

BÁO CÁO ĐỒ ÁN 3

KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ



Mục lục

[I. Thông tin nhóm 2](#_Toc515518765)

[II. Phân công công việc 2](#_Toc515518766)

[III. Mô tả từng bước thuật toán phát sinh key 3](#_Toc515518767)

[1. File Crackme 1.1 3](#_Toc515518768)

[2. File Crackme 2.1 5](#_Toc515518769)

[3. File Crackme 2.2 7](#_Toc515518770)

[1. Tiền xử lý 7](#_Toc515518771)

[2. Chương trình 1 9](#_Toc515518772)

[3. Chương trình 2 11](#_Toc515518773)

[4. Chương trình 3 13](#_Toc515518774)

[5. Chương trình keygen 15](#_Toc515518775)

[4. File crackme 3.1 16](#_Toc515518776)

[5. Bài tập trên trang microcorruption.com 21](#_Toc515518777)

[1. Level Tutorial 21](#_Toc515518778)

[2. Level New Orleans 22](#_Toc515518779)

[3. Sydney 22](#_Toc515518780)

[4. Hanoi 23](#_Toc515518781)

[5. Cusco 23](#_Toc515518782)

[6. Whitehorse 24](#_Toc515518783)

[IV. Tài liệu tham khảo 25](#_Toc515518784)

# Thông tin nhóm

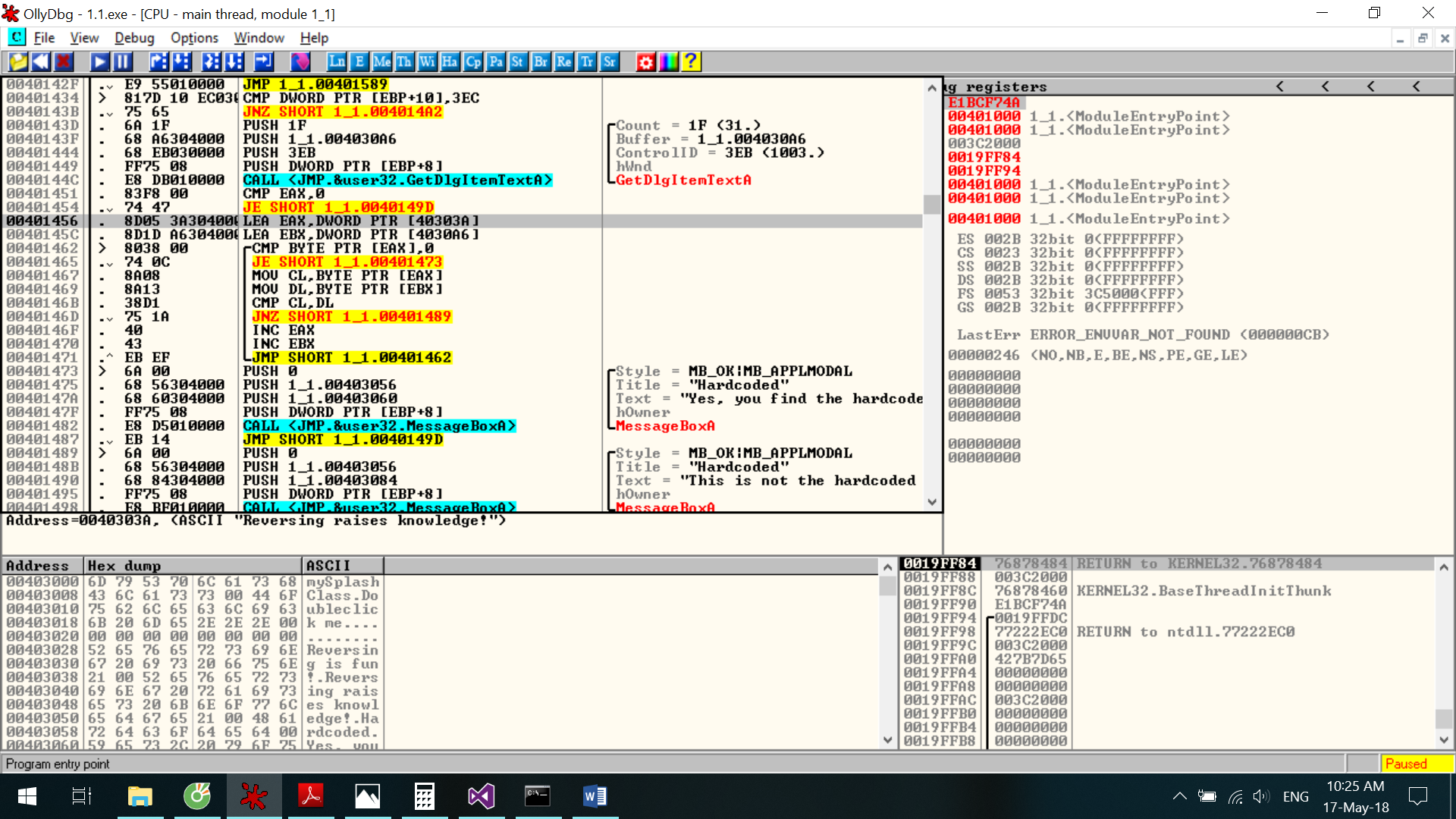
* MSSV: 1612348 Họ tên: Lý Vĩnh Lợi
* MSSV: 1612362 Họ tên: Trần Văn Lượn
* MSSV: 1612756 Họ tên: Nguyễn Hữu Trường

# Phân công công việc

|  |  |  |
| --- | --- | --- |
| **Công việc** | **Người phụ trách** | **Mức độ hoàn thành (%)** |
| Crackme 1.1 | Trường | 100 |
| Crackme 2.1 | Trường | 100 |
| Crackme 2.2 | Lợi | 100 |
| Crackme 3.1 | Lượn | 100 |
| Bài tập trên microcorruption | Lợi | 100 |

# Mô tả từng bước thuật toán phát sinh key

## File Crackme 1.1



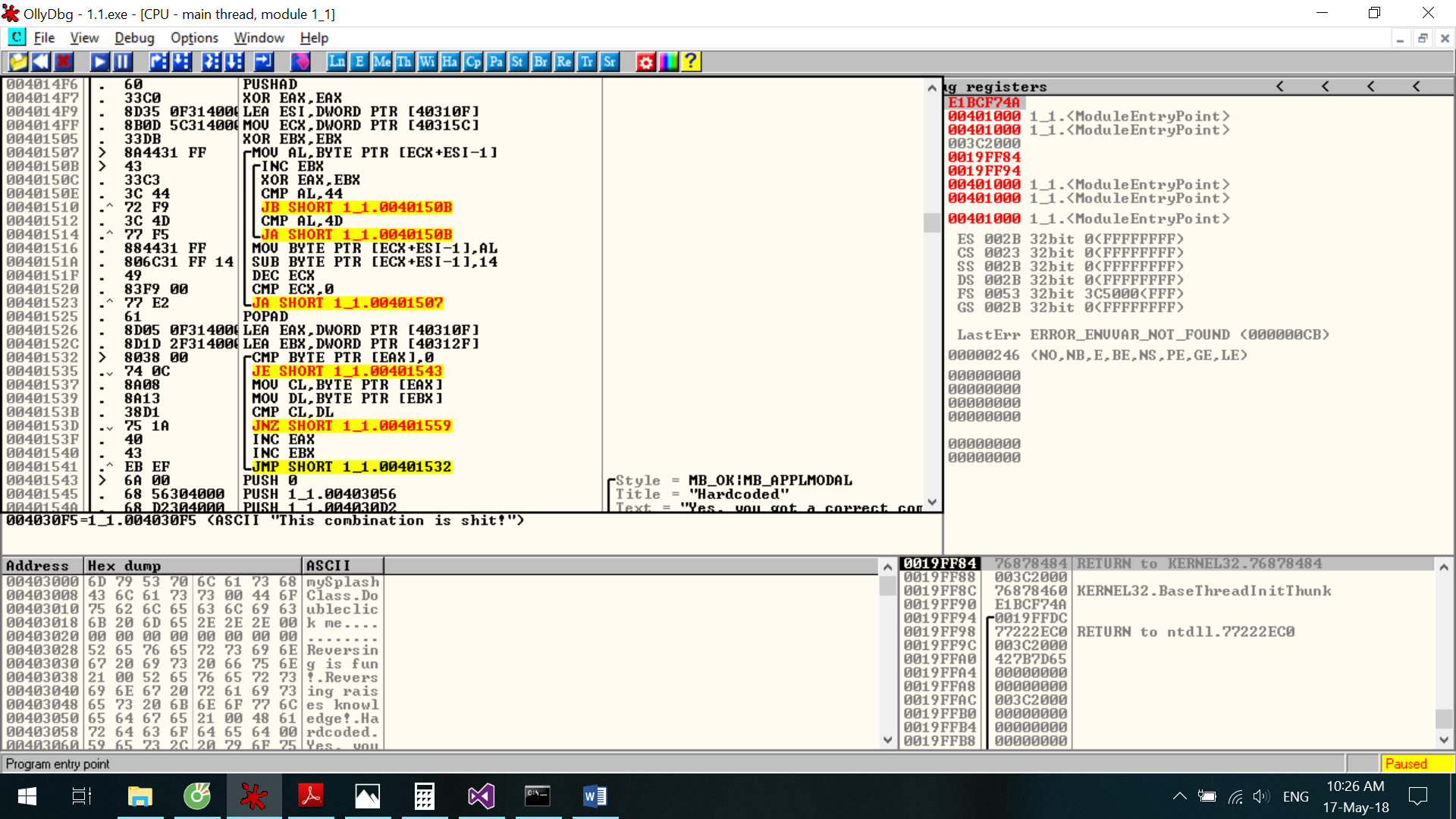
1.Check Hardcoded

EAX chứa chuỗi khóa cố định

EBX chứa chuỗi hardcoded người dùng nhập vào

Nếu EAX=EBX thì chuỗi hardcoded đúng

Kết quả: EBX = “Reversing raiser knowledge!”



ESI: chuỗi name người dùng nhập

int ECX = ESI.length();

int EBX= 0;

char EAX;

while (ECX > 0){

EAX = ESI[ECX - 1];

**Loop:** {

EBX++;

EAX = EAX ^ EBX;

if (EAX < 68 || EAX>77)

goto loop;

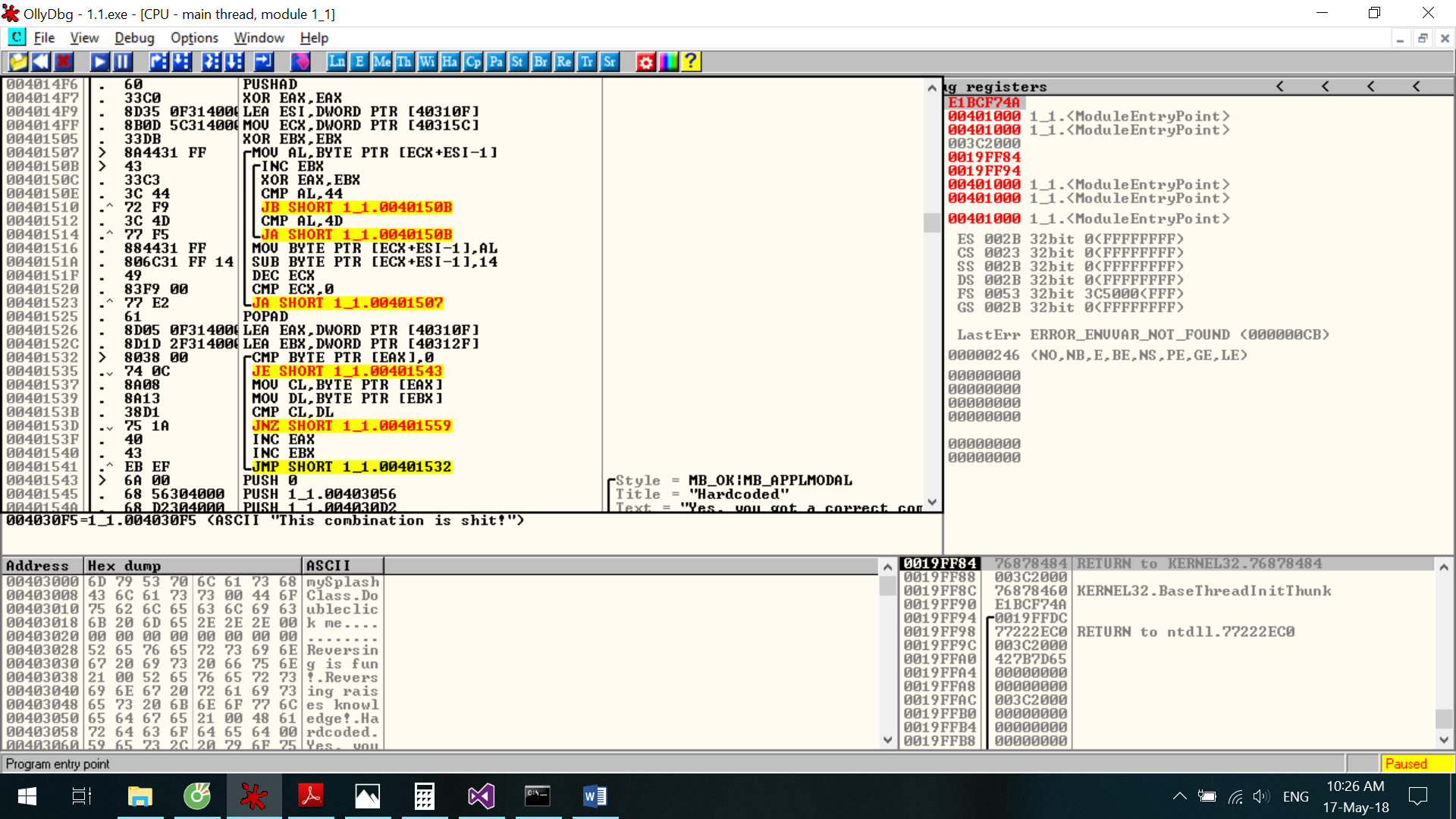
}

ESI[ECX - 1] = EAX – 20;

ECX--;

}

2. Tạo serial từ name người dùng nhập vào



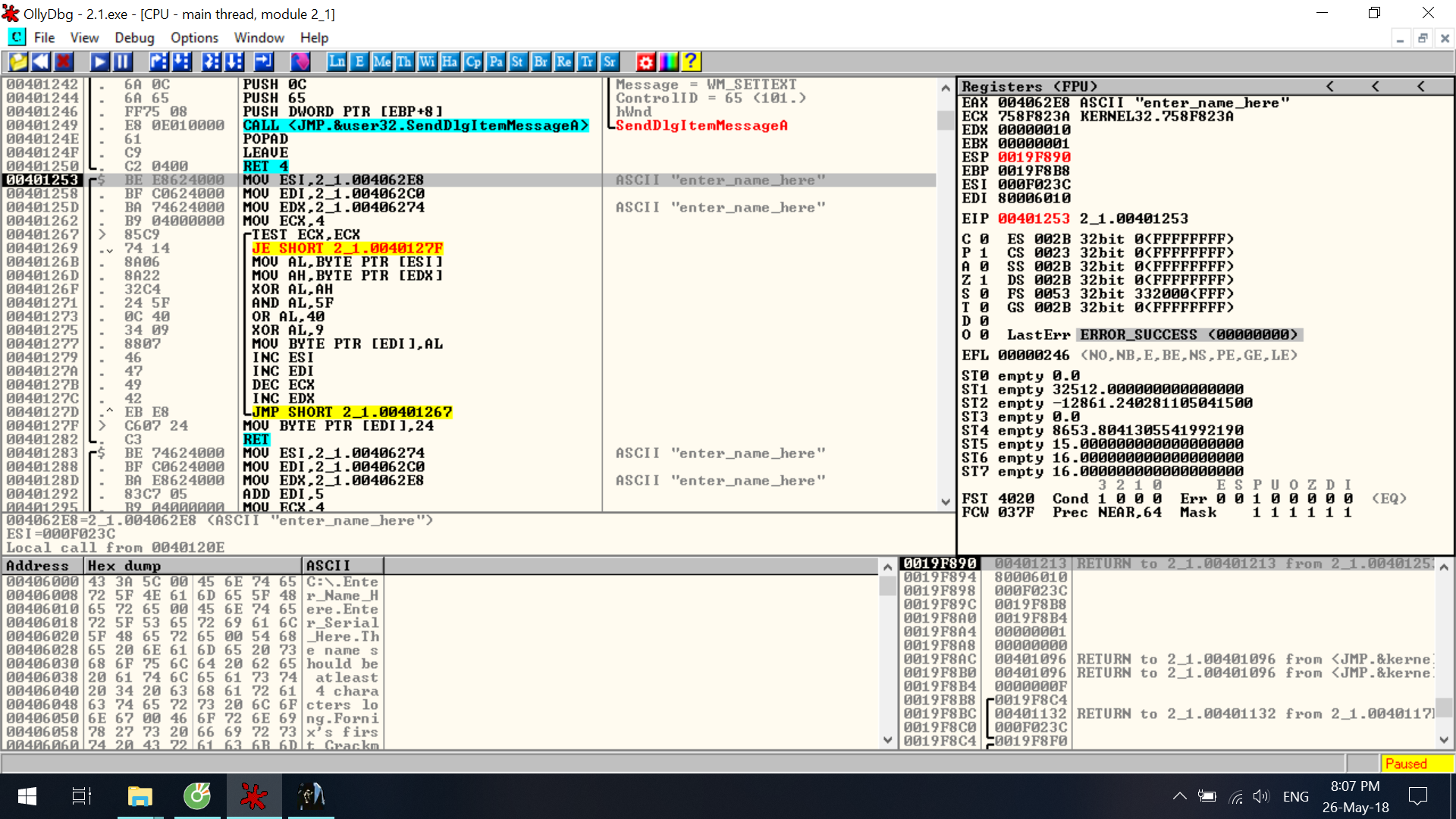
3. Check serial do thuật toán tạo ra và serial do người dùng nhập vào

EAX: chuỗi serial do thuật toán tạo ra

EBX: chuỗi serial do người dùng nhập vào

Nếu EAX = EBX thì người dùng nhập đúng.

## File Crackme 2.1

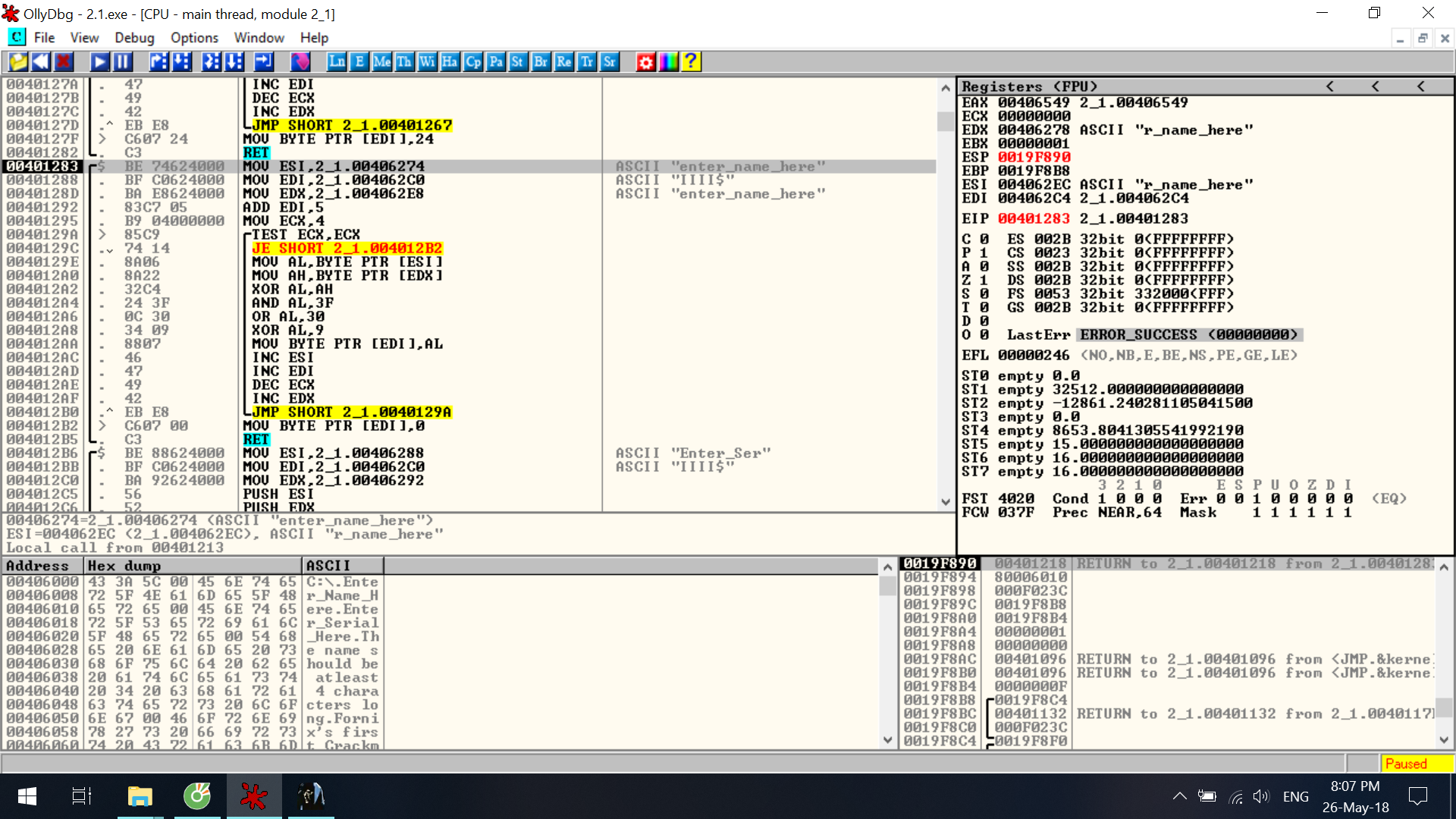


4. Sinh chuỗi khóa từ chuỗi tên do người dùng nhập vào (hàm 1).

EDI và EDX đều trỏ đến chuỗi tên người dùng nhập vào.

**Đoạn code chính:** EDI = (((ESI ^ EDX) & 95) | 64) | 9(dec) = 64 | 9 = ‘I’

* **Chuỗi khóa do hàm trên tạo ra là: “IIII$”**

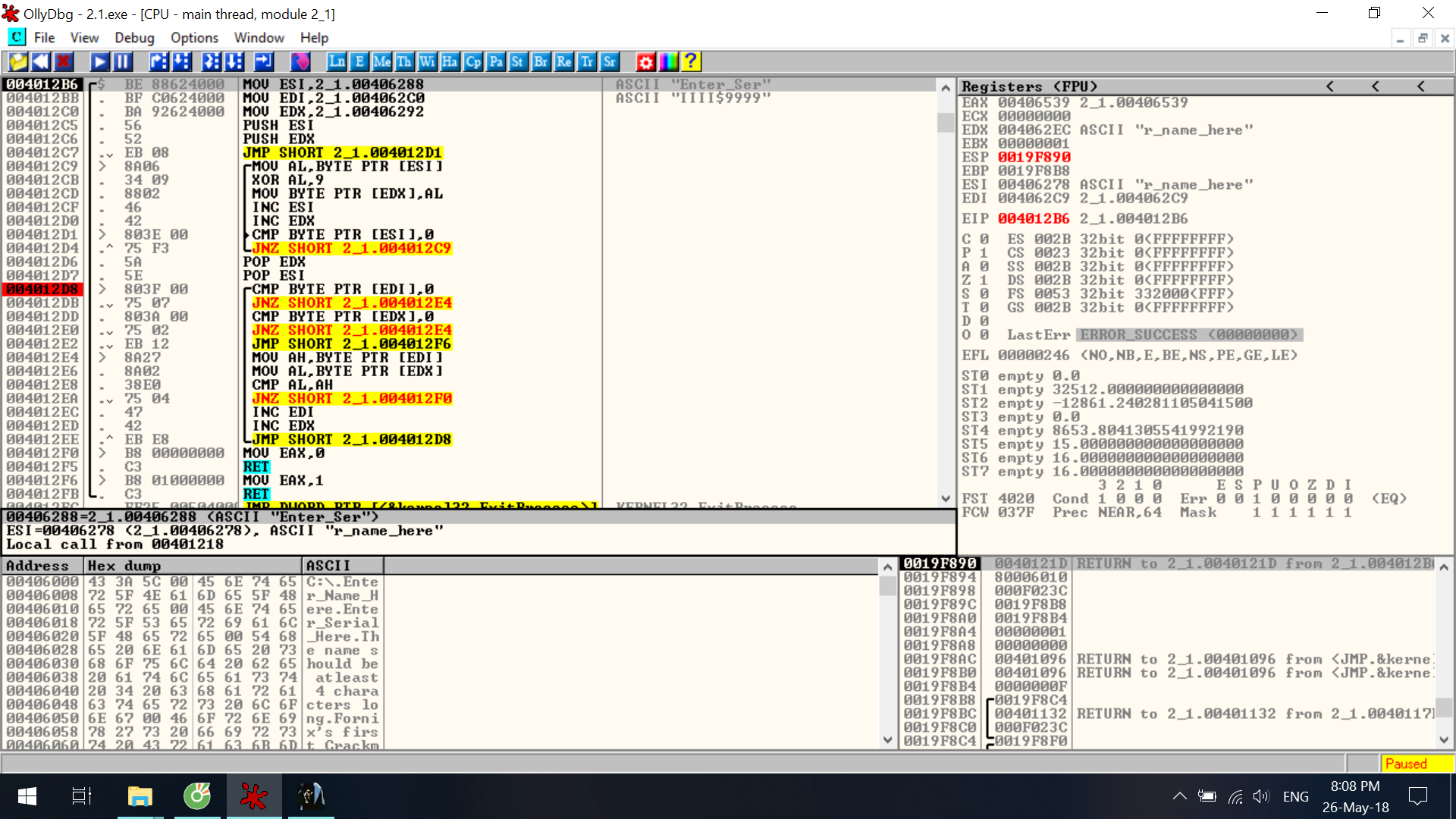


5. Sinh chuỗi khóa từ chuỗi tên do người dùng nhập vào (hàm 2).

EDI và EDX đều trỏ đến chuỗi tên người dùng nhập vào.

**Đoạn code chính:** EDI = (((ESI ^ EDX) & 63) | 48) | 9(dec) = 48 | 9 = ‘9’

* **Chuỗi khóa do 2 hàm trên tạo ra là: “IIII$9999”**



6. Mã hóa tối đa 9 kí tự đầu serial và so sánh chuỗi mã hóa với chuỗi khóa do 2 hàm sinh khóa tạo ra.

* Mã hóa tối đa 9 kí tự đầu serial bằng cách dùng toán tử XOR: XOR AL,9
* So sánh chuỗi mã hóa với chuỗi khóa, nếu đúng thì serial đúng, nếu khác thì serial sai
* Serial **“@@@@-0000”** luôn đúng với mọi chuỗi tên người dùng nhập vào.

## File Crackme 2.2

### Tiền xử lý

* File 2.2.exe ban đầu đã được nén, vì sau khi run trong OllyDbg không xuất hiện thông tin của từng dòng lệnh

A screenshot of a social media post

Description generated with very high confidence

* Sau đó đưa vào PEID để kiểm tra

A screenshot of a cell phone

Description generated with very high confidence

* Sử dụng UPX [1] để giải nén chương trình

A screenshot of a computer screen

Description generated with very high confidence

* Sau đó đưa chương trình đã được giải nén vào OllyDbg và run

A screenshot of a social media post

Description generated with very high confidence

### Chương trình 1

Để nhanh chóng, đặt các breakpoint vào sau các dòng lệnh được gom nhóm và chạy (F9), ta sẽ thấy sự thay đổi của đường dẫn

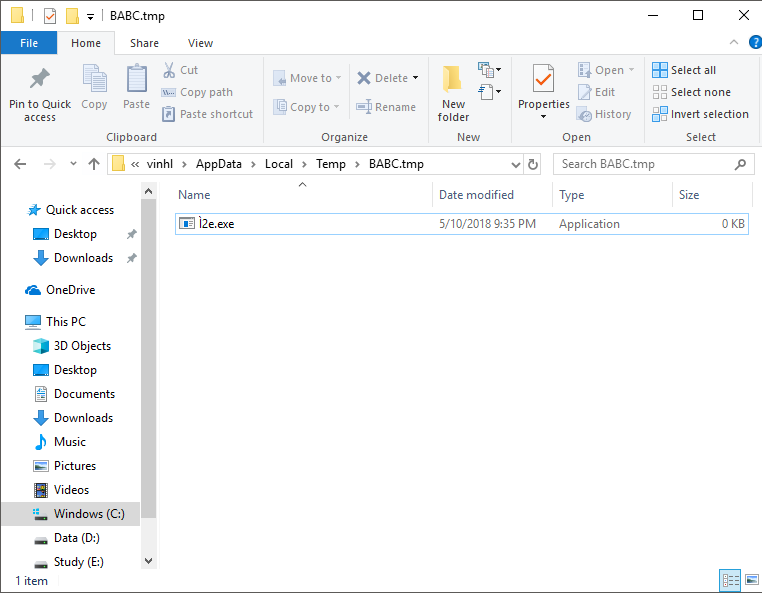
A screenshot of a social media post

Description generated with very high confidence

A screenshot of a social media post

Description generated with very high confidence

Sau khi chạy đến breakpoint tại dòng 0040115A, 1 file exe với tên được sinh ngẫu nhiên



Để khi chương trình chính kết thúc file trên vẫn còn, copy file trên ra một thư mục (chỉ copy được sau khi chạy chương trình đến breakpoint tại dòng 0040115A)

### Chương trình 2

Dùng OllyDbg mở chương trình được sinh ra ở trên

Tương tự ta đặt các breakpoint sau các nhóm lệnh quan trọng (WriteFile, CloseHandle…)

A screenshot of a social media post

Description generated with very high confidence

A screenshot of a social media post

Description generated with very high confidence

Sau đó, vào đường dẫn trong thư mục Temp của User tương tự như trên, ta tìm được file batchfile.bat. Ta copy file này ra một thư mục khác (Lưu ý chỉ copy được sau khi chạy đến breakpoint tại dòng 004014C2)

Đây là file nguồn chính của chương trình được viết bằng BashScript của Windows, nên có thể mở lên rõ rang bằng trình soạn thảo thông thường (ví dụ Notepad, Notepad++,…)

### Chương trình 3

A screenshot of a computer

Description generated with very high confidence

Theo định nghĩa các lệnh Bash của Windows

* echo: Xuất ra màn hình
* set: Đặt <tên biến>=<giá trị>
* Biến sau khi đặt bằng set, chỗ nào cần sử dụng ta ghi %<tên biến>%
* Có thể có một số thao tác xử lý (đặc biệt là chuỗi)

A screenshot of a computer

Description generated with very high confidence

|  |  |  |
| --- | --- | --- |
| Dòng | Giải thích | Ví dụ |
| 165 | Nhập username từ bàn phím và lưu vào biến N | Nếu nhập là “username” thì biến N sẽ có giá trị là “username” (không có dấu “”) |
| 166 | Đặt biến fi = %N:~2,1% nghĩa là bỏ qua 2 kí tự từ trái sang, lấy 1 kí tự trong biến N | fi = e |
| 167 | Đặt biến si = %N:~3,1% nghĩa là bỏ qua 3 kí tự từ trái sang, lấy 1 kí tự trong biến N | si = r |
| 168 | Đặt biến th = %N:~1,1% nghĩa là bỏ qua 1 kí tự từ trái sang, lấy 1 kí tự trong biến N | th = s |
| 169 | Đặt biến fo = %N:~4,1% nghĩa là bỏ qua 2 kí tự từ trái sang, lấy 1 kí tự trong biến N | fo = n |
| 170 | Đặt biến fif = %N:~0,2% nghĩa là bỏ qua 0 kí tự từ trái sang, lấy 2 kí tự trong biến N | fif = us |
| 171 | Đặt biến name=%fif%%fo%%si%%fi%%th% nghĩa là nối các chuỗi trên theo thứ tự tương ứng và lưu vào biến name | name = usnres |
|  | *Lưu ý: Nếu username nhỏ hơn 5 kí tự thì kết quả truy xuất vượt ngoài phạm vi (biến fo chẳng hạn) sẽ trả về là chuỗi rỗng* | |
| 172 | Nhập serial và lưu vào biến O | 123456789 |
| 173 | Nhập key và lưu vào biến ja | <Key này chỉ có giá trị so sánh, chúng ta cần sinh ra key này từ thuật toán> |
| 174 – 180 | Tương tự trên nhưng lấy từng kí tự một của chuỗi serial (biến O)  set one=%O:~0,1%  set two=%O:~1,1%  set three=%O:~2,1%  set four=%O:~3,1%  set five=%O:~4,1%  set six=%O:~5,1%  set seven=%O:~6,1% | One = 1  Two = 2  Three = 3  Four = 4  Five = 5  Six = 6  Seven = 7 |
| 181 | set /a m=%one%\*%two%\*%three%\*%four%\*%five%\*%six%\*%seven%\*3 lưu vào biến m giá trị của các số trên (one, two, … seven) theo công thức  m=one\*two\*three\*four\*five\*six\*seven\*3 | M = 1\*2\*3\*4\*5\*6\*7\*3 = 15120 |
| 182 | set se1=%m:~0,4% Lấy 4 kí tự đầu của biến m | Se1=1512 |
| 183 | set se2=%m:~0,2% Lấy 2 kí tự đầu của biến m | Se2=15 |
| 184 | set se4=%m:~1,3% Bỏ qua kí tự đầu, lấy 3 kí tự tiếp theo | Se4=512 |
| 185 | set se3=%m:~2,1% Bỏ qua 2 kí tự đầu, lấy 1 kí tự tiếp theo | Se3=1 |
| 186 | set se5=%se4%%th%%se3%%si%%se1%%fi%%fif%%se2% Nối các chuỗi lại theo công thức | Se5=512s1r1512eus15 |
| 187 | set la=%se5%%cra%%hac%%name% Nối chuối theo công thức | La=512s1r1512eus15admirable154usnres |
| 188 | So sánh chuỗi serial nhập vào (lưu tại biến ja) và chuỗi key được sinh ra (lưu tại biến la), in thông báo và kết thúc |  |

A screenshot of a computer

Description generated with very high confidence

### Chương trình keygen

Dựa vào các thao tác trên, ta có thể viết được chương trình keygen:

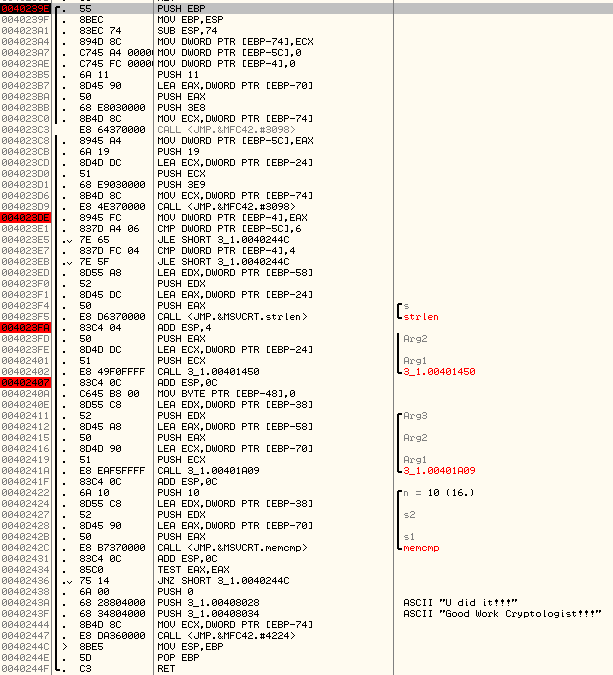
A screenshot of a cell phone

Description generated with very high confidence

## File crackme 3.1

Toàn bộ biến, thanh ghi, địa chỉ hàm được tham chiếu từ lần chạy chương trình lúc thực hiện báo cáo. Địa chỉ của hàm được xem như tên hàm.

1. Bằng cách dung tính năng search for all text string của Olly, ta tìm được Goodboy của file exe này. Qua đó, ta dể thấy được hàm dưới đây được dùng để phát sinh key và kiểm tra tính chính xác dự vào Username mà người dùng nhập vào:



1. Địa chỉ của password được lưu vào địa chỉ [EBP – 24] và của username được lưu vào [EBP – 58]. Chương trình sẽ kiểm tra độ dài của password (> 4) và username (> 6), nếu không thỏa mãn => Đăng nhập thất bại.



1. Lấy độ dài password và lưu vào thanh ghi EAX

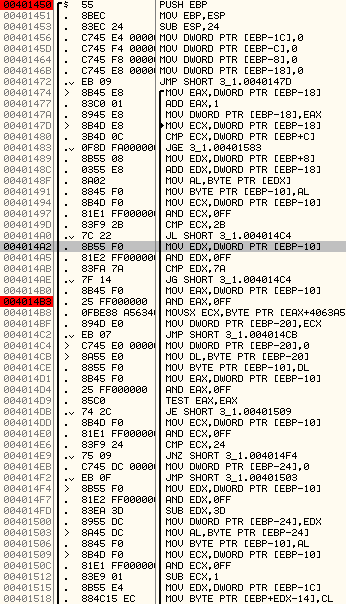


1. Gọi hàm 00401450 với hai tham số là password và độ dài.

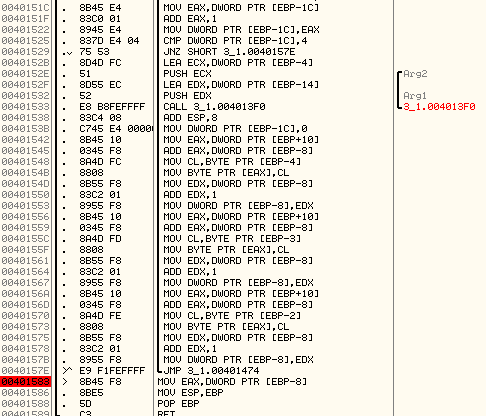


Chi tiết như sau:

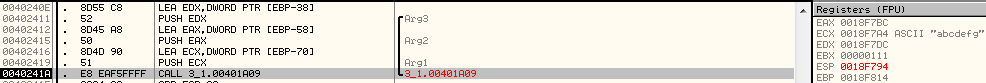
* Lấy mỗi ký tự trong password ra và thực hiện các phép so sánh, phép toán ADD, SUB,… mặc định để chuyển sang 1 ký tự mới.



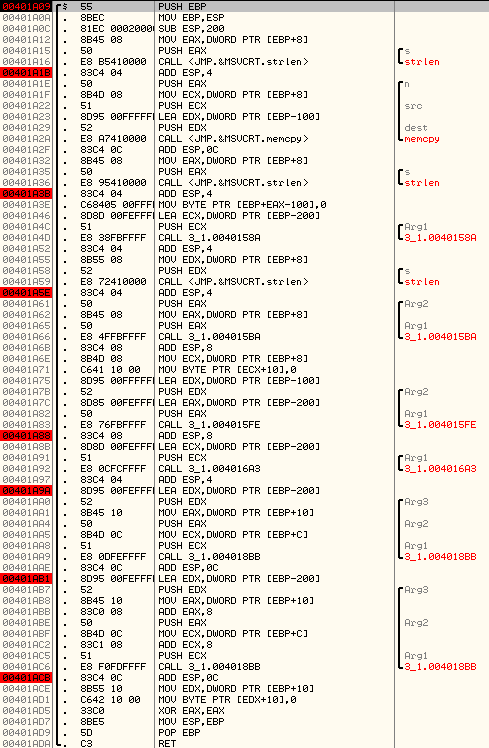
* Cứ mỗi 4 ký tự sẽ thực hiện tiếp một loạt các phép toán để sinh ra 3 ký tự khác (ở hàm 004013F0) và lưu nối tiếp nhau cho đến khi xét hết. Các ký tự dư ra sẽ không xét.



1. Gọi hàm 00401A09 với 3 tham số gồm: địa chỉ dãy ký tự phát sinh sau bước d, địa chỉ username và địa chỉ một vùng nhớ trống để lưu kết quả.



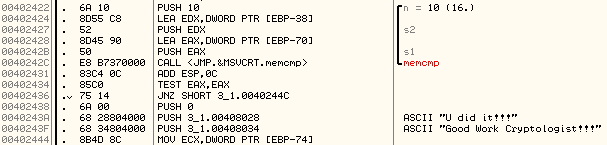
Chi tiết hàm:



Ta quan tâm đến 5 hàm con được gọi trong hàm này:

* + 0040158A: Phát sinh một dãy 256 ký tự với giá trị tăng dần từ 0 đến 255.
  + 004015BA: Dựa vào username mà người dùng nhập vào và qua một thuật toán được cài mặc định phát sinh dãy 16 ký tự mới.
  + 004015FE: Hoán vị các phần tử trong dãy 256 ký tự sinh ra ở hàm 0040158A qua một thuật toán mặc đinh.
  + 004016A3: Lấy từ dãy 256 ký tự đó sau khi gọi hàm 004015FE ra 8 bộ ký tự, mỗi bộ 4 ký tự lưu vào một vùng nhớ (tạm gọi là T). Ký tự tại vị trí nào được lấy ra được quy định từ đầu trong mã nguồn.
  + 004018BB: Lấy 2 bộ ký tự (mỗi bộ 4 ký tự). Qua một thuật toán mặc định, xor với giá trị tại T và lưu vào stack theo thứ tự ngược lại. Qua hai lần gọi hàm, 16 ký tự được sinh ra ở bước d sẽ được chuyển thành 16 ký tự khác, mỗi lần 2 bộ (8 ký tự). Nếu ở bước d sinh ra không đủ 16 ký tự thì đăng nhập sẽ thất bại, nếu dư thì không ảnh hưởng.

1. So sánh giá trị của 16 ký tự sinh ra từ hàm 004015BA với 16 ký tự sinh ra từ hai lần gọi hàm 004018BB. Nếu bằng nhau => Đăng nhập thành công.



## Bài tập trên trang microcorruption.com

### Level Tutorial

|  |
| --- |
| 4484: 6e4f mov.b @r15, r14  4486: 1f53 inc r15  4488: 1c53 inc r12  448a: 0e93 tst r14  448c: fb23 jnz #0x4484 <check\_password+0x0>  448e: 3c90 0900 cmp #0x9, r12  4492: 0224 jeq #0x4498 <check\_password+0x14>  4494: 0f43 clr r15  4496: 3041 ret  4498: 1f43 mov #0x1, r15  449a: 3041 ret |

Từ dòng 4484 đến 448C: Duyệt từng byte từ người dùng nhập vào (lưu trong thanh ghi r15) vào thanh ghi r14 cho đến khi byte đó có giá trị bằng 0x00 (lúc này zero-flag được bật, jnz tại dòng 448C sẽ không thực hiện). Lúc này so sánh r12 (có ý nghĩa là số kí byte người dùng nhập vào) tính cả byte 0x00 (do inc trước rồi mới so sánh với 0x9). Sau lệnh cmp thì zero-flag = 0, gán r15 = 0x1 và nhảy ra hàm main, do r15 = 0x1 nên unlock thành công

* Key: bất cứ chuỗi byte nào đủ 9 byte (kết thúc bằng 0x00), ví dụ
  + 11 11 11 11 11 11 11 11 00
  + 22 22 22 22 22 22 22 22 00

### Level New Orleans

|  |
| --- |
| 44bc: 0e43 clr r14  44be: 0d4f mov r15, r13  44c0: 0d5e add r14, r13  44c2: ee9d 0024 cmp.b @r13, 0x2400(r14)  44c6: 0520 jne #0x44d2 <check\_password+0x16>  44c8: 1e53 inc r14  44ca: 3e92 cmp #0x8, r14  44cc: f823 jne #0x44be <check\_password+0x2>  44ce: 1f43 mov #0x1, r15  44d0: 3041 ret  44d2: 0f43 clr r15  44d4: 3041 ret |

Key được lưu tại vùng nhớ 0x2400 (thấy được khi chạy debug):

2400: 584d 6e55 3575 4e00 0000 0000 0000 0000 XMnU5uN.........

Key do người dùng nhập được lưu tại vùng nhớ trỏ bởi r15, đoạn code duyệt từng kí tự (biến đếm là r14), so sánh từ byte được trỏ bởi r13 (r13, r15 hiện tại chỉ là địa chỉ ô nhớ, @r13 truy xuất đến giá trị tại ô nhớ đó), so sánh từng byte người dùng nhập với từng byte tương ứng tại vùng nhớ 0x2400. Nếu có 1 byte nào sai sẽ gán r15 (kết quả) = 0 và fail. Còn không, sẽ so sánh đến qua byte tại 0x2407 thì kết thúc (mặc dù sau đó có thể còn byte khác)

* Key: Bất kì chuỗi byte nào bắt đầu bằng 8 byte: 584d 6e55 3575 4e00, ví dụ
  + 584d 6e55 3575 4e00
  + 584d 6e55 3575 4e00 1111 1111 1111

### Sydney

|  |
| --- |
| 448a: bf90 3626 0000 cmp #0x2636, 0x0(r15)  4490: 0d20 jnz $+0x1c  4492: bf90 6465 0200 cmp #0x6564, 0x2(r15)  4498: 0920 jnz $+0x14  449a: bf90 6b77 0400 cmp #0x776b, 0x4(r15)  44a0: 0520 jne #0x44ac <check\_password+0x22>  44a2: 1e43 mov #0x1, r14  44a4: bf90 4523 0600 cmp #0x2345, 0x6(r15)  44aa: 0124 jeq #0x44ae <check\_password+0x24>  44ac: 0e43 clr r14  44ae: 0f4e mov r14, r15  44b0: 3041 ret |

Lần này so sánh từng cặp byte (1 word) do người dùng nhập lưu tại vùng nhớ trỏ bởi r15

Do byte được lưu theo Little Endian nên key sẽ là:

* Key: 3626 6465 6b77 4523

### Hanoi

4544: b012 5444 call #0x4454 <test\_password\_valid>

4548: 0f93 tst r15

454a: 0324 jz $+0x8

454c: f240 2200 1024 mov.b #0x22, &0x2410

4552: 3f40 d344 mov #0x44d3 "Testing if password is valid.", r15

4556: b012 de45 call #0x45de <puts>

455a: f290 1500 1024 cmp.b #0x15, &0x2410

4560: 0720 jne #0x4570 <login+0x50>

4562: 3f40 f144 mov #0x44f1 "Access granted.", r15

4566: b012 de45 call #0x45de <puts>

456a: b012 4844 call #0x4448 <unlock\_door>

Nhìn vào dòng 455A thấy so sánh byte tại ô nhớ 2410 với 0x15. Khi chạy debug ta thấy vùng nhớ từ ô 2400 đến 2410 dùng để chứa key do người dùng nhập (tối đa 16 bytes). Do đó ta cố tình nhập dư 1 byte (và byte này có giá trị 0x15) thì key sẽ được chấp nhận

* Key: Chỉ cần có độ dài 17 bytes và bytes cuối cùng là 0x15 thì hợp lệ, ví dụ:
  + 1111 1111 1111 1111 1111 1111 1111 1111 15
  + 2222 2222 3333 3333 4444 4444 5555 5555 15

### Cusco

Giả sử ban đầu ta nhập “1234567890123456” (tương đương hexa: 3132 3334 3536 3738 3930 3132 3334 3536) thì trong bộ nhớ sau khi nhập có dạng:

43d0: 0000 0000 0000 0000 0000 0000 5645 0100 ............VE..

43e0: 5645 0000 a045 0200 ee43 3000 1e45 3132 VE...E...C0..E12

43f0: 3334 3536 3738 3930 3132 3334 3536 0044 34567890123456.D

4400: 3140 0044 1542 5c01 75f3 35d0 085a 3f40 1@.D.B\.u.5..Z?@

Ta để ý ở byte 43fe là 0044 3140 (tức là 0x4400), đây là dòng mà nếu ta chạy tiếp câu lệnh ret (đã đặt breakpoint tại đây) thì thanh ghi pc sẽ quay về địa chỉ này

4532: 3f40 e144 mov #0x44e1 "That password is not correct.", r15

4536: b012 a645 call #0x45a6 <puts>

453a: 3150 1000 add #0x10, sp

**453e: 3041 ret**

Tức là sẽ nhảy vào hàm <\_\_init\_stack>

4400 <\_\_init\_stack>

4400: 3140 0044 mov #0x4400, sp

Và chạy tiếp thì chương trình sẽ yêu cầu nhập lại key. Đây chính là chỗ ta có thể hack dựa vào cơ chế tràn bộ nhớ, tức là nhập quá 16 bytes thì các byte sau sẽ ghi đè lên ô nhớ này

Như vậy, ta chọn giá trị giá trị là ee43 thì sau lệnh ret, thanh ghi pc sẽ đến ô nhớ 0x43ee, tức là ô nhớ lưu 16 bytes đầu của key mình nhập vào, mục đích để điều chỉnh thanh ghi pc thông qua key mình nhập

Như vậy nếu nhập key là 1111 1111 1111 1111 1111 1111 1111 1111 ee43 thì sau lệnh ret, thanh ghi pc sẽ nhảy đến ô có giá trị 11 khoanh đỏ dưới đây:

43d0: 0000 0000 0000 0000 0000 0000 5645 0100 ............VE..

43e0: 5645 0300 ca45 0000 0a00 0000 3a45 1111 VE...E......:E..

43f0: 1111 1111 1111 1111 1111 1111 1111 ee43 ...............C

4400: 0040 0044 1542 5c01 75f3 35d0 085a 3f40 .@.D.B\.u.5..Z?@

Quay lại thanh ghi pc: lúc nãy nếu ô nhớ có giá trị là 0044 3140 thì thanh ghi pc sẽ nhảy đến dòng 4400 tức là hàm <\_\_init\_stack>

4400 <\_\_init\_stack>

**4400: 3140 0044 mov #0x4400, sp**

Do đó nếu ta thay giá trị này để thanh ghi nhảy đến hàm <unlock\_door> thì ta cần để 4 bytes đầu có giá trị là b012 4644 (b012 là lệnh nhảy, 4644 là dòng muốn nhảy đến, tương đương lệnh call #0x4446, tức là hàm <unlock\_door>)

4446 <unlock\_door>

4446: 3012 7f00 push #0x7f

444a: b012 4245 call #0x4542 <INT>

444e: 2153 incd sp

4450: 3041 ret

* Key: Bắt đầu bằng 4 bytes b012 4644 và kết thúc bằng 2 bytes ee43, các bytes giữa không quan trọng, tổng số bytes là 18 (hack qua lỗi tràn bộ nhớ), ví dụ:
  + B012 4644 1111 1111 1111 1111 1111 1111 ee43
  + B012 4644 2222 2222 2222 2222 3333 3333 ee43

### Whitehorse

Tương tự như Cusco, giả sử ban đầu ta nhập “1234567890123456” (tương đương hexa: 3132 3334 3536 3738 3930 3132 3334 3536) thì trong bộ nhớ sau khi nhập có dạng:

3cb0: 0000 4645 0000 9045 0200 c03c 3000 1245 ..FE...E...<0..E

3cc0: 3132 3334 3536 3738 3930 3132 3334 3536 1234567890123456

3cd0: 0044 0000 0000 0000 0000 0000 0000 0000 .D..............

Để ý ta thấy byte 3cd0 có giá trị là 0044, và tương tự Cusco, đây là địa chỉ của <\_\_init\_stack> là chạy lại chương trình. Do đó ta overwrite lại vùng nhớ này bằng cách điền tràn số, giá trị cần chèn vào là c03c để đưa thanh ghi pc sau lệnh ret bên dưới trỏ vào vùng nhớ chứa key

4522: 3f40 d544 mov #0x44d5 "That password is not correct.", r15

4526: b012 9645 call #0x4596 <puts>

452a: 3150 1000 add #0x10, sp

**452e: 3041 ret**

Tức là key lúc này có dạng 3132 3334 3536 3738 3930 3132 3334 3536 c03c

Nhìn qua hàm <conditional\_unlock\_door>, ta thấy khác ở dòng 445c, ở Cusco, giá trị push vào là 0x7f.

445a: 0f12 push r15

445c: 3012 7e00 push #0x7e

4460: b012 3245 call #0x4532 <INT>

4464: 5f44 fcff mov.b -0x4(r4), r15

Tra cứu tài liệu manual [2] thì thấy:

* INT 0x7E. Interface with the HSM-2. Trigger the deadbolt unlock if the password is correct. Takes one argument: the password to test.
* INT 0x7F. Interface with deadbolt to trigger an unlock if the password is correct. Takes no arguments.

Như vậy nếu ta gọi <INT> 0x7F như trước có khả năng sẽ bypass được password (do hàm kiểm tra là <INT> 0x7E)

3012 7f00 push #0x7f

b012 3245 call #0x4532 <INT>

Do đó byte cần truyền vào để thanh pc chạy được 2 dòng trên là 3012 7f00 b012 3245

* Key: Có dạng 3012 7f00 b012 3245 xxxx xxxx xxxx xxxx c03c, ví dụ
  + 3012 7f00 b012 3245 1111 1111 1111 1111 c03c
  + 3012 7f00 b012 3245 2222 3333 4444 5555 c03c

# Tài liệu tham khảo

|  |  |
| --- | --- |
| [1] | L. M. &. J. F. R. Markus F.X.J. Oberhumer, "https://upx.github.io/," The UPX Team, [Online]. Available: https://upx.github.io/. [Accessed 10 05 2018]. |
| [2] | "Microcorruption," [Online]. Available: https://microcorruption.com/manual.pdf. [Accessed 18 May 2018]. |