METODOLOGÍAS DE DESARROLLO SEGURO
EVALUAR LA SEGURIDAD MEDIANTE HERRAMIENTAS AUTOMÁTICAS

Para contextualizar, se procede a realizar una evaluación de la seguridad de la aplicación web Juice-shop, que es la que se ha estado utilizando durante el resto de actividades.

Se ha hecho una selección de herramientas open source, para realizar dichas evaluaciones de seguridad, en consonancia con la aplicación web seleccionada.

Se comenzará realizando la primera prueba con la herramienta Snyk.

¿Qué es exactamente Snyk?

Es una herramienta que ayuda a los desarrolladores a encontrar y solucionar vulnerabilidades en las dependencias y librerías del proyecto.

¿Por qué es importante o relevante para la Juice-shop?

Porque es una aplicación que depende de numerosas librerías de terceros y al poder contener vulnerabilidades conocidas, podrían ser explotadas.

¿Cómo funciona?

En el caso que nos ocupa, se ha utilizado la versión extensión de VScode.

- Detecta vulnerabilidades en las dependencias escaneando la aplicación automáticamente.
- Se puede integrar con los flujos de trabajo de DevSecOps, para garantizar que las vulnerabilidades se detecten y se resuelvan antes de que el código llegue a producción.

- No solo detecta las vulnerabilidades, sino que puede crear pull request automáticos para actualizar las dependencias a versiones seguras.
- Esto lo hace verificando si existe una versión nueva de la dependencia, que no contenga dicha vulnerabilidad.

La siguiente herramienta que se ha utilizado es Docker Scout.

¿Qué es Docker Scout?

Es una herramienta diseñada para escanear e identificar vulnerabilidades y problemas de seguridad en contenedores de Docker.

¿Por qué es relevante para Juice-shop?

Como la aplicación Juice-shop puede ser desplegada en contenedores Docker, Docker Scout es efectivo y eficaz para esta aplicación.

¿Cómo funciona?

- Escanea las imágenes de Docker utilizadas para desplegar Juice Shop, identificando las vulnerabilidades en las librerías incluidas en las imágenes.
- Docker Scout, proporciona recomendaciones para optimizar las imágenes, reduciendo su tamaño y mejorando la eficiencia.
- Fácil integración, ya que también se integra con Docker Hub, permitiendo escanear imágenes almacenadas en repositorios públicos y privados.

En resumen:

Se utiliza para analizar las capas de una imagen de Docker en busca de vulnerabilidades conocidas en bibliotecas o dependencias y configuraciones.

Este escaneo es importante de cara a garantizar que las imágenes de los contenedores sean seguras y estén libres de vulnerabilidades antes de implementarlas en entornos de producción.

La tercera herramienta en cuestión, es OWASP ZAP (Zed Attack Proxy).

¿Qué es ZAP?

Es una herramienta open source, diseñada para encontrar vulnerabilidades en aplicaciones web. ZAP, es un proxy intercomunicador que permite a los desarrolladores capturar y analizar las comunicaciones entre el navegador y la aplicación web para identificar posibles vulnerabilidades.

¿Por qué es relevante para Juice Shop?

Al ser una aplicación web diseñada específicamente para contener una multitud de vulnerabilidades de seguridad comunes en aplicaciones web, ZAP es enormemente adecuado para realizar dicha prueba.

¿Cómo funciona?

- Detectando una amplia gama de vulnerabilidades en aplicaciones web, tales como inyección SQL, XSS, seguridad de cabeceras HTTP…

- ZAP ofrece capacidades de escaneo automatizado y manual, lo cual es útil para explorar áreas específicas de la aplicación y descubrir vulnerabilidades complejas.

- Puede integrarse en pipelines de CI(integración continua)/CD(despliegue continuo), permitiendo la ejecución continua de pruebas de seguridad y asegurando que las nuevas versiones del código se analicen automáticamente.

La cuarta herramienta que se ha ejecutado, es Contrast Scan.

¿Qué es Contrast Scan?

Es una herramienta de análisis estático de seguridad (SAST), que escanea el código fuente de las aplicaciones en busca de vulnerabilidades.

Esta herramienta, se integra en el ciclo de vida de desarrollo de software (SDLC), proporcionando análisis continuos de seguridad a medida que el código se desarrolla y se modifica.

¿Por qué es relevante para Juice Shop?

Porque es relevante para todas las aplicaciones, ya que todas poseen código. Es una herramienta generalista para la seguridad.

¿Cómo funciona?

- Contrast Scan, permite identificar vulnerabilidades en el código fuente antes de que la aplicación sea desplegada, lo cual es crucial para prevenir la explotación de estas vulnerabilidades en producción.
- Se integra perfectamente con herramientas de integración y entrega continua (CI/CD), permitiendo análisis automáticos y continuos del código a medida que se desarrolla.
- Como ventaja, soporta múltiples lenguajes de programación y frameworks, lo que es útil para Juice Shop, ya que incluye componentes en diferentes lenguajes.
- Proporciona después de la identificación de vulnerabilidades, un reporte detallado que incluyen las vulnerabilidades, su severidad y las recomendaciones para mitigarlas.

Por último, se procede a ejecutar la herramienta Dependency-check.

¿Qué es OWASP Dependency-Check?

Es una herramienta de análisis de dependencias que identifica vulnerabilidades conocidas en las bibliotecas y componentes que utiliza el proyecto.

¿Por qué es relevante para Juice Shop?

Porque es una aplicación Node.js, que depende de muchas librerías de terceros y es fácil que contenga vulnerabilidades. (En este caso más, ya que la aplicación está hecha a conciencia para practicar y explotar vulnerabilidades).

¿Cómo funciona?

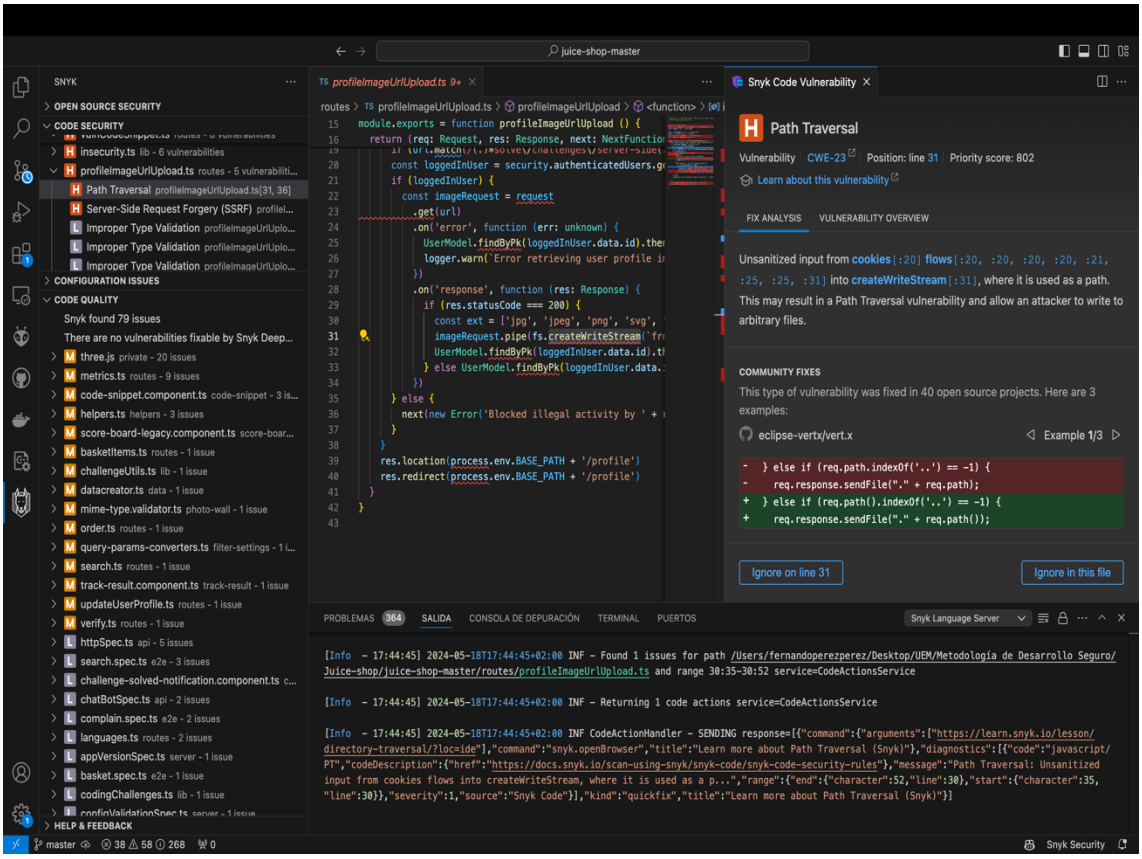
- Escanea las dependencias y detecta las vulnerabilidades conocidas en las versiones utilizadas. Esto lo consigue comparando las dependencias del proyecto con bases de datos de vulnerabilidades conocidas, como el National Vulnerability Database. (https://nvd.nist.gov)
- También puede integrarse en pipelines de CI/CD para asegurar que cada cambio en el código o en las dependencias es verificado automáticamente en busca de vulnerabilidades.
- Proporciona un reporte detallado (en este caso un archivo .xml) en el que incluyen descripciones de las vulnerabilidades, su severidad y posibles soluciones.

Pruebas realizadas

1- **Snyk**

Usando la versión de desarrollador gratuito de la página oficial Snyk.io.

Para esta prueba, se ha procedido a utilizarse esta herramienta mediante la extensión que se encuentra en VSCode.
Una vez instalada la extensión y logeado al usuario, se realiza un escaneo automático de las vulnerabilidades, su severidad y posibles soluciones encontradas en git.
Aquí se presenta una captura de pantalla de la prueba realizada en la que podremos ver una vulnerabilidad concreta.



Aquí se muestran los siguientes datos:

Tipo de vulnerabilidad: Path Traversal
CWE: 23
Posición: Línea 31
Puntuación de prioridad: 802

Se describe detalladamente que es una vulnerabilidad que se debe a la falta de sanitización de las entradas recibidas a través de las cookies :20,

y estas entradas fluyen hacia createWriteStream :31 donde se utilizan como rutas de archivo. Esto permite al atacante escribir en archivos arbitrarios en el sistema.

En cuanto al análisis del impacto, es de riesgo Alto, ya que un atacante podría explotar esta vulnerabilidad escribiendo en cualquier archivo del sistema al que el servidor tenga acceso y comprometería la integridad y seguridad del sistema.

Por último, nos hace recomendaciones de mitigación para solucionar el problema. En este caso, dice que hay 40 proyectos open source para mitigarlo.

2- Docker Scout

Se ha probado las dos opciones que nos da la herramienta. Usando el terminal y ejecutando el siguiente código:

docker scout quickview bkimminich/juice-shop

Nos da un resumen de las vulnerabilidades donde podemos comprobar que hay al menos 10 vulnerabilidades críticas, 18 alta criticalidad y 32 medias. Al ver que hay vulnerabilidades en la aplicación, lo que haremos será correr el scanner total y ver detalladamente las vulnerabilidades.

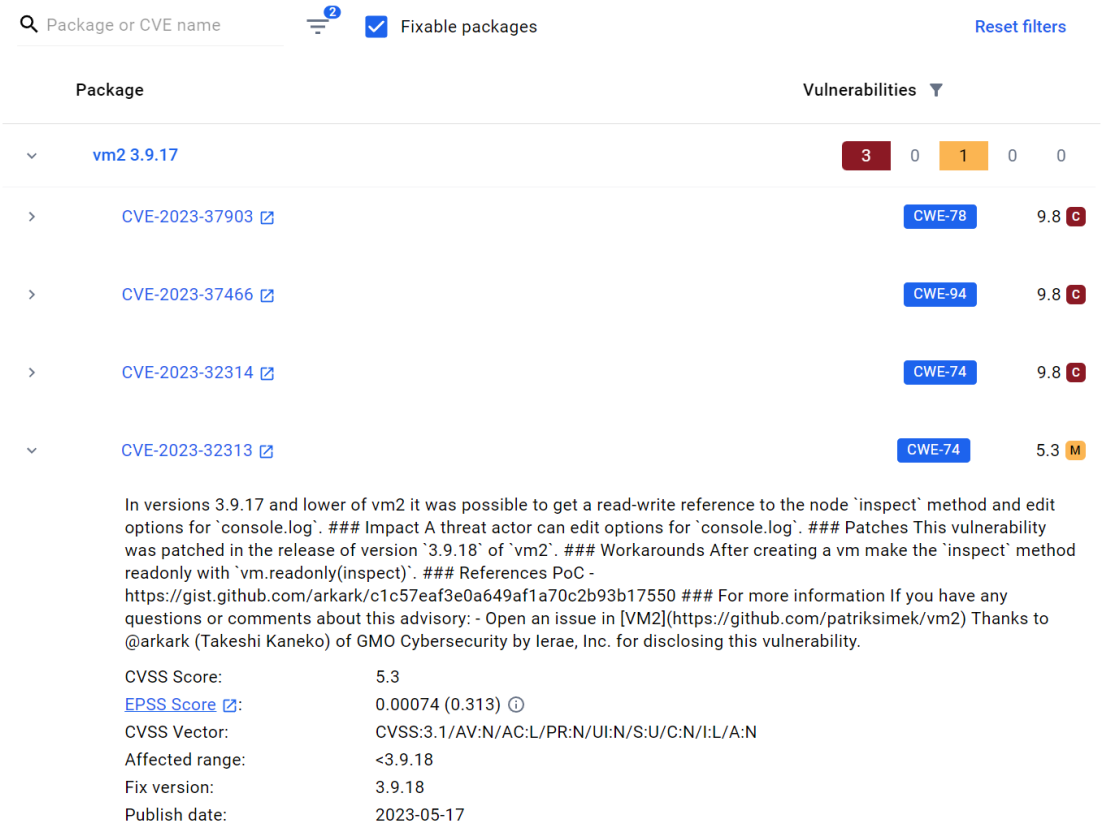
La siguiente opción vía terminal fue enviar a un fichero de texto el full scanner.
docker scout cves bkimminich/juice-shop >> VULNERABILITIES.TXT

También se podría imprimir por pantalla, pero al haber 78 vulnerabilidades es mejor enviarlo a un fichero de texto para su posterior observación de cada una de las vulnerabilidades. Si hacemos un risk assesment habria que mitigar primero las vulnerabilidades que sean críticas y que son un problema para la aplicación.

Librería	CVE	Severidad	Fixed?	Descripción
Vm2 3.9.17	2023-37903	Critica (9.8)	No	En vm2 para versiones hasta 3.9.19, la función de inspección personalizada de Node.js permite a los atacantes escapar del sandbox y ejecutar código arbitrario
	2023-37466	Critica (9.8)	No	En vm2, para versiones hasta la 3.9.19, la sanitización del controlador de Promise puede ser eludida, lo que permite a los atacantes escapar del sandbox y ejecutar código arbitrario.
	CVE-2023-32314	Critica (9.8)	Si versión 3.9.18	Existe una vulnerabilidad de escape de sandbox en vm2 para versiones hasta la 3.9.17. Esta aprovecha la creación

				inesperada de un objeto hospedado basado en la especificación de Proxy
	CVE-2023-32313	Media 5.3	Si versión 3.9.18	"En versiones 3.9.17 y anteriores de vm2 era posible obtener una referencia de lectura-escritura al método inspect de node y editar opciones para console.log

Esto es un ejemplo que el paquete vm2 3.9.17 es susceptible de ser comprometido y la solución sería actualizar a la versión 3.9.18 que arregla varias de las vulnerabilidades críticas y medias.



3- ZAP

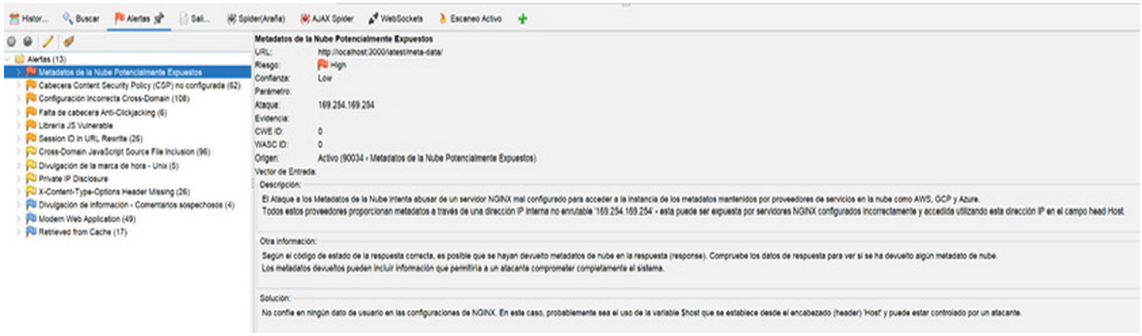
Una vez instalado la herramienta, la iniciamos, configuramos el proxy para poder interceptar el tráfico entre nuestro navegador y la aplicación de Juice Shop, esto lo conseguimos configurando ZAP como un proxy.
Mientras interactuamos en Juice shop, ZAP nos va mostrando el tráfico en tiempo real en la interfaz, podemos ver las solicitudes y respuestas HTTP y cualquier alerta de seguridad que ZAP detecte.

Mostramos una captura de parte de la práctica en la que se muestran 13 alertas de vulnerabilidades, entre las que se encuentra en primer lugar una alerta por metadatos de la nube potencialmente expuestos.

Cuando los metadatos de la nube están potencialmente expuestos, significa que la información sobre la configuración, recursos y datos de una infraestructura en

la nube puede ser accesible públicamente o por personas no autorizadas. Esto podría ocurrir debido a configuraciones incorrectas o malas prácticas de seguridad en la configuración de los servicios en la nube.

Para mitigar esta vulnerabilidad debemos tomar una serie de medidas como la revisión de manera exhaustiva de la configuración de la nube para identificar posibles exposiciones de metadatos, restringir el acceso a los mismos a través de políticas de acceso estrictas y autenticación multifactor. Encriptación de datos sensibles para proteger la información en reposo, así como una monitorización y auditoría continua para detectar y poder responder rápidamente a cualquier acceso no autorizado.



4- Contrast Scan

En esta herramienta, vemos como de igual manera, nos muestra diez vulnerabilidades, ordenadas por orden de prioridad y criticidad. En el primer lugar aparece Critical OPT.HTML.MissingPasswordFieldMasking que indica que se ha identificado una vulnerabilidad crítica relacionada con la falta de enmascaramiento de campos de contraseña en formularios HTML.
Cuando los campos de contraseña en un formulario HTML no están enmascarados correctamente, es decir, cuando el texto ingresado no se oculta (por ejemplo, con asteriscos o puntos) mientras se escribe, existe un riesgo de exposición de información confidencial. Esto significa que los caracteres de la contraseña ingresada pueden ser visibles para cualquier persona que esté

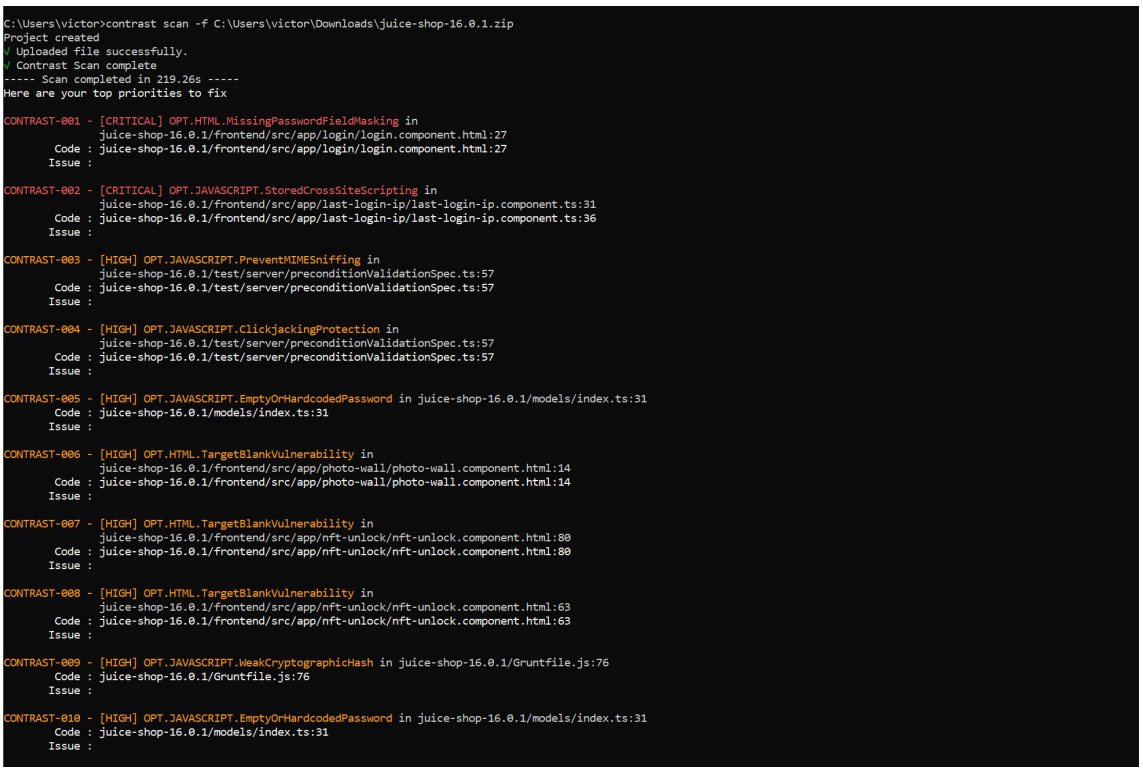
mirando la pantalla en ese momento, lo que podría comprometer la seguridad de la contraseña.

Esto puede ser explotado por atacantes potenciales que estén observando activamente la pantalla de un usuario mientras introducen su contraseña, lo que podría conducir a comprometer cuentas y datos confidenciales.

Para mitigar esta vulnerabilidad, es importante asegurarse de que los campos de contraseña en los formularios HTML estén correctamente enmascarados, de modo que los caracteres ingresados no sean visibles para otros usuarios. Esto se logra utilizando el atributo "type" con el valor "password" en el elemento de entrada HTML, lo que indica al navegador que debe enmascarar el texto ingresado en ese campo.

Un ejemplo podría ser : <input type="password" name="password">

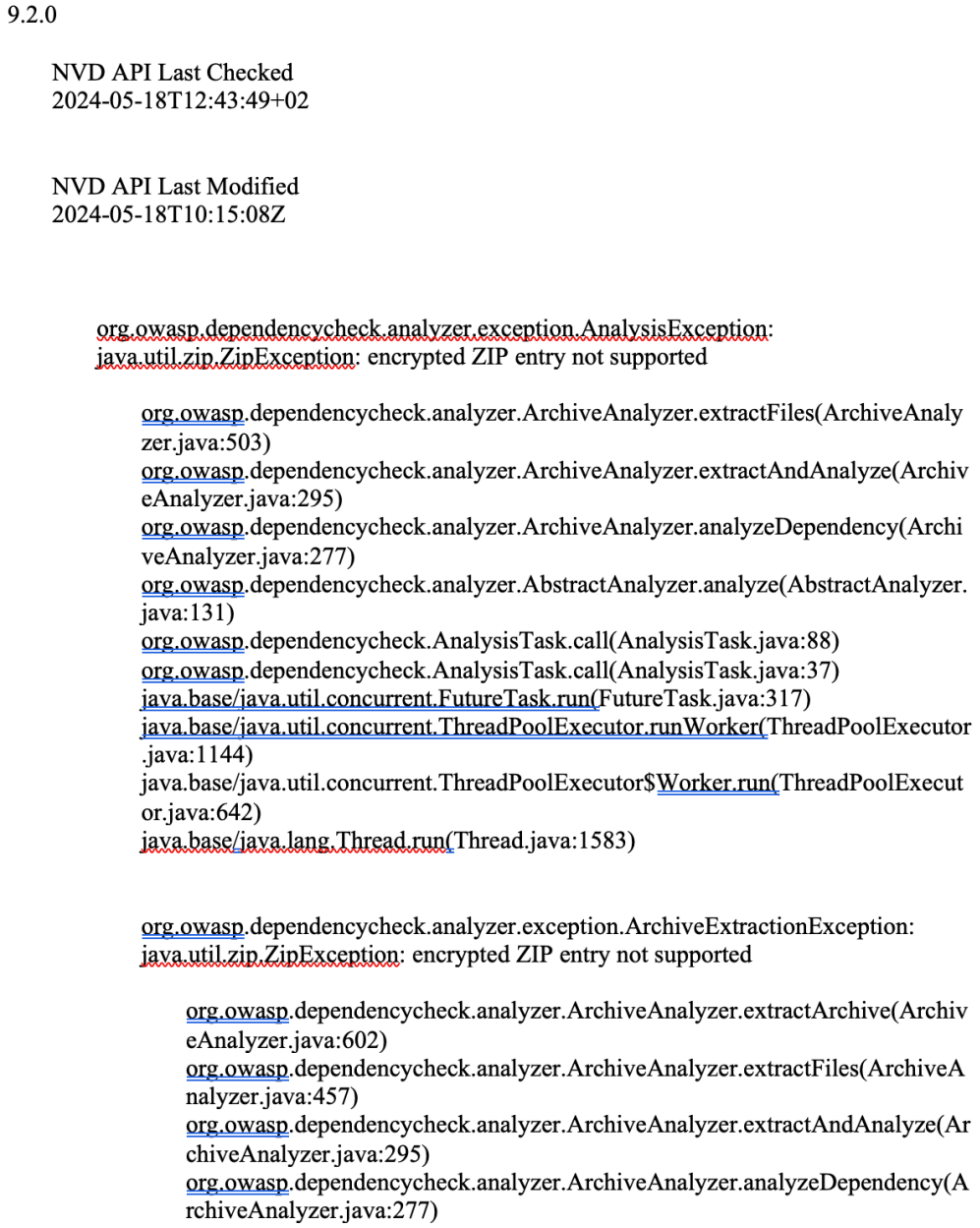
De este modo reduciremos significativamente el riesgo de exposición de información confidencial y se mejora la seguridad de la aplicación web.



5- OWASP Dependency-Check

Esta herramienta se ha ejecutado desde la terminal. Primero se procede a la instalación de la aplicación dependecy check y posteriormente en el directorio de la aplicación, se ejecuta el siguiente comando. “dependency-check --project JuiceShop --scan ./ --format ALL”

Esto creará un reporte dentro del directorio donde está la aplicación.
Allí se encontrará información sobre sus vulnerabilidades.
Se adjunta una captura de pantalla con información parcial del reporte.



9.2.0

NVD API Last Checked
2024-05-18T12:43:49+02

NVD API Last Modified
2024-05-18T10:15:08Z

org.owasp.dependencycheck.analyzer.exception.AnalysisException: java.util.zip.ZipException: encrypted ZIP entry not supported

org.owasp.dependencycheck.analyzer.ArchiveAnalyzer.extractFiles(ArchiveAnalyzer.java:503)
org.owasp.dependencycheck.analyzer.ArchiveAnalyzer.extractAndAnalyze(ArchiveAnalyzer.java:295)
org.owasp.dependencycheck.analyzer.ArchiveAnalyzer.analyzeDependency(ArchiveAnalyzer.java:277)
org.owasp.dependencycheck.analyzer.AbstractAnalyzer.analyze(AbstractAnalyzer.java:131)
org.owasp.dependencycheck.AnalysisTask.call(AnalysisTask.java:88)
org.owasp.dependencycheck.AnalysisTask.call(AnalysisTask.java:37)
java.base/java.util.concurrent.FutureTask.run(FutureTask.java:317)
java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1144)
java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:642)
java.base/java.lang.Thread.run(Thread.java:1583)

org.owasp.dependencycheck.analyzer.exception.ArchiveExtractionException: java.util.zip.ZipException: encrypted ZIP entry not supported

org.owasp.dependencycheck.analyzer.ArchiveAnalyzer.extractArchive(ArchiveAnalyzer.java:602)
org.owasp.dependencycheck.analyzer.ArchiveAnalyzer.extractFiles(ArchiveAnalyzer.java:457)
org.owasp.dependencycheck.analyzer.ArchiveAnalyzer.extractAndAnalyze(ArchiveAnalyzer.java:295)
org.owasp.dependencycheck.analyzer.ArchiveAnalyzer.analyzeDependency(ArchiveAnalyzer.java:277)

Tal y como se explica en la descripción de la herramienta, lo que hace es comparar las vulnerabilidades que tiene la aplicación con la base de datos de NVD.

En este reporte parcial que se ha adjuntado, se ha identificado una posible vulnerabilidad en dependencias de la aplicación. Concretamente ZipException: encrypted ZIP entry not supported. Ya que al encontrar estos archivos cifrados en las dependencias, no pueden ser analizados automáticamente por OWASP Dependency-Check, lo que podría ocultar vulnerabilidades potenciales.

Posible solución: No incluir archivos ZIP cifrados en las dependencias o analizarlos manualmente.

Integrantes del grupo de trabajo:

-Fernando Jesús Pérez Pérez
-Víctor Vila Gómez
-Estefanía Viegas Gómez