

Câu 2.1.exe

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

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

```
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

```
0040153F . 68 24704000 PUSH 2_1.00407024 [Arg1 = 00407024 ASCII "You rule!"]
00401544 . E8 37070000 CALL 2_1.00401C80 2_1.00401C80
```

-Bad boy

```
0040154C .> 68 1C704000 PUSH 2_1.0040701C ASCII "Wrong."
00401551 . E8 2A070000 CALL 2_1.00401C80
```

-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
004014A0 . E8 0B080000 CALL 2_1.00401C80 2_1.00401C80
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 |
| 00401008 | . 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 $ 53 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 EDI,DWORD PTR [408CA0]
00401056 . 01D0 ADD EAX,EDI
00401058 . 8B4C24 08 MOV ECX,DWORD PTR [ESP+8]
0040105C . 31D2 XOR EDI,EDI
0040105E . F7F1 DIV ECX
00401060 . 89D3 MOV EBX,EDI
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 trống được thay bằng '.' cho dễ quan sát).



-Xét quá trình xử lí 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?

| | | |
|----------|------------------|----------------------------|
| 0040115D | 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 | > 0FBE03 | MOVSX EAX, BYTE PTR [EBX] |
| 00401175 | . 8945 FC | MOV DWORD PTR [EBP-4],EAX |
| 00401178 | . 83F8 64 | CMP EAX,64 |
| 0040117B | .v 74 41 | JE SHORT 2_1.004011BE |
| 0040117D | .v 7F 0F | JG SHORT 2_1.0040118E |
| 0040117F | . 837D FC 00 | CMP DWORD PTR [EBP-4],0 |
| 00401183 | .v 0F84 68000000 | JE 2_1.004011F1 |
| 00401189 | .v E9 81000000 | JMP 2_1.0040120F |
| 0040118E | > 8B45 FC | MOV EAX,DWORD PTR [EBP-4] |
| 00401191 | . 83F8 6C | CMP EAX,6C |
| 00401194 | .v 74 3C | JE SHORT 2_1.004011D2 |
| 00401196 | .v 0F8C 73000000 | JL 2_1.0040120F |
| 0040119C | . 8B45 FC | MOV EAX,DWORD PTR [EBP-4] |
| 0040119F | . 83F8 72 | CMP EAX,72 |
| 004011A2 | .v 74 3C | JE SHORT 2_1.004011E0 |
| 004011A4 | . 83F8 75 | CMP EAX,75 |
| 004011A7 | .v 0F85 62000000 | JNZ 2_1.0040120F |
| 004011AD | . 85C9 | TEST ECX,ECX |
| 004011AF | .v 75 07 | JNZ SHORT 2_1.004011B8 |
| 004011B1 | . 31C0 | XOR EAX,EAX |
| 004011B3 | .v E9 7E000000 | JMP 2_1.00401236 |
| 004011B8 | > 49 | DEC ECX |
| 004011B9 | .v E9 55000000 | JMP 2_1.00401213 |
| 004011BE | > A1 A48C4000 | MOV EAX,DWORD PTR [408CA4] |
| 004011C3 | . 48 | DEC EAX |
| 004011C4 | . 39C1 | CMP ECX,EAX |
| 004011C6 | .v 75 07 | JNZ SHORT 2_1.004011CF |
| 004011C8 | . 31C0 | XOR EAX,EAX |
| 004011CA | .v E9 67000000 | JMP 2_1.00401236 |
| 004011CF | > 41 | INC ECX |
| 004011D0 | .v EB 41 | JMP SHORT 2_1.00401213 |
| 004011D2 | > 85F6 | TEST ESI,ESI |
| 004011D4 | .v 75 07 | JNZ SHORT 2_1.004011D0 |
| 004011D6 | . 31C0 | XOR EAX,EAX |
| 004011D8 | .v E9 59000000 | JMP 2_1.00401236 |
| 004011DD | > 4E | DEC ESI |
| 004011DE | .v 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 | ..v 75 04 | JNZ SHORT 2_1.004011EE |
| 004011EA | . 31C0 | XOR EAX,EAX |
| 004011EC | ..v EB 48 | JMP SHORT 2_1.00401236 |
| 004011EE | > 46 | INC ESI |
| 004011EF | ..v 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 | ..v 75 05 | JNZ SHORT 2_1.0040120F |
| 0040120A | . 31C0 | XOR EAX,EAX |
| 0040120C | . 40 | INC EAX |
| 0040120D | ..v EB 27 | JMP SHORT 2_1.00401236 |
| 0040120F | > 31C0 | XOR EAX,EAX |
| 00401211 | ..v 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 | ..v 75 04 | JNZ SHORT 2_1.00401230 |
| 0040122C | . 31C0 | XOR EAX,EAX |
| 0040122E | ..v 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 đụng 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;
}
```


$\{ \dots \}$

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]