**Project: Netwerk Scanner met Python**

Dit project richt zich op het ontwikkelen van een eenvoudig netwerk scanning-tool met Python. Het is gericht op netwerk- en systeembeheer en helpt om inzicht te krijgen in de apparaten die op een lokaal netwerk actief zijn. Dit is nuttig om ongewenste apparaten te detecteren of om netwerkconfiguraties te controleren.

**Wat doet dit project?**

1. Het scant een opgegeven netwerkbereik.
2. Het identificeert actieve apparaten.
3. Het rapporteert IP-adressen en MAC-adressen van de gevonden apparaten.

**Benodigdheden**

1. **Python 3.x** geïnstalleerd.
2. Python-bibliotheken: scapy en argparse.
   * Installeren via:

pip install scapy

**Code voor het Project**

Hier is de Python-code:

python

from scapy.all import ARP, Ether, srp

import argparse

def scan\_network(ip\_range):

"""

Functie om apparaten in een opgegeven IP-bereik te scannen.

:param ip\_range: Het IP-bereik dat gescand moet worden.

"""

# Maak een ARP-verzoek

arp\_request = ARP(pdst=ip\_range)

broadcast = Ether(dst="ff:ff:ff:ff:ff:ff")

arp\_request\_broadcast = broadcast / arp\_request

# Stuur het verzoek en ontvang antwoorden

print(f"Scanning netwerk: {ip\_range}...")

answered\_list = srp(arp\_request\_broadcast, timeout=2, verbose=False)[0]

# Verwerk de antwoorden

devices = []

for sent, received in answered\_list:

devices.append({'ip': received.psrc, 'mac': received.hwsrc})

return devices

def display\_results(devices):

"""

Functie om gescande resultaten weer te geven.

:param devices: Lijst van gedetecteerde apparaten.

"""

print("\nGevonden apparaten:")

print("IP-adres" + " " \* 20 + "MAC-adres")

print("-" \* 50)

for device in devices:

print(f"{device['ip']:<30} {device['mac']}")

if \_\_name\_\_ == "\_\_main\_\_":

# Argumentparser instellen

parser = argparse.ArgumentParser(description="Netwerk Scanner")

parser.add\_argument("ip\_range", help="Het IP-bereik dat gescand moet worden (bijv. 192.168.1.0/24).")

args = parser.parse\_args()

# Netwerk scannen

scanned\_devices = scan\_network(args.ip\_range)

# Resultaten weergeven

display\_results(scanned\_devices)

**Hoe werkt het?**

1. **ARP-verzoeken verzenden**:
   * De tool stuurt ARP-verzoeken (Address Resolution Protocol) om te bepalen welke apparaten actief zijn in een netwerkbereik.
2. **Netwerkbereik invoeren**:
   * Geef het netwerkbereik op, bijvoorbeeld 192.168.1.0/24, dat alle apparaten tussen 192.168.1.1 en 192.168.1.254 scant.
3. **Resultaten weergeven**:
   * De tool toont een lijst van actieve apparaten met hun IP- en MAC-adressen.

**Hoe gebruik je dit?**

1. **Run de code via de terminal**:
   * Sla de code op in een bestand, bijvoorbeeld netwerk\_scanner.py.
   * Voer het uit:

python netwerk\_scanner.py 192.168.1.0/24

1. **Output**:
   * Een lijst met actieve apparaten verschijnt:

markdown

Gevonden apparaten:

IP-adres MAC-adres

-------------------------------------

192.168.1.1 00:1A:2B:3C:4D:5E

192.168.1.10 11:22:33:44:55:66

**Waarom is dit project nuttig?**

* Het helpt netwerkbeheerders snel te controleren welke apparaten actief zijn.
* Het kan worden uitgebreid met functies zoals apparaatdetectie of meldingen bij onbekende apparaten.
* Het biedt hands-on ervaring met netwerken en Python-scripting

Uitgebreide uitleg:

**1. Importeren van Bibliotheken**

from scapy.all import ARP, Ether, srp

import argparse

* **scapy.all**: Een krachtige Python-bibliotheek die wordt gebruikt voor netwerkcommunicatie en pakketmanipulatie.
  + **ARP**: Verzendt een ARP-verzoek om het MAC-adres van een apparaat op te halen.
  + **Ether**: Maakt een Ethernet-frame om netwerkpakketten te verzenden.
  + **srp**: Verzendt en ontvangt pakketten op laag 2 (datalinklaag).
* **argparse**: Helpt bij het maken van een command-line interface (CLI) waarmee gebruikers een IP-bereik kunnen invoeren.

**2. Functie: scan\_network(ip\_range)**

def scan\_network(ip\_range):

"""

Functie om apparaten in een opgegeven IP-bereik te scannen.

:param ip\_range: Het IP-bereik dat gescand moet worden.

"""

# Maak een ARP-verzoek

arp\_request = ARP(pdst=ip\_range)

broadcast = Ether(dst="ff:ff:ff:ff:ff:ff")

arp\_request\_broadcast = broadcast / arp\_request

# Stuur het verzoek en ontvang antwoorden

answered\_list = srp(arp\_request\_broadcast, timeout=2, verbose=False)[0]

# Verwerk de antwoorden

devices = []

for sent, received in answered\_list:

devices.append({'ip': received.psrc, 'mac': received.hwsrc})

return devices

**Stap voor stap uitleg**:

1. **arp\_request = ARP(pdst=ip\_range)**:
   * Maakt een ARP-verzoek dat wordt verzonden naar een reeks IP-adressen (pdst).
   * ARP (Address Resolution Protocol) helpt het MAC-adres van apparaten te vinden.
2. **broadcast = Ether(dst="ff:ff:ff:ff:ff:ff")**:
   * Maakt een Ethernet-frame dat naar het MAC-adres ff:ff:ff:ff:ff:ff wordt verzonden (broadcast naar alle apparaten in het netwerk).
3. **arp\_request\_broadcast = broadcast / arp\_request**:
   * Combineert het Ethernet-frame met het ARP-verzoek in één pakket.
4. **srp**:
   * Verzendt de ARP-pakketten naar alle apparaten in het IP-bereik en wacht op antwoorden.
   * **timeout=2**: Wacht 2 seconden per verzoek.
   * **verbose=False**: Onderdrukt console-uitvoer.
   * **answered\_list**: Bevat een lijst van (verzonden, ontvangen) pakketparen.
5. **Resultaat verwerken**:
   * Voor elk ontvangen antwoord (received) wordt het IP-adres en MAC-adres opgeslagen in een lijst van apparaten (devices).

**3. Functie: display\_results(devices)**

def display\_results(devices):

"""

Functie om gescande resultaten weer te geven.

:param devices: Lijst van gedetecteerde apparaten.

"""

print("\nGevonden apparaten:")

print("IP-adres" + " " \* 20 + "MAC-adres")

print("-" \* 50)

for device in devices:

print(f"{device['ip']:<30} {device['mac']}")

**Stap voor stap uitleg**:

1. **Header printen**:
   * Print een overzichtelijke header met kolommen voor **IP-adres** en **MAC-adres**.
2. **Resultaten weergeven**:
   * Loopt door de lijst van gedetecteerde apparaten en print elk IP- en MAC-adres netjes uitgelijnd.

**4. Hoofdfunctie: if \_\_name\_\_ == "\_\_main\_\_"**

if \_\_name\_\_ == "\_\_main\_\_":

# Argumentparser instellen

parser = argparse.ArgumentParser(description="Netwerk Scanner")

parser.add\_argument("ip\_range", help="Het IP-bereik dat gescand moet worden (bijv. 192.168.1.0/24).")

args = parser.parse\_args()

# Netwerk scannen

scanned\_devices = scan\_network(args.ip\_range)

# Resultaten weergeven

display\_results(scanned\_devices)

**Stap voor stap uitleg**:

1. **Argumentparser instellen**:
   * **argparse.ArgumentParser** maakt een command-line interface waarmee gebruikers een IP-bereik kunnen invoeren.
   * **parser.add\_argument** definieert een verplicht argument voor het IP-bereik (bijv. 192.168.1.0/24).
2. **Netwerk scannen**:
   * Roept de functie scan\_network aan met het opgegeven IP-bereik.
   * De gescande apparaten worden opgeslagen in scanned\_devices.
3. **Resultaten weergeven**:
   * Roept de functie display\_results aan om de lijst met actieve apparaten te tonen.

**Hoe werkt de tool in de praktijk?**

1. Sla de code op in een bestand, bijvoorbeeld netwerk\_scanner.py.
2. Open een terminal en voer de tool uit met een IP-bereik:

python netwerk\_scanner.py 192.168.1.0/24

1. De tool verzendt ARP-verzoeken naar alle IP-adressen in het bereik en toont de gevonden apparaten:

markdown

Gevonden apparaten:

IP-adres MAC-adres

-------------------------------------

192.168.1.1 00:1A:2B:3C:4D:5E

192.168.1.10 11:22:33:44:55:66

**Uitbreidingsideeën**

* Voeg de apparaatnaam toe via een DNS-lookup of NetBIOS-query.
* Schrijf de resultaten naar een CSV-bestand.
* Stel een waarschuwing in voor ongewenste MAC-adressen.