

## Tarea 2.

Daniel Belozo Ossandón, [daniel.belozo@alumnos.uv.cl](mailto:daniel.belozo@alumnos.uv.cl)

Francisco Villagran Madrid, [francisco.villagran@alumnos.uv.cl](mailto:francisco.villagran@alumnos.uv.cl)

Mariajosé Baxmann Román, [mariajose.baxmann@alumnos.uv.cl](mailto:mariajose.baxmann@alumnos.uv.cl)

### 1. Introducción

En el siguiente trabajo nuestro objetivo será la implementación de un programa denominado 'OUILookup' programado en Python, el cual tendrá como objetivo la identificación de los fabricantes de las tarjetas de red de los dispositivos conectados a nuestra red (o cualquier dirección MAC a consultar) mediante la utilización de una API REST desde <https://maclookup.app>. Además, nos permitirá obtener, mediante un parámetro específico, el mapeo de nuestra red, las IPs asociadas, sus direcciones MAC junto a los nombres de los fabricantes de cada tarjeta de red.

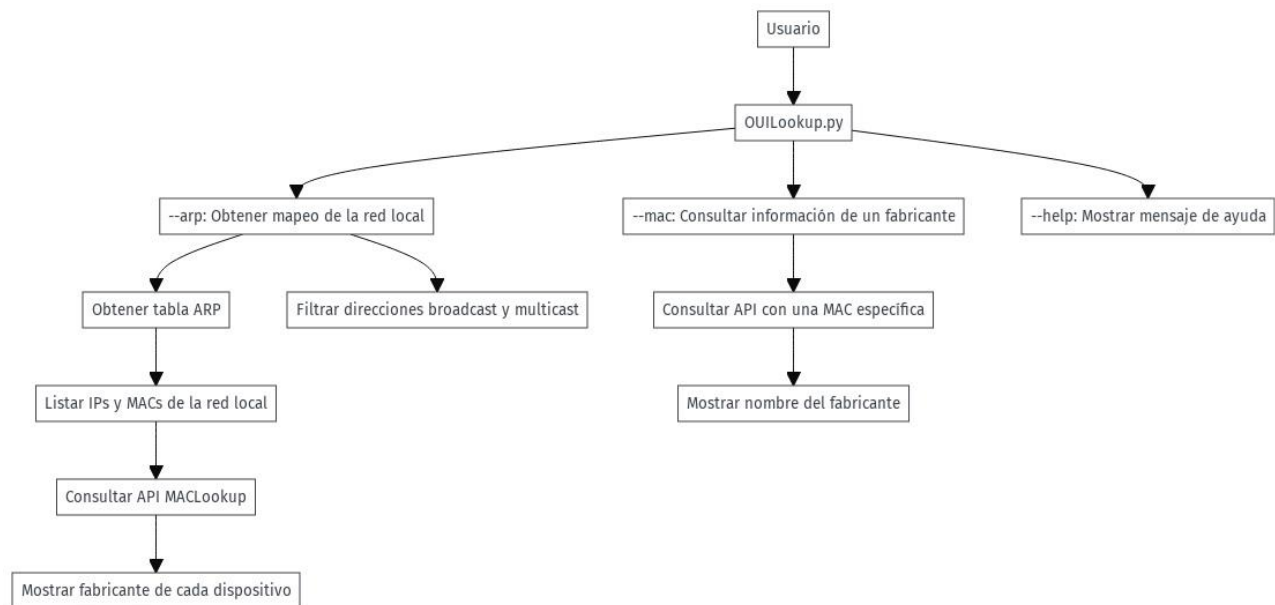
### 2. Descripción del problema y diseño de la solución

Nuestro problema radica en la necesidad de identificar los fabricantes de las tarjetas de red a través de las direcciones MAC. Para resolver esto, utilizaremos una API REST específica que nos permitirá obtener información sobre los fabricantes de cada dispositivo conectado a nuestra red, así como de cualquier dirección MAC que consultemos.

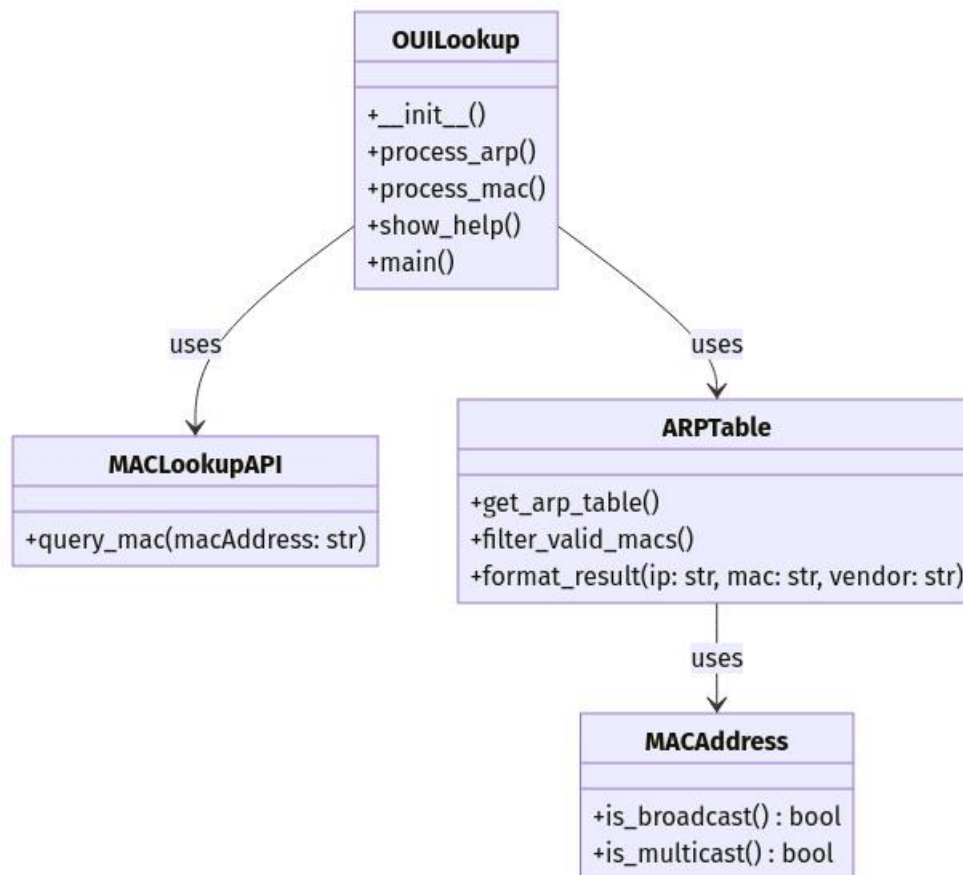
Como solución, implementaremos un código en Python que permita la utilización de tres parámetros denominados --arp, --mac y --help a fin de obtener la información relacionada con la red y los dispositivos conectados. Estos argumentos tendrán las siguientes funcionalidades:

1. --arp: Este parámetro permitirá obtener un mapeo de las direcciones IP y las direcciones MAC de los dispositivos en la red local, utilizando la tabla ARP. El programa listará las IPs de los dispositivos conectados junto con sus correspondientes direcciones MAC, y consultará a la API para identificar el fabricante del hardware asociado a cada dirección MAC.
2. --mac: Permitirá al usuario proporcionar una dirección MAC específica, y el programa consultará la API a fin de obtener información sobre el fabricante del dispositivo al que pertenece esa dirección MAC. El programa devolverá detalles como el nombre del fabricante y el tiempo de respuesta de la consulta.
3. --help: Este argumento mostrará un mensaje de ayuda que explique cómo utilizar el programa y los diferentes argumentos disponibles. Proporcionará una descripción clara de cada opción para que los usuarios puedan interactuar correctamente con el código.

## Modelo de arquitectura

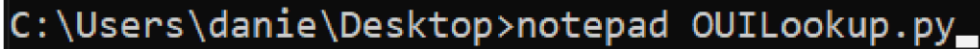


## Diagrama de clases



### 3. Implementación

Para realizar nuestra solución necesitamos dirigirnos a la consola de comandos de Windows (cmd) y crearemos un archivo.py con el nombre de OUILookup.py (En el caso de tener instalado python3, proceder a instalarlo antes de continuar).



```
C:\Users\danie\Desktop>notepad OUILookup.py
```

Figura 1. Creación del archivo.py en directorio Desktop.

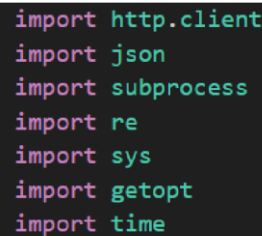
Luego procedemos a abrir nuestro archivo con un editor de texto de su conveniencia.



```
C:\Users\danie\Desktop>code OUILookup.py
```

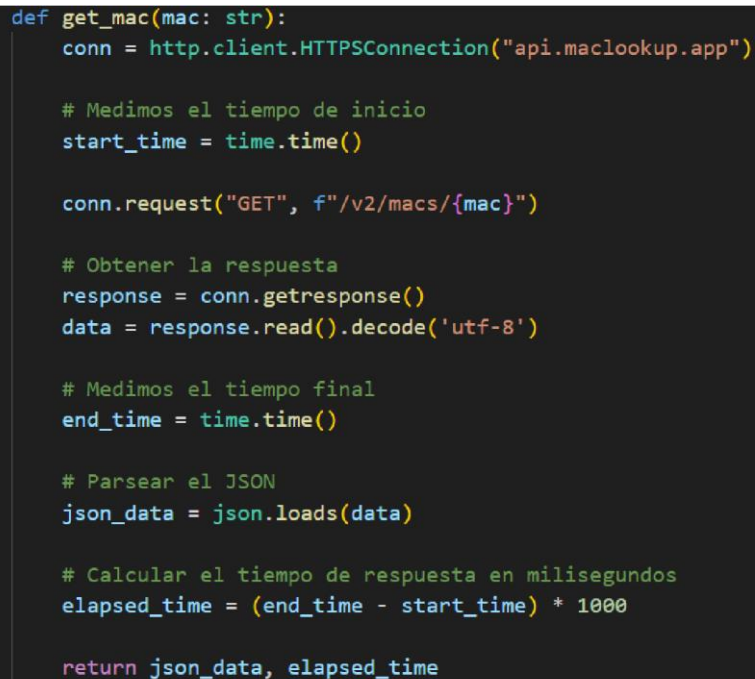
Figura 2. Apertura del archivo mediante software VSCode.

Una vez dentro procedemos a crear nuestro código en Python.



```
import http.client
import json
import subprocess
import re
import sys
import getopt
import time
```

Figura 3. Procedemos a importar las librerías necesarias para nuestro código.



```
def get_mac(mac: str):
    conn = http.client.HTTPSConnection("api.maclookup.app")

    # Medimos el tiempo de inicio
    start_time = time.time()

    conn.request("GET", f"/v2/mac/{mac}")

    # Obtener la respuesta
    response = conn.getresponse()
    data = response.read().decode('utf-8')

    # Medimos el tiempo final
    end_time = time.time()

    # Parsear el JSON
    json_data = json.loads(data)

    # Calcular el tiempo de respuesta en milisegundos
    elapsed_time = (end_time - start_time) * 1000

    return json_data, elapsed_time
```

Figura 4. Definimos nuestra función que nos permitirá obtener los fabricantes de las tarjetas de red, a través de la API pública, de las direcciones MAC a consultar.

```
def get_arp_table():
    tabla_arp = subprocess.run(["arp", "-a"], stdout=subprocess.PIPE)
    macs = re.findall(r'\w\w:\w\w:\w\w:\w\w:\w\w:\w\w', str(tabla_arp.stdout).replace("-", ":"))
    ips = re.findall(r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}', str(tabla_arp.stdout))
    return list(zip(ips, macs))
```

Figura 5. Definimos nuestra función que permite la obtención de la tabla ARP de tu red local. La tabla ARP contiene un mapeo de direcciones IP a direcciones MAC, permitiendo identificar los dispositivos en la red.

```
def is_broadcast_or_multicast(mac):
    return mac.startswith("ff:ff:ff") or mac.startswith("01:00:5e")
```

Figura 6. Creamos una función que nos verifique si una dirección MAC es de broadcast o multicast. Ya que estas no se asocian a dispositivos específicos y no deberían ser consultadas en la API.

```
def process_result(mac_vendor, ip_mac):
    ip, mac = ip_mac
    vendor_data, _ = mac_vendor(mac)
    vendor = vendor_data['company']
    return f"{ip} / {mac} / {vendor if vendor else 'Not Found'}"
```

Figura 7. Diseñamos una función que tome el resultado de la búsqueda de la dirección MAC y la formatee a fin de mostrar la IP, la MAC y el fabricante.

```
def process_arp_table():
    arp_table = get_arp_table()
    mac_vendor = get_mac
    valid_arp_entries = filter(lambda x: not is_broadcast_or_multicast(x[1]), arp_table)
    return list(map(lambda x: process_result(mac_vendor, x), valid_arp_entries))
```

Figura 8. Esta función procesa la **tabla ARP** obtenida por `get_arp_table()`, filtrando las direcciones MAC de broadcast y multicast, y luego obteniendo el nombre del fabricante de las MAC válidas.

```
def search_mac(mac):
    response, elapsed_time = get_mac(mac) # Extraemos tanto el JSON como el tiempo de respuesta
    if response['found']:
        return f"MAC Address : {mac}\nFabricante : {response['company']}\nTiempo de respuesta: {elapsed_time:.2f} ms"
    else:
        return f"MAC Address : {mac}\nFabricante : Not Found\nTiempo de respuesta: {elapsed_time:.2f} ms"
```

Figura 9. Esta función permite buscar información específica sobre una **dirección MAC** dada.

```
def print_usage():
    print("Uso: python OUILookup.py --mac <mac> | --arp")
    print("--mac: MAC a consultar. Ej: aa:bb:cc:00:00:00.")
    print("--arp: Muestra los fabricantes de los hosts disponibles en la tabla ARP.")
    print("--help: Muestra este mensaje y termina.")
```

Figura 10. Ya que no utilizamos librerías que nos den un parámetro `-help` de forma automática, creamos el mensaje de ayuda de forma manual.

```
if __name__ == "__main__":
    # Configuración de getopt
    try:
        opts, args = getopt.getopt(sys.argv[1:], "hm:a", ["help", "mac=", "arp"])
    except getopt.GetoptError as err:
        print(err)
        print_usage()
        sys.exit(2)

    mac = None
    arp_flag = False

    # Procesamiento de las opciones
    for opt, arg in opts:
        if opt in ("-h", "--help"):
            print_usage()
            sys.exit()
        elif opt in ("-m", "--mac"):
            mac = arg
        elif opt in ("-a", "--arp"):
            arp_flag = True
```

Figura 11. Creamos nuestro bloque principal del programa el cual nos procesará los parámetros de la línea de comandos y llamará a las funciones adecuadas según los argumentos ingresados.

```
if arp_flag:
    results = process_arp_table()
    print("IP/MAC/Vendor:")
    print("\n".join(results))
elif mac:
    # Si se ingresa una MAC específica, búscala directamente
    print(search_mac(mac))
else:
    # Si no se ingresan opciones, mostrar el uso del programa
    print_usage()
```

Figura 12. Segunda parte de nuestro 'main'.

## 4. Pruebas

Volvemos a nuestra consola de Windows y ejecutamos nuestro código con los parámetros definidos en nuestro código, y ya que no utilizamos librerías ajenas a las predefinidas en Python3 no existe la necesidad de instalar nuevos paquetes.

### 4.1. Parámetro “—arp”:

```
C:\Users\danie\Desktop>py OUILookup.py --arp_
```

Figura 13. Ejecutamos nuestro programa con el parámetro “—arp”.

```
IP/MAC/Vendor:
239.255.255.250 / 40:0d:10:cf:d9:20 / ARRIS Group, Inc.
192.168.0.10 / 50:41:1c:0e:7f:8e / AMPAK Technology, Inc.
192.168.0.1 / 00:00:ca:01:02:03 / ARRIS Group, Inc.
```

Figura 14. Nos entrega el formato definido en nuestro código con las IPs, MAC y fabricantes, de nuestra red local.

### 4.2. Parámetro “—mac [MAC]”:

```
C:\Users\danie\Desktop>py 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: 636.66 ms
```

Figura 15. Utilizamos parámetro “—mac” y le entregamos el argumento 98:06:3c:92:ff:c5 a fin de conocer el fabricante de su tarjeta de red.

```
C:\Users\danie\Desktop>py OUILookup.py --mac 9c:a5:13
MAC Address : 9c:a5:13
Fabricante : Samsung Electronics Co.,Ltd
Tiempo de respuesta: 2581.81 ms
```

Figura 16. Utilizamos parámetro “—mac” y le entregamos el argumento 9c:a5:13 a fin de conocer el fabricante de su tarjeta de red.

```
C:\Users\danie\Desktop>py OUILookup.py --mac 48-E7-DA
MAC Address : 48-E7-DA
Fabricante : AzureWave Technology Inc.
Tiempo de respuesta: 3016.79 ms
```

Figura 17. Utilizamos parámetro “—mac” y le entregamos el argumento 48-E7-DA a fin de conocer el fabricante de su tarjeta de red.

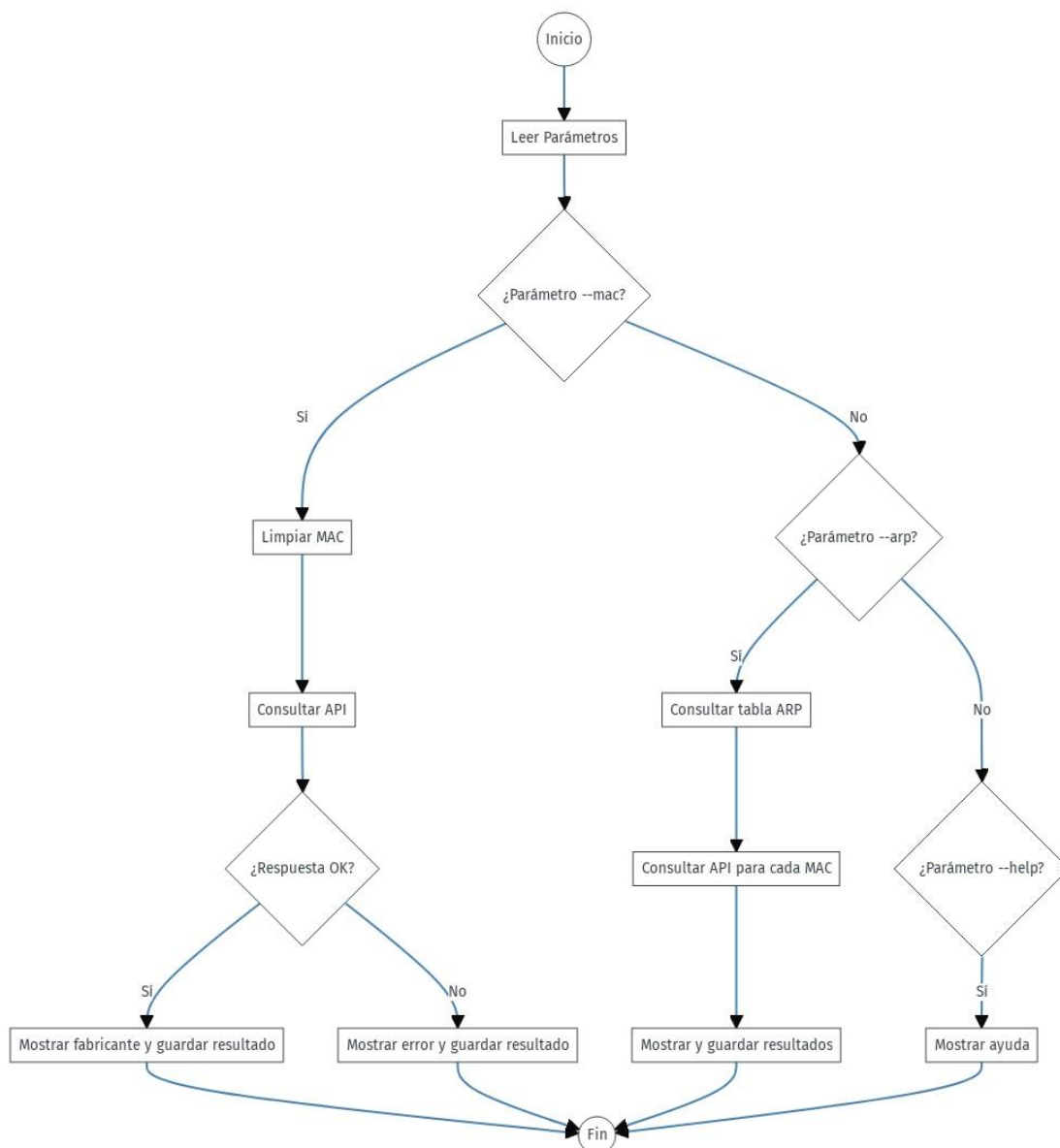


### 4.3. Parámetro “—help”:

```
C:\Users\danie\Desktop>py OUILookup.py --help
Uso: python OUILookup.py --mac <mac> | --arp
--mac: MAC a consultar. Ej: aa:bb:cc:00:00:00.
--arp: Muestra los fabricantes de los hosts disponibles en la tabla ARP.
--help: Muestra este mensaje y termina.
```

Figura 18. Parámetro “—help” nos permite visualizar las series de parámetros que nos permite utilizar nuestro programa mediante la consola de comandos.

### 4.4. Diagrama de flujo:



#### 4.5. MACs Aleatorias:

Las direcciones MAC aleatorias son una técnica utilizada por dispositivos electrónicos para mejorar la privacidad y seguridad al interactuar con redes inalámbricas, como Wi-Fi o Bluetooth. Tradicionalmente, una dirección MAC (Media Access Control) es un identificador único de 48 bits asignado a una interfaz de red para la comunicación en la capa de enlace de datos. Sin embargo, esta dirección fija puede ser utilizada para rastrear dispositivos en diversas redes, lo que representa un riesgo para la privacidad.

Para mitigar este problema, los dispositivos modernos, como smartphones y laptops, han adoptado el uso de direcciones MAC aleatorias. Este enfoque permite que los dispositivos generen temporalmente una dirección MAC aleatoria cuando escanean o se conectan a redes, evitando así la exposición de la dirección MAC original y reduciendo la capacidad de terceros para rastrear el dispositivo a lo largo del tiempo.

#### 5. Discusión y conclusiones:

En este trabajo, se abordó la problemática de identificar los fabricantes de las tarjetas de red utilizando las direcciones MAC de los dispositivos. A partir de esta necesidad, desarrollamos una solución que, mediante el uso de una API REST, nos permite obtener información precisa sobre los fabricantes asociados a cualquier dirección MAC consultada. Implementamos un código en Python que, a través de opciones como `--mac` y `--arp`, facilita tanto la consulta individual de una MAC específica como la obtención del mapeo de los dispositivos conectados en la red local. Esta herramienta contribuye a una mejor gestión y monitoreo de la red, permitiendo una identificación rápida y efectiva de los dispositivos, lo cual es crucial para tareas de auditoría y seguridad en entornos de red.



## 6. Referencias:

1. A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, y Y. Weiss, "Mobile Malware Detection through Analysis of MAC Address Behavior", *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 28–39, 2012. doi: 10.1109/TIFS.2011.2168208.
2. H. K. Patil y R. Seshadri, "Privacy-Preserving MAC Address Randomization in Wireless Networks", *International Journal of Network Security*, vol. 22, no. 3, pp. 394–403, 2020. doi: 10.6633/IJNS.202003\_22(3).06.
3. J. Martin, S. Mayberry, C. Donahue y D. Kuipers, "Enabling MAC address randomization on Android devices," 2017 IEEE Conference on Communications and Network Security (CNS), Las Vegas, NV, USA, 2017, pp. 1-9. doi: 10.1109/CNS.2017.8228700.
4. MACLookup, "MAC Address Vendor Lookup API," Disponible en: <https://maclookup.app> (consultado el 9 de octubre de 2024).
5. D. J. Bernstein, "Random MAC Address: A Comprehensive Privacy Feature," Disponible en: [https://www.networkprivacytech.com/articles/random\\_mac\\_address](https://www.networkprivacytech.com/articles/random_mac_address), 2022.