



TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN
MÔN: KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ
LỚP: 16CNTN

BÁO CÁO ĐỒ ÁN 3

HỢP NGỮ X86 CRACK PHẦN MỀM

Người thực hiện:

Nguyễn Quốc Vương – 1612829

Nguyễn Thanh Tuấn – 1612774

Lê Thành Công – 1612842

Giáo viên:

Lý thuyết: Phạm Tuấn Sơn

Thực hành: Nguyễn Thanh Quân

Mục lục

I. Tổng quan đồ án.....	2
1. Yêu cầu.....	2
2. Đề bài.....	2
3. Mức độ hoàn thành.....	2
II. Giải pháp.....	2
1. Phần 1.....	2
a. Câu 1.1.exe.....	2
b. Câu 2.1.exe.....	5
c. Câu 2.2.exe.....	22
d. Câu 3.1.exe.....	32
2. Phần 2.....	45

I. Tổng quan đồ án:

1. Yêu cầu:

- Phần 1: Với mỗi crackme sinh viên phải chỉ ra đoạn phát sinh key, giải thích ý nghĩa và đưa ra một key tương ứng với user name mình họa. (chụp hình minh họa). Viết chương trình keygen nếu có.
- Phần 2: Một sinh viên đại diện nhóm vào trang web "https://microcorruption.com/" và đăng ký tài khoản. Sinh viên cho biết cách thức khai thác lỗi từ level Tutorial tới Whitehorse (để vượt qua từng level, sinh viên phải tìm được key).

2. Đề bài:

$$(1612829 + 1612842 + 1612774) \% 4 + 1 = 2$$

⇒ **Làm đề 02.**

3. Mức độ hoàn thành:

STT	Nội dung		Tỷ lệ hoàn thành (%)
1	Phần 1	1.1.exe	100
2		2.1.exe	100
3		2.2.exe	100
4		3.1.exe	100
5	Phần 2	Microcorruption	100

⇒ **Đã hoàn thành 100% tất cả yêu cầu của đồ án.**

II. Giải pháp:

1. Phần 1: Crack 4 phần mềm:

a. Câu 1.1.exe:

-Đặt thử breakpoint tại 004014A6 . E8 5B080000 CALL
<JMP.&user32.GetDlgItemTextA> ; \GetDlgItemTextA

Thử nhập key là abcdefgh, key sẽ được lưu vào địa chỉ 0040332C

Ta thấy từ 00401506 - 0040155B, key sẽ được lấy để làm vài việc.

00401506	>	A1 2C334000	MOV EAX,DWORD PTR [40332C]
00401508	.	8B1D 30334000	MOV EBX,DWORD PTR [403330]
00401511	.	A3 E6324000	MOV DWORD PTR [4032E6],EAX
00401516	.	891D EA324000	MOV DWORD PTR [4032EA],EBX
0040151C	.	E8 C8040000	CALL 1_1.004019E9
00401521	.	893D 4B334000	MOV DWORD PTR [40334B],EDI
00401527	.	A1 4B334000	MOV EAX,DWORD PTR [40334B]
0040152C	.	8B1D 30334000	MOV EBX,DWORD PTR [403330]
00401532	.	A3 E6324000	MOV DWORD PTR [4032E6],EAX
00401537	.	33C0	XOR EAX,EAX
00401539	.	891D EA324000	MOV DWORD PTR [4032EA],EBX
0040153F	.	60	PUSHAD
00401540	.	68 ADDE0000	PUSH 0DEAD
00401545	.	58	POP EAX
00401546	.	68 EFBE0000	PUSH 0BEEF
00401548	.	58	POP EBX
0040154C	.	68 AFAAA0A	PUSH 0AAAAAF
00401551	.	59	POP ECX
00401552	.	0FC9	BSWAP ECX
00401554	.	0FC9	BSWAP ECX
00401556	.	61	POPAD
00401557	.	0C 01	OR AL,1
00401559	.	0BC0	OR EAX,EAX
0040155B	~	0F85 8F000000	JNZ 1_1.004015F0

-Đề ý từ 00401557-0040155B, câu lệnh này luôn nhảy, vì vậy, dòng lệnh để hiển thị "Your are Registered" sẽ không được gọi.

-Từ 00401598-004015EB, là đoạn lệnh cho phép hiện “Registered”. Làm sao để vào được đoạn lệnh này?

00401585	>	6A 40	PUSH 40	[Style = MB_OK!MB_ICONASTERISK!MB_APPLMODAL
00401587	.	68 7A154000	PUSH 1_1.0040157A	Title = "Good Work!"
0040158C	.	68 63154000	PUSH 1_1.00401563	Text = "Your are Registered!"
00401591	.	6A 00	PUSH 0	hOwner = NULL
00401593	.	E8 98070000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401598	>	6A 00	PUSH 0	Enable = FALSE
0040159A	.	FF35 60304000	PUSH DWORD PTR [403060]	hWnd = 00260D6C (class="Edit",parent=00390B7A)
004015A0	.	E8 31070000	CALL <JMP.&user32.EnableWindow>	EnableWindow
004015A5	.	6A 01	PUSH 1	Enable = TRUE
004015A7	.	FF35 60304000	PUSH DWORD PTR [403060]	hWnd = 00260D6C (class="Edit",parent=00390B7A)
004015AD	.	E8 24070000	CALL <JMP.&user32.EnableWindow>	EnableWindow
004015B2	.	6A 00	PUSH 0	Enable = FALSE
004015B4	.	FF35 64304000	PUSH DWORD PTR [403064]	hWnd = 00100DA6 (class="Edit",parent=00390B7A)
004015BA	.	E8 17070000	CALL <JMP.&user32.EnableWindow>	EnableWindow
004015BF	.	8D05 CC104000	LEA EAX,DWORD PTR [4010CC]	
004015C5	.	83E8 1F	SUB EAX,1F	
004015C8	.	50	PUSH EAX	[iParam => 4010AD
004015C9	.	6A 00	PUSH 0	wParam = 0
004015CB	.	6A 0C	PUSH 0C	Message = WM_SETTEXT
004015CD	.	FF75 08	PUSH DWORD PTR [EBP+8]	hWnd
004015D0	.	E8 67070000	CALL <JMP.&user32.SendMessageA>	SendMessageA
004015D5	.	8D05 AE324000	LEA EAX,DWORD PTR [4032AE]	
004015DB	.	83C0 0F	ADD EAX,0F	
004015DE	.	50	PUSH EAX	[iParam => 4032BD
004015DF	.	6A 00	PUSH 0	wParam = 0
004015E1	.	6A 0C	PUSH 0C	Message = WM_SETTEXT
004015E3	.	68 CA000000	PUSH 0CA	ControlID = CA (202.)
004015E8	.	FF75 08	PUSH DWORD PTR [EBP+8]	hWnd
004015EB	.	E8 46070000	CALL <JMP.&user32.SendDlgItemMessageA>	SendDlgItemMessageA

-Từ 00401675-004016A2, xử lý WM_TIMER message

00401675	.	6A 00	PUSH 0	Timerproc = NULL
00401677	.	68 D0070000	PUSH 7D0	Timeout = 2000. ms
0040167C	.	68 BE020000	PUSH 2BE	TimerID = 2BE (702.)
00401681	.	FF35 00304000	PUSH DWORD PTR [403000]	hWnd = 00390B7A ('Little Man 1.45 (Unregistered)',class='#32770')
00401687	.	E8 B0600000	CALL <JMP.&user32.SetTimer>	SetTimer
0040168C	.	A3 48304000	MOV DWORD PTR [403048],EAX	
00401691	.	60	PUSHAD	
00401692	.	A1 30334000	MOV EAX,DWORD PTR [403330]	
00401697	.	33DB	XOR EBX,EBX	
00401699	.	8A1D 31334000	MOV BL,BYTE PTR [403331]	
0040169F	.	80FB 2D	CMPL BL,2D	
004016A2	~	75 23	JNZ SHORT 1_1.004016C7	

-Nhận thấy từ **00401692-004016B8**, đây là đoạn lệnh có tác động tới 00403330 (tức là: 0040332C+4), vì thế nếu key là abcdefgh tại 0040332C thì tại 00403330 sẽ là efgh

00401692	. A1 30334000	MOV EAX,DWORD PTR [403330]
00401697	. 33DB	XOR EBX,EBX
00401699	. 8A1D 31334000	MOV BL,BYTE PTR [403331]
0040169F	. 80FB 2D	CMP BL,2D
004016A2	✓ 75 23	JNZ SHORT 1_1.004016C7
004016A4	. 3C 31	CMP AL,31
004016A6	✓ 74 05	JE SHORT 1_1.004016AD
004016A8	. 80FC 2D	CMP AH,2D
004016AB	✓ 75 1A	JNZ SHORT 1_1.004016C7
004016AD	> 0FC8	BSWAP EAX
004016AF	. 3C 53	CMP AL,53
004016B1	✓ 74 05	JE SHORT 1_1.004016B8
004016B3	. 80FC 38	CMP AH,38
004016B6	✓ 75 0F	JNZ SHORT 1_1.004016C7
004016B8	> C705 30334000	MOV DWORD PTR [403330],0

Ta phân tích đoạn mã trên với key là abcdefgh:

00401692 mov eax, [403330] //eax="hgfe"

00401697 xor ebx, ebx

00401699 mov bl, byte ptr [403331] //ebx='f'

0040169F cmp bl, 2Dh //so sánh bl=='-'

004016A2 jnz short loc_4016C7 //nếu không bằng, nhảy tới 4016C7

004016A4 cmp al, 31h //al=='1'? //ta không xét nhiều về so sánh này

004016A6 jz short loc_4016AD

004016A8 cmp ah, 2Dh //ah==bl=='-'

004016AB jnz short loc_4016C7

004016AD bswap eax //đảo => eax="efgh"

004016AF cmp al, 53h // al=='S'? => 'h'=='S' //nếu bằng => đúng

004016B1 jz short loc_4016B8

004016B3 cmp ah, 38h //hoặc ah=='8' => 'g'=='8' //nếu bằng => cũng đúng

004016B6 jnz short loc_4016C7

-Từ đoạn mã trên, ta có thể suy ra quy luật key cứng:

Key: *****-S*** (với * có thể là ký tự bất kỳ)

Hoặc:

Key: *****-8***** (với * có thể là ký tự bất kỳ)

Ví dụ:

Key: 12345-1S234

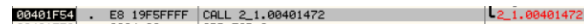
Hoặc:

Key: 12345-81234

b. Câu 2.1.exe:

-Run chương trình, ta đi vào lệnh ở địa chỉ

```
00401F54 . E8 19F5FFFF CALL 2_1.00401472 ;  
2_1.00401472
```

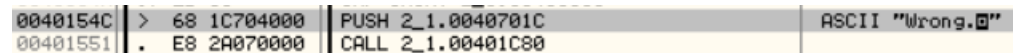


Ta có thể quan sát thấy các công việc nhập username, serial và kiểm tra, xử lý đều nằm ở đây.

-Good boy:



-Bad boy:



-Nhập thử username là abcde và serial là 12345 ta quan sát thấy username được lưu ở 0019FD44 và serial được lưu ở 0019FB44.

-Xét quá trình xử lý ở username:

0040148E	. 68 A1704000	PUSH 2_1.004070A1	[Arg1 = 004070A1 ASCII "Username (5-8 chars) : "
00401493	. E8 E8070000	CALL 2_1.00401C80	2_1.00401C80
00401498	. 59	POP ECX	
00401499	. 8D85 00FEFF	LEA EAX,DWORD PTR [EBP-200]	
0040149F	. 50	PUSH EAX	[Arg1
004014A0	. E8 0B080000	CALL 2_1.00401CB0	2_1.00401CB0
004014A5	. 59	POP ECX	0019FD44
004014A6	. 8D85 00FEFF	LEA EAX,DWORD PTR [EBP-200]	
004014AC	. 50	PUSH EAX	
004014AD	. E8 2E090000	CALL 2_1.00401DE0	
004014B2	. 59	POP ECX	
004014B3	. 83F8 08	CMP EAX,8	
004014B6	. ^ 77 C3	JA SHORT 2_1.0040147B	
004014B8	. 8D85 00FEFF	LEA EAX,DWORD PTR [EBP-200]	
004014BE	. 50	PUSH EAX	
004014BF	. E8 1C090000	CALL 2_1.00401DE0	
004014C4	. 59	POP ECX	
004014C5	. 83F8 05	CMP EAX,5	
004014C8	. ^ 72 B1	JB SHORT 2_1.0040147B	

+Sau khi nhập username, từ 004014A6-004014C8 là đoạn lệnh kiểm tra độ dài của chuỗi username nhập vào. Nếu độ dài nhỏ hơn 5 hoặc lớn hơn 8 thì nhảy ngược lên đoạn lệnh yêu cầu nhập username để tiến hành nhập lại, ngược lại, tiếp tục thực hiện các đoạn lệnh tiếp theo.

004014D1	. E8 2AFBFFFF	CALL 2_1.00401000
004014D6	. 59	POP ECX
004014D7	. E8 8AFBFFFF	CALL 2_1.00401066

+Tiếp theo, là 2 hàm quan trọng trong xử lý chuỗi username.

004014D1 |. E8 2AFBFFFF |CALL 2_1.00401000

Được xem như một hàm để khởi tạo các giá trị ban đầu, ta tạm đặt tên đây là hàm init.

00401000	\$. 8B4C24 04	MOV ECX,DWORD PTR [ESP+4]
00401004	. 8B01	MOV EAX,DWORD PTR [ECX]
00401006	. A3 9C8C4000	MOV DWORD PTR [408C9C],EAX
0040100B	. 8B41 04	MOV EAX,DWORD PTR [ECX+4]
0040100E	. A3 A08C4000	MOV DWORD PTR [408CA0],EAX
00401013	. C3	RET

Hàm init có nhiệm vụ cắt chuỗi username và lưu vào 00408C9C (seed1) và 00408CA0 (seed2). Ta xét cách chạy của hàm qua đoạn code C++ sau:

```
void init()
{
    string s = textToHex(this->name);
    s = dump(s);
    while (s.length() < 16)
    {
        s = "0" + s;
    }
    seed2 = s.substr(0, 8);
    seed1 = s.substr(8, 8);
}
```

Với hàm dump:

```
string dump(string s)
{
    string res = "";
    while (s.length() < 8)
    {
        s = "0" + s;
    }
    int n = s.length();
    for (int i = 0; i < n / 2; i++)
    {
        res += s.substr(s.length() - 2, 2);
        s.erase(s.length() - 2, 2);
    }
    return res;
}
```

VD: username là abcde (dạng hexa là: 6162636465)

Thì seed1 = 61 62 63 64 và seed2 = 65 00 00 00

Vì trong quá trình xử lý, các giá trị lưu trên được lấy lên các thanh ghi nên ta phải hiểu là nó sẽ bị đảo, nên ta cần có hàm dump để mô tả lại quá trình lấy giá trị lên thanh ghi.

004014D7 |. E8 8AFBFFFF |CALL 2_1.00401066

Dựa trên các seed vừa được hàm init khởi tạo, đây là hàm phát sinh mê cung (tạm gọi là hàm generate).

00401066	53	PUSH EBX
00401067	. 6A 14	PUSH 14
00401069	. E8 A6FFFFFF	CALL 2_1.00401014
0040106E	. 59	POP ECX
0040106F	. 83C0 14	ADD EAX,14
00401072	. A3 A48C4000	MOV DWORD PTR [408CA4],EAX
00401077	. 89C2	MOV EDX,EAX
00401079	. 0FAFD0	IMUL EDX,EAX
0040107C	. 89D0	MOV EAX,EDX
0040107E	. 50	PUSH EAX
0040107F	. E8 2C0B0000	CALL 2_1.00401BB0
00401084	. 59	POP ECX
00401085	. A3 A88C4000	MOV DWORD PTR [408CA8],EAX
0040108A	. 31DB	XOR EBX,EBX
0040108C	~ EB 21	JMP SHORT 2_1.004010AF
0040108E	> 6A 04	PUSH 4
00401090	. E8 7FFFFFFF	CALL 2_1.00401014
00401095	. 59	POP ECX
00401096	. 85C0	TEST EAX,EAX
00401098	~ 75 0B	JNZ SHORT 2_1.004010A5
0040109A	. A1 A88C4000	MOV EAX,DWORD PTR [408CA8]
0040109F	. C60418 23	MOV BYTE PTR [EAX+EBX],23
004010A3	~ EB 09	JMP SHORT 2_1.004010AE
004010A5	> A1 A88C4000	MOV EAX,DWORD PTR [408CA8]
004010AA	. C60418 20	MOV BYTE PTR [EAX+EBX],20
004010AE	> 43	INC EBX
004010AF	> A1 A48C4000	MOV EAX,DWORD PTR [408CA4]
004010B4	. 89C2	MOV EDX,EAX
004010B6	. 0FAFD0	IMUL EDX,EAX
004010B9	. 39D3	CMP EBX,EDX
004010BB	^ 7C D1	JL SHORT 2_1.0040108E
004010BD	. A1 A48C4000	MOV EAX,DWORD PTR [408CA4]
004010C2	. 50	PUSH EAX
004010C3	. E8 4CFFFFFF	CALL 2_1.00401014
004010C8	. 59	POP ECX
004010C9	. A3 AC8C4000	MOV DWORD PTR [408CAC],EAX
004010CE	. A1 A48C4000	MOV EAX,DWORD PTR [408CA4]
004010D3	. 50	PUSH EAX
004010D4	. E8 3BFFFFFF	CALL 2_1.00401014
004010D9	. 59	POP ECX
004010DA	. A3 B08C4000	MOV DWORD PTR [408CB0],EAX

```

004010DF > A1 A48C4000 MOV EAX,DWORD PTR [408CA4]
004010E4 . 50 PUSH EAX
004010E5 . E8 2AFFFFFF CALL 2_1.00401014
004010EA . 59 POP ECX
004010EB . A3 B48C4000 MOV DWORD PTR [408CB4],EAX
004010F0 . A1 AC8C4000 MOV EAX,DWORD PTR [408CAC]
004010F5 . 3905 B48C4000 CMP DWORD PTR [408CB4],EAX
004010FB . ^ 74 E2 JE SHORT 2_1.004010DF
004010FD > A1 A48C4000 MOV EAX,DWORD PTR [408CA4]
00401102 . 50 PUSH EAX
00401103 . E8 0CFFFFFF CALL 2_1.00401014
00401108 . 59 POP ECX
00401109 . A3 B88C4000 MOV DWORD PTR [408CB8],EAX
0040110E . A1 B08C4000 MOV EAX,DWORD PTR [408CB0]
00401113 . 3905 B88C4000 CMP DWORD PTR [408CB8],EAX
00401119 . ^ 74 E2 JE SHORT 2_1.004010FD
0040111B . A1 AC8C4000 MOV EAX,DWORD PTR [408CAC]
00401120 . 8B15 A48C4000 MOV EDX,DWORD PTR [408CA4]
00401126 . 0FAFC2 IMUL EAX,EDX
00401129 . 8B15 B08C4000 MOV EDX,DWORD PTR [408CB0]
0040112F . 01D0 ADD EAX,EDX
00401131 . 8B15 A88C4000 MOV EDX,DWORD PTR [408CA8]
00401137 . C60402 73 MOV BYTE PTR [EDX+EAX],73
0040113B . A1 B48C4000 MOV EAX,DWORD PTR [408CB4]
00401140 . 8B15 A48C4000 MOV EDX,DWORD PTR [408CA4]
00401146 . 0FAFC2 IMUL EAX,EDX
00401149 . 8B15 B88C4000 MOV EDX,DWORD PTR [408CB8]
0040114F . 01D0 ADD EAX,EDX
00401151 . 8B15 A88C4000 MOV EDX,DWORD PTR [408CA8]
00401157 . C60402 66 MOV BYTE PTR [EDX+EAX],66
0040115B . 5B POP EBX
0040115C . C3 RET

```

Tại 00401069 |. E8 A6FFFFFF CALL 2_1.00401014

Và liên tục những câu lệnh phía sau có xuất hiện CALL 2_1.00401014 để gọi một hàm khá quan trọng, ta tạm gọi là getBlock

```

00401014 $ S3 PUSH EBX
00401015 . A1 9C8C4000 MOV EAX,DWORD PTR [408C9C]
0040101A . BA C15D0000 MOV EDI,5DC1
0040101F . F7E2 MUL EDI
00401021 . B9 0B560000 MOV ECX,560B
00401026 . 31D2 XOR EDI,EDI
00401028 . F7F1 DIV ECX
0040102A . 8915 9C8C4000 MOV DWORD PTR [408C9C],EDI
00401030 . A1 A08C4000 MOV EAX,DWORD PTR [408CA0]
00401035 . BA 9D540000 MOV EDI,549D
0040103A . F7E2 MUL EDI
0040103C . B9 A1510000 MOV ECX,51A1
00401041 . 31D2 XOR EDI,EDI
00401043 . F7F1 DIV ECX
00401045 . 8915 A08C4000 MOV DWORD PTR [408CA0],EDI
0040104B . A1 9C8C4000 MOV EAX,DWORD PTR [408C9C]
00401050 . 8B15 A08C4000 MOV EDX,DWORD PTR [408CA0]
00401056 . 01D0 ADD EAX,EDX
00401058 . 8B4C24 08 MOV ECX,DWORD PTR [ESP+8]
0040105C . 31D2 XOR EDX,EDX
0040105E . F7F1 DIV ECX
00401060 . 89D3 MOV EBX,EDX
00401062 . 89D8 MOV EAX,EBX
00401064 . 5B POP EBX
00401065 . C3 RET

```

Mô tả hàm getBlock bằng đoạn code C++:

```

string getBlock(string stack)
{
    QInt x(seed1, "16");
    QInt y("5DC1", "16");
    string res1 = (x * y).toString("16");
    while (res1.length() < 16) { res1 = "0" + res1; }
    res1 = res1.substr(res1.length() - 8, 8);

    x.setData(res1, "16");
    y.setData("560B", "16");
    res1 = (x % y).toString("16");

    seed1 = res1;

    x.setData(seed2, "16");
    y.setData("549D", "16");
    string res2 = (x * y).toString("16");
    while (res2.length() < 16) { res2 = "0" + res2; }
    res2 = res2.substr(res2.length() - 8, 8);

    x.setData(res2, "16");
    y.setData("51A1", "16");
    res2 = (x % y).toString("16");

    seed2 = res2;

    x.setData(res1, "16");
    y.setData(res2, "16");
    string res = (x + y).toString("16");

    x.setData(res, "16");
    y.setData(stack, "16");

    return (x % y).toString("16");
}

```

Lưu ý rằng, class QInt chỉ đơn giản là class truyền vào chuỗi và hệ của nó rồi thực hiện các phép tính. Ở đây, gần như chỉ sử dụng trên hệ dec và hex.

Hàm getBlock có nhiệm vụ từ 2 giá trị seed1 và seed2 thực hiện tính toán để trả về một giá trị ngẫu nhiên và seed1, seed2 cũng sẽ bị thay đổi.

Trở lại với hàm phát sinh mê cung generate, ta có thể mô tả bằng đoạn code C++ sau:

```

void generate()
{
    qint x(size, "16");
    string square = (x*x).toString("16");//bình phương

    int square_int = atoi((x*x).toString("10").c_str());
    char *temp = new char[square_int];
    //133311
    int i = 0;
    qint y;
    string eax = "";
    while (i < square_int)
    {
        eax = getBlock("4");
        y.setData(eax, "16");

        int test = atoi((y & y).toString("10").c_str());

        if (test != 0)
        {
            temp[i++] = '.';
        }
        else
        {
            temp[i++] = '#';
        }
    }

    string x_start, y_start, x_finish, y_finish;
    x_start = getBlock(size);
    y_start = getBlock(size);
    x_finish = getBlock(size);
    while (x_start == x_finish)
    {
        x_finish = getBlock(size);
    }
    y_finish = getBlock(size);
    while (y_start == y_finish)
    {
        y_finish = getBlock(size);
    }
    x.setData(x_start, "16");
    y.setData(size, "16");
    string res = (x*y).toString("16");
    x.setData(res, "16");
    y.setData(y_start, "16");
    //res = (x + y).toString("16");
    int pos = atoi((x + y).toString("10").c_str());
    temp[pos] = 's';
}

```

```

x.setData(x_finish, "16");
y.setData(size, "16");
res = (x*y).toString("16");
x.setData(res, "16");
y.setData(y_finish, "16");
//res = (x + y).toString("16");
pos = atoi((x + y).toString("10").c_str());
temp[pos] = 'f';

i = 0; int j = 0;
for (int k = 0; k < square_int; k++)
{
    if (k % size_int == 0 && k != 0)
    {
        j = 0;
        i++;
    }
    if (temp[k] == 's') { this->x_begin = i; this->y_begin = j; }
    if (temp[k] == 'f') { this->x_end = i; this->y_end = j; }
    maze[i][j++] = temp[k];
}
delete[] temp;
temp = NULL;
}

```

Hàm generate cũng là từ các giá trị của username, seed1, seed2 và hàm getBlock để phát sinh các giá trị sau cho có thể đưa ra một mê cung. Sau khi chạy thử chương trình với username là abcde. **Ta có được vài thông số quan trọng:**

00408C9C seed1

00408CA0 seed2

00408CA4 size: kích thước của mê cung (ma trận vuông)

00408CA8 maze: lưu địa chỉ của mê cung, với mê cung được tạo thành từ các ký tự '#' là đường đi bị khóa và '.' là đường đi trống

00408CAC x_start, 00408CB0 y_start: vị trí bắt đầu 's'

00408CB4 x_finish, 00408CB8 y_finish: vị trí kết thúc 'f'

Sau đây là hình ảnh của mê cung với username là abcde (với đường đi trông được thay bằng ‘.’ cho dễ quan sát).



-Xét quá trình xử lý ở serial:

Sau đó khi nhập serial, chương trình sẽ kiểm tra serial với "unsolvable" và nếu nó khớp, sẽ gọi 0040150B |. E8 2CFDFFFF |CALL 2_1.0040123C để giải mê cung. Nếu thất bại, nó đồng ý và ra "Correct. Your name is so ugly, there's no serial for it.". Ta không quan tâm nhiều ở quá trình xử lý này.

00401535 |. E8 23FCFFFF |CALL 2_1.0040115D ;
\2_1.0040115D

Đây là hàm kiểm tra serial có giải được mê cung đã phát sinh từ username hay không?

00401150	55	PUSH EBP
0040115E	. 89E5	MOV EBP,ESP
00401160	. 50	PUSH EAX
00401161	. 53	PUSH EBX
00401162	. 56	PUSH ESI
00401163	. 8B35 AC8C4000	MOV ESI,DWORD PTR [408CAC]
00401169	. 8B0D B08C4000	MOV ECX,DWORD PTR [408CB0]
0040116F	. 8B5D 08	MOV EBX,DWORD PTR [EBP+8]
00401172	> 0FBEE03	MOVSX EAX,BYTE PTR [EBX]
00401175	. 8945 FC	MOV DWORD PTR [EBP-4],EAX
00401178	. 83F8 64	CMP EAX,64
0040117B	. 74 41	JE SHORT 2_1.004011BE
0040117D	. 7F 0F	JG SHORT 2_1.0040118E
0040117F	. 837D FC 00	CMP DWORD PTR [EBP-4],0
00401183	. 0F84 68000000	JE 2_1.004011F1
00401189	. E9 81000000	JMP 2_1.0040120F
0040118E	> 8B45 FC	MOV EAX,DWORD PTR [EBP-4]
00401191	. 83F8 6C	CMP EAX,6C
00401194	. 74 3C	JE SHORT 2_1.004011D2
00401196	. 0F8C 73000000	JL 2_1.0040120F
0040119C	. 8B45 FC	MOV EAX,DWORD PTR [EBP-4]
0040119F	. 83F8 72	CMP EAX,72
004011A2	. 74 3C	JE SHORT 2_1.004011E0
004011A4	. 83F8 75	CMP EAX,75
004011A7	. 0F85 62000000	JNZ 2_1.0040120F
004011AD	. 85C9	TEST ECX,ECX
004011AF	. 75 07	JNZ SHORT 2_1.004011B8
004011B1	. 31C0	XOR EAX,EAX
004011B3	. E9 7E000000	JMP 2_1.00401236
004011B8	> 49	DEC ECX
004011B9	. E9 55000000	JMP 2_1.00401213
004011BE	> A1 A48C4000	MOV EAX,DWORD PTR [408CA4]
004011C3	. 48	DEC EAX
004011C4	. 39C1	CMP ECX,EAX
004011C6	. 75 07	JNZ SHORT 2_1.004011CF
004011C8	. 31C0	XOR EAX,EAX
004011CA	. E9 67000000	JMP 2_1.00401236
004011CF	> 41	INC ECX
004011D0	. EB 41	JMP SHORT 2_1.00401213
004011D2	> 85F6	TEST ESI,ESI
004011D4	. 75 07	JNZ SHORT 2_1.004011D0
004011D6	. 31C0	XOR EAX,EAX
004011D8	. E9 59000000	JMP 2_1.00401236
004011DD	> 4E	DEC ESI
004011DE	. EB 33	JMP SHORT 2_1.00401213
004011E0	> A1 A48C4000	MOV EAX,DWORD PTR [408CA4]

004011E5	. 48	DEC EAX
004011E6	. 39C6	CMP ESI,EAX
004011E8	.> 75 04	JNZ SHORT 2_1.004011EE
004011EA	. 31C0	XOR EAX,EAX
004011EC	.> EB 48	JMP SHORT 2_1.00401236
004011EE	.> 46	INC ESI
004011EF	.> EB 22	JMP SHORT 2_1.00401213
004011F1	.> A1 A48C4000	MOV EAX,DWORD PTR [408CA4]
004011F6	. 89F2	MOV EDX,ESI
004011F8	. 0FAFD0	IMUL EDX,EAX
004011FB	. 8D0411	LEA EAX,DWORD PTR [ECX+EDX]
004011FE	. 8B15 A88C4000	MOV EDX,DWORD PTR [408CA8]
00401204	. 803C02 66	CMP BYTE PTR [EDX+EAX],66
00401208	.> 75 05	JNZ SHORT 2_1.0040120F
0040120A	. 31C0	XOR EAX,EAX
0040120C	. 40	INC EAX
0040120D	.> EB 27	JMP SHORT 2_1.00401236
0040120F	.> 31C0	XOR EAX,EAX
00401211	.> EB 23	JMP SHORT 2_1.00401236
00401213	.> A1 A48C4000	MOV EAX,DWORD PTR [408CA4]
00401218	. 89F2	MOV EDX,ESI
0040121A	. 0FAFD0	IMUL EDX,EAX
0040121D	. 8D0411	LEA EAX,DWORD PTR [ECX+EDX]
00401220	. 8B15 A88C4000	MOV EDX,DWORD PTR [408CA8]
00401226	. 803C02 23	CMP BYTE PTR [EDX+EAX],23
0040122A	.> 75 04	JNZ SHORT 2_1.00401230
0040122C	. 31C0	XOR EAX,EAX
0040122E	.> EB 06	JMP SHORT 2_1.00401236
00401230	.> 43	INC EBX
00401231	.> E9 3CFFFFFF	JMP 2_1.00401172
00401236	.> 5E	POP ESI
00401237	. 5B	POP EBX
00401238	. 89EC	MOV ESP,EBP
0040123A	. 5D	POP EBP
0040123B	. C3	RET

Ta có thể giải thích từng dòng như sau (các địa chỉ, thanh ghi sẽ được gọi bằng tên cho dễ hiểu):

0040115D push ebp

0040115E mov ebp, esp

00401160 push eax

00401161 push curch

00401162 push x

00401163 mov x, x_start

00401169 mov y, y_start

0040116F mov curch, [ebp+serial] (curch chứa giá trị của serial)

00401172

00401172 test_loop: một vòng lặp cho đến khi nào xét hết serial

00401172 movsx eax, byte ptr [curch] (lấy ký tự trở đầu của serial)


```

00401175      mov    [ebp+curchar], eax
00401178      cmp    eax, 'd' (nếu là 'd')
0040117B      jz     short test_DOWN
0040117D      jg     short other_than_d (nếu không phải 'd')
0040117F      cmp    [ebp+curchar], 0 (nếu xét hết serial)
00401183      jz     FINISH
00401189      jmp    bad

0040118E ; -----
0040118E
0040118E other_than_d:
0040118E      mov    eax, [ebp+curchar]
00401191      cmp    eax, 'l' (nếu là 'l')
00401194      jz     short test_LEFT
00401196      jl     bad
0040119C      mov    eax, [ebp+curchar]
0040119F      cmp    eax, 'r' (nếu là 'r')
004011A2      jz     short test_RIGHT
004011A4      cmp    eax, 'u' (nếu là 'u')
004011A7      jnz    bad
004011AD test_UP: kiểm tra xem y==0 không? Nếu có thì exit
004011AD      test   y, y
004011AF      jnz    short UP
004011B1 bad
004011B1      xor    eax, eax
004011B3      jmp    exit

004011B8 ; -----
004011B8

```

004011B8 UP:

004011B8 dec y (y--)

004011B9 jmp test_WALL

004011BE ; -----

004011BE

004011BE test_DOWN: kiểm tra xem y==size-1 không? Nếu có thì exit

004011BE mov eax, side_len

004011C3 dec eax

004011C4 cmp y, eax

004011C6 jnz short DOWN

004011C8

004011C8 xor eax, eax

004011CA jmp exit

004011CF ; -----

004011CF

004011CF DOWN:

004011CF inc y (y++)

004011D0 jmp short test_WALL

004011D2 ; -----

004011D2

004011D2 test_LEFT: kiểm tra xem x==0 không? Nếu có thì exit

004011D2 test x, x

004011D4 jnz short LEFT

004011D6

004011D6 xor eax, eax

004011D8 jmp exit

004011DD ; -----

004011DD

004011DD LEFT:

004011DD dec x (x--)

004011DE jmp short test_WALL

004011E0 ; -----

004011E0

004011E0 test_RIGHT: kiểm tra xem x==size-1 không? Nếu có thì exit

004011E0 mov eax, side_len

004011E5 dec eax

004011E6 cmp x, eax

004011E8 jnz short RIGHT

004011EA

004011EA xor eax, eax

004011EC jmp short exit

004011EE ; -----

004011EE

004011EE RIGHT:

004011EE inc x (x++)

004011EF jmp short test_WALL

004011F1 ; -----

004011F1

004011F1 FINISH: kiểm tra maze[x][y]=='f' không? Nếu không thì nhảy tới bad

004011F1 mov eax, side_len

004011F6 mov edx, x

004011F8 imul edx, eax

004011FB lea eax, [y+edx]

```

004011FE      mov     edx, maze
00401204      cmp     byte ptr [edx+eax], 'f'
00401208      jnz     short bad
0040120A
0040120A      xor     eax, eax
0040120C      inc     eax
0040120D      jmp     short exit
0040120F ; -----
0040120F
0040120F bad:
0040120F
0040120F      xor     eax, eax
00401211      jmp     short exit
00401213 ; -----
00401213
00401213 test_WALL: kiểm tra xem maze[x][y]=='#' không? Nếu đúng
tường thì exit
00401213
00401213      mov     eax, side_len
00401218      mov     edx, x
0040121A      imul    edx, eax
0040121D      lea     eax, [y+edx]
00401220      mov     edx, maze
00401226      cmp     byte ptr [edx+eax], '#'
0040122A      jnz     short continue
0040122C
0040122C      xor     eax, eax

```

```

0040122E          jmp     short exit
00401230 ; -----
00401230
00401230 continue:
00401230          inc     curch (curch++)
00401231          jmp     test_loop
00401236 ; -----
00401236
00401236 exit:
00401236
00401236          pop     x
00401237          pop     curch
00401238          mov     esp, ebp
0040123A          pop     ebp
0040123B          retn
0040123B test_path     endp

```

Có thể hiểu hàm này có nhiệm vụ hướng dẫn đường đi từ vị trí bắt đầu ‘s’ (x_start,y_start) tới vị trí kết thúc ‘f’ (x_finish,y_finish) để giải mê cung bằng cách ký tự điều hướng: u (lên), l (trái), d (xuống), r (phải). Vì vậy, nếu serial có chứa các ký tự khác 4 ký tự trên thì hàm sẽ thoát và kết quả sẽ là bad boy. Ngược lại, ta sẽ phải di chuyển sau cho hợp lý để không đụng tường, để không qua khỏi phạm vi mê cung để đến được đích.

***Bài 2.2 này có keygen**, sau đây là chương trình keygen để giải mê cung được phát sinh từ username viết bằng C++:

```
bool solveMaze(int x, int y)
{
    if (x < 0 || x > size_int - 1 || y < 0 || y > size_int - 1) return false;
    if (maze[x][y] == 'f') return true;
    if (maze[x][y] != '.' && maze[x][y] != 's') return false;
    maze[x][y] = '+';

    if (solveMaze(x - 1, y) == true)
    {
        key += 'l';
        return true;
    }
    if (solveMaze(x + 1, y) == true)
    {
        key += 'r';
        return true;
    }
    if (solveMaze(x, y + 1) == true)
    {
        key += 'd';
        return true;
    }
    if (solveMaze(x, y - 1) == true)
    {
        key += 'u';
        return true;
    }
    maze[x][y] = ' ';
    return false;
}
```

```
string getKey()
{
    if (solveMaze(x_begin, y_begin))
    {
        string tmp = "";
        for (int i = key.length() - 1; i >= 0; i--)
        {
            tmp += key[i];
        }
        return tmp;
    }
    else
    {
        return "unsolvable";
    }
}
```

Từ mê cung đã phát sinh, chương trình keygen sử dụng kỹ thuật đệ quy quay lui để dò tìm đường đi đến đích, trong lúc đệ quy, biến key sẽ được cập nhật liên tục. Cuối cùng, chỉ việc đảo key lại là ta đã có được serial chuẩn để có good boy.

Một ví dụ về keygen với username là abcde

Username: abcde

Serial:

[illegible]

c. Câu 2.2.exe:

-Đầu tiên, ta xét lệnh gọi hàm:

0040A290 . E8 DBAFFFFF CALL 2_2.00405270

Đi vào trong hàm này lại tiếp tục xét lệnh:

004052AA |. E8 3DE7FFFF CALL 2_2.004039EC

Tiếp tục, trong hàm ta xét lệnh:

00403A26 |. E8 61FFFFFF CALL 2 2.0040398C

Trong hàm, ta đặt breakpoint tại: 004039C2 . FFD0 CALL EAX

Và ấn F9 10 lần, sau đó đi vào trong hàm.

Ta thấy, từ **00408EE0** đến **00408F14** thực hiện vòng lặp:

00408EE0	\$ 56	PUSH ESI
00408EE1	. 33F6	XOR ESI,ESI
00408EE3	. B8 ACC74000	MOV EAX,2_2.0040C7AC
00408EE8	> 8930	MOV DWORD PTR [EAX],ESI
00408EEA	. BA 08000000	MOV EDX,8
00408EEF	> 8B08	MOV ECX,DWORD PTR [EAX]
00408EF1	. F6C1 01	TEST CL,1
00408EF4	~ 74 0C	JE SHORT 2_2.00408F02
00408EF6	. D1E9	SHR ECX,1
00408EF8	. 81F1 2083B8E1	XOR ECX,EDB88320
00408EFE	. 8908	MOV DWORD PTR [EAX],ECX
00408F00	~ EB 02	JMP SHORT 2_2.00408F04
00408F02	> D128	SHR DWORD PTR [EAX],1
00408F04	> 4A	DEC EDX
00408F05	^ 75 E8	JNZ SHORT 2_2.00408EEF
00408F07	. 46	INC ESI
00408F08	. 83C0 04	ADD EAX,4
00408F0B	. 81FE 00010001	CMP ESI,100
00408F11	^ 75 05	JNZ SHORT 2_2.00408EE8
00408F13	. 5E	POP ESI
00408F14	. C3	RET

Đây là nơi để thực hiện việc tạo một bảng giá trị để sử dụng về sau.

Ta mô tả đoạn lệnh bằng đoạn code C++ sau:

```
void createTable(vector<unsigned int> &table)
{
    unsigned int v = 0;
    for (int i = 0; i < 0x100; i++)
    {
        v = i;
        for (int j = 0; j < 8; j++)
        {
            if (v % 2 != 0)
            {
                v = (v >> 1) ^ 0xEDB88320;
            }
            else
            {
                v = v >> 1;
            }
        }
        table.push_back(v);
    }
}
```

-Tiếp đến: 0040A2F8 . E8 BF88FFFF CALL 2_2.00402BBC

Thực hiện việc nhập chuỗi serial.

-Tại: 0040A316 . 83F8 0C CMP EAX,0C

Sẽ thực hiện việc kiểm tra độ dài chuỗi serial. Nếu độ dài khác 12 thì báo “Serial must be 12 chars long!”. Ngược lại, tiếp tục chương trình.

-Ta tiến hành công việc xử lý đối với 4 ký tự đầu của serial:

+Vào bên trong câu lệnh gọi hàm: 0040A378 . E8 2BFCFFFF CALL 2_2.00409FA8

Ta xét câu lệnh và đi vào bên trong nó: **0040A04E |. E8 05ECFFFF CALL 2_2.00408C58**

00408C58	\$. 53	PUSH EBX
00408C59	. 56	PUSH ESI
00408C5A	. 57	PUSH EDI
00408C5B	. 51	PUSH ECX
00408C5C	. 890424	MOV DWORD PTR [ESP],EAX
00408C5F	. 8B1C24	MOV EBX,DWORD PTR [ESP]
00408C62	. 8B1B	MOV EBX,DWORD PTR [EBX]
00408C64	. 8B0424	MOV EAX,DWORD PTR [ESP]
00408C67	. 8B40 04	MOV EAX,DWORD PTR [EAX+4]
00408C6A	. 8B35 9CB2400	MOV ESI,DWORD PTR [40B29C]
00408C70	. 0FAFF1	IMUL ESI,ECX
00408C73	. 8BCE	MOV ECX,ESI
00408C75	. 85C9	TEST ECX,ECX
00408C77	√ 74 45	JE SHORT 2_2.00408CBE
00408C79	> 8BF3	MOV ESI,EBX
00408C7B	. C1E6 04	SHL ESI,4
00408C7E	. 8BF8	MOV EDI,EBX
00408C80	. C1EF 05	SHR EDI,5
00408C83	. 33F7	XOR ESI,EDI
00408C85	. 03F3	ADD ESI,EBX
00408C87	. 8BF9	MOV EDI,ECX
00408C89	. C1EF 0B	SHR EDI,0B
00408C8C	. 83E7 03	AND EDI,3
00408C8F	. 8B3CBA	MOV EDI,DWORD PTR [EDX+EDI*4]
00408C92	. 03F9	ADD EDI,ECX
00408C94	. 33F7	XOR ESI,EDI
00408C96	. 2BC6	SUB EAX,ESI
00408C98	. 2B0D 9CB2400	SUB ECX,DWORD PTR [40B29C]
00408C9E	. 8BF0	MOV ESI,EAX
00408CA0	. C1E6 04	SHL ESI,4
00408CA3	. 8BF8	MOV EDI,EAX
00408CA5	. C1EF 05	SHR EDI,5
00408CA8	. 33F7	XOR ESI,EDI
00408CAA	. 03F0	ADD ESI,EAX
00408CAC	. 8BF9	MOV EDI,ECX
00408CAE	. 83E7 03	AND EDI,3
00408CB1	. 8B3CBA	MOV EDI,DWORD PTR [EDX+EDI*4]
00408CB4	. 03F9	ADD EDI,ECX
00408CB6	. 33F7	XOR ESI,EDI
00408CB8	. 2BDE	SUB EBX,ESI
00408CBA	. 85C9	TEST ECX,ECX
00408CBC	. ^ 75 BB	JNZ SHORT 2_2.00408C79
00408CBE	> 8B1424	MOV EDX,DWORD PTR [ESP]
00408CC1	. 891A	MOV DWORD PTR [EDX],EBX
00408CC3	. 8B1424	MOV EDX,DWORD PTR [ESP]
00408CC6	. 8942 04	MOV DWORD PTR [EDX+4],EAX
00408CC9	. 5A	POP EDX
00408CCA	. 5F	POP EDI
00408CCB	. 5E	POP ESI
00408CCC	. 5B	POP EBX
00408CCD	. C3	RET

Đây là hàm có nhiệm vụ dùng 4 bytes ký tự đầu của serial và thực hiện biến đổi để ra 8 bytes và lưu nó.

Ta thực hiện mô tả lại bằng đoạn code C++ sau:

```

} unsigned int getInt(string s)
{
    unsigned int res = 0;
    for (int i = s.length() - 1; i >= 0; i--)
    {
        res = res << 8;
        res += (s[i]);
    }
    return res;
}

} unsigned char* convert4bytesTo8bytes(unsigned int edx)
{
    unsigned int eax = 0xd1fc1e8f, ecx = 0xc6ef3720, ebx = 0x3beabc9a, bc = 0x9e3779b9;
    unsigned int esi, edi;
    while (ecx != 0)
    {
        esi = ebx;
        esi = esi << 4;
        edi = ebx;
        edi = edi >> 5;
        esi = esi xor edi;
        esi = esi + ebx;
        edi = ecx;
        edi = edi >> 11;
        edi = edi & 3;
        if (edi) edi = 0;
        else edi = edx;
        edi = edi + ecx;
        esi = esi xor edi;
        eax = eax - esi;
        ecx = ecx - bc;
        esi = eax;
        esi = esi << 4;
        edi = eax;
        edi = edi >> 5;
    }
}

```

```

        esi = esi xor edi;
        esi = esi + eax;
        edi = ecx;
        edi = edi & 3;
        if (edi) edi = 0;
        else edi = edx;
        edi = edi + ecx;
        esi = esi xor edi;
        ebx = ebx - esi;
    }

    unsigned char *res = new unsigned char[9];

    for (int i = 0; i < 4; i++)
    {
        res[i] = ebx & 0xff;
        ebx = ebx >> 8;
        res[i + 4] = eax & 0xff;
        eax = eax >> 8;
    }
    res[8] = '\0';
    return res;
}

```

Có thể hiểu đoạn code C++ đơn giản là ta truyền vào 4 ký tự đầu của serial vào hàm getInt để nhận được con số và truyền số đó vào edx của hàm convert4bytesTo8bytes và trả về kết quả dạng chuỗi ký tự.

+Xét tại: **0040A3A2 . E8 71EBFFFF CALL 2_2.00408F18**

00408F18	53	PUSH EBX
00408F19	56	PUSH ESI
00408F1A	8BF0	MOV ESI,EAX
00408F1C	83CB FF	OR EBX,FFFFFFFF
00408F1F	8BC6	MOV EAX,ESI
00408F21	E8 DAFFFF	CALL 2_2.00403F00
00408F26	85C0	TEST EAX,EAX
00408F28	7E 21	JLE SHORT 2_2.00408F4B
00408F2A	BA 01000000	MOV EDX,1
00408F2F	33C9	XOR ECX,ECX
00408F31	8A4C16 FF	MOV CL, BYTE PTR [ESI+EDX-1]
00408F35	33CB	XOR ECX,EBX
00408F37	81E1 FF000000	AND ECX,0FF
00408F3D	C1EB 08	SHR EBX,8
00408F40	331C8D ACC741	XOR EBX,DWORD PTR [ECX*4+40C7AC]
00408F47	42	INC EDX
00408F48	48	DEC EAX
00408F49	75 E4	JNZ SHORT 2_2.00408F2F
00408F4B	F7D3	NOT EBX
00408F4D	8BC3	MOV EAX,EBX
00408F4F	5E	POP ESI
00408F50	5B	POP EBX
00408F51	C3	RET

Đây là hàm băm crc32, có nhiệm vụ biến đổi 8 bytes đầu vào về 4 bytes.
Sau đây là đoạn code C++ mô tả hàm băm crc32:

```

unsigned int crc32(unsigned char ESI[9], vector<unsigned int> table)
{
    unsigned int ECX, EDX;
    unsigned int EBX = 0xffffffff;
    for (EDX = 0; EDX < 8; EDX++)
    {
        ECX = 0;
        ECX = ESI[EDX];
        ECX = ECX ^ EBX;
        ECX = ECX & 0xff;
        EBX = EBX >> 8;
        EBX = EBX ^ table[ECX];
    }
    EBX = ~EBX;
    return EBX;
}

```

Có thể hiểu, hàm crc32 nhận vào chuỗi ký tự và sử dụng bảng giá trị đã khởi tạo lúc đầu để tiến hành băm.

+Xét tại: **0040A3A7 . 3D 80B456B4 CMP EAX,B456B480**

Kết quả băm trả về gán vào EAX, sau đó tiến hành so sánh với 0xB456B480. Nếu bằng thì tiếp tục thực hiện các công việc tiếp sau, ngược lại, thông báo sai “Invalid Serial”.

⇒ Từ việc tiến hành khảo sát và mô tả lại công việc xử lý đối với 4 ký tự đầu của serial. Thực hiện thuật toán brute force để vét cạn các trường hợp có thể xảy ra đối với 4 ký tự đầu này. Ta có thể đưa ra kết luận là 4 ký tự đầu của serial đúng là **PSk=**

-Ta tiến hành công việc xử lý đối với 8 ký tự cuối của serial:

+Xét tại: 0040A3D0 . E8 07BDFFFF CALL 2_2.004060DC

004060DC	53	PUSH EBX
004060DD	. 56	PUSH ESI
004060DE	. 57	PUSH EDI
004060DF	. 8BFA	MOV EDI,EDX
004060E1	. 8BF0	MOV ESI,EAX
004060E3	. 8BC6	MOV EAX,ESI
004060E5	. E8 16DEFFFF	CALL 2_2.00403F00
004060EA	. 8BD8	MOV EBX,EAX
004060EC	. 8BC7	MOV EAX,EDI
004060EE	. 8BD3	MOV EDX,EBX
004060F0	. E8 FDFDFFFF	CALL 2_2.004040F4
004060F5	. 8BD6	MOV EDX,ESI
004060F7	. 8B37	MOV ESI,DWORD PTR [EDI]
004060F9	. 85DB	TEST EBX,EBX
004060FB	.v 74 15	JE SHORT 2_2.00406112
004060FD	> 8A02	MOV AL,BYTE PTR [EDX]
004060FF	. 3C 61	CMP AL,61
00406101	.v 72 06	JB SHORT 2_2.00406109
00406103	. 3C 7A	CMP AL,7A
00406105	.v 77 02	JA SHORT 2_2.00406109
00406107	. 2C 20	SUB AL,20
00406109	> 8806	MOV BYTE PTR [ESI],AL
0040610B	. 42	INC EDX
0040610C	. 46	INC ESI
0040610D	. 4B	DEC EBX
0040610E	. 85DB	TEST EBX,EBX
00406110	.^ 75 EB	JNZ SHORT 2_2.004060FD
00406112	> 5F	POP EDI
00406113	. 5E	POP ESI
00406114	. 5B	POP EBX
00406115	. C3	RET

Đây là hàm có chức năng **in hoa** chuỗi ký tự. Cụ thể ở đây là in hoa 8 ký tự cuối của serial để xử lý về sau.

+Xét tại: 0040A3E7 . E8 2CFBFFFF CALL 2_2.00409F18

00409F18	55	PUSH EBP
00409F19	8BEC	MOV EBP,ESP
00409F1B	51	PUSH ECX
00409F1C	53	PUSH EBX
00409F1D	8945 FC	MOV DWORD PTR [EBP-4],EAX
00409F20	8B45 FC	MOV EAX,DWORD PTR [EBP-4]
00409F23	E8 24A1FFFF	CALL 2_2.0040404C
00409F28	33C0	XOR EAX,EAX
00409F2A	55	PUSH EBP
00409F2B	68 8A9F4000	PUSH 2_2.00409F8A
00409F30	64:FF30	PUSH DWORD PTR FS:[EAX]
00409F33	64:8920	MOV DWORD PTR FS:[EAX],ESP
00409F36	33DB	XOR EBX,EBX
00409F38	8B45 FC	MOV EAX,DWORD PTR [EBP-4]
00409F3B	E8 C09FFFFF	CALL 2_2.00403F00
00409F40	85C0	TEST EAX,EAX
00409F42	7E 21	JLE SHORT 2_2.00409F65
00409F44	BA 01000000	MOV EDI,1
00409F49	8B4D FC	MOV ECX,DWORD PTR [EBP-4]
00409F4C	8A4C11 FF	MOV CL,BYTE PTR [ECX+EDI-1]
00409F50	80C1 D0	ADD CL,0D0
00409F53	80E9 0A	SUB CL,0A
00409F56	72 08	JB SHORT 2_2.00409F60
00409F58	80C1 F9	ADD CL,0F9
00409F5B	80E9 06	SUB CL,6
00409F5E	73 01	JNB SHORT 2_2.00409F61
00409F60	43	INC EBX
00409F61	42	INC EDI
00409F62	48	DEC EAX
00409F63	75 E4	JNZ SHORT 2_2.00409F49
00409F65	8B45 FC	MOV EAX,DWORD PTR [EBP-4]
00409F68	E8 939FFFFF	CALL 2_2.00403F00
00409F6D	3BD8	CMPL EBX,EAX
00409F6F	0F94C0	SETL AL
00409F72	8BD8	MOV EBX,EAX
00409F74	33C0	XOR EAX,EAX
00409F76	5A	POP EDI
00409F77	59	POP ECX
00409F78	59	POP ECX
00409F79	64:8910	MOV DWORD PTR FS:[EAX],EDI
00409F7C	68 919F4000	PUSH 2_2.00409F91
00409F81	8D45 FC	LEA EAX,DWORD PTR [EBP-4]
00409F84	E8 F79CFFFF	CALL 2_2.00403C80
00409F89	C3	RET

Sau khi thực hiện in hoa 8 ký tự cuối của serial, thì đến lượt hàm này kiểm tra sự hợp lệ của của 8 ký tự cuối này. Nếu nó ngoài khoảng ký tự từ ‘0’ đến ‘9’, ‘A’ đến ‘F’ thì báo lỗi “Serial have a invalid format”. Ngược lại, tiếp tục thực hiện các công đoạn sau.

Ta có thể hiểu hàm này một cách đơn giản thông qua đoạn code C++ sau:

```

bool checkKey(string s)
{
    for (int i = 0; i < s.length(); i++)
    {
        if (s[i]<'0')
        {
            return false;
        }
        else if (s[i] > '9')
        {
            if (s[i]<'A' || s[i]>'F')
            {
                return false;
            }
        }
    }
    return true;
}

```

+Xét tại: **0040A433 . E8 60FBFFFF CALL 2_2.00409F98**

00409F98	\$ 3E:8130 DEC0	XOR DWORD PTR [EAX],1337C0DE
00409F9F	. 50	PUSH EAX
00409FA0	? C3	RET
00409FA1	. 00C3	ADD BL,AL
00409FA3	. 90	NOP
00409FA4	\$ 0FC8	BSWAP EAX
00409FA6	. C3	RET

Đây là hàm kiểm tra 2 ký tự cuối của serial.

Ta giả sử lúc này 8 ký tự cuối của serial được lưu vào EAX là 13579BDF, thì lúc này hàm sẽ lấy EAX xor với 0x1337C0DE. Nhưng có một lưu ý là tại: 00409FA0 |? C3 RET thì câu lệnh này chỉ trả về nếu byte cuối cùng của kết quả là C3. Như vậy, với ví dụ này thì kết quả là 605B01. Và 01!=C3 nên không thỏa.

Vậy để xor với 0x1337C0DE thì 2 ký tự cuối của serial phải là 1D để kết quả có byte cuối là C3.

+Xét tại: **0040A453 . E8 C0EAF0FF CALL 2_2.00408F18**

00408F18	53	PUSH EBX
00408F19	56	PUSH ESI
00408F1A	8BF0	MOV ESI,EAX
00408F1C	83CB FF	OR EBX,FFFFFFFF
00408F1F	8BC6	MOV EAX,ESI
00408F21	E8 DAF0FF	CALL 2_2.00403F00
00408F26	85C0	TEST EAX,EAX
00408F28	7E 21	JLE SHORT 2_2.00408F4B
00408F2A	BA 01000000	MOV EDX,1
00408F2F	33C9	XOR ECX,ECX
00408F31	8A4C16 FF	MOV CL,BYTE PTR [ESI+EDX-1]
00408F35	33CB	XOR ECX,EBX
00408F37	81E1 FF000000	AND ECX,0FF
00408F3D	C1EB 08	SHR EBX,8
00408F40	331C8D ACC74	XOR EBX,DWORD PTR [ECX*4+40C7AC]
00408F47	42	INC EDX
00408F48	48	DEC EAX
00408F49	75 E4	JNZ SHORT 2_2.00408F2F
00408F4B	F7D3	NOT EBX
00408F4D	8BC3	MOV EAX,EBX
00408F4F	5E	POP ESI
00408F50	5B	POP EBX
00408F51	C3	RET

Ta nhận thấy đây là hàm băm crc32. Lúc này hàm nhận vào 8 bytes ký tự cuối của serial và trả về 4 bytes ký tự.

+Xét tại: **0040A458 . 3D 0CDB67E3 CMP EAX,E367DB0C**

Kết quả băm đem so sánh với 0xE367DB0C. Nếu không trùng thì báo lỗi “Invalid Serial”, ngược lại, thông báo kết quả đúng “Nice work... now write a solution.!”.

⇒ Từ việc tiến hành khảo sát và mô tả lại công việc xử lý đối với 8 ký tự cuối của serial. Ta biết được 2 ký tự cuối cùng nhất là 1D hoặc 1d. Thực hiện thuật toán brute force để vét cạn các trường hợp có thể xảy ra đối với 6 ký tự còn lại. Ta có thể đưa ra kết luận là 8 ký tự cuối của serial đúng có thể là: **4c4f4c1d, 4c4f4c1D, 4c4f4C1d, 4c4f4C1D, 4c4F4c1d, 4c4F4c1D, 4c4F4C1d, 4c4F4C1D, 4C4f4c1d, 4C4f4c1D, 4C4f4C1d, 4C4f4C1D, 4C4F4c1d, 4C4F4c1D, 4C4F4C1d, 4C4F4C1D.**

Vậy bài 2.2 có 16 serial key cứng là:

PSk=4c4f4c1d

PSk=4c4f4c1D

PSk=4c4f4C1d

PSk=4c4f4C1D

PSk=4c4F4c1d

PSk=4c4F4c1D

PSk=4c4F4C1d

PSk=4c4F4C1D

PSk=4C4f4c1d

PSk=4C4f4c1D

PSk=4C4f4C1d

PSk=4C4f4C1D

PSk=4C4F4c1d

PSk=4C4F4c1D

PSk=4C4F4C1d

PSk=4C4F4C1D

d. Câu 3.1.exe:

-Từ **00401092** đến **00401172** là nơi chứa tất cả các hàm và lệnh thực hiện các công việc nhập xuất và xử lý name và serial.

00401092	> 60	PUSHAD	
00401093	. 6A 7F	PUSH 7F	
00401095	. 68 001E4000	PUSH 3_1.00401E00	[Count = 7F (127.)
0040109A	. 68 E8030000	PUSH 3E8	Buffer = 3_1.00401E00
0040109F	. FF35 001F4000	PUSH DWORD PTR [401F00]	ControlID = 3E8 (1000.)
004010A5	. FF15 54204000	CALL DWORD PTR [<&USER32.GetDlgItemTextA]	hWnd = NULL
004010A8	. 85C0	TEST EAX,EAX	GetDlgItemTextA
004010AD	~ 75 1C	JNZ SHORT 3_1.004010CB	
004010AF	. 6A 30	PUSH 30	
004010B1	. 68 00144000	PUSH 3_1.00401400	[Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL
004010B6	. 68 26144000	PUSH 3_1.00401426	Title = "Error"
004010BB	. FF35 001F4000	PUSH DWORD PTR [401F00]	Text = "You forgot to enter a Name..."
004010C1	. FF15 50204000	CALL DWORD PTR [<&USER32.MessageBoxA>]	hOwner = NULL
004010C7	. 61	POPAD	MessageBoxA
004010C8	. 33C0	XOR EAX,EAX	
004010CA	. C3	RET	
004010CB	> 6A 7F	PUSH 7F	
004010CD	. 68 001D4000	PUSH 3_1.00401D00	[Count = 7F (127.)
004010D2	. 68 E9030000	PUSH 3E9	Buffer = 3_1.00401D00
004010D7	. FF35 001F4000	PUSH DWORD PTR [401F00]	ControlID = 3E9 (1001.)
004010DD	. FF15 54204000	CALL DWORD PTR [<&USER32.GetDlgItemTextA]	hWnd = NULL
004010E3	. 85C0	TEST EAX,EAX	GetDlgItemTextA
004010E5	~ 75 1C	JNZ SHORT 3_1.00401103	
004010E7	. 6A 30	PUSH 30	
004010E9	. 68 00144000	PUSH 3_1.00401400	[Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL
004010EE	. 68 44144000	PUSH 3_1.00401444	Title = "Error"
004010F3	. FF35 001F4000	PUSH DWORD PTR [401F00]	Text = "There is something wrong with your Serial..."
004010F9	. FF15 50204000	CALL DWORD PTR [<&USER32.MessageBoxA>]	hOwner = NULL
004010FF	. 61	POPAD	MessageBoxA
00401100	. 33C0	XOR EAX,EAX	
00401102	. C3	RET	
00401103	> E8 A2010000	CALL 3_1.004012AA	
00401108	. 84C0	TEST AL,AL	
0040110A	~ 75 1C	JNZ SHORT 3_1.00401128	
0040110C	. 6A 30	PUSH 30	
0040110E	. 68 00144000	PUSH 3_1.00401400	[Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL
00401113	. 68 64144000	PUSH 3_1.00401464	Title = "Error"
00401118	. FF35 001F4000	PUSH DWORD PTR [401F00]	Text = "There is something wrong with your Serial..."
0040111E	. FF15 50204000	CALL DWORD PTR [<&USER32.MessageBoxA>]	hOwner = NULL
00401124	. 61	POPAD	MessageBoxA
00401125	. 33C0	XOR EAX,EAX	
00401127	. C3	RET	

00401128	> E8 46000000	CALL 3_1.00401173	
0040112D	. E8 69000000	CALL 3_1.0040119B	
00401132	. E8 D5000000	CALL 3_1.0040120C	
00401137	. 84C0	TEST AL,AL	
00401139	.. 75 1C	JNZ SHORT 3_1.00401157	
0040113B	. 6A 40	PUSH 40	
0040113D	. 68 91144000	PUSH 3_1.00401491	
00401142	. 68 9C144000	PUSH 3_1.0040149C	
00401147	. FF35 001F4000	PUSH DWORD PTR [401F00]	
0040114D	. FF15 50204000	CALL DWORD PTR [<&USER32.MessageBoxA>]	Style = MB_OK!MB_ICONASTERISK!MB_APPLMODAL Title = "Too bad..." Text = "Common, you can do better then that! :)" hOwner = NULL MessageBoxA
00401153	. 61	POPAD	
00401154	. 33C0	XOR EAX,EAX	
00401156	. C3	RET	
00401157	> 6A 40	PUSH 40	
00401159	. 68 C4144000	PUSH 3_1.004014C4	
0040115E	. 68 D5144000	PUSH 3_1.004014D5	
00401163	. FF35 001F4000	PUSH DWORD PTR [401F00]	
00401169	. FF15 50204000	CALL DWORD PTR [<&USER32.MessageBoxA>]	Style = MB_OK!MB_ICONASTERISK!MB_APPLMODAL Title = "Congratulations!" Text = "Well done!#Now for the more difficult part, try to keygen it..." hOwner = NULL MessageBoxA
0040116F	. 61	POPAD	
00401170	. 33C0	XOR EAX,EAX	
00401172	. C3	RET	

004010A5 . FF15 54204000 CALL DWORD PTR
[<&USER32.GetDlgItemTextA>; \GetDlgItemTextA

=> Nhập name

004010C1 . FF15 50204000 CALL DWORD PTR
[<&USER32.MessageBoxA>] ; \MessageBoxA

=> Báo lỗi nếu name rỗng

004010DD . FF15 54204000 CALL DWORD PTR
[<&USER32.GetDlgItemTextA>; \GetDlgItemTextA

=> Nhập serial

004010F9 . FF15 50204000 CALL DWORD PTR
[<&USER32.MessageBoxA>] ; \MessageBoxA

=> Báo lỗi nếu serial rỗng

Tại 00401103 > E8 A2010000 CALL 3_1.004012AA

004012AA	> 50	PUSH EAX
004012AB	. 56	PUSH ESI
004012AC	. BE 00174000	MOV ESI,3_1.00401700
004012B1	> 8B06	MOV EAX,DWORD PTR [ESI]
004012B3	. 8326 00	AND DWORD PTR [ESI],0
004012B6	. 83C6 04	ADD ESI,4
004012B9	. 85C0	TEST EAX,EAX
004012BB	. ^ 75 F4	JNZ SHORT 3_1.004012B1
004012BD	. BE 001D4000	MOV ESI,3_1.00401D00
004012C2	> 8A06	MOV AL,BYTE PTR [ESI]
004012C4	. 84C0	TEST AL,AL
004012C6	. ^ 74 18	JE SHORT 3_1.004012E0
004012C8	. 3C 30	CMP AL,30
004012CA	. ^ 72 0C	JB SHORT 3_1.004012D8
004012CC	. 3C 3A	CMP AL,3A
004012CE	. ^ 72 0D	JB SHORT 3_1.004012DD
004012D0	. 3C 41	CMP AL,41
004012D2	. ^ 72 04	JB SHORT 3_1.004012D8
004012D4	. 3C 47	CMP AL,47
004012D6	. ^ 72 05	JB SHORT 3_1.004012DD
004012D8	> 5E	POP ESI
004012D9	. 58	POP EAX
004012DA	. B0 00	MOV AL,0
004012DC	. C3	RET
004012DD	> 46	INC ESI
004012DE	. ^ EB E2	JMP SHORT 3_1.004012C2
004012E0	> 5E	POP ESI
004012E1	. 58	POP EAX
004012E2	. B0 01	MOV AL,1
004012E4	. C3	RET

=> Kiểm tra các ký tự trong serial có phải là một trong các ký tự: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' hay không?

Nếu có ký tự không thuộc các ký tự trên thì báo lỗi:

0040110C	. 6A 30	PUSH 30	[Style = MB_OK!MB_ICONEXCLAMATION!MB_APPLMODAL Title = "Error" Text = "There is something wrong with your Serial..." hOwner = NULL MessageBoxA]
0040110E	. 68 00144000	PUSH 3_1.00401400	
00401113	. 68 64144000	PUSH 3_1.00401464	
00401118	. FF35 001F4000	PUSH DWORD PTR [401F00]	
0040111E	. FF15 50204000	CALL DWORD PTR [<&USER32.MessageBoxA>]	

Còn nếu quá trình nhập name và serial thuận lợi thì ta sẽ có good boy và bad boy là:

-Good boy:

00401157	> 6A 40	PUSH 40	[Style = MB_OK!MB_ICONASTERISK!MB_APPLMODAL Title = "Congratulations!" Text = "Well done! Now for the more difficult part, try to keygen it..." hOwner = NULL MessageBoxA]
00401159	. 68 C4144000	PUSH 3_1.004014C4	
0040115E	. 68 D5144000	PUSH 3_1.004014D5	
00401163	. FF35 001F4000	PUSH DWORD PTR [401F00]	
00401169	. FF15 50204000	CALL DWORD PTR [<&USER32.MessageBoxA>]	

-Bad boy:

0040113B	. 6A 40	PUSH 40	[Style = MB_OK!MB_ICONASTERISK!MB_APPLMODAL Title = "Too bad..." Text = "Common, you can do better then that! :)" hOwner = NULL MessageBoxA]
0040113D	. 68 91144000	PUSH 3_1.00401491	
00401142	. 68 9C144000	PUSH 3_1.0040149C	
00401147	. FF35 001F4000	PUSH DWORD PTR [401F00]	
0040114D	. FF15 50204000	CALL DWORD PTR [<&USER32.MessageBoxA>]	

-Ta xét 3 hàm quan trọng:

00401128	> E8 46000000	CALL 3_1.00401173
0040112D	. E8 69000000	CALL 3_1.0040119B
00401132	. E8 D5000000	CALL 3_1.0040120C

-Đầu tiên, **00401128 > E8 46000000 CALL 3_1.00401173** là hàm tạo ma trận 16x16 và 2 hàng rào chắn

00401173	50	PUSH EAX
00401174	51	PUSH ECX
00401175	57	PUSH EDI
00401176	BF F01A4000	MOV EDI,3_1.00401AF0
0040117B	FC	CLD
0040117C	B9 10000000	MOV ECX,10
00401181	B0 FF	MOV AL,0FF
00401183	F3:AA	REP STOS BYTE PTR ES:[EDI]
00401185	B9 00010000	MOV ECX,100
0040118A	B0 00	MOV AL,0
0040118C	F3:AA	REP STOS BYTE PTR ES:[EDI]
0040118E	B9 10000000	MOV ECX,10
00401193	B0 FF	MOV AL,0FF
00401195	F3:AA	REP STOS BYTE PTR ES:[EDI]
00401197	5F	POP EDI
00401198	59	POP ECX
00401199	58	POP EAX
0040119A	C3	RET

00401183 |. F3:AA REP STOS BYTE PTR ES:[EDI]

⇒ Đây là một dạng như memset trong C++, lưu EAX vào nơi EDI trở đến.

0040117C |. B9 10000000 MOV ECX,10

00401181 |. B0 FF MOV AL,0FF

00401183 |. F3:AA REP STOS BYTE PTR ES:[EDI]

⇒ Cho 16 ô của hàng đầu mang giá trị FF

00401185 |. B9 00010000 MOV ECX,100 ; 256 bytes - 16x16 grid

0040118A |. B0 00 MOV AL,0

0040118C |. F3:AA REP STOS BYTE PTR ES:[EDI]

⇒ Cho liên tục 16 ô của 16 hàng kế tiếp là 00 (16x16=256=100h)

0040118E |. B9 10000000 MOV ECX,10

00401193 |. B0 FF MOV AL,0FF

00401195 |. F3:AA REP STOS BYTE PTR ES:[EDI]

⇒ Cho 16 ô của hàng cuối mang giá trị FF

Ta có dạng ma trận sau khi gọi hàm như sau:

```
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

-Tiếp theo, **0040112D . E8 69000000 CALL 3_1.0040119B** là hàm xử lý name để phát sinh các giá trị cần thiết cho ma trận.

0040119B	> 50	PUSH EAX
0040119C	. 51	PUSH ECX
0040119D	. 52	PUSH EDX
0040119E	. 56	PUSH ESI
0040119F	. 57	PUSH EDI
004011A0	. 33C9	XOR ECX,ECX
004011A2	. BE 001E4000	MOV ESI,3_1.00401E00
004011A7	. BF 001B4000	MOV EDI,3_1.00401B00
004011AC	. 56	PUSH ESI
004011AD	. B2 00	MOV DL,0
004011AF	> 8A06	MOV AL,BYTE PTR [ESI]
004011B1	. 46	INC ESI
004011B2	. 02D0	ADD DL,AL
004011B4	. 84C0	TEST AL,AL
004011B6	. ^ 75 F7	JNZ SHORT 3_1.004011AF
004011B8	. 5E	POP ESI
004011B9	> 0FB606	MOVZX EAX,BYTE PTR [ESI]
004011BC	. 32C2	XOR AL,DL
004011BE	> 2AD0	SUB DL,AL
004011C0	. 800C07 CC	OR BYTE PTR [EDI+EAX],0CC
004011C4	. ^ 75 04	JNZ SHORT 3_1.004011CA
004011C6	. FECA	DEC DL
004011C8	. ^ EB F4	JMP SHORT 3_1.004011BE
004011CA	> 41	INC ECX
004011CB	. 46	INC ESI
004011CC	. 803E 00	CMP BYTE PTR [ESI],0
004011CF	. ^ 75 E8	JNZ SHORT 3_1.004011B9
004011D1	. 890D 041F4000	MOV DWORD PTR [401F04],ECX
004011D7	. 32D0	XOR DL,AL
004011D9	> 2AC2	SUB AL,DL
004011DB	. 803C07 CC	CMP BYTE PTR [EDI+EAX],0CC
004011DF	. ^ 75 04	JNZ SHORT 3_1.004011E5
004011E1	. FECA	DEC DL
004011E3	. ^ EB F4	JMP SHORT 3_1.004011D9
004011E5	> C60407 DD	MOV BYTE PTR [EDI+EAX],0DD
004011E9	. 8AC2	MOV AL,DL
004011EB	> 803C07 CC	CMP BYTE PTR [EDI+EAX],0CC
004011EF	. ^ 74 06	JE SHORT 3_1.004011F7
004011F1	. 803C07 DD	CMP BYTE PTR [EDI+EAX],0DD
004011F5	. ^ 75 04	JNZ SHORT 3_1.004011FB
004011F7	> FEC8	DEC AL
004011F9	. ^ EB F0	JMP SHORT 3_1.004011EB
004011FB	> 8D0407	LEA EAX,DWORD PTR [EDI+EAX]
004011FE	. C600 99	MOV BYTE PTR [EAX],99
00401201	. A3 00174000	MOV DWORD PTR [401700],EAX
00401206	. 5F	POP EDI
00401207	. 5E	POP ESI
00401208	. 5A	POP EDX
00401209	. 59	POP ECX
0040120A	. 58	POP EAX
0040120B	. C3	RET

Ta xét các bước xử lý đáng chú ý của hàm:

004011A2 |. BE 001E4000 MOV ESI,Snake.00401E00 ;

⇒ gán name vào ESI

004011A7 |. BF 001B4000 MOV EDI,Snake.00401B00 ;

⇒ gán ma trận 16x16 vào EDI

004011C0 |. 800C07 CC ||OR BYTE PTR DS:[EDI+EAX],0CC;

⇒ mỗi ký tự trong name được ánh xạ tới một vị trí nào đó trong ma trận và thể hiện bằng “CC”

004011E5 |> C60407 DD MOV BYTE PTR DS:[EDI+EAX],0DD;

⇒ "DD" đại diện cho điểm kết thúc

004011FE |. C600 99 MOV BYTE PTR DS:[EAX],99 ;

⇒ "99" đại diện cho Snake

Thực chất, đây là một dạng mô phỏng lại game Snake.

Dưới đây là code C++ mô tả lại hàm phát sinh ma trận và các giá trị cần thiết của nó:

```
void generate()
{
    vector<string> temp;
    for (int i = 0; i < 256; i++)
    {
        temp.push_back("00");
    }
    string s = textToHex(name);
    qint x("00", "16");
    qint y("00", "16");
    string AL = "", DL = "00";
    for (int i = 0; i < 2 * name.length(); i += 2)
    {
        AL = s.substr(i, 2);
        x.setData(AL, "16");
        y.setData(DL, "16");
        DL = (y + x).toString("16");
        DL = getLastChars(DL, 1);
    }
}
```

```

int pos;
for (int i = 0; i < 2 * name.length(); i += 2)
{
    AL = s.substr(i, 2);
    x.setData(AL, "16");
    y.setData(DL, "16");
    AL = (x ^ y).toString("16");
    AL = getLastChars(AL, 1);

    Label0:
    x.setData(AL, "16");
    y.setData(DL, "16");
    DL = (y - x).toString("16");
    DL = getLastChars(DL, 1);

    x.setData(AL, "16");
    pos = atoi(x.toString("10").c_str());

    if (temp[pos] == "CC")
    {
        y.setData(DL, "16");
        x.setData("01", "16");
        DL = (y - x).toString("16");
        DL = getLastChars(DL, 1);
        goto Label0;
    }

    else temp[pos] = "CC";
}
x.setData(AL, "16");
y.setData(DL, "16");
DL = getLastChars((y^x).toString("16"), 1);

```

Label1:

```

x.setData(AL, "16");
y.setData(DL, "16");
AL = getLastChars((x - y).toString("16"), 1);
x.setData(AL, "16");
pos = atoi(x.toString("10").c_str());

if (temp[pos] == "CC")
{
    y.setData(DL, "16");
    x.setData("01", "16");
    DL = getLastChars((y - x).toString("16"), 1);
    goto Label1;
}

```



```

x.setData(AL, "16");
pos = atoi(x.toString("10").c_str());

temp[pos] = "DD";
AL = DL;

```

Label2:

```

x.setData(AL, "16");
pos = atoi(x.toString("10").c_str());

if (temp[pos] == "CC" || temp[pos] == "DD")
{
    x.setData(AL, "16");
    y.setData("01", "16");
    AL = getLastChars((x - y).toString("16"), 1);
    goto Label2;
}

x.setData(AL, "16");
pos = atoi(x.toString("10").c_str());

temp[pos] = "99";

```

```

int i = 0, j = 0;
for (int k = 0; k < 256; k++)
{
    if (temp[k] == "DD")
    {
        dest.x = i;
        dest.y = j;
        grid[i][j] = 'D';
    }
    if (temp[k] == "CC")
    {
        Point2D food;
        food.x = i;
        food.y = j;
        foods.push_back(food);
        grid[i][j] = 'C';
    }
    if (temp[k] == "99")
    {
        snake.x = i;
        snake.y = j;
        grid[i][j] = '9';
    }
    j++;
}

```

```

    if ((k+1) % 16 == 0)
    {
        j = 0;
        i++;
    }
}

```

Đoạn code đơn giản là sử dụng QInt để mô tả lại quá trình tính toán và phát sinh ma trận. Sau đó lưu vào một ma trận để xử lý về sau. Để việc lưu ma trận đơn giản, “FF” -> ‘F’, “00” -> ‘0’, “CC” -> ‘C’, ”DD” -> ‘D’, “99” -> ‘9’.

Giả sử name nhập vào là: abcde

Ta có ma trận mô phỏng như sau:

```

F F F F F F F F F F F F F F
0 0 0 C 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 D
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 C 0
0 0 0 0 0 C 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 C 0
0 0 0 0 0 0 0 0 0 9 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 C 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
F F F F F F F F F F F F F F

```

-Cuối cùng, **00401132 . E8 D5000000 CALL 3_1.0040120C** là hàm kiểm tra serial. Thực chất serial là một chuỗi các ký tự hướng dẫn đường đi cho snake để có thể đi đến đích cuối cùng.

0040120C	\$ 60	PUSHAD
0040120D	. BE 001D4000	MOV ESI,3_1.00401D00
00401212	. BF 00174000	MOV EDI,3_1.00401700
00401217	> 0FB606	MOVBX EAX, BYTE PTR [ESI]
0040121A	. 84C0	TEST AL,AL
0040121C	.v 74 42	JE SHORT 3_1.00401260
0040121E	. 2C 30	SUB AL,30
00401220	. 3C 0A	CMP AL,0A
00401222	.v 72 02	JB SHORT 3_1.00401226
00401224	. 2C 07	SUB AL,7
00401226	> 8BC8	MOV ECX,EAX
00401228	. 24 03	AND AL,3
0040122A	. C0E9 02	SHR CL,2
0040122D	. BA 10000000	MOV EDX,10
00401232	. 84C0	TEST AL,AL
00401234	.v 74 0B	JE SHORT 3_1.00401241
00401236	. 48	DEC EAX
00401237	.v 74 06	JE SHORT 3_1.0040123F
00401239	. C1EA 04	SHR EDX,4
0040123C	. 48	DEC EAX
0040123D	.v 75 02	JNZ SHORT 3_1.00401241
0040123F	> F7DA	NEG EDX
00401241	> 8B07	MOV EAX,DWORD PTR [EDI]
00401243	. 03C2	ADD EAX,EDX
00401245	. 8A00	MOV AL,BYTE PTR [EAX]
00401247	. 84C0	TEST AL,AL
00401249	.v 74 2F	JE SHORT 3_1.0040127A
0040124B	. 3C 99	CMP AL,99
0040124D	.v 74 11	JE SHORT 3_1.00401260
0040124F	. 3C CC	CMP AL,0CC
00401251	.v 74 15	JE SHORT 3_1.00401268
00401253	. 3C DD	CMP AL,0DD
00401255	.v 75 09	JNZ SHORT 3_1.00401260
00401257	. 833D 041F4000	CMP DWORD PTR [401F04],0
0040125E	.v 74 04	JE SHORT 3_1.00401264
00401260	> 61	POPAD
00401261	. B0 00	MOV AL,0
00401263	. C3	RET
00401264	> 61	POPAD
00401265	. B0 01	MOV AL,1
00401267	. C3	RET
00401268	> FF0D 041F4000	DEC DWORD PTR [401F04]
0040126E	. E8 16000000	CALL 3_1.00401289
00401273	. 8918	MOV DWORD PTR [EAX],EBX
00401275	. C603 99	MOV BYTE PTR [EBX],99
00401278	.v EB 05	JMP SHORT 3_1.0040127F
0040127A	> E8 0A000000	CALL 3_1.00401289
0040127F	> 85C9	TEST ECX,ECX
00401281	.v 74 03	JE SHORT 3_1.00401286
00401283	. 49	DEC ECX
00401284	.^ EB BB	JMP SHORT 3_1.00401241
00401286	> 46	INC ESI
00401287	.^ EB 8E	JMP SHORT 3_1.00401217
00401289	\$ 57	PUSH EDI
0040128A	. 8B07	MOV EAX,DWORD PTR [EDI]
0040128C	. 8BD8	MOV EBX,EAX
0040128E	. 03C2	ADD EAX,EDX
00401290	> 8907	MOV DWORD PTR [EDI],EAX
00401292	. C600 99	MOV BYTE PTR [EAX],99
00401295	. C603 00	MOV BYTE PTR [EBX],0
00401298	. 83C7 04	ADD EDI,4
0040129B	. 833F 00	CMP DWORD PTR [EDI],0
0040129E	.v 74 06	JE SHORT 3_1.004012A6
004012A0	. 8BC3	MOV EAX,EBX
004012A2	. 8B1F	MOV EBX,DWORD PTR [EDI]
004012A4	.^ EB EA	JMP SHORT 3_1.00401290
004012A6	> 8BC7	MOV EAX,EDI
004012A8	. 5F	POP EDI
004012A9	. C3	RET

Ta xét các bước xử lý đáng chú ý của hàm:

00401228 |. 24 03 |AND AL,3 ;

⇒ Nhận chỉ đường để di chuyển

Mỗi ký tự trong serial được giải mã thành một lệnh di chuyển theo lệnh "AND AL, 3":

0	-->	address + 10	-->	đi xuống
1	-->	address - 10	-->	đi lên
2	-->	address - 1	-->	qua trái
3	-->	address + 1	-->	qua phải

0040122A |. C0E9 02 |SHR CL,2 ;

⇒ Số di chuyển để thực hiện theo một hướng nhất định

Xác định số lượng các di chuyển để thực hiện theo một hướng nhất định từ lệnh "SHR CL, 2":

0 đến 3	-->	1 di chuyển
4 đến 7	-->	2 di chuyển
8 đến B	-->	3 di chuyển
C đến F	-->	4 di chuyển

Ta có bảng cách di chuyển dựa vào serial:

Serial Char	Hướng	Số lần di chuyển
0	0	1
1	1	
2	2	
3	3	
4	0	2
5	1	
6	2	
7	3	
8	0	3
9	1	
A	2	
B	3	
C	0	4
D	1	
E	2	
F	3	

0040124B |. 3C 99 |CMP AL,99 ;

⇒ Kiểm tra snake có ăn chính nó không

0040124F |. 3C CC |CMP AL,0CC ;

⇒ Kiểm tra snake có ăn thức ăn “CC” không

00401253 |. 3C DD |CMP AL,0DD ;

⇒ Kiểm tra snake có chạm điểm đích “DD” hay không

00401257 |. 833D 041F4000 00 |CMP DWORD PTR DS:[401F04],0 ;

⇒ Phải ăn hết số lượng thức ăn tương ứng với số ký tự của name, nếu không sẽ game over

Từ name, đã có hàm để phát sinh ma trận và tính toán để đưa các giá trị cần thiết vào ma trận. Sau đó, sẽ có hàm để kiểm tra serial.

Ta có thể mô tả luật chơi của game snake trong 3.1 như sau: Snake có đại diện là “99” trong ma trận, để đi đến chiến thắng, snake phải ăn hết các thức ăn “CC” và sau khi ăn hết thức ăn thì đi đến đích “DD”. Mỗi khi ăn một thức ăn “CC” snake sẽ dài ra thêm một đơn vị “99”, snake không được ăn chính nó, snake không được chạm “DD” khi chưa ăn hết thức ăn “CC” và snake không được đi qua khỏi khu vực ma trận 16x16.

***Bài 3.1 có keygen**, ý tưởng viết keygen là việc tìm đường đi từ đầu của snake đến 1 thức ăn, không được đụng điểm đích, cũng không được ăn chính nó (lưu thân snake vào vector) và không được qua ranh giới ma trận 16x16. Thực hiện vòng lặp với số lần lặp bằng với số ký tự của name, trong vòng lặp ta gọi hàm đệ quy tìm đường đến thức ăn như mô tả ở trên và liên tục cập nhật ma trận và thân snake. Sau khi kết thúc vòng lặp, tìm đường đi từ đầu của snake đến điểm đích, không được ăn chính nó. Cứ mỗi lần đi như vậy sẽ lưu hướng di chuyển vào một chuỗi. Sau khi kết thúc, ta dựa vào bảng cách di chuyển để chuyển thành hướng đi cho đúng quy định của 3.1. Dĩ nhiên, trong quá trình tìm đường, mỗi lần như vậy sẽ có hàm để tối ưu đường đi để cắt bớt đường đi thừa.

Một ví dụ về keygen với:

Name: abcde

Serial:

DD1BCCC83DDD9E2CCC83DDD92C82D9ACCC82DDD92CCC83DD5
3D1FF3CCC83DDD5

2. **Phần 2:** Bài tập trò chơi tìm key trên web:

Embedded Security CTF

<https://microcorruption.com>

Các level được xây dựng để chạy trên thiết bị điều khiển cỡ nhỏ MSP430. MSP430 là dòng vi điều khiển của Texas Instrument (TI), 16 bit, kiến trúc RISC được thiết kế đặc biệt cho siêu năng lượng thấp.

User Profile

Account: **lecongpr98**
 Name: **Cong Le**
 Score: **130**
 Email: **lecongpr98@gmail.com**
[Change password](#)

Levels Passed

Level	Score	Time Beaten	Min Input Size	Min CPU Steps
Tutorial	10	4/26/2018, 8:27:36 PM	8	2249
New Orleans	10	4/26/2018, 9:14:48 PM	7	2392
Sydney	15	4/26/2018, 9:27:05 PM	8	2245
Hanoi	20	4/26/2018, 9:39:39 PM	17	6199
Cusco	25	4/26/2018, 10:01:02 PM	18	5183
Whitehorse	50	4/26/2018, 10:23:07 PM	18	5178

Level Tutorial

Khai thác <check password> vì nó chỉ đơn giản kiểm tra độ dài password có bằng 9 hay không?

```

4484 <check_password>
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
  
```

Dòng 0x4484 load kí tự đầu tiên của password bạn vừa nhập vào r14.

Sau đó tăng thanh ghi r15 và r12 lên 1. Thanh ghi r12 giống như biến đếm số kí tự

Tiếp tục lặp lại cho đến khi nào gặp kí tự kết thúc chuỗi '\0'.

Nếu gặp NULL, cmp #0x9, r12, tức là kiểm tra số kí tự có bằng 9: gồm 8 kí tự và kí tự kết thúc chuỗi '\0'.

Nếu số kí tự là 9, thực hiện nhảy ở 0x4492 đến 0x4498 đến gán r15 thành 0x0001 còn không thì r15 bị clear về 0x0000

Nếu r15=0x0001 thì sau lệnh `tst r15` sẽ nhảy đến 0x445e có chuỗi “Access Granted!” và `<unlock_door>`

Disassembly				
444a:	0f41	mov	sp, r15	
444c:	b012 8444	call	#0x4484 <check_password>	
4450:	0f93	tst	r15	
4452:	0520	jnz	#0x445e <main+0x26>	
4454:	3f40 c744	mov	#0x44c7 "Invalid password; try again.", r15	
4458:	b012 5845	call	#0x4558 <puts>	
445c:	063c	jmp	#0x446a <main+0x32>	
445e:	3f40 e444	mov	#0x44e4 "Access Granted!", r15	
4462:	b012 5845	call	#0x4558 <puts>	
4466:	b012 9c44	call	#0x449c <unlock_door>	
446a:	0f43	clr	r15	
446c:	3150 6400	add	#0x64, sp	

Password: chuỗi bất kì có độ dài là 8

Ví dụ: “password”, “12345678”,...

Level New Orleans

4438	<main>			
4438:	3150 9cff	add	#0xff9c, sp	
443c:	b012 7e44	call	#0x447e <create_password>	
4440:	3f40 e444	mov	#0x44e4 "Enter the password to continue", r15	
4444:	b012 9445	call	#0x4594 <puts>	
4448:	0f41	mov	sp, r15	
444a:	b012 b244	call	#0x44b2 <get_password>	
444e:	0f41	mov	sp, r15	
4450:	b012 bc44	call	#0x44bc <check_password>	
4454:	0f93	tst	r15	
4456:	0520	jnz	#0x4462 <main+0x2a>	
4458:	3f40 0345	mov	#0x4503 "Invalid password; try again.", r15	
445c:	b012 9445	call	#0x4594 <puts>	
4460:	063c	jmp	#0x446e <main+0x36>	
4462:	3f40 2045	mov	#0x4520 "Access Granted!", r15	
4466:	b012 9445	call	#0x4594 <puts>	
446a:	b012 d644	call	#0x44d6 <unlock_door>	
446e:	0f43	clr	r15	
4470:	3150 6400	add	#0x64, sp	

Xem xét các lệnh `call`: `<create_password>` -> `<get_password>` -> `<check_password>`

Sau đó, có vẻ như sẽ dựa vào thanh ghi r15 để có thể `<unlock_door>`, nếu r15 != 0x0000 sẽ nhảy đến “Access Granted!”

Xét hàm `<create_password>`


```

447e <create_password>
447e: 3f40 0024      mov     #0x2400, r15
4482: ff40 7a00 0000 mov.b   #0x7a, 0x0(r15)
4488: ff40 6700 0100 mov.b   #0x67, 0x1(r15)
448e: ff40 5600 0200 mov.b   #0x56, 0x2(r15)
4494: ff40 5c00 0300 mov.b   #0x5c, 0x3(r15)
449a: ff40 2e00 0400 mov.b   #0x2e, 0x4(r15)
44a0: ff40 4f00 0500 mov.b   #0x4f, 0x5(r15)
44a6: ff40 7000 0600 mov.b   #0x70, 0x6(r15)
44ac: cf43 0700      mov.b   #0x0, 0x7(r15)
44b0: 3041          ret

```

Dùng mov.b lần lượt gán từng byte vào các vị trí 0-7 của thanh ghi r15, r15 lưu #0x2400 là địa chỉ của chuỗi password, #0x0 là kí tự kết thúc chuỗi

Kiểm tra hàm <check_password>

```

44bc <check_password>
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

```

Cmp.b so sánh từng byte và cmp #0x8, r14 cho thấy kiểm tra xem số kí tự có bằng 8 chưa. Có thể thấy địa chỉ bắt đầu so sánh giống với <create_password> đó là 0x2400. Như vậy, <create_password> ghi password vào bộ nhớ, <check_password> chỉ việc so sánh những byte đó của password đã lưu với password người dùng nhập. Nếu đúng thì r15 có giá trị #0x1

Đặt breakpoint sau hàm <create_password> vd: 0x4440 sau đó debug và xem tại 0x2400 lưu những giá trị gì

Live Memory Dump									
0000:	00 00	44 00	00 00	00 00	00 00	00 00	00 00	00 00	..D.....
0010:	*								
0150:	00 00	00 00	00 00	00 00	00 00	00 00	08 5a	00 00Z..
0160:	*								
2400:	7a 67	56 5c	2e 4f	70 00	00 00	00 00	00 00	00 00	zgV\Op.....
2410:	*								
4390:	00 00	00 00	00 00	00 00	00 00	40 44	00 00	00 00@D....
43a0:	*								

4438 <main>					
4438:	3150	9c ff	add	#0xff9c, sp	
443c:	b012	7e 44	call	#0x447e <create_password>	
4440:	3f40	e4 44	mov	#0x44e4 "Enter the password to continue", r15	
4444:	b012	94 45	call	#0x4594 <puts>	
4448:	0f41		mov	sp, r15	
444a:	b012	b2 44	call	#0x44b2 <get_password>	
444e:	0f41		mov	sp, r15	
4450:	b012	bc 44	call	#0x44bc <check_password>	
4454:	0f93		tst	r15	
4456:	0520		jnz	#0x4462 <main+0x2a>	
4458:	3f40	03 45	mov	#0x4503 "Invalid password; try again.", r15	
445c:	b012	94 45	call	#0x4594 <puts>	
4460:	063c		jmp	#0x446e <main+0x36>	
4462:	3f40	20 45	mov	#0x4520 "Access Granted!", r15	
4466:	b012	94 45	call	#0x4594 <puts>	
446a:	b012	d6 44	call	#0x44d6 <unlock_door>	
446e:	0f43		clr	r15	
4470:	3150	64 00	add	#0x64, sp	

Sau <check_password> nếu đúng password thì r15 có giá trị 0x0001 và nếu r15 khác 0x0 nó sẽ nhảy đến “Access Granted!”

Sau đó <unlock_door>

Password: 7a67565c2e4f7000 (Hex) hoặc “zgV\Op” (ASCII)

Level Sydney

Hàm <main>:

```
4438 <main>
4438: 3150 9cff      add     #0xff9c, sp
443c: 3f40 b444      mov     #0x44b4 "Enter the password to continue.", r15
4440: b012 6645      call    #0x4566 <puts>
4444: 0f41          mov     sp, r15
4446: b012 8044      call    #0x4480 <get_password>
444a: 0f41          mov     sp, r15
444c: b012 8a44      call    #0x448a <check_password>
4450: 0f93          tst     r15
4452: 0520          jnz     #0x445e <main+0x26>
4454: 3f40 d444      mov     #0x44d4 "Invalid password; try again.", r15
4458: b012 6645      call    #0x4566 <puts>
445c: 093c          jmp     #0x4470 <main+0x38>
445e: 3f40 f144      mov     #0x44f1 "Access Granted!", r15
4462: b012 6645      call    #0x4566 <puts>
4466: 3012 7f00      push    #0x7f
446a: b012 0245      call    #0x4502 <INT>
446e: 2153          incd    sp
4470: 0f43          clr     r15
4472: 3150 6400      add     #0x64, sp
```

Xem xét hàm <main> có thể thấy ở level này không còn <create_password> nữa

Qui trình: <get_password> -> <check_password> -> tst r15 ở 0x4450

Cần kiểm tra hàm <check_password>

```
448a <check_password>
448a: bf90 5033 0000 cmp     #0x3350, 0x0(r15)
4490: 0d20          jnz     $+0x1c
4492: bf90 5651 0200 cmp     #0x5156, 0x2(r15)
4498: 0920          jnz     $+0x14
449a: bf90 2848 0400 cmp     #0x4828, 0x4(r15)
44a0: 0520          jne     #0x44ac <check_password+0x22>
44a2: 1e43          mov     #0x1, r14
44a4: bf90 3838 0600 cmp     #0x3838, 0x6(r15)
44aa: 0124          jeq     #0x44ae <check_password+0x24>
44ac: 0e43          clr     r14
44ae: 0f4e          mov     r14, r15
44b0: 3041          ret
```

Có vẻ như thực hiện so sánh với các giá trị nằm tại vùng nhớ lưu trong thanh ghi r15. Có thể đó là từng phần của password cần tìm.

Lần lượt thực hiện so sánh 0x3350 với 0x0(r15), 0x5156 với 0x2(r15), 0x4828 với 0x4(r15) và 0x3838 với 0x6(r15). Chỉ cần một so sánh trả về kết quả khác nhau, tức zero flag = 0 sẽ nhảy đến 44ac để clear r14, ngược lại r14 sẽ có giá trị 0x1. Kết thúc hàm, nếu password đúng r15 sẽ có giá trị 0x1, sai thì r15 sẽ có giá trị 0x0

Tuy nhiên, ở đây sử dụng nền tảng Little Endian nên bit có trọng số nhỏ nhất LSB luôn được lưu ở ô nhớ có địa chỉ nhỏ nhất còn bit có trọng số lớn nhất MSB lưu ở ô nhớ có địa chỉ lớn nhất của vùng lưu trữ biến nên chỉ cần đảo lại giá trị của từng phần đã so sánh ở trên.

3350 5156 4828 3838 -> 5033 5651 2848 3838

```

4438 <main>
4438: 3150 9cff      add     #0xff9c, sp
443c: 3f40 b444      mov     #0x44b4 "Enter the password to continue.", r15
4440: b012 6645      call    #0x4566 <puts>
4444: 0f41           mov     sp, r15
4446: b012 8044      call    #0x4480 <get_password>
444a: 0f41           mov     sp, r15
444c: b012 8a44      call    #0x448a <check_password>
4450: 0f93           tst     r15
4452: 0520           jnz     #0x445e <main+0x26>
4454: 3f40 d444      mov     #0x44d4 "Invalid password; try again.", r15
4458: b012 6645      call    #0x4566 <puts>
445c: 093c           jmp     #0x4470 <main+0x38>
445e: 3f40 f144      mov     #0x44f1 "Access Granted!", r15
4462: b012 6645      call    #0x4566 <puts>
4466: 3012 7f00      push    #0x7f
446a: b012 0245      call    #0x4502 <INT>
446e: 2153           incd    sp
4470: 0f43           clr     r15
4472: 3150 6400      add     #0x64, sp

```

Password: 5033565128483838 (Hex) hoặc P3VQ(H88 (ASCII)

Level Hanoi

Check hàm <main>

```

4438 <main>
4438: b012 2045      call    #0x4520 <login>
443c: 0f43           clr     r15

```

Hàm main chỉ gọi hàm <login>

4520 <login>			
4520:	c243 1024	mov.b	#0x0, &0x2410
4524:	3f40 7e44	mov	#0x447e "Enter the password to continue.", r15
4528:	b012 de45	call	#0x45de <puts>
452c:	3f40 9e44	mov	#0x449e "Remember: passwords are between 8 and 16 characters
4530:	b012 de45	call	#0x45de <puts>
4540:	3f40 0024	mov	#0x2400, r15
4544:	b012 5444	call	#0x4454 <test_password_valid>
4548:	0f93	tst	r15
454a:	0324	jz	+\$+0x8
454c:	f240 ee00 1024	mov.b	#0xee, &0x2410
4552:	3f40 d344	mov	#0x44d3 "Testing if password is valid.", r15
4556:	b012 de45	call	#0x45de <puts>
455a:	f290 3200 1024	cmp.b	#0x32, &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>
456e:	3041	ret	
4570:	3f40 0145	mov	#0x4501 "That password is not correct.", r15
4574:	b012 de45	call	#0x45de <puts>
4578:	3041	ret	

Thanh ghi r15 một lần nữa có vai trò quan trọng trong việc mở khóa, nên ta sẽ xem hàm <test_password_valid> sẽ thay đổi giá trị thanh ghi r15 như thế nào.

Hàm <test_password_valid>

```
4454 <test_password_valid>
4454: 0412      push     r4
4456: 0441      mov      sp, r4
4458: 2453      incd     r4
445a: 2183      decd     sp
445c: c443 fcff  mov.b    #0x0, -0x4(r4)
4460: 3e40 fcff  mov      #0xfffc, r14
4464: 0e54      add      r4, r14
4466: 0e12      push     r14
4468: 0f12      push     r15

446a: 3012 7d00  push     #0x7d
446e: b012 7a45  call     #0x457a <INT>
4472: 5f44 fcff  mov.b    -0x4(r4), r15
4476: 8f11      sxt      r15
```

Sau khi đưa 0x7d vào stack, tại dòng 457a lại gán r14=0x7d, sau đó lại gán r15=r14=0x7d

Swpb r15 -> sau khi call #0x457a <INT> thì r15=0x7d00

Tại dòng 4472, gán 1 byte -0x4(r4) vào r15, mà trước đó tại dòng 445c có thể thấy -0x4(r4) = 0x0

Do đó r15=0x0000

```
457a <INT>
457a: 1e41 0200  mov      0x2(sp), r14
457e: 0212      push     sr
4580: 0f4e      mov      r14, r15
4582: 8f10      swpb     r15
4584: 024f      mov      r15, sr
4586: 32d0 0080  bis      #0x8000, sr
458a: b012 1000  call     #0x10
458e: 3241      pop      sr
4590: 3041      ret
```

Thử nhập password: “12345678”, Password vừa nhập lưu tại vùng nhớ 0x2400

```
2400: 3132 3334 3536 3738 0000 0000 0000 0000 12345678.....
2410: 0000 0000 0000 0000 0000 0000 0000 0000 .....
2420: 0000 0000 0000 0000 0000 0000 0000 0000 .....
2430: *
43e0: 0000 0000 0000 0000 0000 0000 8e45 0100 .....E..
43f0: 8e45 0300 0246 0000 0a00 0000 5a45 8c44 .E...F.....ZE<D
```

Xét lại đoạn chương trình trong <login>

```
4544: b012 5444    call    #0x4454 <test_password_valid>
4548: 0f93        tst     r15
454a: 0324        jz      $+0x8
454c: f240 ee00 1024 mov.b   #0xee, &0x2410
4552: 3f40 d344    mov     #0x44d3 "Testing if password is valid.", r15
4556: b012 de45    call    #0x45de <puts>
455a: f290 3200 1024 cmp.b   #0x32, &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>
456e: 3041        ret
```

Sau khi gọi hàm <test_password_valid> trả về r15 mang giá trị 0x0, do đó tại dòng 454a sẽ nhảy đến dòng 4552 “Testing if password is valid.”

Sau đó so sánh 0x32 với giá trị tại 0x2410, có thể thấy 0x2410 cách 0x2400 là vùng nhớ lưu password ở trên vừa đúng 16 bytes.

Tức là 0x2410 lưu kí tự thứ 17 trong chuỗi password, kết quả so sánh phải bằng nhau để có thể <unlock_door> nên kí tự thứ 17 là 32(Hex) hoặc ‘2’(ASCII)

Password: 16 kí tự tùy ý + kí tự ‘2’

Nhận xét: Hóa ra <test_password_valid> chỉ là tượng trưng còn chỗ mở khóa thực sự nằm ở chỗ cmp.b #0x32, &0x2410

Level Cusco

Tương tự Hanoi, ở hàm <main> chỉ gọi <login>

Nên ta xem hàm <login>

```
4500 <login>
4500: 3150 f0ff      add     #0xffff0, sp
4504: 3f40 7c44      mov     #0x447c "Enter the password to continue.", r15
4508: b012 a645      call    #0x45a6 <puts>
450c: 3f40 9c44      mov     #0x449c "Remember: passwords are between 8 and 16 characters", r15
4510: b012 a645      call    #0x45a6 <puts>
4514: 3e40 3000      mov     #0x30, r14
4518: 0f41           mov     sp, r15
451a: b012 9645      call    #0x4596 <getsn>
451e: 0f41           mov     sp, r15
4520: b012 5244      call    #0x4452 <test_password_valid>
4524: 0f93           tst     r15
4526: 0524           jz      #0x4532 <login+0x32>
4528: b012 4644      call    #0x4446 <unlock_door>
452c: 3f40 d144      mov     #0x44d1 "Access granted.", r15
4530: 023c           jmp     #0x4536 <login+0x36>
4532: 3f40 e144      mov     #0x44e1 "That password is not correct.", r15
4536: b012 a645      call    #0x45a6 <puts>
```

Hàm <test_password_valid> cùng logic với level Hanoi

```
4452 <test_password_valid>
4452: 0412           push    r4
4454: 0441           mov     sp, r4
4456: 2453           incd    r4
4458: 2183           decd    sp
445a: c443 fcff      mov.b   #0x0, -0x4(r4)
445e: 3e40 fcff      mov     #0xffffc, r14
4462: 0e54           add     r4, r14
4464: 0e12           push    r14
4466: 0f12           push    r15
4468: 3012 7d00      push    #0x7d
446c: b012 4245      call    #0x4542 <INT>
4470: 5f44 fcff      mov.b   -0x4(r4), r15
4474: 8f11           sxt     r15
4476: 3152           add     #0x8, sp
4478: 3441           pop     r4
447a: 3041           ret
```

Quay lại <login> ta có thể thấy, sau hàm <test_password_valid> r15 mang giá trị 0x0000 nên sẽ luôn nhảy đến 4532 “That password is not correct.”

Vậy làm cách nào để có thể nhảy đến <unlock_door>

Thử password 16 ký tự: “0123456789012345”

```
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
4540: <__do_nothing>
4540: 3041        ret

43d0: 0000 0000 0000 0000 0000 0000 5645 0100  ....VE..
43e0: 5645 0300 ca45 0000 0a00 0000 3a45 3031  VE...E.....:E01
43f0: 3233 3435 3637 3839 3031 3233 3435 0044  23456789012345.D      sp
4400: 3140 0044 1542 5c01 75f3 35d0 085a 3f40  1@.D.B\..u.S..Z?@
4410: 0000 0f93 0724 8245 5c01 2f83 9f4f d445  ....$.E\../..O.E
```

Có thể thấy với 16 ký tự vừa khít chạm tới stack pointer khi sắp kết thúc hàm <login>. Như vậy, do lệnh tiếp theo là ret nên địa chỉ tiếp theo được thực thi do stack pointer quyết định, do đó ta chỉ cần thêm 2 bytes vừa đúng ngay chỗ stack pointer với giá trị là địa chỉ của hàm <unlock_door> hoặc lệnh gọi <unlock_door>.

Mặt khác, <unlock_door> có địa chỉ 0x4446, lệnh call <unlock_door> có địa chỉ 0x4528. Mà ở đây sử dụng nền tảng Little Endian nên 2 ký tự còn lại cho password có mã hex là 4644 (“FD” trong ASCII) hoặc 2845 (“(E” trong ASCII).

Password: 16 ký tự bất kỳ + “FD” hoặc “(E”

Level Whitehorse

Hàm <main> chỉ gọi duy nhất hàm <login>

```
4438 <main>
4438: b012 f444    call    #0x44f4 <login>
```

Ta xem hàm <login>

```
44f4 <login>
44f4: 3150 f0ff    add     #0xffff0, sp
44f8: 3f40 7044    mov     #0x4470 "Enter the password to continue.", r15
44fc: b012 9645    call    #0x4596 <puts>
4500: 3f40 9044    mov     #0x4490 "Remember: passwords are between 8 and 16 characters", r15
4504: b012 9645    call    #0x4596 <puts>
4508: 3e40 3000    mov     #0x30, r14
450c: 0f41        mov     sp, r15
450e: b012 8645    call    #0x4586 <getsn>
4512: 0f41        mov     sp, r15
4514: b012 4644    call    #0x4446 <conditional_unlock_door>
4518: 0f93        tst     r15
451a: 0324        jz      #0x4522 <login+0x2e>
451c: 3f40 c544    mov     #0x44c5 "Access granted.", r15
4520: 023c        jmp     #0x4526 <login+0x32>
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
```

Ở <login> chủ yếu là hàm <conditional_unlock_door> sau đó kiểm tra r15 có bằng 0x0 hay không? Nếu r15=0x0 thì nhảy đến 4522 “That password is not correct.”. Ngược lại r15 != 0x0 thì “Access granted.”

Xét <conditional_unlock_door>

```
4446 <conditional_unlock_door>
4446: 0412        push    r4
4448: 0441        mov     sp, r4
444a: 2453        incd    r4
444c: 2183        decd    sp
444e: c443 fcff    mov.b   #0x0, -0x4(r4)
4452: 3e40 fcff    mov     #0xffffc, r14
4456: 0e54        add     r4, r14
4458: 0e12        push    r14
445a: 0f12        push    r15
445c: 3012 7e00    push    #0x7e
4460: b012 3245    call    #0x4532 <INT>
4464: 5f44 fcff    mov.b   -0x4(r4), r15
4468: 8f11        sxt     r15
446a: 3152        add     #0x8, sp
446c: 3441        pop     r4
446e: 3041        ret
```

Một lần nữa, kết quả r15 luôn trả về 0x00 nên có thể tiếp tục là password gây stack overflow bằng cách nhập nhiều hơn 16 ký tự.

Ví dụ nhập pass: “AAAAAAAAAAAAAAAAAAAAA”

```

3ec0: 4645 0100 4645 0300 ba45 0000 0a00 0000 FE..FE...E.....
3ed0: 2a45 4141 4141 4141 4141 4141 4141 4141 *EAAAAAAAAAAAAA
3ee0: 4141 4141 4141 0000 0000 0000 0000 0000 AAAAAA.....
3ef0: *

```

Có thể thấy vị trí bắt đầu vùng nhớ lưu trữ password vừa nhập bắt đầu từ 0x3ed2, tương tự như bài Cusco, stack pointer nằm ở byte thứ 17, 18 và stack pointer quyết định sau khi return sẽ gọi lệnh ở địa chỉ nào, nên cần phải nhập password khoảng 18 ký tự.

Trong file hướng dẫn Lock manual hay cũng như các level trước, có một mã số được dùng để unlock the door đó chính là 0x7f

```

445c: 3012 7e00    push    #0x7e
4460: b012 3245    call    #0x4532 <INT>

```

Ở đây dùng 0x7e, tuy nhiên, ta có thể dùng opcode của 2 lệnh trên và thay đổi thành 0x7f sau đó chèn vào đoạn Password nhập vào để nó có thể lưu vào trong Live Memory Dump, khi cần có thể gọi đến Dump tại địa chỉ đó để thực hiện lệnh theo opcode.

Instructions			Assembled Objects
3012 7f00	push	#0x7f	3012 7f00
b012 3245	call	#0x4532	b012 3245
3012 7e00	push	#0x7e	3012 7e00
b012 3245	call	#0x4532	b012 3245

Như vậy phần opcode chèn vào password là: 3012 7f00 b012 3245.

Kết luận: Phần opcode chèn vào đầu password chiếm 8 bytes, tiếp theo điền thêm 8 bytes bất kỳ để được 16 bytes, sau đó byte thứ 17 và 18 quyết định lệnh tiếp theo được thực hiện nên sẽ mang địa chỉ bắt đầu của password mà user nhập, mà địa chỉ bắt đầu là 0x3ed2 nhưng giá trị ghi trong password phải là d23e do định dạng Little Endian.

Password: 3012 7f00 b012 3245 + 8 bytes bất kỳ + d23e