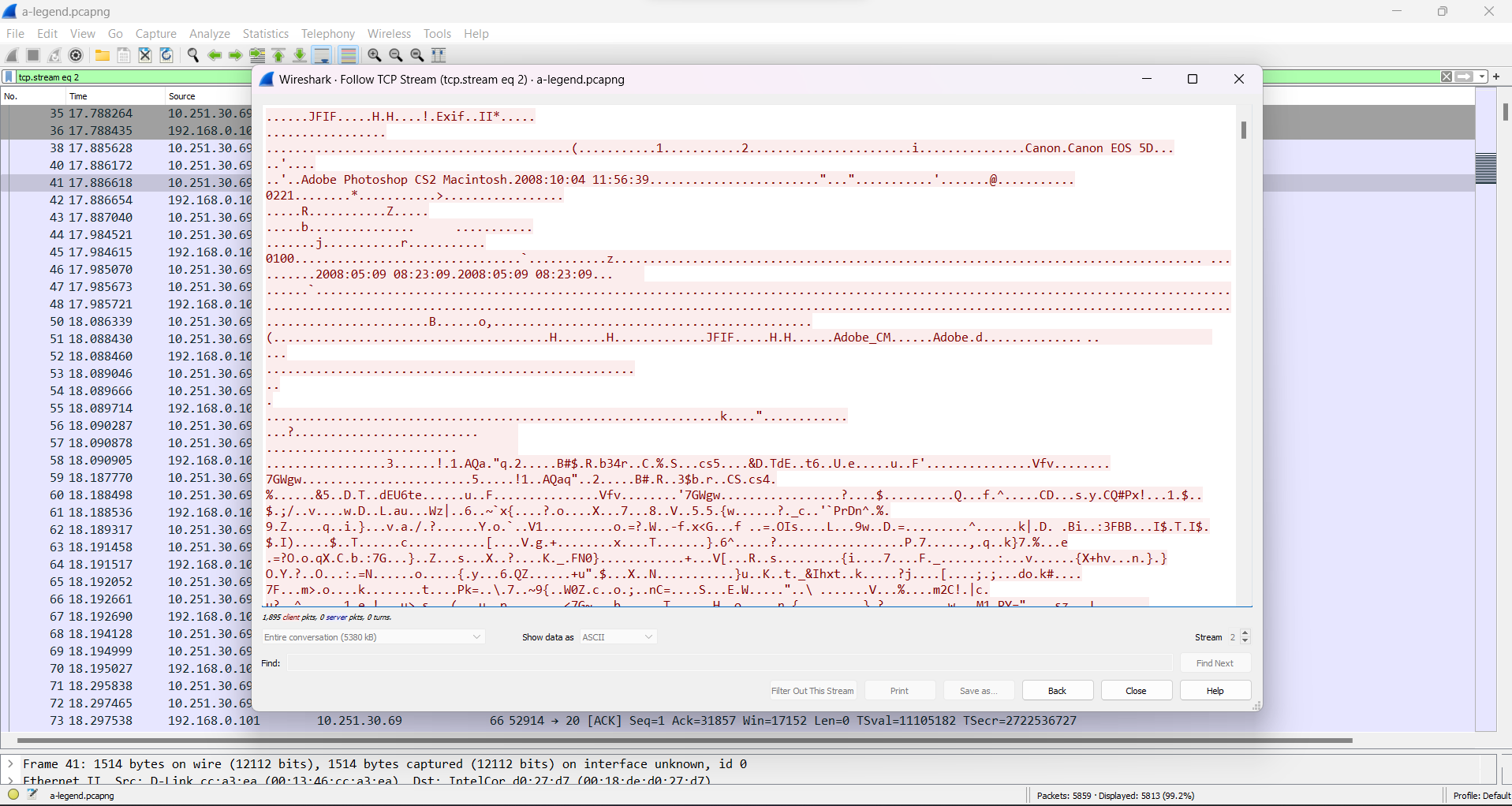
# Nguyễn Đình Hải \_ B21DCAT004

# WRITE UP ĐỘI TUYỂN SVATTT PTIT

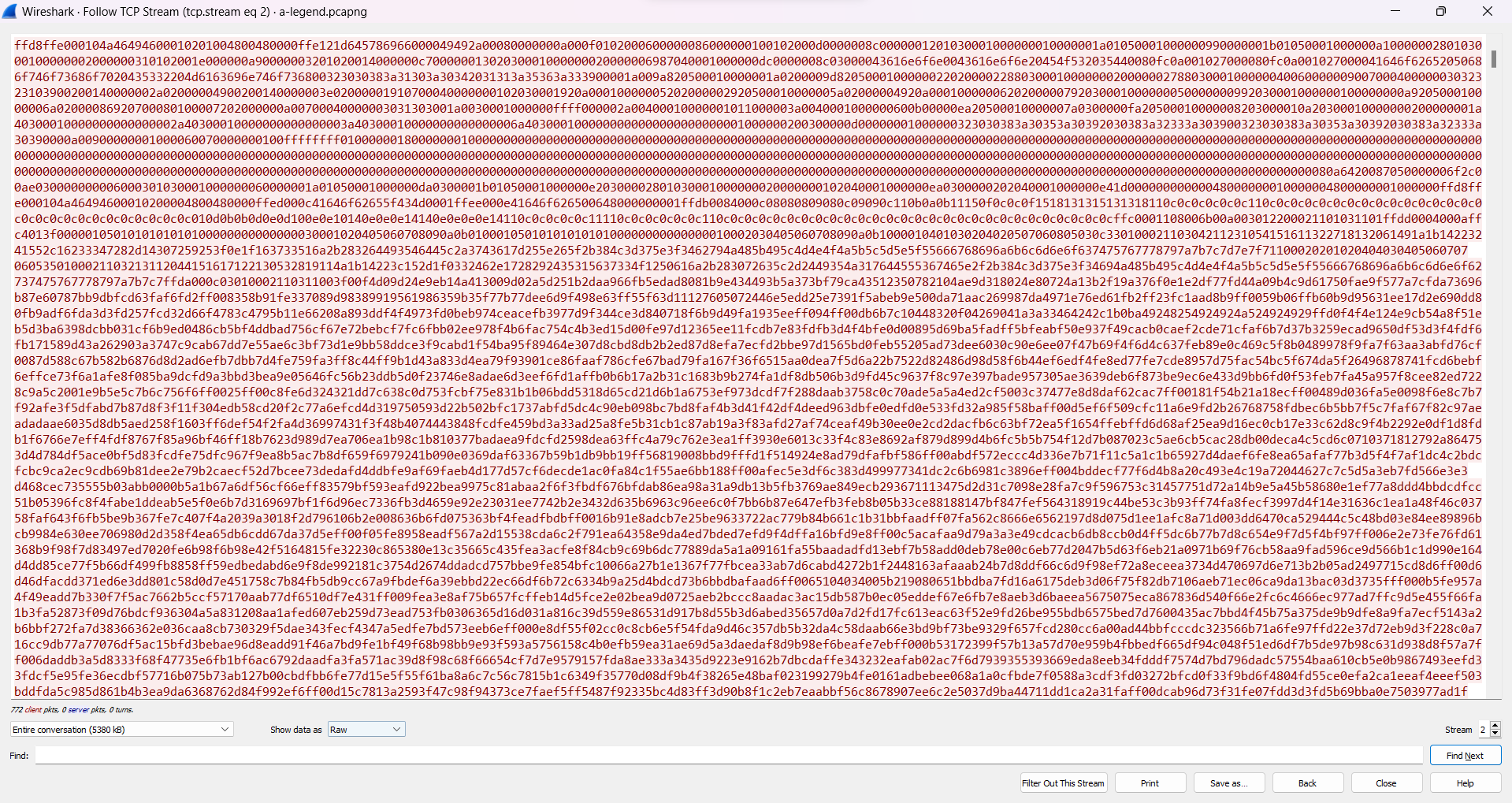
**Forensics**

**A Legend**

* Sau khi giải nén file ta nhận được 1 file zip và 1 file pcap. Khi check thử zip xong thì thấy đây là 1 file zip có mật khẩu và chúng ta không nhận được mật khẩu này từ để bài. Vậy nhiệm vụ của ta sẽ là tìm kiếm mật khẩu để extract ra được file pass.txt.

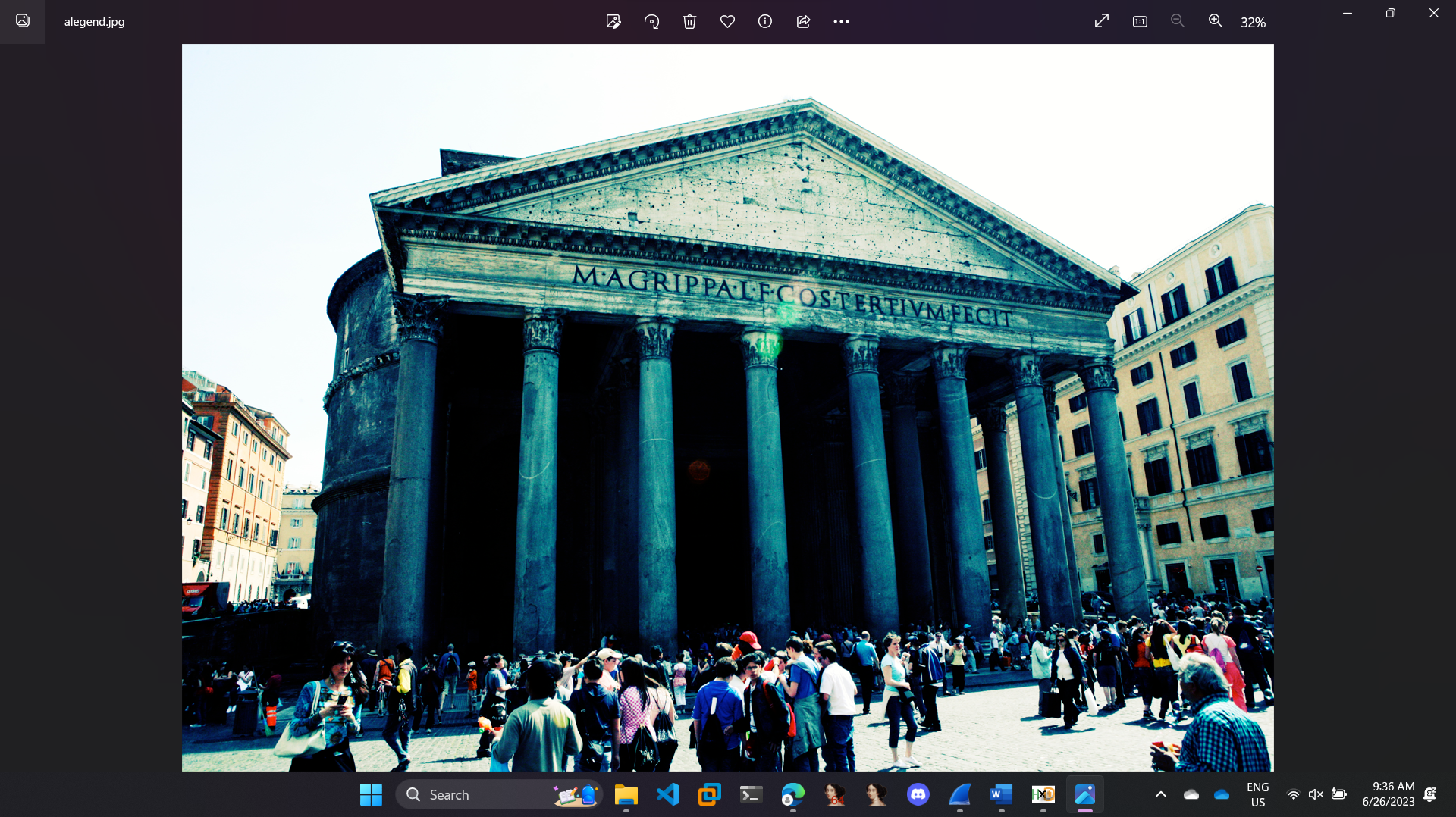


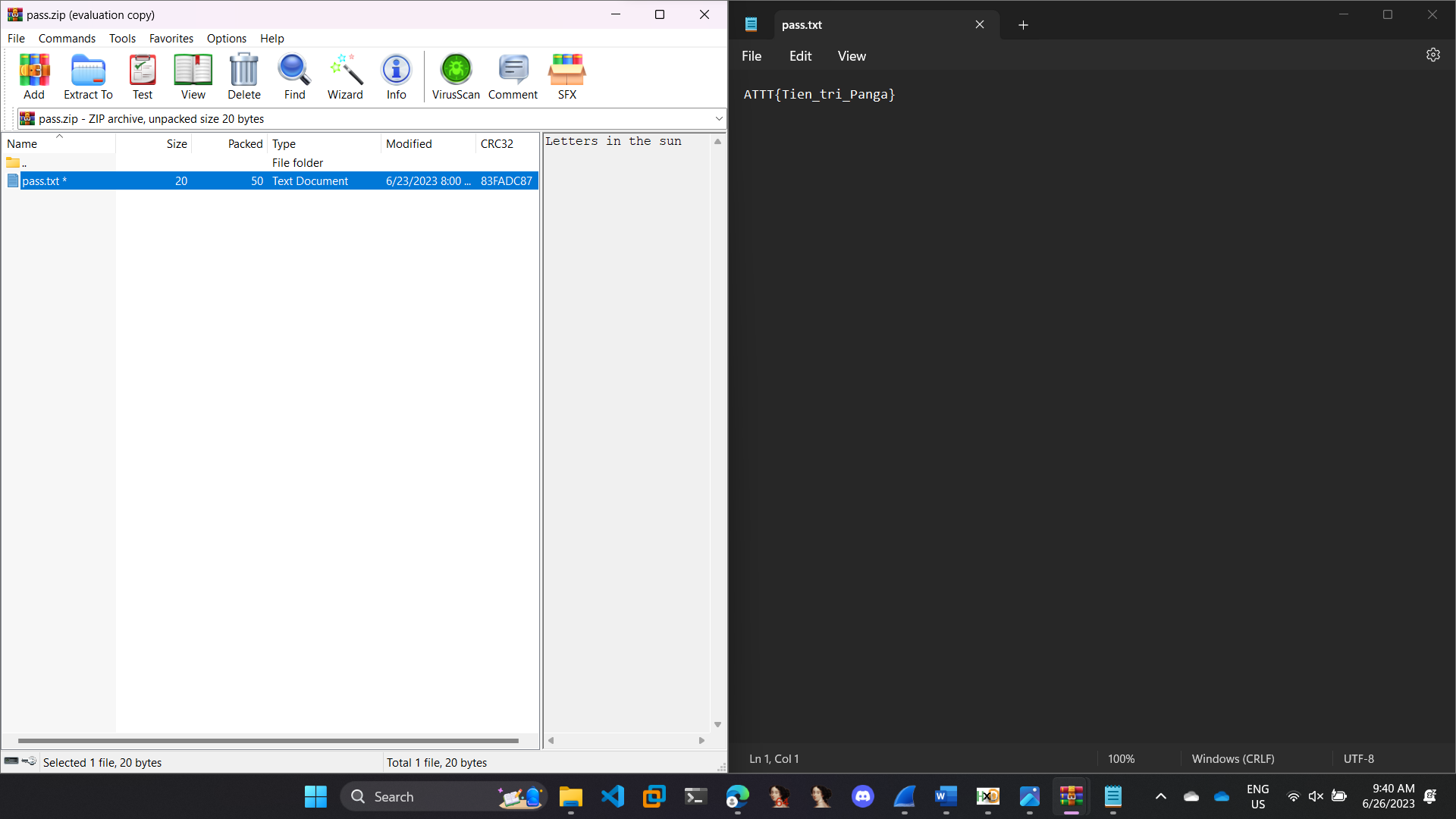
* Sau khi xem ta có thể nhận ra đây chính là file jpg. Vì vậy ta sẽ sử dụng HXD để xem ảnh đề bài cho.
* Chuyển nó sang dạng raw và copy sau đó paste sang HXD và lưu lại vào thành file jpg.





* Đây là ảnh mà ta nhận được. Sau khi xem xét dòng chữ trên tường của bức ảnh và thử với file zip thì nhận được "CO" là mật khẩu của file pass.zip, ngoài ra vệt sáng ở chỗ CO cũng là 1 dấu hiệu cho mật khẩu của file.





### **Flag**

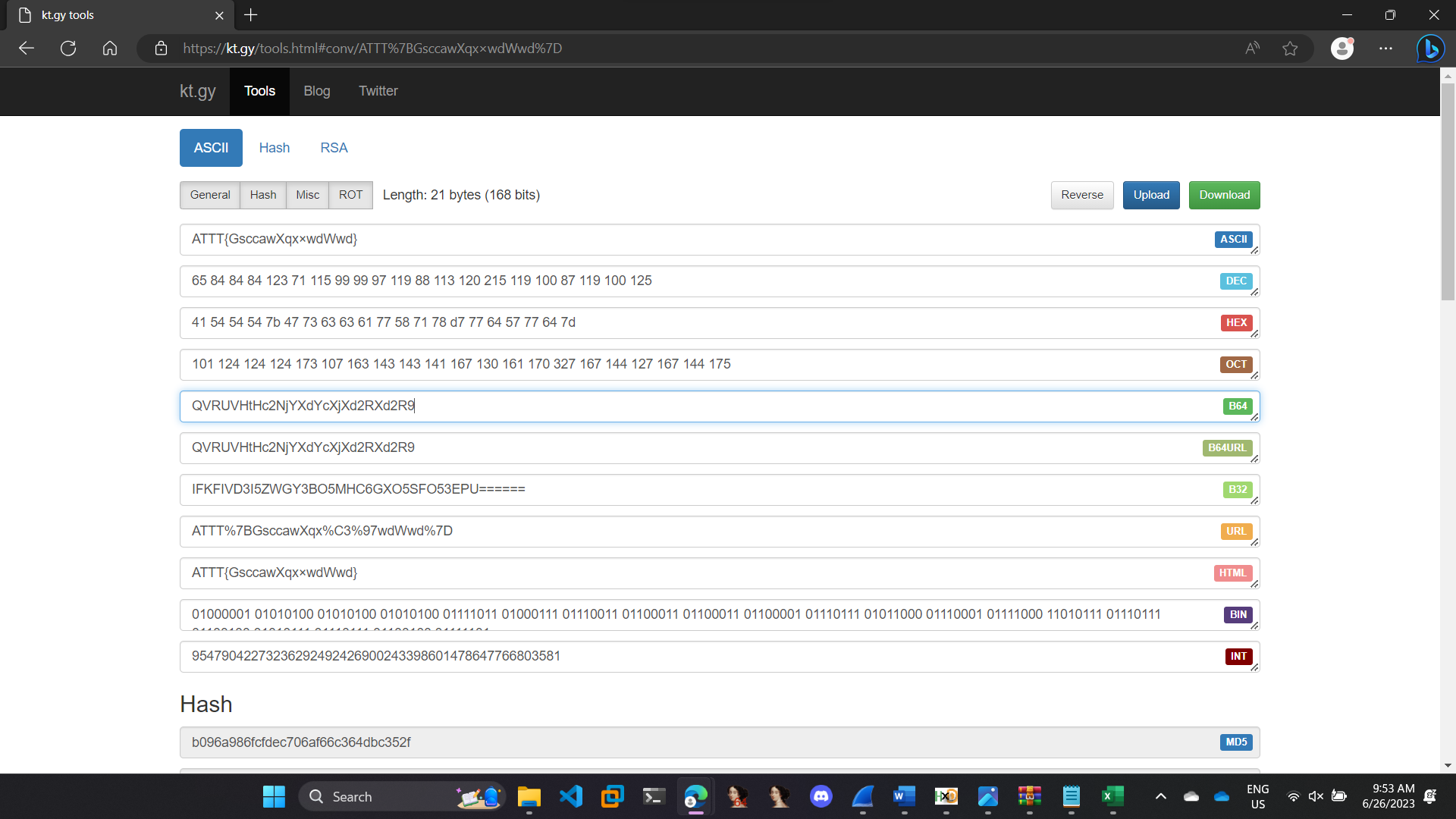
* Chúng ta sẽ có flag: ATTT{Tien\_tri\_Panga}

**List**

* Ta nhận được từ đề bài 1 file zip và có thể nhận ra các số thứ tự của file trong folder lần lượt là các chữ cái trong chuỗi cần tìm vì sau khi lướt qua ta có thể thấy đây hoàn toàn là các folder và file rỗng.



* Sau khi decode sang từ base64 ta thu được flag cần tìm.



### **Flag**

* Chúng ta sẽ có flag: ATTT{GsccawXqx×wdWwd}

**Crypto**

**Cry2**

* Đây là đoạn code của đề bài:

from Crypto.Util.number import bytes\_to\_long, getPrime

import random

FLAG = b'FLAG{????????????????????????????????????????????????}'

def gen\_params():

p = getPrime(1024)

g = random.randint(2, p - 2)

x = random.randint(2, p - 2)

h = pow(g, x, p)

return (p, g, h), x

def encrypt(pubkey):

p, g, h = pubkey

m = bytes\_to\_long(FLAG)

y = random.randint(2, p - 2)

s = pow(h, y, p)

return (g \* y % p, m \* s % p)

def main():

pubkey, \_ = gen\_params()

c1, c2 = encrypt(pubkey)

with open('out.txt', 'w') as f:

f.write(

f'p = {pubkey[0]}\ng = {pubkey[1]}\nh = {pubkey[2]}\n(c1, c2) = ({c1}, {c2})\n'

)

if \_\_name\_\_ == "\_\_main\_\_":

main()

* Nhìn vào các biến ở file output ta sẽ thấy đc các mối liên hệ của các biến

p = 140513998383733505878882484463832810400773610216464396082090299095443413923172235117457913756664550823719549662902317573376081431193542463845487697907226369167600481120163289426938922732100465539059713541309125384262046419688008790735321015697953751228026907656116801742757145900027205145531958711127892554959

g = 65556906335252739492138792165542310729847274604215491204377659859487941883331877324828668014945205688539538625347791245434017456867211045274632515903354031987751781443770705945360580464425457828577976457587248703877110979526996451487269992053016854914380845578885990371833497436103486601509329626084083968660

h = 15273528733818364088944825564931416580246804828551511408185386076550844914061247386402976630878359077550812761024414293357647284108322195874431618705318411581501804308347288296685298101428242754270584056686889362688314067600725185045055940503536863894910812191441369971393731456033180082689578691282489630975

(c1, c2) = (32930763137657591245671682661065378907876157470073696324706860924174151413813899545628267737923954182518355580308482582884395083978855627014399016379054293488253402225390954280471448716452508005409111284576745690532031769595122489477338312943614861813951717110271786478521584713391571602054127763725617981811, 7670001641599113231861941980345322847092796276245234617022952335365852477157226432010123907328990528181522443870120480148135852521715550020767705947636650531440958788351061564446985872556473800469658897282244374663426300248809407921886806704736303693384639428103469288766910384891882482020192891012087623322)

Liên hệ:

c1 = 32930763137657591245671682661065378907876157470073696324706860924174151413813899545628267737923954182518355580308482582884395083978855627014399016379054293488253402225390954280471448716452508005409111284576745690532031769595122489477338312943614861813951717110271786478521584713391571602054127763725617981811

g = 65556906335252739492138792165542310729847274604215491204377659859487941883331877324828668014945205688539538625347791245434017456867211045274632515903354031987751781443770705945360580464425457828577976457587248703877110979526996451487269992053016854914380845578885990371833497436103486601509329626084083968660

p = 140513998383733505878882484463832810400773610216464396082090299095443413923172235117457913756664550823719549662902317573376081431193542463845487697907226369167600481120163289426938922732100465539059713541309125384262046419688008790735321015697953751228026907656116801742757145900027205145531958711127892554959

c1 = g \* y % p

=> tìm được y

h = 15273528733818364088944825564931416580246804828551511408185386076550844914061247386402976630878359077550812761024414293357647284108322195874431618705318411581501804308347288296685298101428242754270584056686889362688314067600725185045055940503536863894910812191441369971393731456033180082689578691282489630975

s = pow(h, y\_value, p)

c2 = 7670001641599113231861941980345322847092796276245234617022952335365852477157226432010123907328990528181522443870120480148135852521715550020767705947636650531440958788351061564446985872556473800469658897282244374663426300248809407921886806704736303693384639428103469288766910384891882482020192891012087623322

c2 = m \* s % p

=> tìm được m

=> Flag

* Để thuận tiện cho việc tìm những số lớn và tính toán trong các biểu thức ta sử dụng z3solver để tìm ra giá trị của y từ đó có thể tìm m. Đây là script để tìm ra flag.

from z3 import \*

from Crypto.Util.number import \*

c1 = 32930763137657591245671682661065378907876157470073696324706860924174151413813899545628267737923954182518355580308482582884395083978855627014399016379054293488253402225390954280471448716452508005409111284576745690532031769595122489477338312943614861813951717110271786478521584713391571602054127763725617981811

g = 65556906335252739492138792165542310729847274604215491204377659859487941883331877324828668014945205688539538625347791245434017456867211045274632515903354031987751781443770705945360580464425457828577976457587248703877110979526996451487269992053016854914380845578885990371833497436103486601509329626084083968660

p = 140513998383733505878882484463832810400773610216464396082090299095443413923172235117457913756664550823719549662902317573376081431193542463845487697907226369167600481120163289426938922732100465539059713541309125384262046419688008790735321015697953751228026907656116801742757145900027205145531958711127892554959

solver = Solver()

y = Int('y')

solver.add(c1 == g \* y % p)

solver.add(y > 0)

if solver.check() == sat:

model = solver.model()

y\_value = model[y].as\_long()

print("y = ", y\_value)

else:

print("ko co")

solver = Solver()

h = 15273528733818364088944825564931416580246804828551511408185386076550844914061247386402976630878359077550812761024414293357647284108322195874431618705318411581501804308347288296685298101428242754270584056686889362688314067600725185045055940503536863894910812191441369971393731456033180082689578691282489630975

s = pow(h, y\_value, p)

c2 = 7670001641599113231861941980345322847092796276245234617022952335365852477157226432010123907328990528181522443870120480148135852521715550020767705947636650531440958788351061564446985872556473800469658897282244374663426300248809407921886806704736303693384639428103469288766910384891882482020192891012087623322

m = Int('m')

solver.add(c2 == m \* s % p)

solver.add(m > 0)

if solver.check() == sat:

model = solver.model()

m\_value = model[m].as\_long()

print(long\_to\_bytes(m\_value))

else:

print("ko co")

### **Flag**

 Halston  {  \Downloads\7FBB625F8793198942C61A50F3D0B49C  ﮫ 55ms  10:03 AM  

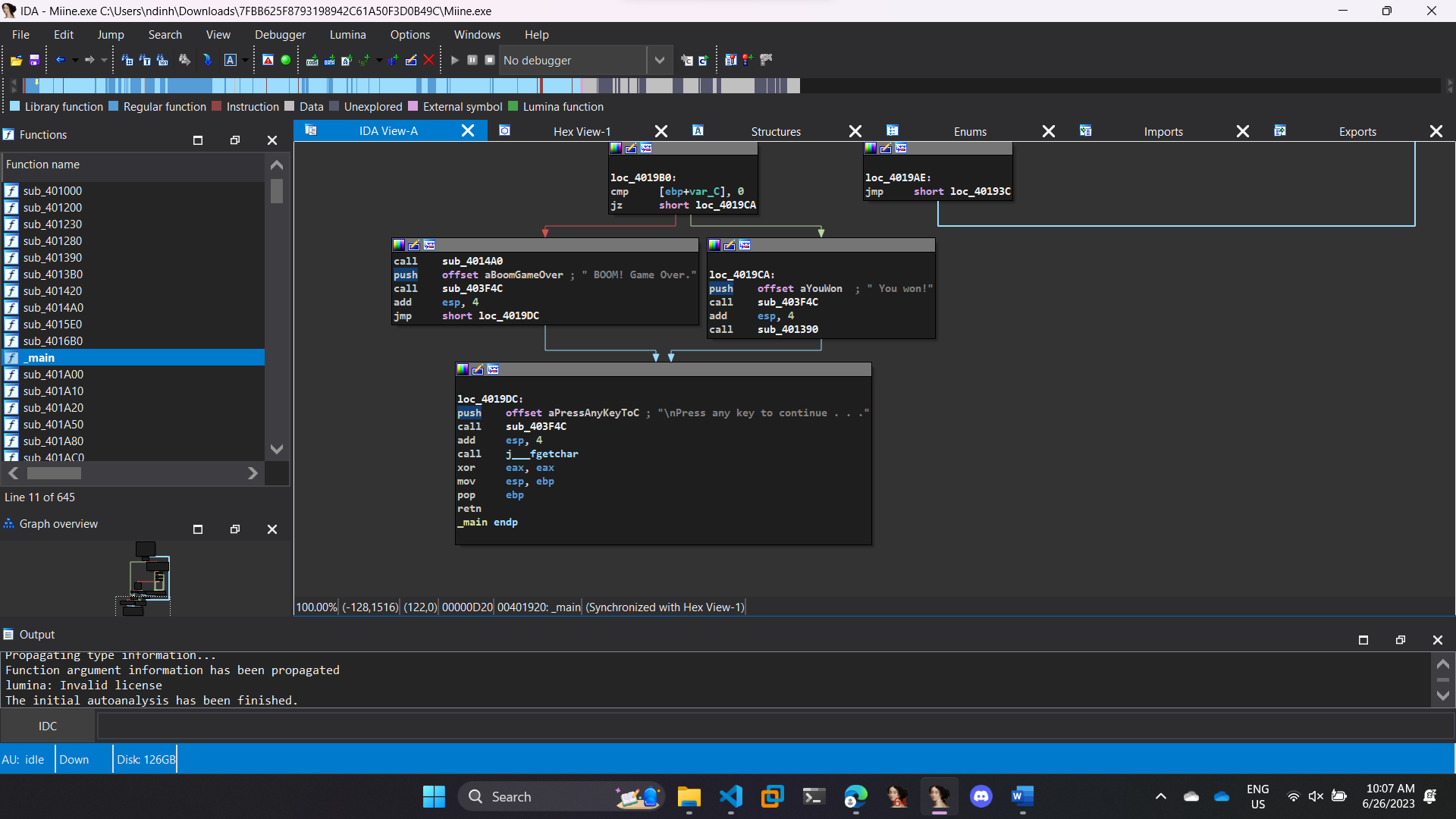
⚡ndinh ❯❯ python -u "c:\Users\ndinh\Downloads\7FBB625F8793198942C61A50F3D0B49C\s.py"

y = 82559313940155171724100424983174601161122093362972397116745780555709140804427460987213556652252846319464089764533428529236156695279989357277195698492344999203503864514371067496892119493463052458064881908151082303648037208440392614611827710667344748365855114065406607534592563320937172571268028959931995990121

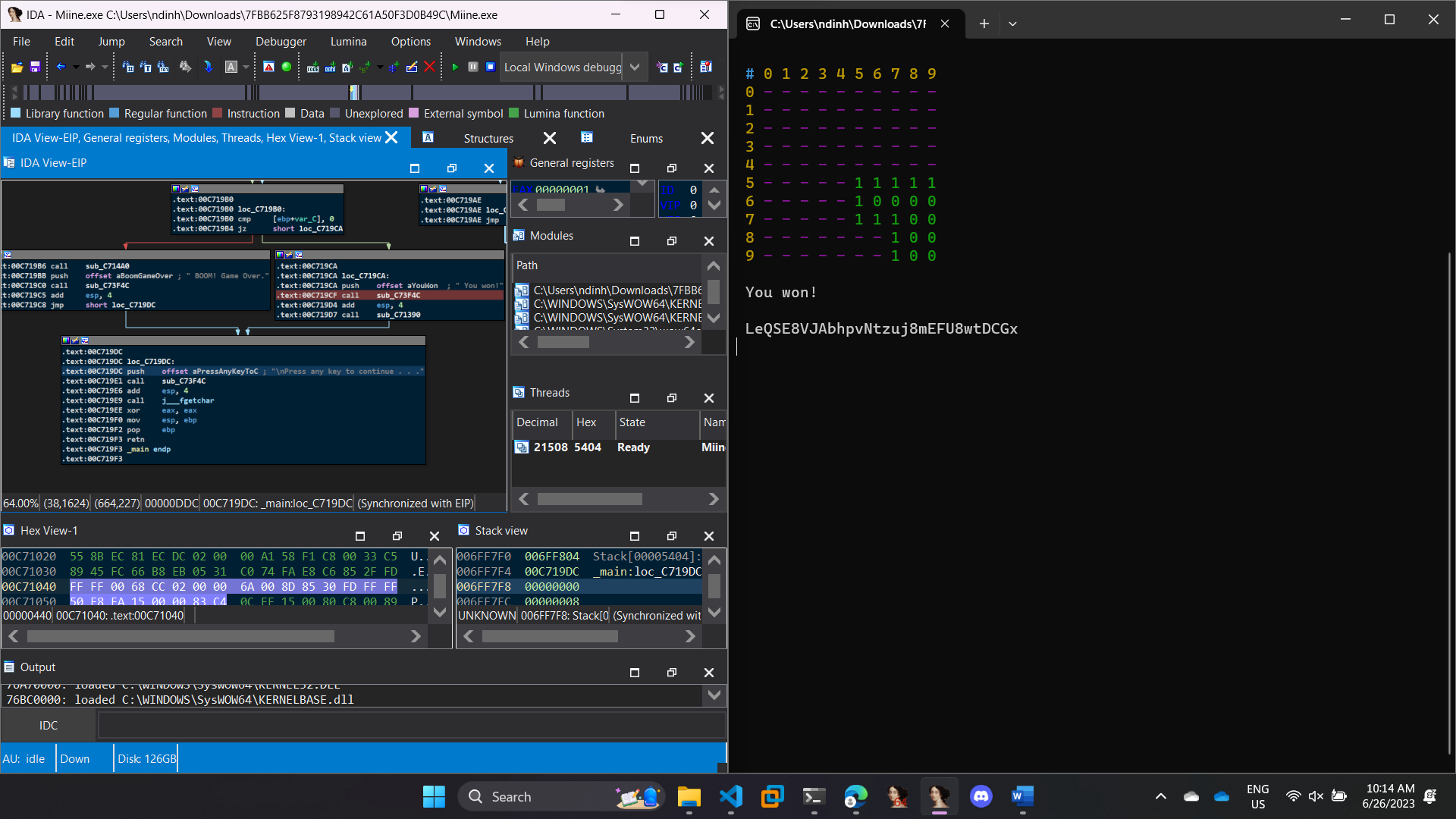
b'FLAG{s0me\_m4th\_1s\_3asy\_1f\_y0u\_kn0w\_4b0ut\_m0dular\_4r1thm3t1c}'

# Reverse

## Re1



* Sau khi check qua luồng thực thi ta sẽ nhìn thấy được hàm win và hiểu được rằng ta sẽ nhận được 1 chuỗi Str sau đó sẽ được build và decode qua 1 đoạn loằng ngoằng sẽ cho ra 1 chuỗi win. Trước tiên ta sẽ thử patch đến nó xem ta nhận được những gì. Các bước patch em sẽ skip qua vì chỉ cần call đến các hàm cần gọi điều hướng đến hàm win. Sau khi nhảy đến đó ta nhận được chuỗi như sau.



* Sau khi decode qua base58 dưới dạng Ripple ta thu được chuỗi Happy\_Minesweeper\_Game và thử sub với flag ATTT{Happy\_Minesweeper\_Game} nhưng không đúng.
* Sau khi xem qua thì có thể thấy hàm TlsCallback\_1 này sẽ xor các giá trị của chuỗi Str với 1 chuỗi nào đó. Vì vậy ta sẽ tạo function để check.

public TlsCallback\_1

.text:00C71100 TlsCallback\_1 proc near ; DATA XREF: .rdata:00C88150↓o

.text:00C71100

.text:00C71100 var\_2C = dword ptr -2Ch

.text:00C71100 var\_28 = dword ptr -28h

.text:00C71100 var\_24 = byte ptr -24h

.text:00C71100 var\_23 = byte ptr -23h

.text:00C71100 var\_22 = byte ptr -22h

.text:00C71100 var\_21 = byte ptr -21h

.text:00C71100 var\_20 = byte ptr -20h

.text:00C71100 var\_1F = byte ptr -1Fh

.text:00C71100 var\_1E = byte ptr -1Eh

.text:00C71100 var\_1D = byte ptr -1Dh

.text:00C71100 var\_1C = byte ptr -1Ch

.text:00C71100 var\_1B = byte ptr -1Bh

.text:00C71100 var\_1A = byte ptr -1Ah

.text:00C71100 var\_19 = byte ptr -19h

.text:00C71100 var\_18 = byte ptr -18h

.text:00C71100 var\_17 = byte ptr -17h

.text:00C71100 var\_16 = byte ptr -16h

.text:00C71100 var\_15 = byte ptr -15h

.text:00C71100 var\_14 = byte ptr -14h

.text:00C71100 var\_13 = byte ptr -13h

.text:00C71100 var\_12 = byte ptr -12h

.text:00C71100 var\_11 = byte ptr -11h

.text:00C71100 var\_10 = byte ptr -10h

.text:00C71100 var\_F = byte ptr -0Fh

.text:00C71100 var\_E = byte ptr -0Eh

.text:00C71100 var\_D = byte ptr -0Dh

.text:00C71100 var\_C = byte ptr -0Ch

.text:00C71100 var\_B = byte ptr -0Bh

.text:00C71100 var\_A = byte ptr -0Ah

.text:00C71100 var\_9 = byte ptr -9

.text:00C71100 var\_8 = byte ptr -8

.text:00C71100 var\_7 = byte ptr -7

.text:00C71100 var\_4 = dword ptr -4

.text:00C71100

.text:00C71100 push ebp

.text:00C71101 mov ebp, esp

.text:00C71103 sub esp, 2Ch

.text:00C71106 mov eax, \_\_\_security\_cookie

.text:00C7110B xor eax, ebp

.text:00C7110D mov [ebp+var\_4], eax

.text:00C71110 xor eax, eax

.text:00C71112 jz short near ptr loc\_C71114+1

.text:00C71114

.text:00C71114 loc\_C71114: ; CODE XREF: TlsCallback\_1+12↑j

.text:00C71114 jmp near ptr 20CCCF28h

.text:00C71114 ; ---------------------------------------------------------------------------

.text:00C71119 db 0FAh, 0C8h, 0

.text:00C7111C ; ---------------------------------------------------------------------------

.text:00C7111C test eax, eax

.text:00C7111E jz short loc\_C71125

.text:00C71120 jmp loc\_C711E6

.text:00C71125 ; ---------------------------------------------------------------------------

.text:00C71125

.text:00C71125 loc\_C71125: ; CODE XREF: TlsCallback\_1+1E↑j

.text:00C71125 mov [ebp+var\_24], 0

.text:00C71129 mov [ebp+var\_23], 0Dh

.text:00C7112D mov [ebp+var\_22], 33h ; '3'

.text:00C71131 mov [ebp+var\_21], 1Ch

.text:00C71135 mov [ebp+var\_20], 10h

.text:00C71139 mov [ebp+var\_1F], 62h ; 'b'

.text:00C7113D mov [ebp+var\_1E], 78h ; 'x'

.text:00C71141 mov [ebp+var\_1D], 0Dh

.text:00C71145 mov [ebp+var\_1C], 1Ah

.text:00C71149 mov [ebp+var\_1B], 1Dh

.text:00C7114D mov [ebp+var\_1A], 24h ; '$'

.text:00C71151 mov [ebp+var\_19], 24h ; '$'

.text:00C71155 mov [ebp+var\_18], 2Ch ; ','

.text:00C71159 mov [ebp+var\_17], 22h ; '"'

.text:00C7115D mov [ebp+var\_16], 38h ; '8'

.text:00C71161 mov [ebp+var\_15], 1Ch

.text:00C71165 mov [ebp+var\_14], 5Ch ; '\'

.text:00C71169 mov [ebp+var\_13], 11h

.text:00C7116D mov [ebp+var\_12], 60h ; '`'

.text:00C71171 mov [ebp+var\_11], 3Eh ; '>'

.text:00C71175 mov [ebp+var\_10], 0Ch

.text:00C71179 mov [ebp+var\_F], 2

.text:00C7117D mov [ebp+var\_E], 24h ; '$'

.text:00C71181 mov [ebp+var\_D], 4Dh ; 'M'

.text:00C71185 mov [ebp+var\_C], 0Ah

.text:00C71189 mov [ebp+var\_B], 0Bh

.text:00C7118D mov [ebp+var\_A], 32h ; '2'

.text:00C71191 mov [ebp+var\_9], 1Ch

.text:00C71195 mov [ebp+var\_8], 3Bh ; ';'

.text:00C71199 mov [ebp+var\_7], 3Ch ; '<'

.text:00C7119D push offset Str ; "LcKUC8JBInrfjVzdeb8qCHE8ozXSMt"

.text:00C711A2 call \_strlen

.text:00C711A7 add esp, 4

.text:00C711AA mov [ebp+var\_2C], eax

.text:00C711AD mov [ebp+var\_28], 0

.text:00C711B4 jmp short loc\_C711BF

.text:00C711B6 ; ---------------------------------------------------------------------------

.text:00C711B6

.text:00C711B6 loc\_C711B6: ; CODE XREF: TlsCallback\_1+E4↓j

.text:00C711B6 mov ecx, [ebp+var\_28]

.text:00C711B9 add ecx, 1

.text:00C711BC mov [ebp+var\_28], ecx

.text:00C711BF

.text:00C711BF loc\_C711BF: ; CODE XREF: TlsCallback\_1+B4↑j

.text:00C711BF mov edx, [ebp+var\_28]

.text:00C711C2 cmp edx, [ebp+var\_2C]

.text:00C711C5 jge short loc\_C711E6

.text:00C711C7 mov eax, [ebp+var\_28]

.text:00C711CA movsx ecx, [ebp+eax+var\_24]

.text:00C711CF mov edx, [ebp+var\_28]

.text:00C711D2 movsx eax, byte ptr Str[edx] ; "LcKUC8JBInrfjVzdeb8qCHE8ozXSMt"

.text:00C711D9 xor eax, ecx

.text:00C711DB mov ecx, [ebp+var\_28]

.text:00C711DE mov byte ptr Str[ecx], al ; "LcKUC8JBInrfjVzdeb8qCHE8ozXSMt"

.text:00C711E4 jmp short loc\_C711B6

.text:00C711E6 ; ---------------------------------------------------------------------------

.text:00C711E6

.text:00C711E6 loc\_C711E6: ; CODE XREF: TlsCallback\_1+20↑j

.text:00C711E6 ; TlsCallback\_1+C5↑j

.text:00C711E6 mov ecx, [ebp+var\_4]

.text:00C711E9 xor ecx, ebp ; StackCookie

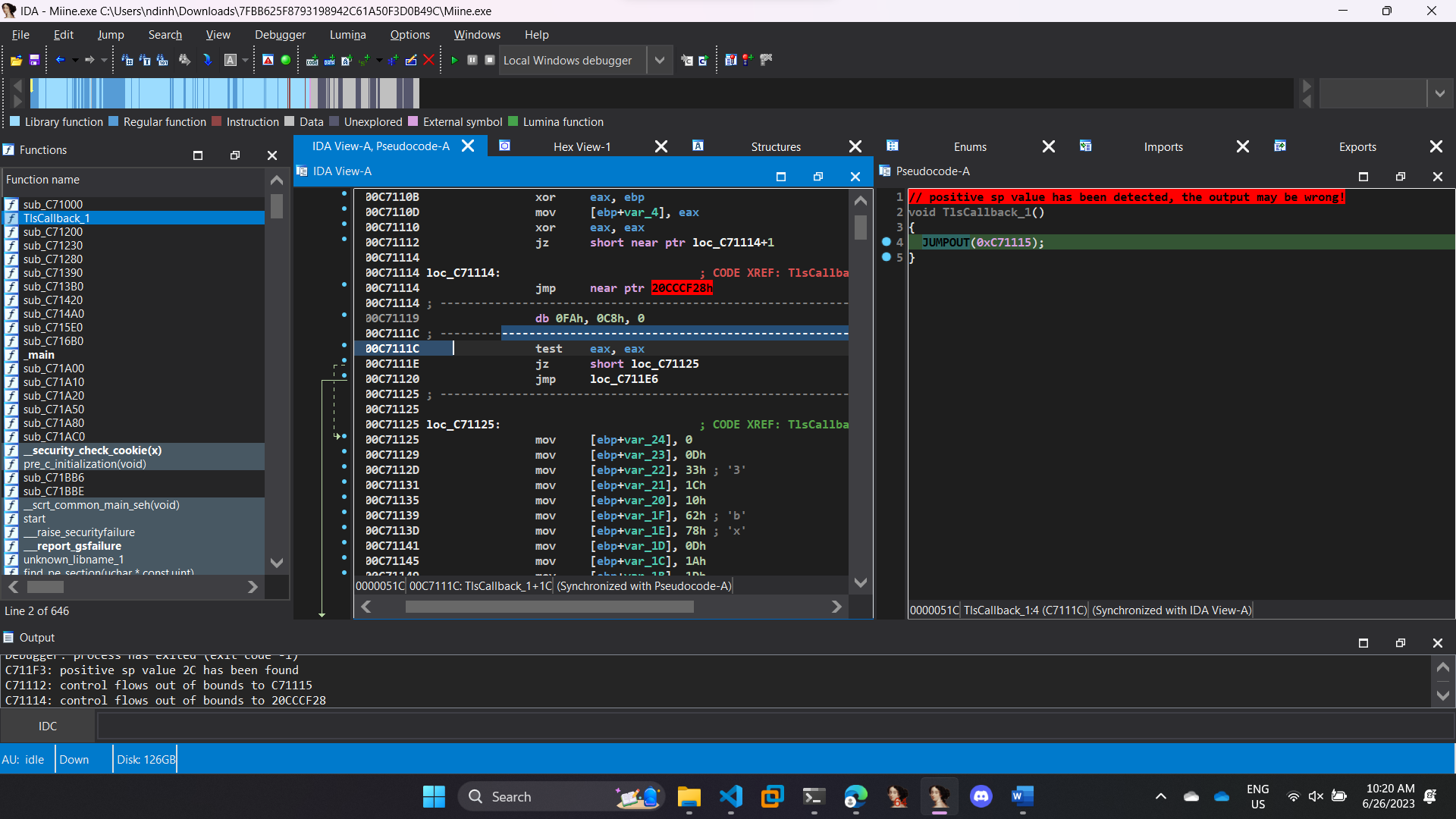
.text:00C711EB call @\_\_security\_check\_cookie@4 ; \_\_security\_check\_cookie(x)

.text:00C711F0 mov esp, ebp

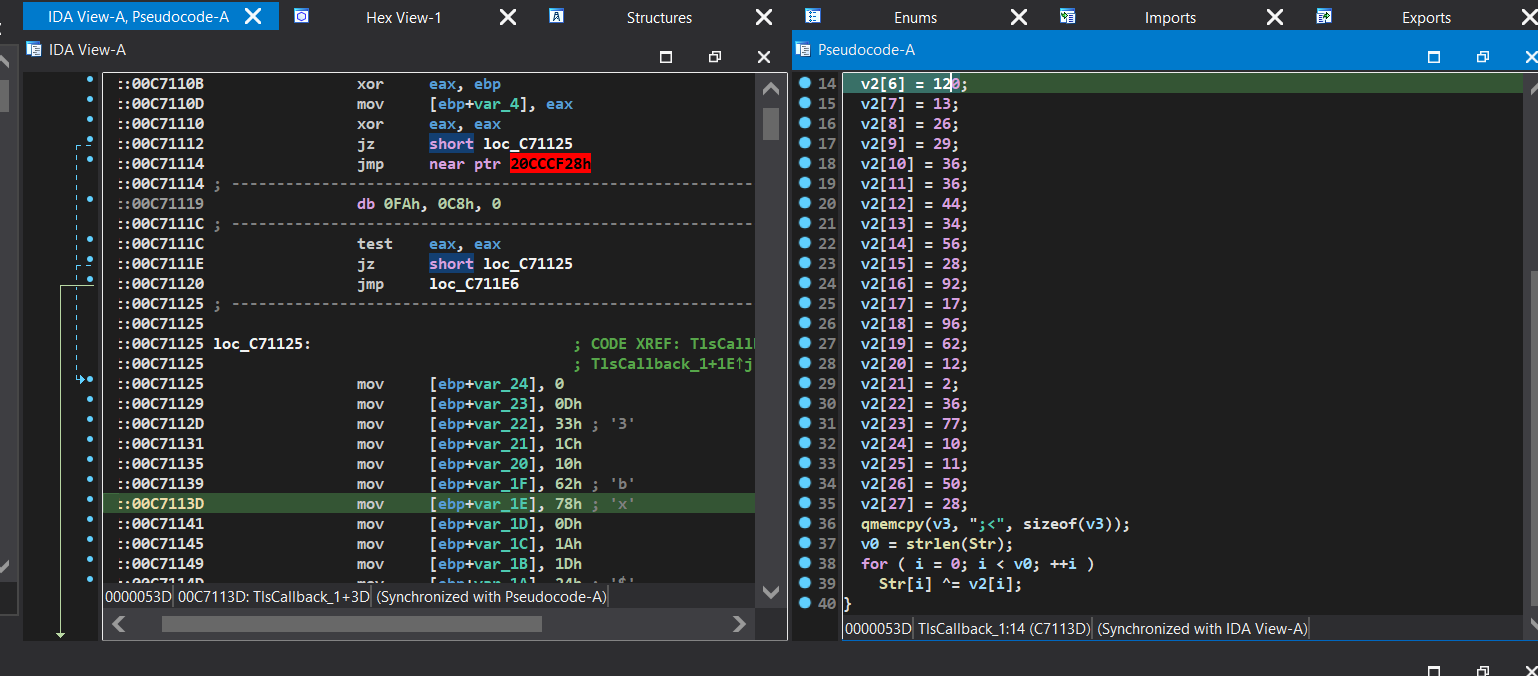
.text:00C711F2 pop ebp

.text:00C711F3 retn 0Ch

.text:00C711F3 TlsCallback\_1 endp

****

* Tại đây có thể thấy nó bị jumpout ngay khi bắt đầu khởi tạo biến và stack. Ta sẽ patch qua để cho nó tiếp tục thực hiện luồng code tiếp theo của mình.

****

* Sau khi patch xong ta đọc source c thì có thể thấy biến Str sẽ được thay đổi khi xor với v2.

#include <bits/stdc++.h>

using namespace std;

int main(){

signed int v0; // [esp+0h] [ebp-2Ch]

signed int i; // [esp+4h] [ebp-28h]

char v2[28]; // [esp+8h] [ebp-24h]

char Str[] = "LcKUC8JBInrfjVzdeb8qCHE8ozXSMt";

v2[0] = 0;

v2[1] = 13;

v2[2] = 51;

v2[3] = 28;

v2[4] = 16;

v2[5] = 98;

v2[6] = 120;

v2[7] = 13;

v2[8] = 26;

v2[9] = 29;

v2[10] = 36;

v2[11] = 36;

v2[12] = 44;

v2[13] = 34;

v2[14] = 56;

v2[15] = 28;

v2[16] = 92;

v2[17] = 17;

v2[18] = 96;

v2[19] = 62;

v2[20] = 12;

v2[21] = 2;

v2[22] = 36;

v2[23] = 77;

v2[24] = 10;

v2[25] = 11;

v2[26] = 50;

v2[27] = 28;

v0 = strlen(Str);

for ( i = 0; i < v0; ++i )

Str[i] ^= v2[i];

cout << Str;

return 0;

}

 Halston  {  \Downloads\7FBB625F8793198942C61A50F3D0B49C  ﮫ 53ms  10:25 AM  

⚡ndinh ❯❯ cd "c:\Users\ndinh\Downloads\7FBB625F8793198942C61A50F3D0B49C\" ; if ($?) { g++ t.cpp -o t } ; if ($?) { .\t }

LnxISZ2OSsVBFtBx9sXOOJaueqjOMt

* Cầm chuỗi trên để decrypt với hàm win với đoạn code sau.

#include <bits/stdc++.h>

using namespace std;

int check(int a1, int a2){

int v4 = a1 % a2;

for (int i = 1; i < a2; ++i )

{

if ( i \* v4 % a2 == 1 )

return i;

}

return -1;

}

int main(){

char s[] = "LnxISZ2OSsVBFtBx9sXOOJaueqjOMt";

int a2 = 5;

char flag[100];

int a3 = 8;

int cnt = strlen(s) + 1;

int x = 26;

int v3 = check(a2, x);

for( int i = 0; i < cnt; i++){

int v7 = s[i];

if ( v7 < 65 || v7 > 90 )

{

if ( v7 >= 97 && v7 <= 122 )

v7 = v3 \* ((v7 - 97 - a3 + 26) % 26) % 26 + 97;

}

else

{

v7 = v3 \* ((v7 - 65 - a3 + 26) % 26) % 26 + 65;

}

flag[i] = v7;

}

cout << flag;

return 0;

}

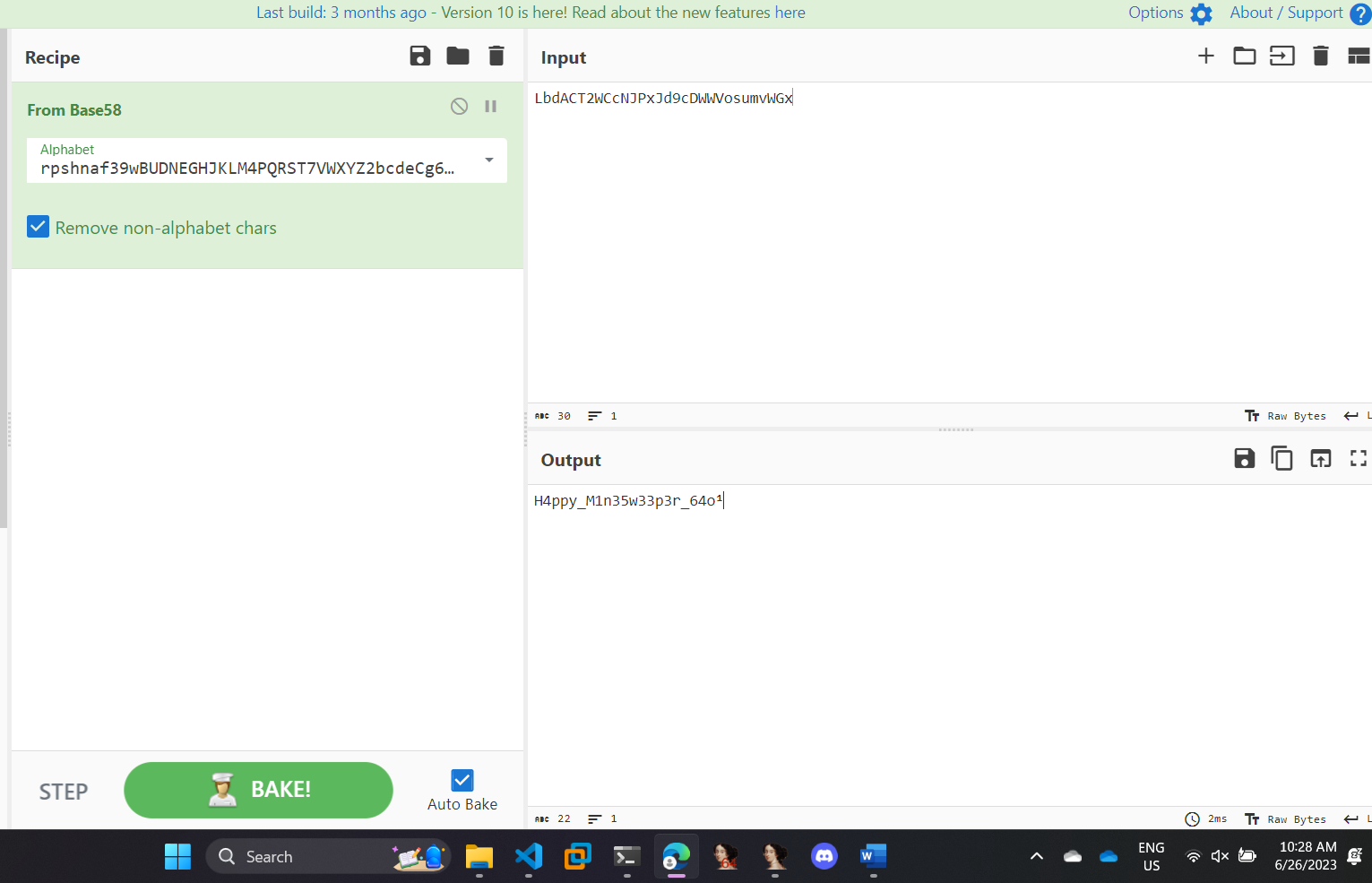
* Ta thu được chuỗi base58 mới.

 Halston  {  \Downloads\7FBB625F8793198942C61A50F3D0B49C  ﮫ 2.129s  10:25 AM  

⚡ndinh ❯❯ cd "c:\Users\ndinh\Downloads\7FBB625F8793198942C61A50F3D0B49C\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

LbdACT2WCcNJPxJd9cDWWVosumvWGx

• Decode nó với base58 ripple thì có thể thấy 2 byte cuối bị lem và sau khi nhìn qua các chữ khác cũng có thể thấy byte cuối sẽ là 3 còn byte kề nó sẽ là m

****

### **Flag**

* Sau khi thử sub ta thu được flag ATTT{H4ppy\_M1n35w33p3r\_64m3} thì đã correct