Herramienta OUILookup

Lucas Orellana Bruna, lucas.orellana@alumnos.uv.cl

1. Introducción

El direccionamiento MAC (Media Access Control) es un identificador único asignado a las interfaces de red, esencial para la comunicación dentro de redes locales. Cada fabricante de hardware tiene un rango de direcciones MAC asociado, lo que permite identificar la empresa fabricante de un dispositivo a partir de su dirección MAC. El objetivo de este trabajo fue desarrollar una herramienta de línea de comandos, llamada OUILookup, que permita consultar el fabricante de una tarjeta de red dado su dirección MAC, utilizando una API pública.

Esta herramienta es útil en diferentes contextos, como la administración de redes o auditorías de seguridad, donde se requiere conocer la procedencia de los dispositivos conectados a una red. El propósito principal fue desarrollar una solución que permita realizar consultas rápidas y eficientes, ya sea para una MAC específica o para los dispositivos presentes en la tabla ARP local.

Las conclusiones principales de este trabajo incluyen la exitosa integración de la API pública maclookup.app y la gestión de los parámetros de línea de comandos. También se logró manejar la decodificación adecuada en diferentes sistemas operativos, resolviendo problemas relacionados con la codificación de caracteres.

2. Descripción del problema y diseño de la solución

2.1.1. Descripción del problema

El problema a resolver consiste en diseñar una herramienta que permita identificar el fabricante de una tarjeta de red, dada su dirección MAC, a través de una consulta a una base de datos en línea. Además, se requiere que la herramienta sea capaz de listar los fabricantes de los dispositivos presentes en la red local a partir de la tabla ARP del sistema.

Los requisitos y especificaciones de la tarea son los siguientes:

1. La herramienta debe funcionar en sistemas operativos Windows y Linux.
2. Debe aceptar tres parámetros principales: `--mac` para consultar un fabricante mediante una MAC, `--arp` para consultar todos los dispositivos presentes en la tabla ARP y `--help` para mostrar las instrucciones de uso.
3. Utilizar una API pública que permita realizar consultas a la base de datos de fabricantes de direcciones MAC (OUIs).

4. El código debe seguir el paradigma de la programación funcional y manejar los parámetros con la biblioteca `getopt`.

2.1.2. Diseño de la solución

El diseño de la solución se basó en la creación de una arquitectura simple, donde la funcionalidad principal se concentra en tres partes clave:

1. **Interfaz de línea de comandos**: Se procesan los argumentos utilizando `getopt`, lo que permite aceptar parámetros como `--mac`, `--arp` y `--help`.
2. **Consulta a la API**: Para obtener el fabricante de una dirección MAC, se realiza una solicitud a la API REST pública de maclookup.app. Se implementó una función para manejar el tiempo de respuesta y extraer el nombre del fabricante.
3. **Consulta de la tabla ARP**: Se extrajeron las direcciones MAC de la tabla ARP local usando comandos del sistema (`arp -a` en Windows y `arp` en Linux) y se buscaron los fabricantes correspondientes.

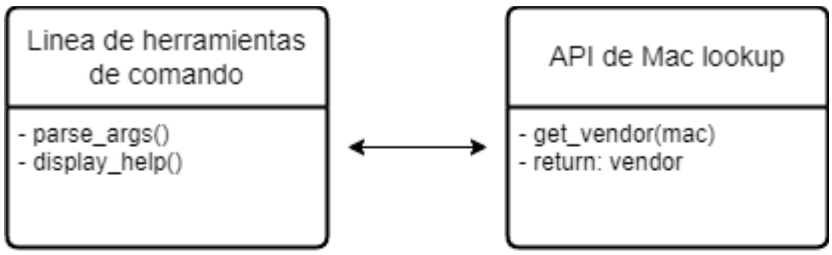
2.1.3. Arquitectura general

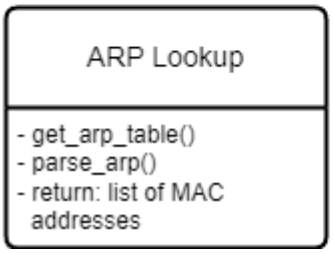
1. Módulo de entrada de parámetros: Se encarga de procesar los argumentos de línea de comandos.

2. Módulo de consulta de la API: Realiza las solicitudes HTTP y maneja las respuestas de la API.

3. Módulo de consulta de la tabla ARP: Ejecuta el comando del sistema para obtener la tabla ARP y realiza las búsquedas de fabricantes por cada MAC encontrada.

Diagrama de clases





3. Implementación y Pruebas

Estructura general

El código de la herramienta OUILookup fue diseñado para funcionar en una línea de comandos, y se estructura en tres módulos principales: el procesamiento de argumentos de entrada, la consulta de fabricantes a través de una API, y la extracción de la tabla ARP local. Todo esto se implementó en Python usando bibliotecas estándar como getopt, subprocess y requests.

A continuación, se explican las partes más importantes del código y cómo se abordaron los desafíos encontrados en la implementación.

Procesamiento de argumentos de línea de comandos

Para procesar los parámetros de entrada, se utilizó la biblioteca getopt, que permite manejar los diferentes modos de operación del script. El propósito era permitir al usuario realizar tres tipos de operaciones:

1. Consultar un fabricante a partir de una dirección MAC con el argumento --mac.
2. Obtener los fabricantes de los dispositivos presentes en la tabla ARP con --arp.
3. Mostrar un mensaje de ayuda con --help.

Implementación

Función main(argv, parse_args)

La función main(argv) es la que se encarga de gestionar los argumentos pasados por línea de comandos y reemplaza lo que en otros contextos se llamaría parse_args(). Esta función utiliza getopt.getopt() para analizar las opciones y los parámetros proporcionados por el usuario. Según la opción seleccionada, llama a las funciones correspondientes.

```python
# Función principal para procesar los argumentos
def main(argv):
    mac = None
    show_arp = False

    try:
        opts, args = getopt.getopt(argv, "", ["mac=", "arp", "help"])
    except getopt.GetoptError:
        print_help()
        sys.exit(2)

    for opt, arg in opts:
        if opt == "--mac":
            mac = arg
        elif opt == "--arp":
            show_arp = True
        elif opt == "--help":
            print_help()
            sys.exit()

    if mac:
        mac_cleaned = mac.replace("-", ":").replace(".", ":")
        vendor, response_time = get_vendor_from_mac(mac_cleaned)
        print(f"MAC address : {mac_cleaned}")
        print(f"Fabricante  : {vendor}")
        print(f"Tiempo de respuesta: {response_time}ms")

    elif show_arp:
        arp_vendors = get_arp_table_vendors()
        print("IP/MAC/Vendor:")
        for entry in arp_vendors:
            print(entry)

    else:
        print_help()
```

Desafío: Compatibilidad multiplataforma

Un desafío clave fue hacer que la herramienta funcionara tanto en Windows como en Linux. En particular, el comando arp funciona de manera diferente en estos sistemas operativos, por lo que era necesario gestionar ambas plataformas. Para resolver este problema, se añadió un chequeo del sistema operativo con sys.platform, ajustando los comandos y la decodificación de salida según el sistema.

```python
try:
    if sys.platform == "win32":
        arp_result = subprocess.check_output("arp -a", shell=True).decode()
    else:
        arp_result = subprocess.check_output("arp", shell=True).decode()
```

Función “get_vendor_from_mac()”

Para obtener el fabricante de una dirección MAC, se usó la API de maclookup.app. La función principal, get_vendor_from_mac(mac), se encarga de realizar una solicitud HTTP usando la biblioteca requests y procesar la respuesta para extraer el nombre del fabricante.

La función hace una solicitud HTTP `GET` a la API pasando la dirección MAC como parámetro. Si la respuesta es exitosa (código 200), se procesa el resultado en formato JSON para obtener el nombre del fabricante.

```python
# Función para obtener el fabricante de la MAC desde la API
def get_vendor_from_mac(mac_address):
    url = f'https://api.maclookup.app/v2/macs/{mac_address}'
    start_time = time.time()

    try:
        response = requests.get(url)
        response_time = int((time.time() - start_time) * 1000)  # Tiempo de respuesta en milisegundos

        if response.status_code == 200:
            data = response.json()
            if 'company' in data:
                return data['company'], response_time
            else:
                return "Not found", response_time
        else:
            return "Error: Unable to fetch data", response_time

    except requests.RequestException as e:
        return f"Error: {str(e)}", 0
```

Desafío: Manejo de errores y tiempos de respuesta

Un reto importante fue manejar las posibles fallas en la solicitud a la API (como la falta de conectividad o errores de la API). Se implementó una lógica de control para manejar códigos de estado distintos de 200, proporcionando un resultado adecuado en esos casos. Además, para cada consulta se registra el tiempo de respuesta de la API, que se muestra al usuario junto con el resultado de la búsqueda.

Función “get_arp_table_vendors()”

La función get_arp_table_vendors() ejecuta el comando del sistema (arp -a o arp), extrae las direcciones MAC de la salida usando expresiones regulares y llama a la función de consulta de la API para cada dirección.

```python
# Función para extraer las MACs de la tabla ARP y obtener los fabricantes
def get_arp_table_vendors():
    # Obtener la tabla ARP en sistemas Linux/Windows
    try:
        if sys.platform == "win32":
            arp_result = subprocess.check_output("arp -a", shell=True).decode()
        else:
            arp_result = subprocess.check_output("arp", shell=True).decode()

        arp_entries = re.findall(r'([0-9a-fA-F:.-]{17})', arp_result)

        results = []
        for mac in arp_entries:
            mac_cleaned = mac.replace("-", ":").replace(".", ":")
            vendor, _ = get_vendor_from_mac(mac_cleaned)
            results.append(f'{mac_cleaned} / {vendor}')

        return results
    except subprocess.CalledProcessError as e:
        return f"Error: Could not retrieve ARP table: {str(e)}"
```

Desafío: Codificación y decodificación de la salida

Uno de los principales desafíos fue manejar la codificación de la salida del comando arp -a en Windows, que genera texto en una codificación distinta a UTF-8 (como ISO-8859-1). Esto generó un error UnicodeDecodeError al intentar decodificar la salida en sistemas Windows. Para solucionarlo, se especificó la decodificación usando ISO-8859-1 al procesar la salida en Windows.

Función “Print_help()”

Esta función muestra un mensaje de ayuda para que el usuario observe que comando usar y con que finalidad

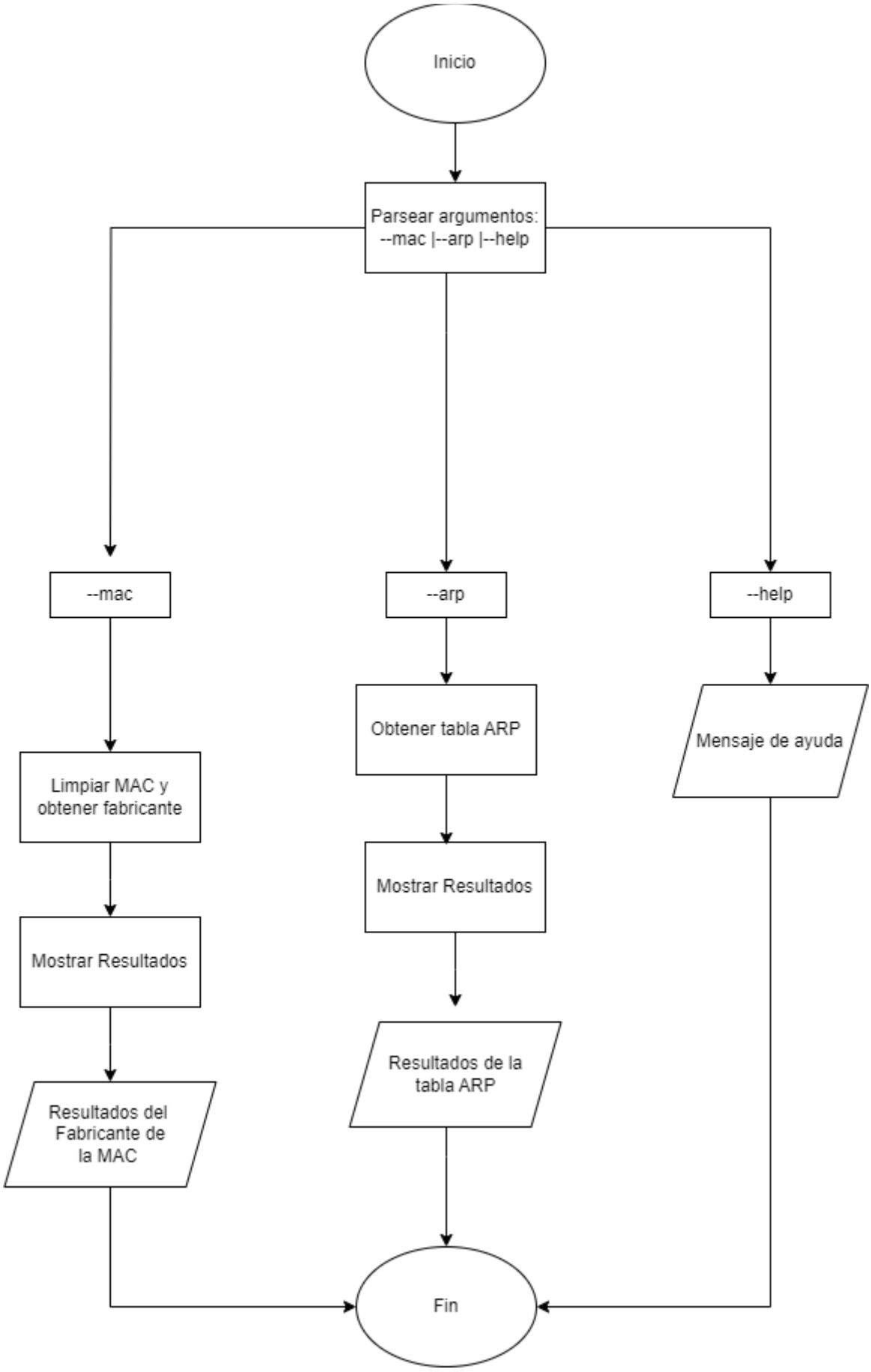
```python
# Función para mostrar la ayuda
def print_help():
    help_message = """
    Use: python OUILookup.py --mac <mac> | --arp | [--help]

    --mac: MAC a consultar. P.e. aa:bb:cc:00:00:00.
    --arp: Muestra los fabricantes de los hosts disponibles en la tabla ARP.
    --help: Muestra este mensaje y termina.
    """
    print(help_message)
```

Pruebas

Para verificar el correcto funcionamiento de la herramienta, se realizaron pruebas con diferentes direcciones MAC y en distintos sistemas operativos. A continuación, algunos ejemplos de pruebas realizadas:

```
PS C:\Users\lucas\Desktop\gaa> python OUILookup.py --mac 98:06:3c:92:ff:c5
MAC address : 98:06:3c:92:ff:c5
Fabricante  : Samsung Electronics Co.,Ltd
Tiempo de respuesta: 632ms
PS C:\Users\lucas\Desktop\gaa> python OUILookup.py --mac 9c:a5:13
MAC address : 9c:a5:13
Fabricante  : Samsung Electronics Co.,Ltd
Tiempo de respuesta: 426ms
PS C:\Users\lucas\Desktop\gaa> python OUILookup.py --mac 48-E7-DA
MAC address : 48:E7:DA
Fabricante  : AzureWave Technology Inc.
Tiempo de respuesta: 1527ms
```

Diagrama de Flujo


Inicio
Parsear argumentos:
--mac |--arp |--help
--mac
--arp
--help
Limpiar MAC y
obtener fabricante
Obtener tabla ARP
Mensaje de ayuda
Mostrar Resultados
Mostrar Resultados
Resultados del
Fabricante de
la MAC
Resultados de la
tabla ARP
Fin

Descripción del flujo:

1. Inicio: El programa comienza su ejecución.
2. Parsear argumentos: Se analizan los argumentos de la línea de comandos.
3. Condición --mac:

   Si se proporciona una dirección MAC, el programa limpia la dirección y obtiene el fabricante.

   Luego, muestra los resultados.

4. Condición --arp:

   Si se solicita la opción --arp, el programa obtiene la tabla ARP y muestra los fabricantes.

5. Condicion –help:

   Muestra un mensaje de ayuda al usuario y información de entradas

6. Fin: Termina la ejecución del programa.

4. Discusión y conclusiones

Resumen de resultados

La implementación de OUILookup logró cumplir con todos los requisitos establecidos. La herramienta permite consultar de manera eficiente el fabricante de una MAC individual, así como listar los fabricantes de los dispositivos presentes en la tabla ARP local. La integración de la API REST pública fue exitosa y permitió obtener resultados precisos y rápidos. Además, se manejó correctamente la decodificación de la salida del comando arp -a en sistemas Windows, resolviendo problemas de codificación no UTF-8.

Reflexión sobre el trabajo

Durante el desarrollo de la herramienta, se aprendió a manejar adecuadamente las bibliotecas estándar de Python para la gestión de parámetros de línea de comandos (getopt), la ejecución de comandos del sistema (subprocess) y la realización de solicitudes HTTP (requests). Además, se profundizó en la comprensión del funcionamiento de las tablas ARP y la estructura de las direcciones MAC.

Uno de los mayores desafíos fue la gestión de la salida del comando arp -a en sistemas Windows debido a diferencias en la codificación de caracteres, lo cual fue resuelto mediante la utilización de la codificación ISO-8859-1 en lugar de UTF-8.

Posibles mejoras

Soporte para más APIs: Actualmente, la herramienta depende de una única API pública. Una posible mejora sería añadir soporte para múltiples APIs, aumentando la disponibilidad y confiabilidad del servicio.

Optimización de la búsqueda ARP: El proceso de consulta de los fabricantes en la tabla ARP podría optimizarse mediante técnicas de caché, evitando consultas repetidas a la API.

Interfaz gráfica: Aunque el objetivo principal era crear una herramienta de línea de comandos, una interfaz gráfica simple podría mejorar la accesibilidad y el uso de la herramienta por parte de usuarios no técnicos.

5. Referencias

maclookup.app API Documentation, Subprocess documentation in Python, IEEE Standards for MAC Addresses, ARP command for Windows and Linux