

## BabyVm

### Chạy thử chương trình

```
C:\Users\BILL\Desktop\Release_3\BabyVm>.\VirtualMachine.exe
V1r7u41 M4ch1n3 by <@shockbyte>
password->fsaf
fsdfdfdfd
fdfdf
Press any key to continue . . .
```

### Phân tích với IDA

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     int v3; // eax
4     void *Buffer; // [esp+8h] [ebp-8h]
5     FILE *Stream; // [esp+Ch] [ebp-4h] BYREF
6
7     v3 = sub_7E1040(std::cout, "V1r7u41 M4ch1n3 by <@shockbyte>");
8     std::ostream::operator<<(v3, sub_7E1410);
9     Stream = 0;
10    Buffer = (void *)alloc(1340);
11    fopen_s(&Stream, "vm_dump.vm", "rb");
12    if ( !Stream )
13        exit(0);
14    fread_s(Buffer, 0x53Cu, 1u, 0x53Cu, Stream);
15    fclose(Stream);
16    sub_7E38E0((void **)&unk_7E9090, Buffer, 0x53Cu);
17    sub_7E3820(&unk_7E9090);
18    j_j_free(Buffer);
19    system("\r\npause");
20    return 0;
21 }
```

Ta có thể đoán được đây là một bài thuộc dạng Virtual Machine Implement. Chương trình sẽ chạy dựa trên nội dung file “vm\_dump.vm”. Đầu tiên chương trình đọc nội dung file vm\_dump vào mảng Buffer, sau đó gọi hàm sub\_7E38E0

```

1 void *__thiscall sub_4038E0(void **this, void *Src, size_t Size)
2 {
3     this[6] = (void *)sub_4052B0(Size);
4     this[7] = (void *)sub_4052B0(Size);
5     while ( !(unsigned __int8)sub_403790(this + 1) )
6         sub_403A00(this + 1);
7     memset(this[6], 0, Size);
8     memcpy(this[6], Src, Size);
9     *this = this[6];
10    if ( *(_DWORD *)*this != 4919 )
11        exit(0);
12    this[7] = (void *)sub_4052B0(*((_DWORD *)*this + 1));
13    return memset(this[7], 0, *((_DWORD *)*this + 1));
14 }

```

unk\_7E9090 là nơi chứa thông tin của “máy ảo” mà chúng ta đang implement.

Trong đó mảng tại vị trí unk\_7E9090 [6] chứa toàn bộ Buffer vừa được đọc từ file “vm\_dump.vm”

Tiếp theo đến với hàm sub\_7E3820 ở ngay dưới.

```

1 _DWORD *__thiscall sub_7E3820(_DWORD *this)
2 {
3     _DWORD *result; // eax
4     _BYTE *v2; // [esp+4h] [ebp-8h]
5
6     this[12] = this[6] + 8;
7     do
8     {
9         v2 = (_BYTE *)this[12];
10        sub_7E3D90(v2);
11        result = this;
12        this[12] += 12;
13    }
14    while ( *v2 != 63 );
15    return result;
16 }

```

Hàm này lấy từng block giá trị bắt đầu từ unk\_7E9090 [6] + 8, mỗi block cách nhau 12 byte. Sau đó từng block được đưa vào hàm sub\_7E3D90 để xử lí. Có vẻ như các block 12byte này sẽ là từng câu lệnh riêng biệt và đưa vào chương trình để xử lí. Mỗi câu lệnh có độ dài cố định như trong cấu trúc MIPS hay ARM. Vị trí unk\_7E9090[12] có tác dụng giống như thanh ghi eip.

```

1 BYTE *__stdcall sub_7E3D90( BYTE *a1)
2 {
3     BYTE *result; // eax
4
5     result = a1;
6     switch ( *a1 )
7     {
8         case 0:
9         case 1:
10        case 2:
11        case 3:
12        case 4:
13        case 5:
14        case 6:
15            result = ( BYTE *)sub_7E4600(a1);
16            break;
17        case 7:
18        case 8:
19        case 9:
20            result = ( BYTE *)sub_7E4D70(a1);
21            break;
22        case 0xA:
23        case 0xB:
24        case 0xC:
25            result = ( BYTE *)sub_7E4CB0(a1);
26            break;
27        case 0xD:
28        case 0xE:
29        case 0xF:
30        case 0x10:
31        case 0x11:
32            result = ( BYTE *)sub_7E3F40(a1);

```

Đến với hàm sub\_7E3D90, là một lệnh switch bao gồm tổng cộng 0x3E trường hợp. Đến đây thì gần như chắc chắn byte đầu tiên của từng block 12 byte chính là opcode của câu lệnh. Và dựa vào đó chương trình sẽ thực hiện tương ứng.

Một số hàm cần để ý của chương trình

Hàm sub\_7E3D00 gán 1 trong các giá trị của this[8->11] theo giá trị đầu vào

```

1 _BYTE *__thiscall sub_7E3D00(_DWORD *this, _BYTE *a2, _DWORD *a3)
2 {
3     _BYTE *result; // eax
4
5     result = a2;
6     switch ( *a2 )
7     {
8         case 0:
9             result = this;
10            this[8] = *a3;
11            break;
12        case 1:
13            result = this;
14            this[9] = *a3;
15            break;
16        case 2:
17            result = this;
18            this[10] = *a3;
19            break;
20        case 3:
21            result = this;
22            this[11] = *a3;
23            break;
24        default:
25            return result;
26    }
27    return result;
28 }

```

Hàm sub\_7E21F0 lại gán giá trị a3 vào vùng nhớ this[7] + a2

```

1 int __thiscall sub_7E21F0(_DWORD *this, int *a2, _DWORD *a3)
2 {
3     int result; // eax
4
5     result = *a2;
6     *(_DWORD *)(this[7] + *a2) = *a3;
7     return result;
8 }

```

Hàm sub\_7E21C0 trả về giá trị tương ứng tại this[7] + a2

```
1 int __thiscall sub_7E21C0(_DWORD *this, _DWORD *a2)
2 {
3     return *(unsigned __int8 *)(this[7] + *a2);
4 }
```

Hàm sub\_7E3880 trả về giá trị tương ứng tại this[8->11] dựa trên a2

```
1 int __thiscall sub_7E3880(_DWORD *this, _BYTE *a2)
2 {
3     int result; // eax
4
5     switch ( *a2 )
6     {
7         case 0:
8             result = this[8];
9             break;
10        case 1:
11            result = this[9];
12            break;
13        case 2:
14            result = this[10];
15            break;
16        case 3:
17            result = this[11];
18            break;
19        default:
20            result = 0;
21            break;
22    }
23    return result;
24 }
```

Ta có nhận xét như sau

Phần nhớ unk\_7E9090 [8->11] tương ứng với giá trị các thanh ghi mà chương trình sử dụng, unk\_7E9090[7] là đại diện cho vùng nhớ của chương trình.

Đến với các case từ 0->6: hàm sub\_7E4600 được gọi. Trong hàm sub\_7E4600 lại dựa theo các giá trị để gọi hàm sub\_7E3D00 hoặc sub\_7E21F0.

```
switch ( v11 )
{
    case 0:
        v20 = a2[4];
        result = (unsigned __int8 *)sub_7E3D00(&v20, a2 + 8);
        break;
    case 1:
        v19 = a2[8];
        v10 = sub_7E3880(&v19);
        v18 = a2[4];
        result = (unsigned __int8 *)sub_7E3D00(&v18, &v10);
        break;
    case 2:
        v3 = sub_7E21C0(a2 + 8);
        v13 = a2[4];
        result = (unsigned __int8 *)sub_7E3D00(&v13, &v3);
        break;
    case 3:
        v17 = a2[4];
        v9 = sub_7E3880(&v17);
        result = (unsigned __int8 *)sub_7E21F0(&v9, a2 + 8);
        break;
    case 4:
        v16 = a2[8];
        v8 = sub_7E3880(&v16);
        v7 = sub_7E21C0(&v8);
        v15 = a2[4];
        result = (unsigned __int8 *)sub_7E3D00(&v15, &v7);
        break;
    case 5:
        v6 = a2[4];
```

Từ 2 hàm trên ta có thể đoán được các opcode từ 0->6 là các lệnh mov, trong đó chia ra 4 lệnh mov đến register được chạy bởi hàm sub\_7E3D00, 3 lệnh mov đến vùng nhớ được chạy bởi sub\_7E21F0 (trong đó unk\_7E9090[7] là vùng nhớ của chương trình, unk\_7E9090[8->11] lần lượt chứa giá trị của các register chương trình sử dụng).

Tiếp theo là hàm sub\_7E4D70 cho các case từ 7->9

```
1 char *__thiscall sub_7E4D70(void *this, char *a2)
2 {
3     char *result; // eax
4     int v3; // [esp+0h] [ebp-18h] BYREF
5     int v4; // [esp+4h] [ebp-14h] BYREF
6     int v5; // [esp+8h] [ebp-10h] BYREF
7     char v6; // [esp+Ch] [ebp-Ch]
8     void *v7; // [esp+10h] [ebp-8h]
9     char v8; // [esp+16h] [ebp-2h] BYREF
10    char v9; // [esp+17h] [ebp-1h] BYREF
11
12    v7 = this;
13    result = a2;
14    v6 = *a2;
15    switch ( v6 )
16    {
17        case 7:
18            return (char *)sub_7E3A50(a2 + 4);
19        case 8:
20            v9 = a2[4];
21            v5 = sub_7E3880(&v9);
22            result = (char *)sub_7E3A20(&v5);
23            break;
24        case 9:
25            v8 = a2[4];
26            v4 = sub_7E3880(&v8);
27            v3 = sub_7E21C0(&v4);
28            result = (char *)sub_7E3A20(&v3);
29            break;
30    }
31    return result;
32 }
```

Đây là hàm mình đau đầu nhất vì nó quá phức tạp và dài, tuy nhiên dựa vào kinh nghiệm và sau khi đã dịch hết tất cả các hàm trong các case còn lại, mình đoán đây là hàm xử lý cho stack, cụ thể là push stack. Case 7 chính là “push val”, case 8 là “push reg”, case 9 là “push [reg]”. Vì hàm sub\_7E3880 sẽ trả về giá trị của register và hàm sub\_7E21C0 sẽ trả về giá trị tại vùng nhớ unk\_7E9090[7]

Tiếp theo là hàm sub\_7E4CB0 dành cho các lệnh pop stack, case 10 là “pop reg”, case 11 là “pop [reg]”, và case 12 là “pop [mem]”,

```
1 int __thiscall sub_7E4CB0(_DWORD *this, int a2)
2 {
3     int result; // eax
4     _DWORD *v3; // eax
5     _DWORD *v4; // eax
6     _DWORD *v5; // eax
7     int v6; // [esp+0h] [ebp-10h] BYREF
8     char v7; // [esp+4h] [ebp-Ch]
9     _DWORD *v8; // [esp+8h] [ebp-8h]
10    char v9; // [esp+Eh] [ebp-2h] BYREF
11    char v10; // [esp+Fh] [ebp-1h] BYREF
12
13    v8 = this;
14    result = a2;
15    v7 = *(_BYTE *)a2;
16    switch ( v7 )
17    {
18        case 10:
19            v10 = *(_BYTE *)(a2 + 4);
20            v3 = (_DWORD *)sub_7E3D70(v8 + 1);
21            sub_7E3D00(v8, &v10, v3);
22            result = sub_7E3A00(v8 + 1);
23            break;
24        case 11:
25            v9 = *(_BYTE *)(a2 + 4);
26            v6 = sub_7E3880(v8, &v9);
27            v4 = (_DWORD *)sub_7E3D70(v8 + 1);
28            sub_7E21F0(v8, &v6, v4);
29            result = sub_7E3A00(v8 + 1);
30            break;
31        case 12:
32            v5 = (_DWORD *)sub_7E3D70(v8 + 1);
33            sub_7E21F0(v8, (int *)(a2 + 4), v5);
34            result = sub_7E3A00(v8 + 1);
35            break;
36    }
```

Tiếp theo là hàm sub\_7E3F40 dành cho các phép toán cộng



```

result = v10,
switch ( (unsigned int)v10 )
{
    case 0u:
        v25 = a2[4];
        v12 = sub_7E3880(v13, &v25);
        v24 = a2[8];
        v11 = sub_7E3880(v13, &v24);
        v9 = v11 + v12;
        v23 = a2[4];
        result = sub_7E3D00(v13, &v23, &v9);
        break;
    case 1u:
        v22 = a2[4];
        v12 = sub_7E3880(v13, &v22);
        v11 = *((_DWORD *)a2 + 2);
        v8 = v11 + v12;
        v21 = a2[4];
        result = sub_7E3D00(v13, &v21, &v8);
        break;
    case 2u:
        v20 = a2[4];
        v12 = sub_7E3880(v13, &v20);
        v11 = sub_7E21C0(v13, (_DWORD *)a2 + 2);
        v7 = v11 + v12;
        v19 = a2[4];
        result = sub_7E3D00(v13, &v19, &v7);
        break;
    case 3u:
        v18 = a2[4];
        v12 = sub_7E3880(v13, &v18);
        v11 = *((_DWORD *)a2 + 2);
        v6 = v11 + v12;
        v17 = a2[4];
        v5 = sub_7E3880(v13, &v17);
        result = (_BYTE *)sub_7E21F0(v13, &v5, &v6);
}

```

Case 13 là “add reg reg”, case 14 là “add reg val”, case 15 là “add reg [mem]”, case 16 là “add [reg] val”, case 17 là “add [reg] reg”

Các hàm tiếp theo nữa rất dài tuy nhiên mình tóm gọn như sau:

Case 18: sub reg reg

Case 19: sub reg val

Case 20: sub reg [mem]

Case 21: sub [reg] val

Case 22: sub [reg] reg

Case 23: mul reg reg

Case 24: mul reg val

Case 25: mul reg [mem]

Case 26: mul [reg] val

Case 27: mul [reg] reg

Case 28: xor reg reg

Case 29: xor reg val

Case 30: xor reg [mem]

Case 31: xor [reg] val

Case 32: xor [reg] reg

Case 33: and reg reg

Case 34: and reg val

Case 35: and reg [mem]

Case 36: and [reg] val

Case 37: and [reg] reg

Case 38:

Case 39:

Case 40:

Case 41:

Case 42:

Case 43: not reg

Case 44: not [reg]

Case 45: not [mem]

Case 46: cmp reg reg

Case 47: cmp reg val

Case 48: cmp reg [mem]

Case 49: cmp [reg] val

Case 50:

Case 51: save\_return\_address (unk\_7E9090[13] = unk\_7E9090[12])

Case 52: make\_label\_tojump val (unk\_7E9090[14] = unk\_7E9090[12] + 12 \* val)

Case 53: delete\_save\_return\_address (unk\_7E9090[13] = 0)

Case 54: jmp return\_address (unk\_7E9090[12] = unk\_7E9090[13])

Case 55: jmp label (unk\_7E9090[12] = unk\_7E9090[14])

Case 56: jz return\_address (unk\_7E9090[12] = unk\_7E9090[13])

Case 57: jz label (unk\_7E9090[12] = unk\_7E9090[14])

Case 58: jnz return\_address (unk\_7E9090[12] = unk\_7E9090[13])

Case 59: jnz label (unk\_7E9090[12] = unk\_7E9090[14])

Case 60: putchar reg

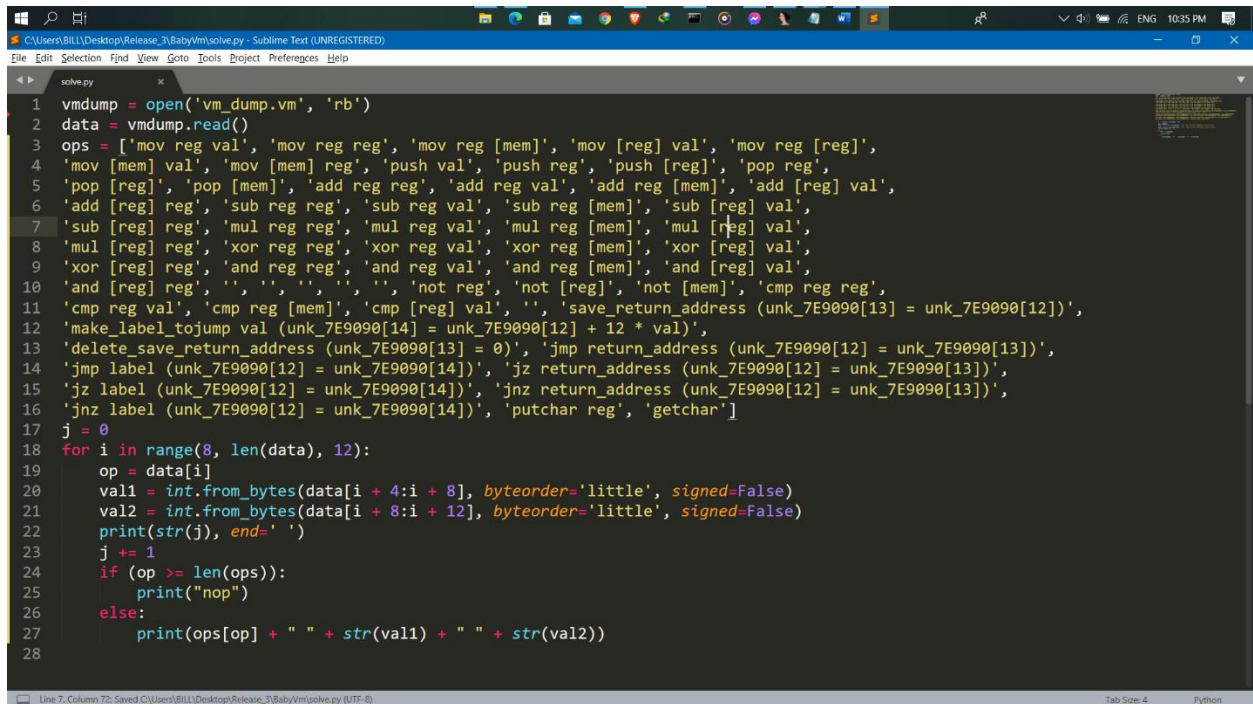
Case 61: getchar

Default:

Trong đó ngoài trừ 2 hàm putchar và getchar là tương tác với màn hình, còn lại tất cả các lệnh sẽ được chạy trên một bộ máy ảo với các thông tin lưu ở unk\_7E9090, trong đó unk\_7E9090[6] chứa các lệnh, unk\_7E9090[7] chứa vùng nhớ, unk\_7E9090[15] chứa vị trí lớn nhất của vùng nhớ của unk\_7E9090[7], unk\_7E9090[8->11] chứa các thanh ghi, unk\_7E9090[12] là địa chỉ con trỏ lệnh, unk\_7E9090[13] là địa chỉ lưu về và unk\_7E9090[14] là địa chỉ nhảy đến, (byte)(unk\_7E9090 + 64) là flag zf sử dụng trong các nhảy có điều kiện. Trong đó

các lệnh từ 51->53 sẽ kết hợp với các lệnh từ 54->59 để tạo ra các lệnh nhảy.  
unk\_7E9090[13] giống như thanh ghi \$ra trong kiến trúc MIPS vậy.

Từ các phân tích trên, ta có thể viết đoạn script có để biên dịch file “vm\_dump.vm” thành một đoạn mã giả như sau:



```
1 vmdump = open('vm_dump.vm', 'rb')
2 data = vmdump.read()
3 ops = ['mov reg val', 'mov reg reg', 'mov reg [mem]', 'mov [reg] val', 'mov reg [reg]',
4 'mov [mem] val', 'mov [mem] reg', 'push val', 'push reg', 'push [reg]', 'pop reg',
5 'pop [reg]', 'pop [mem]', 'add reg reg', 'add reg val', 'add reg [mem]', 'add [reg] val',
6 'add [reg] reg', 'sub reg reg', 'sub reg val', 'sub reg [mem]', 'sub [reg] val',
7 'sub [reg] reg', 'mul reg reg', 'mul reg val', 'mul reg [mem]', 'mul [reg] val',
8 'mul [reg] reg', 'xor reg reg', 'xor reg val', 'xor reg [mem]', 'xor [reg] val',
9 'xor [reg] reg', 'and reg reg', 'and reg val', 'and reg [mem]', 'and [reg] val',
10 'and [reg] reg', 'not reg', 'not [reg]', 'not [mem]', 'cmp reg reg',
11 'cmp reg val', 'cmp reg [mem]', 'cmp [reg] val', 'save_return_address (unk_7E9090[13] = unk_7E9090[12])',
12 'make_label_tojump val (unk_7E9090[14] = unk_7E9090[12] + 12 * val)',
13 'delete_save_return_address (unk_7E9090[13] = 0)', 'jmp return_address (unk_7E9090[12] = unk_7E9090[13])',
14 'jmp label (unk_7E9090[12] = unk_7E9090[14])', 'jz return_address (unk_7E9090[12] = unk_7E9090[13])',
15 'jz label (unk_7E9090[12] = unk_7E9090[14])', 'jnz return_address (unk_7E9090[12] = unk_7E9090[13])',
16 'jnz label (unk_7E9090[12] = unk_7E9090[14])', 'putchar reg', 'getchar']
17 j = 0
18 for i in range(8, len(data), 12):
19     op = data[i]
20     val1 = int.from_bytes(data[i + 4:i + 8], byteorder='little', signed=False)
21     val2 = int.from_bytes(data[i + 8:i + 12], byteorder='little', signed=False)
22     print(str(j), end=' ')
23     j += 1
24     if (op >= len(ops)):
25         print("nop")
26     else:
27         print(ops[op] + " " + str(val1) + " " + str(val2))
28
```

Chạy đoạn script ta được

0 push val 2 0

1 push val 17 0

2 push val 88 0

3 push val 78 0

4 push val 83 0

5 push val 75 0

6 push val 79 0

7 push val 79 0

8 push val 93 0

9 push val 76 0

10 mov reg val 0 10

11 mov reg val 1 60  
12 save\_return\_address (unk\_7E9090[13] = unk\_7E9090[12]) 0 0  
13 pop reg 2 0  
14 xor reg reg 2 1  
15 putchar reg 2 0  
16 sub reg val 0 1  
17 cmp reg val 0 0  
18 jnz return\_address (unk\_7E9090[12] = unk\_7E9090[13]) 0 0  
19 delete\_save\_return\_address (unk\_7E9090[13] = 0) 0 0  
20 push val 141 0  
21 push val 222 0  
22 push val 159 0  
23 push val 199 0  
24 push val 207 0  
25 push val 152 0  
26 push val 222 0  
27 push val 207 0  
28 push val 243 0  
29 push val 220 0  
30 push val 156 0  
31 push val 216 0  
32 push val 243 0  
33 push val 193 0  
34 push val 152 0  
35 push val 243 0

```
36 push val 197 0
37 mov reg val 0 17
38 push reg 0 0
39 save_return_address (unk_7E9090[13] = unk_7E9090[12]) 0 0
40 getchar 0 0
41 sub reg val 0 1
42 cmp reg val 0 0
43 jnz return_address (unk_7E9090[12] = unk_7E9090[13]) 0 0
44 make_label_tojump val (unk_7E9090[14] = unk_7E9090[12] + 12 * val) 16 0
45 delete_save_return_address (unk_7E9090[13] = 0) 0 0
46 pop reg 0 0
47 mov reg val 1 0
48 save_return_address (unk_7E9090[13] = unk_7E9090[12]) 0 0
49 mov reg [reg] 2 1
50 xor reg val 2 172
51 pop reg 3 0
52 cmp reg reg 3 2
53 jnz label (unk_7E9090[12] = unk_7E9090[14]) 0 0
54 add reg val 1 1
55 sub reg val 0 1
56 cmp reg val 0 0
57 jnz return_address (unk_7E9090[12] = unk_7E9090[13]) 0 0
58 make_label_tojump val (unk_7E9090[14] = unk_7E9090[12] + 12 * val) 4 0
59 jmp label (unk_7E9090[12] = unk_7E9090[14]) 0 0
60 nop
```

61 nop  
62 nop  
63 push val 49 0  
64 push val 54 0  
65 push val 70 0  
66 push val 8 0  
67 push val 86 0  
68 push val 100 0  
69 push val 8 0  
70 push val 14 0  
71 push val 73 0  
72 push val 8 0  
73 push val 77 0  
74 push val 8 0  
75 push val 73 0  
76 push val 100 0  
77 push val 12 0  
78 push val 85 0  
79 push val 11 0  
80 push val 95 0  
81 push val 100 0  
82 push val 8 0  
83 push val 126 0  
84 push val 15 0  
85 push val 8 0

86 push val 10 0  
87 push val 75 0  
88 push val 64 0  
89 push val 121 0  
90 push val 115 0  
91 push val 104 0  
92 push val 5 0  
93 push val 22 0  
94 push val 92 0  
95 push val 90 0  
96 push val 87 0  
97 push val 93 0  
98 mov reg val 0 35  
99 mov reg val 1 59  
100 save\_return\_address (unk\_7E9090[13] = unk\_7E9090[12]) 0 0  
101 pop reg 2 0  
102 xor reg reg 2 1  
103 putchar reg 2 0  
104 sub reg val 0 1  
105 cmp reg val 0 0  
106 jnz return\_address (unk\_7E9090[12] = unk\_7E9090[13]) 0 0  
107 delete\_save\_return\_address (unk\_7E9090[13] = 0) 0 0  
108 nop  
109 nop  
110 nop



Đoạn từ 0->9 chương trình push một chuỗi giá trị lên stack, sau đó đoạn 12->18 lặp 10 giá trị trên stack, xor với reg[1] (= 60) và xuất ra màn hình.

Tính thử chuỗi xuất ta thấy chuỗi xuất là

```
>>> a = [76, 93, 79, 79, 75, 83, 78, 88, 17, 2]
>>> for i in a:
...     print(chr(i ^ 60), end='')
... else:
...     print()
...
password->
>>>
```

Bằng với chuỗi lúc ta chạy thử chương trình, vậy là chúng ta đã biên dịch đúng. Tiếp tục với đoạn phía sau.

Đoạn từ 20->36 tiếp tục push một chuỗi các giá trị lên stack, sau đó đoạn từ 37->43 lặp lại 17 lần, mỗi lần lấy 1 ký tự từ hàm getchar() và lưu vào vùng nhớ, đoạn từ 44->57 lặp lại 17 lần, lấy 17 giá trị vừa nhập lần lượt xor với 172 và so sánh với chuỗi vừa đưa vào ở đoạn 20->36 trên stack. Ta tiếp tục tính toán chuỗi.

```
>>> a = [197,243, 152, 193, 243, 216, 156, 220, 243, 207, 222, 152, 207, 199, 159, 222, 141]
>>> for i in a:
...     print(chr(i ^ 172), end='')
... else:
...     print()
...
i_4m_t0p_cr4ck3r!
```

Nhập thử chuỗi “i\_4m\_t0p\_cr4ck3r!” vào phần password của chương trình, ta tìm được flag:

```
C:\Users\BILL\Desktop\Release_3\BabyVm>.\VirtualMachine.exe
V1r7u41 M4ch1n3 by <@shockbyte>
password->i_4m_t0p_cr4ck3r!
flag->SHB{p134E3_d0n7_r3v3r53_m3}
Press any key to continue . . .
```

Flag: SHB{p134E3\_d0n7\_r3v3r53\_m3}