

BÁO CÁO THỰC HÀNH

Môn học: **Lập trình hệ thống (NT209)**

Lab 3 – Kỹ thuật dịch ngược – Cơ bản

GVHD: *Đỗ Thị Hương Lan*

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT209.N21.ATCL

STT	Họ và tên	MSSV	Email
1	Trần Thị Mỹ Huyền	21520269	21520269@gm.uit.edu.vn
2	Lâm Hải Đăng	21520682	21520682@gm.uit.edu.vn
3	Phan Thị Hồng Nhung	21521250	21521250@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Công việc	Kết quả tự đánh giá
1	Yêu cầu 2.1	100%
2	Yêu cầu 2.2	90%
3	Yêu cầu 2.3	90%

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

¹ Ghi nội dung công việc, yêu cầu trong bài Thực hành

BÁO CÁO CHI TIẾT

1. Yêu cầu 2.1

Set up stack frame

```
.text:0804894B var_14          = dword ptr -14h
.text:0804894B var_4           = dword ptr -4
.text:0804894B argc            = dword ptr 8
.text:0804894B argv             = dword ptr 0Ch
.text:0804894B envp             = dword ptr 10h
```

Đoạn mã trên là phần định nghĩa các biến cục bộ của hàm main

```
.text:0804894B ; __ unwind {
.text:0804894B                 lea      ecx, [esp+4]
.text:0804894F                 and      esp, 0FFFFFFF0h
.text:08048952                 push     dword ptr [ecx-4]
.text:08048955                 push     ebp
.text:08048956                 mov      ebp, esp
.text:08048958                 push     ecx
.text:08048959                 sub      esp, 14h
.text:0804895C                 sub      esp, 0Ch
```

lea ecx, [esp+4]: tính địa chỉ của argc và lưu vào thanh ghi ecx.

and esp, 0FFFFFFF0h: làm cho esp trở đến địa chỉ chia hết cho 16.

push dword ptr [ecx-4]: push địa chỉ của envp lên stack.

push ebp: push địa chỉ của ebp lên stack.

mov ebp, esp: gán giá trị của esp vào ebp, thiết lập stack frame của hàm main().

push ecx: push địa chỉ của argc lên stack.

sub esp, 14h: Cấp phát không gian cho chuỗi được sử dụng trong lệnh _printf()

sub esp, 0ch : Giảm giá trị của esp 12 bytes để cấp phát không gian cho các biến cục bộ

- Các biến cục bộ được sử dụng để tính địa chỉ các ô nhớ ở phía sau.
- Việc setup stack frame cho các hàm bên trong hàm main cũng tương tự.
- Việc sử dụng stack frame sẽ tùy thuộc vào từng hàm.

```

    lea    ecx, [esp+4]
    and    esp, 0FFFFFFF0h
    push   dword ptr [ecx-4]
    push   ebp
    mov    ebp, esp
    push   ecx
    sub    esp, 14h
    sub    esp, 0Ch
    push   offset aSupportedAuth
    call   _printf
    add    esp, 10h

```

- Quan sát hàm main, ta thấy có 1 lệnh call tới _printf, tức in ra màn hình. Trên lệnh call _printf ta có thể thấy 1 địa chỉ được push vào stack. Chính là địa chỉ của chuỗi cần in.
- Double click vào chuỗi ta có thể thấy được đầy đủ chuỗi mà chương trình muốn in

```

; char aSupportedAuth[]
aSupportedAuth db 'Supported authentication methods:',0Ah
                ; DATA XREF: main+14↑o
                db '1. Hard-coded password',0Ah
                db '2. Another hard-coded password',0Ah
                db '3. Username/password',0Ah
                db 'Enter your choice: ',0
aD                 db '%d',0           ; DATA XREF: main+28↑o
                .

```

```

    sub    esp, 14h
    sub    esp, 0Ch
    push   offset aSupportedAuth
    call   _printf
    add    esp, 10h
    sub    esp, 8
    lea    eax, [ebp+var_14]
    push   eax
    push   offset aD      ; "%d"
    call   __isoc99_scanf
    add    esp, 10h
    . . .

```

- Ở các câu lệnh tiếp theo, chương trình lấy địa chỉ tại ô nhớ [ebp+var_14] và truyền 2 tham số: 1 là địa chỉ ô nhớ vừa lấy, 2 là kiểu định dạng cho dữ liệu nhập vào và gọi lệnh scanf
- Phần assembly trên đang thể hiện việc chờ input từ người nhập là một số nguyên

```

add    esp, 10h
mov    eax, [ebp+var_14]
cmp    eax, 1
jnz    short loc_80489A0
call   hardCode
jmp    short loc_80489D8
;

loc_80489A0: ; CODE XREF: main+4C↑j
    mov    eax, [ebp+var_14]
    cmp    eax, 2
    jnz    short loc_80489AF
    call   otherhardCode
    jmp    short loc_80489D8
;

loc_80489AF: ; CODE XREF: main+5B↑j
    mov    eax, [ebp+var_14]
    cmp    eax, 3
    jnz    short loc_80489BE
    call   userpass
    jmp    short loc_80489D8
;
```

- Ở phần assembly tiếp theo, ta thấy giá trị ô nhớ [ebp+var_14] được đem ra để so sánh với lần lượt các số là 1,2,3; tương ứng với việc gọi đến các hàm hardcoded(), otherhardCode(), và userpass()
- Ta sẽ lần lượt phân tích các hàm trên.
- Xét hàm hardcoded()

```

s1      = byte ptr -3F0h

push    ebp
mov     ebp, esp
sub    esp, 3F8h
call   _getchar
sub    esp, 0Ch
push    offset aEnterTheHardCo ; "Enter the hard-coded password (option 1".
call   puts
add    esp, 10h
sub    esp, 8
lea     eax, [ebp+s1]
push    eax
push    offset asc_804922A ; "%[^n]"
call   isoc99 scanf
;
```

- Tương tự như hàm main, bắt đầu của hàm hardcoded() cũng sẽ in ra một chuỗi với nội dung yêu cầu người dùng nhập input với puts và đọc input người dùng sử dụng scanf.
- Địa chỉ tham chiếu đến chuỗi input là [ebp+s1]

```

add    esp, 10h
sub    esp, 8
lea    eax, [ebp+s1]
push   eax
push   offset format ; "Your input hard-coded password: %s\n"
call   _printf
add    esp, 10h
sub    esp, 8

```

- Ở các câu lệnh tiếp theo, sau khi nhận input từ người dùng, chương trình truyền 2 tham số vào lệnh printf: 1 là địa chỉ của biến muốn in là [ebp+s1], tức địa chỉ tham chiếu đến chuỗi input của người dùng, 2 là địa chỉ của chuỗi chương trình muốn in được push vào stack. Tức chương trình sẽ in ra chuỗi “Your input...” kèm theo input của người dùng vừa mới nhập

```

add    esp, 10h
sub    esp, 8
push   offset s2      ; "One day, all your hard work will pay off"...
lea    eax, [ebp+s1]
push   eax             ; s1

```

- Tiếp theo, một địa chỉ tham chiếu đến chuỗi “One day, all your hard work will pay off” và địa chỉ tham chiếu đến input người dùng được truyền vào và gọi đến hàm strcmp.

```

call   _strcmp
add    esp, 10h
test   eax, eax
jnz   short loc_80486FE
call   success_1
jmp   short loc_8048703

```

```

loc_80486FE:           ; CODE XREF: hardCode+65↑j
    call   failed

```

- Nếu chuỗi người dùng và chuỗi ““One day, all your hard work will pay off.”” là giống nhau theo hàm strcmp (output hàm bằng 0) thì nhảy đến hàm success_1 rồi rời hàm hardCode(), ngược lại nhảy đến hàm failed
- Output từ hàm strcmp sẽ được bỏ vào thanh ghi eax => test eax, eax nhằm xét output của hàm strcmp có bằng 0 hay không.

```

success_1    public success_1
              proc near             ; CODE XREF: hardCode+67↑p
              push   ebp
              mov    ebp, esp
              sub    esp, 8
              sub    esp, 0Ch
              push   offset s          ; "Congrats! You found the hard-coded secret...."
              call   _puts
              add    esp, 10h
              sub    esp, 0Ch
              push   offset aHandInThisToYo ; "Hand in this to your instructor as a proof...."
              call   _puts
              add    esp, 10h
              sub    esp, 0Ch
              push   offset aLearningNeverE ; "\"Learning never exhausts the mind.\\""
              call   _puts
              add    esp, 10h
              nop
              leave
              retn
success_1    endp

```

- Nhìn qua hàm success_1 ta có thể thấy chính là thông báo cho việc nhập đúng password.
- Tức password đúng ở đây chính là chuỗi “One day, all your hard work will pay off”. Ta chỉ cần tìm đầy đủ chuỗi trên và sẽ có được password ta mong muốn.

```

; char s2[]
s2           db 'One day, all your hard work will pay off',0

```

- Đem chuỗi sau vào chương trình để kiểm tra:

```

(kali㉿kali)-[~]
$ /home/kali/Desktop/basic-reverse
Supported authentication methods:
1. Hard-coded password
2. Another hard-coded password
3. Username/password
Enter your choice: 1
Enter the hard-coded password (option 1):
One day, all your hard work will pay off
Your input hard-coded password: One day, all your hard work will pay off
Congrats! You found the hard-coded secret, good job :).
Hand in this to your instructor as a proof:
"Learning never exhausts the mind."

```

2. Yêu cầu 2.2

- Xét tiếp theo đến hàm otherhardCode()



```

push    ebp
mov     ebp, esp
sub    esp, 3F8h
call   _getchar
sub    esp, 8Ch
push    offset aEnterTheHard_0 ; "Enter the hard-coded password (option 2"...
call   _puts
add    esp, 10h
sub    esp, 8
lea     eax, [ebp+s1]
push    eax
push    offset asc_804922A ; "%[^\\n]"
call   __isoc99_scanf
add    esp, 10h
sub    esp, 8
lea     eax, [ebp+s1]
push    eax
push    offset format    ; "Your input hard-coded password: %s\\n"
call   _printf
add    esp, 10h
mov    [ebp+var_C], 28h
mov    eax, [ebp+var_C]
mov    eax, WHAT_THAT[eax*4]
mov    [ebp+s2], eax

```

- Tương tự như hàm main, bắt đầu của hàm otherHardcode() cũng sẽ in ra một chuỗi với nội dung yêu cầu người dùng nhập input với puts và đọc input người dùng sử dụng scanf.
- Địa chỉ tham chiếu đến chuỗi input là địa chỉ của ô nhớ [ebp+s1]
- Tiếp theo ta có thể thấy một giá trị 2B theo mã Hex (tương đương 43 trong decimal) được đem vào ô nhớ có địa chỉ [ebp+var_C]
- Sau đó giá trị trong ô nhớ [ebp+var_C] được đem vào thanh ghi eax
- Lúc này có sự xuất hiện của 1 chuỗi WHAT_THAT
- Sau khi double click vào chuỗi ta được như sau:

data:0804B060	public WHAT_THAT
data:0804B060 WHAT_THAT	dd offset aYouScratchMyBa ; DATA XREF: otherhardCode+56Tr
data:0804B060	aYouScratchMyBa ; "You scratch my back and I'll scratch yo"...
data:0804B064	dd offset aNewOneInOldOne ; "New one in, old one out"
data:0804B068	dd offset aITTooLateToLoc ; "It' too late to lock the stable when th"...
data:0804B06C	dd offset aWithAgeComesWi ; "With age comes wisdom"
data:0804B070	dd offset aNothingIsMoreP ; "Nothing is more precious than independe"...
data:0804B074	dd offset aHandsomeIsASha ; "Handsome is as handsome does"
data:0804B078	dd offset aNeverOfferToTe ; "Never offer to teach fish to swim"
data:0804B07C	dd offset aToTryToRunBefo ; "To try to run before the one can walk"
data:0804B080	dd offset aNobodyHasEverS ; "Nobody has ever shed tears without seei"...
data:0804B084	dd offset aYouGetWhatYouP ; "You get what you pay for"
data:0804B088	dd offset aAsStrongAsAHor ; "As strong as a horse"
data:0804B08C	dd offset aAllRoadsLeadTo ; "All roads lead to Rome"
data:0804B090	dd offset aGoodWineNeedsN ; "Good wine needs no bush"
data:0804B094	dd offset aDiamondCutsDia ; "Diamond cuts diamond"
data:0804B098	dd offset aSpareTheRodAnd ; "Spare the rod and spoil the child"
data:0804B09C	dd offset aSpeakOneWayAnd ; "Speak one way and act another"
data:0804B0A0	dd offset aDonTJudgeABook ; "Don't judge a book by its cover"
data:0804B0A4	dd offset aITSNtUseBeatin ; "It's no use beating around the bush"
data:0804B0A8	dd offset aManProposesGod ; "Man proposes God deposes"

- Trước hết WHAT_THAT ở đây chính là địa chỉ 0804B060 – địa chỉ đầu tiên của 1 chuỗi offset (base address). Các cell nằm kế tiếp nhau từ base address tạo nên thành mảng WHAT_THAT.

- Theo nghiên cứu, offset ở đây chính là 1 địa chỉ mà phải cộng với base address để có được địa chỉ chính xác
- Sau khi biến đổi, ta có được giá trị thực sự của các phần tử trong WHAT_THAT

```

0804B060
0804B060 WHAT_THAT
0804B064
0804B068
0804B06C
0804B070
0804B074
0804B078
0804B07C
0804B080
0804B084
0804B088
0804B08C
0804B090
0804B094
0804B098
0804B09C
0804B0A0
0804B0A4
0804B0A8
0804B0AC
0804B0B0
0804B0B4

public WHAT THAT
dd 8048A70h
dd 8048A9Bh
dd 8048AB4h
dd 8048AEDh
dd 8048B04h
dd 8048B3Bh
dd 8048B58h
dd 8048B7Ch
dd 8048BA4h
dd 8048BD7h
dd 8048BF0h
dd 8048C05h
dd 8048C1Ch
dd 8048C34h
dd 8048C4Ch
dd 8048C6Eh
dd 8048C8Ch
dd 8048CACH
dd 8048CD0h
dd 8048CE9h
dd 8048D02h
dd 8048D1Ch
;
```

- Vậy về cơ bản, mảng WHAT_THAT ở đây là một mảng các con trỏ, trỏ đến địa chỉ nơi chứa các chuỗi string liên tiếp nhau. Mỗi con trỏ chiếm 4 bytes
- Vậy mov eax, WHAT_THAT [eax*4] ở đây sẽ bằng giá trị của WHAT_THAT (base address) + offset (eax*4) = 0804B060 + 2B*4 = 0804B10C

```

data:0804B10C dd offset aBeautyIsInTheE ; "Beauty is in the eye of the beholder"

        mov    [ebp+s2], eax
        sub    esp, 8
        push   [ebp+s2]      ; s2
        lea    eax, [ebp+s1]
        push   eax           ; s1
        call   _strcmp
        add    esp, 10h
        test   eax, eax
        jnz   short loc_8048786
        call   success_2
        jmp   short loc_8048788

```

- Sau đó giá trị thanh ghi eax và địa chỉ của ô nhớ [ebp+s1] tức chuỗi input của người dùng được truyền vào hàm strcmp.
- Điều kiện so sánh cũng là nếu 2 chuỗi giống nhau thì nhảy đến hàm success_2
- Tiếp theo ta hãy xem mảng WHAT_THAT chứa gì

- Test input này trong chương trình

```
(kali㉿kali)-[~]
$ /home/kali/Desktop/basic-reverse
Supported authentication methods:
1. Hard-coded password
2. Another hard-coded password
3. Username/password
Enter your choice: 2
Enter the hard-coded password (option 2):
Beauty is in the eye of the beholder
Your input hard-coded password: Beauty is in the eye of the beholder
Good job! You defeated a harder level of finding hard-coded secret :).
Hand in this to your instructor as a proof:
"Don't let what you cannot do interfere what you can do."
```

3. Yêu cầu 2.3

MÃ GIẢ

Lưu ý: Các số trong mã giả đều là decimal nhưng hiển thị thì sử dụng ASCII.

Bài này để dễ làm thì ta xài mã giả. Đầu tiên là khai báo biến và kiểm tra xem độ dài của username và password có bằng 9 hay không.

Còn mảng v7 đang chứa các số ở dạng decimal. Sau khi dò bảng mã ASCII thì ra v7="Et--".

```

int userpass()
{
    size_t v0; // ebx@2
    int result; // eax@3
    size_t v2; // eax@15
    size_t v3; // edx@16
    char v4[9]; // [sp+Ah] [bp-2Eh]@6
    char v5[10]; // [sp+13h] [bp-25h]@1
    char s[10]; // [sp+1Dh] [bp-1Bh]@1
    char v7; // [sp+27h] [bp-11h]@1
    char v8; // [sp+28h] [bp-10h]@1
    char v9; // [sp+29h] [bp-Fh]@1
    char v10; // [sp+2Ah] [bp-Eh]@1
    char v11; // [sp+2Bh] [bp-Dh]@1
    unsigned int i; // [sp+2Ch] [bp-Ch]@4

    v7 = 96;
    v8 = 69;
    v9 = 116;
    v10 = 45;
    v11 = 45;
    getchar();
    puts("Enter your username:");
    __isoc99_scanf("%[^\\n]", s);
    getchar();
    puts("Enter your password:");
    __isoc99_scanf("%[^\\n]", v5);

    if ( strlen(s) == 9 && (v0 = strlen(s), v0 == strlen(v5)) )

```

Sau đó chương trình tạo một mảng v4 mới được hình thành như dưới hình. Cho s = "269682250" (MSSV 3 bạn). 4 ký tự đầu của v4 tương ứng với vị trí 2,3,7,8 của s. 5 vị trí sau chính là chuỗi v7 = "Et--".

```

        for ( i = 0; (signed int)i <= 8; ++i )
    {
        if ( (signed int)i > 1 )
        {
            if ( (signed int)i > 3 )
                v4[i] = *(&v7 + i - 4);
            else
                v4[i] = s[i + 5];
        }
        else
        {
            v4[i] = s[i + 2];
        }
    }

```

Biến v2 = strlen(s) dùng để làm giới hạn vòng lặp, nếu thỏa một trong những điều kiện tại "if" thì vòng lặp dừng. Sau đó ta xét v3 = i hay không. Nếu có thì success_3() được gọi và thành công.

Ta cho i chạy hết vòng lặp tới khi v1 <= i. Đồng thời cho điều kiện $(s[i] + v4[i]) / 2 == v5[i]$ để vòng lặp không bị break.

```

for ( i = 0; ; ++i )
{
    v2 = strlen(s);
    if ( v2 <= i || (s[i] + v4[i]) / 2 != v5[i] )
        break;
}
v3 = strlen(s);
if ( v3 == i )
    result = success_3();
else
    result = failed();
}
else
{
    result = failed();
}
return result;
}

```

Giờ ta sẽ chạy 1 file python để tìm password. Phép tính cộng trên từng ký tự được chuyển sang decimal rồi tính tổng bình thường, sau đó chia cho 2, convert về lại char để so sánh. Ta được kết quả password = "5655L;S1."

```

s = "269682250"
v7 ="^Et--"
v4 = "9625`Et--"
v2 = 8

username = s
key = v4
password = ""
for i in range(9):
    password += chr((ord(username[i]) + ord(key[i])) // 2)
print(password)

```

Chạy trong linux:

```
(kali㉿kali)-[~/Downloads]
$ ./basic-reverse
Supported authentication methods:
1. Hard-coded password
2. Another hard-coded password
3. Username/password
Enter your choice: 3
Enter your username:
269682250
Enter your password:
5673L;S1.ami04
Your input username: 269682250 and password: 5673L;S1.
Awesome! You found your own username/password pair. Nice work.
Hand in this to your instructor as a proof:
"It always seems impossible until it's done."
```

ASSEMBLY

```
mov    [ebp+var_11], 60h ; ``
mov    [ebp+var_10], 45h ; 'E'
mov    [ebp+var_F], 74h ; 't'
mov    [ebp+var_E], 20h ; '-'
mov    [ebp+var_D], 20h ; '-'
```

- Đem các giá trị lần lượt vào các ô nhớ.

```
push   offset aEnterYourUsern ; "Enter your username:"
call   _puts
add    esp, 10h
sub    esp, 8
lea    eax, [ebp+s]
push   eax
push   offset asc_804922A ; "%[^\\n]"
call   __isoc99_scanf
add    esp, 10h
call   _getchar
sub    esp, 0Ch
push   offset aEnterYourPassw ; "Enter your password:"
call   _puts
add    esp, 10h
sub    esp, 8
lea    eax, [ebp+var_25]
push   eax
push   offset asc_804922A ; "%[^\\n]"
call   __isoc99_scanf
```

- Đẩy địa chỉ của chuỗi “Enter your username” và gọi đến hàm puts để in ra màn hình. Tương tự với chuỗi “Enter your password”.
- Đẩy địa chỉ của ô nhớ [ebp+s] cùng với địa chỉ của chuỗi “%[^\\n]” (nhận input cho đến khi ấn enter) lên stack và gọi đến hàm scanf để nhập input. Tương tự với hàm scanf cho ô nhớ [ebp+var_25]

```

add    esp, 10h
sub    esp, 4
lea    eax, [ebp+var_25]
push   eax
lea    eax, [ebp+s]
push   eax
push   offset aYourInputUsern ; "Your input username: %s and password: %"
call   _printf
...
```

- Đẩy địa chỉ của ô nhớ [ebp+var_25] (password) và địa chỉ ô nhớ [ebp+s](username) lên stack và địa chỉ của chuỗi “Your input username..” lên stack và gọi đến hàm printf để in ra màn hình lại password và username đã nhập.

```

add    esp, 10h
sub    esp, 0Ch
lea    eax, [ebp+s]
push   eax          ; s
call   _strlen
add    esp, 10h
cmp    eax, 9
jnz    short loc_804884B
sub    esp, 0Ch
lea    eax, [ebp+s]
push   eax          ; s
call   _strlen
```

- Lấy độ dài của chuỗi username bằng cách lấy địa chỉ ô nhớ [ebp+s] và push lên stack rồi gọi đến hàm strlen.
- So sánh độ dài chuỗi username với 9. Nếu bằng nhau thì ZF (Zero flag – Condition codes) sẽ được gán. Nên nếu không bằng thì Zero flag không được gán và so sánh nếu không bằng 0 sẽ nhảy đến loc_804884b, nơi gọi đến hàm failed thông báo sai đáp án.
- Nếu bằng nhau, tiếp tục lấy độ dài của chuỗi username push lên stack và lấy độ dài của chuỗi password. Độ dài của chuỗi username được chuyển vào thanh ghi ebx

```

add    esp, 10h
mov    ebx, eax
sub    esp, 0Ch
lea    eax, [ebp+var_25]
push   eax          ; s
call   _strlen
add    esp, 10h
cmp    ebx, eax
jz    short loc_8048855

loc_804884B:           ; CODE XREF: userpass+97↑j
    call   failed
    jmp    loc_8048945
```

- So sánh độ dài chuỗi password với độ dài của chuỗi username. Nếu bằng nhau Zero flag sẽ được gán bằng 0. Khi gọi so sánh jz (so sánh có bằng 0 hay không) sẽ nhảy đến loc_8048855. Ngược lại chạy tiếp tục xuống và gọi đến hàm failed. loc_8048945 là nơi thoát hàm userpass (hàm của 2.3).

```
loc_8048855: ; CODE XREF: userpass+BB↑j
    mov     [ebp+var_C], 0
    mov     [ebp+var_C], 0
    jmp     short loc_80488B8
```

- Xét loc_8048855, giá trị 0 được mov vào ô nhớ [ebp+var_C] và nhảy tiếp đến loc_80488B8.

```
loc_80488B8: ; CODE XREF: userpass+D5↑j
    cmp     [ebp+var_C], 8
    jle     short loc_8048865
    mov     [ebp+var_C], 0
    jmp     short loc_8048906
```

- Xét loc_80488B8, giá trị trong ô nhớ [ebp+var_C] được so sánh với 8. Nếu giá trị này <= 8 thì nhảy đến loc_8048865. Ngược lại, gán [ebp+var_C] = 0 và nhảy đến loc_8048906.

```
loc_8048865: ; CODE XREF: userpass+12E↓j
    cmp     [ebp+var_C], 1
    jg      short loc_8048882
    mov     eax, [ebp+var_C]
    add     eax, 2
    movzx   eax, [ebp+eax+s]
    lea     ecx, [ebp+var_2E]
    mov     edx, [ebp+var_C]
    add     edx, ecx
    mov     [edx], al
    jmp     short loc_80488B4
```

- Xét loc_8048865, giá trị trong ô nhớ [ebp+var_C] được so sánh với 1. Nếu > 1 thì nhảy đến short loc_8048882. Ngược lại đem giá trị trong ô nhớ [ebp+var_C] vào thanh ghi eax. Cộng giá trị này với 2. Đem giá trị trong ô nhớ [ebp+eax+s]. Lấy địa chỉ của ô nhớ [ebp+var_2E] vào ecx. Lấy giá trị ô nhớ [ebp+var+C] vào edx. Edx = [ebp+var_C] + ebp+var_2E. Lấy giá trị trong thanh ghi al (1 byte cuối của thanh ghi eax) đem vào ô nhớ [edx]. Nhảy đến loc_80488B4

```

loc_80488B4:          ; CODE :
                     ; userp
    add     [ebp+var_C], 1

loc_80488B8:          ; CODE :
    cmp     [ebp+var_C], 8
    jle     short loc_80488E5
    mov     [ebp+var_C], 0
    jmp     short loc_8048906

```

- Xét loc_80488B4, ô nhớ [ebp+var_C] được cộng thêm 1 và quay lại loc_80488B8. **Ta có thể nhận ra đây là một vòng lặp thực hiện tính toán giá trị gì đó, với biến lặp tại ô nhớ [ebp+var_C] (đặt là i).** Vậy ta khoan đào sâu vào vòng lặp. Ta xét tiếp khi kết thúc vòng lặp sẽ làm gì.

```

loc_8048906:          ; CODE XREF: userpass+137↑j
    sub     esp, 0Ch
    lea     eax, [ebp+s]
    push   eax           ; s
    call   _strlen
    add    esp, 10h
    mov    edx, eax
    mov    eax, [ebp+var_C]
    cmp    edx, eax
    ja    short loc_80488C7
    jmp    short loc_8048921

```

- Xét loc_8048906, ta lấy độ dài của chuỗi username bằng cách lấy địa chỉ ô nhớ [ebp+s] và push lên stack rồi gọi tới hàm strlen. Ta gán edx bằng độ dài của chuỗi username, gán eax bằng giá trị tại ô nhớ p [ebp+var_C] (i)
- So sánh độ dài chuỗi username với giá trị i. Nếu giá trị i lớn hoặc bằng hơn độ dài chuỗi username thì nhảy đến loc_80488C7. Ngược lại nhảy đến loc_8048921

```

loc_80488C7:          ; CODE XREF: userpass+18E↓j
    lea    edx, [ebp+s]
    mov    eax, [ebp+var_C]
    add    eax, edx
    movzx  eax, byte ptr [eax]
    movsx  edx, al
    lea    ecx, [ebp+var_2E]
    mov    eax, [ebp+var_C]
    add    eax, ecx
    movzx  eax, byte ptr [eax]
    movsx  eax, al
    add    eax, edx
    mov    edx, eax
    shr    edx, 1Fh
    add    eax, edx
    sar    eax, 1
    mov    ecx, eax
    lea    edx, [ebp+var_25]
    mov    eax, [ebp+var_C]
    add    eax, edx
    movzx  eax, byte ptr [eax]
    movsx  eax, al
    cmp    ecx, eax
    jnz    short loc_8048920
    add    [ebp+var_C], 1

```

- Xét loc_80488C7, ta có thể thấy các tính toán phức tạp nhằm ra một giá trị nào đó để so sánh với một phần tử thứ i trong chuỗi password ([ebp+25]). Nếu bằng nhau thì zero flag sẽ được gán bằng 0 và khi so sánh jnz (không bằng 0) sẽ nhảy đến loc_8048920. Ngược lại cộng biến i lên 1 đơn vị và quay lại loc_8048906 vừa xét bên trên.
- Ta nhận ra đây cũng là một vòng lặp nữa. Và nếu nhìn lên loc_8048906, ta có thể thấy nếu vòng lặp này có thể chạy đến khi biến i bằng độ dài của chuỗi username (=9) sẽ nhảy đến loc_8048921.

```

loc_8048921:          ; CODE XREF: userpass+190↑j
    sub    esp, 0Ch
    lea    eax, [ebp+s]
    push   eax           ; s
    call   _strlen
    add    esp, 10h
    mov    edx, eax
    mov    eax, [ebp+var_C]
    cmp    edx, eax
    jnz    short loc_8048940
    call   success_3
    jmp    short loc_8048945

```

- Xét loc_8048921, chương trình lấy độ dài của chuỗi password ([ebp+s]) với cách tương tự và so sánh với biến i ([ebp+var_C]). Nếu bằng zero flag được gán bằng 0 và khi so sánh jnz (không bằng 0 hay không) sẽ gọi đến hàm success_3, thông

báo đã nhập đúng mật khẩu và nhảy đến loc_8048945 và thoát hàm userpass.

Ngược lại nhảy đến loc_8048940 gọi đến hàm failed và thông báo nhập sai đáp án.

- Từ đó ta có thể suy đoán, nếu có thể cho vòng lặp phía trên vừa xét lặp đến khi i = 9 thì ta có thể biết được password.
- Khi So sánh độ dài chuỗi username với giá trị i. Nếu giá trị i lớn hoặc bằng độ dài chuỗi username thì nhảy đến loc_80488C7. i bắt đầu từ giá trị 0 nên nếu xét i > 9 thì luôn fail cho đến khi i = 9
- Ta xét loc_80488C7 1 lần nữa

```
loc_80488C7: ; CODE XREF: userpass+18E↓j
    lea    edx, [ebp+s]
    mov    eax, [ebp+var_C]
    add    eax, edx
    movzx eax, byte ptr [eax]
    movsx edx, al
    lea    ecx, [ebp+var_2E]
    mov    eax, [ebp+var_C]
    add    eax, ecx
    movzx eax, byte ptr [eax]
    movsx edx, al
    add    eax, edx
    mov    edx, eax
    shr    edx, 1Fh
    add    eax, edx
    sar    eax, 1
    mov    ecx, eax
    lea    edx, [ebp+var_25]
    mov    eax, [ebp+var_C]
    add    eax, edx
    movzx eax, byte ptr [eax]
    movsx edx, al
    cmp    ecx, eax
    jnz    short loc_8048920
    add    [ebp+var_C], 1
```

- loc_8048920 là nơi nhảy đến thực hiện loc_8048921 để xét i có bằng 9 hay không để gọi làm success. Vậy ta phải làm sao để khi so sánh một giá trị nào đó với phần tử thứ i trong chuỗi password ([ebp+25]) luôn bằng nhau để i được cộng lên 9.
- Nhìn cụ thể loc_8048920, ta có thể thấy chương trình lấy phần tử thứ i của chuỗi username ([ebp+s]) và phần tử thứ i của giá trị gì đó được tạo từ vòng lặp đầu tiên. Cộng lại với nhau rồi chia 2 (sar eax,1), rồi lấy giá trị đó so sánh với phần tử thứ i của chuỗi password ([ebp+25]).

- Vậy ta chỉ cần nhập 1 password có độ dài 9 kí tự, in ra phần giá trị dùng để so sánh với password. Ta sẽ có được password đúng cho username.
- Về phần tìm password bằng debug nhóm em chưa thực hiện được.

HẾT