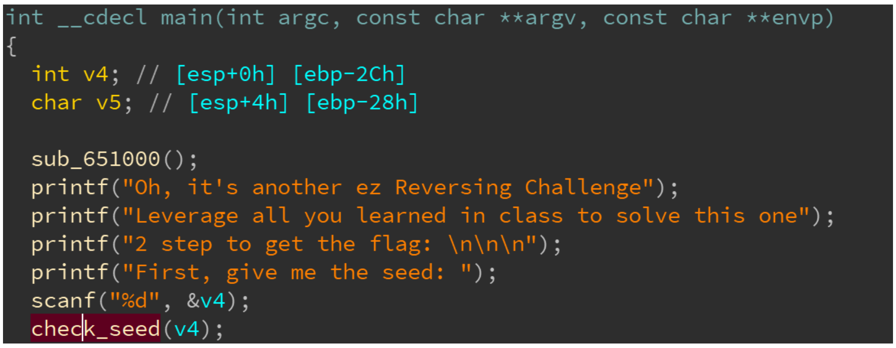
Computer Security Hw0x02 Writeup

- Realname: 胡安鳳
- ID on course web: alfons0329
- Student ID: R08922024

tags: `Computer Security` `NTU CS` `CS` `CTF` `Writeup`

Step 1, Trace the execution progress of seed

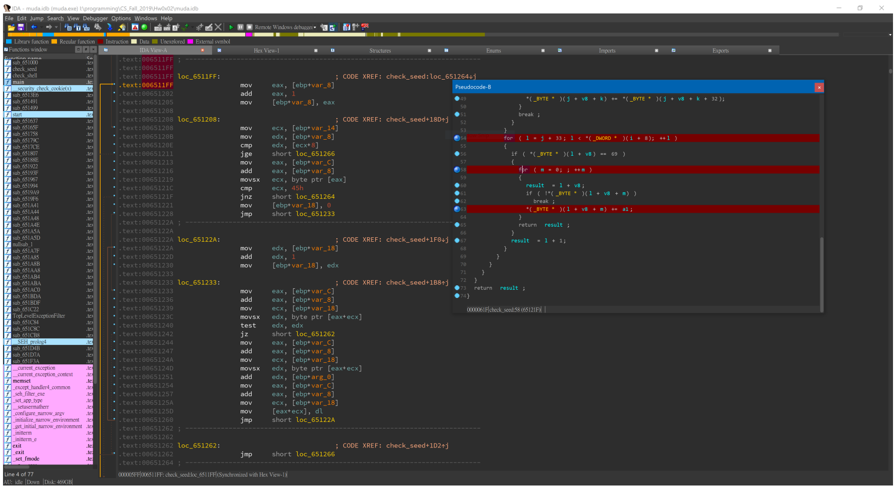
- Use IDA Pro to reverse and find out such function.



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [esp+0h] [ebp-2Ch]
  char v5; // [esp+4h] [ebp-28h]

  sub_651000();
  printf("Oh, it's another ez Reversing Challenge");
  printf("Leverage all you learned in class to solve this one");
  printf("2 step to get the flag: \n\n\n");
  printf("First, give me the seed: ");
  scanf("%d", &v4);
  check_seed(v4);
```

- Set 3 breakpoints in its function to see how the seed is being processed.

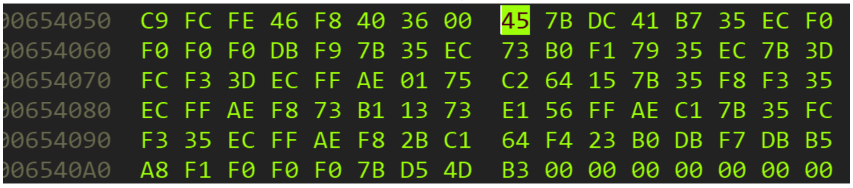


```
.text:006511FF ; ---------------------------------------------------------------------------
.text:006511FF
.text:006511FF loc_6511FF:                             ; CODE XREF: check_seed:loc_651264↓j
.text:006511FF                 mov     eax, [ebp+var_8]
.text:00651202                 add     eax, 1
.text:00651205                 mov     [ebp+var_8], eax
.text:00651208
.text:00651208 loc_651208:                             ; CODE XREF: check_seed+18D↑j
.text:00651208                 mov     ecx, [ebp+var_14]
.text:0065120B                 mov     edx, [ebp+var_8]
.text:0065120E                 cmp     edx, [ecx+8]
.text:00651211                 jge     short loc_651266
.text:00651213                 mov     eax, [ebp+var_C]
.text:00651216                 add     eax, [ebp+var_8]
.text:00651219                 movsx   ecx, byte ptr [eax]
.text:0065121C                 cmp     ecx, 45h
.text:0065121F                 jnz     short loc_651264
.text:00651221                 mov     [ebp+var_18], 0
.text:00651228                 jmp     short loc_651233
.text:0065122A ; ---------------------------------------------------------------------------
.text:0065122A
.text:0065122A loc_65122A:                             ; CODE XREF: check_seed+1F0↓j
.text:0065122A                 mov     edx, [ebp+var_18]
.text:0065122D                 add     edx, 1
.text:00651230                 mov     [ebp+var_18], edx
.text:00651233
.text:00651233 loc_651233:                             ; CODE XREF: check_seed+1B8↑j
.text:00651233                 mov     eax, [ebp+var_C]
.text:00651236                 add     eax, [ebp+var_8]
.text:00651239                 mov     ecx, [ebp+var_18]
.text:0065123C                 movsx   edx, byte ptr [eax+ecx]
.text:00651240                 test    edx, edx
.text:00651242                 jz      short loc_651262
.text:00651244                 mov     eax, [ebp+var_C]
.text:00651247                 add     eax, [ebp+var_8]
.text:0065124A                 mov     ecx, [ebp+var_18]
.text:0065124D                 movsx   edx, byte ptr [eax+ecx]
.text:00651251                 add     edx, [ebp+arg_0]
.text:00651254                 mov     eax, [ebp+var_C]
.text:00651257                 add     eax, [ebp+var_8]
.text:0065125A                 mov     ecx, [ebp+var_18]
.text:0065125D                 mov     [eax+ecx], dl
.text:00651260                 jmp     short loc_65122A
.text:00651262 ; ---------------------------------------------------------------------------
.text:00651262
.text:00651262 loc_651262:                             ; CODE XREF: check_seed+1D2↑j
.text:00651262                 jmp     short loc_651266
.text:00651264 ; ---------------------------------------------------------------------------
000005FF|006511FF: check_seed:loc_6511FF|(Synchronized with Hex View-1)
```

Pseudocode-B

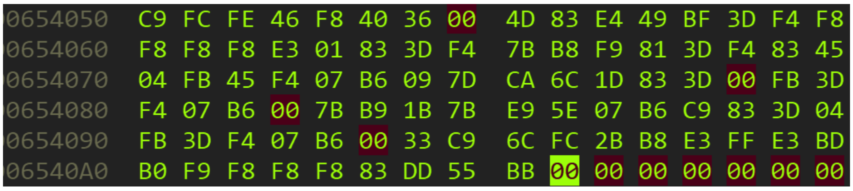
```
              *(_BYTE *)(j + v8 + k) += *(_BYTE *)(j + v8 + k + 32);
            }
            break;
          }
        }
        for ( l = j + 33; l < *(_DWORD *)(i + 8); ++l )
        {
          if ( *(_BYTE *)(l + v8) == 69 )
          {
            for ( m = 0; ; ++m )
            {
              result = l + v8;
              if ( !*(_BYTE *)(l + v8 + m) )
                break;
              *(_BYTE *)(l + v8 + m) += a1;
            }
            return result;
          }
          result = l + 1;
        }
      }
    }
  }
  return result;
}
0000061F|check_seed:58 (65121F)|
```

  - We can clearly see that it checks whether the data in memory is `69` or say `0x45`, and add the following 80 bytes with seed until it reaches NULL byte.
  - To verify the aforementioned property, compare the data from `unk_654058` before `seed = 8` has been inserted
    and the following 80 bytes, we can see that `unk_654058` has been add up from 45 to 4D and the same is true for all the following 80 bytes.



```
00654050  C9 FC FE 46 F8 40 36 00  45 7B DC 41 B7 35 EC F0
00654060  F0 F0 F0 DB F9 7B 35 EC  73 B0 F1 79 35 EC 7B 3D
00654070  FC F3 3D EC FF AE 01 75  C2 64 15 7B 35 F8 F3 35
00654080  EC FF AE F8 73 B1 13 73  E1 56 FF AE C1 7B 35 FC
00654090  F3 35 EC FF AE F8 2B C1  64 F4 23 B0 DB F7 DB B5
006540A0  A8 F1 F0 F0 F0 7B D5 4D  B3 00 00 00 00 00 00 00
```

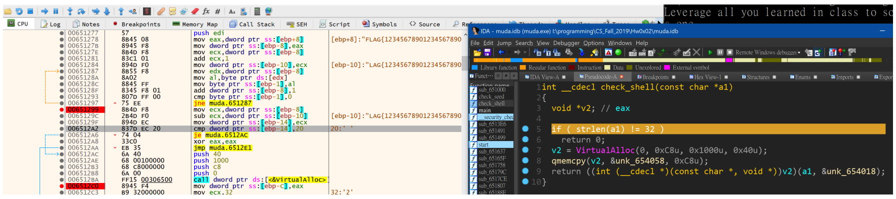
up before, down after



```
0654050  C9 FC FE 46 F8 40 36 00  4D 83 E4 49 BF 3D F4 F8
0654060  F8 F8 F8 E3 01 83 3D F4  7B B8 F9 81 3D F4 83 45
0654070  04 FB 45 F4 07 B6 09 7D  CA 6C 1D 83 3D 00 FB 3D
0654080  F4 07 B6 00 7B B9 1B 7B  E9 5E 07 B6 C9 83 3D 04
0654090  FB 3D F4 07 B6 00 33 C9  6C FC 2B B8 E3 FF E3 BD
06540A0  B0 F9 F8 F8 F8 83 DD 55  BB 00 00 00 00 00 00 00
```

Step 2, Choose the seed

- After the function for seed, we have the function for shellcode, located at `unk_654058` in the form just like HW0.



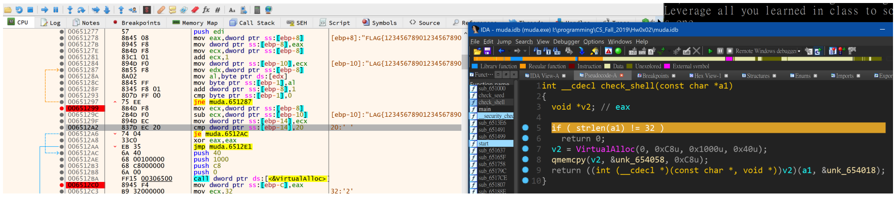
```
int __cdecl check_shell(const char *a1)
{
  void *v2; // eax

  if ( strlen(a1) != 32 )
    return 0;
  v2 = VirtualAlloc(0, 0xC8u, 0x1000u, 0x40u);
  qmemcpy(v2, &unk_654058, 0xC8u);
  return ((int (__cdecl *)(const char *, void *))v2)(a1, &unk_654018);
}
```

- For shellcode, it serves as a function, so we need `push ebp` first(i.e. the function prologue), with `opcode == 0x55` , therefore, seed 16 is suitable since `0x10 + 0x45 == 0x55`

Step 3, What is the flag?

- After the flag has been parsed from stdin, the `check_shell` function will first check whether the length of flag equals 32.



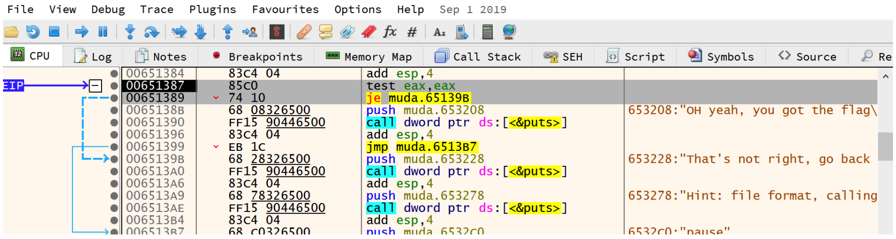
```
int __cdecl check_shell(const char *a1)
{
  void *v2; // eax

  if ( strlen(a1) != 32 )
    return 0;
  v2 = VirtualAlloc(0, 0xC8u, 0x1000u, 0x40u);
  qmemcpy(v2, &unk_654058, 0xC8u);
  return ((int (__cdecl *)(const char *, void *))v2)(a1, &unk_654018);
}
```

Left is the asm of checking `strlen(flag_in) == 32`

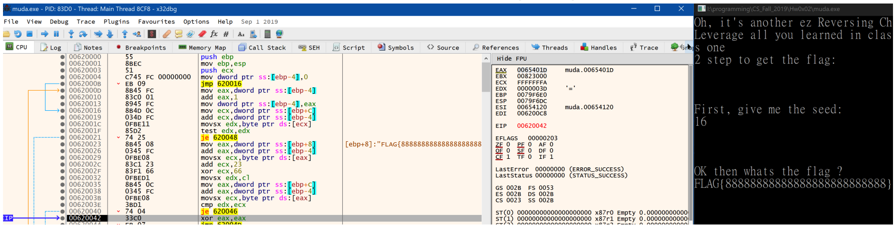
`cmp dword ptr ss:[ebp - 14], 0x20`

- Then we can trace where the flag lies. We see that `test eax eax` implies if `eax == 0` then program output `That is not right...`



```
00651384   83C4 04       add esp,4
00651387   85C0          test eax,eax
00651389   74 10         je muda.65139B
0065138B   68 08326500   push muda.653208            653208:"OH yeah, you got the flag\
00651390   FF15 90446500 call dword ptr ds:[<&puts>]
00651396   83C4 04       add esp,4
00651399   EB 1C         jmp muda.6513B7
0065139B   68 28326500   push muda.653228            653228:"That's not right, go back
006513A0   FF15 90446500 call dword ptr ds:[<&puts>]
006513A6   83C4 04       add esp,4
006513A9   68 78326500   push muda.653278            653278:"Hint: file format, calling
006513AE   FF15 90446500 call dword ptr ds:[<&puts>]
006513B4   83C4 04       add esp,4
006513B7   68 C0326500   push muda.6532C0            6532C0:"pause"
```

- Hence we **backtrace to find what makes eax 0**, turns out that the program process the input flag with the following: `input_flag = (input_flag + 0x23) ^ 0x66` . Then it loads the data where `eax` points to (in this picture it starts from address `0x654018` and now moves up to `0x65401D`). Finally if the comparison of `flag[i] != 0x654018 + i` , it will process `xor eax eax` to clear eax, making program fail.



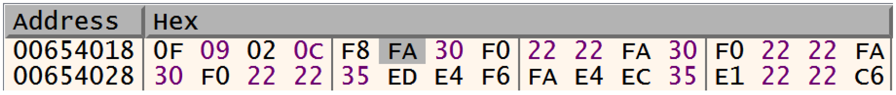
```
Oh, it's another ez Reversing Ch
Leverage all you learned in clas
s one
2 step to get the flag:


First, give me the seed:
16


OK then whats the flag ?
FLAG{8888888888888888888888888}
```

For example, `\x0F\x09\x02\x0C\xF8\xFA` is `(FLAG{y + 23) ^ 66` , and next character of my input is 8, on account of `('8' + 23) ^ 66` is `\x3D` , it fails.

- Dig out the data in `[0x654018, 0x654038)` , xor with 0x66 and minus 0x23 for the final flag.



```
Address  | Hex
00654018 | 0F 09 02 0C | F8 FA 30 F0 | 22 22 FA 30 | F0 22 22 FA
00654028 | 30 F0 22 22 | 35 ED E4 F6 | FA E4 EC 35 | E1 22 22 C6
```

FLAG{y3s!!y3s!!y3s!!0h_my_g0d!!}

Fake flag pitfall

- Note, if we directly find data lies in 0x654018 merely with IDA Pro, it will direct us for the wrong fake flag.
FL4G{Oh-yeah-U-G07-7h3-f4keflag}