**PORT SCANNER**

Oggi parleremo di come i Security Griffin hanno configurato tramite python il portscanner al meglio per farlo funzionare correttamente tramite GUI (Interfaccia Grafica)

Per prima cosa si è andati a cercare su fonti esterne come effettivamente funzionasse un portscanner… le principali fonti sono state: YouTube, Python Docs e Forum vari.. poi sempre dalle stesse fonti si è andati a capire come creare una struttura di codice ottimale per il funzionamento del Port Scanner.. ora infatti andiamo a scoprire proprio come è fatta questa struttura..

**LIBRERIE**

In questa sezione andremo a spiegare come sono state importate le librerie, qual è la loro funzione e come sono state utilizzate nel contesto del port scanning..

**import socket** La libreria Socket ci permette di creare e gestire connessioni di rete inviando e ricevendo dati. In questo caso la libreria è stato utilizzato per permetterci la connessione alle porte e poter verificare quale porta fosse aperta o chiusa.

**import threading** La libreria threading consente (come ben possiamo capire dal nome) l’esecuzione di più thread in contemporanea. In questo caso se nel codice del port scanning non ci fosse la libreria threading il sistema invierebbe un solo pacchetto a volta invece di inviarne molteplici in contemporanea aumentando il tempo di attesa per individuare una porta

**import tkinter** La libreria tkinter ci permette di creare interfacce grafiche (GUI) su python. Una libreria con tantissime funzioni che ci permette quasi la completa flessibilità

**import os** Come possiamo ben capire dal nome OS è una libreria che ci permette di interagire con il sistema operativo. Ci sarebbe moltissimo da dire su questa libreria ma nel nostro caso l’abbiamo utilizzata per sistemare i path delle cartelle

**import queue** La libreria queue permette di salvare i thread (thread-safe). Questo ci permette di gestire e condividere i dati tra i diversi thread. Grazie alla libreria queue siamo riusciti ad avere un output sulla GUI delle porte aperte.

**import subprocess** La libreria subprocess ci permette di creare nuovi processi e chiudere quelli vecchi. In questo progetto la libreria subprocess è stata utilizzata principalmente per aprire nuove pagine appena premuto il button

**import PIL** La libreria PIL ci permette di avere più controllo sulle immagini, ma nel caso di questo progetto è stata utilizzata per consentire il caricamento di un formato .png

Ora che abbiamo parlato delle librerie utilizzate all’interno del progetto possiamo cominciare ad analizzare il codice vero e proprio del portscanner…

**Portscanner.py**

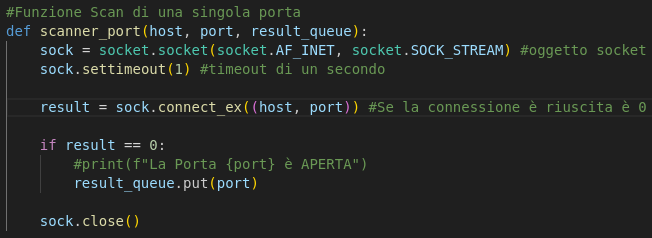
**import librerie**

****

La prima fase del codice riguarda l’importazione delle librerie socket e threading.

*“A pagina n* ***2*** *si possono trovare tutte le info riguardanti le librerie”*

**creazione socket**



Questa è la fase dove andiamo a creare una connessione socket a singola porta… tralasciamo i commenti e concentriamoci sul codice..

**codice**

1. def scanner\_port(host, port, result\_queue):
2. sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)
3. sock.settimeout(1)
4. result = sock.connect\_ex((host, port))
5. if result == 0:

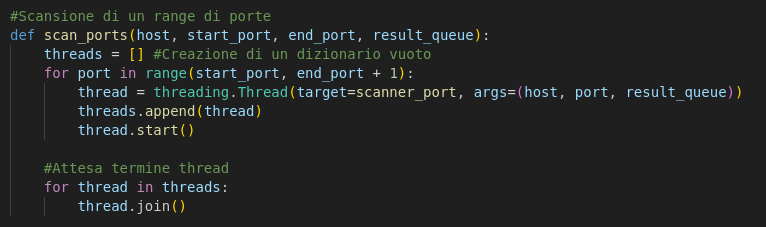
result\_queue.put(port)

1. sock.close()

**spiegazione**

1. Stiamo definendo la funzione con il nome “scanner\_port” e li stiamo assegnando i seguenti parametri “host, port, result\_queue”
2. Stiamo creando un oggetto socket che appartiene alla famiglia degli indirzzi IPv4 (AF\_INET) e il tipo di socket ovvero TCP (SOCK\_STREAM)
3. Impostiamo il timeout a un secondo, se non il dispositivo non si connette al server entro 1 secondo, la connessione è fallita
4. Grazie a “connect\_ex” possiamo stabilire una connessione con l’indirizzo IP e la porta. Se la connessione avrà successo ci restituirà 0 in caso contrario il numero sarà diverso
5. Se il risultato è uguale a 0 (connessione riuscita) la porta è considerata aperta e il risultato viene messo nella result queue
6. Il socket viene chiuso e libera tutte le risorse

**creazione thread**

****

In questa fase andiamo a creare i thread che ci permetteranno di poter eseguire più richieste in contemporanea

**codice**

1. def scan\_ports(host, start\_port, end\_port, result\_queue):
2. threads = [ ]
3. for port in range(start\_port, end\_port + 1):
4. thread = threading.Thread(target=scanner\_port, args=(host, port, result\_queue))
5. threads.append(thread)
6. thread.start()
7. for thread in threads
8. thread.join()

**spiegazione**

1. Definiamo la funzione e assegnamo 4 parametri
2. Creazione di una lista vuota per memorizzare i thread durante l’esecuzione
3. Creazione di un ciclo che scorre su tutte le porte (ultima porta inclusa)
4. Creazione di un nuovo thread usando il modulo threading. Il target è lo scanner\_port che abbiamo visto prima e gli argomenti ovviamente sono i parametri
5. tramite il comando .append stiamo dicendo di aggiungere il thread appena creato alla lista
6. con thread start sta a significare che ogni thread sarà separato e il ciclo ricomincerà di nuovo
7. Questo fa entrare la funzione in un’altro ciclo
8. Con .join stiamo a indicare che il thread corrente completi la sua esecuzione prima di passare all’esecuzione di un’altro thread

**Risultati**

Così come mostrato lo script ovviamente è privo di ogni GUI e sarà eseguibile solo da linea di comando tramite il comando “python portscanner.py”. Inserendo il range di porte (porta iniziale e porta finale) e l’IP target sarà possibile vedere tutte le porte aperte. Importante è ricordare che alla fine del codice bisogna specificare tramite ciclo while la result queue, solo così lo script darà il risultato voluto. Facciamo un esempio..

**Esempio**:

while not result\_queue.empty():

print(f”Porta {result\_queue.get()} è aperta”)