CMP_SCI-3780 Project 2

Cynthia Brown

Partner:none

Environment: Windows 10 L480 Lenovo laptop

Language: Python

Cryptography Library: cryptography.io

Programs:

project2.py

> A blanket program that allows you to access all three programs and run the different processes without having to run a different program.

pass_auth.py

> Allows you to sign up or sign in to an "account"

rand_pass_gen.py

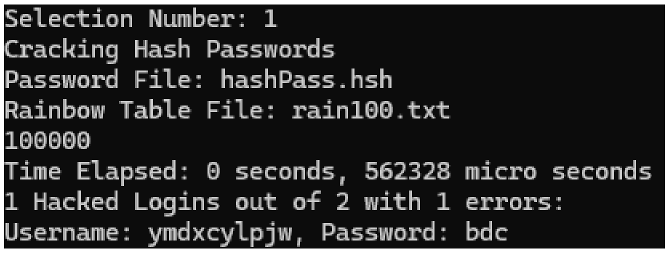
> Can be used to create login files and rainbow table files.

pass_crack.py

> Can be used to crack hash or salt hash passwords

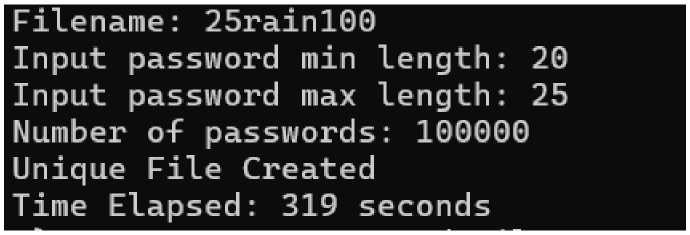
Results:

Hash Passwords: if you have the password in the rainbow file then it can be cracked in seconds. 3-char password took less than a sec.



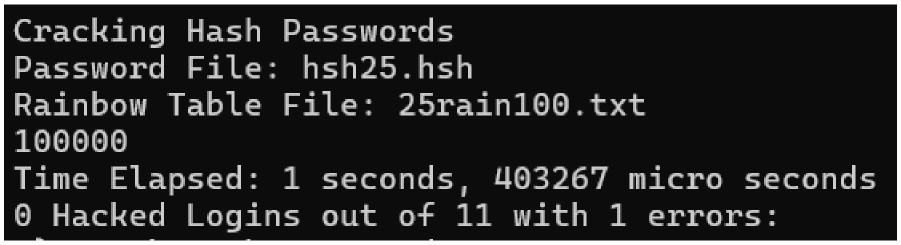
```
Selection Number: 1
Cracking Hash Passwords
Password File: hashPass.hsh
Rainbow Table File: rain100.txt
100000
Time Elapsed: 0 seconds, 562328 micro seconds
1 Hacked Logins out of 2 with 1 errors:
Username: ymdxcylpjw, Password: bdc
```

To create a rainbow table file of 100000 20-25 length passwords with hashes took 319 seconds.



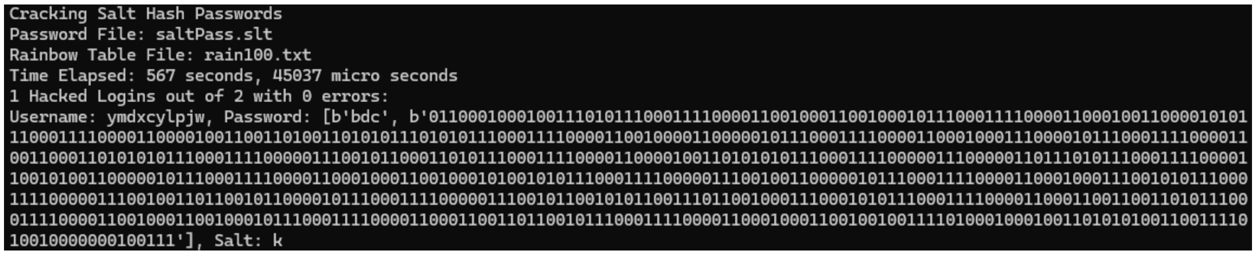
```
Filename: 25rain100
Input password min length: 20
Input password max length: 25
Number of passwords: 100000
Unique File Created
Time Elapsed: 319 seconds
```

And it failed to have the same passwords as the 10 logins that I created with lengths of 20-25.



```
Cracking Hash Passwords
Password File: hsh25.hsh
Rainbow Table File: 25rain100.txt
100000
Time Elapsed: 1 seconds, 403267 micro seconds
0 Hacked Logins out of 11 with 1 errors:
```

Salt Hash Passwords: this takes much longer since you have to apply the salt and hash every password option. This takes more time since you can't pre-hash the passwords. A 3-char password took 538 seconds.



```
Cracking Salt Hash Passwords
Password File: saltPass.slt
Rainbow Table File: rain100.txt
Time Elapsed: 567 seconds, 45037 micro seconds
1 Hacked Logins out of 2 with 0 errors:
Username: ymdxcylpjw, Password: [b'bdc', b'011000100010011101011100011110000110010001100100010111000111100001100010011000010101
1100011110000110000100110011010011010101110101011100011110000110010000110000010111000111100001100010001110000101110001111000011
0011000110101010111000111100000111001011000110101110001111000011000010011010101011100011110000011100000110111010111000111100001
1001010011000000101110001111000011000100011001000101001010111000111100000111001001100000101110001111000011000100011100101011 1000
1111000000111001001101100101100001011100011110000001110010110010101100111011001000111000101011100011110000110001100110011010111 00
0111100001100100011001000101110001111000011000110011011001011100011110000110001000110010010011110100010001001101010100110011110
10010000000100111'], Salt: k
```

Conclusions:

Longer passwords are safer with a hash but with enough time they can still be easily hacked. If I had a rainbow table already created with every possible combination, then I could have hacked the passwords without much effort. With the salt hash passwords, a hacker must use a lot more time and energy to possibly hack a password. None of this rules out luck but pretty sure only a 2-step authentication system can stop that.

```
import pass_auth
import rand_pass_gen
import pass_crack


def main():
    while True:
        print("1) Password Authentication")
        print("2) Random Password Generator")
        print("3) Password Cracking")
        print("0) Exit\n")
        try:
            choice = int(input("Selection Number: "))
            if choice >= 0 and choice < 4:
                match choice:
                    case 1:
                        pass_auth.main()
                    case 2:
                        rand_pass_gen.main()
                    case 3:
                        pass_crack.main()
                    case 0:
                        return
            else:
                raise ValueError
        except ValueError:
            print("\nPick a number from 0-3\n")


main()
```

```
# This is a basic program on Python
#
# Writing this program has taught me that I really need
# Dinosaurs with lazer guns. pew pew
from ast import Bytes
from msilib import Binary
import os
from sys import maxsize
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC


def get_username(su):
    # 10 char length a-z
    while True:
        try:
            username = input("Username: ")
            tst = len(username)
            tt = username.isalpha()
            if len(username) <= 10 and username.isalpha():
                if su == 0:
                    exists = exists_username(bytes(username, 'utf-8'))
                    if exists:
                        raise ValueError
                    else:
                        return bytes(username, 'utf-8')
                else:
                    return bytes(username, 'utf-8')
            else:
                raise ValueError
        except ValueError:
            print("Invalid username.")


def get_password():
    while True:
        try:
            password = input("Password: ")
            # lowercase a-z with configurable length
            if len(password) < maxsize and len(password) > 0 and
              password.isalpha() and password.islower():
                return bytes(password, 'utf-8')
            else:
                raise ValueError
        except ValueError:
            print("Invalid Password")


def hash_password(password: bytes):
    # Create a hash of the password
    hashPass = hashes.Hash(hashes.SHA256())
    hashPass.update(password)
    hPass = hashPass.finalize()
    return bytes(''.join(format(i, '08b') for i in bytearray(str(hPass),
      encoding ='utf-8')), 'utf-8')


def salt_password(password: bytes, salt: bytes):
    # Lastly hash your password using a salt
    if salt == b'':
        salt = os.urandom(1)
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=480000,
    )
    # Add the password to the mix
    key = kdf.derive(password)
    return bytes(''.join(format(i, '08b') for i in bytearray(str(key), encoding
      ='utf-8')), 'utf-8'), salt


def write_plain(username: bytes, password: bytes):
    try:
        with open("./files/plaintext.txt", "ab") as fp:
            # Write user data to the file
            fp.write(username + b':' + password[0] + b';')
    except IOError:
        print("Failed to save text login")
        raise IOError


def write_hash(username: bytes, password: bytes):
    try:
        with open("./files/hashPass.hsh", "ab") as fh:
            # write your data and remember to use your hashed password and not
              our plain text
            hshpass = hash_password(password[0])
            fh.write(username + b':' + hshpass + b';')
    except IOError:
        print("Failed to save hash login")
        raise IOError


def write_salt(username: bytes, password: bytes):
    try:
        with open("./files/saltPass.slt", "ab") as fs:
            # Write your username and the salt hashed password
            passw, salt = salt_password(password[0], b'')
            fs.write(username + b':' + passw + b':' + salt + b';')
    except IOError:
        print("Failed to save salt login")
        raise IOError


def exists_username(un):
    if os.path.isfile("./files/plaintext.txt"):
        with open("./files/plaintext.txt", "rb") as fp:
            users = fp.read().split(b';')
            for line in users:
                user = line.split(b':')
                if user[0] == un:
                    return True
    return False


def login_txt(username: bytes, password: bytes):
    if os.path.isfile("./files/plaintext.txt"):
        with open("./files/plaintext.txt", "rb") as fp:
            users = fp.read().split(b';')
            for line in users:
                user = line.split(b':')
                if user[0] == username and user[1] == password:
                    return True
    return False


def login_hsh(username: bytes, password: bytes):
    if os.path.isfile("./files/hashPass.hsh"):
        with open("./files/hashPass.hsh", "rb") as hp:
            users = hp.read().split(b';')
            for line in users:
                values = line.split(b':')
                if values[0] == username:
                    hpass = hash_password(password)
                    if values[1] == hpass:
                        return True
                    else:
                        return False
    return False


def login_slt(username: bytes, password: bytes):
    if os.path.isfile("./files/saltPass.slt"):
        with open("./files/saltPass.slt", "rb") as sp:
            users = sp.read().split(b';')
            for line in users:
                if line.startswith(username):
                    values = line.split(b':')
                    print(values)
                    sltpass, salt = salt_password(password, values[2])
                    if values[1] == sltpass:
                        return True
                    else:
                        return False
    return False


def signup():
    print("Signup for an account\n")
    # Get user data
    username = get_username(0)
    password = get_password(),
    # Open plain text document and create if it doesn't exist
    write_plain(username, password)
    # Open/create the file that stores username and hashed passwords
    write_hash(username, password)
    # Open/create your file
    write_salt(username, password)
    # Close you file
    print("Signup Successful")
    signin()


def signin():
    print("signin")
    exists = False
    while exists == False:
        username = get_username(1)
        password = get_password()
        exists = exists_username(username)
        if exists:
            print("Trying to login")
            # find stored username and password
            txtlogin = login_txt(username, password)
            hshlogin = login_hsh(username, password)
            sltlogin = login_slt(username, password)
            if txtlogin:
                print("Text Login worked\n")
            else:
                print("Text login failed\n")
            if hshlogin:
                print("Hash login worked\n")
            else:
                print("Hash login failed\n")
            if sltlogin:
                print("Salt login worked\n")
            else:
                print("Salt login failed\n")
        else:
            print("Invalid Login\n")
            if input("Do you want to create an accout?(y/n) ").lower().strip() ⮒
              == 'y':
                signup()


def main():
    print("Welcome!\n")
    uInput = input("Do you have an account?(y/n)")
    if uInput.lower() == "y":
        signin()
    else:
        signup()


# main()
```

```
import datetime
from fileinput import filename
import os
import random
import string
import pass_auth


def exists_username(file, username: bytes):
    ty = 'rb'
    if os.path.isfile('./files/' + file):
        with open('./files/' + file, ty) as fp:
            users = fp.read().split(';')
            for line in users:
                user = line.split(':')
                if user[0] == username:
                    return True
    return False


def write_file(file_name, data):
    ty = "ab"
    try:
        with open('./files/' + file_name, ty) as fp:
            # Write user data to the file
            fp.writelines(data)
    except IOError:
        print("Failed to save text login")
        raise IOError


def write_plain(username: bytes, password: bytes):
    return username + b':' + password + b';'


def write_hash(username: bytes, password: bytes):
    hshpass = pass_auth.hash_password(password)
    # hshpass = ''.join(format(i, '08b') for i in bytearray(str
      (pass_auth.hash_password(password)), encoding ='utf-8'))
    return username + b':' + hshpass + b';'


def write_salt(username: bytes, password: bytes):
    password, salt = pass_auth.salt_password(password, b'')
    # spass = ''.join(format(i, '08b') for i in bytearray(str(password),
      encoding ='utf-8'))
    return username + b':' + password + b':' + salt + b';'
```

```
def gen_user(users):
    while True:
        # ty = "rb"
        username = bytes(''.join(random.choices(string.ascii_lowercase, k=10)),
           'utf-8')
        # exists = exists_username(file_name, username)
        if username not in users:
            return username


def gen_pass(pass_min, pass_max):
    pass_length = random.randint(pass_min, pass_max)
    return bytes(''.join(random.choices(string.ascii_lowercase,
      k=pass_length)), 'utf-8')


def get_userinput():
    file_name = input("Filename: ") or 'test'
    pass_min = int(input("Input password min length: ") or '3')
    pass_max = int(input("Input password max length: ") or '5')
    num_pass = int(input("Number of passwords: ") or '100')
    return file_name, pass_min, pass_max, num_pass


def create_txt():
    file_name, pass_min, pass_max, num_pass = get_userinput()
    pdata = []
    users = []
    while num_pass > 0:
        pdata.append(write_plain(users.append(gen_user(users)), gen_pass
          (pass_min, pass_max)))
        num_pass = num_pass - 1
    write_file(file_name + '.txt', pdata)
    print("Textfile Created")
    return


def create_unique_txt():
    file_name, pass_min, pass_max, num_pass = get_userinput()
    elapse = datetime.datetime.now()
    passwords = []
    write_pass = []
    while num_pass > 0:
        m = True
        while m:
            password = gen_pass(pass_min, pass_max)
            hpass = pass_auth.hash_password(password)
            password = password + b':' + hpass + b';'
            if password not in passwords:
                passwords.append(password)
                num_pass = num_pass - 1
                m = False
    write_file(file_name + '.txt', passwords)
    print("Unique File Created")
    elapse = datetime.datetime.now() - elapse
    print(f"Time Elapsed: {elapse.seconds} seconds")
    return


def create_hsh():
    file_name, pass_min, pass_max, num_pass = get_userinput()
    elapse = datetime.datetime.now()
    hdata = []
    users = []
    while num_pass > 0:
        users.append(gen_user(users))
        hdata.append(write_hash(users[-1], gen_pass(pass_min, pass_max)))
        num_pass = num_pass - 1
    write_file(file_name + '.hsh', hdata)
    print("Hashfile Created")
    elapse = datetime.datetime.now() - elapse
    print(f"Time Elapsed: {elapse.seconds} seconds")
    return


def create_slt():
    file_name, pass_min, pass_max, num_pass = get_userinput()
    elapse = datetime.datetime.now()
    sdata = []
    users = []
    while num_pass > 0:
        users.append(gen_user(users))
        sdata.append(write_salt(users[-1], gen_pass(pass_min, pass_max)))
        num_pass = num_pass - 1
    write_file(file_name + '.slt', sdata)
    print("Saltfile Created")
    elapse = datetime.datetime.now() - elapse
    print(f"Time Elapsed: {elapse.seconds} seconds")
    return


def create_all():
    file_name, pass_min, pass_max, num_pass = get_userinput()
    elapse = datetime.datetime.now()
    pdata = []
    hdata = []
    sdata = []
    users = []
    while num_pass > 0:
        users.append(gen_user(users))
        password = gen_pass(pass_min, pass_max)
        pdata.append(write_plain(users[-1], password))
        hdata.append(write_hash(users[-1], password))
        sdata.append(write_salt(users[-1], password))
        num_pass = num_pass - 1
    write_file(file_name + '.txt', pdata)
    write_file(file_name + '.hsh', hdata)
    write_file(file_name + '.slt', sdata)
    print("Files Created")
    elapse = datetime.datetime.now() - elapse
    print(f"Time Elapsed: {elapse.seconds} seconds")
    return


def main():
    print("random pass program")
    print("Written by: Cynthia Brown")
    while True:
        print("1) Create Text Password File")
        print("2) Create Hash Password File")
        print("3) Create Salt-Hash Password File")
        print("4) Create All Types Password Files")
        print("5) Create Unique Text Password File")
        print("0) Exit\n")
        try:
            choice = int(input("Selection Number: "))
            if choice >= 0 and choice < 6:
                match choice:
                    case 1:
                        create_txt()
                    case 2:
                        create_hsh()
                    case 3:
                        create_slt()
                    case 4:
                        create_all()
                    case 5:
                        create_unique_txt()
                    case 0:
                        return
            else:
                raise ValueError
        except ValueError:
            print("\nPick a number from 0-5\n")


# main()
```

```
import datetime
from os import error
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from rand_pass_gen import write_file
files = ["hashPass.hsh", "saltPass.slt", "plainuniquetext.txt"]

def get_password_file(file):
    ty = 'rb'
    try:
        with open('./files/' + file, ty) as pf:
            txt = pf.read()
            return txt.split(b';')
    except IOError:
        print("failed to open file\n")
        return


def user_input():
    passfile = input("Password File: ") or ''
    crackfile = input("Rainbow Table File: ") or files[2]
    return passfile, crackfile


def hash_password(password: bytes):
    # Create a hash of the password
    hashPass = hashes.Hash(hashes.SHA256())
    hashPass.update(password)
    return hashPass.finalize()


def salt_password(password: bytes, salt: bytes):
    # Lastly hash your password using a salt
    if salt == b'':
        salt = os.urandom(1)
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=480000,
    )
    # Add the password to the mix
    key = kdf.derive(password)
    return bytes(''.join(format(i, '08b') for i in bytearray(str(key), encoding
       ='utf-8')), 'utf-8'), salt


def crack_hsh():
    print("Cracking Hash Passwords")
    passfile, crackfile = user_input()
    elapse = datetime.datetime.now()
    if not passfile:
        passfile = files[0]
    if not crackfile:
        crackfile = files[2]
    password_file = get_password_file(passfile)
    crack_file = get_password_file(crackfile)
    hash_crack = []
    hash_pass = []
    hacked_users = []
    hacked_users_file = []
    errors = []
    for line in crack_file:
        if line:
            tmp = line.split(b':')
            hash_pass.append(tmp[0])
            hash_crack.append(tmp[1])
    print(len(hash_crack))
    for line in password_file:
        user = line.split(b':')
        try:
            for hsh in hash_crack:
            # if user[1] in hash_crack:
                if user[1] == hsh:
                    hacked_users.append([user[0],hash_pass[hash_crack.index
                    (hsh)]])
                    hacked_users_file.append(user[0] + b':' + hash_pass
                    [hash_crack.index(hsh)] + b';')
        except:
            errors.append(line)
    # Write hacked users to file and screen
    write_file("hacked_logins.txt", hacked_users_file)
    elapse = datetime.datetime.now() - elapse
    print(f"Time Elapsed: {elapse.seconds} seconds, {elapse.microseconds} micro
      seconds")
    print(str(len(hacked_users)) + " Hacked Logins out of " + str(len
      (password_file)) + " with " + str(len(errors)) + " errors: ")
    for user in hacked_users:
        print("Username: " + user[0].decode('utf-8') + ", Password: " + user
          [1].decode('utf-8') + "")


def crack_slt():
    print("Cracking Salt Hash Passwords")
    passfile, crackfile = user_input()
    elapse = datetime.datetime.now()
    if not passfile:
        passfile = files[1]
    if not crackfile:
        crackfile = files[2]
    password_file = get_password_file(passfile)
    crack_file = get_password_file(crackfile)
    salt_crack = []
    hacked_users = []
    hacked_users_file = []
    errors = []
    users = []
    passwords = []
    salts = []
    for line in crack_file:
        if line:
            tmp = line.split(b':')
            salt_crack.append([tmp[0], tmp[1]])
    for login in password_file:
        if login:
            tst = login.split(b':')
            users.append(tst[0])
            passwords.append(tst[1])
            salts.append(tst[2])
    if salts:
        for salt in salts:
            if salt != b'':
                for line in salt_crack:
                    i = salts.index(salt)
                    if line:
                        spass, s = salt_password(line[0],salt)
                        salt_crack.append([spass, salt])
                        if spass == passwords[i]:
                            hacked_users.append([users[i], line, salt])
                            hacked_users_file.append(users[i]+ b':' + line[0] +
                        b':' + salt)
                            del salts[i]
                            del users[i]
                            del passwords[i]
                            break
            #if hacked_users:
                #break

    # Write hacked users to file and screen
    write_file("hacked_salt_logins.txt", hacked_users_file)
    elapse = datetime.datetime.now() - elapse
    print(f"Time Elapsed: {elapse.seconds} seconds, {elapse.microseconds} micro
       seconds")
    print(str(len(hacked_users)) + " Hacked Logins out of " + str(len
      (password_file)) + " with " + str(len(errors)) + " errors: ")
    for user in hacked_users:
        print("Username: " + user[0].decode('utf-8') + ", Password: " + str
          (user[1]) + ", Salt: " + user[2].decode('utf-8'))


def main():
    print("pass crack program")
    print("Written by: Cynthia Brown")
    while True:
        print("1) Crack Hash Password")
        print("2) Crack Salt Hash Password")
        print("0) Exit\n")
        try:
            choice = int(input("Selection Number: "))
            if choice >= 0 and choice < 3:
                match choice:
                    case 1:
                        crack_hsh()
                    case 2:
                        crack_slt()
                    case 0:
                        return
            else:
                raise ValueError
        except ValueError:
            print("\nPick a number from 1-3\n")


# main()
```