1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

Corso di
Reti di calcolatori

Anno accademico
2020-2021

Progetto realizzato da
Barzotti Cristian

Matricola
290725

Corso tenuto dal professore
Della Selva Antonio

Port Scanner Multithread: risorsa o pericolo?

Indice

Capitolo 1

Introduzione

Questo progetto è stato realizzato per soli scopi educativi. Ci tengo a precisare che, nonostante il port scanning sia una tecnica antica quasi quanto lo stesso internet, ancora oggi è molto utilizzata per scoprire falle nei vari sistemi. In quanto strumento, il port scanning in sé non è né buono né cattivo.
Un port scanner fornisce informazioni sullo stato delle porte di un host. Queste informazioni possono essere utilizzate sia a scopo benevolo che a scopo malevolo. Essendo il progetto solo a scopo educativo, mi sono limitato a effettuare scan delle porte e a "buttare via" le informazioni ricavate.

1.1 Obiettivi

I principali obiettivi del progetto, al quale mi riferirò come PSM da ora in avanti, sono l'approfondimento della mia conoscenza riguardante gli attacchi di rete più comuni (basti pensare che anche un port scanner può essere portato a livelli tali da consentire attacchi di tipo DoS) e l'implementazione tramite Python di uno di essi.

1.2 Scelte architetturali

1.2.1 Perché Python?

Python è un linguaggio multi-paradigma dinamico, semplice e flessibile che ha un numero sempre maggiore di utilizzatori. Secondo varie fonti, al momento Python è il secondo linguaggio più utilizzato al mondo, subito dopo C.
Con queste premesse, la scelta di Python è stata fatta sia come scelta ponderata, essendo Python utilizzatissimo per lo sviluppo di progetti simili (cosa che ha portato anche dei problemi, dei quali parlerò in seguito), sia come scelta personale. Era mia intenzione, infatti, approcciarmi all'utilizzo di Python tramite questo progetto.
Il progetto è scritto interamente in Python 3.9.5 (l'ultimo disponibile al momento dello sviluppo). Dovrebbe essere retrocompatibile con un qualsiasi Python 3.x, ma è stato provato solo fino Python 3.8.

1.2.2 Perchè un Port Scanner?

Con port scanning si intende una tecnica progettata per sondare un server o un host per capire quali porte siano in ascolto sulla macchina. Questa tecnica viene utilizzata, come già spiegato, per diversi motivi. Da una parte un amministratore di rete può utilizzare un port scan per cercare eventuali falle nel suo sistema, dall'altra un hacker può identificare i servizi disponibili su un host e sfruttarne eventuali vulnerabilità. Con il passare del tempo la tecnica del port scanning è stata associata con intenti maligni, per cui ci tengo a precisare nuovamente che lo scopo del progetto non ha secondi fini. È stato scelto un port scanner perchè è uno dei primi programmi che solitamente vengono studiati da chi vuole diventare un "hacker".

1.2.3 TCP vs UDP

Ai fini del mio progetto ho scelto di utilizzare il protocollo **TCP**.
Il protocollo TCP (Transmission Control Protocol) è un protocollo di rete di livello 4 nello stack ISO/OSI. È un protocollo orientato alla connessione, per cui i due host prima di comunicare dovranno instaurare una connessione tramite un *three-way handshake*. È proprio questa meccanica del protocollo TCP a renderlo adatto per il mio progetto.
Al contrario, il protocollo UDP è un protocollo di rete, anche lui di livello 4 nello stack ISO/OSI, ma non orientato alla connessione. Questo rende il protocollo UDP particolarmente inadatto per l'implementazione di un port scanner, anche se non impossibile. Essendo UDP un protocollo incentrato sui datagrammi, non richiede una connessione di tipo end-to-end, rendendolo più difficile da gestire.

1.2.4 SYN scan

Come già accennato, PSM utilizza una caratteristica del protocollo TCP per effettaure lo scan. Questa particolare tipologia di scan viene comunemente chiamata SYN scan, e il suo funzionamento è relativamente semplice.
Si crea un pacchetto TCP e si setta il flag SYN ad 1. Se la porta specificata è aperta risponderà con un pacchetto TCP con flag SYN e ACK attivi, iniziando cosi il *three-way handshake*. A questo punto si invia un pacchetto con il flag RST impostato a 1, per terminare la connessione. Sappiamo infatti che la porta è aperta, e non ci interessa continuare la connessione.
Se la porta è chiusa, l'host risponderà con un pacchetto TCP con flag RST attivo. Anche in questo caso la nostra scansione ci ha portato informazioni, e non si prosegue oltre.

1.2.5 SYN scan: problematiche

Un problema del SYN scan è la quasi impossibilità nel distinguere tra porte filtrate e/o una "sonda" caduta nel vuoto per vari motivi. Per ovviare almeno in parte al problema, e dopo aver cercato maggiori informazioni e testato con tool diversi l'accuratezza del mio codice, ho deciso di leggere il pacchetto di risposta nei seguenti modi:

- Risposta nulla: pacchetto filtrato e/o mai arrivato.
  Un pacchetto potrebbe perdersi e non arrivare mai a destinazione, o altri problemi di rete potrebbero occorrere durante l'invio. Un pacchetto di questo tipo non darà nessuna risposta e la porta target verrà categorizzata come **No answer/filtered**. PSM non può distinguere tra una porta filtrata e una porta che non ha dato risposta. Esiste la possibilità di ritentare lo scan sulle porte filtrate, ma questo si limita ad eliminare errori dovuti alla connessione.

- Risposta con protocollo ICMP.
  Una risposta con il protocollo ICMP è ideale per quanto riguarda l'identificazione di porte bloccate da firewall. ICMP è un protocollo di servizio, incapsulato direttamente nel protocollo IP (livello 3 dello stack TCP/IP). In base al tipo e al codice del pacchetto ICMP, si posssono riconoscere con certezza le porte bloccate da firewall. Questo tipo di risposta, assieme alle risposte con flag SYN-ACK o RST, sono ideali perchè danno la certezza dello stato della porta, o dell'irraggiungibilità di essa.

Parte I

Realizzazione del progetto

Capitolo 2

Struttura

PSM è un port scanner implementato cercando di rispettare il paradigma della programmazzione ad oggetti. Il file sorgente è composto da:

- Port Scanner.py: file Python contenente il "main" del progetto. Si occupa principalmente di gestire l'interfaccia grafica e di instanziare la classe **PortScan**.
- port_scan.py: file contenente la classe PortScan. Qui avviene lo scan vero e proprio della porta.
- utils.py: file contenente codice di supporto.
- common_ports.json: file .json contenente una serie di coppie chiave-valore con le porte e i relativi servizi ad esse associati. Il file è stato scaricato dal sito della IANA[1].

È prevista la creazione di un file eseguibile .exe (utilizzabile su piattaforma Windows) ma, dati alcuni problemi derivanti da Python[2] stesso, non posso garantire il funzionamento su ogni singola macchina. Nel caso in cui il file eseguibile sia funzionante, è **necessario** lanciarlo con privilegi di amministratore. Questo perchè un modulo presente all'interno del programma, del quale parleremo in seguito, richiede tali privilegi per poter funzionare.

[1] Fonte: `https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`

[2] Python è molto utilizzato per creare software malevoli, e molti antivirus (compreso Windows Defender) rilevano gli eseguibili Python come trojan, bloccandoli

Capitolo 3

Funzionamento

Il funzionamento di PSM è relativamente semplice, come già accennato in precedenza; si sfrutta la libreria **scapy** per forgiare dei pacchetti creati "ad hoc". In particolare, si vanno a creare dei pacchetti con il flag SYN impostato ad 1. Una volta inviato, questo pacchetto richiederà una connessione di tipo TCP all'host, nella porta specificata.
Andando, quindi, a leggere il pacchetto di risposta, viene determinato lo stato della porta:

- Nessuna risposta:
  il tipo di risposta "peggiore". Non si può dire con certezza nulla sullo stato della porta quando si ricevono risposte di questo tipo; il pacchetto potrebbe non essere mai arrivato o potrebbe essere stato fatto cadere da un firewall. Le porte con questo tipo di risposte vengono salvate, ed è possibile ritentare uno scan su di esse per escludere problemi di connettività.

- Risposta con flag 0x12:
  questa risposta è un pacchetto SYN-ACK, e ci specifica con chiarezza che la porta è aperta. Avendo scoperto lo stato della porta, non siamo interessati a procedere oltre e viene inviato un pacchetto con flag RST impostato ad 1, per terminare il three-way handshake e la connessione.

- Risposta con flag 0x14:
  questa risposta denomina una porta chiusa. La porta ha ricevuto il nostro pacchetto, ne fa l'acknowledgment e chiude la connessione (inviando i flag ACK e RST). Anche in questo caso abbiamo una risposta precisa sullo stato della porta.

- Risposta con protocollo ICMP:
  il protocollo ICMP[1] è un protocollo che si occupa della trasmissione di informazioni riguardanti malfunzionamenti, informazioni di controllo o messaggi tra host. Nel nostro caso, andando a controllare il tipo del protocollo e il codice contenuto, possiamo definire con certezza che la porta in questione è filtrata.

[1] Fonte: `https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol`

Capitolo 4

Breakdown del progetto

Il progetto si affida a delle librerie Python per il corretto svolgimento. Sono principalmente librerie base di Python che non dovrebbero richiedere una installazione particolare, a parte una chiamata **scapy**. Nella sezione dell'installazione e walkthrough del progetto verrà spiegato come installarle, in caso di necessità.
Con queste premesse, di seguito verranno introdotti brevemente i vari moduli e verrà spiegato come sono stati utilizzati. I moduli in questione sono:

- tkinter;
- multiprocessing;
- socket;
- os;
- json;
- logger;
- scapy.

4.1 tkinter

Tkinter è un modulo standard presente, per quel che sono riuscito a trovare, in tutte le versioni di Python e per tutti i sistemi operativi. Tkinter è un modulo di creazione GUI per Python e, sebbene non sia il migliore, è stato scelto per la relativa semplicità di utilizzo e per evitare all'utente di dover installare altri moduli.
Tkinter non è un modulo perfetto, sia per sua natura sia per mio utilizzo ancora da inesperto; questo significa che potrebbe portare la GUI a bloccarsi ogni tanto, o a funzionare un po' a scatti. Sono a conoscenza di questi problemi, e ho cercato di risolverli al meglio. In generale, è difficile che la GUI "crashi" o si blocchi permanentemente, per cui si consiglia di lasciar lavorare il programma se non si è assolutamente sicuri che sia bloccato.

4.2 multiprocessing

Multiprocessing è, come il nome suggerisce, il modulo che si occupa del multithreading all'interno del mio programma. Nel file **utils.py** viene definita una funzione che prende in ingresso un'altra funzione, un iterabile (nel mio caso, le keys di un dictionary), la lunghezza dell'iterabile e delle coordinate x e y relative alla finestra principale. Lunghezza dell'iterabile e coordinate sono utilizzate semplicemente per la creazione di un popup con una barra di caricamento per visualizzare il lavoro svolto e non sono necessari al funzionamento del multithreading.
Di multiprocessing viene anche sfruttata la funzione **freeze_support()**, per cercare di ridurre il più possibile stalli e/o blocchi dell'interfaccia in primis e del programma in genere.

4.3 socket

Inizialmente il progetto prevedeva l'utilizzo del modulo socket per il controllo delle porte ma, per via di scelte progettuali, questo modulo non era il più conveniente per l'invio e la ricezione di pacchetti. Il modulo è comunque presente all'interno della classe **PortScan** perchè viene utilizzato per fare un controllo dell'host target. Normalmente scapy potrebbe effettuare da solo il controllo, ma ho deciso di utilizzarlo ugualmente per eventuali future estensioni del progetto che non prevedono l'utilizzo del modulo scapy. Il modulo socket viene utilizzato solo per ricavare l'indirizzo IP dal nome di un host. Fondamentalmente, è il "DNS" del mio progetto (anche se, nei fatti, il modulo ricava l'indirizzo IP tramite i DNS preimpostati nella macchina).

4.4 os

Il modulo OS, come ci indica il nome, facilita la comunicazione tra un programma e il sistema operativo presente sulla macchina. Nel mio caso, OS è utilizzato per fare uno scan dei core a disposizione nel dispositivo (in realtà sarebbe più opportuno parlare di *thread*) ed assegnare tanti worker allo scan quanti sono i thread disponibili. Per ora la relazione core (thread) - processi è di 1 a 1, per cui ad ogni core (thread) disponibile nella macchina verrà assegnato uno scan differente.
Nel mio caso, avendo un processore a 8 core e 16 thread, il progetto utilizza 16 workers durante l'esecuzione.

4.5 json

Il modulo json, come intuibile dal nome, viene utilizzato per aprire e gestire un file json presente nel progetto. Come detto in precedenza, questo file contiene una coppia chiavi-valori; ogni chiave è una porta well known o registrata[1][2] e ogni valore è il servizio associato ad essa. L'insieme dei due tipi di porte è stato chiamato **common_ports** e contiene circa 700 coppie porta-servizio.

[1] Fonte: `https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`

[2] Le porte well known vanno da 0 a 1023, quelle registrate da 1024 a 49151

4.6 logger

Logger è un modulo che implementa un log degli eventi di sistema. Viene utilizzato in fase di debug per accertarsi del corretto funzionamento del programma e per eliminare warning fastidiosi durante l'esecuzione del programma. Il warning in questione è derivato dal modulo scapy, che potrebbe riscontrare errori nel caso in cui l'host di destinazione non può ricevere il pacchetto a causa di problemi di rete. Essendo questo problema (legato solo alla connettività) totalmente risolto con l'implementazione del pulsante di "Retry"[3], è sembrato opportuno eliminare questi warning.

4.7 scapy

Scapy è un programma stand-alone interattivo che consente la creazione e decodifica di pacchetti in un ampio numero di protocolli. Essendo scapy vastamente utilizzato per progetti simili a questo, ne è stato effettuato il porting e reso disponibile anche come modulo opzionale per Python.Scapy nasce come programma/modulo per sistemi operativi Unix based, ma il recente port su Windows è stabile e perfettamente funzionante[4]. L'utilizzo di scapy in questo progetto è fondamentale per come è stato progettato, ma scapy può essere utilizzato per fare tanto altro che non verrà esplorato in questo progetto e che richiederebbe mesi di studio per poter capire appieno[5].

[3] Vedere sezione "GUI e walkthrough" per maggiori informazioni
[4] Vedere sezione "Primo avvio" per maggiori informazioni
[5] Tracerouting, probing, attacchi, ping e altro ancora

Capitolo 5

Primo avvio

Come detto nella sezione "Struttura del progetto", è stata prevista la creazione di un file eseguibile per rendere il progetto quasi "Plug and Play" su piattaforme Windows. L'unica cosa necessaria per farlo funzionare è l'installazione di **Npcap**. Idealmente, una volta installato Npcap e estratto il file .zip, non servirà altro che entrare nella cartella **Build** e far partire l'eseguibile. Nel caso in cui il progetto non funzionasse, si consiglia di provare ad avviarlo con privilegi amministrativi.
Durante la fase di testing si sono riscontrati diversi problemi dovuti ad antivirus che rilevano il progetto come "trojan. Nel caso in cui durante le prove questo accada, si consiglia di terminare il programma e cancellare il file etichettato come "trojan" dall'antivirus.
Ci tengo a precisare che sono assolutamente certo del programma e del codice al suo interno, e niente di PSM può nuocere al sistema ospite; detto questo, consiglio di evitare qualsiasi rischio[1].

5.1 Installazione

Nel caso in cui il file eseguibile non sia disponibile, è necessario isntallare Python e il modulo scapy per poter far funzionare il progetto. In questa sezione verrà spiegato come installare per Windows, ma si può fare riferimento alla pagina di installazione[2] per altri sistemi operativi.
Per installare scapy su Windows è necessario:

1. Python 3.4+: si consiglia di installare Python 3.9.5, in quanto il progetto è stato scritto con l'ultima versione disponibile. Una volta installato Python, è necessario aggiungerlo ai PATH di sistema (solitamente viene effettuato in automatico).

2. Npcap: libreria di forgiatura, decodifica e sniffing di pacchetti per Windows sviluppata da Nmap Project[3]. È richiesta l'ultima versione.

3. scapy: ultima versione dal GitHub ufficiale[4]. Una volta scaricato, aprire il terminale in quella cartella e avviare

   ```
   python setup.py install
   ```

[1] Windows è molto suscettibile e non si può prevedere come reagirebbe all'apertura di un file "trojan"
[2] `https://scapy.readthedocs.io/en/latest/installation.html`
[3] `https://nmap.org/npcap/#download`
[4] `https://github.com/secdev/scapy`

Capitolo 6

GUI e walkthrough

In questa sezione verrà spiegata la GUI (Graphical User Interface) del programma e come navigare all'interno di esso.

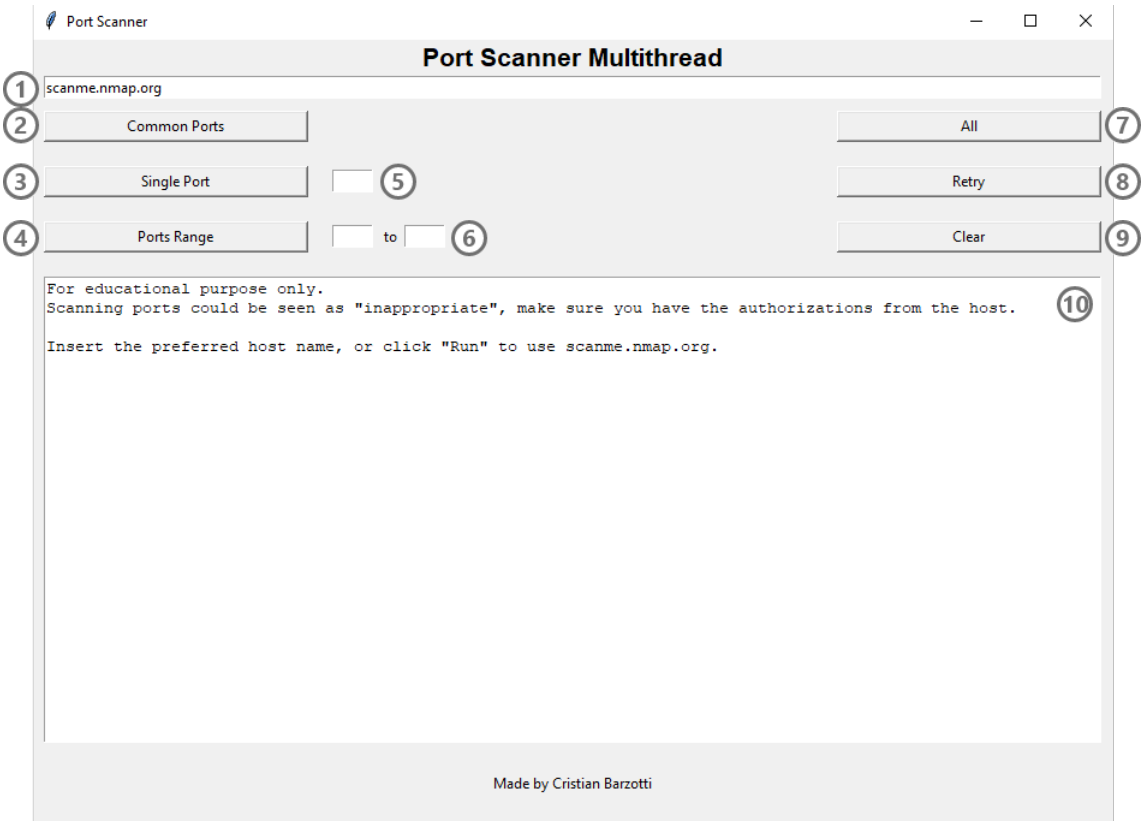
6.1 GUI

Come già spiegato in precedenza, la parte grafica del mio progetto è stata sviluppata sfruttando il modulo Tkinter, modulo standard di Python.
La GUI, in un progetto simile, non è strettamente necessaria ma è stato deciso di implementarla comunque per rendere il progetto più "user friendly"; all'interno della finestra principale ci sono relativamente pochi elementi, ognuno con una funzione ben precisa, che verranno spiegati in seguito.

6.1.1 Bug conosciuti

La GUI non è ancora perfetta; sia per sua natura, sia per mia implementazione sicuramente migliorabile, ogni tanto l'interfaccia potrebbe bloccarsi. Questo difetto si nota soprattutto nella barra di caricamento durante uno scan. Il motivo di tale blocco è presto detto: la barra di caricamento si aggiorna solo dopo aver ricevuto una risposta dallo scan di una porta. Avendo (nel mio caso) fino a 16 threads attivi allo stesso momento, la barra di caricamento viene avanzata alla fine dello scan di 16 porte, portando un leggero delay tra un avanzamento e l'altro. Si è cercato di minimizzare questo problema utilizzando la funziona **freeze_support()** del modulo multiprocessing.
Si consiglia di lasciar lavorare PSM in caso di un "freeze" dell'interfaccia; dai miei test il progetto continua a funzionare anche se la GUI si blocca e, tendenzialmente, tornerà a funzionare a calcoli finiti.

6.1.2 Breakdown della GUI



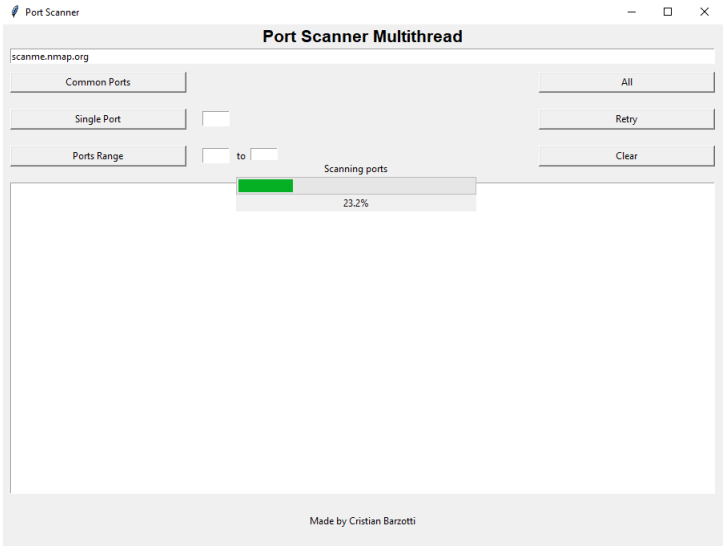
1. Casella di inserimento dell'host name o indirizzo IP. Precompilata con **scanme.nmap.org**
2. Pulsante per fare scan delle common ports (well known + registered ports)
3. Pulsante per fare lo scan di una singola porta. Richiede l'inserimento di una porta nella casella di inserimento associata (5)
4. Pulsante per fare lo scan di un range di porte. Richiede l'inserimento di due porte nelle caselle di inserimento associate (6)
5. Casella di inserimento per una singola porta
6. Caselle di inserimento per un range di porte
7. Pulsante per fare lo scan di TUTTE le porte[1]
8. Pulsante per ritentare lo scan delle porte etichettate come "No answer/filtered", per escludere problemi di connettività
9. Pulsante di pulizia per la finestra di visualizzazione dei risultati
10. Finestra di visualizzazione dei risultati

[1] È sconsigliato utilizzarlo in quanto fare lo scan di 65536 porte richiede molto tempo

6.2 Walkthrough

Una volta avviato il programma, l'utente si troverà di fronte alla finestra principale spiegata in precedenza. Questa è l'unica finestra del programma e dove tutto avviene. Da qui l'utente può inserire il nome o indirizzo IP di un host, oppure utilizzare l'host predefinito **scanme.nmap.org**[2]. Questo host è stato messo a disposizione da Nmap Security Scanner Project[3] e Insecure.Org[4] appositamente per testare scanner come questo progetto.
È possibile fare lo scan delle porte "common" (well known + registered), fare lo scan di una singola porta, di un range di porte o di tutte le porte[5]. Inoltre è possibile ritentare lo scan di porte etichettate come "No answer/filtered" o pulire la finestra di visualizzazione dei risultati.

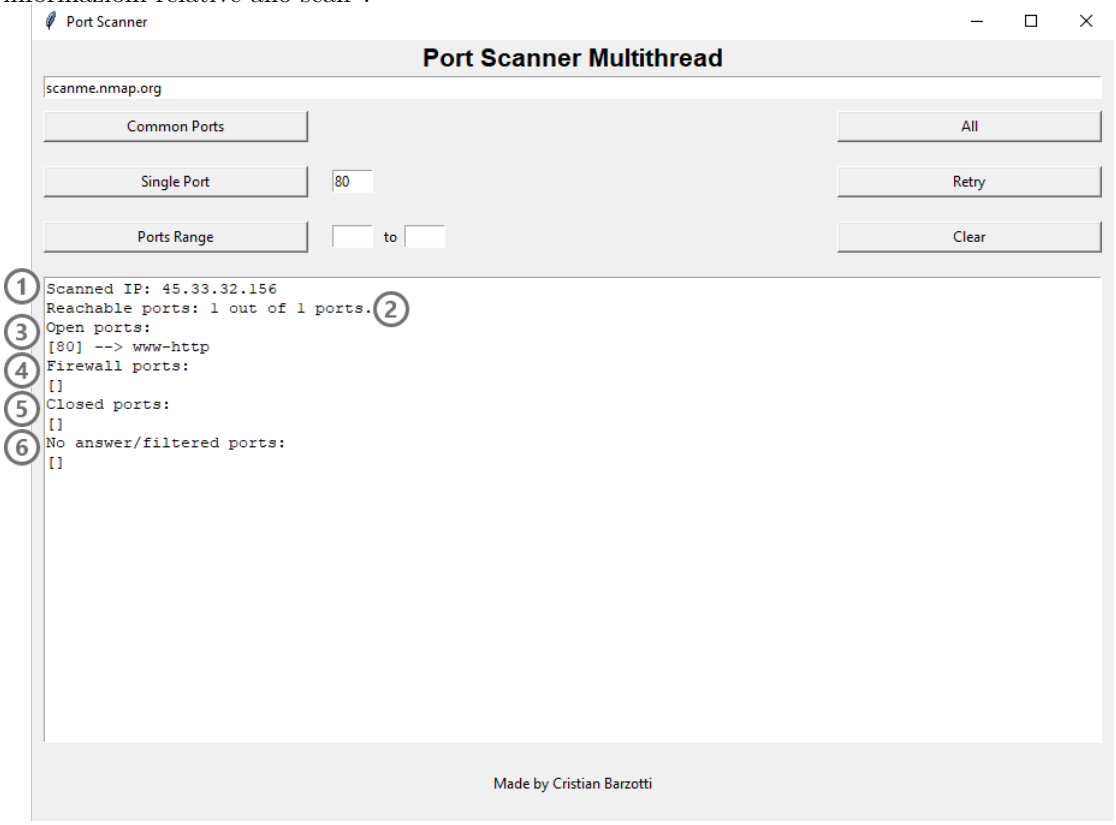
Una volta selezionato un tipo di scan, apparirà un popup di caricamento per mostrare i progressi dello scan.



La barra di caricamento potrebbe bloccarsi per qualche secondo[6], ma si consiglia di lasciar eseguire il programma se non si è assolutamente certi di un malfunzionamento.

[2] `http://scanme.nmap.org/`
[3] `https://nmap.org/`
[4] `https://insecure.org/`
[5] È sconsigliato utilizzarlo in quanto fare lo scan di 65536 porte richiede molto tempo
[6] Guardare capitolo 6.1.1

Una volta completato lo scan, la finestra di visualizzazione dei risultati verrà popolata con le informazioni relative allo scan[7].



1. Indirizzo IP del target
2. Numero di porte raggiungibili[8]
3. Porte aperte con relativo servizio associato
4. Porte filtrate con risposta ICMP
5. Porte chiuse con risposta ACK-RST
6. Porte senza risposta

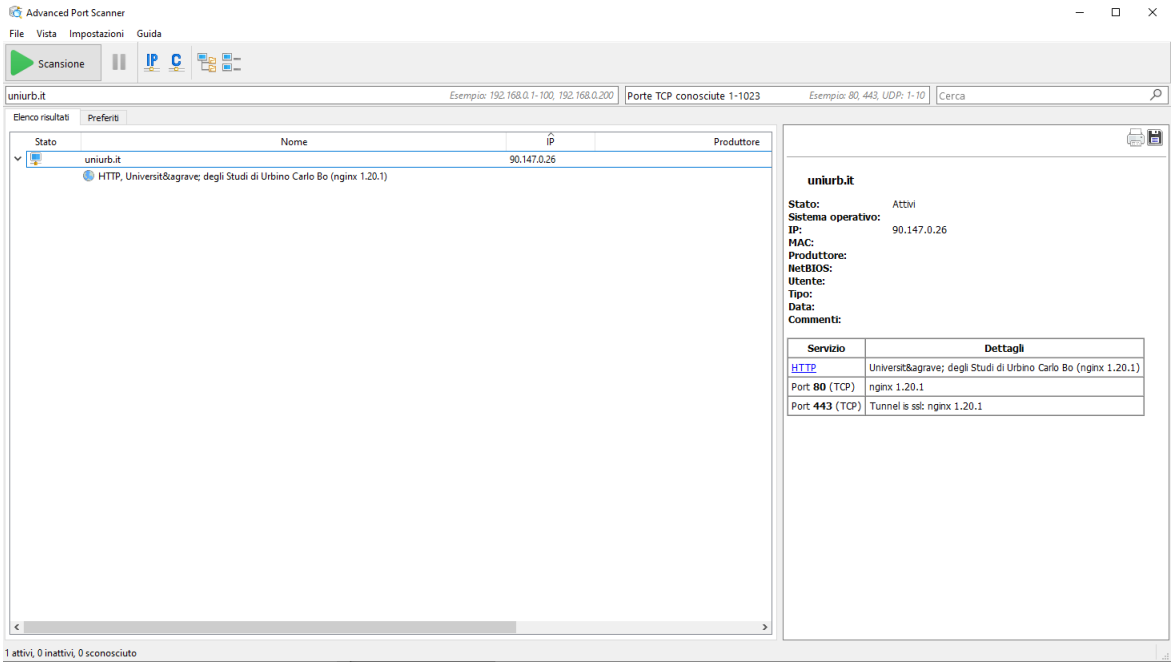
[7] Nell'esempio lo scan di una singola porta
[8] Calcolato come la somma delle porte che hanno dato una risposta precisa

Capitolo 7

Testing e validazione del programma

Per poter testare il programma sono stati confrontati i risultati con quelli di altri scanner (Advanced Port Scanner[1], Online TCP Port Scanner[2]). Di seguito i risultati per uno scan di **uniurb.it**.

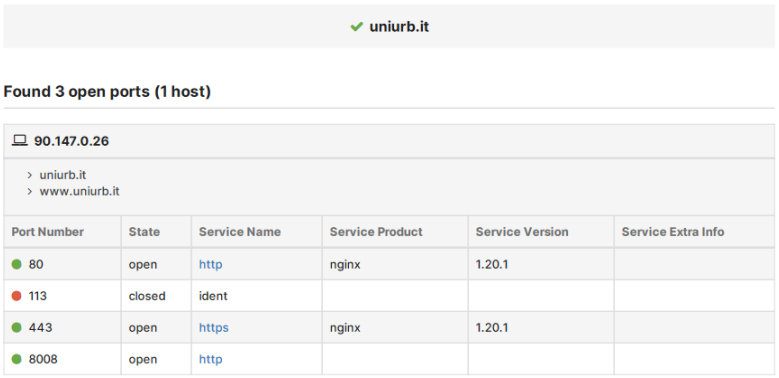
Advanced Port Scanner



[1] https://www.advanced-port-scanner.com/it/

[2] https://pentest-tools.com/network-vulnerability-scanning/tcp-port-scanner-online-nmap

Online TCP Port Scanner



✔ uniurb.it

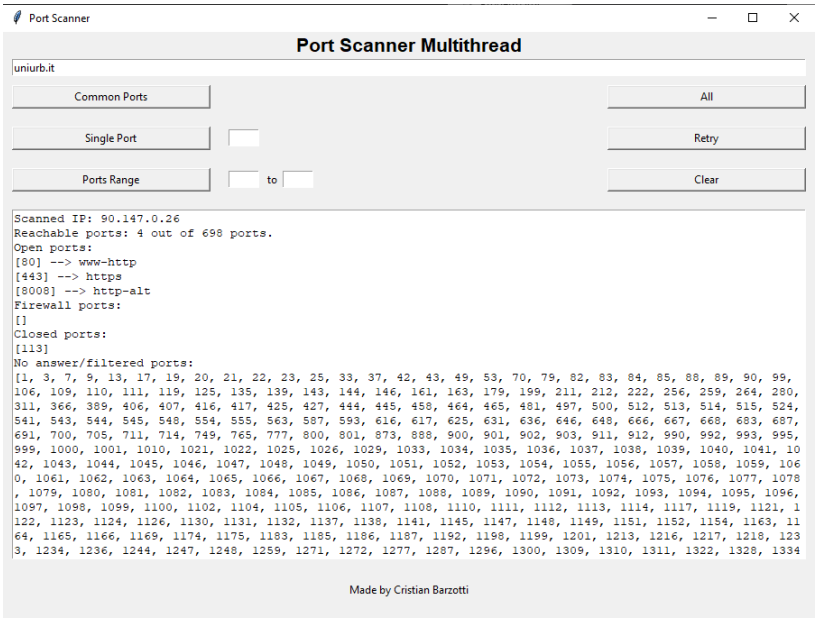
Found 3 open ports (1 host)

90.147.0.26

› uniurb.it
› www.uniurb.it

Port Number	State	Service Name	Service Product	Service Version	Service Extra Info
80	open	http	nginx	1.20.1	
113	closed	ident			
443	open	https	nginx	1.20.1	
8008	open	http			

Port Scanner Multithread



Parte II

Conclusioni

Capitolo 8

Riflessioni e commenti finali

Parto con il dire che lo sviluppo di questo progetto è stato molto interessante e divertente; dallo scoprire metodologie differenti di port scanning ad "attacchi" più avanzati come sniffing, DoS[1], MITM[2] fino alla creazione di malware e ransomware, ritengo che al giorno d'oggi si presti fino poca attenzione alla **sicurezza informatica**.

Ritengo soddisfatti tutti gli obiettivi iniziali di questo progetto. Sono riuscito ad implementare un port scanner in Python e allo stesso tempo a ricercare maggiori informazioni riguardanti eventuali tipologie di attacchi. Ritengo che essere informati in questo campo sia estremamente utile per una persona che, presumibilmente, andrà a lavorare sul settore.

Come abbiamo visto negli ultimi due anni, sempre più persone sono state "costrette" a studiare e/o lavorare da casa; questo ha portato tante aziende, scuole ed università a spostare il loro lavoro parzialmente, o totalmente, online. Riuscire a proteggersi da attacchi di rete è fondamentale in una società della quale Internet sta diventando un caposaldo, sia nel lavoro sia nella vita privata. Basti pensare che una delle tecniche più semplici di attacco, il phishing[3], è ad oggi ancora onnipresente e, relativamente parlando, estremamente efficace.

Tornando al mio progetto, volevo dimostrare che chiunque con un minimo di voglia può prepararsi per effettuare attacchi di rete; ci sono innumerevoli informazioni e tool online gratuiti che sono messi a disposizione degli utenti. Proprio per questo è sempre più importante riuscire a proteggersi, e capire come vengono effettuati questi attacchi è sicuramente un primo passo nella giusta direzione.

[1] `https://it.wikipedia.org/wiki/Denial_of_service`
[2] `https://it.wikipedia.org/wiki/Attacco_man_in_the_middle`
[3] `https://it.wikipedia.org/wiki/Phishing`

Capitolo 9

Bibliografia

`https://docs.python.org/3/`
Documentazione Python

`https://scapy.readthedocs.io/en/latest/introduction.html`
Documentazione della libreria Scapy

`https://it.wikipedia.org/wiki/Pagina_principale`
Utilizzato per alcune definizioni più precise

`https://stackoverflow.com/`
Debug del codice

Black Hat Python: Python Programming for Hackers and Pentesters
Informazioni generali sul port scanning

Violent Python: A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers
Informazioni generali sul port scanning