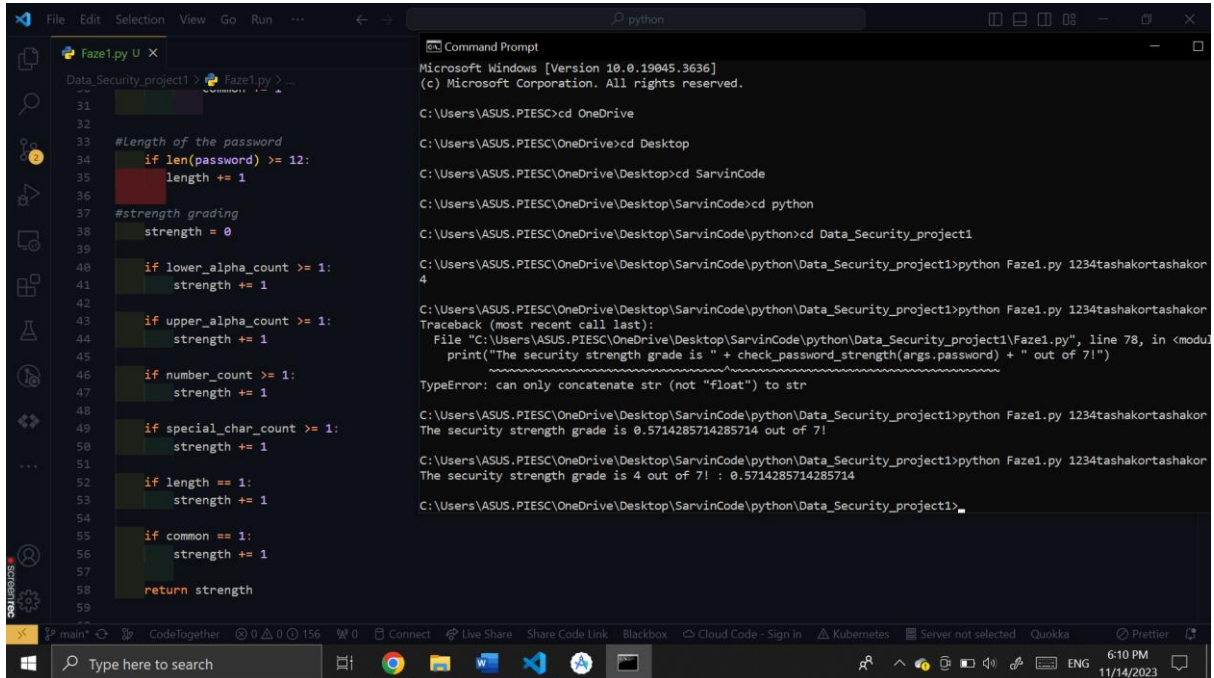


گزارش پروژه عملی اول مبانی امنیت اطلاعات

سروین نامی ۹۹۳۱۱۰۳

فاز ۱-۱:



```
File Edit Selection View Go Run ... python
Data_Security_project1 Faze1.py
31
32
33 #Length of the password
34 if len(password) >= 12:
35     length += 1
36
37 #strength grading
38 strength = 0
39
40 if lower_alpha_count >= 1:
41     strength += 1
42
43 if upper_alpha_count >= 1:
44     strength += 1
45
46 if number_count >= 1:
47     strength += 1
48
49 if special_char_count >= 1:
50     strength += 1
51
52 if length == 1:
53     strength += 1
54
55 if common == 1:
56     strength += 1
57
58 return strength
59

Command Prompt
Microsoft Windows [Version 10.0.19045.3636]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS.PIESC>cd OneDrive
C:\Users\ASUS.PIESC\OneDrive>cd Desktop
C:\Users\ASUS.PIESC\OneDrive\Desktop>cd SarvinCode
C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode>cd python
C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python>cd Data_Security_project1
C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1>python Faze1.py 1234tashakortashakor
4
C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1>python Faze1.py 1234tashakortashakor
Traceback (most recent call last):
  File "C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1\Faze1.py", line 78, in <module>
    print("The security strength grade is " + check_password_strength(args.password) + " out of 7!")
TypeError: can only concatenate str (not "float") to str
C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1>python Faze1.py 1234tashakortashakor
The security strength grade is 0.5714285714285714 out of 7!
C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1>python Faze1.py 1234tashakortashakor
The security strength grade is 4 out of 7! : 0.5714285714285714
C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1>
```

کد:

```
import argparse
import string

parser = argparse.ArgumentParser(description="Password Security Assessment Tool")
parser.add_argument("password", help="Password")

args = parser.parse_args()

def check_password_strength(password):
    lower_alpha_count = upper_alpha_count = number_count =
    special_char_count = length = common = 0

    #lowercase, uppercase, digits
    for char in list(password):
        if char in string.ascii_lowercase:
            lower_alpha_count += 1
        elif char in string.ascii_uppercase:
            upper_alpha_count += 1
        elif char in string.digits:
            number_count += 1
        else:
            special_char_count += 1

    #dictionary check
    with open("10-million-password-list-top-1000000.txt", 'r') as file:
        content = file.read()
```

```

        if password not in content:
            common += 1

#length of the password
    if len(password) >= 12:
        length += 1

#strength grading
    strength = 0

    if lower_alpha_count >= 1:
        strength += 1

    if upper_alpha_count >= 1:
        strength += 1

    if number_count >= 1:
        strength += 1

    if special_char_count >= 1:
        strength += 1

    if length == 1:
        strength += 1

    if common == 1:
        strength += 1

    return strength

grade = check_password_strength(args.password)
print("The security strength grade is " + str(grade) + " out of 7! : " +
      str(grade/7))

```

توضیحات:

۱. `argparse` برای دریافت ورودی از کاربر استفاده می‌شود. در اینجا، یک متغیر `password` از کاربر دریافت می‌شود که بعداً برای ارزیابی امنیت آن استفاده می‌شود.

۲. `check_password_strength` یک تابع است که وظیفه ارزیابی قوت رمز عبور را دارد:

- بررسی تعداد حروف کوچک، حروف بزرگ، اعداد و کاراکترهای خاص در رمز عبور.

- بررسی اینکه رمز عبور در یک لیست از رمزهای شناخته‌شده و رایج قرار دارد یا خیر.

- بررسی طول رمز عبور برای اطمینان از حداقل طول موردنیاز برای امنیت.

- امتیازدهی به امنیت رمز عبور براساس موارد بالا و برگرداندن امتیاز کلی.

۳. در آخر، این ابزار امنیت رمز عبور با استفاده از توابع تعریف شده، امتیاز امنیت رمز عبور و نسبت این امتیاز به امتیاز کلی (۷ امتیاز) را چاپ می‌کند.

به طور خلاصه، این کد به کمک ورودی از کاربر رمز عبور را ارزیابی می‌کند و یک امتیاز امنیتی به آن اختصاص می‌دهد، همچنین نسبت این امتیاز به حداکثر امتیاز ممکن (۷) را نشان می‌دهد.

فاز ۱-۲:

```
File Edit Selection View Go Run ... python
Fazel-1.py Fazel-2.py M X
Data_Security_project1 > Fazel-2.py > main
32 execution_time_unit = 'seconds'
33 elif bits < 52:
34     execution_time /= 60
35     execution_time_unit = 'minutes'
36 elif bits < 57:
37     execution_time /= (60*60)
38     execution_time_unit = 'hours'
39 elif bits < 60:
40     execution_time /= (60*60*60)
41     execution_time_unit = 'days'
42 elif bits < 62:
43     execution_time /= (60*60*60*24)
44     execution_time_unit = 'weeks'
45 elif bits < 65:
46     execution_time /= (60*60*60*24*7)
47     execution_time_unit = 'months'
48 elif bits >= 65:
49     execution_time /= (60*60*60*24*7*12)
50     execution_time_unit = 'years'
51 return attempts, execution_time, execution_time_unit
52
53 def main():
54     parser = argparse.ArgumentParser()
55     parser.add_argument("password", help="Specify the password to crack")
56     parser.add_argument("mode", type=int, help="Specify the mode of attack")
57     parser.add_argument("search_space", type=int, help="Specify the search space")
58     parser.add_argument("n", type=int, help="Specify the length of password")
59     parser.add_argument("k", help="Specify a part of password")
60
61     args = parser.parse_args()
62     password = args.password
63     mode = args.mode
64     search_space = args.search_space
65     n = args.n
66     k = args.k
67
68     attempts, execution_time, execution_time_unit = crack_password(password, mode, search_space, n, k)
69     print(f"Cracked password: {password} in {attempts} attempts, {execution_time} {execution_time_unit}")
70
71 if __name__ == '__main__':
72     main()
```

Traceback (most recent call last):
File "C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1\Fazel-2.py", line 90, in <module>
 main()
File "C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1\Fazel-2.py", line 79, in main
 attempts, execution_time, execution_time_unit = crack_password(args.password, args.mode, search_space, args.n, args.k)
File "C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1\Fazel-2.py", line 12, in crack_password
 attempts, execution_time, execution_time_unit = model(length, search_space)
File "C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1\Fazel-2.py", line 26, in model
 execution_time = (attempts/2)/pow(10,(-12))
OverflowError: integer division result too large for a float
C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1>python Fazel-2.py e45 1 numbers 2 a
Attempts: 6.66801e+240
Execution Time: 1.1484537644813462e+245 years
C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1>python Fazel-2.py e45 2 numbers 3 a
Attempts: 6.66801e+240
Execution Time: 1.1484537644813462e+245 years
C:\Users\ASUS.PIESC\OneDrive\Desktop\SarvinCode\python\Data_Security_project1>python Fazel-2.py e45 3 numbers 7 abson
Attempts: 6.66801e+240
Execution Time: 1.1484537644813462e+245 years

:55

```
import argparse
import string

def crack_password(password, mode, search_space, length, k=None):
    attempts = 0

    #time required when computing 10^6 encryption per second
    execution_time = 0
    execution_time_unit = ""

    if mode == 1:
        attempts, execution_time, execution_time_unit = model(length,
search_space)
    elif mode == 2:
        attempts, execution_time, execution_time_unit = model(length-1,
search_space)
    elif mode == 3 and k!=None:
        attempts, execution_time, execution_time_unit = model(length-
(len(k)), search_space)
    else:
        attempts, execution_time, execution_time_unit = -1, 0, ""
    return attempts, execution_time , execution_time_unit

def model(length, search_space):
    bits = (pow(len(search_space), length)*8)
    attempts = pow(2, bits)
    #time required when computing 10^6 encryption per second
    execution_time = (attempts/2)/pow(10,(-12))
    execution_time_unit = ""
    if bits < 40:
        execution_time *= pow(10,3)
        execution_time_unit = 'milliseconds'
    elif bits < 46:
        execution_time_unit = 'seconds'
    elif bits < 52:
```

```

        execution_time /= 60
        execution_time_unit = 'minutes'
    elif bits < 57:
        execution_time /= (60*60)
        execution_time_unit = 'hours'
    elif bits < 60:
        execution_time /= (60*60*24)
        execution_time_unit = 'days'
    elif bits < 62:
        execution_time /= (60*60*24*7)
        execution_time_unit = 'weeks'
    elif bits < 65:
        execution_time /= (60*60*24*7*4)
        execution_time_unit = 'months'
    elif bits >= 65:
        execution_time /= (60*60*24*7*4*12)
        execution_time_unit = 'years'
    return attempts, execution_time, execution_time_unit

def main():
    parser = argparse.ArgumentParser(description="Password Cracker Tool")
    parser.add_argument("password", help="Password to crack")
    parser.add_argument("mode", type=int, choices=[1, 2, 3], help="Select
mode (1: Known first character, 2: Standard, 3: Known k characters)")
    parser.add_argument("search_space", choices=['numbers', 'lowercase',
'uppercase', 'character', 'all'], help="Select search space (numbers,
lowercase, uppercase, character, all)")
    parser.add_argument("n", type=int, help="Specify the length of
password")
    parser.add_argument("k", help="Specify a part of password")

    args = parser.parse_args()
    # password = input("Password: ")
    # n = int(input("Length of Password: "))
    # k = input("Part of Password: ")
    # mode = int(input("Mode: "))
    # search_space = string.printable

    if args.search_space == 'numbers':
        search_space = string.digits
    elif args.search_space == 'lowercase':
        search_space = string.ascii_lowercase
    elif args.search_space == 'uppercase':
        search_space = string.ascii_uppercase
    elif args.search_space == 'character':
        search_space = string.punctuation
    else:
        search_space = string.ascii_letters + string.digits +
string.punctuation

    attempts, execution_time, execution_time_unit =
crack_password(args.password, args.mode, search_space, args.n, args.k)

    # attempts, execution_time, execution_time_unit =
crack_password(password, mode, search_space, n, k)

    if attempts == -1:
        print(f"Password not found within the specified {args.k}
characters.")
    else:
        print(f"Attempts: {attempts:g}")

```

```
print(f"Execution Time: {execution_time} {execution_time_unit}")

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

توضیحات:

این کد یک ابزار برای تخمین تعداد تلاش‌های لازم برای کرک کردن یک رمز عبور است. ما از کتابخانه `argparse` برای دریافت ورودی‌ها استفاده می‌کنیم تا مشخصات مربوط به رمز عبور و نحوه تلاش برای کرک کردن آن را بدست آوریم.

این برنامه دارای یک تابع به نام `crack_password` است که توسط ورودی‌های مختلف و نوع حمله، تعداد تلاش‌ها و زمان انجام را برمی‌گرداند.

- تابع `main` تابع اصلی است که مقادیر ورودی از کاربر را با استفاده از `argparse` دریافت می‌کند و این مقادیر را به تابع `crack_password` ارسال می‌کند.

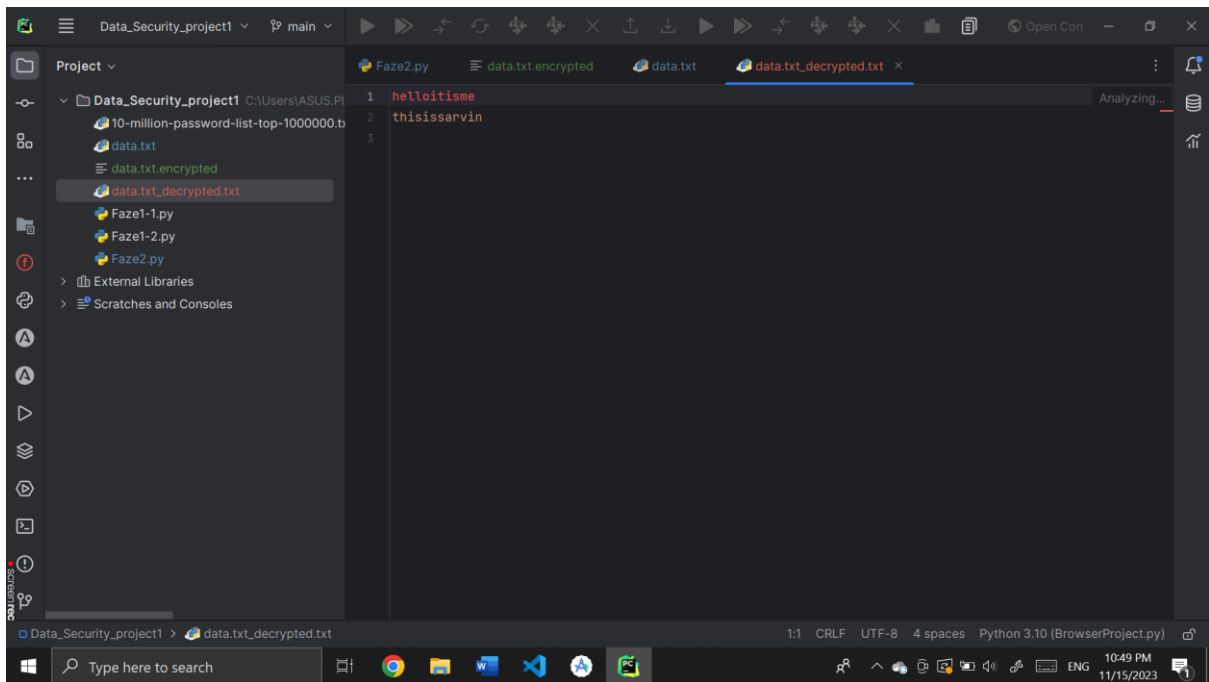
- `crack_password` با توجه به حالت‌های مختلف مشخص شده توسط کاربر، تعداد تلاش‌ها و زمان انجام را بر اساس محاسبات خود برمی‌گرداند.

- تابع `model` محاسبات مربوط به تعداد تلاش‌ها و زمان انجام را انجام می‌دهد و این مقادیر را برمی‌گرداند.

- در نهایت، اطلاعات دریافت شده از تابع `crack_password` نمایش داده می‌شود که شامل تعداد تلاش‌ها و زمان انجام لازم برای کرک کردن رمز عبور می‌باشد.

این ابزار برای بررسی میزان امنیت رمز عبور و تخمین تعداد تلاش‌های مورد نیاز برای کرک کردن آن از الگوریتم‌های مختلف استفاده می‌کند.

فاز ۲:



:كد

```
import argparse
import base64

from cryptography.fernet import Fernet
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes

def generate_key(password):
    # Derive a key from the password using PBKDF2
    salt = b'salt_' # Salt value to make the key unique
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32, # Length of the key
        salt=salt,
        iterations=100000, # Number of iterations
        backend=default_backend()
    )
    key = base64.urlsafe_b64encode(kdf.derive(password))

    return key

def encrypt_file(file_path, key):
    with open(file_path, 'rb') as file:
        data = file.read()

    f = Fernet(key)
    encrypted_data = f.encrypt(data)

    with open(f"{file_path}.encrypted", 'wb') as file:
        file.write(encrypted_data)

def decrypt_file(file_path, key):
    with open(file_path, 'rb') as file:
        encrypted_data = file.read()
```

```

f = Fernet(key)
decrypted_data = f.decrypt(encrypted_data)

with open(f"{file_path[:-10]}_decrypted.txt", 'wb') as file:
    file.write(decrypted_data)

def main():
    parser = argparse.ArgumentParser(description="File Encryption and
Decryption Tool")
    parser.add_argument("mode", type=int, choices=[1, 2], help="Select mode
(1: Encryption, 2: Decryption)")
    parser.add_argument("file_path", help="Path of the file")
    parser.add_argument("password", help="Password for
encryption/decryption")

    args = parser.parse_args()
    key = generate_key(args.password.encode())

    if args.mode == 1:
        encrypt_file(args.file_path, key)
        print("File encrypted successfully.")
    elif args.mode == 2:
        decrypt_file(args.file_path, key)
        print("File decrypted successfully.")
    else:
        print("Invalid mode. Please select 1 for encryption or 2 for
decryption.")

if __name__ == "__main__":
    main()

```

توضیحات:

این کد یک ابزار برای رمزنگاری و رمزگشایی فایل‌ها استفاده می‌شود. از کتابخانه‌های `base64`، `argparse` و `cryptography` در پایتون استفاده می‌کند.

۱. `generate_key`: این تابع یک کلید رمزنگاری ایجاد می‌کند. از الگوریتم `PBKDF2HMAC` استفاده می‌کند تا از رمز عبور یک کلید مناسب تولید کند. با استفاده از پارامترهای مختلف مانند `salt` (نمک)، `length` (طول کلید)، `iterations` (تعداد تکرارها)، یک کلید تصادفی و منحصر به فرد برای رمزنگاری ایجاد می‌کند.

۲. `encrypt_file`: این تابع یک فایل را با استفاده از یک کلید رمزگذاری مشخص (`key`) رمزنگاری می‌کند. ابتدا محتوای فایل را خوانده، سپس با استفاده از `Fernet` (یک رمزگذار)، محتوای رمزنگاری شده را ایجاد کرده و در یک فایل جدید با پسوند `encrypted` ذخیره می‌کند.

۳. `decrypt_file`: این تابع فایل رمزنگاری شده را با استفاده از یک کلید مشخص (`key`) رمزگشایی می‌کند. محتوای فایل را می‌خواند، سپس با استفاده از `Fernet`، محتوای رمزگشایی شده را بازیابی می‌کند و در یک فایل جدید با پسوند `decrypted.txt` ذخیره می‌کند.

۴. `main`: این تابع از `argparse` برای دریافت ورودی‌ها (حالت، مسیر فایل و رمز عبور) استفاده می‌کند. سپس با استفاده از توابع `generate_key`، `encrypt_file` و `decrypt_file`، فایل مورد نظر را رمزنگاری یا رمزگشایی می‌کند و پس از انجام عملیات، پیام مناسب را چاپ می‌کند.

این کد با استفاده از `Fernet` از کتابخانه `cryptography` برای رمزنگاری و رمزگشایی استفاده می‌کند و از الگوریتم PBKDF2 برای ایجاد کلید استفاده می‌کند تا اطمینان حاصل کند که کلید رمزگذاری بسیار قوی و امن است.