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

Rodrigo Pino Araya, rodrigo.pino@alumnos.uv.cl

1. Introducción

En el contexto redes informáticas, las direcciones MAC (Media Access Control) son identificadores únicos asignados a las interfaces de red de los dispositivos. Estos identificadores son asignados por los fabricantes y permiten que los dispositivos puedan ser identificados dentro de una red local. Conocer el fabricante de una tarjeta de red es importante en diversos escenarios, como la gestión de redes, auditorías de seguridad o simplemente para fines de diagnóstico.

Este informe documenta el desarrollo de "OUILookup", una herramienta basada en línea de comandos escrita en Python que permite consultar el fabricante de una tarjeta de red dado su identificador MAC. OUILookup utiliza una API pública para obtener los datos de los fabricantes, facilitando la identificación de dispositivos dentro de una red.

2. Descripción del problema y diseño de la solución

Cuando se administra una red, es común encontrar múltiples dispositivos conectados a la misma, y puede resultar necesario identificar qué dispositivo pertenece a cada dirección MAC. Sin embargo, las direcciones MAC no proporcionan directamente información legible para los humanos sobre el dispositivo o su fabricante, ya que están representadas en un formato hexadecimal. Para identificar el fabricante a partir de una MAC, es necesario consultar una base de datos de fabricantes, comúnmente conocida como OUI (Organizationally Unique Identifier).

El problema que aborda este trabajo es la falta de una herramienta sencilla y accesible para consultar esta información desde la línea de comandos, usando una API pública para acceder a la base de datos OUI.

Para eso, se realizo un programa en Python llamada “OUILookup” que permite al usuario obtener el fabricante de una tarjeta de red a partir de su dirección MAC.

3. Implementación

Para esto, tuve que usar bibliotecas para realizar el programa y cumpla su función

a) requests: permite hacer peticiones HTTP desde Python de manera sencilla

b) getopt: facilita el procesamiento de argumentos en la línea de comandos

c) sys: proporciona funciones y variables para interactuar con el S.O

d) time: permite medir y gestionar el tiempo, como calcular tiempos de ejecución

e) subprocess: permite ejecutar comandos del sistema operativo desde Python y obtener su salida

f) re: permite realizar operaciones con expresiones regulares

```
import requests          # Para realizar solicitudes HTTP a la API de búsqueda de MAC
import getopt            # Para manejar argumentos de línea de comandos
import sys               # Para acceder a los argumentos y manejar salidas del programa
import time              # Para medir tiempos de respuesta
import subprocess        # Para ejecutar comandos del sistema, específicamente el comando 'arp -a'
import re                # Para manejar expresiones regulares y procesar la salida del comando 'arp'
```

Figura 1. Muestra de las librerías usadas para el programa

ya teniendo estas bibliotecas incorporadas y anteriormente mencionadas, podemos iniciar con la implementación del programa y poder realizar todo con total eficiencia, a continuación, usaremos unas funciones que fueron creadas para un uso en específico.

1) get_vendor_from_mac (mac): el propósito de esta función, es de tomar la dirección MAC como una entrada y consulta a la API para obtener y mostrar el nombre del fabricante de la tarjeta de red asociada con esa MAC, donde esta primero "Normalizara" la MAC para realizar un mejor uso, que lo saca desde la URL de "maclookup.app" para consultar el nombre del fabricante, también realizara una solicitud de HTTP para obtener el fabricante y al final entrega el valor del fabricante

```
def get_vendor_from_mac(mac):
    """
    Obtiene el nombre del fabricante (vendor) asociado a una dirección MAC utilizando la API de maclookup.app.
    Args:
        mac (str): Dirección MAC a consultar.
    Returns:
        str or None: Nombre del fabricante si se encuentra, de lo contrario None.
    """
    # Normaliza la dirección MAC eliminando ':' y '-', y convirtiéndola a minúsculas
    mac = mac.replace(":", "").replace("-", "").lower()
    # Construye la URL de la API con la dirección MAC normalizada
    url = f"https://api.maclookup.app/v2/macs/{mac}/company/name"
    try:
        # Realiza una solicitud GET a la API
        response = requests.get(url)
    except requests.exceptions.RequestException as e:
        # Maneja posibles excepciones de conexión
        print(f"Error de conexión: {e}")
        return None

    # Verifica si la respuesta HTTP fue exitosa (código de estado 200)
    if response.status_code != 200:
        print(f"Error: {response.status_code} - {response.text}")
        return None

    # La respuesta es texto plano con el nombre del fabricante
    return response.text.strip()  # Retorna el nombre del fabricante limpio
```

Figura 2. Muestra de la Función "get_vendor_from_mac"

2) display_vendor_info (vendor_name, mac): El propósito de la función es mostrar en consola el Fabricante de la MAC consultada, pero si no lo encuentra, mostrara el mensaje "Not found", donde tomara como parámetros el nombre del fabricante y la MAC.

```python
def display_vendor_info(vendor_name, mac):
    """
    Muestra en la consola la dirección MAC y el nombre del fabricante.
    Args:
        vendor_name (str): Nombre del fabricante.
        mac (str): Dirección MAC.
    """
    print(f"MAC address : {mac}")
    if vendor_name and vendor_name.lower() != "not found":
        print(f"Fabricante : {vendor_name}")
    else:
        print("Fabricante : Not found")
```

Figura 3. Muestra de la Función "display_vendor_info"

3) get_arp_table (): El propósito de la función es obtener la tabla ARP del sistema, que contiene las direcciones MAC de los dispositivos conectados a la red local

```python
def get_arp_table():
    """
    Obtiene la tabla ARP del sistema ejecutando el comando 'arp -a'.
    Returns:
        list: Lista de tuplas con (MAC, Tipo) para cada entrada ARP.
    """
    try:
        # Ejecuta el comando 'arp -a' y decodifica la salida usando 'cp1252' (común en Windows)
        arp_output = subprocess.check_output("arp -a", shell=True).decode('cp1252')
    except subprocess.CalledProcessError as e:
        # Maneja errores al ejecutar el comando 'arp'
        print(f"Error al ejecutar arp: {e}")
        return []
    except UnicodeDecodeError as e:
        # Maneja errores de decodificación de la salida del comando
        print(f"Error de decodificación: {e}")
        return []

    # Expresión regular para extraer MAC y Tipo de cada línea de la salida del comando 'arp'
    arp_entries = re.findall(r'(\d+\.\d+\.\d+\.\d+)\s+([\da-fA-F:-]+)\s+([\w\s]+)', arp_output)
    return arp_entries  # Retorna la lista de entradas ARP extraídas
```

Figura 4. Muestra de la Función "get_arp_table"

4) process_mac (mac): El propósito de la función es la de consultar el fabricante de una dirección MAC en especifico y mostrar el resultado de este por consola junto con su tiempo de respuesta, donde llamamos 2 funciones anteriormente mencionadas para obtener la información del Fabricante de la MAC consultada

```
def process_mac(mac):
    """
    Procesa una dirección MAC específica: obtiene el fabricante y muestra la información junto con el tiempo de respuesta.
    Args:
        mac (str): Dirección MAC a procesar.
    """
    start_time = time.time()  # Registra el tiempo de inicio
    vendor_name = get_vendor_from_mac(mac)  # Obtiene el fabricante de la MAC
    response_time = time.time() - start_time  # Calcula el tiempo de respuesta
    display_vendor_info(vendor_name, mac)  # Muestra la información del fabricante
    print(f"Tiempo de respuesta: {int(response_time * 1000)}ms")  # Muestra el tiempo de respuesta en ms
```

Figura 5. Muestra de la Función "process_mac"

5) process_arp (): El propósito de la función es la de obtener todas las entradas de la tabla ARP local, consultar sus fabricantes de las MACs y mostrar su información correspondiente.

```
def process_arp():
    """
    Procesa todas las entradas de la tabla ARP del sistema, obteniendo y mostrando el fabricante para cada dirección MAC.
    """
    print("MAC/Vendor:")  # Imprime el encabezado de la salida
    arp_entries = get_arp_table()  # Obtiene la tabla ARP
    if not arp_entries:
        # Informa si no se encontraron entradas o hubo un error al obtenerlas
        print("No se encontraron entradas en la tabla ARP o hubo un error al obtenerlas.")
        return
    for entry in arp_entries:
        _, mac, _ = entry  # Desempaqueta la entrada ARP
        vendor_name = get_vendor_from_mac(mac)  # Obtiene el fabricante de la MAC
        vendor_name = vendor_name if vendor_name else 'Unknown'  # Asigna 'Unknown' si no se encuentra el fabricante
        print(f"{mac} / {vendor_name}")  # Imprime la información en formato "IP / MAC / Vendor"
```

Figura 6. Muestra de la Función "process_arp"

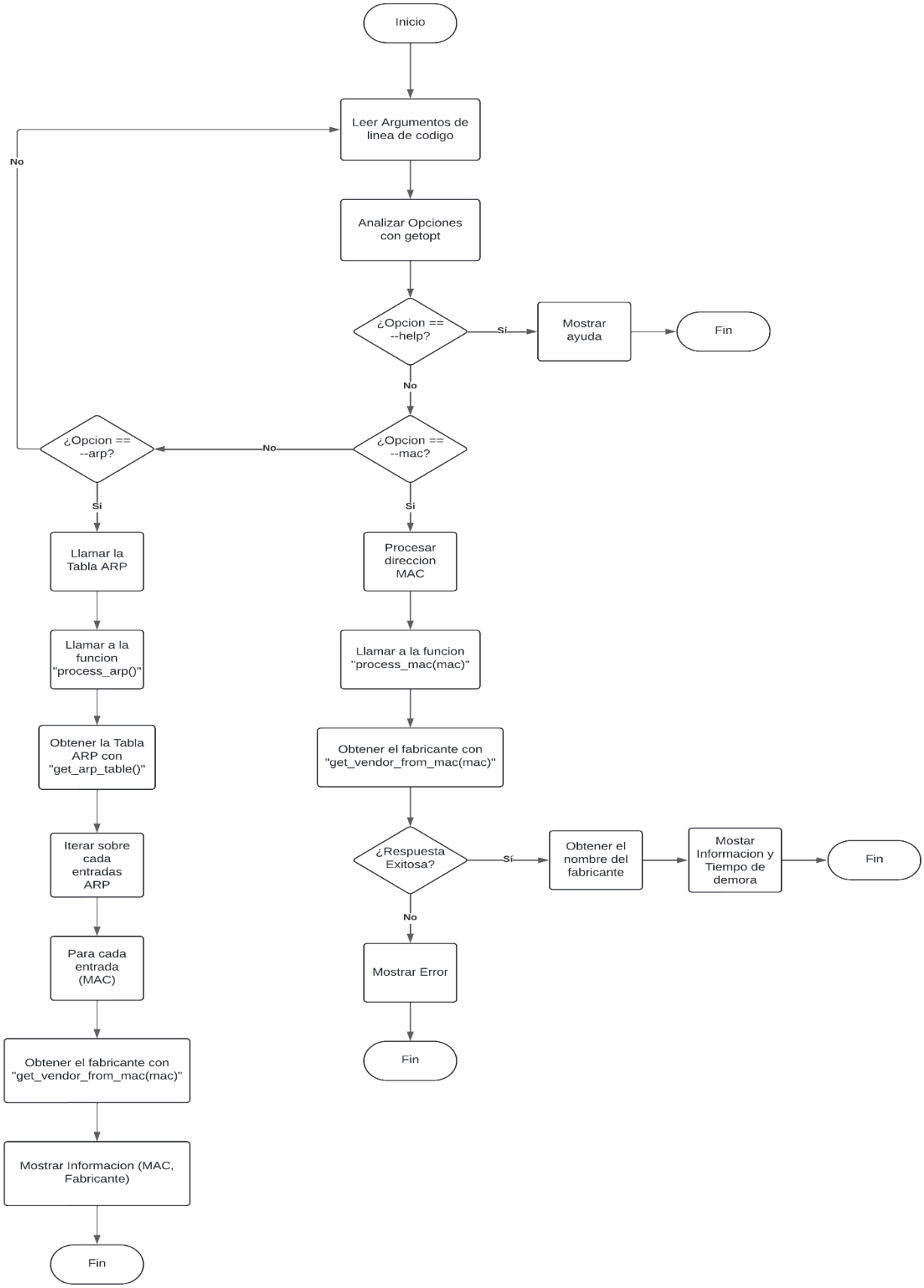
6) main(argv): El propósito de esta funcion es la de manejar los argumentos de la línea de comando y poder ejecutar las acciones correspondientes según la opción ingresada por el usuario, donde aquí entra en juego las funciones anteriormente mencionadas donde luego al final, será lo principal para que el programa comience a ejecutar y realizar las acciones solicitadas.

```python
def main(argv):
    """
    Función principal que maneja los argumentos de línea de comandos y dirige la ejecución del programa.
    Args:
        argv (list): Lista de argumentos de línea de comandos.
    """
    try:
        # Define las opciones cortas y largas que el script acepta
        opts, args = getopt.getopt(argv, "hm:a", ["mac=", "arp", "help"])
    except getopt.GetoptError:
        # Informa sobre el uso correcto si hay un error en los argumentos
        print("Uso: OUILookup.py --mac <mac> | --arp | [--help]")
        sys.exit(2)

    for opt, arg in opts:
        if opt in ("-h", "--help"):
            # Muestra la ayuda de uso
            print("Uso: OUILookup.py --mac <mac> | --arp | [--help]")
            sys.exit()
        elif opt in ("-m", "--mac"):
            # Procesa una dirección MAC específica
            process_mac(arg)
        elif opt in ("--arp"):
            # Procesa todas las entradas de la tabla ARP
            process_arp()
        else:
            # Informa si se proporciona una opción no válida
            print("Opción no válida")
            sys.exit(2)
```

Figura 7. Muestra de la funcion “main”

4. Diagrama de Flujos del Programa



5. MACs Aleatorios para cada dispositivo electrónico.

Ahora, la explicación del porque los dispositivos electrónicos contienen MAC aleatorias, básicamente es una característica común para cada dispositivo electrónico como los que usamos a diario (computadoras, celulares, etc.), ya que estas se generan de manera temporal para mejorar la privacidad a cualquier usuario para conectarse a redes inalámbricas, evitar como que se rastree el dispositivo con la MAC original, pero ojo, solamente algunas MAC son aleatorias en cada dispositivo, ya que algunos puntos de acceso requieren MACs permanentes para la autentificación.

6. Conclusión

El desarrollo de la herramienta **OUILookup** en Python demuestra cómo una aplicación sencilla en línea de comandos puede facilitar el acceso a información relevante sobre los dispositivos conectados a una red, a partir de sus direcciones MAC. La implementación del programa, utilizando una API pública para la consulta de fabricantes y comandos nativos del sistema para obtener la tabla ARP, ofrece una solución eficiente y de fácil uso para administradores de redes y usuarios que deseen identificar dispositivos en su entorno local.

Este trabajo demuestra el potencial de las herramientas de código abierto para mejorar la administración de redes y destaca la importancia de recursos públicos como APIs y bases de datos abiertas. En resumen, **OUILookup** ofrece una solución práctica, fácil de usar y adaptable, que contribuye a la gestión y control de los dispositivos conectados a redes locales.

7. Referencias

[1] Python Software Foundation. *The Python Standard Library* https://docs.python.org/3/library/
[2] MACLookup API https://maclookup.app/

[3] ARP command documentation https://man7.org/linux/man-pages/man8/arp.8.html.

[4] IEEE Standards Association. *Randomized and Privacy-Preserving MAC Addresses* https://standards.ieee.org/

[5] *Wi-Fi Alliance Technical Document*, 2021 https://www.wi-fi.org

[6] "MAC randomization on Android devices." *Google Help*, 2022 https://support.google.com