TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
----------o0o----------

AN TOÀN VÀ PHỤC HỒI DỮ LIỆU

BÀI TẬP PHÂN CÔNG 01.04

SINH VIÊN THỰC HIỆN
- Bùi Đăng Tuấn Kiệt - 20127218
- Lê Nguyễn Minh Quang - 20127295

Ngày 18 Tháng 10 Năm 2023

Đề bài

Xây dựng chương trình mã hóa & giải mã một file

i. Key mã hóa (password) do người dùng nhập vào được hash thành 1 chuỗi tối thiểu 100bit.

Với yêu cầu này, tụi em sử dụng phương pháp băm bậc 3 (SHA-3), hay còn gọi là **SHA-512**, tức là sẽ băm chuỗi mật khẩu thành 512 bit

```python
import hashlib

#InputPassword = "20127218"
#InputPassword2 = "20127295"

# Đọc mật khẩu
rf = open('readpassword.txt', 'r')
InputPassword = rf.read()
rf.close()

HashPassword = hashlib.sha512(InputPassword.encode()).hexdigest() # Dùng thuật toán băm SHA-512 để băm mật khẩu ra thành chuỗi 512 bit

#hashed_password_100_bits = hashed_password[:100 // 4] # Trong trường hợp mật khẩu cần đúng 100bit

def PrintOutEncryptedPassword(CustomPassword, EncryptedPassword):
    print("Password:", CustomPassword)
    print("Hashed Password:", EncryptedPassword)

PrintOutEncryptedPassword(InputPassword, HashPassword)

# Ghi mật khẩu đã hash vào file
wf = open('encryptedpassword.txt', 'w')
wf.write(HashPassword)
wf.close()
```

✓ 0.0s

```
Password: 20127218
Hashed Password: f3a72263955a4e4cbfc1188c1e15ac9ed74e74da34efa6befc9bd0ff28e2a34631991a0dca58bcb5686a60cd44d9c778ef94d71555fe663190e79bf491f1d01f
```

Ở đây sẽ có 2 file: File để đọc mật khẩu là *readpassword.txt* và file ghi mật khẩu đã băm là *encryptedpassword.txt*. Tại câu này, chương trình sẽ đọc mật khẩu từ file và dùng nó để thực hiện quy trình băm thành chuỗi các ký tự. Thủ tục băm sử dụng thư viện hashlib.

Sau khi băm thành chuỗi ký tự kỳ lạ này, chương trình tiến hành ghi nó vào file *encryptedpassword.txt* để lưu trữ.

Ở câu này có 1 cái tụi em chưa làm để tối ưu hóa tính an toàn bảo mật cho hash là thêm salt vào nó. Salt đóng vai trò như một cái IV của AES, tức là một chuỗi ngẫu nhiên để gắn vào phía trước của đoạn mã hash. Trong trường hợp tin tặc tấn công và dò ra danh sách mật khẩu, nếu hash thì thường sẽ cho ra kết quả giống nhau nên chúng sẽ sử dụng cách tấn công là "Dictionary Attacks", nghĩa là chúng sẽ lấy danh sách các mật khẩu phổ biến cùng với đoạn mã hash của chúng để dò ra mật khẩu của chúng ta. Ví dụ "123" sẽ là "a1b2c3", thì salt sẽ đóng vai trò gây nhiễu đoạn hash đó, nếu salt = "xyz" thì khi kết hợp với hash, nó sẽ ra "xyza1b2c3".

iii. Tuy cùng 1 key mã hóa nhưng mỗi lần mã sẽ /phải được một kết quả khác nhau (về cả chiều dài lẫn nội dung nhị phân).

Sau khi có được phần chuỗi băm tối thiểu 100 bit. Tiếp theo sẽ mã hóa chuỗi băm đó bằng thuật toán AES để tăng tính bảo mật.

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

# Khởi tạo khóa và IV (Initialization Vector)
key = b'Meow'   # 128-bit key
PaddedKey = Adding_pkcs7Padding (key,16)

iv = get_random_bytes(16)   # 128-bit IV
data = HashPassword.encode()
PaddedData = Adding_pkcs7Padding (data,16)

# Create AES cipher
cipher = AES.new(PaddedKey, AES.MODE_CBC, iv)

# Encrypt
EncryptedData = cipher.encrypt(PaddedData)

# Decrypt
decipher = AES.new(PaddedKey, AES.MODE_CBC, iv)
decrypted_data = Removing_pkcs7Padding(decipher.decrypt(EncryptedData))

print("Original Password:", InputPassword)
print("Hashed Password:", HashPassword)
print("Encrypted Data:", EncryptedData)
print("Decrypted Data:", decrypted_data)
```

Ở đây tụi em cũng sẽ dùng 1 thư viện hỗ trợ của Python là Crypto để sử dụng thuật toán mã hóa AES cũng như là một phần random và padding của nó. Để mã hóa từ một key nhiều lần ra nhiều kết quả khác nhau, tụi em cũng khởi tạo một iv, nó sẽ random chuỗi ký tự và mã hóa cùng với key.

Kết quả thực hiện được:

Original Password: 20127218

Hashed Password:
f3a72263955a4e4cbfc1188c1e15ac9ed74e74da34efa6befc9bd0ff28e2a34631991a0dca58bcb5686a60cd44d9c778ef94d71555fe663190e79bf491f1d01f

Encrypted Data: b'\xe0\xf6\xd7\xbb\x85\x82_^\xe8\x84Wa\xc6E\xd5r-\x17^\x1a\xc3\xf1\x9a%\x82\x13\x9b@Ar9\x85\xe3\xaa\xe6\xf2\xdbN\\\xcb\x15\xa6\xa9\x9a\xef\xb3Y\xbb\xdf*(]\x9f\x02\xc0S\xa3?\xce\x15R\x93w/\xbe\x97W\x91b\xa8\x96\xba\xa6I\xb8V\x91\xb7\x12w\xcb\xfd\xaf\xb2X

\x03k\x18MV\x1cR\xfe\xdb\xd6WH\xe3\x0c\x8dj\xdf\xafuc\xd2(\x97\xf7*o4\xdc\xdc\xf3\xc7n\xdf\xac\x07\x9e$\xdf\x94\x9dz0(\xa5LW.\xb2\x12wj\x82"\t7\xae\xd3+$'

Decrypted Data:
b'f3a72263955a4e4cbfc1188c1e15ac9ed74e74da34efa6befc9bd0ff28e2a34631991a0dca58bcb5686a60cd44d9c778ef94d71555fe663190e79bf491f1d01f'

v. Mỗi khi chương trình chạy phải hỏi và kiểm tra 1 mật khẩu "động" và nếu đúng thì mới tiếp tục chạy.

Ở câu này, tụi em tiếp tục sử dụng thư viên hỗ trợ sinh khóa otp, đó là **pyotp** đi kèm với **base64**.

Ở đây có biến shared_secret là một dạng chuỗi byte, nó sẽ được encode sang thành dạng base32 và nhờ đó có thể khởi tạo được mã otp.

Đây là một thư viện khá tiện lợi để thực hiện mã otp ngay trên python.

iv. Chương trình mã hóa có obfurcasted code để tăng độ an toàn, tránh không cho người khác dò ra cách làm và giải mã.

Theo như tìm hiểu về obfurcasted code, nó có nghĩa là thay đổi code của mình sao cho khi bị lộ code, người khác không để đọc được hoặc đọc không hiểu gì, rất khó hiểu, hay còn gọi là **làm rối code**
Cách làm cơ bản nhất cho điều này là thay đổi tên biến thành tên vô nghĩa và tụi em đã áp dụng cách làm đó. Ở bài này, tụi em sẽ obfurcast code trên như sau:

```python
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

# Khởi tạo khóa và IV (Initialization Vector)
b = b'Meow'   # 128-bit key
pk = p7 (b,16)

iv = get_random_bytes(16)   # 128-bit IV
d = x.encode()
pd = p7 (d,16)

# Create AES cipher
c = AES.new(pk, AES.MODE_CBC, iv)

# Encrypt
q = c.encrypt(pd)

# Decrypt
dph = AES.new(pk, AES.MODE_CBC, iv)
w = r7(dph.decrypt(q))

print("Original Password:", ip)
print("Hashed Password:", x)
print("Encrypted Data:", q)
print("Decrypted Data:", w)
```

```python
import hashlib

rf = open('readpassword.txt', 'r')
ip = rf.read()
rf.close()
x = hashlib.sha512(ip.encode()).hexdigest()

def pp(a, b):
    print("Password:", a)
    print("Hashed Password:", b)

pp(ip, x)

wf = open('encryptedpassword.txt', 'w')
wf.write(x)
wf.close()

def p7(d, bs):
    l = bs - len(d) % bs
    p = bytes([l] * l)
    return d + p

def r7(d):
    l = d[-1]
    if not all(x == l for x in d[-l:]):
        raise ValueError("Invalid")
    return d[:-l]
```

Như có thể thấy thì tụi em đa phần chỉ thay đổi tên biến và tên hàm thành những tên cực ngắn gọn và dường như vô nghĩa. Mặc dù đoạn code chưa được rút ngắn, người giỏi code đôi khi cũng mất một khoảng thời gian để có thể ngồi phân tích. Và tất nhiên, cách làm của tụi em chưa dùng đến tool hỗ trợ như jsobfuscate hay tham khảo thêm.

Full code: https://github.com/KevinZ2807/AN-PHDL

Nguồn tham khảo:

https://youtu.be/qgpsIBLvrGY?si=8UTX-41kR27jdlqK

https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html

https://stackoverflow.com/questions/12524994/encrypt-and-decrypt-using-pycrypto-aes-256

https://pypi.org/project/pycrypto/

https://github.com/pyauth/pyotp

https://pyauth.github.io/pyotp/

https://www.youtube.com/watch?v=-ps98ixrP1U

https://youtu.be/kfFacplFY4Y?si=w-iiGSY_sTmpfAd4
https://viblo.asia/p/obfuscated-code-trong-lap-trinh-XL6lANQg5ek