BÁO CÁO THỰC HÀNH

LẬP TRÌNH HỆ THỐNG

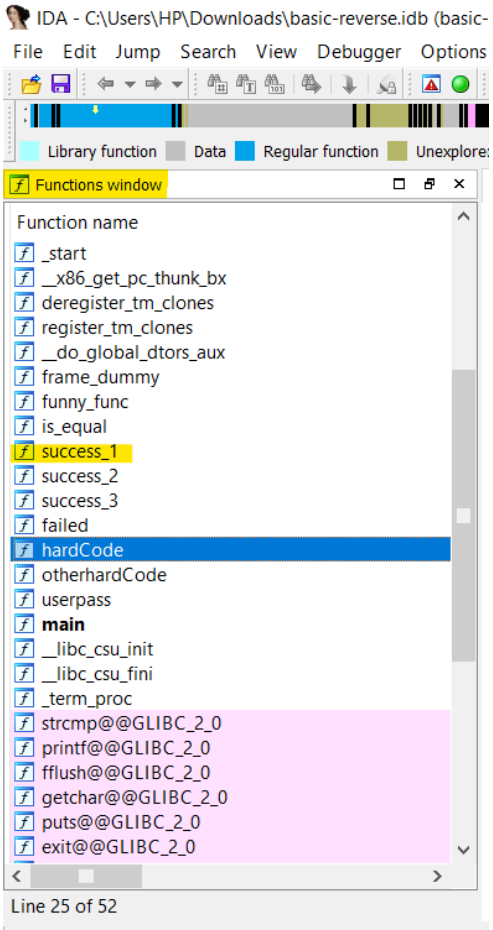
Tên bài Thực hành: Lab 04 - Lớp: NT209.P12.ANTT

Giáo viên hướng dẫn: Đỗ Thị Thu Hiền

Họ và tên sinh viên	MSSV
Nguyễn Trần Minh Khôi	23520780
Lê Đăng Khôi	23520766
Vương Thành Đạt	23520281

Yêu cầu 1. Phân tích và tìm ***passphrase cố định*** của **basic-reverse** với option 1. Báo cáo phương pháp phân tích, input tìm được và hình ảnh minh chứng chạy file.

- Với **option 1**, để có được đáp án ta cần tìm hàm tên **success_1** góc bên trái của thanh **Functions Window:**



- Vào hàm **success_1**, ta thấy:

```
.text:080486D2
.text:080486D2                  public success_1
.text:080486D2 success_1        proc near               ; CODE XREF: hardCode+67↓p
.text:080486D2                  push    ebp
.text:080486D3                  mov     ebp, esp
.text:080486D5                  sub     esp, 8
.text:080486D8                  sub     esp, 0Ch
.text:080486DB                  push    offset s        ; "Congrats! You found the hard-coded secr"...
.text:080486E0                  call    _puts
.text:080486E5                  add     esp, 10h
.text:080486E8                  nop
.text:080486E9                  leave
.text:080486EA                  retn
.text:080486EA success_1        endp
.text:080486EA
.text:080486EB
```

- Hàm được chia thành **5 phần**:
    - Phần đầu:
        - **proc near**: xác định đây là hàm có phạm vi truy cập gần (cùng đoạn mã trong bộ nhớ).
        - **push ebp**: lưu giá trị thanh ghi ebp hiện tại vào stack. Thanh ghi ebp thường được dùng làm điểm tham chiếu để truy cập biến cục bộ và tham số trong stack.
        - **mov ebp, esp:** gán giá trị của esp (stack pointer) vào ebp (base pointer) để thiết lập một khung stack (stack frame) mới cho hàm. Điều này giúp quản lý biến cục bộ và tham số một các dễ dàng.
    - Phần cấp phát bộ nhớ trên stack:
        - **sub esp, 8:** trừ đi 8 bytes từ esp, tạo không gian trên stack để lưu biến cục bộ hoặc sử dụng cho các hoạt động khác.
        - sub esp, 0Ch: 0Ch ở đây tức là 0xC hệ hexa, trừ tiếp 12 bytes để mở rộng stack. Tổng cộng hàm này cấp phát 20 bytes trên stack cho các biến cục bộ và dữ liệu tạm thời.
    - Phần hiển thị chuỗi ký tự:
        - **push offset s ; "Congrats! You found the hard-coded secr"...**
            - Đẩy địa chỉ của chuỗi ký tự (cụ thể là " Congrats!..." vào hàm stack làm tham số cho hàm _puts.
        - **call _puts:** gọi hàm _puts trong thư viện chuẩn C, có nhiệm vụ in chuỗi ký tự ra màn hình. Địa chỉ chuỗi được truyền qua stack.
    - Phần giải phóng bộ nhớ trến stack:
        - **add esp, 10h:** 10h ở đây tức là 0x10 hệ hexa, tăng giá trị esp lên 16 bytes, giải phóng không gian mà trước đó đã được dùng cho các tham số và biến tạm thời. Điều này giúp đưa con trỏ stack trở về trạng thái trước khi gọi hàm _puts.
    - Phần kết thúc hàm:
        - **nop:** không thực hiện thao tác nào, thường dùng để căn chỉnh hoặc làm chậm trễ.
        - **leave:** thực hiện 2 lệnh:
            - Gán giá trị của ebp trở lại cho esp (phục hồi con trỏ stack về trạng thái ban đầu của hàm)
            - pop (lấy) giá trị của ebp từ stack, phục hồi giá trị ebp ban đầu

- **retn:** kết thúc hàm và trả quyền điều khiển lại cho hàm gọi (return)

- Tiếp đến, ta phải kiểm tra xem có hàm nào gọi hàm **success_1** không, để tìm **điều kiện cần** của hàm được gọi.
- Sau khi thử tìm kiếm qua góc bên trái của thanh **Functions Window**, ta thấy hàm **hardCode** có gọi hàm **success_1:**

```
.text:08048736 ; Attributes: bp-based frame
.text:08048736
.text:08048736                 public hardCode
.text:08048736 hardCode        proc near               ; CODE XREF: main+4E↓p
.text:08048736
.text:08048736 s1              = byte ptr -3F0h
.text:08048736
.text:08048736                 push    ebp
.text:08048737                 mov     ebp, esp
.text:08048739                 sub     esp, 3F8h
.text:0804873F                 call    _getchar
.text:08048744                 sub     esp, 0Ch
.text:08048747                 push    offset aEnterTheHardCo ; "Enter the hard-coded password (option 1"...
.text:0804874C                 call    _puts
.text:08048751                 add     esp, 10h
.text:08048754                 sub     esp, 8
.text:08048757                 lea     eax, [ebp+s1]
.text:0804875D                 push    eax
.text:0804875E                 push    offset asc_804926A ; "%[^\n]"
.text:08048763                 call    ___isoc99_scanf
.text:08048768                 add     esp, 10h
.text:0804876B                 sub     esp, 8
.text:0804876E                 lea     eax, [ebp+s1]
.text:08048774                 push    eax
.text:08048775                 push    offset format   ; "Your input hard-coded password: %s\n"
.text:0804877A                 call    _printf
.text:0804877F                 add     esp, 10h
.text:08048782                 sub     esp, 8
.text:08048785                 push    offset s2       ; "New one in, old one out"
.text:0804878A                 lea     eax, [ebp+s1]
.text:08048790                 push    eax             ; s1
.text:08048791                 call    _strcmp
.text:08048796                 add     esp, 10h
.text:08048799                 test    eax, eax
.text:0804879B                 jnz     short loc_80487A4
.text:0804879D                 call    success_1
.text:080487A2                 jmp     short loc_80487A9
```

- Hàm được chia thành **5 phần**:
    - Thiết lập khung stack:
        - **s1 = byte prt –3F0h**: Biểu thị rằng biến s1 là một con trỏ trỏ đến vùng nhớ hoặc địa chỉ nào đó dựa trên offset hiện có ở đây là 3F0 hệ hexa thông qua con trỏ stack hoặc base.
        - **push ebp**: Lưu giá trị hiện tại của **ebp** vào stack.
        - **mov ebp, esp**: Thiết lập **ebp** để làm điểm tham chiếu cố định cho khung stack.
        - **sub esp, 3F8h**: Cấp phát **1016 bytes (0x3F8)** trên stack để làm bộ nhớ cục bộ. Bộ nhớ này sẽ chứa các biến tạm thời, như **s1**.
    - Gọi hàm nhập 1 ký tự (stdin):
        - **call _getchar**: Gọi hàm **getchar**, có thể để xóa ký tự rác (như ký tự newline \n) trong bộ đệm đầu vào.
        - **sub esp, 0Ch**: Cấp thêm **12 bytes** (0x0C)trên stack (cho mục đích căn chỉnh bộ nhớ).
        - **push offset aEnterTheHardCo**: Đẩy địa chỉ của chuỗi **"Enter the hard-coded password..."** lên stack.

- **call _puts**: Gọi hàm **puts** để in chuỗi ra màn hình.
- **add esp, 10h**: Giải phóng **16 bytes** (bao gồm địa chỉ chuỗi được truyền).

- Nhập password:
  - **sub esp, 8**: Cấp phát thêm không gian trên stack.
  - **lea eax, [ebp+s1]**: Lấy địa chỉ của biến **s1** (lưu trong khung stack) vào thanh ghi **eax**.
  - **push eax**: Đẩy địa chỉ của **s1** lên stack.
  - **push offset asc_804926A**: Đẩy định dạng **"%[^\n]"** lên stack (định dạng này đọc chuỗi đầu vào cho đến khi gặp ký tự newline \n).
  - **call ___isoc99_scanf**: Gọi hàm **scanf** để đọc chuỗi từ người dùng và lưu vào **s1**.
  - **add esp, 10h**: Giải phóng không gian dùng cho tham số hàm.
- In password:
  - **sub esp, 8**: Cấp phát không gian tạm thời trên stack.
  - **lea eax, [ebp+s1]**: Lấy địa chỉ của **s1** (chuỗi người dùng nhập).
  - **push eax**: Đẩy địa chỉ này lên stack.
  - **push offset format**: Đẩy định dạng **"Your input hard-coded password: %s\n"** lên stack.
  - **call _printf**: Gọi hàm **printf** để hiển thị chuỗi người dùng nhập.
  - **add esp, 10h**: Giải phóng không gian của các tham số.
- So sánh chuỗi password:
  - **sub esp, 8**: Cấp thêm không gian trên stack.
  - **push offset s2**: Đẩy địa chỉ của chuỗi **"New one in, old one out"** (hard-coded password) lên stack.
  - **lea eax, [ebp+s1]**: Lấy địa chỉ chuỗi người dùng nhập (**s1**).
  - **push eax**: Đẩy địa chỉ này lên stack.
  - **call _strcmp**: Gọi hàm **strcmp** để so sánh hai chuỗi. Trả về 0 nếu hai chuỗi khớp nhau.
  - **add esp, 10h**: Giải phóng không gian dùng cho tham số.
  - **test eax, eax**: Kiểm tra giá trị trả về của **strcmp**:
    - **eax = 0**: Chuỗi khớp.
    - **eax ≠ 0**: Chuỗi không khớp.
  - **jnz short loc_80487A4**: Nếu **eax != 0**, nhảy tới đoạn xử lý lỗi.
  - **call success_1**: Nếu **eax = 0**, gọi hàm **success_1** (hiển thị thông báo thành công).
  - **jmp short loc_80487A9**: Nhảy tới phần cuối để kết thúc.

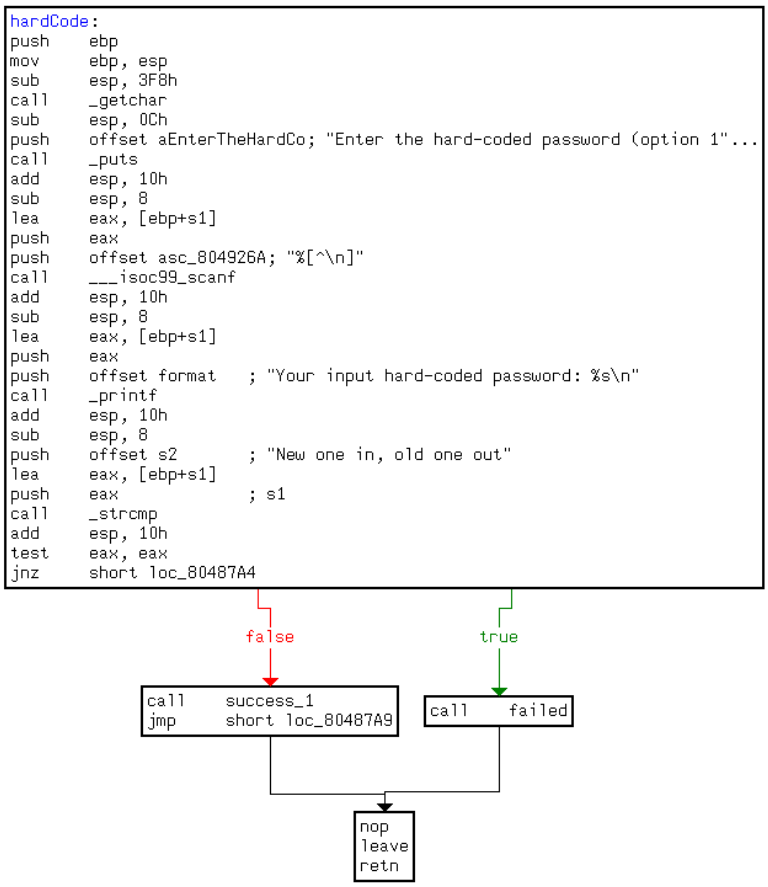
```
.text:080487A9 loc_80487A9:                            ; CODE XREF: hardCode+6C↑j
.text:080487A9                 nop
.text:080487AA                 leave
.text:080487AB                 retn
.text:080487AB hardCode        endp
```

- Ta thấy hàm **loc_80487A9** có chức năng tương tự như kết thúc hàm.

- Vậy khi so sánh bằng **strcmp** trả về **true** sẽ gọi hàm **success_1.** Do đó, đáp án ở đây là

"New one in, old one out"

- **Flow chart:**



- Kết quả:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ./basic-reverse
Supported authentication methods:
1. Hard-coded password
2. A pair of 2 numbers
3. Username/password
Enter your choice: 1
Enter the hard-coded password (option 1):
New one in, old one out
Your input hard-coded password: New one in, old one out
Congrats! You found the hard-coded secret, good job :).
```

> **Yêu cầu 2.** Phân tích và tìm ***cặp số nguyên*** của **basic-reverse** với option 2. Báo cáo phương pháp phân tích, input tìm được và hình ảnh minh chứng chạy file.

- Giải thích:

Mảng funny_seq:

```
.rodata:08048B60 funny_seq       dd 0
.rodata:08048B64                 db   2
.rodata:08048B65                 db   0
.rodata:08048B66                 db   0
.rodata:08048B67                 db   0
.rodata:08048B68                 db   4
.rodata:08048B69                 db   0
.rodata:08048B6A                 db   0
.rodata:08048B6B                 db   0
.rodata:08048B6C                 db   6
.rodata:08048B6D                 db   0
.rodata:08048B6E                 db   0
.rodata:08048B6F                 db   0
.rodata:08048B70                 db   8
.rodata:08048B71                 db   0
.rodata:08048B72                 db   0
.rodata:08048B73                 db   0
.rodata:08048B74                 db   1
.rodata:08048B75                 db   0
.rodata:08048B76                 db   0
.rodata:08048B77                 db   0
.rodata:08048B78                 db   3
.rodata:08048B79                 db   0
.rodata:08048B7A                 db   0
.rodata:08048B7B                 db   0
.rodata:08048B7C                 db   5
.rodata:08048B7D                 db   0
.rodata:08048B7E                 db   0
.rodata:08048B7F                 db   0
.rodata:08048B80                 db   7
.rodata:08048B81                 db   0
.rodata:08048B82                 db   0
.rodata:08048B83                 db   0
.rodata:08048B84                 db   9
.rodata:08048B85                 db   0
.rodata:08048B86                 db   0
.rodata:08048B87                 db   0
```

Flow_chart otherhardCode:

```
1  int otherhardCode()
2  {
3    int v0; // edx@2
4    int result; // eax@3
5    int v2; // [sp+4h] [bp-14h]@1
6    int v3; // [sp+8h] [bp-10h]@1
7    int v4; // [sp+Ch] [bp-Ch]@1
8
9    getchar();
10   puts("Enter your 2 numbers (separated by space) (option 2):");
11   __isoc99_scanf("%d %d", &v3, &v2);
12   printf("Your input: %d %d\n", v3, v2);
13   v4 = 8;
14   if ( v3 == 8 )
15   {
16     v0 = funny_func(*(&funny_seq + 8), 8);
17     if ( v0 == v2 )
18       result = success_2();
19     else
20       result = failed();
21   }
22   else
23   {
24     result = failed();
25   }
26   return result;
27 }
```

Funny_func:

IDA View-A | Pseudocode-B | Pseu

```
1  int __cdecl funny_func(int a1, int a2)
2  {
3    return a2 + a1 - 2 * a1;
4  }
```

- Để tìm được cặp số nguyên, ta cần phân tích code của hàm **otherhardCode** và các hàm con của nó.

Nhập:

Câu lệnh **__isoc99_scanf("%d %d", &v3, &v2)** trong hàm hardCode sẽ gọi hàm **scanf** và gán hai số nguyên mà người dùng nhập vào các biến cục bộ v3 và v2, tạm thời gọi 2 giá trị được nhập vào v3 và v2 là x1 và x2.

Kiểm tra x1:

Câu lệnh if đầu tiên kiểm tra giá trị x1 nhập vào có khớp với giá trị cho trước không, nếu có thì tiếp tục thực hiện kiểm tra x2, nếu không thì báo **failed**. Và giá trị cho trước trong mệnh đề điều kiện của if là 8 --> x1 = 8.

Kiểm tra x2:

Sau khi thỏa điều kiện ở hàm if thứ nhất, chương trình gọi hàm **funny_func** và lưu giá trị trả về vào v0. Hàm **funny_func** có 2 tham số truyền vào, tham số đầu tiên được truy xuất thông qua **mảng funny_seq** là ***(&funny_seq + 8)**, có địa chỉ = &funny_seq + 8 byte = 0x08048B60 + 8*4 = 0x08048B60 + 0x20 = 0x08048B80 --> tham số đầu tiên là 7 và tham số thứ hai là 8.

Sau khi thực hiện hàm **funny_func**, kết quả trả về là: 7 + 8 - 2 * 7 = 1 --> v0 = 1.

Câu lệnh if thứ 2 này có nhiệm vụ kiểm tra giá trị x2, nếu bằng giá trị lưu trong v0 thì sẽ thực hiện gọi hàm success_2 --> x2 = 1.

Vậy cặp số nguyên cần tìm là **8 1**.

- Kết quả:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ./basic-reverse
Supported authentication methods:
1. Hard-coded password
2. A pair of 2 numbers
3. Username/password
Enter your choice: 2
Enter your 2 numbers (separated by space) (option 2):
8
1
Your input: 8 1
Congrats! You found a secret pair of numbers :).
```

Yêu cầu 3. Phân tích, tìm **cặp *username/password*** phù hợp của **basic-reverse** với option 3. Báo cáo phương pháp phân tích, input tìm được và hình ảnh minh chứng chạy file.

Lưu ý bắt buộc: ***username*** được tạo từ MSSV của các thành viên trong nhóm.

- Nhóm **3 sinh viên:** sắp xếp 3 MSSV theo thứ tự tăng dần rồi lấy 3 số cuối nối nhau. Ví dụ 22520021 < 23520123 < 23521013 sẽ có username là **021123013**.
- Nhóm **2 sinh viên:** sắp xếp 2 MSSV theo thứ tự tăng dần rồi lấy 4 số cuối nối nhau bằng dấu "-". Ví dụ 23520123 < 23521013 sẽ có username là **0123-1013**.
- Nhóm có **1 sinh viên** có MSSV là 2352xxxx thì username là **2352-xxxx**.

- Giải thích:
  - Code bằng mả giả:

```
1  int userpass()
2  {
3    size_t v0; // ebx@2
4    int result; // eax@3
5    long double v2; // fst7@13
6    size_t v3; // eax@15
7    size_t v4; // edx@16
8    char v5[9]; // [sp+1Ah] [bp-2Eh]@6
9    char v6[10]; // [sp+23h] [bp-25h]@1
10   char s[10]; // [sp+2Dh] [bp-1Bh]@1
11   char v8; // [sp+37h] [bp-11h]@1
12   char v9; // [sp+38h] [bp-10h]@1
13   char v10; // [sp+39h] [bp-Fh]@1
14   char v11; // [sp+3Ah] [bp-Eh]@1
15   char v12; // [sp+3Bh] [bp-Dh]@1
16   unsigned int i; // [sp+3Ch] [bp-Ch]@4
17
18   v8 = 95;
19   v9 = 125;
20   v10 = 118;
21   v11 = 93;
22   v12 = 93;
23   getchar();
24   puts("Enter your username:");
25   __isoc99_scanf("%[^\n]", s);
26   getchar();
27   puts("Enter your password:");
28   __isoc99_scanf("%[^\n]", v6);
29   printf("Your input username: %s and password: %s\n", s, v6);
30   if ( strlen(s) == 9 && (v0 = strlen(s), v0 == strlen(v6)) )
```

```
{
  for ( i = 0; (signed int)i <= 8; ++i )
  {
    if ( (signed int)i > 1 )
    {
      if ( (signed int)i > 3 )
        v5[i] = *(&v8 + i - 4);
      else
        v5[i] = s[i + 5];
    }
    else
    {
      v5[i] = s[i + 2];
    }
  }
  for ( i = 0; ; ++i )
  {
    v3 = strlen(s);
    if ( v3 > i )
    {
      v2 = ceil((long double)((s[i] + v5[i]) / 2));
      if ( (long double)v6[i] == v2 )
        continue;
    }
    break;
  }
  v4 = strlen(s);
  if ( v4 == i )
    result = success_3();
  else
    result = failed();
}
else
{
  result = failed();
}
return result;
}
```

- Dựa theo mã giả ta thấy được các biến từ v8 đến v12 là các biến được khai báo sẵn mang các giá trị {95,125,118,93,93}.
- Hàm **__isoc99_scanf** sẽ gọi thực thi tương ứng lệnh scanf trong ngôn ngữ C để nhập vào username và lưu trong biến s.
- Tương tự gọi hàm này một lần nữa để nhận vào **password** từ người dùng nhập vào và lưu vào trong biến v6.
- Ý tưởng chính của hàm **userpass()**:
    - Đầu tiên sẽ kiểm tra xem độ dài của **username** nhập vào có đúng bằng 9 hay không, tiếp theo sẽ kiểm tra độ dài của **username** và **password** có bằng nhau hay không.
    - Nếu có, thì tiếp tục thực thi các lệnh mã hóa của đoạn lệnh.
    - Nếu không thì trả về kết quả trong hàm **failed()**.
    - Trong hàm này ta thấy v5 sẽ là một mảng số nguyên để chứa các dữ liệu được tính toán với logic như sau:
        - i từ 0-1: v5[i] = s[i+2]
        - i từ 1-2: v5[i] = s[i+5]
        - i từ 4-8: v5[i] = v8[i-4]

```
v5[0] = s[2] = 1 = 49
v5[1] = s[3] = 7 = 55
v5[2] = s[7] = 8 = 56
v5[3] = s[8] = 0 = 48
v5[4] = v8[0] = 95
v5[5] = v8[1] = 125
v5[6] = v8[2] = 118
v5[7] = v8[3] = 93
v5[8] = v8[4] = 93
```

- Với vòng for thứ 2: for (i = 0; ++i) : Điều kiện để hàm trả về kết quả đúng là vòng lặp này cần chạy đủ 9 lần với điều kiện là những phần tử trong mảng v6 phải bằng kết quả tính toán:
  - **v2 = ceil((long double)((s[i] + v5[i]) / 2))**, kết quả được làm tròn xuống.
- Kết quả thu được như sau:

```
i = 0 -> v2 = (s[0] + v5[0]) / 2 = (50+49)/2 = 49 = v6[0]
i = 1 -> v2 = (s[1] + v5[1]) / 2 = (56+55)/2 = 55 = v6[1]
i = 2 -> v2 = (s[2] + v5[2]) / 2 = (49+56)/2 = 52 = v6[2]
i = 3 -> v2 = (s[3] + v5[3]) / 2 = (55+48)/2 = 51 = v6[3]
i = 4 -> v2 = (s[4] + v5[4]) / 2 = (54+95)/2 = 74 = v6[4]
i = 5 -> v2 = (s[5] + v5[5]) / 2 = (54+125)/2 = 89 = v6[5]
i = 6 -> v2 = (s[6] + v5[6]) / 2 = (55+118)/2 = 86 = v6[6]
i = 7 -> v2 = (s[7] + v5[7]) / 2 = (56+93)/2 = 74 = v6[7]
i = 8 -> v2 = (s[8] + v5[8]) / 2 = (48+93)/2 = 70 = v6[8]
```

- Tương ứng với từng phần tử trong mảng v6 là những mã **ASCII** của các ký tự trong **password** cần tìm kiếm.
- Vòng lặp này chỉ chạy đủ 9 lần khi và chỉ khi từng ký tự trong **password** được nhập vào tương ứng với từng ký tự mà hàm trên tính toán ra được.
- Nếu thành công thì chương trình trả về đoạn string mà hàm **success_3** thực thi để in ra màn hình với input đầu vào thỏa yêu cầu bài toán.
- Nhãn **aCongratsYouF_1: “Congrats! You found your own username/password pair :).”**

- Kết quả:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ./basic-reverse
Supported authentication methods:
1. Hard-coded password
2. A pair of 2 numbers
3. Username/password
Enter your choice: 3
Enter your username:
281766780
Enter your password:
1743JYVJF
Your input username: 281766780 and password: 1743JYVJF
Congrats! You found your own username/password pair :).
```