Port Scanner i Python

Formål: At kunne finde åbne TCP-porte i Python

Dette kan gøres som enten White-Hat angreb eller et Black-Hat angreb. Hvad er forskellen?
Du skal fokusere på at finde åbne TCP-porte og ikke UDP–porte. Hvorfor er det svært at scanne UDP-porte?

Opgave 1.1: Lav et nyt Python - projekt

Lav et nyt python-projekt 'PortScanner'.

Opgave 1.2: Simple Port Scanner

Note: Hvis du ønsker at scanne andet end din egen PC skal du være på MGV-xxxx netværket.

I main.py

- Lave en variabel med den adresse du ønsker at undersøge for åbne TCP porte (kan være 127.0.0.1 til at starte med – senere måske din nabos maskines IP-adresse)
- Du skal lave et loop der gennemløber alle mulige porte (hvor mange er det?)
- For hver port skal du prøve at oprette en socket (TCP) forbindelse til den pågældende port. Hvis der IKKE er en åben port kastes en exception, ellers er der en åben port (evt. gemmes i en liste).
- Efter loopet udskrives alle åbne porte

Hint: Brug W3Schools til at finde syntaksen for try-except I python.

Opgave 1.3: Refactorer ved at benytte Threads

Hastigheden for din scanner er sikkert ikke imponerende. Hvorfor er den langsom?

En måde at gøre det hurtigere på er ved at benytte Threads. Du kan starte med at hver Thread prøver hver sin port. – Bliver det hurtigere?
Måske kunne hver Thread tage sig af en lille gruppe af port-numre?
Hvordan vil du udskrive fundne åbne porte?

Ekstra A: Kan det gøres endnu hurtigere?

Hvis du skal finde åbne tcp-porte på din egen maskine kan det gøres endnu hurtigere – hvordan?

Opgave 1.4: Hvem finder den åbne port

I en lille gruppe (fx: dit bord) – Husk at være på MGV2-DMUXXX nettet.
Definer én computer til at være ”offer”. Offeret starter (i SocketTest) en TCP server på et ukendt portnummer. De andre skal så finde frem til den åbne port.