LABORATORIO DI INTERNET
Report individuale
Analisi del comando *nmap*

Maggio 2021

Diego Zanfardino s256536
Gruppo 21

prof. Mellia Marco e Trevisan Martino

0 Configurazione di rete

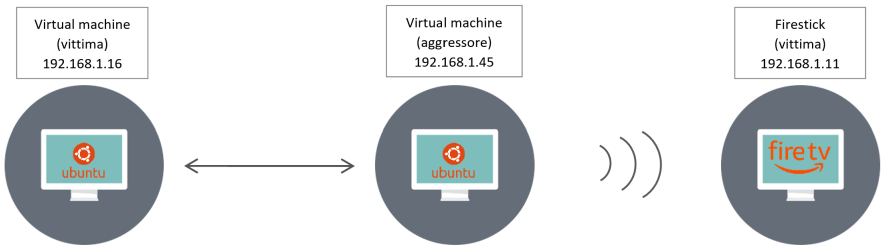

Figura 1: Configurazione iniziale della rete

L'assegnazione degli indirizzi privati è affidata al DHCP del router a cui i dispositivi sono connessi. Tutti i dispositivi coinvolti appartengono alla stessa sottorete. Le due virtual machine montano un sistema operativo linux-based (Ubuntu), mentre per conoscere il sistema operativo della Firestick è possibile sfruttare il comando *"sudo nmap 192.168.1.11 -O"* che fornisce come ipotesi un sistema operativo Android-based (4.X|5.X|6.X).

1 Analisi delle prime 100 porte della virtual machine

1.1 Scan TCP

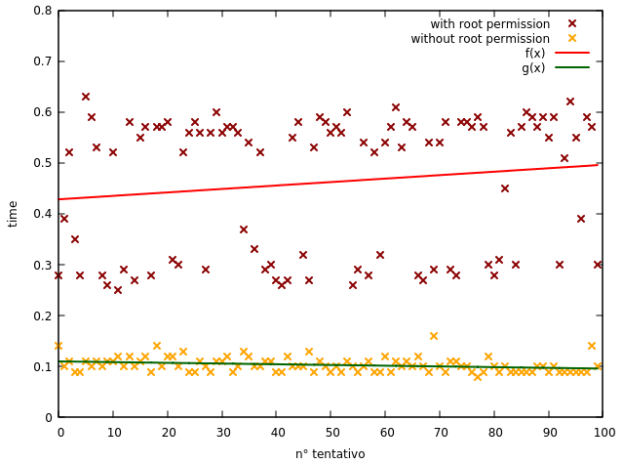

Figura 2: Sudo vs normal user

Il comando

$$sudo\ nmap\ 192.168.1.16\ -p\ 1-100$$

permette di eseguire uno scan delle prime 100 porte TCP dell'indirizzo ip selezionato. Poichè il comando è eseguito come superuser il comportamento di default è in modalita -sS (*SYN Stealth*) che, sfruttando i privilegi di root di poter creare pacchetti raw decidendone l'intestazione, manda pacchetti TCP SYN alla destinazione, in attesa di risposta con flag SYN-ACK (o RST-ACK). In caso di risposta affermativa, invece di chiudere la connessione con un pacchetto ACK, viene mandato un pacchetto RST per chiudere bruscamente la connessione (da qui il nome di questo tipo di scan: *half-open scanning*). Per essere più precisi è il *kernel* della macchina che usa nmap a mandare il RST poichè non riconosce nel SYN-ACK mandato dalla vittima nessun suo tentativo di aprire una connessione, essendo il pacchetto generato *ad hoc* da nmap. Questo tipo di comportamento è utile per destare meno sospetti nei confronti del firewall di rete aprendo diverse connessioni in un intervallo di tempo molto breve. Avendo privilegi di root il comando *nmap* può mandare ARP requests interfacciandosi direttamente con il livello rete, così da avere un riscontro certo se l'host sia attivo o meno, non potendo il protocollo ARP rifiutare di rispondere ad una richiesta. Essendo eseguito come superuser si nota come la porta sorgente sia la stessa per ogni pacchetto SYN mandato.
Nel caso in cui il comando venga eseguito senza permessi di root si fa uso della system call *connect* che tenta di aprire la connessione nel modo convenzionale, usando ogni volta una porta casuale tra quelle non assegnate dallo IANA.
Nel grafico in Figura 2 sono riportati i tempi di esecuzione per 100 comandi nmap verso lo stesso host, eseguiti sia come superuser che come utente normale. Sono inoltre rappresentate le due rette $f(x)$ e $g(x)$ che interpolano i punti ricavati, per cercare di ricavarne l'andamento macroscopico. Il tempo di esecuzione medio per la prima è di circa $0.428s$, mentre per la modalità senza permessi di root è di circa $0.110s$

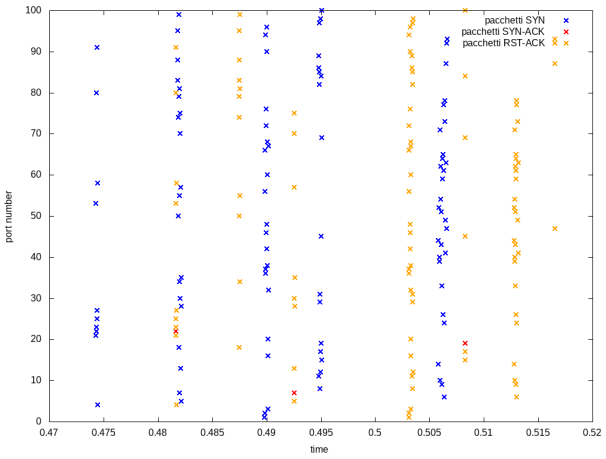

Figura 3: TCP scan

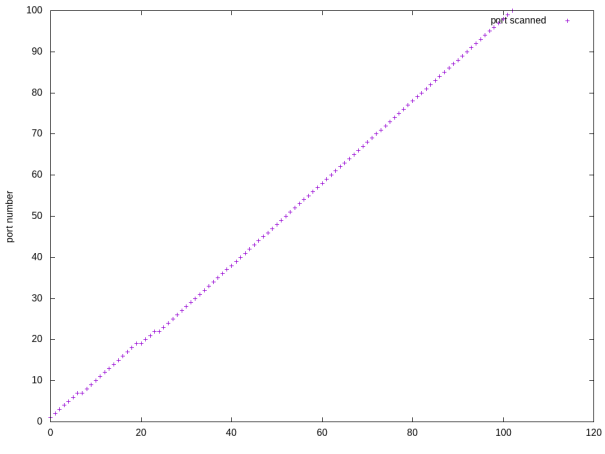

Figura 4: numero di scan per porta

```
laboratorio@laboratorio:~$ sudo nmap 192.168.1.16 -p 1-100
Starting Nmap 7.80 ( https://nmap.org ) at 2021-05-01 22:21 CEST
Nmap scan report for 192.168.1.16
Host is up (0.011s latency).
Not shown: 97 closed ports
PORT   STATE SERVICE
7/tcp  open  echo
19/tcp open  chargen
22/tcp open  ssh
MAC Address: 9C:B6:D0:F1:8C:51 (Rivet Networks)

Nmap done: 1 IP address (1 host up) scanned in 0.47 seconds
laboratorio@laboratorio:~$
```

Lo scan delle porte TCP ha richiesto 0.47 secondi complessivamente, ed il tempo per mandare i singoli pacchetti TCP è anche minore. Per rendere meno evidente l'invio di pacchetti quasi simultaneo verso un singolo host, è importante notare l'ordine delle porte con cui viene eseguito il comando (Figura 3): le prime ad essere interrogate sono le *well-known port* presenti nel range di porte specificato, poichè solitamente più utilizzate; la lista delle porte rimanenti è poi attraversato in maniera casuale. Dalla Figura 4 si nota che ogni porta è interrogata una singola volta, poichè TCP è un protocollo affidabile e *connection-oriented*. I pacchetti duplicati che si vedono dal grafico sono i pacchetti di RST mandati dalla sorgente per chiudere le connessioni attive, che corrispondono alle porte aperte trovate [7-19-22] visibili anche in Figura 3.

1.2 Scan UDP

Il comando

$$sudo\ nmap\ 192.168.1.16\ -p\ 1-100\ -sU$$

permette di eseguire lo scan delle prime 100 porte utilizzando il protocollo UDP. Per comprendere il comportamento del comando con questa ulteriore opzione è importante tenere a mente che UDP è un protocollo *connectionless* e *best-effort*. Per questo motivo l'invio di un pacchetto UDP può portare a 3 scenari indistinguibili dal lato del mittente:

- la destinazione non risponde
- la porta è filtrata
- il pacchetto è andato perso

```
laboratorio@laboratorio:~$ sudo nmap 192.168.1.16 -p 1-100 -sU
Starting Nmap 7.80 ( https://nmap.org ) at 2021-05-01 22:23 CEST
Nmap scan report for 192.168.1.16
Host is up (0.0076s latency).
All 100 scanned ports on 192.168.1.16 are closed
MAC Address: 9C:B6:D0:F1:8C:51 (Rivet Networks)

Nmap done: 1 IP address (1 host up) scanned in 101.95 seconds
laboratorio@laboratorio:~$
```

L'informazione fondamentale da notare è il tempo di esecuzione: per terminare il comando impiega 101.95 secondi, di molto superiore al tempo di esecuzione per lo scan TCP. Il motivo principale è riconducibile al tentativo del sistema operativo di limitare le risposte *ICMP port unreachable*, poprio per rendere il port scanning più oneroso in termini di tempo. In particolare l'informazione è un valore intero espresso in millisecondi, impostato di default al valore 1000 (1 secondo) salvato in */proc/sys/net/ipv4/icmp_ratelimit*. Questo valore è modificabile con il seguente comando: $sudo\ sysctl\ -w\ net.ipv4.icmp\_ratelimit\ =\ 10$, che imposta il limite delle risposte a 1 ogni $10ms$. La differenza tra i tempi impiegati è apprezzabile nel grafico in Figura 6.

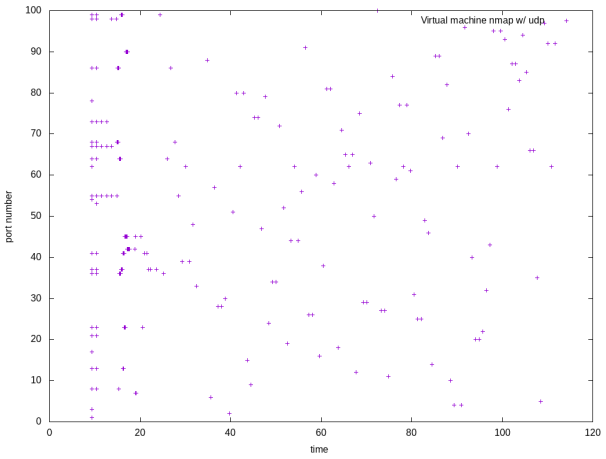

Figura 5: UDP scan

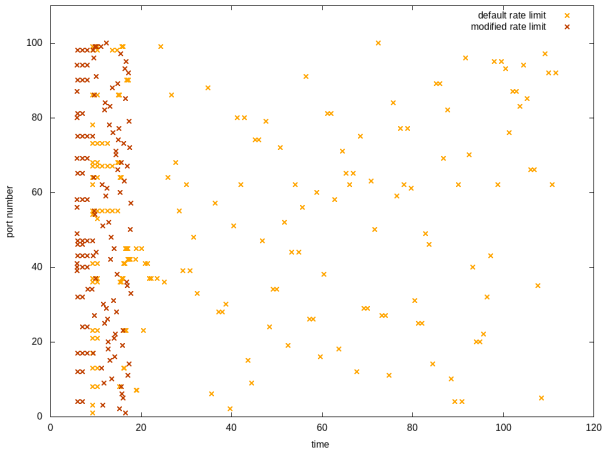

Figura 6: Modifiche a ICMP port unreach. ratelimit

In figura 7 è possibile notare l'altra grande differenza con uno scan TCP. Essendo UDP *best-effort* non è sufficiente interrogare la porta una sola volta. Prima di marcare la porta come filtrata (protetta da un firewall) o chiusa vengono fatti diversi tentativi. Uno dei motivi per cui i pacchetti sonda UDP sono scartati è la loro composizione: la maggior parte di questi viene mandata come pacchetto privo di payload, il quale viene marcato come non valido dal ricevitore anche se la porta UDP è aperta, poichè non è conforme ai pacchetti che si aspetta di ricevere. Dal grafico in Figura 5 è possibile inoltre notare come UDP si adatti alla rete. Inizialmente infatti vengono mandati diversi pacchetti ma, non appena il protocollo si accorge di stare inondando la rete con pacchetti che la destinazione scarterebbe avendo già raggiunto il limite di ICMP port unreachable discusso precedentemente, adatta la propria velcità a tale limite.

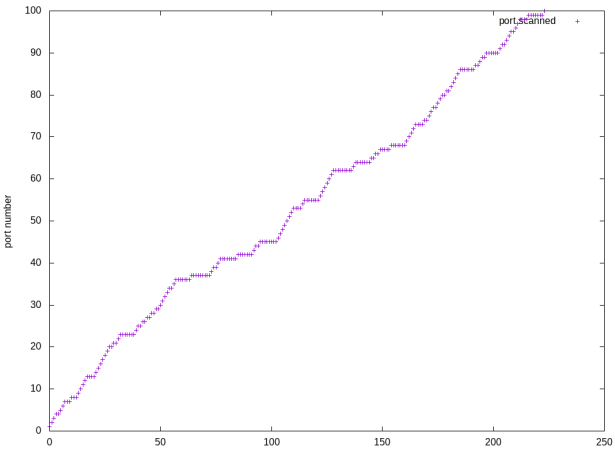

Figura 7: numero di scan per porta

2 Analisi delle prime 100 porte della Firestick TV

2.1 Scan TCP

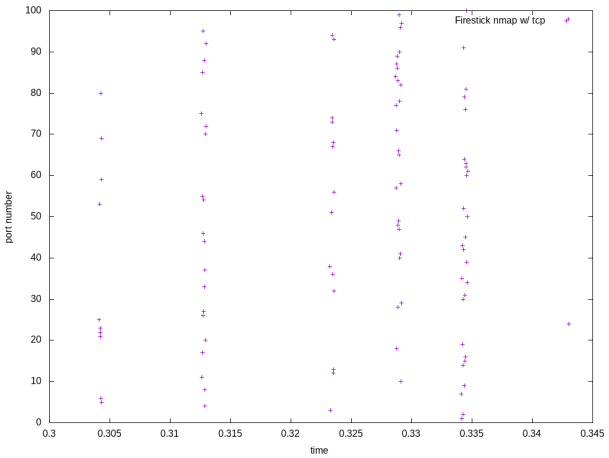

Figura 8: TCP scan

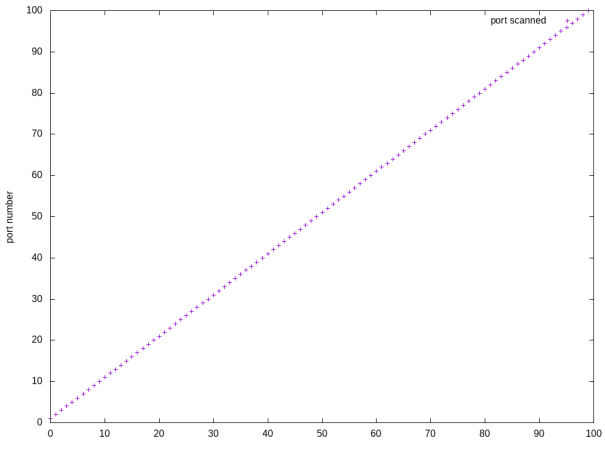

Figura 9: numero di scan per porta

Eseguendo il comando *sudo nmap* 192.168.1.11 $-p\ 1-100$ si esegue lo scan tcp delle prime 100 porte dell'host indicato. Dal grafico in Figura 8 è possibile avere conferma delle considerazioni fatte al caso precedente. Questa modalità di *host-discovery* e scan è particolarmente efficiente, è possibile infatti notare come vengano mandati tutti e 100 pacchetti TCP SYN in alcune decine di millisecondi. I primi pacchetti ad essere mandati hanno nuovamente come destinazione le *well-known* port nel range specificato, che ospitano servizi quali ECHO, FTP, SSH, SMTP, DNS, HTTP...
Queste porte vengono privilegiate per rendere la scansione più rapida: se l'host dovesse essere attivo è più probabile che abbia queste porte aperte; in questo modo si può dire rapidamente se l'host è attivo, evitando di mandare inutilmente pacchetti in rete.

In Figura 9 è possibile notare il numero di tentativi per porta eseguiti da nmap, che restano costanti al valore 1, non essendoci servizi aperti sull'host ed essendo TCP affidabile e *connection-orentied*. Un particolare degno di nota è che con questo tipo di scansione non sempre tutte le porte aperte vengono rilevate, principalmente per la presenza di firewall, mentre eseguendo una scnasione più completa risulata aperta su questo dispositivo la porta TCP 8009, che ospita il servizio "ajp13", vulnerabile a LFI (Local File Inclusion).

2.2 Scan UDP

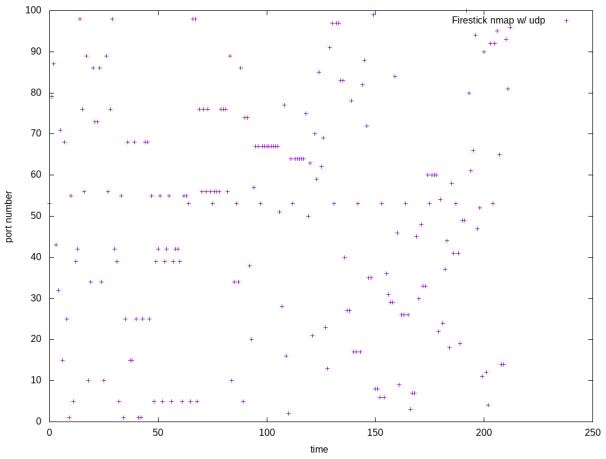

Figura 10: UDP scan

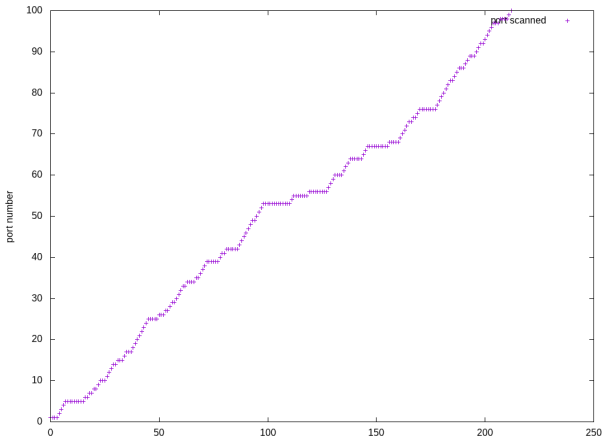

Figura 11: Numero di scan per porta

Eseguendo il comando *sudo nmap* 192.168.1.11 $-p\ 1-100\ -sU$ si ottiene anche in questo caso conferma dei ragionamenti fatti in precedenza per uno scan UDP. Non essendo noti con certezza il sistema operativo dell'host, e di conseguenza il suo comportamento in risposta ai *probe* UDP, si può solo evidenziare come in questo caso rispetto al precedente ci metta più tempo ad adattarsi ai limiti imposti alla destinazione, evitando *flood* della rete. Infatti se nel primo caso come si nota in Figura 5 il transitorio verso la fase in cui il mittente è allineato con le capacità di risposta del destinatario (limitate dalla rete o da altri fattori come visto nel punto 1.2) dura una decina di secondi, in questo secondo caso si risolve parzialmente solo a metà scan, come si può evincere dalla Figura 10, vista la densità di pacchetti nella prima metà dello scan.

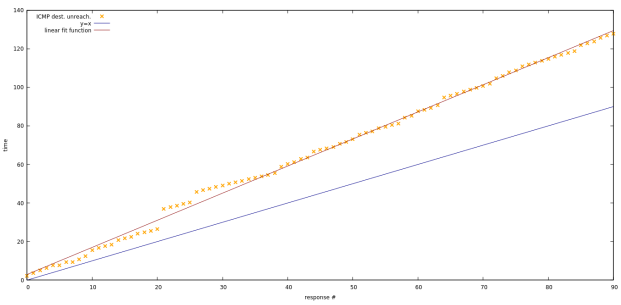

Figura 12: ICMP destination unreachable

É inoltre possibile notare che la quantità di pacchetti mandati e il tempo impiegato per mandarli (e di conseguenza ricevere i messaggi ICMP in risposta) è praticamente analogo al caso esaminato precedentemente. Questo fa supporre un comportamento simile del sistema operativo per limitare il port scanning. In figura 12 è possibile vedere i pacchetti *ICMP destination unreachable* mandati dalla vittima. Se non ci fossero intervalli di tempo in cui l'host non ha mandato alcun pacchetto in risposta, l'andamento avrebbe meglio rispecchiato una retta parallela ad $y(t) = x(t)$, una relazione lineare che fa corrispondere ad ogni secondo un singolo pacchetto mandato dalla vittima in risposta al *probe*.

3 Appendice

3.1 Script per preparazione dataset

```
#!/bin/bash

final=$(echo "$1" | cut -f 1 -d '.')_def.dat
cat $1 | tr -s ' ' | cut -d ' ' -f 10 > nmap_port.dat
cat $1 | tr -s ' ' | cut -d ' ' -f 3 > nmap_time.dat

paste nmap_time.dat nmap_port.dat > $final

rm nmap_port.dat
rm nmap_time.dat
```

3.2 Script per grafico Figura 2

3.2.1 Collezione dati

```
  for((s=0; s<100; s+=1))
do
sudo nmap 192.168.1.1-100  | grep "scanned in" | cut -d ' ' -f11 >> nmap_sudo_time.dat
done
  for((s=0; s<100; s+=1))
do
nmap 192.168.1.1-100  | grep "scanned in" | cut -d ' ' -f11 >> nmap_time.dat
done
```

3.2.2 Grafico

```
  set xlabel "n° tentativo"
  set ylabel "time"
  set terminal png size 1024, 768
  set output "sudo_vs_normaluser.png"

  f(x)=a*x+b
  g(x)=c*x+d
  fit f(x) 'nmap_time.dat' via a,b
  fit g(x) 'nmap_sudo_time.dat' via c,d

  plot 'nmap_time.dat' title "without root permission" lc rgb "orange" pt 2 lw 2,
       'nmap_sudo_time.dat' title "with root permission" lc rgb "dark-red" pt 2 lw 2.5,
       f(x) lc "red",
       g(x) lc "green"
```

3.3 Script per grafici TCP Figura 3

```
  set xlabel "time"
  set ylabel "port number"
  set terminal png size 1024, 768
  set output "vm_tcp.png"

  plot  'nmap_syn_def.dat' using 1:2 title "pacchetti SYN" lc rgb "blue" pt 2 lw 2,
    'nmap_synack_def.dat' using 1:2 title "pacchetti SYN-ACK" lc rgb "red" pt 2 lw 2.5,
    'nmap_rstack_def.dat' using 1:2 title "pacchetti RST-ACK" lc rgb "orange" pt 2 lw 2
```

3.4 Script per grafici Figura 8, 9, 10, 11 e simili

```
set terminal png size 1024, 768

set xlabel "time"
set ylabel "port number"
set output "firestick_tcp.png"
plot 'nmap_def.dat' using 1:2 title "Firestick nmap w/ tcp"

set xlabel "time"
set ylabel "port number"
set output "Firestick_udp.png"
plot 'nmap_udp_def.dat' using :2 title "Firestick nmap w/ udp"

set output "Firestick_tcp_num_test.png"
set ylabel "port number"
plot '<sort -n -k 2 nmap_def.dat' using :2 title "port scanned"

set output "Firestick_udp_num_test.png"
set ylabel "port number"
plot '<sort -n -k 2 nmap_udp_def_dnsecho.dat' using :2 title "port scanned"
```

3.5 Script per Figura 12

```
  set xlabel "response #"
  set ylabel "time"
  set terminal png size 1024, 768
  set output "ICMP_d_u.png"

  f(x)=a*x+b
  fit f(x) 'icmp_pu.dat' via a,b
  set yrange [0:140]
  set xrange [0:90]

  plot 'icmp_pu.dat' title "ICMP dest. unreach." lc rgb "orange" pt 2 lw 2,
       x lc "dark-blue" title "y=x",
       f(x) lc "dark-red" title "linear fit function"
```

3.6 Log file dei fit

3.6.1 fit nello script al punto 3.5

```
*******************************************************************************
Sun May  9 00:53:17 2021


FIT:    data read from 'data.dat'
        format = z
        #datapoints = 92
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
  f(x) = a*x+b
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   a             b
  0 4.9279558493e+04   0.00e+00  3.73e+01    1.000000e+00   1.000000e+00
  4 4.1866803645e+02  -9.63e-02  3.73e-03    1.405480e+00   2.925743e+00

After 4 iterations the fit converged.
final sum of squares of residuals : 418.668
rel. change during last iteration : -9.62567e-07

degrees of freedom    (FIT_NDF)                        : 90
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 2.15682
variance of residuals (reduced chisquare) = WSSR/ndf   : 4.65187

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 1.40548          +/- 0.008467     (0.6025%)
b               = 2.92574          +/- 0.4461       (15.25%)

correlation matrix of the fit parameters:
                a      b
a               1.000
b              -0.864  1.000
```

3.6.2 fit nello script al punto 3.2.2

```
*******************************************************************************
Sat May  8 00:50:24 2021


FIT:    data read from 'nmap_sudo_time.dat'
        format = z
        #datapoints = 100
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
  f(x)=a*x+b
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   a             b
  0 3.3359392660e+05   0.00e+00  4.05e+01    1.000000e+00   1.000000e+00
  5 1.7438891689e+00  -1.17e-08  4.05e-04    6.780678e-04   4.284356e-01

After 5 iterations the fit converged.
final sum of squares of residuals : 1.74389
rel. change during last iteration : -1.16886e-13
```

```
degrees of freedom    (FIT_NDF)                        : 98
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.133397
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.0177948

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 0.000678068      +/- 0.0004621    (68.15%)
b               = 0.428436         +/- 0.02648      (6.181%)

correlation matrix of the fit parameters:
                a      b
a               1.000
b              -0.864  1.000


*******************************************************************************
Sat May  8 00:50:44 2021


FIT:    data read from 'nmap_time.dat'
        format = z
        #datapoints = 100
        residuals are weighted equally (unit weight)

function used for fitting: g(x)
  g(x)=c*x+d
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   c             d
  0 3.3733228500e+05   0.00e+00  4.05e+01    1.000000e+00   1.000000e+00
  5 1.8316262826e-02  -2.75e-06  4.05e-04   -1.411341e-04   1.101861e-01

After 5 iterations the fit converged.
final sum of squares of residuals : 0.0183163
rel. change during last iteration : -2.7494e-11

degrees of freedom    (FIT_NDF)                        : 98
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0136712
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.000186901

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
c               = -0.000141134     +/- 4.736e-05    (33.56%)
d               = 0.110186         +/- 0.002714     (2.463%)

correlation matrix of the fit parameters:
                c      d
c               1.000
d              -0.864  1.000
```