Python Files

---------- **FileName:** 10NetworkC.py

Content:

```
common_ports = {
20: "FTP (Data Transfer)",
21: "FTP (Command Control)",
22: "SSH (Secure Shell)",
23: "Telnet",
25: "SMTP (Email Sending)",
53: "DNS (Domain Name System)",
67: "DHCP (Server)",
68: "DHCP (Client)",
69: "TFTP (Trivial File Transfer)",
80: "HTTP (Web Traffic)",
110: "POP3 (Email Retrieval)",
123: "NTP (Time Sync)",
143: "IMAP (Email Retrieval)",
161: "SNMP (Monitoring)",
443: "HTTPS (Secure Web Traffic)",
3389: "RDP (Remote Desktop Protocol)"
}


print("Common Ports and Services:\n")
for port, service in common_ports.items():
print(f"Port {port:<5} → {service}")
```

---------- **FileName:** 12NetworkC.py

Content:

```
Input = input("enter to know the port valid or no: ")

def Main(port):
return isinstance(port, int) and 0 <= port <= 65535

print(Main(Input))
```

---------- **FileName:** 13NetworkC.py

Content:

```
import socket

ip = input("Enter target IP address: ").strip()
open_ports = []
```

```
print(f"\nScanning {ip} for open ports 1–1000...\n")

for port in range(1, 1001):
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(0.5)
if s.connect_ex((ip, port)) == 0:
print(f"Port {port} is open")
open_ports.append(port)
s.close()

print("\nScan complete.")
if open_ports:
print("Open ports:", open_ports)
else:
print("No open ports found.")
```

FileName: 14NetworkC.py

Content:

```
import socket

target_ip = input("Enter the IP to check: ").strip()
port = int(input("Enter the port number: "))

tcp_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcp_sock.settimeout(0.5)
tcp_result = tcp_sock.connect_ex((target_ip, port))
tcp_sock.close()

udp_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
udp_sock.settimeout(0.5)
try:
udp_sock.sendto(b"Hello", (target_ip, port))
udp_sock.recvfrom(1024)
udp_result = True
except socket.timeout:
udp_result = False
except ConnectionRefusedError:
udp_result = True
finally:
udp_sock.close()

if tcp_result == 0:
print(f"Port {port} is open and uses TCP.")
elif udp_result:
print(f"Port {port} is open and likely uses UDP.")
else:
print(f"Port {port} is closed or not responding.")
```

---------- **FileName:** 15NetworkC.py

Content:

```
import random

Randomgenerate1 = random.randint(1024, 65535)
Randomgenerate2 = random.randint(1024, 65535)
Randomgenerate3 = random.randint(1024, 65535)

print(f"First {Randomgenerate1} Second {Randomgenerate2} Third {Randomgenerate3}")
```

---------- **FileName:** 16NetworkC.py

Content:

```
def is_privileged_port(port):
return 0 <= port <= 1023

port = int(input("Enter a port number: "))

if is_privileged_port(port):
print(f"Port {port} is in the privileged range.")
else:
print(f"Port {port} is not in the privileged range.")
```

---------- **FileName:** 17NetworkC.py

Content:

```
import socket

ip = "127.0.0.1"
open_ports = []


for port in range(1, 65536):
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(0.5)
if s.connect_ex((ip, port)) == 0:
print(f"Port {port} is open")
open_ports.append(port)
s.close()

print("\nScan complete.")
if open_ports:
print("Open ports:", open_ports)
else:
print("No open ports found.")
```

---------- **FileName:** 18NetworkC.py

Content:

```
import socket
from datetime import datetime

def tcp_port_scanner(target_host, start_port, end_port):
print(f"Scanning ports on {target_host} from {start_port} to {end_port}...")
print("-" * 50)
start_time = datetime.now()

try:
for port in range(start_port, end_port + 1):
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.settimeout(1)

result = sock.connect_ex((target_host, port))

if result == 0:
print(f"Port {port} is open")
sock.close()

except socket.gaierror:
print("Hostname could not be resolved.")
except socket.error:
print("Could not connect to server.")
except KeyboardInterrupt:
print("\nExiting program.")

end_time = datetime.now()
total_time = end_time - start_time
print("-" * 50)
print(f"Scanning finished in: {total_time}")

if __name__ == "__main__":
target = input("Enter the target IP address or hostname: ")
try:
start_p = int(input("Enter the starting port number: "))
end_p = int(input("Enter the ending port number: "))
if start_p > end_p or start_p < 1 or end_p > 65535:
print("Invalid port range. Please enter valid port numbers (1-65535).")
else:
tcp_port_scanner(target, start_p, end_p)
except ValueError:
print("Invalid port number. Please enter integers.")
```

---------- **FileName:** 19NetworkC.py

Content:

```
import socket
import sys
```

```
def udp_scan(target_host, start_port, end_port, timeout=1):
open_ports = []
for port in range(start_port, end_port + 1):
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.settimeout(timeout)
try:
sock.sendto(b"test", (target_host, port))
data, addr = sock.recvfrom(1024)
open_ports.append(port)
except socket.timeout:
pass
except socket.error as e:
pass
finally:
sock.close()
return open_ports

if __name__ == "__main__":
if len(sys.argv) != 4:
print("Usage: python udp_scanner.py ")
sys.exit(1)

target_host = sys.argv[1]
start_port = int(sys.argv[2])
end_port = int(sys.argv[3])

print(f"Scanning UDP ports on {target_host} from {start_port} to {end_port}...")
found_ports = udp_scan(target_host, start_port, end_port)

if found_ports:
print("Open UDP ports found:")
for p in found_ports:
print(f"- {p}")
else:
print("No open UDP ports found in the specified range.")
```

FileName: 1NetworkC.py

Content:

```
while True:
input_Ter = input("type to see this is ip IPv4 or IPv6: ")
input_Ter_number = len(input_Ter)
if '.' in input_Ter and input_Ter_number > 10:
print("This is an IPv4 address.")
break
if '192.168.' in input_Ter:
print("This is a private IPv4 address.")
break
if input_Ter_number < 11 and ':' in input_Ter:
print("This is an IPv6 address.")
break
if input_Ter == 'exit':
```

```
print("Exiting the program.")
break
else:
print("Error: Invalid input. Please try again.")
```

FileName: 20NetworkC.py

Content:

```
import socket

ports_services = {
20: "FTP Data",
21: "FTP Control",
22: "SSH",
23: "Telnet",
25: "SMTP",
53: "DNS",
67: "DHCP Server",
68: "DHCP Client",
69: "TFTP",
79: "Finger",
80: "HTTP",
88: "Kerberos",
110: "POP3",
111: "RPC",
119: "NNTP",
123: "NTP",
135: "MS RPC",
137: "NetBIOS Name",
138: "NetBIOS Datagram",
139: "NetBIOS Session",
143: "IMAP",
161: "SNMP",
162: "SNMP Trap",
389: "LDAP",
443: "HTTPS",
445: "SMB",
465: "SMTPS",
500: "ISAKMP / IKE",
514: "Syslog",
515: "LPD Printer",
520: "RIP",
587: "SMTP Submission",
631: "IPP Printing",
993: "IMAPS",
995: "POP3S"
}

ip = input("Enter target IP address: ").strip()
open_ports = []

print(f"\nScanning {ip} for high-risk reserved ports...\n")
```

```
for port, service in ports_services.items():
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(0.5)
if s.connect_ex((ip, port)) == 0:
print(f"Port {port} OPEN - {service}")
open_ports.append((port, service))
s.close()

print("\nScan complete.")
if open_ports:
print("\nSummary of open high-risk ports:")
for p, s in open_ports:
print(f"{p} - {s}")
else:
print("No risky ports found.")
```

FileName: 21NetworkCClient.py

Content:

```
import socket
import threading
INPUT = input("enter to massage to send to the server: ")
HEADER = 64
Port = 5050
FORMAT = 'utf-8'
DISCONNECT_MESSAGE = "!DISCONNECT"
SERVER = "192.168.1.12"
ADDR = (SERVER, Port)
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)


def send(msg):
message = msg.encode(FORMAT)
msg_lenght = len(message)
send_lenght = str(msg_lenght).encode(FORMAT)
send_lenght += b' ' * (HEADER - len(send_lenght))
client.send(send_lenght)
client.send(message)

send(INPUT)
```

FileName: 21NetworkCServer.py

Content:

```
import socket
import threading

Port = 5050
```

```
SERVER = socket.gethostbyname(socket.gethostname())
HEADER = 64
ADDR = (SERVER, Port)
FORMAT = 'utf-8'
DISCONNECT_MESSAGE = "!DISCONNECT"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(ADDR)


def handle_client(conn, addr):
print(f"[NEW CONNECTION] {addr} connected.")
connected = True
while connected:
msg_lenght = conn.recv(HEADER).decode(FORMAT)
if msg_lenght:
msg_lenght = int(msg_lenght)
msg = conn.recv(msg_lenght).decode(FORMAT)
if msg == DISCONNECT_MESSAGE:
connected = False
print(f"[{addr}] {msg}")
conn.close()


def start():
s.listen()
print(f"[LISTENING] Server is listening on {SERVER}")
while True:
conn, addr = s.accept()
thread = threading.Thread(target=handle_client, args=(conn, addr))
thread.start()
print(f"[ACTIVE CONNECTIONS] {threading.active_count() - 1}")

print("[STARTING] server is starting...")
start()
```

FileName: 22NetworkCClient.py

Content:

```
import socket

INPUT = input("Enter the message to send to the server: ")
PORT = 6060 # use a high port to avoid permission errors
FORMAT = 'utf-8'
DISCONNECT_MESSAGE = "!DISCONNECT"
SERVER = "192.168.1.12"
ADDR = (SERVER, PORT)

# Create UDP socket
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

def send(msg):
```

```
client.sendto(msg.encode(FORMAT), ADDR) # Send message to server
try:
client.settimeout(2) # Wait max 2 seconds for reply
data, server_addr = client.recvfrom(1024)
print(f"Server replied: {data.decode(FORMAT)}")
except socket.timeout:
print("No response from server.")

send(INPUT)

# Optionally send disconnect message
send(DISCONNECT_MESSAGE)

client.close()
```

FileName: 22NetworkCServer.py

Content:

```
import socket
import threading

PORT = 6060
SERVER = socket.gethostbyname(socket.gethostname())
ADDR = (SERVER, PORT)
FORMAT = 'utf-8'
DISCONNECT_MESSAGE = "!DISCONNECT"

# Create a UDP socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(ADDR)

print(f"[STARTING] UDP server is starting on {SERVER}:{PORT}")

clients = set() # Track client addresses

def handle_message(data, addr):
message = data.decode(FORMAT)
print(f"[{addr}] {message}")

if message == DISCONNECT_MESSAGE:
print(f"[DISCONNECTED] {addr} left")
clients.discard(addr)
else:
clients.add(addr)
# Example: echo back to sender
s.sendto(f"Server received: {message}".encode(FORMAT), addr)

def start():
print(f"[LISTENING] UDP server is ready on {SERVER}:{PORT}")
while True:
data, addr = s.recvfrom(1024) # Receive up to 1024 bytes
threading.Thread(target=handle_message, args=(data, addr)).start()
```

start()

---------- **FileName:** 23NetworkCClient.py

Content:

```
import socket
import threading
INPUT = "Hello, Client!"
HEADER = 64
Port = 9999
FORMAT = 'utf-8'
DISCONNECT_MESSAGE = "!DISCONNECT"
SERVER = "192.168.1.12"
ADDR = (SERVER, Port)
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)


def send(msg):
message = msg.encode(FORMAT)
msg_lenght = len(message)
send_lenght = str(msg_lenght).encode(FORMAT)
send_lenght += b' ' * (HEADER - len(send_lenght))
client.send(send_lenght)
client.send(message)

send(INPUT)
```

---------- **FileName:** 23NetworkCServer.py

Content:

```
import socket
import threading

Port = 9999

SERVER = socket.gethostbyname(socket.gethostname())
HEADER = 64
ADDR = (SERVER, Port)
FORMAT = 'utf-8'
DISCONNECT_MESSAGE = "!DISCONNECT"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(ADDR)


def handle_client(conn, addr):
print(f"[NEW CONNECTION] {addr} connected.")
```

```
connected = True
while connected:
msg_lenght = conn.recv(HEADER).decode(FORMAT)
if msg_lenght:
msg_lenght = int(msg_lenght)
msg = conn.recv(msg_lenght).decode(FORMAT)
if msg == DISCONNECT_MESSAGE:
connected = False
print(f"[{addr}] {msg}")
conn.close()


def start():
s.listen()
print(f"[LISTENING] Server is listening on {SERVER}")
while True:
conn, addr = s.accept()
thread = threading.Thread(target=handle_client, args=(conn, addr))
thread.start()
print(f"[ACTIVE CONNECTIONS] {threading.active_count() - 1}")

print("[STARTING] server is starting...")
start()
```

---------- **FileName:** 24NetworkC.py

Content:

```
import socket, time

def tcp_rtt(host, port):
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
start = time.time()
s.connect((host, port))
end = time.time()
s.close()
return (end - start) * 1000

print(tcp_rtt("google.com", 80))
```

---------- **FileName:** 24NetworkCClient.py

Content:

```
\
```

---------- **FileName:** 25NetworkC.py

Content:

```
import random, time

target_ip = "192.168.1.10"
target_port = 80
count = 0
throttle = 0.1

print(f"Simulating SYN flood on {target_ip}:{target_port}")

try:
while True:
src_ip = ".".join(str(random.randint(1, 254)) for _ in range(4))
src_port = random.randint(1024, 65535)
seq = random.randint(0, 4294967295)
print(f"SYN packet from {src_ip}:{src_port} -> {target_ip}:{target_port} seq={seq}")
count += 1
if count % 10 == 0:
print(f"Simulated {count} packets")
time.sleep(throttle)
except KeyboardInterrupt:
print(f"Simulation stopped after {count} packets")
```

FileName: 26NetworkC.py

Content:

```
import socket
from datetime import datetime

host = '0.0.0.0'
port = 9999
log_file = "tcp_connections.log"

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.bind((host, port))
server.listen(5)

print(f"Listening for incoming TCP connections on port {port}...")

try:
while True:
conn, addr = server.accept()
timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
log_entry = f"{timestamp} - Connection from {addr[0]}:{addr[1]}"
print(log_entry)
with open(log_file, "a") as f:
f.write(log_entry + "\n")
conn.close()
except KeyboardInterrupt:
print("\nStopped.")
server.close()
```

---------- **FileName:** 27NetworkC.py

Content:

```
import random
import time

def generate_seq():
return random.randint(1000, 9999)

def tcp_handshake_simulation():
client_seq = generate_seq()
server_seq = generate_seq()

print("Client: sending SYN")
print(f"Client Seq={client_seq}")
time.sleep(1)

print("Server: received SYN, sending SYN-ACK")
print(f"Server Seq={server_seq}, ACK={client_seq + 1}")
time.sleep(1)

print("Client: received SYN-ACK, sending ACK")
print(f"Client ACK={server_seq + 1}")
time.sleep(1)

print("\nTCP connection established!")

if __name__ == "__main__":
tcp_handshake_simulation()
```

---------- **FileName:** 28NetworkC.py

Content:

```
import socket

server_ip = input("Enter server IP or hostname: ")
port = 23
timeout = 5

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.settimeout(timeout)

try:
sock.connect((server_ip, port))
print(f"Telnet is allowed on {server_ip}:{port}")
except (socket.timeout, socket.error):
print(f"Telnet is NOT allowed on {server_ip}:{port}")
finally:
```

```
sock.close()
```

---------- **FileName:** 29NetworkC.py

Content:

```
import socket
import random
import struct

def build_dns_query(domain):
tid = random.randint(0, 65535)
flags = 0x0100
qdcount = 1
ancount = 0
nscount = 0
arcount = 0

header = struct.pack(">HHHHHH", tid, flags, qdcount, ancount, nscount, arcount)

query = b""
for part in domain.split("."):
query += struct.pack("B", len(part)) + part.encode()
query += b"\x00"
qtype = 1
qclass = 1
query += struct.pack(">HH", qtype, qclass)

return header + query, tid

def parse_dns_response(data, tid):
resp_tid = struct.unpack(">H", data[:2])[0]
if resp_tid != tid:
print("Transaction ID mismatch")
return
print("Received DNS response:")
print(data.hex())

if __name__ == "__main__":
server = ("8.8.8.8", 53)
domain = input("Enter domain to query: ")

packet, tid = build_dns_query(domain)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.settimeout(5)
sock.sendto(packet, server)

try:
data, _ = sock.recvfrom(512)
parse_dns_response(data, tid)
except socket.timeout:
print("No response from server")
```

```
finally:
sock.close()
```

---------- **FileName:** 2NetworkC.py

Content:

```
#maked by kockerpro94
# second code in network
a = input("enter ip for get binary Code: ")
b = len(a)

if '.' in a:
print("Binary representation of the first octet (if it's a valid number):")
try:
octets = a.split('.')
if octets:
print(bin(int(octets[0])))
except ValueError:
print("Invalid IP format for binary conversion.")
```

---------- **FileName:** 30NetworkC.py

Content:

```
import socket
import struct
import os

def get_host_ip():
s = None
try:
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.connect(('8.8.8.8', 80))
ip = s.getsockname()[0]
except Exception:
ip = socket.gethostbyname(socket.gethostname())
finally:
# Ensure the socket is closed even if an exception occurs
if s:
s.close()
return ip

def main():
if os.name == 'nt':
sniffer = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_IP)
host_ip = get_host_ip()
print(f"Binding to interface: {host_ip}")
sniffer.bind((host_ip, 0))
sniffer.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)
```

```
else:
sniffer = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_IP)

print("Packet sniffer started. Press Ctrl+C to stop.")

try:
while True:
raw_buffer, addr = sniffer.recvfrom(65535)

ip_header = raw_buffer[0:20]
iph = struct.unpack('!BBHHHBBH4s4s', ip_header)

version_ihl = iph[0]
ihl = version_ihl & 0xF
ip_header_length = ihl * 4
protocol = iph[6]
s_addr = socket.inet_ntoa(iph[8])
d_addr = socket.inet_ntoa(iph[9])

print(f'\n[+] IP Packet: {s_addr} -> {d_addr} | Protocol: {protocol}')

if protocol == 6:
tcp_header = raw_buffer[ip_header_length:ip_header_length+20]
tcph = struct.unpack('!HHLLBBHHH', tcp_header)

source_port = tcph[0]
dest_port = tcph[1]

print(f' [TCP] {s_addr}:{source_port} -> {d_addr}:{dest_port}')

elif protocol == 17:
udp_header = raw_buffer[ip_header_length:ip_header_length+8]
udph = struct.unpack('!HHHH', udp_header)

source_port = udph[0]
dest_port = udph[1]

print(f' [UDP] {s_addr}:{source_port} -> {d_addr}:{dest_port}')

except KeyboardInterrupt:
print("\nStopping sniffer.")
finally:
if os.name == 'nt':
sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_OFF)
sniffer.close()

if __name__ == '__main__':
main()
```

---------- **FileName:** 3NetworkC.py

Content:

```
file = "3NetworkCFile.txt"
```

```
with open(file, "r") as f:
ReadFile = f.read()
print(ReadFile)
```

FileName: 4NetworkC.py

Content:

```
#maked by kocekrpro94
# this code is easy
def Main():
input_Ter = input("type to see this private ip or no: ")
if '192.168.' in input_Ter:
print("this is private ip")
if '.' in input_Ter:
print("this is public ip")

Main()
```

FileName: 5NetworkC.py

Content:

```
import ipaddress

def calculate_network_broadcast(ip_address, subnet_mask):
"""
Calculates the network address and broadcast address given an IP address and subnet mask.

Args:
ip_address (str): The IP address in string format (e.g., "192.168.1.10").
subnet_mask (str): The subnet mask in string format (e.g., "255.255.255.0").

Returns:
tuple: A tuple containing the network address (str) and broadcast address (str).
Returns (None, None) if an invalid IP or subnet mask is provided.
"""
try:
# Create an IPv4Interface object from the IP address and subnet mask
# This automatically calculates network and broadcast addresses
network_interface = ipaddress.IPv4Interface(f"{ip_address}/{subnet_mask}")

# Extract the network address
network_address = str(network_interface.network.network_address)

# Extract the broadcast address
broadcast_address = str(network_interface.network.broadcast_address)

return network_address, broadcast_address
except ipaddress.AddressValueError as e:
print(f"Error: Invalid IP address or subnet mask provided. {e}")
```

```
return None, None
except Exception as e:
print(f"An unexpected error occurred: {e}")
return None, None

ip = input("Enter IP Address: ")
mask = input("Enter Subnet Mask: ")
network, broadcast = calculate_network_broadcast(ip, mask)

if network and broadcast:
print(f"IP Address: {ip}")
print(f"Subnet Mask: {mask}")
print(f"Network Address: {network}")
print(f"Broadcast Address: {broadcast}")
```

FileName: 6NetworkC.py

Content:

```
import ipaddress

def get_ips_in_subnet(subnet_cidr):
try:
network = ipaddress.ip_network(subnet_cidr)
ip_list = [str(ip) for ip in network.hosts()]
return ip_list
except ValueError as e:
print(f"Error: Invalid subnet CIDR '{subnet_cidr}'. {e}")
return []


subnet = input("Enter a subnet CIDR (e.g., '192.168.1.0/24'): ")
all_ips = get_ips_in_subnet(subnet)

if all_ips:
print(f"All possible IP addresses in {subnet}:")
for ip in all_ips:
print(ip)
```

FileName: 7NetworkC.py

Content:

```
import ipaddress

ipv6_expanded = input("Enter an IPv6 address in expanded form: ")
ipv6_obj = ipaddress.IPv6Address(ipv6_expanded)
ipv6_compressed = ipv6_obj.compressed
print(f"Compressed form: {ipv6_compressed}")
```

---------- **FileName:** 8NetworkC.py

Content:

```
def get_ipv4_class(ip_address):
try:
first_octet = int(ip_address.split('.')[0])
except (ValueError, IndexError):
return "Invalid IP Address"

if 1 <= first_octet <= 126:
return "Class A"
elif 128 <= first_octet <= 191:
return "Class B"
elif 192 <= first_octet <= 223:
return "Class C"
elif 224 <= first_octet <= 239:
return "Class D (Multicast)"
elif 240 <= first_octet <= 254:
return "Class E (Experimental)"
else:
return "Invalid IP Address"
```

---------- **FileName:** 9NetworkC.py

Content:

```
import ipaddress

ip1 = input("enter first ip: ")
ip2 = input("enter first ip1: ")
mask = input("enter mask: ")

def same_subnet():
global ip1, ip2, mask
net1 = ipaddress.ip_network(f"{ip1}/{mask}", strict=False)
net2 = ipaddress.ip_network(f"{ip2}/{mask}", strict=False)
return net1.network_address == net2.network_address
print(same_subnet())
```

---------- **FileName:** main.py

Content:

```
from reportlab.lib.pagesizes import A4
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib import colors
import os

def read_file_content(filename):
"""Safely read file content as text."""
```

```
try:
with open(filename, "r", encoding="utf-8") as f:
return f.read()
except Exception as e:
return f"Error reading file: {e}"

def add_file_section(story, styles, filename, content):
"""Add one file’s section to the PDF."""
section = f"""
---
**FileName:** {filename}


**Content:**
{content.replace('\n', '
')}

---
"""
story.append(Paragraph(section, styles["Normal"]))
story.append(Spacer(1, 20))

def make_pdf():
doc = SimpleDocTemplate("result.pdf", pagesize=A4)
styles = getSampleStyleSheet()
story = []

include_all = input("Do you want to include all files in this folder? (yes/no): ").strip().lower()

if include_all == "yes":
py_files = [f for f in os.listdir() if f.endswith(".py")]
txt_files = [f for f in os.listdir() if f.endswith(".txt")]

# ---- Add .py files sections ----
if py_files:
story.append(Paragraph("**Python Files**", styles["Heading2"]))
story.append(Spacer(1, 10))
for f in py_files:
content = read_file_content(f)
add_file_section(story, styles, f, content)

# ---- Add .txt files sections ----
if txt_files:
story.append(Paragraph("**Text Files**", styles["Heading2"]))
story.append(Spacer(1, 10))
for f in txt_files:
content = read_file_content(f)
add_file_section(story, styles, f, content)

# ---- Python Files Table ----
if py_files:
data = [["Python File Name", "Preview (first 80 chars)"]]
for f in py_files:
code = read_file_content(f)
preview = (code[:80] + "...") if len(code) > 80 else code
data.append([f, preview])
```

```
table = Table(data, colWidths=[200, 300])
table.setStyle(TableStyle([
("BACKGROUND", (0, 0), (-1, 0), colors.lightblue),
("GRID", (0, 0), (-1, -1), 1, colors.black),
("ALIGN", (0, 0), (-1, -1), "LEFT"),
]))
story.append(Paragraph("**Python Files Summary**", styles["Heading3"]))
story.append(table)
story.append(Spacer(1, 20))

# ---- Text Files Table ----
if txt_files:
data = [["Text File Name", "Preview (first 80 chars)"]]
for f in txt_files:
text = read_file_content(f)
preview = (text[:80] + "...") if len(text) > 80 else text
data.append([f, preview])
table = Table(data, colWidths=[200, 300])
table.setStyle(TableStyle([
("BACKGROUND", (0, 0), (-1, 0), colors.lightgreen),
("GRID", (0, 0), (-1, -1), 1, colors.black),
("ALIGN", (0, 0), (-1, -1), "LEFT"),
]))
story.append(Paragraph("**Text Files Summary**", styles["Heading3"]))
story.append(table)

print("■ Added all .py and .txt files to result.pdf")

else:
# ---- Single file mode ----
filename = input("Enter the file name (.py or .txt): ").strip()
if not os.path.exists(filename):
print("■ File not found.")
return
if not (filename.endswith(".py") or filename.endswith(".txt")):
print("■ Only .py and .txt files are allowed.")
return

content = read_file_content(filename)
add_file_section(story, styles, filename, content)
print(f"■ Added {filename} to result.pdf")

# ---- Build PDF ----
doc.build(story)
print("■ PDF created successfully: result.pdf")

if __name__ == "__main__":
make_pdf()
```

Text Files

---------- **FileName:** 3NetworkCFile.txt

Content:
192.168.1.1
192.168.1.2
192.168.1.3
192.168.1.4
192.168.1.5
192.168.1.6
192.168.1.7
192.168.1.8
192.168.1.9

Python Files Summary

Python File Name	Preview (first 80 chars)

10NetworkC.py	common_ports = {
	  20: "FTP (Data Transfer)",
	  21: "FTP (Command Control)"...

12NetworkC.py	Input = input("enter to know the port valid or no: ") def Main(port): retur...
13NetworkC.py	import socket ip = input("Enter target IP address: ").strip() open_ports = [] ...
14NetworkC.py	import socket target_ip = input("Enter the IP to check: ").strip() port = int(i...
15NetworkC.py	import random Randomgenerate1 = random.randint(1024, 65535) Randomgenerate2 = r...
16NetworkC.py	def is_privileged_port(port): return 0 <= port <= 1023 port = int(input("En...

17NetworkC.py	import socket ip = "127.0.0.1" open_ports = [] for port in range(1, 65536): ...
18NetworkC.py	import socket from datetime import datetime def tcp_port_scanner(target_host, s...
19NetworkC.py	import socket import sys def udp_scan(target_host, start_port, end_port, timeou...

1NetworkC.py	while True:
	  input_Ter = input("type to see this is ip IPv4 or IPv6: ")
	  i...

20NetworkC.py	import socket
	
	ports_services = {
	  20: "FTP Data",
	  21: "FTP Control",
	  ...

21NetworkCClient.py	import socket import threading INPUT = input("enter to massage to send to the s...
21NetworkCServer.py	import socket import threading Port = 5050 SERVER = socket.gethostbyname(socke...
22NetworkCClient.py	import socket INPUT = input("Enter the message to send to the server: ") PORT =...
22NetworkCServer.py	import socket import threading PORT = 6060 SERVER = socket.gethostbyname(socket...
23NetworkCClient.py	import socket import threading INPUT = "Hello, Client!" HEADER = 64 Port = 9999 ...

23NetworkCServer.py	import socket import threading Port = 9999 SERVER = socket.gethostbyname(socke...
24NetworkC.py	import socket, time def tcp_rtt(host, port): s = socket.socket(socket.AF_IN...
24NetworkCClient.py	\
25NetworkC.py	import random, time target_ip = "192.168.1.10" target_port = 80 count = 0 throt...
26NetworkC.py	import socket from datetime import datetime host = '0.0.0.0' port = 9999 log_fi...
27NetworkC.py	import random import time def generate_seq(): return random.randint(1000, 9...
28NetworkC.py	import socket server_ip = input("Enter server IP or hostname: ") port = 23 time...
29NetworkC.py	import socket import random import struct def build_dns_query(domain): tid ...

2NetworkC.py	#maked by kockerpro94
	# second code in network
	a = input("enter ip for get binar...

30NetworkC.py	import socket
	import struct
	import os
	
	def get_host_ip():
	  s = None
	  try:
	...

3NetworkC.py	file = "3NetworkCFile.txt" with open(file, "r") as f: ReadFile = f.read() ...
4NetworkC.py	#maked by kocekrpro94 # this code is easy def Main(): input_Ter = input("typ...
5NetworkC.py	import ipaddress def calculate_network_broadcast(ip_address, subnet_mask): ...

6NetworkC.py	import ipaddress
	
	def get_ips_in_subnet(subnet_cidr):
	  try:
	  network =...

7NetworkC.py	import ipaddress ipv6_expanded = input("Enter an IPv6 address in expanded form:...

8NetworkC.py	def get_ipv4_class(ip_address):
	  try:
	  first_octet = int(ip_address.sp...

9NetworkC.py	import ipaddress ip1 = input("enter first ip: ") ip2 = input("enter first ip1: ...
main.py	from reportlab.lib.pagesizes import A4 from reportlab.platypus import SimpleDocT...

Text Files Summary

Text File Name	Preview (first 80 chars)
3NetworkCFile.txt	192.168.1.1 192.168.1.2 192.168.1.3 192.168.1.4 192.168.1.5 192.168.1.6 192.168....