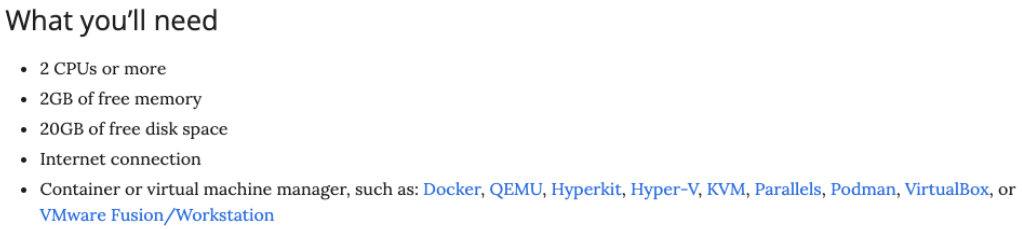

Figura 21 - Requerimientos mínimos para instalar minikube - Fuente: https://minikube.sigs.k8s.io/docs/start

Se instala la aplicación por los haciendo click en el link indicado de acuerdo al sistema operativo y arquitectura de la pc donde se realizará la prueba de concepto

[15] Sitio oficial de minikube: https://minikube.sigs.k8s.io/docs/start

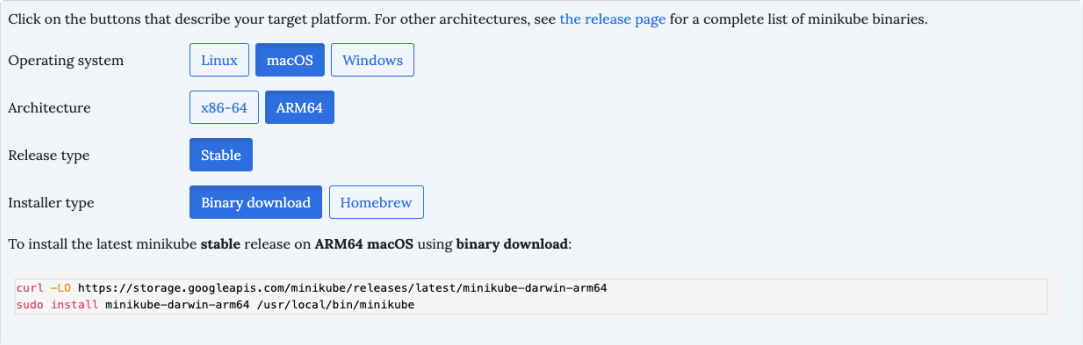

Figura 22 - Distintas opciones para instalar minikube según el sistema operativo - Fuente: https://minikube.sigs.k8s.io/docs/start

Una vez instalado minikube, se puede Iniciar con el siguiente comando:

```
minikube start
```

De esta manera, se logra crear un clúster de k8s de manera local. Luego se pueden instalar las herramientas necesarias para continuar la instalación de SCB

8.1.1.2. Opción 2: Instalar un cluster de kubernetes en GCP

Instalar un cluster de kubernetes en GCP [46] se puede realizar por medio de la interfaz gráfica, o mediante línea de comandos. Para instalar un cluster por medio de la interfaz gráfica, se puede realizar con el siguiente link que explica como realizarlo de manera detallada

Por línea de comandos, es necesario tener instalado el SDK gcloud, para esto se realiza de la siguiente forma:[16]

- Es necesario python 3.11 o mayor
- El instalador se descarga del sitio oficial, dependiendo del sistema operativo y arquitectura

[16] Linea de comandos de GCP: https://cloud.google.com/sdk/docs/install

2. Download one of the following:

Note: To determine your machine hardware name, run `uname -m` from a command line.

Platform	Package	Size	SHA256 Checksum
macOS 64-bit (x86_64)	google-cloud-cli-darwin-x86_64.tar.gz	53.9 MB	9d1af0d2b5e7c8a30145831b99ce46272fc0aba754d7d23dcd3ede414b9ff75a
macOS 64-bit (ARM64, Apple M1 silicon)	google-cloud-cli-darwin-arm.tar.gz	53.9 MB	d986d0c6531be038b71218f8e7e666c5b4d18ef580d6a063550406ed07e460f9
macOS 32-bit (x86)	google-cloud-cli-darwin-x86.tar.gz	52.6 MB	e98b4bf6449b9207ff5c4bd3df12a1b38ffa0f498274b26f2d80cfe7fb124f81

Figura 23 - Distintas opciones para instalar minikube - Fuente: https://minikube.sigs.k8s.io/docs/start

- Instalar el instalador. Dependiendo del sistema operativo, este comando puede cambiar. En macOS el comando es el siguiente:

```
./google-cloud-sdk/install.sh
```

Una vez instalado gcloud, es necesario autenticarse en GCP. El siguiente comando redirecciona al navegador y pide las credenciales para la autenticación.

```
➜  templates git:(main) ✗ gcloud auth login
Your browser has been opened to visit:

    https://accounts.google.com/o/oauth2/auth?response_type=code&cl
```

Figura 24 - Comando para autenticarse con google usando línea de comandos

Ahora se crea el cluster en GCP usando GKE. Hay algunos parámetros mínimos que hay que definir[17].

[17] NOTA: Tener en cuenta que para el propósito de este trabajo, el cluster está sobredimensionado, se puede realizar el mismo laboratorio con menores requerimientos de hardware requerimientos

```
➜  ~ PROJECT_ID=secure-code-box
➜  ~ CLUSTER_NAME=securecodebox-cluster
➜  ~ ZONE=us-central1-a
➜  ~ gcloud container clusters create $CLUSTER_NAME \
    --project $PROJECT_ID \
    --zone $ZONE \
    --num-nodes 3 --machine-type e2-standard-4
```

Figura 25 - Como crear un cluster de GKE en GCP

Esta debería ser la respuesta del cluster creado:

```
➜  templates git:(main) ✗ gcloud container clusters create $CLUSTER_NAME \
    --project $PROJECT_ID \
    --zone $ZONE \
    --num-nodes 3 --machine-type e2-standard-4
Default change: VPC-native is the default mode during cluster creation for versions greater than 1.21.0-gke.1500. To create adv
anced routes based clusters, please pass the `--no-enable-ip-alias` flag
Note: Your Pod address range (`--cluster-ipv4-cidr`) can accommodate at most 1008 node(s).
Creating cluster securecodebox-cluster in us-central1-a... Cluster is being health-checked (Kubernetes Control Plane is healthy
)...done.
Created [https://container.googleapis.com/v1/projects/secure-code-box/zones/us-central1-a/clusters/securecodebox-cluster].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-central1-a/secu
recodebox-cluster?project=secure-code-box
kubeconfig entry generated for securecodebox-cluster.
NAME                  LOCATION       MASTER_VERSION      MASTER_IP      MACHINE_TYPE   NODE_VERSION        NUM_NODES  STATUS
securecodebox-cluster  us-central1-a  1.30.5-gke.1443001  35.232.78.116  e2-standard-4  1.30.5-gke.1443001  3          RUNNING
```

Figura 26 - Línea de comando que muestra cómo es la creación de un cluster GKE en GCP

8.1.2. Instalar la línea de comandos de kubernetes *kubectl*

Dependiendo del sistema operativo, se debe elegir la opción a instalar:

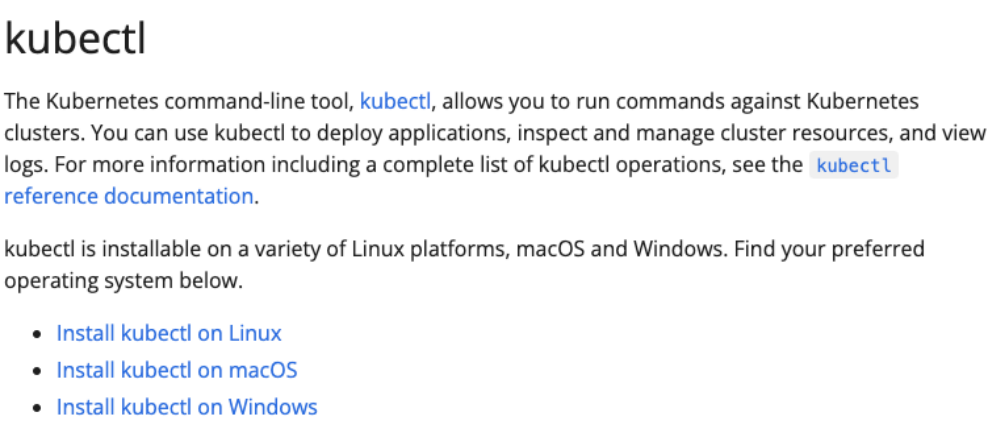

Figura 27 - Como instalar la línea de comandos kubectl - Fuente: https://kubernetes.io/docs/tasks/tools/

Seguir los pasos indicados en la página oficial para instalar la línea de comandos kubectl

8.1.3. Instalar el gestor de paquetes de Kubernetes *Helm*

Desde la página oficial de helm descargar e instalar la aplicación:

Figura 28 - Como instalar la línea de comandos helm - Fuente: https://helm.sh/

8.1.4. Instalar Secure Code Box

Una vez instalado el cluster de kubernetes, helm y kubectl, instalar Secure Code Box es tan simple como ejecutar el siguiente comando[18]:

```
→  templates git:(main) ✗ helm --namespace securecodebox-system upgrade --install --create-namespace securecodebox-operator
 oci://ghcr.io/securecodebox/helm/operator
```

Figura 29 - Instalar Secure Code Box en un cluster de k8s usando helm

La línea de comando debería entregar un texto similar al siguiente:

```
→  templates git:(main) ✗ helm --namespace securecodebox-system upgrade --install --create-namespace securecodebox-operator oci://ghcr.io/secur
ecodebox/helm/operator
Release "securecodebox-operator" does not exist. Installing it now.
Pulled: ghcr.io/securecodebox/helm/operator:4.10.0
Digest: sha256:41e3b1c19c6e4a3a48a5a06fbaaf48a6d3fb9fdfb922296db64a494b1437312f
NAME: securecodebox-operator
LAST DEPLOYED: Wed Nov 27 15:30:40 2024
NAMESPACE: securecodebox-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
secureCodeBox Operator Deployed 🚀

The operator can orchestrate the execution of various security scanning tools inside of your cluster.
You can find a list of all officially supported scanners here: https://www.securecodebox.io/
The website also lists other integrations, like persisting scan results to DefectDojo or Elasticsearch.

The operator send out regular telemetry pings to a central service.
This lets us, the secureCodeBox team, get a grasp on how much the secureCodeBox is used.
The submitted data is chosen to be as anonymous as possible.
You can find a complete report of the data submitted and links to the source-code at: https://www.securecodebox.io/docs/telemetry
The first ping is send one hour after the install, you can prevent this by upgrading the chart and setting `telemetryEnabled` to `false`.
```

Figura 30 - Una vez instalado SCB en el namespace securecodebox-system, la línea de comando entrega este resultado

Para verificar la instalación se puede ver el namespace securecodebox-system creado ejecutando el siguiente comando:

[18] En el siguiente video se muestra como instalar SCB en un K8s de GCP: https://drive.google.com/file/d/1A28eL77ByuVYW7JJgxqcpH_YTcbzivjE/view?usp=drive_link

```
→  templates git:(main) ✗ kubectl get namespaces
NAME                   STATUS   AGE
default                Active   13m
gke-managed-cim        Active   12m
gke-managed-system     Active   12m
gmp-public             Active   12m
gmp-system             Active   12m
kube-node-lease        Active   13m
kube-public            Active   13m
kube-system            Active   13m
securecodebox-system   Active   72s
```

Figura 31 - Listar los namespaces de una clúster GKE por línea de comando

Para visualizar los pods creados, se puede ejecutar el siguiente comando:

```
→  templates git:(main) ✗ kubectl get pods -n securecodebox-system
NAME                                                READY   STATUS    RESTARTS   AGE
securecodebox-controller-manager-9dbdc9689-8z78j    1/1     Running   0          3h25m
securecodebox-operator-minio-59bc674c74-pc2vr       1/1     Running   0          3h25m
```

Figura 32 - Listar los pods namespace securecodebox-system. Se puede observar el operador y el bucket mini.io

En ese comando se puede visualizar que hay 2 pods instalados en el namespace securecodebox-system, uno es el controlador y el otro es un pod que se usa para almacenar la información de los scans que se llama mini.io.

Para poder visualizar la interfaz gráfica de mini.io, se puede generar un proxy por ejemplo en el puerto 9000 de manera local, ya que kubernetes no expone el servicio a internet por defecto, se puede realizar con el siguiente comando:

```
→  ~ kubectl port-forward -n securecodebox-system service/securecodebox-operator-minio 9000:9001
Forwarding from 127.0.0.1:9000 -> 9001
Forwarding from [::1]:9000 -> 9001
```

Figura 33 - Crear un proxy para acceder al bucket mini.io. Se expone el puerto 9000 de manera local (localhost)

Se ingresa a la url http://localhost:9000, y se accede a un portal que se debe autenticar, el usuario por defecto de mini.io es “admin” y la

contraseña es “password”. Luego de autenticarse, se puede visualizar la siguiente interfaz:

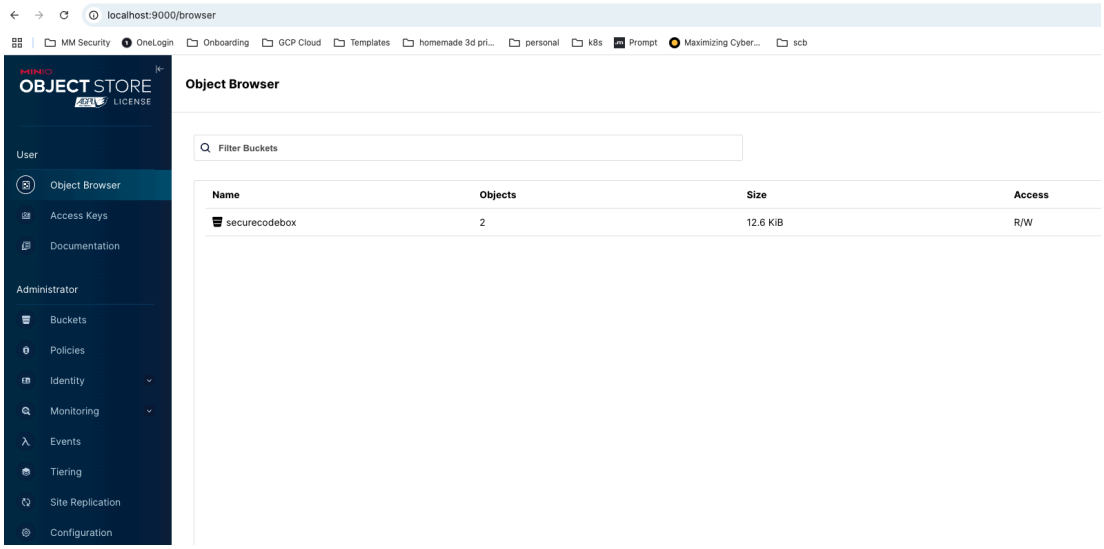

Figura 34 - Interfaz del bucket mini.io

8.1.5. Instalar DefectDojo

Para instalar DefectDojo, en necesario descargar el repositorio, creamos un namespace llamado defectdojo para separar los ambientes y que el cluster esté ordenado y por último instalamos DefectDojo[19].

[19] NOTA: Si el ambiente que se ha creado es en minikube, el host es el siguiente: defectdojo.default.minikube.local

```
# Descargamos el repositorio de DefectDojo
git clone https://github.com/DefectDojo/django-DefectDojo
cd django-DefectDojo

# Obtenemos los charts de helm
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update
helm dependency update ./helm/defectdojo

# Creamos un namespace llamado defectdojo
kubectl create namespace defectdojo

# Instalamos el helm chart de DefectDojo
helm upgrade --install \
  defectdojo \
  ./helm/defectdojo \
  --set django.ingress.enabled=true \
  --set django.ingress.activateTLS=false \
  --set createSecret=true \
  --set createRedisSecret=true \
  --set createPostgresqlSecret=true \
  --set host="defectdojo-django.defectdojo.svc" \
  --set "alternativeHosts={localhost, \
  defectdojo.defectdojo.cluster.local:8080, \
  defectdojo-django.defectdojo.svc}" -n defectdojo
```

Figura 35 - Instalar DefectDojo en el namespace defectdojo usando helm

Este comando va a instalar DefectDojo en el namespace defectdojo. Luego de unos minutos, se pueden ver los servicios de DefectDojo corriendo normalmente:

```
→  django-DefectDojo git:(master) ✗ kubectl get pods -n defectdojo
NAME                                          READY   STATUS    RESTARTS   AGE
defectdojo-celery-beat-5fdb485b7d-xxh5r       1/1     Running   0          3m16s
defectdojo-celery-worker-847457b5d8-f4xmz     1/1     Running   0          3m16s
defectdojo-django-6f9477c788-66pkj            2/2     Running   0          3m15s
defectdojo-initializer-2024-11-27-15-38-srqgr 1/1     Running   0          3m14s
defectdojo-postgresql-0                       1/1     Running   0          3m15s
defectdojo-redis-master-0                     1/1     Running   0          3m15s
```

Figura 36 - DefectDojo necesita de varios servicios disponibles para operar correctamente, se enumeran cuando se visualiza los pods creados en el namespace defectdojo por helm

8.1.6. Configurar DefectDojo

DD genera unas credenciales para el usuario admin de manera aleatoria, para poder acceder a esas credenciales, se requiere correr el siguiente comando:

```
→  django-DefectDojo git:(master) ✗ echo "DefectDojo admin password: $(kubectl \
    get secret defectdojo \
    --namespace=defectdojo \
    --output jsonpath='{.data.DD_ADMIN_PASSWORD}' \
    | base64 --decode)"
```

Figura 37 - Las credenciales para acceder a DD se generan de manera automática cuando se inicia el servicio. Con este comando se visualiza como obtener esas credenciales

Para poder visualizar la interfaz gráfica, se puede generar un proxy en la computadora de manera local, ya que el servicio no está expuesto en internet, se puede realizar con el siguiente comando:

```
→  ~ kubectl port-forward -n defectdojo service/defectdojo-django 8080:80
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

Figura 38 - Crear un proxy para acceder a Defect Dojo. Se expone el puerto 8080 de manera local (localhost)

Ahora se puede acceder a la interfaz gráfica de manera local ingresando la url http://localhost:8080.

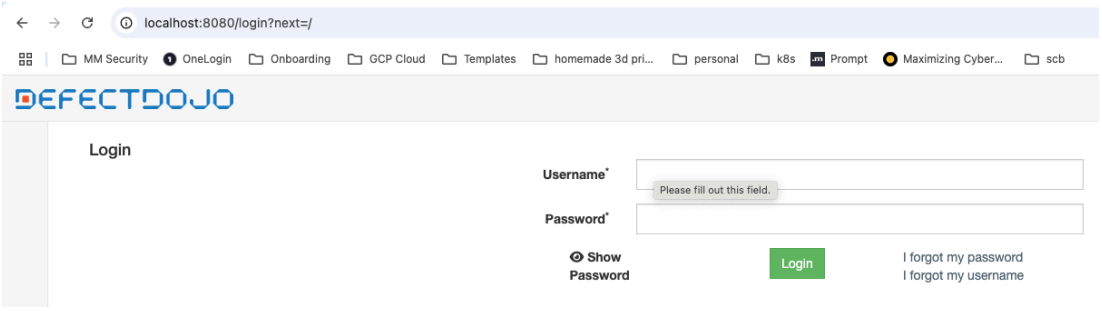

Figura 39 - Interfaz de defect dojo

Acceder con el usuario admin y colocar las credenciales obtenidas en los pasos previos. Una vez que accedimos a DD, hay que buscar el token de la API que nos va a servir para poder conectar SCB con DD. Para esto, en el panel superior derecho, hacer click en el icono del usuario y luego en APIv2 Key.

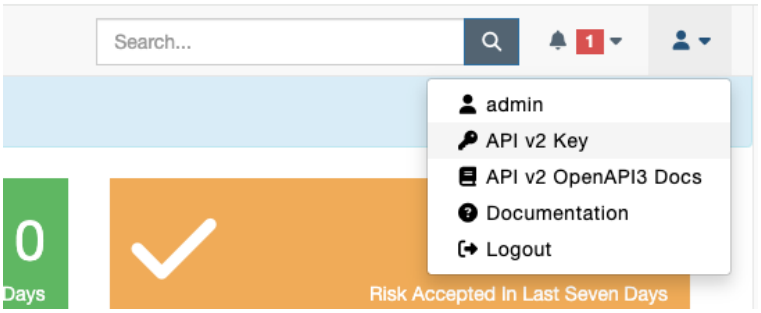

Figura 40 - Como acceder a la API key de DD

Copiar el valor del token (solo la clave generada, sin la palabra token, por ejemplo: `7cxxxxxxxxxxxxxxxxxxxxx9c5d`) para usarlo luego cuando hay que instalar el persistent hook de defectdojo.

Para que DD detecte los duplicados, es necesario realizar la configuración correspondiente. En el panel de DD, hacer click en System Settings y luego tildar “Deduplicate findings”. en la siguiente imagen se muestra la configuración:

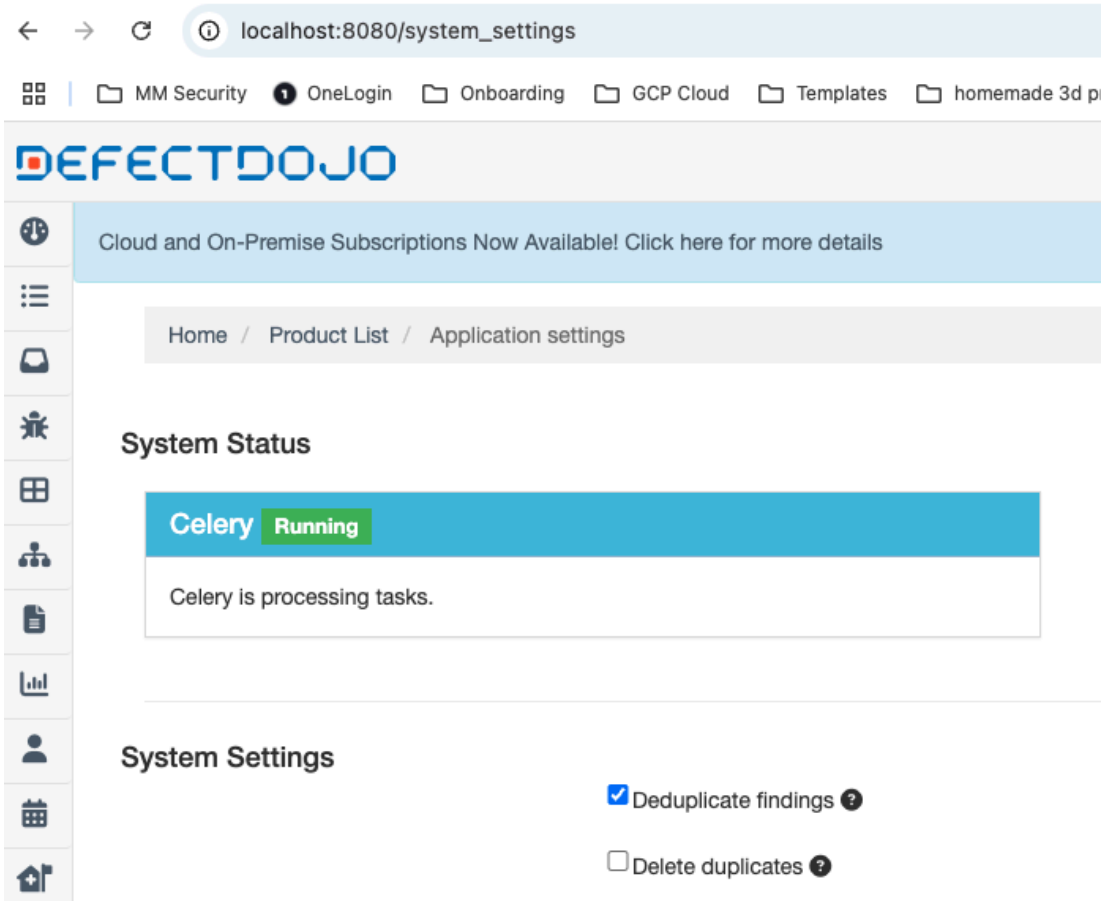

Figura 41 - Configuración para eliminar duplicados en DD

8.1.7. Instalar los scanners de seguridad

Crear un namespace llamado scanners y luego instalar las herramientas de seguridad en ese namespace:

```
→  ~ kubectl create namespace scanners
namespace/scanners created
```

Figura 42 - scanners es el namespace creado para alojar los escáneres que se van a utilizar en el laboratorio

El siguiente comando instala nmap:

```
→  ~ helm upgrade --install nmap oci://ghcr.io/securecodebox/helm/nmap -n scanners
Release "nmap" does not exist. Installing it now.
Pulled: ghcr.io/securecodebox/helm/nmap:4.10.0
Digest: sha256:ca42bcc57725b162d5e699626f5c1b7f72052c783678343670ce9e59acdd13e1
NAME: nmap
LAST DEPLOYED: Wed Nov 27 16:43:14 2024
NAMESPACE: scanners
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Figura 43 - Instalación de nmap

El siguiente comando instala semgrep:

```
→  ~ helm upgrade --install semgrep oci://ghcr.io/securecodebox/helm/semgrep -n scanners
Release "semgrep" does not exist. Installing it now.
Pulled: ghcr.io/securecodebox/helm/semgrep:4.10.0
Digest: sha256:c44031fcc3d637bf4128d825e6bcdeeda8bb94421e3509b6b74c2779f1b3e3d5
NAME: semgrep
LAST DEPLOYED: Wed Nov 27 16:44:41 2024
NAMESPACE: scanners
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Figura 44 - Instalación de semgrep

El siguiente comando instala OWASP ZAP:

```
→  ~ helm upgrade --install zap-automation-framework secureCodeBox/zap-automation-framework -n scanners
Release "zap-automation-framework" does not exist. Installing it now.
NAME: zap-automation-framework
LAST DEPLOYED: Wed Nov 27 16:43:50 2024
NAMESPACE: scanners
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Figura 45 - Instalación de OWASP Zap

Para verificar que las herramientas de seguridad están instaladas, podemos ejecutar el siguiente comando:

```
➜  ~ kubectl get scantypes -n scanners
NAME                        IMAGE
amass                       docker.io/securecodebox/scanner-amass:v4.2.0
nmap                        docker.io/securecodebox/scanner-nmap:7.95-r0
semgrep                     docker.io/returntocorp/semgrep:1.95.0
zap-automation-framework    softwaresecurityproject/zap-stable:2.15.0
```

Figura 46 - Lista de las herramientas de seguridad instaladas en el namespace scanners

Con las herramientas de seguridad ya estamos listos para hacer los análisis de seguridad pero no vamos a persistir los datos en DD hasta que lo conectemos con SCB.

8.1.8. Instalar el Persistent Hook de DefectDojo

Para proveer al Persistent Hook de las credenciales necesarias para conectarse con DD, es necesario correr el siguiente comando con el API token obtenido en el paso 8.1.6.

```
➜  ~ kubectl create secret generic defectdojo-credentials -n scanners \
  --from-literal="username=admin" --from-literal="apikey=7cxxxxxxxxxxxxxxxxxxxx9c5d"
```

Figura 47 - Se crea un secreto que tiene alojado la API key de DD para poder integrarlo con SCB

Luego se procede a instalar el persistent hook de DD

```
➜  ~ helm upgrade --namespace scanners --install persistence-defectdojo secureCodeBox/persistence-defectdojo
 --set="defectdojo.url=http://defectdojo-django.defectdojo.svc"
Release "persistence-defectdojo" does not exist. Installing it now.
NAME: persistence-defectdojo
LAST DEPLOYED: Wed Nov 27 17:07:44 2024
NAMESPACE: scanners
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
DefectDojo PersistenceProvider succesfully deployed 🎉.
```

Figura 48 - Instalación del hook persistente de DD para poder completar la integración entre SCB y DD. Es necesario definir la URL en la cual está alojado DD

Los análisis de seguridad que se realicen en el namespace scanners ahora van a estar reflejados también en DD.

8.2. Ejecución de los análisis de seguridad con Secure Code Box

La configuración de las herramientas de seguridad se hacen mediante archivos con formato YAML en el cual se especifica el tipo de herramienta a utilizar, los parámetros específicos relacionados con la herramienta, por ejemplo puede ser tipo de scan, cual es el alcance, etc. Estos valores dependen exclusivamente del tipo de herramienta que se esté utilizando.

Luego, existen otros datos que se agregan relacionados como cargar la información en DD.

En las siguientes subsecciones vamos a ver el archivo de configuración, como se ejecutan los análisis de seguridad y ver algunos resultados por línea de comando.

8.2.1. Análisis de Seguridad con Nmap

Para ejecutar un análisis de seguridad con nmap, debemos crear un archivo de configuración (que llamamos *nmap_dd.yml*) en el cual le indicamos que queremos escanear y como vamos a cargar los datos en DD:

```yaml
apiVersion: "execution.securecodebox.io/v1"
kind: Scan
metadata:
  name: "nmap-scanme.nmap.org"
  annotations:
    defectdojo.securecodebox.io/product-type-name: "Lab-Tesis-Maestria"
    defectdojo.securecodebox.io/product-name: "Nmap scan - Target: scanme.nmap.org"
    defectdojo.securecodebox.io/product-description: |
      Nmap realiza analiza puertos expuestos a internet. El target es un sitio llamado
      snanme.nmap.org que tiene puertos expuestos
    defectdojo.securecodebox.io/product-tags: nmap
    defectdojo.securecodebox.io/engagement-deduplicate-on-engagement: "true"
spec:
  scanType: "nmap"
  parameters:
    - scanme.nmap.org
```

Figura 49 - Archivo de configuración nmap con información adicional que se agregan a los hallazgos de seguridad en DD

Se ejecuta el scan con el siguiente comando:

```
templates git:(main) ✗ kubectl apply -f nmap_dd.yml -n scanners
scan.execution.securecodebox.io/nmap-scanme.nmap.org created
```

Figura 50 - Scan con nmap con el archivo de configuración

Si se desea ver cómo están ejecutando internamente los jobs, se puede ejecutar el siguiente comando:

```
templates git:(main) ✗ kubectl get jobs -n scanners
NAME                                                STATUS     COMPLETIONS   DURATION   AGE
parse-nmap-scanme.nmap.org-q4zjt                    Complete   1/1           14s        2m31s
persistence-defectdojo-nmap-scanme.nmap.org-9jtsz   Complete   1/1           26s        2m17s
scan-nmap-scanme.nmap.org-f9hgr                     Complete   1/1           10s        2m41s
```

Figura 51 - Tareas ejecutadas para completar el scan de nmap. Una tarea es el scan, otra el parseo de los hallazgos de seguridad y persistir los datos en DD

Para ver los resultados, es con el siguiente comando:

```
templates git:(main) ✗ kubectl get scans -n scanners
NAME                  TYPE   STATE   FINDINGS
nmap-scanme.nmap.org  nmap   Done    6
```

Figura 52 - Visualización del resultado del scan por línea de comando

Para ver los resultados por línea de comando se ejecuta el siguiente comando:

```
templates git:(main) ✗ kubectl describe scan nmap-scanme.nmap.org -n scanners
Name:         nmap-scanme.nmap.org
Namespace:    scanners
Labels:       <none>
Annotations:  defectdojo.securecodebox.io/engagement-deduplicate-on-engagement: true
              defectdojo.securecodebox.io/product-description:
                Nmap realiza analiza puertos expuestos a internet. El target es un sitio llamado
                snanme.nmap.org que tiene puertos expuestos
              defectdojo.securecodebox.io/product-name: Nmap scan - Target: scanme.nmap.org
              defectdojo.securecodebox.io/product-tags: nmap
              defectdojo.securecodebox.io/product-type-name: Lab-Tesis-Maestria
API Version:  execution.securecodebox.io/v1
Kind:         Scan
```

Figura 53 - Se pueden ver los detalles de los hallazgos por línea de comando

Para visualizar el resultado del scan en el bucket de mini.io, se puede ver de la siguiente manera[20]:

[20] Hay que tener habilitado el proxy para visualizar esta interfaz

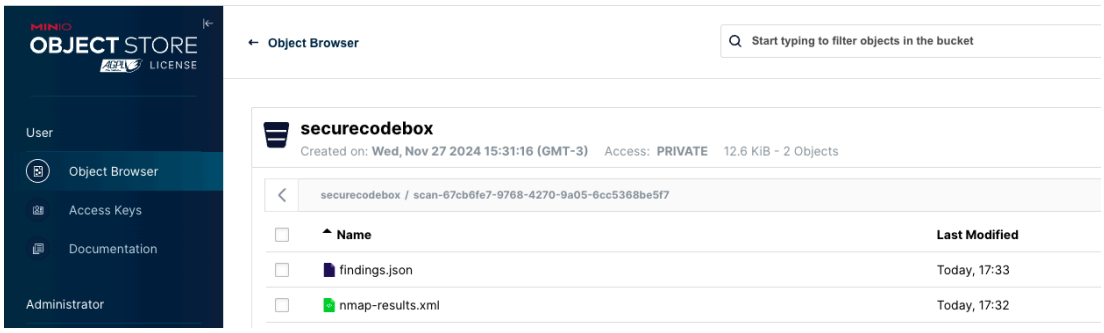

Figura 54 - Se pueden ver los hallazgos de seguridad en mini.io

El bucket contiene 2 archivos, uno en el formato que entrega SCB, y el otro archivo es en el formato que entrega la herramienta que hace el scan. Los archivos se pueden descargar y abrirlos con algún editor de texto compatible. En los siguientes scans se va a omitir mostrar los resultados en mini.io ya que se puede observar en mayor detalle usando DD.

En el siguiente gráfico se muestra como se visualiza el resultado en DD:

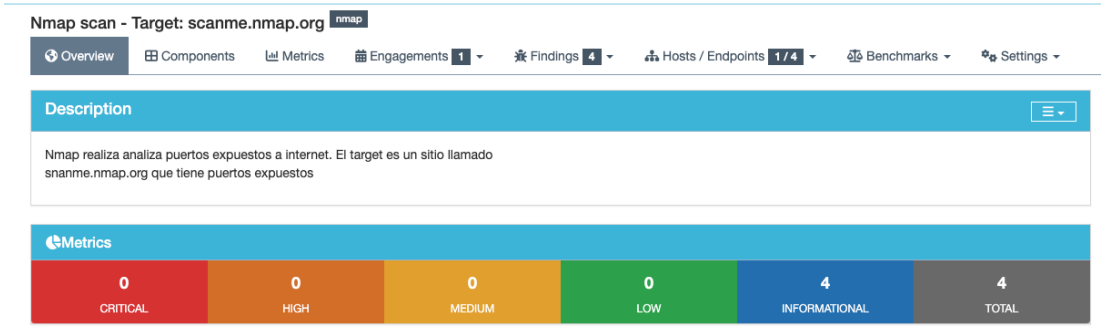

Figura 55 - Se pueden ver los hallazgos de seguridad en DD

Si queremos ver el detalle de los hallazgos de seguridad:

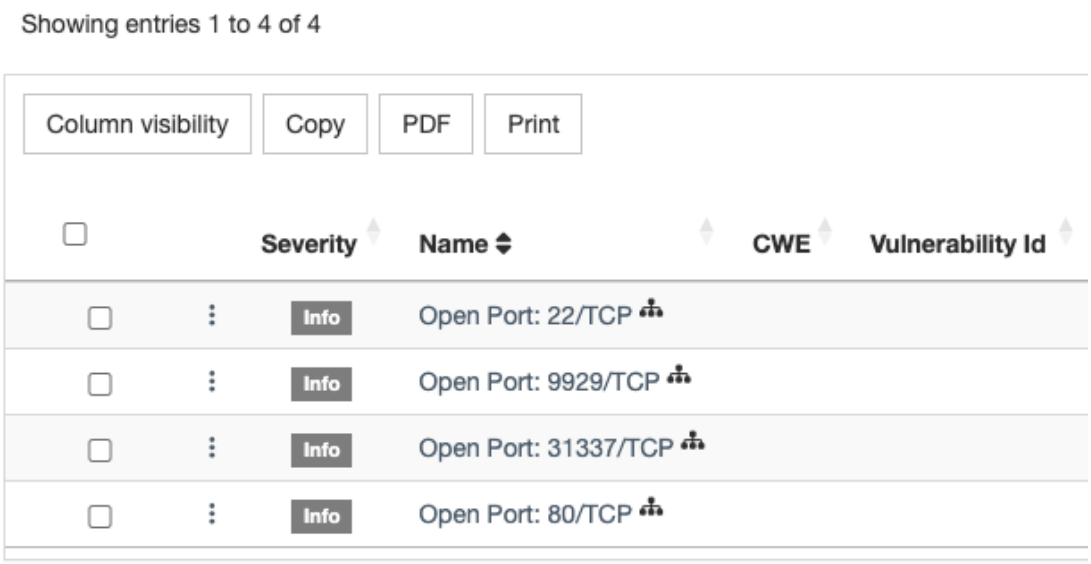

Figura 56 - Lista de los hallazgos de seguridad en DefectDojo

Luego en cada hallazgo, se puede ver el detalle que provee la herramienta que hace el scan.

8.2.2. Ejecución de análisis periódicos con Nmap

Es posible ejecutar un scan de seguridad con cualquiera de las herramientas disponibles en SCB de manera periódica. Es posible hacerlo con una frecuencia determinada, o agendados en un momento determinado con un cronjob[21]. El archivo ejecutable es el siguiente[22]:

[21] Más detalle de como realizar scan agendados: https://www.securecodebox.io/docs/api/crds/scheduled-scan

[22] En el siguiente video se muestra como ejecutar scans en cascada usando Nmap y SCB: https://drive.google.com/file/d/1jmN55EIDOgrpWWG49QdoI6mkUd9JgUYO/view?usp=drive_link

```yaml
apiVersion: "execution.securecodebox.io/v1"
kind: ScheduledScan
metadata:
  name: "scanme-nmap-org-1min"
spec:
  interval: 1m
  scanSpec:
    scanType: nmap
    parameters:
      - p22, 80, 443
      - scanme.nmap.org
```

Figura 57 - Se pueden agendar scans para que se ejecuten de manera periódica. En el ejemplo se ejecuta un scan cada 1 minuto

Se puede ver en el archivo de configuración que se va a ejecutar el mismo tipo de scan que se realizó previamente con una frecuencia de 1 minuto entre cada escaneo. Ejecutamos el archivo de configuración:

```
→  templates git:(main) ✗ kubectl apply -f scheduled_scans.yml -n scanners
scheduledscan.execution.securecodebox.io/scanme-nmap-org-1min created
```

Figura 58 - Ejecutar scans de manera periódica

Se pueden ver los scan agendados:

```
→  templates git:(main) ✗ kubectl get scheduledscans -n scanners
NAME                  TYPE   INTERVAL   SCHEDULE   FINDINGS
scanme-nmap-org-1min  nmap   1m                    6
```

Figura 59 - Visualizar los scans agendados por línea de comandos

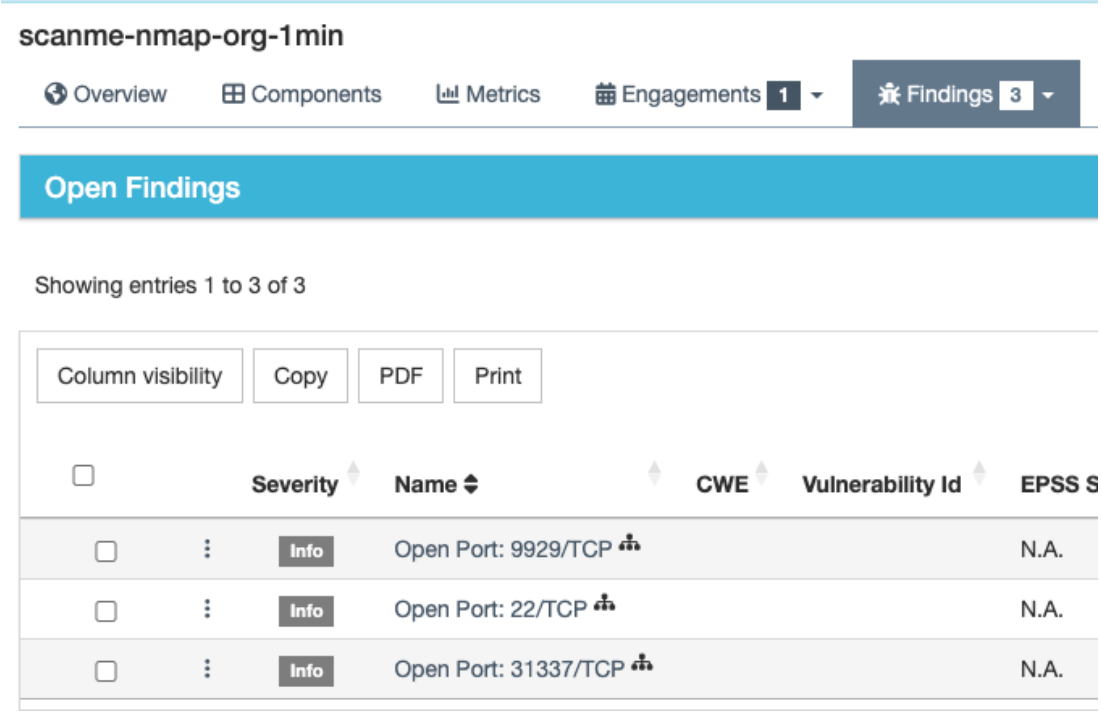

Figura 60 - Se pueden listar las vulnerabilidades de los scans agendados que fueron ejecutados

8.2.3. Análisis estático de Seguridad (SAST) con Semgrep

Semgrep es una herramienta que hace análisis de código estático. Normalmente, el código suele estar alojado en Sistemas de Control de Versiones (VCS) como github, gitlab, bitbucket, etc. Para que pueda realizar el escaneo, es necesario clonar el código alojado en el repositorio, luego realizar el scan, y por último, los resultados serán alojados en el bucket mini.io y en DD[23].

El repositorio elegido para hacer esta actividad es el siguiente:

https://github.com/vira-vira/vulnerable-code

El archivo ejecutable para hacer análisis es el siguiente:

[23] En el siguiente video se muestra como ejecutar un scan con Semgrep y SCB sobre un repositorio de Github:
https://drive.google.com/file/d/15oo7kb7iDM2EQGWckokBBAp3fqOiMWw6/view?usp=drive_link

```yaml
apiVersion: "execution.securecodebox.io/v1"
kind: Scan
metadata:
  name: "semgrep-vulnerable-code"
  annotations:
    defectdojo.securecodebox.io/product-type-name: "Lab-Tesis-Maestria"
    defectdojo.securecodebox.io/product-name: "Semgrep scan - Target: https://github.com/vira-vira/vulnerable-code"
    defectdojo.securecodebox.io/product-description: |
      Semgrep realiza análisis de código estático (SAST). El target es un sitio llamado
    defectdojo.securecodebox.io/product-tags: semgrep
    defectdojo.securecodebox.io/engagement-deduplicate-on-engagement: "true"
spec:
  # Se crea un volumen dentro de la pod donde será alojado el código
  volumes:
    - name: repository
      emptyDir: {}
  # Se agrega el volumen al contenedor
  volumeMounts:
    - mountPath: "/repo/"
      name: repository
  # Se crea un initcontainer, que realiza el clonado del repositorio
  initContainers:
    - name: "provision-git"
      # La imagen usada tiene git incluído
      image: bitnami/git
      # Se agrega el volumen al contenedor
      volumeMounts:
        - mountPath: "/repo/"
          name: repository
      # Se especifica el comando a ejecutar y donde alojarlo, mounted at /repo/
      command:
        - git
        - clone
        - "https://github.com/vira-vira/vulnerable-code"
        - /repo/flask-app
  # Se ejecuta un escan Semgrep con los siguientes parámetros
  scanType: "semgrep"
  parameters:
    - "-c"
    - "p/ci"
    - "/repo/flask-app"
```

Figura 61 - Archivo de configuración para escanear con semgrep

Se ejecuta el scan con el siguiente comando:

```
templates git:(main) ✗ kubectl apply -f semgrep_scan.yml -n scanners
scan.execution.securecodebox.io/semgrep-vulnerable-code created
```

Figura 62 - Ejecutar un scan con semgrep

El resultado en DD es el siguiente:

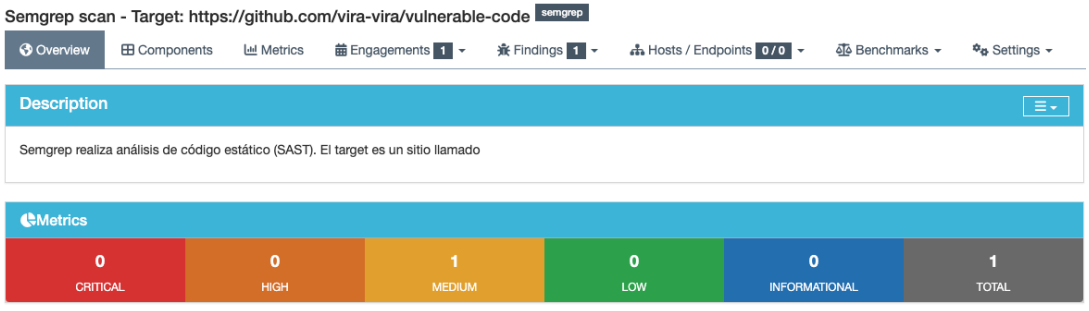

Figura 63 - Visualización de los resultados en DD

El detalle de la vulnerabilidad es el siguiente:

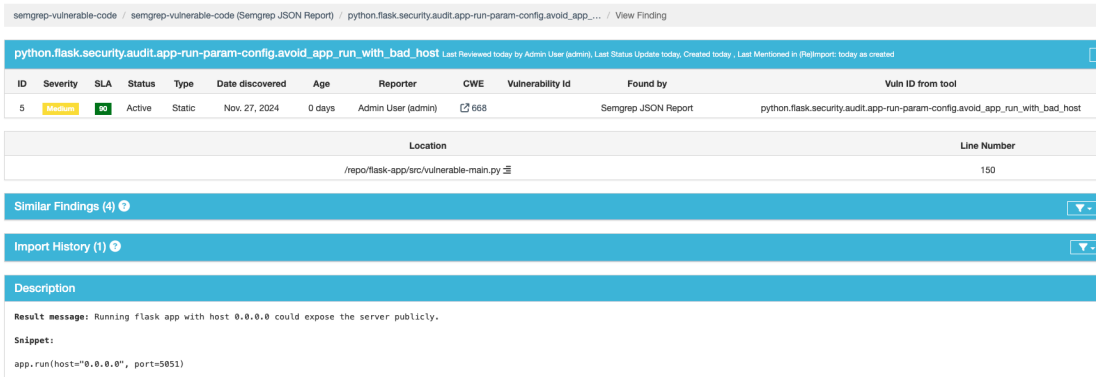

Figura 64 - Detalle de un hallazgo de seguridad

Este escaneo se hace de manera asincrónica, es decir, cuando uno ejecuta el comando para realizar el escaneo.

8.2.4. Ejecutando Semgrep dentro de un pipeline

Los desarrolladores trabajan publicando código frecuentemente en las ramas (branches) que estén trabajando (por ejemplo main). Para hacer un escaneo de seguridad de manera sincrónica, osea cada vez que el desarrollador pública código en el repositorio central, se puede ejecutar el escaneo desde el pipeline (o tubería provisto por la herramienta de SVC usada). En nuestro caso, usamos github workflows. Para esto, se realizó el siguiente pipeline:

```yaml
name: Run SAST with Securecodebox

on:
  push:
    branches:
      - main

jobs:
  sast-scan:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - id: 'auth'
        uses: 'google-github-actions/auth@v2'
        with:
          credentials_json: '${{ secrets.GCP_SERVICE_ACCOUNT_KEY }}'

      - name: Set up Google Cloud SDK
        uses: google-github-actions/setup-gcloud@v1

      - name: Install gke-gcloud-auth-plugin
        run: |
          gcloud components install gke-gcloud-auth-plugin
          gcloud components install kubectl

      - name: Get GKE credentials
        run: gcloud container clusters get-credentials
        ${{ secrets.GCP_CLUSTER_NAME }} --zone ${{ secrets.GCP_ZONE }} --project ${{ secrets.GCP_PROJECT_ID }}

      - name: Run SEMGREP
        run: kubectl apply -f k8s/semgrep_scan.yml
```

Figura 65 - Cómo configurar SAST con semgrep en un pipeline de github workflows

Este pipeline toma una imagen con la línea de comando de gcloud e instala kubectl (de la misma manera que lo hicimos previamente cuando nos conectamos al cluster de manera local). Para poder crear este servicio, se creó previamente una service account en GCP con los permisos necesarios para poder realizar acciones sobre el cluster de k8s.

El pipeline se ejecuta de la siguiente manera:

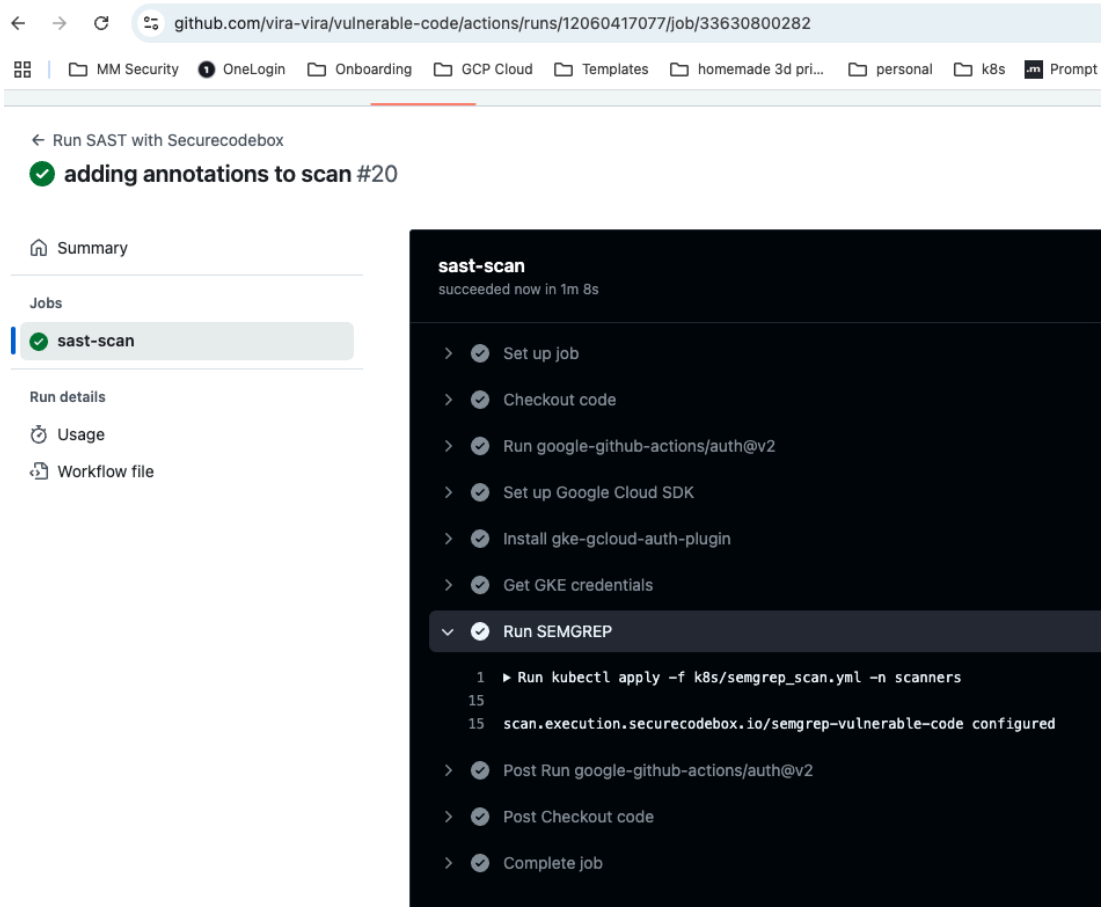

Figura 66 - Visualización del pipeline dentro de github

Como el archivo de la aplicación que se escaneo es la misma que hicimos previamente, SCB no detectó cambios y no ejecutó un scan.

8.2.5. Análisis dinámico de seguridad (DAST) con OWASP ZAP

OWASP ZAP es una herramienta que se utiliza sobre aplicaciones desplegadas. Para poder realizar un scan de seguridad, necesitamos crear una aplicación en la cual se pueda realizar el análisis sin tener algún problema ético o legal. Una aplicación vulnerable conocida que se puede desplegar en nuestro ambiente es OWASP JuiceShop. OWASP JuiceShop [47] es un portal de ventas vulnerable. Para poder crear esta aplicación,

podemos desplegarla en el cluster que creamos con los siguientes comandos.[24]

```
→  templates git:(main) ✗ kubectl create namespace demo-targets
namespace/demo-targets created
→  templates git:(main) ✗ helm upgrade --install juice-shop oci://ghcr.io/securecodebox/helm/juice-shop -n demo-targets
Release "juice-shop" does not exist. Installing it now.
Pulled: ghcr.io/securecodebox/helm/juice-shop:4.10.0
Digest: sha256:f56bc4489b7dc6e76360064773279e22954ca792827393e6b9f211739030b772
NAME: juice-shop
LAST DEPLOYED: Wed Nov 27 22:29:45 2024
NAMESPACE: demo-targets
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  echo "Visit http://127.0.0.1:3000 to use your application"
  kubectl --namespace demo-targets port-forward service/juice-shop 3000:3000
```

Figura 67 - Creación de una aplicación web vulnerable en el namespace demo-targets

Para ver la aplicación creada, se puede crear un proxy de manera local en el puerto 3000

```
→  templates git:(main) ✗ kubectl port-forward juice-shop-77d5bbd8-mhjjx -n demo-targets 3000:3000
Error from server (NotFound): pods "juice-shop-77d5bbd8-mhjjx" not found
→  templates git:(main) ✗ k get pods -n demo-targets
NAME                          READY   STATUS    RESTARTS   AGE
juice-shop-77d5bbd8-w98pb   1/1     Running   0          3m2s
→  templates git:(main) ✗ kubectl port-forward juice-shop-77d5bbd8-w98pb -n demo-targets 3000:3000
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
```

Figura 68 - Crear un proxy de manera local en el puerto 3000 para poder visualizar la aplicación

La aplicación tiene el siguiente portal:

[24] En el siguiente video se explica de manera detallada como realizar un analisis DAST con OWASP ZAP y SCB:
https://drive.google.com/file/d/1kKtSbttagmgRdFMHeOreOsGF5sMuFYhZ/view?usp=drive_link

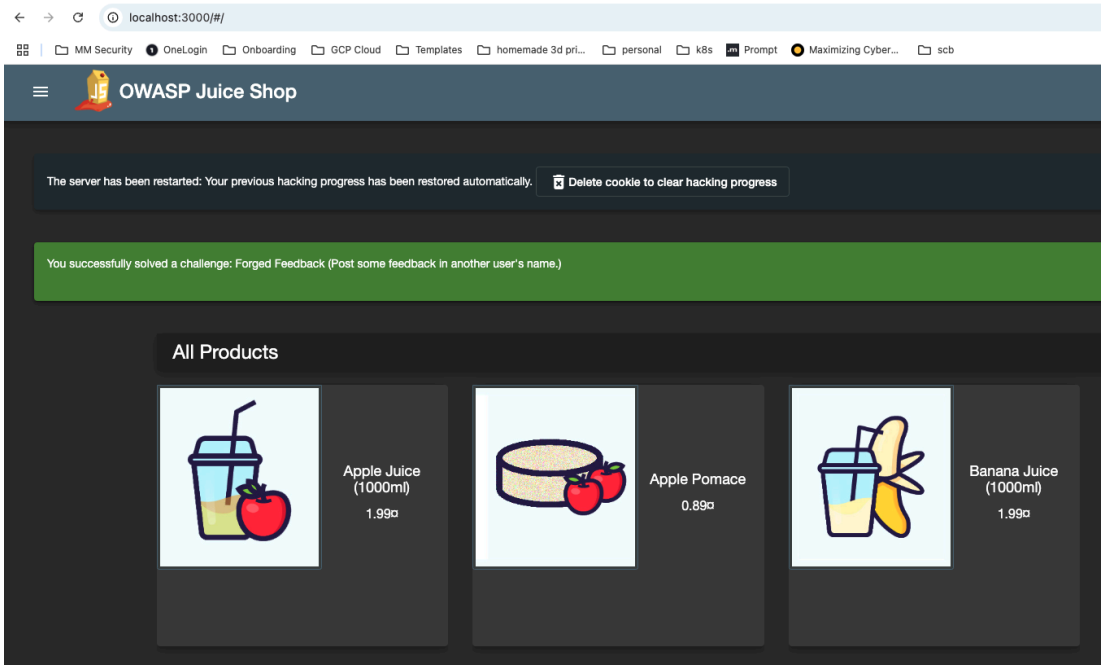

Figura 69 - Interfaz de la aplicación vulnerable OWASP JuiceShop

Con la aplicación lista para ser analizada y con OWASP ZAP instalado, se procede a realizar la configuración del scan. OWASP ZAP tiene una funcionalidad que se llama ZAP Automation framework [48], que mediante un archivo de configuración específico de OWASP ZAP, se definen todas las características necesarias, como por ejemplo el tipo de scan a realizar, el objetivo a analizar, la duración máxima del scan, etc. Este archivo se crea en k8s como un ConfigMap [48], que es un objeto en el cual se puede almacenar esta información. Es por esto que para realizar este scan se necesitan 2 archivos, uno que contenga el ConfigMap con todas las configuraciones específicas de OWASP ZAP y otro archivo que contenga las indicaciones para ejecutar el scan con SCB. Para hacer esto dentro de un mismo archivo, se puede agregar una nueva línea al final del ConfigMap y luego 3 guiones (---), esto permite tener la configuración y la ejecución del scan en un mismo archivo. En las siguientes 2 figuras se muestra como queda configurado el ConfigMap y el scan en un mismo archivo.

```yaml
# SPDX-FileCopyrightText: the secureCodeBox authors
#
# SPDX-License-Identifier: Apache-2.0


apiVersion: v1
kind: ConfigMap
metadata:
  name: zap-baseline-automation-framework-config
data:
  automation.yaml: |-

    env:                                # The environment, mandatory
      contexts :                        # List of 1 or more contexts, mandatory
        - name: zap-baseline-automation-scan # Name to be used to refer to this context in c
          urls: ["http://juice-shop.demo-targets.svc:3000/"] # A mandatory list of top level
    jobs:
      - type: spider                    # The traditional spider - fast but doesnt handle
        parameters:
          context: zap-baseline-automation-scan # String: Name of the context to spider, def
          maxDuration: 1                # Int: The max time in minutes the spider will be
      - type: passiveScan-wait          # Passive scan wait for the passive scanner to fi
        parameters:
          maxDuration: 5                # Int: The max time to wait for the passive scanr
      - type: report                    # Report generation
        parameters:
          template: traditional-xml                     # String: The template id, defaul
          reportDir: /home/securecodebox/              # String: The directory into which t
          reportFile: zap-results                     # String: The report file name patterr
        risks:                          # List: The risks to include in this report, defa
          - high
          - medium
          - low

---
```

Figura 70 - Archivo de configuración de OWASP Zap para realizar un baseline scan en OWASP JuiceShop. En la imagen se ve como se configura un scan con ConfigMaps usando automation framework

En la siguiente figura se muestra el scan:

```yaml
apiVersion: "execution.securecodebox.io/v1"
kind: Scan
metadata:
  name: "zap-automation-framework-juice-shop"
  annotations:
    defectdojo.securecodebox.io/product-type-name: "Lab-Tesis-Maestria"
    defectdojo.securecodebox.io/product-name: "Juice Shop"
    defectdojo.securecodebox.io/product-description: |
      OWASP Juice Shop is probably the most modern and sophisticated insecure web application!
    defectdojo.securecodebox.io/product-tags: vulnerable,appsec,owasp-top-ten,vulnapp
    defectdojo.securecodebox.io/engagement-name: "Juice Shop"
    defectdojo.securecodebox.io/engagement-version: "v12.6.1"
    defectdojo.securecodebox.io/engagement-tags: "automated,daily"
    defectdojo.securecodebox.io/engagement-deduplicate-on-engagement: "true"
    defectdojo.securecodebox.io/test-title: "Juice Shop - v12.6.1"
  labels:
    organization: "OWASP"
spec:
  scanType: "zap-automation-framework"
  parameters:
    - "-autorun"
    - "/home/securecodebox/scb-automation/automation.yaml"
  volumeMounts:
    - name: zap-baseline-automation-framework-config
      mountPath: /home/securecodebox/scb-automation/automation.yaml
      subPath: automation.yaml
  volumes:
    - name: zap-baseline-automation-framework-config
      configMap:
        name: zap-baseline-automation-framework-config

```

Figura 71 - Configuración del scan usando la configuración de automation framework

Para ejecutar el scan se debe ejecutar el siguiente comando:

```
templates git:(main) ✗ kubectl apply -f zap_automation_framework.yml -n scanners
configmap/zap-baseline-automation-framework-config created
scan.execution.securecodebox.io/zap-automation-framework-juice-shop created
```

Figura 72 - Ejecutar un scan a una aplicación vulnerable con OWASP Zap

El resultado con los hallazgos de seguridad se puede observar en DD

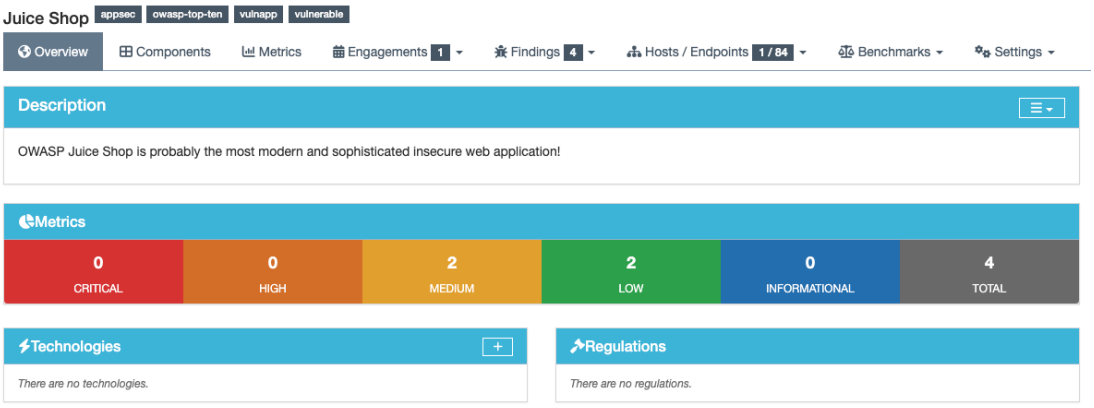

Figura 73 - Cómo se visualizan los resultados en DefectDojo

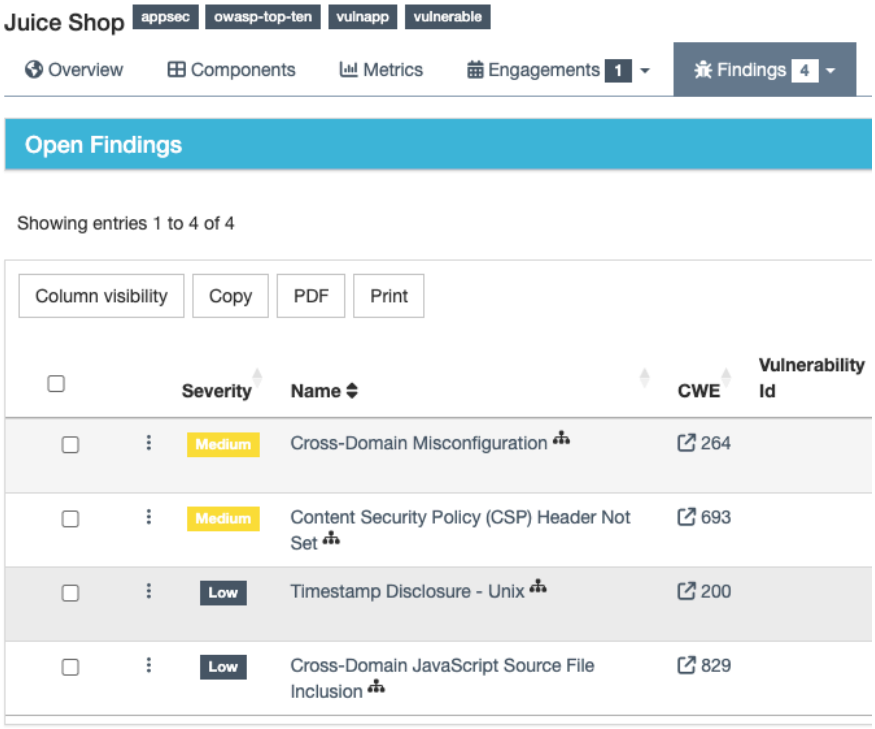

Figura 74 - Lista de hallazgos de seguridad descubiertos por OWASP Zap

8.2.6. Enumeración de dominios con Amass

Amass es una herramienta que sirve para enumerar los dominios y subdominios que tiene una aplicación web [49]. La enumeración la realizamos sobre el sitio https://securecodebox.io como se muestra en el siguiente archivo de configuración:

```yaml
# SPDX-FileCopyrightText: the secureCodeBox authors
#
# SPDX-License-Identifier: Apache-2.0

apiVersion: "execution.securecodebox.io/v1"
kind: Scan
metadata:
  name: "amass-securecodebox.io"
  labels:
    organization: "secureCodeBox"
spec:
  scanType: "amass"
  parameters:
    - "-norecursive"
    - "-d"
    - "securecodebox.io"
```

Figura 75 - Archivo de configuración de amass para la enumeración de dominios

Con amass ya instalado en SCB, ejecutamos el scan:

```
)→  templates git:(main) ✗ kubectl apply -f amass_scb.yml -n scanners
scan.execution.securecodebox.io/amass-securecodebox.io created
```

Figura 76 - Ejecutar amass por línea de comando usando SCB

Los resultados pueden visualizar en DD una vez finalizado el scan:

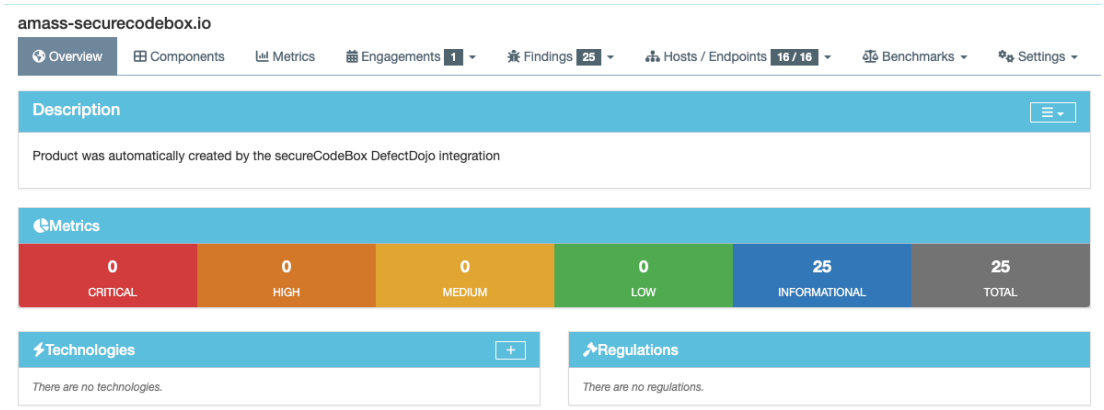

Figura 77 - Resumen de la lista de dominios y subdominios enumerados

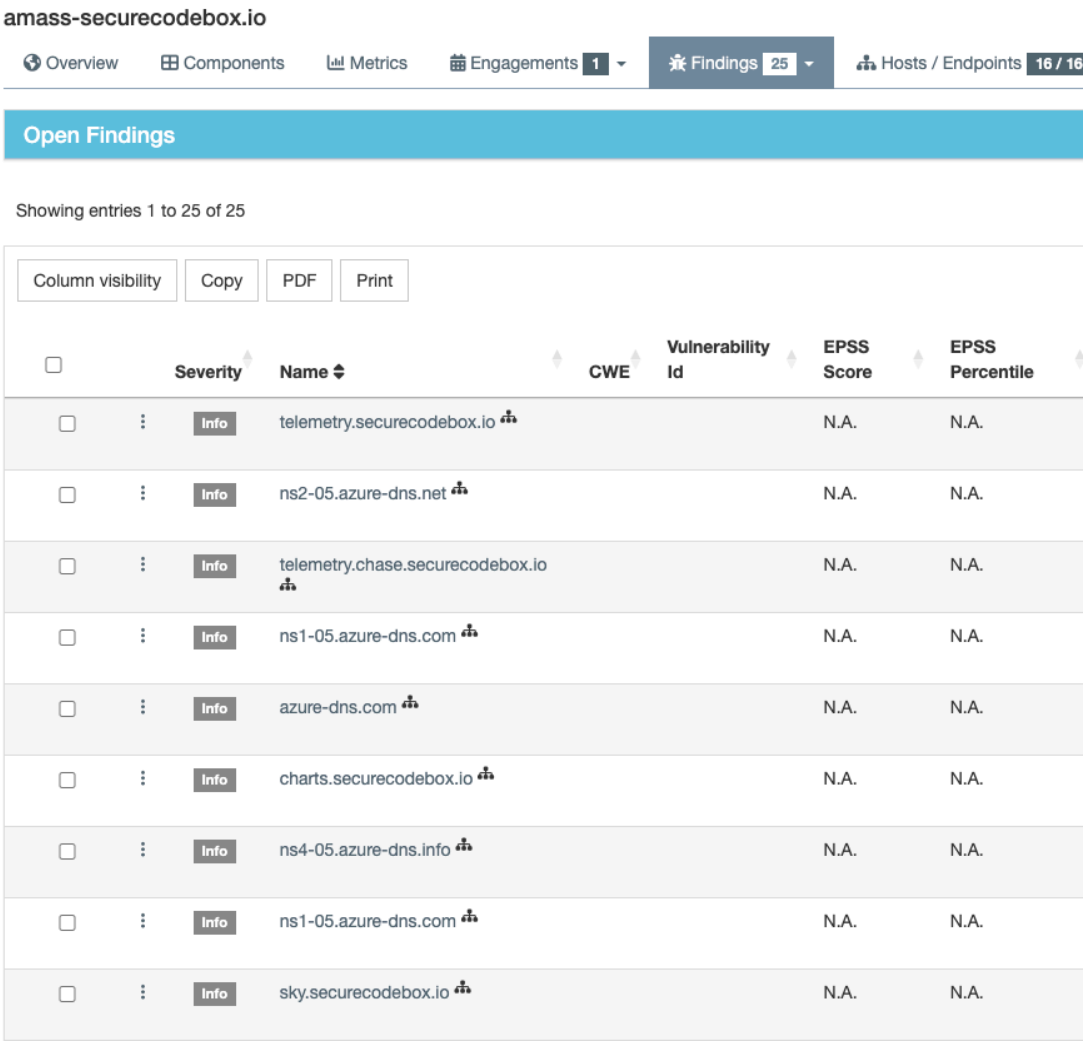

Figura 78 - Lista de dominios y subdominios enumerados

Es posible en SCB limitar el alcance de amass solo a dominios que estén dentro del alcance [50] para realizar análisis solo a objetivos que están dentro del dominio de la organización que estamos analizando.

8.2.6. Scans en cascada con Semgrep y Git Repo Scanner

Con los scans en cascada se pueden realizar infinidad de combinaciones de scans [35]. Para poder ejecutar un scan seguido de otro, se debe definir un patrón o condición por el cual un segundo scan va a ser ejecutado. Un ejemplo es Git Repo Scanner con Semgrep.

Para entender cómo usar la combinación de git repo scanner con semgrep es necesario entender la situación particular que puede resolver.

Un problema habitual que existe en el área de seguridad de las aplicaciones, es que no todos los repositorios de una compañía tienen implementado SAST y por lo tanto, si una aplicación no se le ha realizado un análisis estático, no se puede determinar si tiene o no vulnerabilidades conocidas[25][26].

Una de las herramientas de seguridad que dispone SCB se llama Git Repo Scanner. Esta herramienta puede listar todos los repositorios que tiene un espacio de trabajo en github o gitlab. El formato que entrega este tipo de herramienta es conocido, y se puede crear una condición que a todos los repositorios descubiertos por Git repo Scanner se le realice un scan Semgrep. Para esto, necesitamos crear la condición de seguridad en la cual se va a ejecutar semgrep, esta es:

[25] Ejecutando un análisis SAST con semgrep o cualquier otra herramienta de seguridad, y que no encuentre hallazgos de seguridad o vulnerabilidades, no nos da ninguna garantía de que esa aplicación es segura. Existen vulnerabilidades que no han sido reportadas, o algunas vulnerabilidades no las detecta la herramienta de seguridad, dependiendo del contexto y el tipo de scan realizado

[26] El siguiente video muestra como ejecutar un scan en cascada con Git Repo Scanner y Semgrep usando SCB:
https://drive.google.com/file/d/1NC4xlfxzJzOv9aYVP2V6pDfYW3ezmLF9/view?usp=drive_link

```yaml
apiVersion: "cascading.securecodebox.io/v1"
kind: CascadingRule
metadata:
  name: "semgrep-public-github-repos"
  labels:
    securecodebox.io/invasive: non-invasive
    securecodebox.io/intensive: medium
spec:
  matches:
    anyOf:
      # We want to scan public GitHub repositories. Change "public" to "private" t
      - name: "GitHub Repo"
        attributes:
          visibility: public
  scanSpec:
    # Configure the scanSpec for semgrep
    scanType: "semgrep"
    parameters:
      - "-c"
      - "p/ci"  # Change this to use a different rule set
      - "/repo/"
    volumes:
      - name: repo
        emptyDir: {}
    volumeMounts:
      - name: repo
        mountPath: "/repo/"
    initContainers:
      - name: "git-clone"
        image: bitnami/git

        command:
          - git
          - clone
          - "https://github.com/{{{attributes.full_name}}}"
          - /repo/
        volumeMounts:
          - mountPath: "/repo/"
            name: repo
```

Figura 79 - Archivo de configuración de la regla en cascada usando semgrep

Se instala Git Repo Scanner y Semgrep en el namespace default:

```
templates git:(main) ✗ helm upgrade --install git-repo-scanner oci://ghcr.io/securecodebox/helm/git-repo-scanner
Release "git-repo-scanner" does not exist. Installing it now.
Pulled: ghcr.io/securecodebox/helm/git-repo-scanner:4.10.0
Digest: sha256:d6c0609572bd2d17d0e6f5271058bd206ac0b7badacf63797393d44d0752382c
NAME: git-repo-scanner
LAST DEPLOYED: Wed Nov 27 23:27:57 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
templates git:(main) ✗ helm upgrade --install semgrep oci://ghcr.io/securecodebox/helm/semgrep
Release "semgrep" does not exist. Installing it now.
Pulled: ghcr.io/securecodebox/helm/semgrep:4.10.0
Digest: sha256:c44031fcc3d637bf4128d825e6bcdeeda8bb94421e3509b6b74c2779f1b3e3d5
NAME: semgrep
LAST DEPLOYED: Wed Nov 27 23:28:18 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Figura 80 - Instalación de git repo scanner y semgrep en el default namespace

Se instala la extensión de SCB para usar reglas en cascada:

```
templates git:(main) ✗ helm upgrade --install cascading-scans oci://ghcr.io/securecodebox/helm/cascading-scans
Release "cascading-scans" does not exist. Installing it now.
Pulled: ghcr.io/securecodebox/helm/cascading-scans:4.10.0
Digest: sha256:d9aad2b60f03f0aa37488f09109bc05414337c7f1c99f3d2202e3e7ca0b4c8f4
NAME: cascading-scans
LAST DEPLOYED: Thu Nov 28 12:13:58 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Cascading Scan Hook deployed.
```

Figura 81 - Instalar las reglas en cascada en el default namespace

Se instala la regla de scan en cascada para usarse git repo scanner con semgrep:

```
templates git:(main) ✗ kubectl apply -f cascading_github_repos.yml
cascadingrule.cascading.securecodebox.io/semgrep-public-github-repos created
```

Figura 82 - Instalar la regla en cascada creada para ejecutar semgrep

Para el propósito del laboratorio, se ha creado una organización en github con repositorios con vulnerabilidades conocidas:

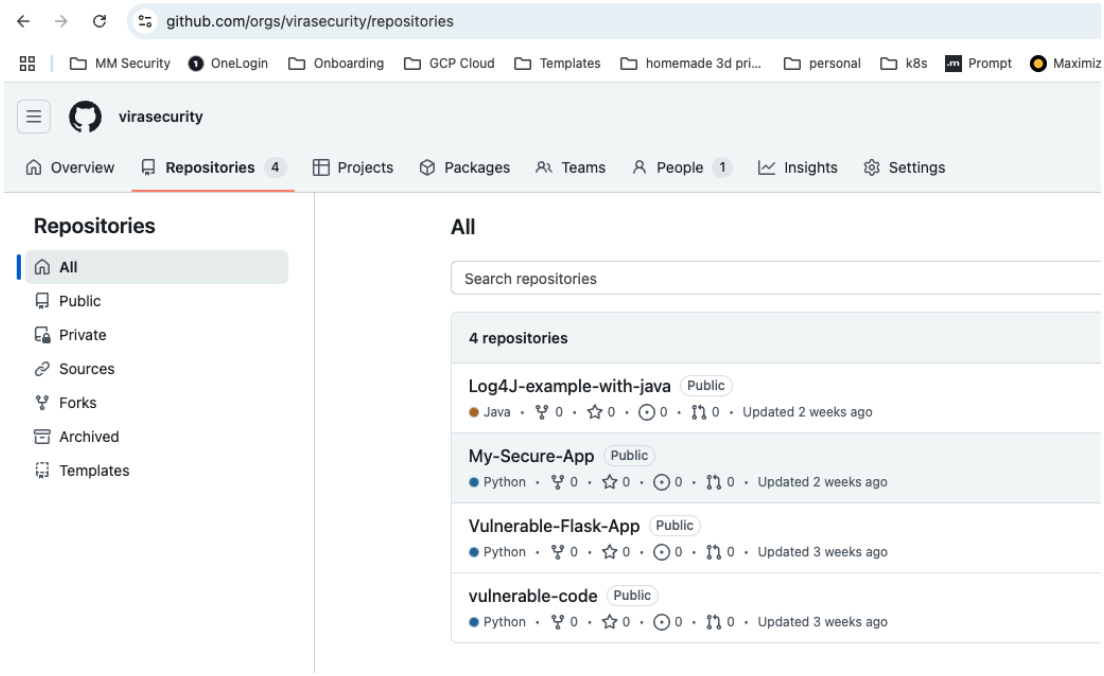

Figura 83 - Lista de repositorios en workspace que se va ha analizar

El siguiente archivo de configuración que utiliza Git Repo Scanner sobre la organización virasecurity (https://github.com/orgs/virasecurity).

```yaml
# SPDX-FileCopyrightText: the secureCodeBox authors
#
# SPDX-License-Identifier: Apache-2.0

apiVersion: "execution.securecodebox.io/v1"
kind: Scan
metadata:
  name: "scan-github"
spec:
  scanType: "git-repo-scanner"
  parameters:
    - "--git-type"
    - "github"
    - "--organization"
    - "virasecurity"
  cascades:
    matchLabels:
      securecodebox.io/intensive: medium
      securecodebox.io/invasive: non-invasive
```

Figura 84 - Archivo de configuración de git repo scanner

Se ejecuta ese archivo de configuración:

```
→  templates git:(main) ✗ kubectl apply -f git_repo_scanner.yml
scan.execution.securecodebox.io/scan-github created
```

Figura 85 - Ejecutar el scan git repo scanner por línea de comando. Este comando luego dispara el un scan con semgrep en cada repositorio encontrado

El resultado de los scans se puede ver ejecutando el siguiente comando:[27]

[27] NOTA: Como se puede observar, se instalaron las herramientas de seguridad en el namespace default y no se mostraron los resultados en DD. Esto es debido a que en el momento de realizar el laboratorio, hay un conflicto entre los scan en cascada y el persistent hook de DD, por lo tanto se realizó este análisis en un ambiente separado para poder ver los resultados.

```
templates git:(main) ✗ k get scan
NAME                                          TYPE              STATE   FINDINGS
scan-github                                   git-repo-scanner  Done    4
scan-github-semgrep-public-github-repos-67mf6 semgrep           Done    1
scan-github-semgrep-public-github-repos-dphvk semgrep           Done
scan-github-semgrep-public-github-repos-fchkq semgrep           Done    1
scan-github-semgrep-public-github-repos-qfnxw semgrep           Done    1
```

Figura 86 - Lista de todos los scan ejecutados. git repo scanner encontró 4 repositorios, luego realizó 4 scans semgrep a los repositorios encontrados

Las vulnerabilidades reportadas se pueden descargar en mini.io y ver en formato json:


```
[
  {
    "name": "python.flask.security.audit.app-run-param-config.avoid_app_run_with_bad_host",
    "location": "/repo/app/main.py:28-28",
    "description": "Running flask app with host 0.0.0.0 could expose the server publicly.",
    "category": "security",
    "severity": "MEDIUM",
    "attributes": {
      "cwe": [
        "CWE-668: Exposure of Resource to Wrong Sphere"
      ],
      "owasp_category": [
        "A01:2021 - Broken Access Control"
      ],
      "references": [
        {
          "type": "URL",
          "value": "https://owasp.org/Top10/A01_2021-Broken_Access_Control"
        },
        {
          "type": "CWE",
          "value": "CWE-66"
        },
        {
          "type": "URL",
          "value": "https://cwe.mitre.org/data/definitions/66.html"
        }
      ],
      "rule_source": "https://semgrep.dev/r/python.flask.security.audit.app-run-param-config.avoid_app_run_with_bad_host"
    },
    "id": "671de519-83c6-4c8e-aa78-7d09ba804a1b",
    "parsed_at": "2024-11-28T15:19:10.527Z"
  }
]
```


Figura 87 - Las vulnerabilidades encontradas también se pueden visualizar en el bucket mini.io

8.2.7. Otros casos de uso con SCB y los scans en cascada

Los scan en cascada tienen infinidad de casos de uso. Teniendo en cuenta las herramientas disponibles en SCB, se pueden generar condiciones que ejecuten una secuencia de scans. Algunos ejemplos son los siguientes:

- Realizar una búsqueda de una librería vulnerable crítica (como por ejemplo Log4J)[28] y se necesita saber si los repositorios de una organización tienen esa vulnerabilidad en alguno de sus repositorios. Con Git repo Scanner se obtienen los repositorios de una organización, con semgrep se define una regla para detectar esta vulnerabilidad y luego se ejecuta sobre todos los repositorios encontrados
- Se debe escanear la infraestructura de una organización. Con amass se pueden enumerar los dominios y subdominios, luego con OWASP ZAP se realiza un análisis dinámico DAST
- Hay que realizar ataques de fuerza bruta a ciertos servicios SSH para determinar si son vulnerables a estos ataques. Se puede realizar esta actividad con nmap para escanear la red y enumerar los puertos SSH, y luego realizar el ataque de fuerza bruta con ncrack

8.2.8. Autodiscovery con OWASP ZAP

Con la función autodiscovery, se puede realizar scans de seguridad a aplicaciones desplegadas en un namespace determinado, en un cluster o un recurso específico [36]. La función autodiscovery tiene 2 modos de operación, uno llamado service autodiscovery con OWASP ZAP y container autodiscovery con Trivy.

Con OWASP ZAP se pueden detectar vulnerabilidades en aplicaciones WEB. Cuando se expone una aplicación con los puertos 80, 443, 3000, 5000, 8000, 8443, 8080 se agenda un scan. En el caso de container autodiscovery, cuando se crea un contenedor nuevo se analizan las imágenes de los contenedores buscando librerías vulnerables con Trivy. En el ejemplo mostramos cómo ejecutar OWASP ZAP.

[28] Según IBM: “La vulnerabilidad Log4j, también conocida como Log4Shell, es una vulnerabilidad crítica detectada en la biblioteca de registro Apache Log4j en noviembre de 2021. Log4Shell otorga a los hackers un control total de los dispositivos que ejecutan versiones sin parches de Log4j” - Fuente: https://www.ibm.com/mx-es/topics/log4j

Para ejecutar OWASP ZAP en un namespace determinado, es necesario tener instalada la herramienta como un scantype para poder ejecutar el scan. El namespace que se utiliza es demo-targets, donde tenemos instalado la aplicación vulnerable OWASP JuiceShop. Para instalar OWASP ZAP, el comando que ejecutamos es el siguiente:

```
→  templates git:(main) ✗ helm upgrade --install zap-advanced oci://ghcr.io/securecodebox/helm/zap-advanced -n demo-targets
Release "zap-advanced" does not exist. Installing it now.
Pulled: ghcr.io/securecodebox/helm/zap-advanced:4.10.0
Digest: sha256:d705d3c3f9b19ef48c64b32f2f53d61f1bcb5248029d2c482c739941c8151a51
NAME: zap-advanced
LAST DEPLOYED: Thu Nov 28 18:12:12 2024
NAMESPACE: demo-targets
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Figura 88 - Instalación de OWASP Zap para ejecutar scans en el namespace

Luego, instalamos el servicio autodiscovery:

```
→  templates git:(main) ✗ helm upgrade --namespace securecodebox-system --install auto-discovery-kubernetes oci://ghcr.io/securecodebox/helm/au
to-discovery-kubernetes --set config.containerAutoDiscovery.enabled=true
Release "auto-discovery-kubernetes" does not exist. Installing it now.
Pulled: ghcr.io/securecodebox/helm/auto-discovery-kubernetes:4.10.0
Digest: sha256:eac35adf05e0c9533e6a3464e3849da40a6aee3fafb21942e9b8939a75ec820d
NAME: auto-discovery-kubernetes
LAST DEPLOYED: Thu Nov 28 18:26:46 2024
NAMESPACE: securecodebox-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Figura 89- Instalación del servicio autodiscovery

El namespace demo-targets debe tener en su metadatos la información que le indica al servicio autodiscovery de SCB que se debe escanear cada vez que se ejecute un servicio web nuevo.

```
→  templates git:(main) ✗ kubectl annotate namespace demo-targets auto-discovery.securecodebox.io/enabled=true
namespace/demo-targets annotated
```

Figura 90 - Configuración del namespace para que pueda ser detectado por el servicio autodiscovery

Ahora si re desplegamos OWASP JuiceShop:

```
→  templates git:(main) ✗ helm upgrade --install juice-shop oci://ghcr.io/securecodebox/helm/juice-shop -n demo-targets
Release "juice-shop" does not exist. Installing it now.
```

Figura 91 - Instalación de un servicio http nuevo en el namespace con autodiscovery configurado

El servicio de OWASP JuiceShop se ejecuta y automáticamente se ejecuta un scan:

```
templates git:(main) ✗ k get pods -n demo-targets
NAME                                                      READY   STATUS    RESTARTS   AGE
juice-shop-77d5bbd8-2dfhr                                 1/1     Running   0          51s
scan-juice-shop-service-zap-port-3000-1732833438-rjz6x-wvqzj   3/3     Running   0          26s
```

Figura 92- Scans que se realizan sobre la nueva aplicación con servicio http

El scan que se ejecuta es el siguiente:

```
templates git:(main) ✗ kubectl get scans -n demo-targets
NAME                                        TYPE               STATE      FINDINGS
juice-shop-service-zap-port-3000-1732833438   zap-advanced-scan   Scanning
```

Figura 93- Proceso de escaneo usando el servicio de autodiscovery y OWASP Zap sobre un servicio http

De esta manera, se ejecuta un scan con OWASP ZAP cada vez que se crea un servicio en el namespace que tiene configurado la función service autodiscovery.

9. Conclusiones

9.1. Objetivo y resultados

Este trabajo ha examinado las iniciativas fundamentales para una gestión efectiva de vulnerabilidades en organizaciones. Se abordaron procesos de estandarización que incluyen la creación de políticas, el establecimiento de un baseline de seguridad y la implementación de una política de desarrollo seguro (SSDLC).

Se presentaron conceptos clave relacionados con la gestión de vulnerabilidades, tales como:

- Análisis de Seguridad de Código Estático (SAST)
- Análisis de Composición de Software (SCA)
- Análisis Dinámico de Seguridad en Aplicaciones (DAST)
- Análisis de Seguridad en Infraestructura como Código (IaC Scans)
- Análisis de Seguridad en Contenedores (Container Scanning)
- Modelado de Amenazas (Threat Modeling)
- Pruebas de Penetración (Pentest), entre otros.

Se explicaron algunas herramientas de código abierto que hay disponibles en el mercado que realizan estas actividades de seguridad como SAST con Semgrep, DAST con OWASP ZAP, enumeración de puertos abiertos con nmap, enumeración de dominios con amass, etc.

Se demostró la orquestación de estas actividades utilizando Secure Code Box (SCB), presentando su funcionamiento basado en Kubernetes. Se propuso la integración de DefectDojo (DD) para gestionar vulnerabilidades reportadas, incluyendo su implementación dentro de SCB [51].

Se desarrolló exitosamente un laboratorio en la nube de Google (GCP) que involucró la creación de un clúster de Kubernetes, así como la implementación de SCB con diversas herramientas y la visualización en DefectDojo.

9.2. Aportes sobre la gestión de vulnerabilidades

Se concluyó que es viable gestionar vulnerabilidades de manera efectiva utilizando herramientas de código abierto. SCB actúa como orquestador de herramientas de seguridad, mientras que DD facilita la visualización y el manejo de vulnerabilidades.

La estructura contenerizada de SCB junto con su capacidad de integración en un clúster de Kubernetes garantiza su escalabilidad, permitiendo su implementación en organizaciones de diversos tamaños. A través de GitHub Workflows, los escáneres de seguridad se integran en un entorno DevSecOps, permitiendo escaneos asíncronos en SCB.

Los análisis en cascada de SCB permiten automatizar flujos de seguridad complejos, y se presentó un caso donde se buscó una vulnerabilidad específica en múltiples repositorios. La gestión centralizada en DD favorece una visualización clara de las vulnerabilidades, organizadas por producto y tipo de iniciativa.

DD tiene la flexibilidad de permitir una gran cantidad de formatos que proveen las distintas herramientas de seguridad para agregar los hallazgos de manera transparente y sin esfuerzo adicional. Con una API que tiene integrada, se pueden enviar vulnerabilidades de distintos escáneres y visualizarlos para su posterior tratamiento

La capacidad de DD para evitar duplicados y clasificar hallazgos como falsos positivos mejora significativamente la gestión. Además, su flexibilidad en el manejo de diferentes formatos y su integración mediante API simplifican el tratamiento de vulnerabilidades.

La configuración DD es customizable y se puede integrar con otras soluciones para SSO, para notificaciones y con otras herramientas para la gestión de proyectos.

DD tiene a su vez la posibilidad de gestionar los accesos y los usuarios permitiendo que se cumpla el principio de mínimo privilegio.

La calidad de los hallazgos de seguridad encontrados dependen de la herramienta que se utilice. Algunas herramientas como Semgrep tienen la posibilidad de agregar reglas personalizadas para agregar la información adicional o detectar una vulnerabilidad en particular.

9.3. Limitaciones del uso de SCB y DD

Dentro de las limitaciones de Secure Code Box podemos mencionar:

- Es una herramienta que depende de distintas tecnologías como lo es Kubernetes y Docker. Esta dependencia, hace que sea necesario conocer cómo trabajar con estas tecnología para poder resolver distintos desafíos que existen en el camino
- Es una herramienta open source, por lo cual el incentivo del equipo que lo desarrolla es simplemente colaborar con la comunidad. Esto hace que no existan grandes equipos de desarrollo y las respuesta ante algún reporte de seguridad sea limitada. Existen distintos canales de comunicación como Slack y Github issues, pero el soporte es limitado
- Algunos componentes de SCB no se encuentran del todo maduros, por ejemplo se puede mencionar que cuando se ejecutan los scans en cascada no funciona correctamente la integración con DefectDojo
- Si bien está documentado cómo agregar una herramienta de seguridad nueva, puede ser complicado la integración de una solución
- SCB no es el indicado para todos los casos de uso. Se identificaron algunos escenarios en los cuales no es recomendable el uso de SCB. Por ejemplo, cuando se realiza análisis SAST dentro de SCB, es necesario clonar el repositorio, esta es una actividad asíncrona, mientras que si se realiza el scan dentro del pipeline de git (sin usar SCB), sería un scan sincrónico. Esto tiene algunas características:

- Que sea asincrónico es una ventaja cuando los repositorios son de gran tamaño y puede demorar el flujo para desplegar una aplicación o cuando se quiere hacer scan de manera periódica si afectar el pipeline.
- Que sea sincrónico puede utilizarse para bloquear el despliegue de aplicaciones si existen vulnerabilidades críticas o para ver los resultados de vulnerabilidades inmediatamente después de que se publica el código en una rama del repositorio.

DefectDojo, por su lado tiene las siguientes limitaciones:

- La interfaz gráfica tiene funcionalidades limitadas, y no es posible crear indicadores propios.
- DD tiene varios componentes como la interfaz, la base de datos, celery que pueden ser complejos de configurar en ambientes productivos.
- La puesta a punto de DD también puede ser compleja, ya que se puede integrar un sistema de SSO, se pueden gestionar usuarios, grupos, roles, gestión de duplicados y falsos positivos
- Cuando existen grandes volúmenes de hallazgos de seguridad, la escalabilidad puede verse afectada

9.4. Recomendaciones de uso en organizaciones

Dado que se realizó el ambiente en un laboratorio, es posible tener algunas consideraciones adicionales de seguridad para realizar esta actividad en un ambiente productivo. Respecto a DD, algunas recomendaciones son las siguientes:

- Configurar SSO con SAML: Implementar inicio de sesión único (SSO) usando el protocolo SAML para mejorar la seguridad y la gestión de usuarios.

- Configurar la Base de Datos: Asegurar que la base de datos esté correctamente configurada con políticas de acceso restringido y cifrado.
- Usar Credenciales de Menos Privilegios: Evitar el uso de credenciales administrativas. En su lugar, utilizar cuentas de usuario con privilegios mínimos necesarios.
- Ajustar Permisos de la API: Configurar la API para que opere con permisos reducidos, limitándose a las acciones necesarias como publicar, borrar vulnerabilidades, y gestionar productos y compromisos.

Respecto a SCB, algunas recomendaciones son las siguientes:

- Usar un Bucket en lugar de Mini.io: Optar por un almacenamiento seguro y gestionado en la nube en vez de mini.io para mejorar la seguridad de los datos.
- Separar Ambientes por Namespaces: Organizar los entornos de desarrollo, prueba y producción usando namespaces para minimizar riesgos de seguridad.

Estas son algunas recomendaciones pero no las únicas, se pueden mencionar también la necesidad de otras actividades como monitoreo backups, encriptación, etc.

10. Bibliografía

[1] IBM. “Gestión de Vulnerabilidades.” ¿Qué es la gestión de vulnerabilidades?, https://www.ibm.com/es-es/topics/vulnerability-management. Último acceso 9 12 2024.

[2] “Cinco Días - El País - Así ha sido la ciberseguridad en 2024 y esta es la gran apuesta para 2025: más IA.” https://cincodias.elpais.com/smartlife/lifestyle/2024-11-26/ciberseguridad-en-2024-apuesta-2025-inteligencia-artificial.html. Último acceso 9-12-2024.

[3] “How to use AI for Social Engineering Hacking.” StationX, 2024, https://www.stationx.net/how-to-use-ai-for-social-engineering/. Último acceso 9-12-2024.

[4] “General Data Protection Regulation.” https://gdpr-info.eu/. Último acceso 9-12-2024.

[5] “What is the Sarbanes-Oxley Act – SOX in Cyber Security?” Institute of Data, https://www.institutedata.com/blog/sarbanes-oxley-act/. Último acceso 1-03-2025.

[6] “PCI DSS.” https://docs-prv.pcisecuritystandards.org/PCI%20DSS/Standard/PCI-DSS-v4_0_1.pdf. Último acceso 9-12-2024.

[7] “OWASP - Open Web Application Security Project.” https://owasp.org/. Último acceso 9-12-2024.

[8] “Mitre Attack.” ATT&CK Matrix for Enterprise, https://attack.mitre.org/. Último acceso 9-12-2024.

[9] “iprofesional BCRA: nuevas normas para las entidades financieras en materia de tecnología y seguridad” https://www.iprofesional.com/impuestos/378678-nuevas-normas-para-bancos-en-materia-de-tecnologia-y-seguridad. Último acceso 17-12-2024

[10] “OWASP - Source Code Analysis tools” https://owasp.org/www-community/Source_Code_Analysis_Tools. Último acceso 17-12-2024

[11] “Semgrep - Our mission.” https://semgrep.dev/about/. Último acceso 10-12-2024.

[12] BlackDuck. “Dynamic Application Security Testing (DAST).” BlackDuck, https://www.blackduck.com/glossary/what-is-dast.html#A. Último acceso 5-12-2024.

[13] Checkpoint. “What is Dynamic Application Security Testing (DAST)?” What is Dynamic Application Security Testing (DAST)?, https://www.checkpoint.com/es/cyber-hub/cloud-security/what-is-dynamic-application-security-testing-dast/. Último acceso 5-12-2024.

[14] AppSecSanta. “DAST Tools : 23 Best Free and Paid Tools (2022 update)”, https://www.appsecsanta.com/dast-tools. Último acceso 17-12-2024

[15] “OWASP ZAP - Getting Started.” https://www.zaproxy.org/getting-started/. Último acceso 11-12-2024.

[16] “Penetration testing.” National Cyber Security Center, https://www.ncsc.gov.uk/guidance/penetration-testing. Último acceso 1-03-2025.

[17] “NIST SP 800-115 - Technical Guide to Information Security Testing and Assessment.” NIST SP 800-115, 1 2020. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-115.pdf. Último acceso 17-12-2024

[18] “AWS - What is DevOps?” https://aws.amazon.com/devops/what-is-devops/. Último acceso 11-12-2024.

[19] “Locuz - What is DevSecOps?“ https://www.locuz.com/devsecops. Último acceso 17-12-2024.

[20] “IBM - What is DevSecOps?” https://www.ibm.com/topics/devsecops. Último acceso 11-12-2024.

[21] “Wikipedia - GIT.” https://es.wikipedia.org/wiki/Git. Último acceso 11-12-2024.

[22] “Github Actions documentation.” https://docs.github.com/en/actions. Último acceso 11-12-2024.

[23] “Secure Code Box - Automated Security Testing Tool.” https://www.securecodebox.io/. Último acceso 16-12-2024.

[24] “martinFowler - Microservices.” https://martinfowler.com/articles/microservices.html. Último acceso 12-12-2024.

[25] “Docker - Use containers to build, share and run your applications.” https://www.docker.com/resources/what-container/. Último acceso 12-12-2024.

[26] “Kubernetes - Overview.” https://kubernetes.io/docs/concepts/overview/. Último acceso 17-12-2024.

[27] “Helm - Introduction to Helm.” https://helm.sh/docs/intro/. Último acceso 17-12-2024.

[28] “OWASP - Secure Code Box.” https://owasp.org/www-project-securecodebox/. Último acceso 16-12-2024.

[29] “Secure Code Box - CRDs.” https://www.securecodebox.io/docs/api/crds/. Último acceso 16-12-2024.

[30] “Kubernetes - Custom Resources.” https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/. Último acceso 16-12-2024.

[31] “Secure Code Box - Scanners.” https://www.securecodebox.io/docs/scanners. Último acceso 16-12-2024.

[32] “Secure Code Box - Hooks.” https://www.securecodebox.io/docs/hooks. Último acceso 16-12-2024.

[33] “Secure Code Box - Notification WebHook.” https://www.securecodebox.io/docs/hooks/notification-webhook. Último acceso 17-12-2024.

[34] “Secure Code Box - Repeating a scan on a schedule.” https://www.securecodebox.io/docs/how-tos/automatically-repeating-scans. Último acceso 17-12-2024.

[35] “Secure Code Box - Cascading Scans.” https://www.securecodebox.io/docs/hooks/cascading-scans. Último acceso 17-12-2024.

[36] “Secure Code Box - Autodiscovery.” https://www.securecodebox.io/docs/auto-discovery/overview. Último acceso 17-12-2024.

[37] “Secure Code Box - Generic WebHook.” https://www.securecodebox.io/docs/hooks/generic-webhook. Último acceso 17-12-2024.

[38] “Secure Code Box - What is "Persistence DefectDojo" Hook about?” https://www.securecodebox.io/docs/hooks/defectdojo. Último acceso 16-12-2024.

[39] “DefectDojo.” https://defectdojo.com/. Último acceso 17-12-2024.

[40] “OWASP - Defectdojo.” https://owasp.org/www-project-defectdojo/. Último acceso 16-12-2024.

[41] “DefectDojo - What is DefectDojo?” https://docs.defectdojo.com/en/about_defectdojo/about_docs/. Último acceso 16-12-2024.

[42] “DefectDojo - Product Hierarchy - Overview.” https://support.defectdojo.com/en/articles/8545273-product-hierarchy-overview. Último acceso 17-12-2024.

[43] “DefectDojo - About Permissions & Roles.” https://support.defectdojo.com/en/articles/8758189-about-permissions-roles. Último acceso 16-12-2024.

[44] “DefectDojo - Introduction to findings.” https://support.defectdojo.com/en/articles/9958731-introduction-to-findings. Último acceso 17-12-2024.

[45] “DefectDojo - About Deduplication.” https://support.defectdojo.com/en/articles/9658085-about-deduplication. Último acceso 17-12-2024.

[46] “Medium - How To Set up a GKE Cluster on the Google Cloud Platform.” https://medium.com/@mrayandutta/how-to-set-up-a-gke-cluster-on-the-google-cloud-platform-6b5a74ace235. Último acceso 16-12-2024.

[47] “OWASP Juice Shop.” https://owasp.org/www-project-juice-shop/. Último acceso 17-12-2024.

[48] “ZAP - Automation Framework.” https://www.zaproxy.org/docs/desktop/addons/automation-framework/. Último acceso 17-12-2024.

[48] “Kubernetes - Config Maps.” https://kubernetes.io/docs/concepts/configuration/configmap/. Último acceso 17-12-2024.

[49] “OWASP Amass.” https://owasp.org/www-project-amass/. Último acceso 17-12-2024.

[50] “Secure Code Box - Enforcing Engagement Scope.” https://www.securecodebox.io/docs/how-tos/scope. Último acceso 17-12-2024.

[51] “Orquestando un ambiente DevSecOps con Secure Code Box - Sergio Correa - Ekoparty 2024” https://www.youtube.com/watch?v=JYev018b9W8&t=4s Último acceso 29-12-2024.