PORT SCANNER – PYTHON

This simple TCP port scanner was built with Python, using the IDE Pycharm for educational purposes and to practice my knowledge on cybersecurity and Python programming.

Additionally, there will be comments in the code explaining socket parameters.

It reports on which ports are open or closed on the specified IP Address.

Modules used:

`Import socket` – Used for TCP/UDP connections and needed for port scanning.

`Import sys` – Used to exit script safely if there are errors or interruptions.

`Import os` – Used to interact with the operating system, in this case clear the terminal screen on my running O.S.

`From datetime import datetime` – Used to record start and end times of scan.

Code Breakdown:

I. Clear Screen

```
def clear_screen():
```

- Clears any text from terminal for a clean output.

```
if os.name == 'nt':
os.system('cls')
```

- ‘nt’ determines if the OS running is Windows and if yes ‘cls’ clears the screen.

```
elif os.getenv('TERM'):
os.system('clear')
```

- ‘TERM’ variable is for macOS/Linux terminals (my running OS was macOS for this project). ‘Clear’ command runs if the OS running is either of the two.

```
clear_screen()
```

- Function defined to clear the screen before printing the prompt.

II. Execution

```
remoteServer = input("Enter a remote host
to scan: ")
```

- Prompts the user to type in the desired IP Address to be scanned and is stored under the variable ‘remoteServer’.

```
try:
```

- ‘try’ block used to capture bad hostnames.

```
remoteServerIP =
socket.gethostbyname(remoteServer)
```

- Converts any hostname to a readable IPv4 address.

```
print("_" * 60)
print("Please wait, scanning remote
host", remoteServerIP)
print("_" * 60)
```

- Underscores for formatting and tells the user what address is being scanned and another 60 underscores for separation.

```
t1 = datetime.now()
```

- Records initial time of scan.

```
for port in range(1, 11):
```

- Define number of ports that want to be scanned, 11 was chosen for example purposes.

```
sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
```

- Creates a TCP socket for IPv4 to try and connect to the port.

```
sock.settimeout(0.5)
```

- Timeout set to 0.5 seconds to prevent hanging if port is filtered or slow to respond.

```
result =
sock.connect_ex((remoteServerIP, port))
```

- Attempts to connect to target IP on this port and returns 0 if connection is established otherwise returns error code if filtered.

```
if result == 0:
print("Port {} : Open".format(port))
else:
print("Port {} : Closed".format(port))
```

- Prints ports that are open or closed regardless.

```
sock.close()
```

- Closes socket and frees up resources for the next iteration.

III. Exceptions

```
except KeyboardInterrupt:
print("\nYou pressed Ctrl+C")
sys.exit()
```

- Catches interruption and exits the script in a clean manner.

```
except socket.gaierror:
print("\nHostname could not be resolved.
Exiting...")
sys.exit()
```

- Catches bad hostname errors that DNS cannot resolve and exits.

```
except socket.error:
    print("\nCouldn't connect to server")
    sys.exit()
```

- Catches other socket errors.

IV. Conclusion

```
t2 = datetime.now()
```

- Records end time of scan.

```
total = t2 - t1
print("Scanning completed in:", total)
```

- Computes duration of the process.