Tarea 2: Consultar Fabricantes de tarjetas de red a través de una API

Javier Rivera Pizarro, javier.riverapi@alumnos.uv.cl

Juan Pablo Páez Salas, juan.paez@alumnos.uv.cl

Fabian Cornejo Silva, fabian.cornejo@alumnos.uv.cl

1. Introducción

En esta tarea, se implementará una herramienta llamada OUILookup, la cual tiene la función de consultar el fabricante de una red a partir de su dirección MAC usando una API pública. Para cumplir con esta tarea tendremos los siguientes objetivos:

- Interactuar con una API REST.
- Manejar parámetros de líneas de comandos.
- Programación funcional.
- Mostrar los fabricantes de direcciones MAC a través de una tabla ARP.

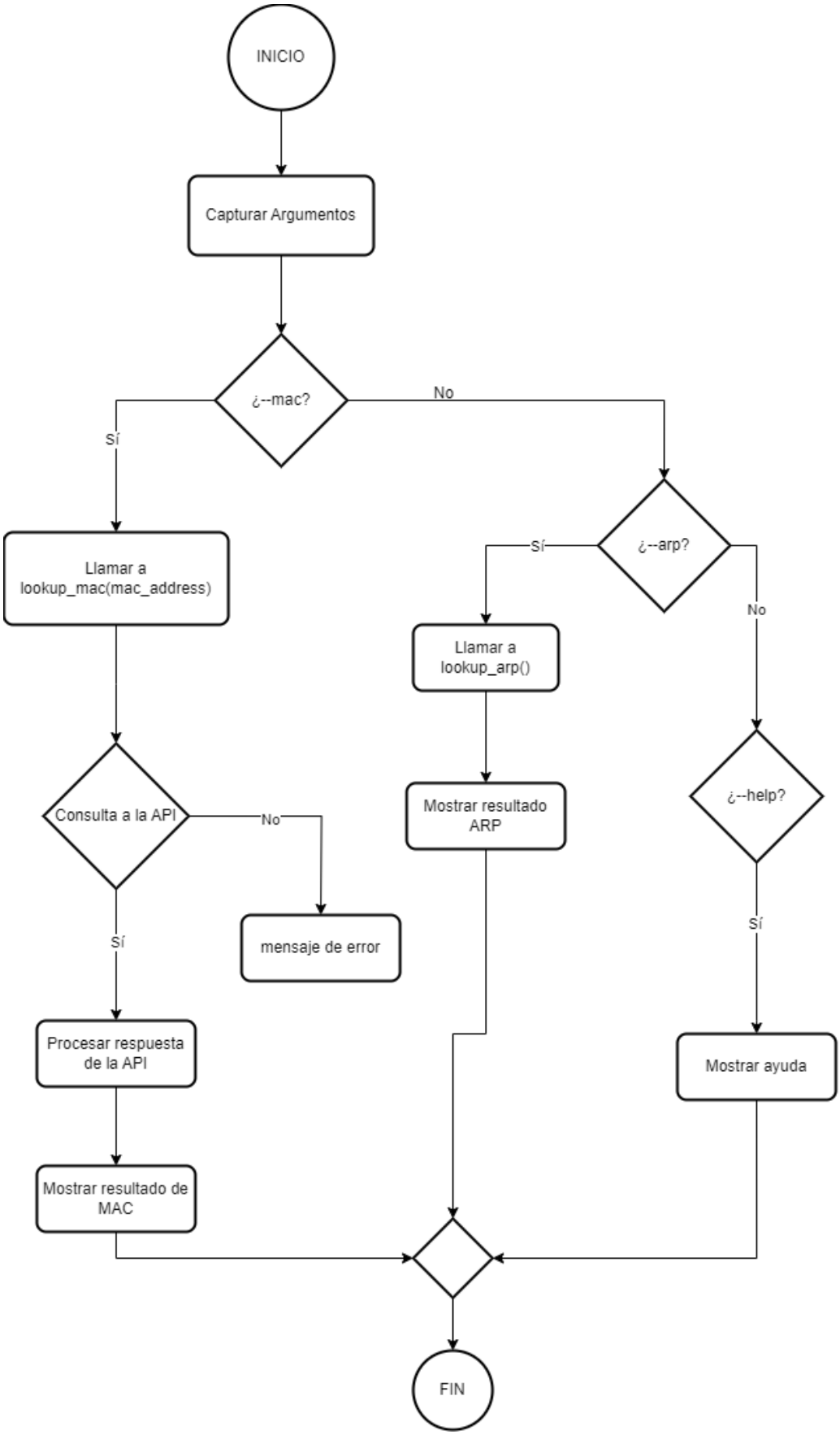
2. Descripción del problema y diseño de la solución

Como se mencionó en la sección anterior tendremos como objetivo general el implementar una herramienta basada en líneas de comandos para poder consultar con los fabricantes de una tarjeta de red a partir de su dirección MAC usando una API REST.

Los objetivos específicos serán:

- Diseñar diagrama de flujo.
- Desarrollar un programa que procese parámetros de línea de comandos.
- Integrar una API publica para consultar fabricantes a partir de direcciones MAC.
- Mostrar la información de las direcciones MAC de la tabla ARP.
- Documentar la implementación del código.
- Pruebas del código.

Diagrama de flujo:


INICIO
Capturar Argumentos
¿--mac?
No
Sí
Llamar a lookup_mac(mac_address)
¿--arp?
Sí
No
Llamar a lookup_arp()
Consulta a la API
No
Mostrar resultado ARP
¿--help?
mensaje de error
Sí
Sí
Procesar respuesta de la API
Mostrar ayuda
Mostrar resultado de MAC
FIN

3. Implementación

El código se ha implementado en Python para desarrollar una herramienta de línea de comandos llamada **OUILookup**, cuyo propósito es consultar el fabricante de una tarjeta de red a partir de su dirección MAC, utilizando una API pública de búsqueda de MAC (https://maclookup.app). A continuación, se explican las partes más importantes del código y los principales desafíos enfrentados durante la implementación.

3.1. Función lookup_mac(mac_address)

```
def lookup_mac(mac_address):                                    # Funcion buscar direccion mac especificada y
    """Consulta la API para obtener el fabricante a partir de una MAC."""
    try:
        start_time = time.time()                                # Variable que empieza a contar el tiempo de ejecucion

        # Crear un contexto SSL que no verifica certificados
        context = ssl._create_unverified_context()

        # Configuración de la conexión HTTP con la API
        conn = http.client.HTTPSConnection(API_HOST,context=context)
        conn.request("GET", f"{API_PATH}{mac_address}")         # Donde se realiza la solicitud de tipo get solicitando al info
        response = conn.getresponse()                           # Respuesta del servidor
        data = response.read()                                  # Lectura de la respuesta
        conn.close()                                            # Fin de la conexion

        response_time = int((time.time() - start_time) * 1000)  # Tiempo en ms
        result = json.loads(data)                               # Conversion de la respuesta de JSON a un diccionario pytho

        if isinstance(result, dict) and result.get('company'):                  # Muestra en pantalla la direccion ma
            return f"MAC address : {mac_address}\nFabricante : {result['company']}\nTiempo de respuesta: {response_time}ms"
        else:                                                                   # Muestra en pantalla la direccion ma
            return f"MAC address : {mac_address}\nFabricante : Not found\nTiempo de respuesta: {response_time}ms"

    except Exception as e:
        return f"Error consultando la MAC: {e}"
```

Esta función es la responsable de consultar el fabricante de una dirección MAC específica utilizando la API pública. Los pasos clave de esta función son:

- **Contexto SSL no verificado:** dado que en el entorno de ejecución hubo problemas se implementó "ssl._create_unverified_context()" para omitir esta verificación y evitar que el programa falle debido a errores de certificado, en cambio cuando se ejecutaba en visual studio code y apretaba ejecutar esto hacia que el comando si funcionara, pero esto era un comando muy largo ya que tomaba toda la ruta para la ejecución y es por eso por lo que uso el "ssl._create_unverified_context()".
- **Solicitud HTTP GET:** esto se usa importando la librería de http.client para poder realizar una solicitud HTTPS a la API, cuando reciba la respuesta la pasara de un JSON a un diccionario de Python para extraer la información relevante.
- **Manejo de tiempo de respuesta**: Se mide el tiempo de respuesta de la consulta para proporcionar retroalimentación sobre el rendimiento.
- **Manejo de errores**: En caso de que ocurra algún error durante la solicitud a la API o el procesamiento de la respuesta, se captura la excepción y se muestra un mensaje de error apropiado.

3.2. Función lookup_arpmac(mac_address)

Esta función se encarga de buscar el fabricante de una dirección MAC específica obtenida de la tabla ARP mediante la API pública. Es una subfunción de lookup_arp(), ya que se

invoca cada vez que se encuentra una dirección MAC en la tabla ARP para consultar su fabricante.

```
def lookup_arpmac(mac_address):
    """Consulta la API para obtener el fabricante a partir de una MAC."""
    try:
        # Configuración de la conexión HTTP con la API
        conn = http.client.HTTPSConnection(API_HOST)
        conn.request("GET", f"{API_PATH}{mac_address}")      # Donde se realiza
        response = conn.getresponse()                        # Respuesta del se
        data = response.read()                               # Lectura de la re
        conn.close()                                         # Fin de la conexi
        result = json.loads(data)                            # Conversion de la

        if isinstance(result, dict) and result.get('company'):               #
            return f"{mac_address} / {result['company']}"
        else:
            return 0

    except Exception as e:
        return f"Error consultando la MAC: {e}"
```

Lo mas importante de esta parte del código es:

Conexión HTTPS: Se establece una conexión HTTPS con la API para obtener información sobre el fabricante asociado con la dirección MAC.

- La solicitud HTTP GET se realiza para obtener los datos relacionados con la dirección MAC.
- **Nota importante**: No se usa el contexto SSL aquí, porque esta función no lo tiene incorporado.

3.3. Función lookup_arp()

Esta función es responsable de obtener las direcciones MAC presentes en la tabla ARP del sistema y luego consultar sus fabricantes:

```python
def lookup_arp():
    """Obtiene la tabla ARP en Windows y consulta los fabricantes."""
    try:
        arp_output = os.popen('arp -a').read()
        arp_lines = arp_output.splitlines()
        results = []
        result = "IP/MAC/Vendor:"                           # Agrega formato a el arreglo
        results.append(result)

        for line in arp_lines:                              # Busca las direcciones mac y las compara de
            # Regex para capturar direcciones MAC
            match = re.search(r'([0-9a-fA-F]{2}[:-]){5}([0-9a-fA-F]{2})', line)
            if match:
                mac_address = match.group(0)
                result = lookup_arpmac(mac_address)
                if result != 0:                             # Se agrega a el arreglo si este tiene fabr
                    results.append(result)


        return "\n".join(results) if results else "No se encontraron direcciones MAC en la tabla ARP."
    except Exception as e:
        return f"Error obteniendo la tabla ARP: {e}"
```

- **Comando ARP**: La función ejecuta el comando arp -a en el sistema operativo para obtener una lista de las direcciones MAC de los dispositivos conectados a la red local.
- **Expresión regular**: Utiliza una expresión regular (regex) para capturar las direcciones MAC presentes en la salida de la tabla ARP.
- **Llamada a lookup_arpmac**: Para cada dirección MAC encontrada, se llama a la función lookup_arpmac, que realiza una consulta a la API para obtener el fabricante correspondiente.
- **Formato de resultados**: Los resultados se formatean en un arreglo, agregando las direcciones MAC y sus respectivos fabricantes, y se devuelven en formato de texto.

3.4. Función main(argv)

Esta función es el punto de entrada principal del programa. Su objetivo es procesar los argumentos de la línea de comandos y llamar a las funciones correspondientes:

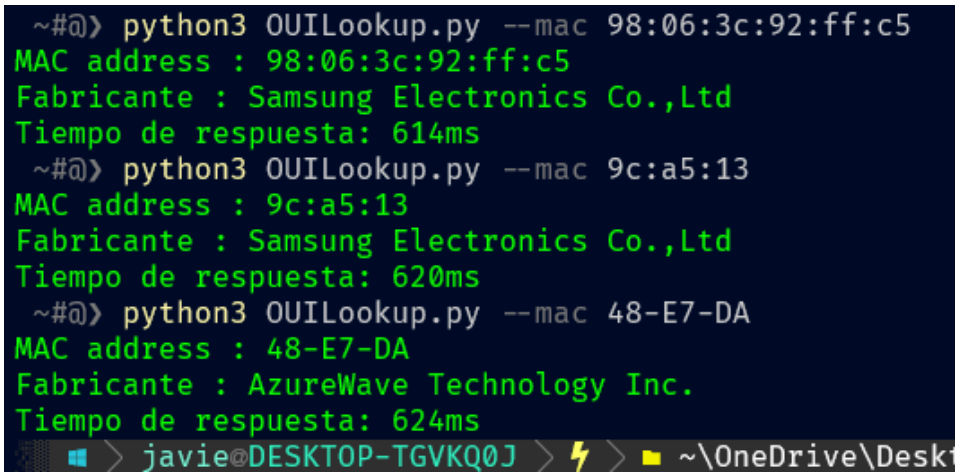
```python
def main(argv):
    mac_address = '' # Instancia de la direccion mac como vacia
    arp_flag = False # Instancia de la variable arp_flag como falsa para variable de control

    try:
        opts, args = getopt.getopt(argv, "", ["mac=", "arp", "help"])  # Aqui se obtienen lo
    except getopt.GetoptError as err:
        print(err)                                                     # Si falla muestra er
        sys.exit(2)

    for opt, arg in opts:                                              # Comparacion del arg
        if opt == '--mac':                                             # ingresa en mac_addr
            mac_address = arg
        elif opt == '--arp':                                           # ingresa en arp_flag
            arp_flag = True
        elif opt == '--help':                                                # Muestra en la
            print('Uso: OUILookup.py --mac <mac_address> | --arp | --help')
            sys.exit()

    if mac_address:
        result = lookup_mac(mac_address)                #si la variable mac_address tiene a
        print(result)
    elif arp_flag:                                      #si la variable arp_flag tiene algu
        result = lookup_arp()
        print(result)
    else:
        print('No se proporcionó ninguna opción válida. Usa --help para más información.')

if __name__ == "__main__":
    main(sys.argv[1:])
```

- **Procesamiento de argumentos**: Utiliza la librería `getopt` para capturar las opciones de la línea de comandos. Se aceptan las opciones `--mac` para consultar un fabricante específico, `--arp` para obtener fabricantes de las MAC en la tabla ARP y `--help` para mostrar las instrucciones de uso.
- **Flujo de control**: Dependiendo de los argumentos proporcionados, llama a `lookup_mac()` para una dirección MAC específica o a `lookup_arp()` para la tabla ARP.

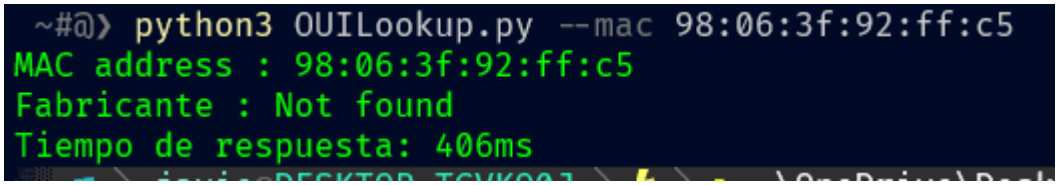
4. Pruebas

Primero haremos el –help, luego haremos los casos de prueba solicitados los cuales son:

- python3 OUILookup.py --mac 98:06:3c:92:ff:c5
- python3 OUILookup.py --mac 9c:a5:13
- python3 OUILookup.py --mac 48-E7-DA



Ahora en caso de que busquemos una mac que no esté en la base de datos veremos el siguiente resultado:



En la parte de fabricante saldrá el mensaje de "Not found".

5. Discusión y conclusiones

5.1. Funcionamiento de las MAC

Lo primero que debemos preguntarnos es: ¿qué es la aleatorización de direcciones MAC? Básicamente, es una técnica que permite ocultar la dirección MAC real de un dispositivo, generando una dirección MAC aleatoria cada vez que el dispositivo se conecta a una red Wi-Fi. Esta funcionalidad es especialmente útil para proteger la privacidad del usuario, ya que evita que su dispositivo sea rastreado mediante su dirección MAC única, lo cual es particularmente importante cuando se conectan a redes públicas, donde los riesgos de seguimiento y exposición son mayores.[1][2][3]

5.2. Resumen de los resultados obtenidos

La implementación de **OUILookup** ha sido exitosa al cumplir con los objetivos de consultar fabricantes de direcciones MAC tanto individuales como presentes en la tabla ARP, utilizando una API pública. Se implementó un bypass de SSL para resolver los problemas de verificación de certificados, lo que permitió realizar las consultas correctamente.

5.3. Desafíos y errores

El principal desafío fue el error de verificación de certificados SSL, que se solucionó con la función `ssl._create_unverified_context()`. También se enfrentaron retos al extraer y procesar las direcciones MAC de la tabla ARP, lo que se resolvió mediante expresiones regulares.

5.4. Conclusión

OUILookup es una herramienta funcional que ha superado los principales retos encontrados. Con algunas mejoras en la validación de entradas y manejo de SSL, puede optimizarse aún más.

6. Referencias

[1]https://www.tarlogic.com/bsam/es/controles/mac-aleatoria-bluetooth/#:~:text=En%20Bluetooth%2C%20existen%20dos%20tipos,dispositivo%20y%20es%20posible%20cambiarla.

[2] https://support.apple.com/es-cl/guide/security/secb9cb3140c/web

[3]https://www.redeszone.net/noticias/wifi/direccion-mac-aleatoria-wifi/