ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

Desarrollo de Software Seguro

Análisis estático del código

Kelly Sangoluisa

Contenido

1. Herramientas usadas

Para el siguiente análisis se usó algunas herramientas como:

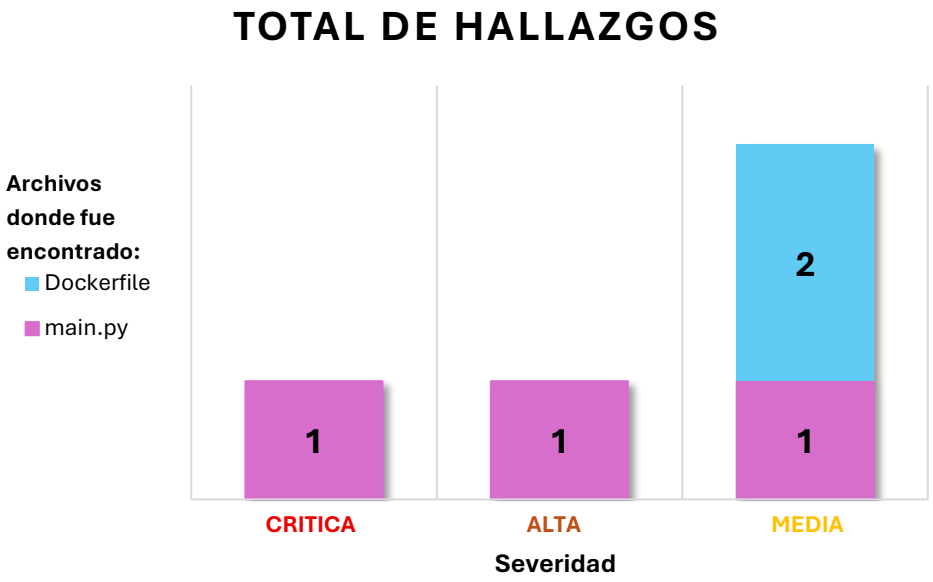
- **Bandit,** la cual es una herramienta de análisis estático especializada en el lenguaje Python, el lenguaje principal utilizado en el examen.
- **SonarCloud,** la usamos para complementar los resultados de Bandit y además de su uso para integrarla a GitHub y así automatizar los nuevos cambios realizados durante el examen.

2. Resumen ejecutivo

Mediante un análisis manual y con ayuda de las herramientas mencionadas anteriormente, se encontraron:

- 5 vulnerabilidades de seguridad.
- 1 de severidad critica, 1 de severidad alta y 3 severidad de media.

Además, estas se encuentran solo en dos archivos, *main.py* y *Dockerfile*.



Todas las vulnerabilidades encontradas en el archivo *main.py* han sido mitigadas, y las encontradas en el Dockerfile por el momento no, debido a las instrucciones del examen, pero se encuentra una recomendación correspondiente en el caso de implementar la mitigación.

3. Hallazgos

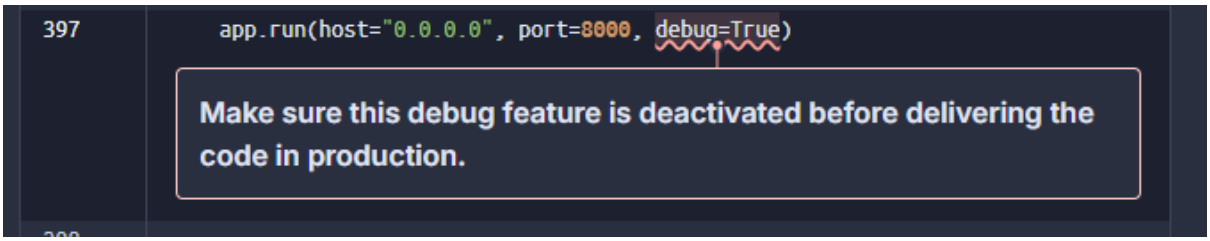
Hallazgo 1

Código de identificación del hallazgo: H01

Descripción de la vulnerabilidad:

Se encontró un *debug=true,* lo que significa que está habilitado el depurador, y en el caso de acceder desde el navegador se podría ingresar comandos peligrosos para todo el sistema y sus datos. lo cual lo hace una vulnerabilidad grave de seguridad si se manda así a producción.

Origen:
Archivo: .\app\main.py Línea: 397



Severidad:

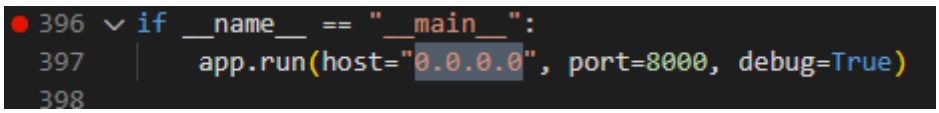
Con la herramienta de CVSS el puntaje obtenido es 9.8 por lo que es una Severidad Critica

Recomendación / Mecanismo de mitigación:
Aunque esto se puede usar en desarrollo, para evitar problemas como los detectados, se recomienda cambiarla a *debug=False*.

Hallazgo 2

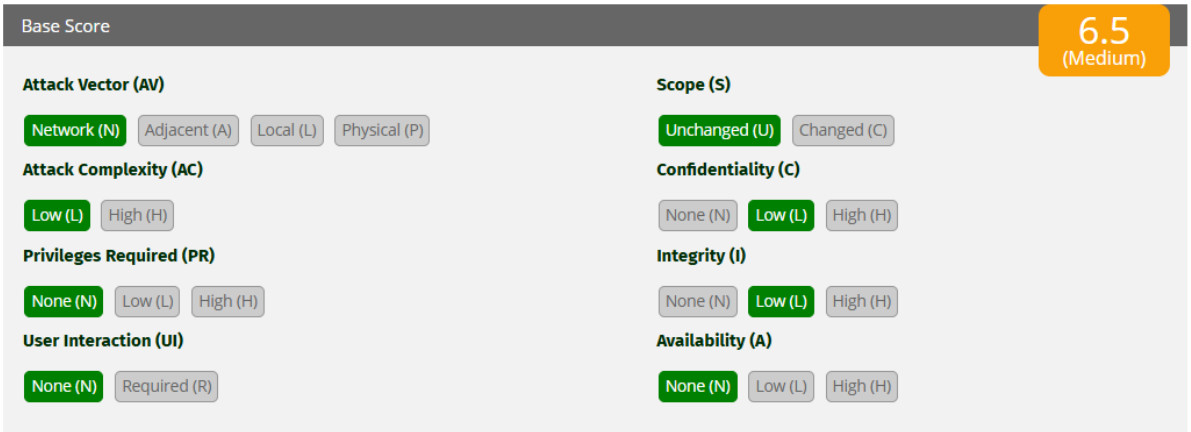
Código de identificación del hallazgo: H02
Descripción de la vulnerabilidad:
Se encontró que la aplicación está usando un *(host="0.0.0.0*"), lo cual hace que se acepten conexiones desde cualquier IP, aumentando los riesgos de vulnerabilidad si no se tiene un control adecuado de acceso o se aplica un firewall.

Origen:
Archivo: .\app\main.py Línea: 397

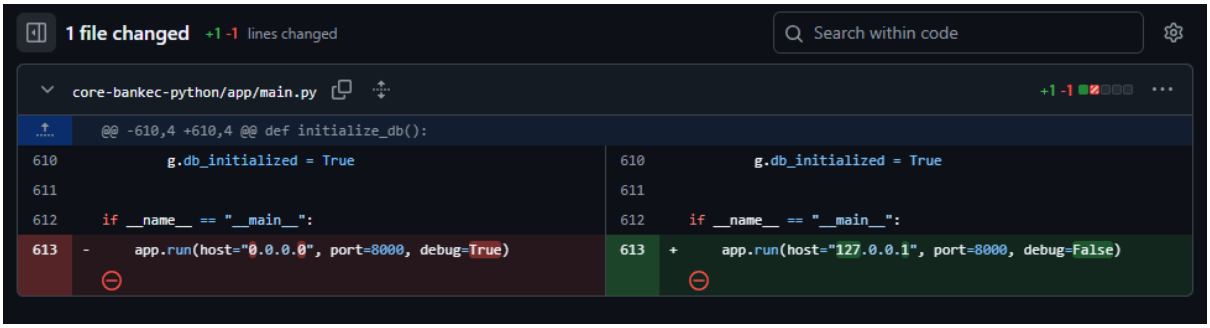


Severidad:

Con la herramienta de CVSS el puntaje obtenido es 6.5 por lo que es una Severidad Media



Recomendación / Mecanismo de mitigación:
Se debe proteger el acceso con firewalls o proxy inversos, o su vez usar interfaces específicas como *“host="127.0.0.1*" en caso de desarrollo y para cuando este en producción usar una ip especifica.

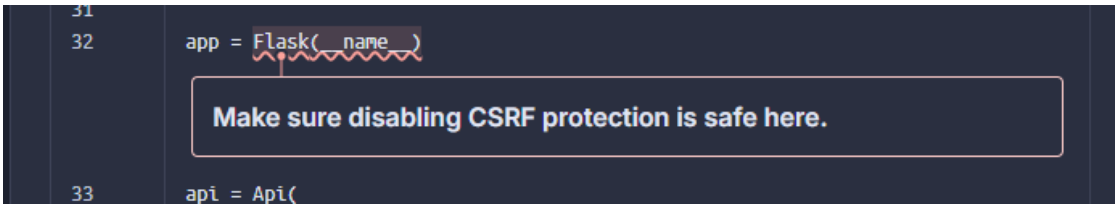


Hallazgo 3

Código de identificación del hallazgo: H03
Descripción de la vulnerabilidad:

Se encontró que la aplicación flask es sensible a ataques CSRF, ya que no se encuentra alguna configuración para controlarla, esto es una vulnerabilidad grande ya que se pueden realizar solicitudes peligrosas en nombre de algún usuario autenticado, poniendo en riesgo la confidencialidad e integridad de sus datos.

Origen:
Archivo: . /app/main.py Linea 32



Severidad:

Con la herramienta de CVSS el puntaje obtenido es 7.3 por lo que es una Severidad Alta

Recomendación / Mecanismo de mitigación:
Se recomienda habilitar la protección CSRF en la aplicación, o si es el caso configurarla.

En el examen como mecanismo de mitigación se usará verificación por medio de JWT para todos los endpoints, primero se envía el header *Authorization* en lugar de almacenarse en cookies y así evitar CSRF:

```
authorizations = {
    'Bearer': {
        'type': 'apiKey',
        'in': 'header',
        'name': 'Authorization',
        'description': "Enter your token in the format **Bearer <token>**"
    }
}
```

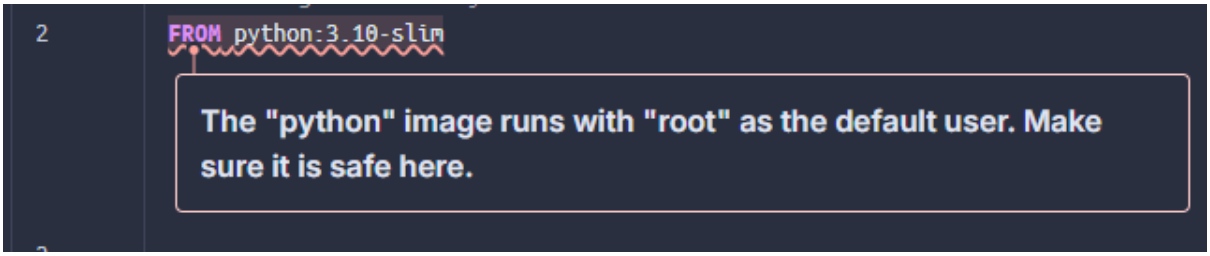
Luego el decorator asegura que el token este en el header *Authorization,* y validamos el token:

```
    @wraps(f)
    def decorated(*args, **kwargs):
        # Obtener el header de autorización
        auth_header = request.headers.get("Authorization", "")

        if not auth_header.startswith("Bearer "):
            abort(401, "Authorization header missing or invalid")

        # Extraer el token
        token = auth_header.split(" ")[1]

        # Obtener el gestor JWT desde el contexto de la aplicación
        jwt_manager = current_app.jwt_manager

        # Verificar el token
        payload = jwt_manager.verify_token(token)
        if not payload:
            abort(401, "Invalid or expired token")

```

Hallazgo 4

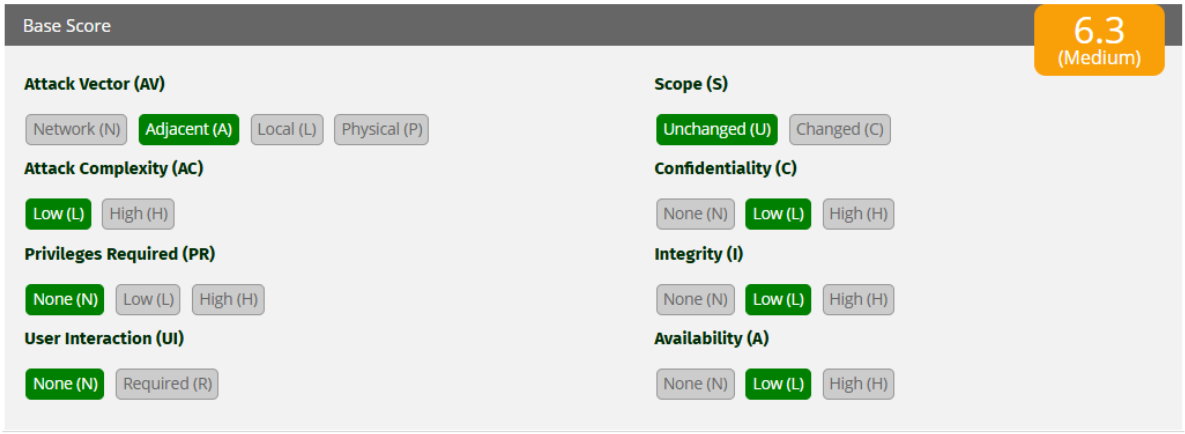
Código de identificación del hallazgo: H04
Descripción de la vulnerabilidad:
En la imagen de Docker se encontró que python:3.10-slim ejecuta los procesos dentro del contendor con el usuario root por defecto, siendo una vulnerabilidad por los privilegios que se obtienen con el usuario root, ya que si se tiene acceso a este se compromete el sistema.

Origen:
Archivo: ./ Dockerfile Linea 2



Severidad:

Con la herramienta de CVSS el puntaje obtenido es 6.3 por lo que es una Severidad Media

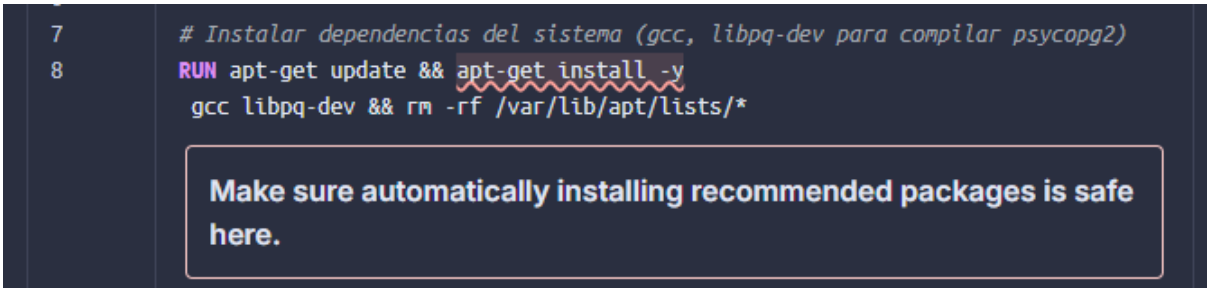


Recomendación / Mecanismo de mitigación:
Crear y configurar un usuario, con los permisos necesarios, no con acceso grant como el root, para mitigar el riesgo de la obtención maliciosa del usuario.

Hallazgo 5

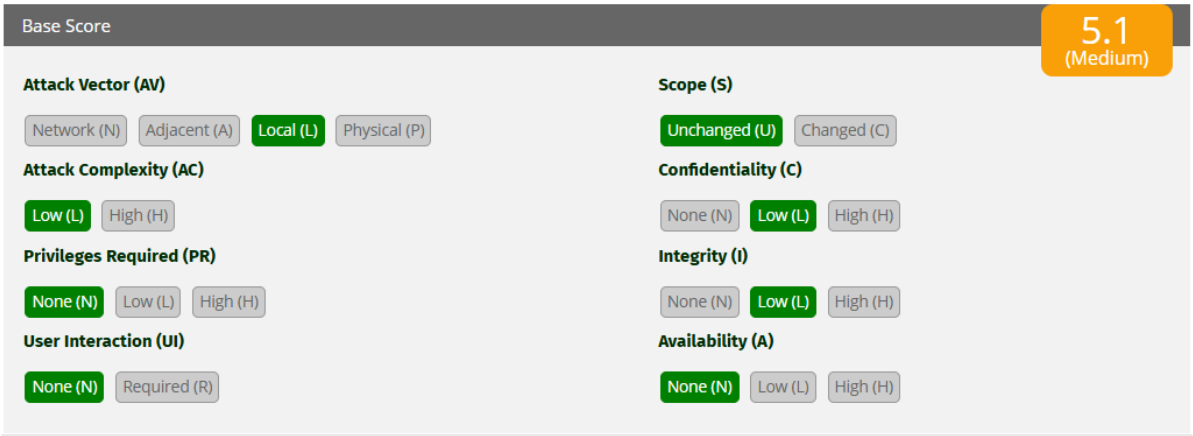
Código de identificación del hallazgo: H05
Descripción de la vulnerabilidad:
En el archivo Docker se encontró que el comando para instalar paquetes, *apt-get install -y gcc libpq-dev*, también instala paquetes recomendados, y no solamente los necesarios, lo cual puede convertirse en una vulnerabilidad ya que al aumentar paquetes recomendados de los cuales no sabemos si son seguros aumentar el riesgo de ataques por ese medio.

Origen:
Archivo: ./ Dockerfile Linea 8



Severidad:

Con la herramienta de CVSS el puntaje obtenido es 5.1 por lo que es una Severidad Media



Recomendación / Mecanismo de mitigación:
Agregar el comando *--no-install-recommends* para que evitar que se instalen paquetes innecesarios.

4. Anexos

- SonarCloud: https://sonarcloud.io/
- CVSS Calculator: https://www.first.org/cvss/calculator/3-1
- Bandit: https://bandit.readthedocs.io/en/latest