# PROJECTS:

## PYTHON O.S. INFO
## PORTSPY™
## CIPHERCHAT™

## PYTHON FUNDAMENTALS

CFC011023 [ GROUP 2]

STUDENT: TAN KE HAN KEITH [ S9 ]
TRAINER: JAMES

# Table of Contents


**1. INTRODUCTION ..... 2**

**2. LINUX O.S. INFORMATION GENERATOR ..... 3**

2.1 GET USER INPUTS ..... 4
2.2 IMPORT MODULES ..... 4
2.3 DISPLAY O.S. VERSION ..... 5
2.4 DISPLAY IP ADDRESSES ..... 6
*2.4.1 Display Private IP Address ..... 6*
*2.4.2 Display Public IP Address ..... 7*
*2.4.3 Display Default Gateway IP Address ..... 8*
2.5 DISPLAY HARD DISK SPACE, FREE & USED SPACE ..... 9
2.6 DISPLAY TOP 5 DIRECTORIES & THEIR SIZE ..... 10
2.7 DISPLAY CPU USAGE (REFRESH EVERY 10 SECONDS) ..... 11

**3. PORTSPY™ - MULTI-THREADED OPEN PORT SCANNER ..... 13**

3.1 IMPORT MODULES & DEFINE USER INPUTS ..... 14
3.2 QUEUE & LIST INITIALIZATION, DEFINE SCAN PROCESS OF A SINGLE PORT ..... 15
3.3 PUT & GET PORT NUMBERS FROM QUEUE, DEFINE OVERALL SCAN PROCESS ..... 15
3.4 SET UP MULTI-THREADING, START & COMPLETE PORT SCANS ..... 16

**4. CIPHERCHAT™ - RSA-ENCRYPTED CHATROOM ..... 18**

4.1 IMPORT MODULES & DEFINE USER INPUTS ..... 19
4.2 SET UP RSA ENCRYPTION ..... 20
4.3 SET UP HOSTING VS CONNECTING LOGIC ..... 20
4.4 SET UP SENDING & RECEIVING MESSAGES ..... 21
4.5 SET UP MULTI-THREADING ..... 22

**CONCLUSION ..... 23**

**REFERENCES (LINUX O.S. INFORMATION GENERATOR) ..... 24**

**REFERENCES (PORTSPY™) ..... 26**

**REFERENCES (CIPHERCHAT™) ..... 27**

# 1. Introduction

This report documents 3 Python projects developed to hone my Python scripting skills, while concurrently satiating my curiosity:

- **Linux O.S. Information Generator:** Automatically displays vital system information (O.S. details, IP Addresses, disk usage, largest directories, real-time CPU usage) of any given Linux machine.
- **PortSpy™:** Performs a multi-threaded port scan for open ports, of any given target IP address, using socket connections
- **CipherChat™:** A responsive, private chatroom that uses multi-threading coupled with RSA to encrypt data-in-transit, ensuring the confidentiality of messages sent.

## 2. Linux O.S. Information Generator

This Python script automatically extracts and displays key O.S. information of a Linux machine. The overall script output is displayed below, and explained over the subsequent sub-sections:

```
┌──(kali㉿kali)-[~/PF/pyproj]
└─$ python3 LinuxOSGen_KeithTan.py
Tell me your name: Keith Tan

Hello, Keith Tan.

Welcome to O.S. InfoGen™ by Keith Tan.
(starting the script in 3 seconds ... )

Operating System details:
Operating System: Linux
- Version: #1 SMP Debian 6.5.6-1kali1 (2023-10-09)
- Version Release: 6.5.0-kali3-arm64
- Processor Architecture: aarch64

Private IP Address:
172.16.156.132

Public IP Address:
109.70.100.70

Default Gateway IP Address:
172.16.156.2

Hard Disk Details:
- Hard Disk Size: 18.086 GB
- Hard Disk Free Space: 0.546 GB
- Hard Disk Used Space: 16.597 GB

Top 5 Largest Directories & Sizes:
Rank 1: /                    16.737354669 GB
Rank 2: /usr                 12.759016616 GB
Rank 3: /usr/lib             7.777246384 GB
Rank 4: /usr/share           3.453458086 GB
Rank 5: /var                 3.306896666 GB

Display CPU Usage/Utilization Rate:
(refreshes every 10 seconds, press Control+C to exit ... )
CPU Usage: 0.5% (Time now: 20 Jan 2024 12:17:25 AM)
CPU Usage: 0.8% (Time now: 20 Jan 2024 12:17:36 AM)
CPU Usage: 0.3% (Time now: 20 Jan 2024 12:17:47 AM)
CPU Usage: 1.2% (Time now: 20 Jan 2024 12:17:58 AM)
CPU Usage: 0.5% (Time now: 20 Jan 2024 12:18:09 AM)
CPU Usage: 0.8% (Time now: 20 Jan 2024 12:18:20 AM)
CPU Usage: 1.2% (Time now: 20 Jan 2024 12:18:31 AM)
^CTraceback (most recent call last):
  File "/home/kali/PF/pyproj/LinuxOSGen_KeithTan.py", line 171, in <module>
    time.sleep(10)
KeyboardInterrupt
```

## 2.1 Get User Inputs

The script starts off by defining the shebang to inform the machine to interpret the script in Python. Subsequently, the user is prompted to provide their name to create a personalized experience:

```python
#!/usr/bin/python3



#i. GET USER INPUTS
name=input("Tell me your name: ")
print('')
print(f"\033[1mHello, {name}.\033[0m")
print('')
print("\033[1mWelcome to O.S. InfoGen™ by Keith Tan.\033[0m")
print("(starting the script in 3 seconds...)")
import time
time.sleep(3)
print('')
```

The script outputs the following results:

```
┌──(kali㉿kali)-[~/PF/pyproj]
└─$ python3 LinuxOSGen_KeithTan.py
Tell me your name: Keith Tan

Hello, Keith Tan.

Welcome to O.S. InfoGen™ by Keith Tan.
(starting the script in 3 seconds ... )
```

## 2.2 Import Modules

Python modules are files that define certain functions in an organized manner, and promotes code reusability to improve the scripting efficiency of other Python users (realpython.com, 2024). The following Python modules are installed for the following purposes, to ensure that subsequent parts of the script run smoothly:

```python
#ii. IMPORT MODULES
import platform #to get O.S. information
import netifaces #to get Private IP Address
import requests #to get Public IP Address
import socket #to get Default Gateway IP Address
import psutil #to get hard disk information
import subprocess #to execute shell command 'du' and get top 5 largest directories
import re #to use REGEX to better-display directory results
```

- **Platform:** Allow extraction of a host's O.S. information, such as O.S. name, version and architecture. (askpython.com, 2021)
- **Netifaces:** Allow retrieval of network interface information, such as private IP address of the host. Examples of network interfaces include (baeldung.com, 2022):
    - Ethernet Interface (eth0, eth1, etc.)
    - Wireless Interface (wlan0, wlan1, etc.)
    - Loopback Interface, for internal communication within same machine (lo)
    - Virtual Interfaces, for virtualization purposes (e.g., br0, virbr0)
    - Tunnel Interfaces, for tunnelling protocols/VPNs (e.g., tun0, tap0)
    - Bluetooth Interface (bnep0)
    - Cellular Interface, for cellular network connections (ppp0, wwan0)
    - Bonded Interfaces, single-logical interface for load balancing to aggregate several physical interfaces (bond0)
    - VLAN Interfaces (vlan0, vlan1):
    - InfiniBand Interfaces, for high performance (cluster) computing over InfiniBand networks (ib0, ib1):

- **Requests:** Enable HTTP requests to fetch public IP address information from an external API.
- **Socket:** Provides networking functions, and is used to obtain the default gateway IP address.
- **Psutil:** Allow retrieval of system utilization information, and is used to obtain information on hard disk usage as well as CPU usage.
- **Subprocess:** Allows the Pytho script to run shell commands directly, and is used to run the du command with Linux text manipulation to obtain information on the Top 4 directories
- **Re:** Supports the use of REGEX, and is used to better display results of Top 5 directories after executing the 'du' command.

## 2.3 Display O.S. Version

The 'display_os_version()' function is defined and called with the following commands, using various methods from the platform module, to print out the host's O.S. details:

```
29  #iii. DISPLAY O.S. VERSION
30  print("\033[1mOperating System details:\033[0m")
31
32  def display_os_version():
33      os_name=platform.system()
34      os_ver=platform.version()
35      os_verrelease=platform.release()
36      os_mach=platform.machine()
37      print(f"Operating System: {os_name}")
38      print(f"- Version: {os_ver}")
39      print(f"- Version Release: {os_verrelease}")
40      print(f"- Processor Architecture: {os_mach}")
41
42  display_os_version()
43  print('')
44
45
46
```

- **Line 33:** platform.system() extracts the string 'Linux' and saves to a new variable 'os_name'
- **Line 34:** platform.version() extracts the O.S. version and saves to a new variable 'os_ver'
- **Line 35:** platform.release() extracts the O.S. version release and saves to a new variable 'os_verrelease'
- **Line 36:** platform.machine() extracts the host's processor architecture and saves to a new variable 'os_mach'
- **Lines 37 to 40:** O.S. information are printed to the user.
- **Line 42:** Function is called.

The script outputs the following results to display the host's O.S. information:

```
Operating System details:
Operating System: Linux
- Version: #1 SMP Debian 6.5.6-1kali1 (2023-10-09)
- Version Release: 6.5.0-kali3-arm64
- Processor Architecture: aarch64
```

## 2.4 Display IP Addresses

Subsequently, private IP address, public IP address, and default gateway IP address are printed.

### 2.4.1 Display Private IP Address

The 'get_private_ip()' function is defined and called with the following commands, using various methods from the netifaces module, to print out the host's private IP address:

```
47  #iv. DISPLAY PRIVATE IP ADDRESS, PUBLIC IP ADDRESS, DEFAULT GATEWAY
48
49  # Display Private IP address:
50
51  def get_private_ip(): #function that gets private IPv4 address of host, using netifaces module
52      try:
53          for el in netifaces.interfaces(): #look through network interfaces of host machine
54              if el != "lo":  # 'lo' represents loopback interface, to ignore it...
55                  addresslst = netifaces.ifaddresses(el) #extract dictionary of info for each network interface
56                  if netifaces.AF_INET in addresslst: #check if IPv4 is within dictionary of info, if yes...
57                      priv_ip = addresslst[netifaces.AF_INET][0]['addr'] #get the 1st IPv4 address of interface
58                      print(priv_ip) #private IP address obtained
59          return "No private IP address found."
60      except Exception as err:
61          return f"Unable to fetch private IP. Error: {err}"
62
63  print("\033[1mPrivate IP Address:\033[0m")
64  get_private_ip()
65  print('')
66
```

- **Lines 51 to 52:** The 'get_private_ip()' function uses netifaces.interfaces() to iterate through a list of network interfaces on the host machine.
- **Lines 53 to 54:** For each network interface (denoted by 'el'), check if the interface is a loopback interface and exclude it (because loopback interface is only used for internal communication/troubleshooting).
- **Line 55:** For each network interface (excluding loopback interface), retrieve a dictionary of interface information using 'netifaces.ifaddresses(el)'. Save result to 'addresslst'.
- **Lines 56 to 59:** Check if the dictionary of interface information within 'addresslst' includes IPv4 address. If yes, extract and print the private IP address. If no, print to the user that no private IP address was found.
- **Lines 60 to 61:** If an exception occurs (for example, if the netifaces module is not installed), print the error message to alert the user.
- **Line 64:** Function is called.

### 2.4.2 Display Public IP Address

The 'get_public_ip()' function is defined and called with the following commands, using various methods from the requests module, to print out the host's public IP address:

```
67  # Display Public IP address:
68
69  def get_public_ip(): #function that gets public IP address of host, using requests module
70      try:
71          #HTTP GET request to a specific URL that gets info on the host's public IP address:
72          response = requests.get('https://api64.ipify.org?format=json')
73          data = response.json() #parse API response (in JSON format), to python dictionary format
74          public_ip = data['ip'] #extract value (public IP address) from key-value pair, using key 'ip'
75          print(public_ip)
76      except Exception as err:
77          return f"Error: {err}"
78  print("\033[1mPublic IP Address:\033[0m")
79  get_public_ip()
80  print('')
81
```

- **Lines 69 to 72:** The function uses ‘requests.get()’ to run a HTTP GET request to an API, by connecting to the URL 'https://api64.ipify.org?format=json'. This API provides information on the public IP address of the host.
- **Line 73:** ‘response.json()’ parses the API response (in JSON format), and converts it into a Python dictionary. Results are saved into new variable ‘data’.
- **Lines 74 to 75:** Using the key ‘ip’, the public IP address (value of key ‘ip’) is extracted from the dictionary ‘data’, and printed out.
- **Lines 76 to 77:** If an exception occurs (for example, if failed to fetch information from API), print the error message and alert the user.
- **Line 79:** Function is called.

### 2.4.3 Display Default Gateway IP Address

The ‘get_default_gateway()’ function is defined and called with the following commands, using various methods from the netifaces module, to print out the default gateway IP address:

```
82  # Display Default Gateway:
83
84  def get_default_gateway(): #function that gets Default Gateway IP addr, using netifaces module
85      try:
86          # Get default gateway for the default interface (usually 'eth0' or 'wlan0'):
87          # Use netifaces.gateways() to get dictionary of gateway info...
88          # Then filter for 'default', then filter for IPv4, and get the first result...
89          # Which is the Default Gateway IP Address
90          dg_ip = netifaces.gateways()['default'][netifaces.AF_INET][0]
91          print(dg_ip)
92      except (KeyError, netifaces.gaierror) as err:
93          print(f"Error: {err}")
94          return None
95  print("\033[1mDefault Gateway IP Address:\033[0m")
96  get_default_gateway()
97  print('')
98
99
100
```

- **Lines 84 to 91:** The function uses ‘netifaces.gateways()’ to get a dictionary of gateway information of the host machine. The result is then filtered for default gateway using ‘[‘default’]’, and then filtered for the first IPv4 address using ‘[netifaces.AF_INET][0]’. The default gateway IP address is finally obtained, and printed out.
- **Lines 92 to 94:** Two types of exceptions are caught in this exception-handling block –
    - KeyError: Errors that occur when accessing a key that is not found in a dictionary.
    - ‘netifaces.gaierror’: Errors that occur when running the netifaces module (abstractapi.com, 2023).
- **Line 96:** Function is called.

The script outputs the following results to display the host's private IP address, public IP address, and default gateway:

```
Private IP Address:
172.16.156.132

Public IP Address:
109.70.100.70

Default Gateway IP Address:
172.16.156.2
```

## 2.5 Display Hard Disk Space, Free & Used Space

The 'get_disk_size()' function is defined and called with the following commands, using various methods from the psutil module, to print out the host's hard disk details:

```
101 #v. DISPLAY HARD DISK SIZE, FREE & USED SPACE
102
103 def get_disk_size(): #function that gets and prints hard disk size & space, using psutil module
104     try:
105         # Get disk usage statistics:
106         disk_usage = psutil.disk_usage('/') #use psutil to get disk size & space info of directory '/'
107
108         hd_size = disk_usage.total
109         free_space = disk_usage.free
110         used_space = disk_usage.used
111
112         # Convert from bytes (default unit) to gigabytes:
113         hd_size_gb = round(hd_size / (1024 ** 3), 3)
114         free_space_gb = round(free_space / (1024 ** 3), 3)
115         used_space_gb = round(used_space / (1024 ** 3), 3)
116
117         print(f"- Hard Disk Size: {hd_size_gb} GB")
118         print(f"- Hard Disk Free Space: {free_space_gb} GB")
119         print(f"- Hard Disk Used Space: {used_space_gb} GB")
120
121     except Exception as err:
122         print(f"Error: {err}")
123
124 print("\033[1mHard Disk Details:\033[0m")
125 get_disk_size()
126 print('')
127
128
129
```

- **Lines 103 to 106:** The function uses the method 'psutil.disk_usage('/') to extract the disk usage statistics for the root directory ('/'), which is by default the biggest directory in a Linux machine. Results are saved into a new variable 'disk_usage'.
- **Lines 108 to 110:** The following methods are used on 'disk_usage' to extract the following:
    - Total hard disk size: 'disk_usage.total' extracts total hard disk size and saves to a new variable 'hd_size'.
    - Free hard disk space: 'disk_usage.free' extracts free hard disk space and saves to a new variable 'free_space'.
    - Used hard disk space: 'disk_usage.used' extracts used hard disk space and saves to a new variable 'used_space'.

- **Lines 112 to 119:** Results are converted from bytes to gigabytes to improve readability, before being printed.
- **Lines 121 to 122:** If an exception occurs (for example, permission issues to access the hard disk statistics), print the error message and alert the user.
- **Line 125:** Function is called.

The script outputs the following results to display the host's hard disk size, free and used spaces:

```
Hard Disk Details:
- Hard Disk Size: 18.086 GB
- Hard Disk Free Space: 0.546 GB
- Hard Disk Used Space: 16.597 GB
```

## 2.6 Display Top 5 Directories & Their Size

The 'top_directories()' function is defined and called with the following commands, using various methods from the subprocess module, to search and print out the host's top 5 largest directories and their sizes:

```
130  #vi. DISPLAY TOP 5 DIRECTORIES & THEIR SIZE
131
132  def top_directories(): #function that gets directory info, by running command in Terminal with subprocess module
133      # Define the command to type in terminal:
134      # Virtual and dynamic directories are excluded as they do not reflect true/persistent storage
135      termcmd = "sudo du -b --exclude=/proc --exclude=/sys --exclude=/dev --exclude=/run / | sort -nr | head -n5"
136
137      # Use 'subprocess.run()' to execute the command in terminal, and capture the output
138      result = subprocess.run(termcmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
139
140      # Print the output or handle it as needed
141      if result.returncode == 0: #if the command is successfully executed in terminal... print the output...
142          # Use regex and split the output into lines based on whitespace (size relative to directory)
143          lines = result.stdout.split('\n')
144          for i, line in enumerate(lines):
145              if line:
146                  size, directory = re.split(r'\s+', line, maxsplit=1)
147                  print(f"Rank {i+1}: {directory.ljust(20)} {int(size)/10**9} GB") #justify text & convert to GB
148      else: #if the command failed to execute in terminal... print the error message...
149          print(result.stderr)
150
151  print("\033[1mTop 5 Largest Directories & Sizes:\033[0m")
152  top_directories()
153  print('')
154
155
156
```

- **Lines 132 to 135:** The command to be typed onto Terminal is defined and saved to a new variable 'termcmd'. This command uses 'du' to calculate disk usage. Virtual and dynamic directories (/proc, /sys, /dev, /run) that do not reflect persistent storage are excluded. The results are sorted from largest directory sizes, and top 5 directories are to be displayed.

- **Lines 137 to 138:** The terminal command stored in 'termcmd' is executed using 'subprocess.run()':
    - 'shell=TRUE': run command on shell (Terminal)
    - 'stdout=subprocess.PIPE': capture standard outputs
    - 'stderr=subprocess.PIPE': capture standard errors
    - 'text=True': tell subprocess to communicate in text mode, instead of bytes
- **Lines 140 to 147:** If command is executed successfully in Terminal, the 'result.returncode' will be 0. If so, use REGEX to display the output nicely, separating the directory name and directory size. Print results in gigabytes and justify the rows for improved readability.
- **Lines 148 to 149:** If the command fails to execute in Terminal, print out the standard error and alert the user.
- **Line 152:** Function is called.

The script outputs the following results to display the top 5 largest directories and their sizes:

```
Top 5 Largest Directories & Sizes:
Rank 1: /                    16.737354669 GB
Rank 2: /usr                 12.759016616 GB
Rank 3: /usr/lib             7.777246384 GB
Rank 4: /usr/share           3.453458086 GB
Rank 5: /var                 3.306896666 GB
```

## 2.7 Display CPU Usage (refresh every 10 seconds)

The 'get_cpu_usage()' function is defined and called with the following commands, using various methods from both the time and psutil modules, to print out the host's live CPU usage at 10-second intervals:

```
#vii. DISPLAY CPU USAGE (REFRESH EVERY 10s)

from datetime import datetime #to display the 10s refreshes

def get_cpu_usage():
    return psutil.cpu_percent(interval=1) #use 'psutil.cpu_percent()' to get live CPU usage percentage
    #'interval=1' informs Python to get the average CPU usage over the last 1 second

print("\033[1mDisplay CPU Usage/Utilization Rate:\033[0m")
print("(refreshes every 10 seconds, press Control+C to exit...)")

while True: #run this loop forever unless user stops it...
    cpu_usage = get_cpu_usage() #get current CPU usage (refer to function above)
    current_time = datetime.now().strftime("%d %b %Y %I:%M:%S %p") #format current time
    print(f"CPU Usage: {cpu_usage}% (Time now: {current_time})") #print current time
    time.sleep(10) #to pause for 10 seconds before printing CPU usage again
```

- **Line 159:** Import the ‘datetime’ class from datetime module. This allows ‘datetime’ to be directly referenced in subsequent code.
- **Lines 161 to 163:** The method ‘psutil.cpu_percent()’ retrieves the live CPU usage percentage. The parameter ‘interval=1’ informs Python to get the average CPU usage over the last 1 second.
- **Lines 168 to 172:** A ‘while True’ loop is created, which runs indefinitely until the user stops the script manually.
    - ‘get_cpu_usage()’ function is called to retrieve the live CPU usage, and saved to a new variable ‘cpu_usage’.
    - ‘datetime.now().strftime("%d %b %Y %I:%M:%S %p")’ is used to retrieve and display the current time, and saved to a new variable ‘current_time’.
    - The CPU usage and time is printed every 10 seconds, in perpetuity.

The script outputs the following results showing the host’s live CPU usage every 10 seconds:

```
Display CPU Usage/Utilization Rate:
(refreshes every 10 seconds, press Control+C to exit...)
CPU Usage: 0.5% (Time now: 20 Jan 2024 12:17:25 AM)
CPU Usage: 0.8% (Time now: 20 Jan 2024 12:17:36 AM)
CPU Usage: 0.3% (Time now: 20 Jan 2024 12:17:47 AM)
CPU Usage: 1.2% (Time now: 20 Jan 2024 12:17:58 AM)
CPU Usage: 0.5% (Time now: 20 Jan 2024 12:18:09 AM)
CPU Usage: 0.8% (Time now: 20 Jan 2024 12:18:20 AM)
CPU Usage: 1.2% (Time now: 20 Jan 2024 12:18:31 AM)
^CTraceback (most recent call last):
  File "/home/kali/PF/pyproj/LinuxOSGen_KeithTan.py", line 171, in <module>
    time.sleep(10)
KeyboardInterrupt
```

## 3. PortSpy™ - Multi-Threaded Open Port Scanner

Introducing PortSpy™. PortSpy™ is a Python script that allow users to quickly identify open ports on a target machine. PortSpy™ performs a multi-threaded port scan of a specified range of port numbers, for a specific target IPv4 address. The user inputs the target IPv4 address, range of port numbers to scan, and number of threads to use to control scanning efficiency. The Python module socket is used to make connections attempts and check for open ports.

The overall script output is displayed below, and explained over the subsequent sub-sections:

- Single-Thread: Slower scan time = 160 seconds

```
┌──(kali㉿kali)-[~/PF/pyproj]
└─$ python3 PortSpy_KeithTan.py
Welcome to PortSpy™ by Keith Tan!
Enter the target IP Address to scan:192.168.50.193
Enter the lowest port number to scan:0
Enter the highest port number to scan:65535
Enter the number of threads to use:1
Port 22 is OPEN.
Port 5000 is OPEN.
Port 7000 is OPEN.
Port 55248 is OPEN.
Port 57621 is OPEN.
Port 57872 is OPEN.
Summary of OPEN Ports: [22, 5000, 7000, 55248, 57621, 57872]
Scan time: 160 seconds
```

- Multi-Threads (10000 nos.): Faster scan time = 134 seconds

```
┌──(kali㉿kali)-[~/PF/pyproj]
└─$ python3 PortSpy_KeithTan.py
Welcome to PortSpy™ by Keith Tan!
Enter the target IP Address to scan:192.168.50.193
Enter the lowest port number to scan:0
Enter the highest port number to scan:65535
Enter the number of threads to use:10000
Port 22 is OPEN.
Port 5000 is OPEN.
Port 7000 is OPEN.
Port 55248 is OPEN.
Port 57621 is OPEN.
Port 57872 is OPEN.
Summary of OPEN Ports: [22, 5000, 7000, 55248, 57621, 57872]
Scan time: 134 seconds
```

## 3.1 Import Modules & Define User Inputs

```
1   #!/usr/bin/python3
2
3   #PREFACE: IMPORT MODULES
4   import time #to track time taken to finish scan
5   import socket #to create a virtual communication channel, to attempt reaching specific IPv4 address & its port numbers
6   import threading #to perform multi-threading and speed up the port scanning process
7       #(note: python simulates multi-threading, but does not actually perform it)
8   from queue import Queue #Queue analogy: List of elements, each element deleted after being 'executed upon', shortening queue
9
10
11
12  #1. DEFINE USER INPUTS:
13  print("Welcome to PortSpy™ by Keith Tan!")
14  targetIP=input("Enter the target IP Address to scan:")
15  startport=int(input("Enter the lowest port number to scan:"))
16  endport=int(input("Enter the highest port number to scan:"))
17  numthreads=int(input("Enter the number of threads to use:"))
18
19
20
```

- **Lines 3 to 8:** The following Python modules are installed for the following purposes, to ensure that subsequent parts of the script run smoothly:
    - **Time:** Track the amount of time needed to complete the scan.
    - **Socket:** Create a virtual communication channel and attempt to reach the port numbers of a particular IPv4 address.
    - **Threading:** To incorporate multi-threading in Python and speed up the port scanning process. Multi-threading allows multiple processes to run concurrently, using the same shared memory (TechTarget.com, 2023).
        - However, it is worth noting that in Python, there is something called the Global Interpreter Lock (GIL), which only allows 1 thread to execute Python bytecode at any single point in time and therefore, even with multi-threading, threads cannot really run parallel during code execution (realpython.com, 2018).
        - In the context of PortSpy™, each thread checks a different port for openness. While each thread waits for a response from their respective ports, other threads continue to scan other ports. This creates a minor overlapping of tasks, thereby speeding up the port scanning process.
    - **Queue:** The 'Queue' class from queue module is used to create a queue. This queue is created to handle multi-threading, to manage the delegation of port numbers to scan, across different threads.
- **Lines 12 to 17:** Users are prompted to enter their target IP address to scan, desired port range, and number of threads for multi-threading of the scanning process.

## 3.2 Queue & List Initialization, Define Scan Process of a Single Port

```
21  #2: DEFINE STORAGES:
22  queue=Queue() #create empty queue to prepare 'put'-ting in port numbers that user inputs
23  openports=[] #create empty list to store all open ports, as queue progresses
24
25
26
27  #3: DEFINE SCANNING OF SINGLE PORT (CHECK IF CONNECTION CAN BE ESTABLISHED - USING SOCKET):
28  def scanport(targetport):
29      try:
30          # socket analogy: virtual communication channel
31          # - 'socket.AF_INET' states to use IPv4 only
32          # - 'socket.SOCK_STREAM' states to use TCP connection
33          s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
34          s.connect((targetIP, targetport)) #attempt to connect to specific IPv4 and port number
35          return True #connection is successful (boolean outcome to be used for 'scanner' function)
36      except:
37          return False #connection is unsuccessful (boolean outcome to be used for 'scanner' function)
38
39
40
```

- **Lines 21 to 23:** An empty queue is created with 'Queue()', for ports to be 'put' into this queue for processing by multiple threads. An empty list is also created to store port numbers discovered to be open (happens after port scans).
- **Lines 27 to 34:** The function 'scanport(targetport)' uses socket module to check if connections can be established with specific ports.
    - 'socket.AF_INET' is a flag that states to use IPv4 address for the socket connection attempts, while 'socket.SOCK_STREAM' specifies to use TCP connection.
    - 's.connect()' represents an attempt to connect to the target IPv4 address and a specific port number.
- **Lines 35 to 37:** If connection is established, return True. If unestablished, return False. This Boolean outcome will be used to make decisions at a later stage of the script.

## 3.3 Put & Get Port Numbers from Queue, Define Overall Scan Process

```
41  #4: 'PUT' PORT NUMBERS INTO QUEUE:
42  portlist=range(startport,endport)
43
44  def fillqueue(portlist):
45      for el in portlist:
46          queue.put(el) #FIFO: the port 'put' into queue first, will be scanned first
47  fillqueue(portlist)
48
49
50
51  #5: 'GET' PORT NUMBERS FROM QUEUE, 'scanport' EACH PORT NUMBER, PRINT PORT NUMBER IF OPEN:
52  def scanner():
53      while not queue.empty(): #while there are elements (yet to be scanned) still in the queue...
54          portnum=queue.get() #select the NEXT element from queue (a port number), and assign it to 'port' variable
55          if scanport(portnum): #if the function 'scanport(targetport)' is true (aka if the port is OPEN)...
56              print(f"Port {portnum} is OPEN.".format(portnum))
57              openports.append(portnum) #add the newly-found OPEN port to the 'openports' list
58
59
60
```

- **Lines 41 to 47:** Based on user's inputs, the desired port range is captured by the variable 'portlist'. This 'portlist' is 'put' into the queue using a for-loop and the method 'queue.put(el)' – where 'el' represents each port number.

- **Lines 51 to 57:** A new function ‘scanner’ contains a ‘while not queue.empty()’ loop that keeps retrieving port numbers and will not stop until all port numbers are scanned.
    - ‘queue.get()’ method extracts the next port number (not scanned yet) from the queue, as ‘portnum’.
    - ‘portnum’ is passed as a parameter into the ‘scanport()’ function, to attempt socket connection and check if the port is open.
    - If port is open (True), print the open port number to the user. Also, add the open port number to the empty list, ‘openports’.
    - Repeat the steps until the queue is empty.

## 3.4 Set Up Multi-Threading, Start & Complete Port Scans

```
61  #6: MULTI-THREAD 'scanner' FUNCTION TO SPEED UP SCANS:
62  threadlist=[]
63  for el in range(numthreads):
64      thread = threading.Thread(target=scanner) #define new 'thread' variable, allow MULTIPLE 'scanner'-s at once
65      threadlist.append(thread) #add each 'thread' into the list 'threadlist'
66
67
68
69  #7: START SCAN, TRACK SCAN TIME, END SCAN:
70  starttime=time.time() #record start time
71
72  for thread in threadlist:
73      thread.start() #start the thread (which will execute multiple 'scanner' functions concurrently)
74  for thread in threadlist:
75      thread.join() #wait for ALL threads to complete
76
77  endtime=time.time()
78  elapsed=endtime-starttime
79
80
81
82  #CONCLUDE: PRINT RESULTS:
83  print(f"Summary of OPEN Ports: {openports}") #print all OPEN ports
84  print(f"Scan time: {int(elapsed)} seconds")
```

- **Lines 61 to 65:** This section creates multiple threads to allow many ‘scanner’ function to run concurrently:
    - Firstly, an empty list ‘threadlist’ is created.
    - Next, using a for-loop, for each count of threads (represented by ‘el’) in number of threads desired by the user (represented by ‘numthreads’), create a thread using the method ‘threading.Thread(target=<parameter>)’ – where the ‘scanner()’ function is fed to create each thread.
    - Finally, these threads are appended into the empty list ‘threadlist’.
- **Lines 69 to 78:** This section starts the threads using a for-loop on ‘threadlist’, to call on each thread to start (‘scanner()’ function starts running), using the method ‘thread.start()’. Next, the method ‘thread.join()’ is called on each thread to ensure that the script waits for

each thread to finish, before continuing, thereby ensuring that all specified ports will be scanned. The scan duration (elapsed time) is recorded using the method ‘time.time()’.

- **Lines 82 to 84:** Finally, results are printed as ‘openports’, which displays as a list of open ports for the user. The elapsed time taken for the scan to complete is also printed.

# 4. CipherChat™ - RSA-Encrypted Chatroom

Welcome to CipherChat™, where secure communication meets simplicity. CipherChat™ is private chatroom between two users, where they either host a new chatroom or connect to an existing one. Data-in-transit is secured using RSA encryption. As the script runs, both users exchange public keys to establish a secure communications channel. To enhance responsiveness, multi-threading is added to support the simultaneous sending and receiving of RSA-encrypted messages.

To simulate CipherChat™ the following two machines were set up, with the conversations (sent & received messages) recorded:

1. **Kali Linux Machine (SERVER):**
   - IP Address: 172.16.156.132

```
┌──(kali㉿kali)-[~/PF/pyproj]
└─$ python3 CipherChat_KeithTan.py
Welcome to CipherChat™, your go-to encrypted chatroom!
Enter your name:Keith Tan
Enter your peer's name:CFC James
Enter '1' to HOST new chat. Enter '2' to CONNECT to existing chat.1
You will be HOSTING a new CipherChat™ room.
Enter your machine's IP Address:172.16.156.132
Choose a Port Number on your machine to listen:9995
Hi James, I need help for Q2 of this scenario ...
Keith Tan(YOU): Hi James, I need help for Q2 of this scenario ...
CFC James: So what have you tried?
I tried this and that but did not manage to get the answer.
Keith Tan(YOU): I tried this and that but did not manage to get the answer.
CFC James: Your approach is correct, check for typo errors.
Ok got it thanks!
Keith Tan(YOU): Ok got it thanks!
CFC James: Good job!
Bye James!
```

2. **Macbook Pro Machine (CLIENT):**
   - Connects to the host.

```
keithtan@Keiths-MBP:~
# ~                                                    ✔ 11:51:27
scp kali@172.16.156.132:/home/kali/PF/pyproj/CipherChat_KeithTan.py /Users/keithtan
kali@172.16.156.132's password:
CipherChat_KeithTan.py                      100% 2414     3.2MB/s   00:00

# ~                                                    ✔ 3s 11:51:49
clear
# ~                                                    ✔ 11:51:54
python3 CipherChat_KeithTan.py
Welcome to CipherChat™, your go-to encrypted chatroom!
Enter your name:CFC James
Enter your peer's name:Keith Tan
Enter '1' to HOST new chat. Enter '2' to CONNECT to existing chat.2
You will be CONNECTING to an existing CipherChat™ room.
Enter the server's IP Address:172.16.156.132
Enter the server's Port Number:9995
Keith Tan: Hi James, I need help for Q2 of this scenario...
So what have you tried?
CFC James(YOU): So what have you tried?
Keith Tan: I tried this and that but did not manage to get the answer.
Your approach is correct, check for typo errors.
CFC James(YOU): Your approach is correct, check for typo errors.
Keith Tan: Ok got it thanks!
Good job!
CFC James(YOU): Good job!
Keith Tan: Bye James!
```

## 4.1 Import Modules & Define User Inputs

```
1   #!/usr/bin/python3
2
3   #PREFACE: IMPORT MODULES
4   import socket
5   import threading
6   import rsa
7
8
9
10  #1. DEFINE USER INPUTS:
11  print("Welcome to CipherChat™, your go-to encrypted chatroom!")
12  yourname=input("Enter your name:")
13  peername=input("Enter your peer's name:")
14  userinp=str(input("Enter '1' to HOST new chat. Enter '2' to CONNECT to existing chat."))
15
16
17
```

- **Lines 1 to 9:** The following Python modules are installed for the following purposes, to ensure that subsequent parts of the script run smoothly:
    - **Socket:** Create a virtual communication channel between the client and server, to send and receive messages.
    - **Threading:** To incorporate multi-threading in Python, which helps to better manage inbound/outbound (I/O), receiving and sending of messages concurrently, without blocking the execution of each other. This also improves chat responsiveness.
        - However, it is worth noting that in Python, there is something called the Global Interpreter Lock (GIL), which only allows 1 thread to execute Python bytecode at any single point in time and therefore, even with multi-threading, threads cannot really run parallel during code execution (realpython.com, 2018).
        - In the context of CipherChat™, two threads are defined. The 1st thread is set up to handle sending of messages, while the 2nd thread handles receiving of messages.
    - **RSA:** The RSA module enables secure communication between client and server (chat participants), by generating a pair of public and private keys. In the context of CipherChat™, a brief breakdown on RSA encryption is described below:
        - **Key Generation:** Client and server generates a pair of RSA keys (public key + private key). Public key is shared, while private key is kept confidential.
        - **Sending Messages:** When a user sends a message, a public key is extracted from the recipient. This message is converted to bytes and encrypted using the recipient's public key, before the encrypted message is sent to the recipient.
        - **Receiving Messages:** The recipient receives the encrypted message from the sender. The recipient uses their own private key to decrypt the message, before printing it out to the user.
- **Lines 10 to 17:** Both users (client and server) are prompted to enter each others' chatroom names and state their intent – to host a new chatroom or to connect to an existing chatroom.

## 4.2 Set Up RSA Encryption

```
18  #2. SET UP RSA ENCRYPTION:
19  pub_key, priv_key=rsa.newkeys(1024) #generate 1024-bits pair of public key + private key, for the session
20  pub_keystore = None #created to store the public key of the chat partner, initialized with no value
21
22
23
```

- **Lines 18 to 19:** RSA encryption is set up, by using the method 'rsa.newkeys(1024)' to generate a new 1024-bit key strength pair of public key 'pub_key' and private key 'priv_key'.
- **Line 20:** An empty variable 'pub_keystore' is created, with the purpose to store the public key of the chat partner. When a chat participant initiates (server) or joins (client) the chatroom, their public key is sent to the other party. This public key will be stored in 'pub_keystore' for later use, when encrypting messages before sending them out.

## 4.3 Set Up Hosting VS Connecting Logic

```
24  #3. SET UP HOSTING VS CONNECTING LOGIC:
25  if userinp == "1": #if user chooses to host a chatroom...
26      print("You will be HOSTING a new CipherChat™ room.")
27      serverIPinp=input("Enter your machine's IP Address:") #prompt and save user's desired IPv4 address
28      serverPORTinp=int(input("Choose a Port Number on your machine to listen:")) #prompt and save user's desired port number
29
30      server=socket.socket(socket.AF_INET, socket.SOCK_STREAM) #configure a new socket for server using user-specified IPv4 & TCP
31      server.bind((serverIPinp, serverPORTinp)) #BIND the server's socket to the specified IPv4 address and port number
32
33      server.listen() #server starts listening for incoming connections
34      client, _ = server.accept() #when client makes a connection attempt, accept the client
35
36      #HOSTING client SENDS public key first, then RECEIVES the public key of the connected client:
37      client.send(pub_key.save_pkcs1("PEM"))
38      pub_keystore = rsa.PublicKey.load_pkcs1(client.recv(1024))
39
40
41  elif userinp == "2": #else if user chooses to join a chatroom...
42      print("You will be CONNECTING to an existing CipherChat™ room.")
43      serverIPinp=input("Enter the server's IP Address:") #prompt the user for the server's IPv4 address
44      serverPORTinp=int(input("Enter the server's Port Number:")) #prompt the user for the server's port number
45
46      client=socket.socket(socket.AF_INET, socket.SOCK_STREAM) #configure a new socket for client using user-specified IPv4 & TCP
47      client.connect((serverIPinp, serverPORTinp)) #CONNECT the client to the server's IPv4 address and port number
48
49      #CONNECTING client RECEIVES public key first, then SENDS its own public key:
50      pub_keystore = rsa.PublicKey.load_pkcs1(client.recv(1024))
51      client.send(pub_key.save_pkcs1("PEM"))
52
53
54  else: #if user keys in an invalid input for IP address or port number, alert user and exit the script
55      print("Invalid input, exiting CipherChat™.")
56      exit()
57
58
59
```

- **Lines 24 to 28:** If, user chooses to host a new chatroom, prompt the user to enter the server's IP address and port number to listen from. This machine is the SERVER:
    - **Lines 30 to 31:** Configure a new socket using the method 'socket.socket(socket.AF_INET, socket.SOCK_STREAM)', for the server using the IP address and port number provided. BIND the server's socket using the method '.bind()'.

- **Lines 33 to 34:** Using the '.listen()' method, allow the server to start listening for incoming connections. When the client makes a connection attempt, use the method '.accept()' to accept and grant access to the client.
- **Lines 36 to 38: Client send its own public key, then receives server's public key:**
  - Sending Public Key using 'client.send()':
    The client sends its own public key to the server, which is converted to a string in 'PEM' format. This 'PEM' format is commonly used to encode cryptographic objects such as public keys.
  - Receiving Public Key using 'rsa.PublicKey.load_pkcs1(client.recv(1024))':
    The client receives the server's public key. The 'load_pkcs1' method loads and parses the received message into an RSA public key object, and then stored into new variable 'pub_keystore' to store the server's public key.

- **Lines 41 to 44:** Else if, user chooses to join an existing chatroom, prompt the user to enter the server's IP address and port number to connect to. This machine is the CLIENT:
  - **Lines 46 to 47:** Configure a new socket using the method 'socket.socket(socket.AF_INET, socket.SOCK_STREAM)', for the client using the IP address and port number provided. CONNECT the client to the server's IP address and port number, using '.connect()'.
  - **Lines 49 to 51: Server receives client's public key, then sends its own public key:**
    - Receiving Public Key using 'rsa.PublicKey.load_pkcs1(client.recv(1024))':
      The server first receives the client's public key. The 'load.pkcs1' method loads and parses the received message into an RSA public key object, and then stored into variable 'pub_keystore' to store the client's public key.
    - Sending Public Key using 'client.send():
      The server sends its own public key to the client, which is converted to a string in 'PEM' format.
- **Lines 54 to 56:** Or else, user keyed in an invalid input. Alert user and exit the script.

## 4.4 Set Up Sending & Receiving Messages

```
60  #4. SENDING & RECEIVING MESSAGES
61  def sendmessage(clnt): #function is defined for sending messages
62      while True:
63          msg=input('') #prompt the user to input their message, save to 'msg' variable
64          encryptedmsg=rsa.encrypt(msg.encode(), pub_keystore) #encrypt the 'msg' using the other user's public key
65          clnt.send(encryptedmsg) #send the encrypted message to the other user
66          print(f"{yourname}(YOU): {msg}") #display the message
67
68  def receivemessage(clnt): #function is defined for receiving messages
69      while True:
70          #decrypt message using private key, print the result to the user
71          print(peername + ': ' + rsa.decrypt(clnt.recv(1024), priv_key).decode())
72
73
74
```

- **Lines 61 to 66: Sending messages:**
    - **Lines 62 to 64:** A function is defined to handle the process of sending a message between the client and server. Using a 'while True' loop, loop runs indefinitely and waits for user inputs which is saved to the variable 'msg'. The 'msg' is then encoded '.encode()' from string to bytes, before getting encrypted with the public key 'pub_key', using 'rsa.encrypt()'.
    - **Lines 65 to 66:** The encrypted message is sent using '.send(encryptedmsg)'. Upon receiving, the message string is printed.
- **Lines 68 to 71: Receiving messages:**
    - **Lines 68 to 69:** A function is defined to handle the process of receiving a message between the client and server. Using a 'while True' loop, loop runs indefinitely and waits for a message.
    - **Lines 70 to 71:** The encrypted message is received using '.receive(1024)' where '1024' specifies the maximum bytes allowed to be received at once. Next, using 'rsa.decrypt()', the encrypted message is decrypted using the private key 'priv_key', before being decoded '.decode()' from bytes to string. The message string is printed.

## 4.5 Set Up Multi-Threading

```
75  #5. MULTI-THREADING
76  # To better manage inbound (receiving messages) & outbound (sending messages) tasks...
77  # ... where sending and receiving messages can run concurrently...
78  # ... thereby improving chat responsiveness
79
80  #set up 1st thread for sending messages, using 'sendmessage()' function
81  #pass the client socket as argument to the 'sendmessage()' function
82  threading.Thread(target=sendmessage, args=(client,)).start()
83
84  #set up 2nd thread for receiving messages, using 'receivemessage()' function
85  #pass the client socket as argument to the 'receivemessage()' function
86  threading.Thread(target=receivemessage, args=(client,)).start()
```

- **Lines 75 to 78:** Multi-threading is set up to better manage I/O tasks, and allow seamless concurrent receiving and sending of messages.
- **Lines 80 to 82:** Using 'threading.Thread()', the 1st thread is set up for sending messages, where the 'sendmessage()' function defned earlier is set as the 'target=' and the socket defined earlier is passed as an argument.
- **Lines 84 to 86:** Using 'threading.Thread()', the 2nd thread is set up for receiving messages, where the 'receivemessage()' function defined earlier is set as the 'target=' and the socket defined earlier is passed as an argument.

## Conclusion

The completion of these 3 distinct Python projects has provided me with new scripting concepts and technical insights.

The Linux O.S. Information Generator project has allowed me to better understand system-level commands and interactions, to extract system information from a Linux machine. I have also learned to utilize various Python modules to retrieve system information. Key modules include:

- Platform: Used to gather information on the O.S. itself.
- Socket: Used to retrieve IP addresses.
- Psutil: Used to retrieve real-time information on system resources, such as disk and CPU usage.
- Subprocess: A very useful tool that allow Terminal commands to be executed directly.

The PortSpy™ (multi-threaded open port scanner) project has exposed me to new scripting concepts such as the use of Socket module to make attempts to reach specific IPv4 address and port numbers, and detect open ports. I have also learned to utilize the queue module, which uses FIFO to manage a large range of port numbers and ensure that each and every specified port number finishes scanning, in a sequential manner. I have also learned how to better handle errors by incorporating try-except blocks to alert the script user on any failed connections, while not disrupting the overall scanning process. Lastly, I experimented with the concept of multi-threading to speed up the port scanning process – where the scanning process per port is assigned to each single thread, allowing multiple ports to be scanned concurrently.

Finally, the CipherChat™ ( ) project has exposed me to the entire process of asymmetric encryption in a client-server model, utilizing the RSA module to generate public and private keys for encryption and decryption respectively. I have also learned to script the conversion between string and bytes, using formats like 'PEM' to encode and denode, prior to encryption and decryption. Finally, I have learned to marry infinite 'while True' loops with multi-threading, which creates real-time concurrent sending and receiving of messages, thereby mitigating blockages and improving chat responsiveness.

Moving forward, the Python scripting and technical concepts gained from these projects (establishing connections with sockets, utilizing secure encryption in a client-server model, running multi-threads, implementing 'try-except' error-handling, adopting REGEX to format prints) serve as proper foundation for tackling more advanced cybersecurity challenges.

## References (Linux O.S. Information Generator)


Abstract API. (2023). How to Find Your IP Address Using Python. Retrieved from https://www.abstractapi.com/guides/get-ip-address-python (Accessed: January 13, 2023).

AskPython. (2021). Python platform Module: Quick Introduction. Retrieved from https://www.askpython.com/python-modules/python-platform-module (Accessed: January 12, 2023).

Baeldung. (2022). Understanding and Configuring Linux Network Interfaces. Retrieved from https://www.baeldung.com/linux/network-interface-configure (Accessed: January 13, 2023).

CopyProgramming. (2023). How to calculate a directory size using Python. Retrieved from https://copyprogramming.com/howto/how-to-calculate-a-directory-size-using-python (Accessed: January 13, 2023).

DataCamp. (2023). An Introduction to Python Subprocess: Basics and Examples. Retrieved from https://www.datacamp.com/tutorial/python-subprocess (Accessed: January 15, 2023).

Dataquest. (2023). Python Subprocess: The Simple Beginner's Tutorial (2023). Retrieved from https://www.dataquest.io/blog/python-subprocess/ (Accessed: January 15, 2023).

GeeksforGeeks. (2022). How to get current CPU and RAM usage in Python? Retrieved from https://www.geeksforgeeks.org/how-to-get-current-cpu-and-ram-usage-in-python/ (Accessed: January 16, 2023).

Real Python. (2018). Python Modules and Packages: An Introduction. Retrieved from https://realpython.com/python-modules-packages/ (Accessed: January 12, 2023).

StackOverflow. (2010). Getting a machine's external IP address with Python. Retrieved from https://stackoverflow.com/questions/2311510/getting-a-machines-external-ip-address-with-python (Accessed: January 14, 2023).

Tutorial Example. (2020). Python Get Default Gateway IP Using Netifaces: A Step Guide – Python Tutorial. Retrieved from https://www.tutorialexample.com/python-get-default-gateway-ip-using-netifaces-a-step-guide-python-tutorial/ (Accessed: January 14, 2023).

YouTube. (2021). Finding Sizes of Folders/Directories With Python. https://www.youtube.com/watch?v=LWC3D4liKy0 (Accessed: January 13, 2023).

YouTube. (2018). Python Tutorial: re Module - How to Write and Match Regular Expressions (Regex) https://www.youtube.com/watch?v=K8L6KVGG-7o (Accessed: January 17, 2023).

## References (PortSpy™)

Real Python. (2018). How to deal with Python's GIL? Retrieved from https://realpython.com/python-gil/#how-to-deal-with-pythons-gil (Accessed: January 15, 2023).

Real Python. (2018). What Is the Python Global Interpreter Lock (GIL)?Retrieved from https://realpython.com/python-gil/ (Accessed: January 15, 2023)

Simplilearn. (2023). Queue in Python: Working With Queue Data Structure in Python. Retrieved from https://www.simplilearn.com/tutorials/python-tutorial/queue-in-python (Accessed: January 17, 2023).

TechTarget. (n.d.). What is multithreading? Retrieved from https://www.techtarget.com/whatis/definition/multithreading (Accessed: January 15, 2023).

YouTube. (2021). Network Programming with Python Course https://www.youtube.com/watch?v=FGdiSJakIS4&list=PLQIXaoKWER84rsnx72tYwKDyz_isF69RR&index=2 (Accessed: January 13, 2023).

## References (CipherChat™)


Encryption Consulting. (n.d.) What is RSA? How does an RSA work? Retrieved from https://www.encryptionconsulting.com/education-center/what-is-rsa/ (Accessed: January 18, 2023).

StackOverflow. (2012). RSA encryption and decryption in Python. Retrieved from https://stackoverflow.com/questions/30056762/rsa-encryption-and-decryption-in-python (Accessed: January 18, 2023).

YouTube. (2023). Coding Encrypted Chat in Python https://www.youtube.com/watch?v=U_Q1vqaJi34 (Accessed: January 19, 2023).

YouTube. (2022). Learn Network Programming in Python https://www.youtube.com/watch?v=SimNmTciQTs (Accessed: January 18, 2023).

YouTube. (2021). Network Programming with Python Course https://www.youtube.com/watch?v=FGdiSJakIS4&list=PLQIXaoKWER84rsnx72tYwKDyz_isF69RR&index=2 (Accessed: January 13, 2023).

YouTube. (2021). Simple TCP Chat Room in Python https://www.youtube.com/watch?v=3UOyky9sEQY (Accessed: January 21, 2023).

YouTube. (2021). Advanced TCP Chat Room in Python https://www.youtube.com/watch?v=F_JDA96AdEI (Accessed: January 21, 2023).

END.