README5.md

Índice de subtermas:

6.17 Abuso de APIs

En caso de que veáis que tras desplegar el laboratorio, siguen habiendo errores en el despliegue de ciertos contenedores, probad a hacer un 'docker rm $(docker ps -a -q) –force' y aplicad el último comando de los 3 mencionados anteriormente para volver a desplegar los contenedores. Llegará un momento en el que todos serán desplegados sin ningún problema.

Por otro lado, si de pronto véis que el comando 'docker rm $(docker ps -a -q) –force' os da algún problema, esperad unos segundos y volved a probar el comando hasta que veáis que todos los contenedores han sido eliminados.

Cuando hablamos del abuso de APIs, a lo que nos referimos es a la explotación de vulnerabilidades en las interfaces de programación de aplicaciones (APIs) que se utilizan para permitir la comunicación y el intercambio de datos entre diferentes aplicaciones y servicios en una red.

Un ejemplo sencillo de API podría ser la integración de Google Maps en una aplicación de transporte. Imagina que una aplicación de transporte necesita mostrar el mapa y la ruta a seguir para que los usuarios puedan ver la ubicación del vehículo y el camino que se va a seguir para llegar a su destino. En lugar de crear su propio mapa, la aplicación podría utilizar la API de Google Maps para mostrar el mapa en su aplicación.

En este ejemplo, la API de Google Maps proporciona una serie de funciones y protocolos que permiten a la aplicación de transporte comunicarse con los servidores de Google y acceder a los datos necesarios para mostrar el mapa y la ruta. La API de Google Maps también maneja la complejidad de mostrar el mapa y la ruta en diferentes dispositivos y navegadores, lo que permite a la aplicación de transporte centrarse en su funcionalidad principal.

Adicionalmente, una de las utilidades que vemos en esta clase es Postman. Postman es una herramienta muy popular utilizada para probar y depurar APIs. Con Postman, los desarrolladores pueden enviar solicitudes a diferentes endpoints y ver las respuestas para verificar que la API está funcionando correctamente. Sin embargo, los atacantes también pueden utilizar Postman para explorar los endpoints de una API en busca de vulnerabilidades y debilidades de seguridad.

Algunos endpoints de una API pueden aceptar diferentes métodos de solicitud, como GET, POST, PUT, DELETE, etc. Los atacantes pueden utilizar herramientas de fuzzing para enviar una gran cantidad de solicitudes a un endpoint en busca de vulnerabilidades. Por ejemplo, un atacante podría enviar solicitudes

GET a un endpoint para enumerar todos los recursos disponibles, o enviar solicitudes POST para agregar o modificar datos.

Algunas de las vulnerabilidades comunes que se pueden explotar a través del abuso de APIs incluyen:

- Inyección de SQL: los atacantes pueden enviar datos maliciosos en las solicitudes para intentar inyectar código SQL en la base de datos subyacente.
- Falsificación de solicitudes entre sitios (CSRF): los atacantes pueden enviar solicitudes maliciosas a una API en nombre de un usuario autenticado para realizar acciones no autorizadas.
- Exposición de información confidencial: los atacantes pueden explorar los endpoints de una API para obtener información confidencial, como claves de API, contraseñas y nombres de usuario.

Para evitar el abuso de APIs, los desarrolladores deben asegurarse de que la API esté diseñada de manera segura y que se validen y autentiquen adecuadamente todas las solicitudes entrantes. También es importante utilizar cifrado y autenticación fuertes para proteger los datos que se transmiten a través de la API.

Los desarrolladores pueden utilizar herramientas como Postman para probar la API y detectar posibles vulnerabilidades antes de que sean explotadas por los atacantes.

A continuación, se proporciona el enlace al proyecto de Github que utilizamos para desplegar con Docker el laboratorio vulnerable donde poder practicar la enumeración de APIs:

- crAPI: https://github.com/OWASP/crAPI

6.17.1 Ejercicio

DISCLAIMER: Si a la hora de desplegar el laboratorio con Docker, os encontráis con problemas y alguno de los contenedores que se despliegan véis que causan error, probad a desplegar como alternativa el laboratorio de desarrollo. Primero instalad la última versión de ‘docker-compose‘ y una vez hecho, ejecutad los siguientes comandos:

```
curl -o docker-compose.yml
https://raw.githubusercontent.com/OWASP/crAPI/develop/deploy/docker/docker-
compose.yml
VERSION=develop docker-compose pull
VERSION=develop docker-compose -f docker-compose.yml –compatibility up -d
```

Empecemos:

Descargamos el repo y nos vamos a la carpeta con el docker compose. Allí descargamos las imágenes Docker.

```
git clone https://github.com/OWASP/crAPI.git
cd crAPI/deploy/docker
docker-compose pull
```

Ahora desplegamos el laboratorio.

```
docker compose -f docker-compose.yml --compatibility up -d
```

A veces no funciona a la primera. El laboratoria es inestable, por lo que si no funciona a la primera, probad a ejecutar el comando varias veces empezando desde cero, borrando contenedores e imágenes. Merece la pena. La comunidad ha documentado algunos errores en su repositorio: https://github.com/OWASP/crAPI/blob/main/docs/troubleshooting.md

Entonces, vamos a `http://localhost:8080` y vemos que hay una página para iniciar sesión. Vamos a Sing Up y creamos un usuario.

Ahora abrimos el inspector de elementos y vamos a la pestaña de Network. Vamos a la pestaña de XHR y nos logueamos. Tenemos que ver una petición a `http://localhost:8888/identity/api/auth/login`, que si la inspeccionamos veremos:

- **Headers**
- **Payload**
  - view source

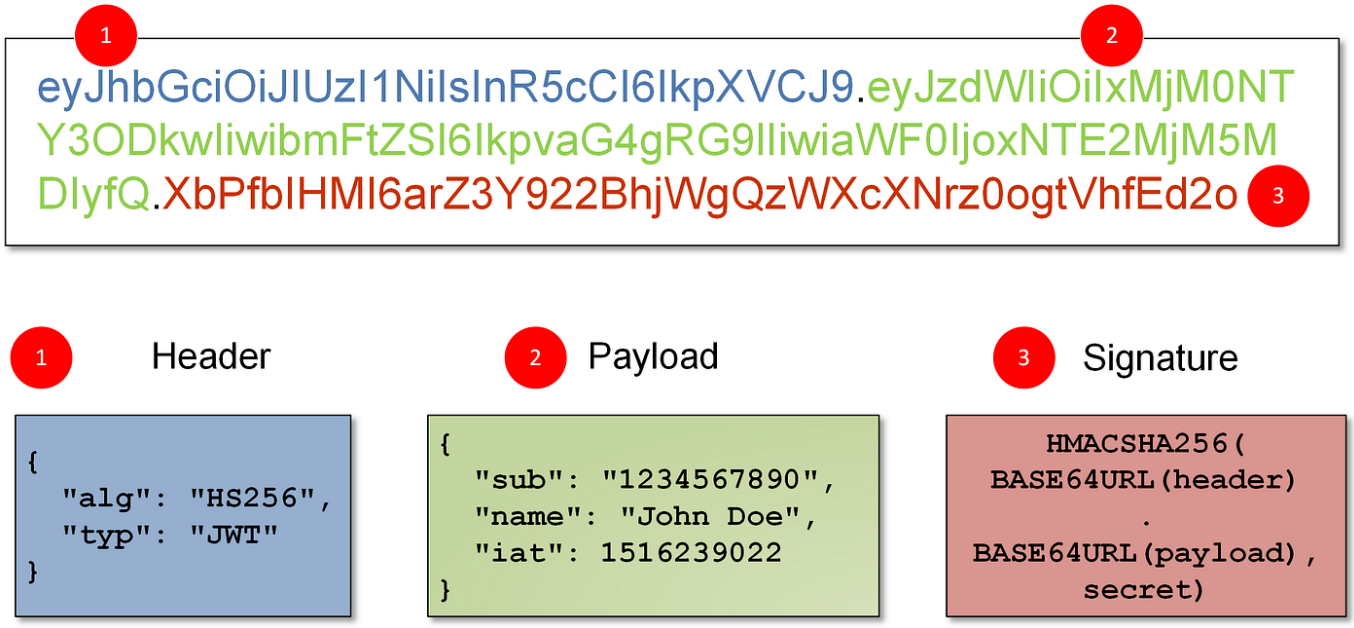
    ```
    {email: "man@invent.com", password: "Man1234$"}
    ```

- **Preview**
- **Response**

  ```
  {
    "token":
  "eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJtYW5AaW52ZW50LmNvbSIsInJvbGUiOiJ1c2Vy
  IiwiaWF0IjoxNzA4MjAwMjk2LCJleHAiOjE3MDg4MDUwOTZ9.EKGBU4uxfpWxlZiRmtRG6
  m6JUrZsVsEf7xzSppIE9FlbpxTackor_KYdBLZOJYK5D3KRkbO9KCfa4GbnccjdmsSFipN
  JDZkATa-hC51wYvesaA15f0yTm26sb6W-
  W5icuv269kkWVaCw_3SCSOzoU3L50YoY0pZH7wPbf4-
  k6vU4nYI7gVAWIPZloJfKwpjqjWMFA2oZHBFg6NP5YjKLyhQAYdak0fK89vVFadLdLUy_m
  mEy3nVgfpV2_2wNPLQc2rDX9XA4WemF5o1rI484JjXaq7Qa6EMBFTc2l0xDZQJT0ok9rPs
  5jPvyj8Mamt01CX13tV_jd4gybsJhm2O4kA",
    "type": "Bearer",
    "message": null
  }
  ```

- **Initiator**
- **Timing**

He detallado la request y el response porque es en lo que tendremos que fijarnos. Vemos que el token es un JWT.



6.18 Abuso de subidas de archivos

6.19 Prototype Pollution

6.20 Ataques de transferencia de zona (AXFR - Full Zone Transfer)

README.md principal

README4.md <--> README6.md