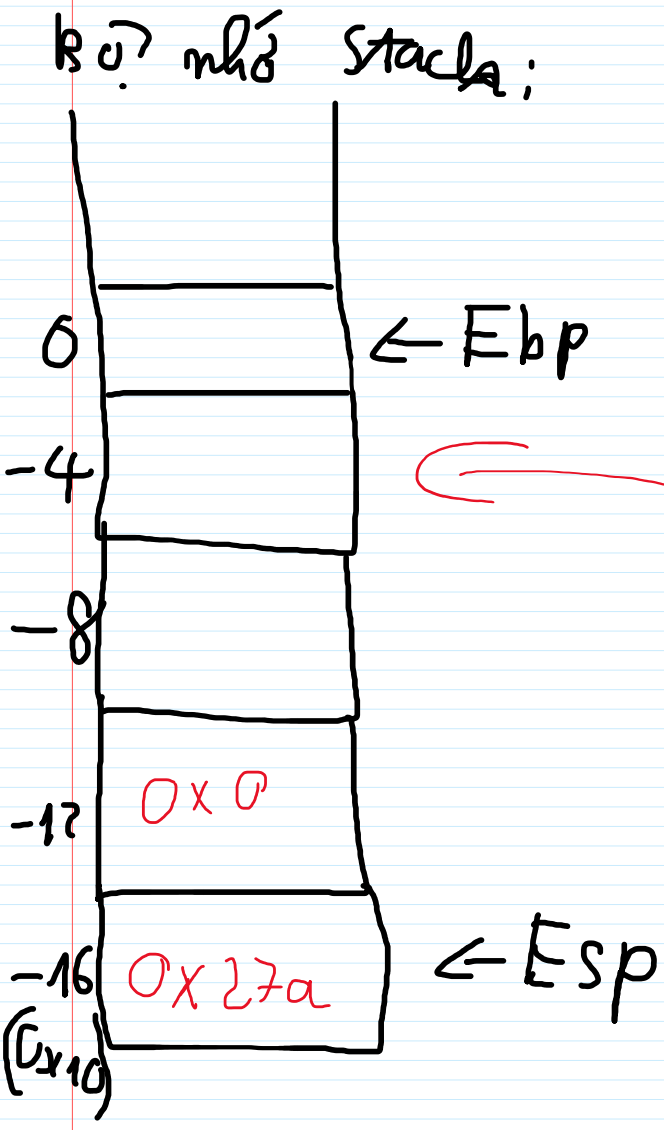
Assembly #2

Saturday, May 24, 2025 1:09 PM

```
<+0>:    push   ebp
   <+1>:    mov    ebp,esp
   <+3>:    push   ebx
   <+4>:    sub    esp,0x10
```

Đoạn này trong assembly dùng để cho bạn biết rằng chúng ta khởi tạo stack để lưu value sẽ Được thêm vào như thế nào, <+4> là để dời thanh con trỏ xuống 0x10 (hex)




```
<+7>:    mov    DWORD PTR [ebp-0x10],0x27a
<+14>:   mov    DWORD PTR [ebp-0xc],0x0
```

```
<+21>:   jmp    0x518 <asm4+27>
```

<+21> : đoạn này cho ta biết được rằng chương trình sẽ jump vào hàm 0x518 tức là
Của function asm4 (có base address là xxxxx) thì ta sẽ truy cập vào offset có value xxxxx + 27.

```
<+23>:   add    DWORD PTR [ebp-0xc],0x1
<+27>:   mov    edx,DWORD PTR [ebp-0xc]
<+30>:   mov    eax,DWORD PTR [ebp+0x8]
<+33>:   add    eax,edx
<+35>:   movzx  eax,BYTE PTR [eax]
<+38>:   test   al,al
<+40>:   jne    0x514 <asm4+23>
```

Ok, sau khi đã hiểu lệnh trên thì ta sẽ có dòng lệnh sau đây, như đã biết ta đang trong hàm asm4 nên ta khi thực hiện lệnh jump mình sẽ bay thẳng qua <+27> mà không thực hiện lệnh add của <+23>. Vậy ta có gì?

<+27> tương tự : int a = ebp[0-0xc] tức là bạn sẽ gán value của thanh ebp tại vị trị -12 cho thằng edx

Vậy Edx = 0x0.

<30> Hmm dòng này khá thú vị vì thanh [ebp + 0x8] = 0x70 á, Vì sau ư ? Rõ ràng ta truyền vào hàm một string "picoCTF_f97bb" . Mà bản chất của string là tập hợp nhiều char mà đã là char thì chắc chắn sẽ có trong bảng mã ASCII và p chính là 0x70 (nếu đổi sang dec).

Okay, vậy ebp + 0x8 chính là con trỏ trỏ vào giá trị đầu tiên của string (tương tự s[0]).

Vậy ta sẽ hiểu eax lúc này đang trỏ vào 'p'

Okay, tiếp tục phân tích.

<+33> dòng này chỉ là phép cộng bình thường giữa rồi gán vào eax thôi

eax = 0x70 + 0 = 0x70

<+35> movzx : là kiểu gán nhưng mở rộng thêm byte. Và ta gán value của nó cho nó

và lệnh này chỉ là gán 1 byte của giá trị eax vào eax nhưng mà được mở rộng với zero-extend thoai.

<+38> test al,al . What's that?

Nói tóm tắt thì nó là so sánh bit (a & a) trong C/C++ thì nó ko lưu value mà chỉ cập nhật cây flag thôi.

Trong case này al != 0 --> flag = 1;

<+40> jne   0x514 <asm4+23> :

--> jne (jump if not equal) , lệnh này nghĩa là nếu như flag = 0 thì sẽ nhảy vào hàm 0x514 và nó nhảy vì flag= 0.

Tới đây nếu đọc kĩ đoạn này

```
<+23>:   add    DWORD PTR [ebp-0xc],0x1
   <+27>:   mov    edx,DWORD PTR [ebp-0xc]
```

```
<+27>:   mov    edx,DWORD PTR [ebp-0xc]
<+30>:   mov    eax,DWORD PTR [ebp+0x8]
<+33>:   add    eax,edx
<+35>:   movzx  eax,BYTE PTR [eax]
<+38>:   test   al,al
<+40>:   jne    0x514 <asm4+23>
```

Thì nó không khác gì một vòng for/while cả, nhưng vấn đề là loop dùng để làm gì ?

Ok cùng phân tích , do giá trị DWORD PTR [ebp-0xc] lặp lại liên tục nên mình sẽ đặt là i cho dễ biết nha.

Còn DWORD PTR [ebp+0x8] thì cx liên tục cơ mà dòng này là sao nhỉ ? Sao nó cứ gán lại cho eax ?

--> Theo như ta biết eax là 1 con trỏ và nó trỏ vào [ebp+0x8] nên ta có thể hiểu là mỗi vòng lặp thì nó sẽ tự động quay lại s[0].

Ok. Modify outline pseodu code thử :

Do i là biến toàn cục được khai báo dòng <+14> = 0 nên ==> int i =0;

Address	Int i =0
<+23>	add -- > ++i;
<+27>	Edx = i;
<+30>	Eax = *s (eax = s[0] nhưng vì là con trỏ nên phải là *s)
<+33>	Eax = eax + edx; (eax += edx;) --> eax = eax + i;
<+35>	Chỉ là gán lại 1 byte rồi mở rộng với zero-extend
<+38>	Vì flag luôn = 0 nên ==> chỉ thay đổi khi không thỏa mãn. If(al != al) return flag = 1;
<40>	If(flag == 1) break;

Ok tô kê, outline có rồi, viết code thử nhe :

```
char *t = "picoCTF_f97bb";
For(int i =1; *(eax+i) == '\0' ;i++)
{
    Char *eax = t;
}
```

Hmm, giải thích 1 tí là vì chuỗi là 1 loạt các kí tự sắp xếp với nhau có thứ tự nên không thể dùng *string mà phải dùng *char nha.

Thứ 2 là eax == nullptr là gì ? Như đã đề cập thì eax là 1 con trỏ và nó trỏ vào giá trị đầu của chuỗi ,nên khi mà nó duyệt mà = con trỏ null thì tức là lúc này (al != al) đó.

ủa mà == thì nó nghịch rồi? . Thật chất là không vì test là phép '&' (tức là al & al) và 2 phần tử chỉ trả về 0 <=> nó al = 0 thôi. Vậy khi nào nó = 0 thì cờ mới được bật lên thì đây cũng tương tự chỉ là mình cho nó khi nào = 0 thì mới break.

Thôi thì đem cái chỗ break làm điều kiện dừng luôn cho gọn và dễ hiểu.

Rewrite ;

```
Char *t = "picoCTF_f97bb";
Int I =1;
While(*(t + i) != '\0')
{
    ++I;
}
Return I;
```

Vậy i sẽ dừng ở value nào ?

tất nhiên là len(t) rồi dk ?

Vậy nó sẽ là i = 13.

Tức là [ebp-0xc] =13.

```
<+42>:   mov    DWORD PTR [ebp-0x8],0x1
<+49>:   jmp    0x587 <asm4+138>
<+51>:   mov    edx,DWORD PTR [ebp-0x8]
<+54>:   mov    eax,DWORD PTR [ebp+0x8]
<+57>:   add    eax,edx
<+59>:   movzx  eax,BYTE PTR [eax]
<+62>:   movsx  edx,al
<+65>:   mov    eax,DWORD PTR [ebp-0x8]
```

Đây là các lệnh cơ bản mà mình đã phân tích rồi. Không giải thích thêm nhe.

Pseudo code :

[ebp - 0x8] = 0x1;

Nhảy thẳng tới thẳng <+138> luôn không thực hiện lệnh dưới

```
<+138>: mov    eax,DWORD PTR [ebp-0xc]
<+141>: sub    eax,0x1
<+144>: cmp    DWORD PTR [ebp-0x8],eax
<+147>: jl     0x530 <asm4+51>
<+149>: mov    eax,DWORD PTR [ebp-0x10]
<+152>: add    esp,0x10
<+155>: pop    ebx
<+156>: pop    ebp
<+157>: ret
```

Eax = 13;

Eax = 13-1 = 12;

So sánh eax và ebp-0x8

If([eax] < [ebp-0x8]) thì nhảy vào lệnh +51.

Eax = 12 và ebp = 0;

!Nhỏ thiệt nên nó không nhảy lên +51 mà thực hiện tiếp

<+149> eax = 0x27a

<+152> trả con trỏ stack về giá trị cũ

Kết thúc chương trình.