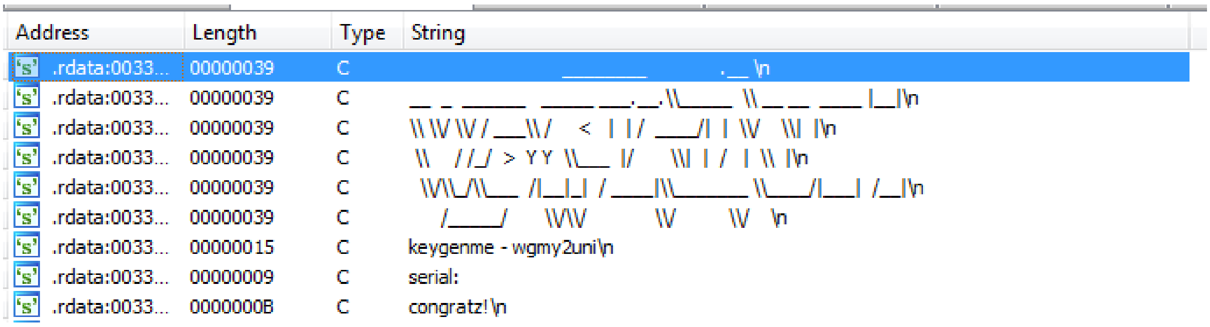
INITIAL ANALYSIS

TL; DR: the serial key is 19 characters with format of XXXX-XXXX-XXXX-XXXX and only uppercase alphabets are allowed.

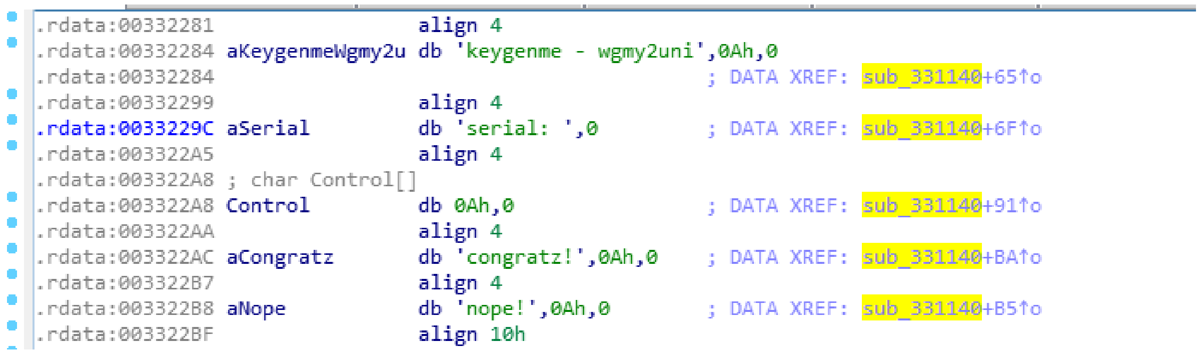
1. Load the executable in IDA
2. Search for strings (SHIFT+F2). Refer to Figure 1.



```
Address         Length     Type  String
.rdata:0033...  00000039   C          ___________          .__ \n
.rdata:0033...  00000039   C     __ _ ______  _____ ___.__.\\_____  \\__ __  ____ |__|\n
.rdata:0033...  00000039   C     \\ \/ \/ /___\\/   <  | |/ ____/| |  \/  \\|  |\n
.rdata:0033...  00000039   C      \\   / /_/ > Y Y \\___  |/     \\| |  /  |  \\ |\n
.rdata:0033...  00000039   C       \/\_/\___  /|__|_| / ____|\\_______  \\____/|___|  /__|\n
.rdata:0033...  00000039   C         /_____/      \/\/          \/         \/   \n
.rdata:0033...  00000015   C     keygenme - wgmy2uni\n
.rdata:0033...  00000009   C     serial:
.rdata:0033...  0000000B   C     congratz!\n
```

Figure 1: IDA strings function

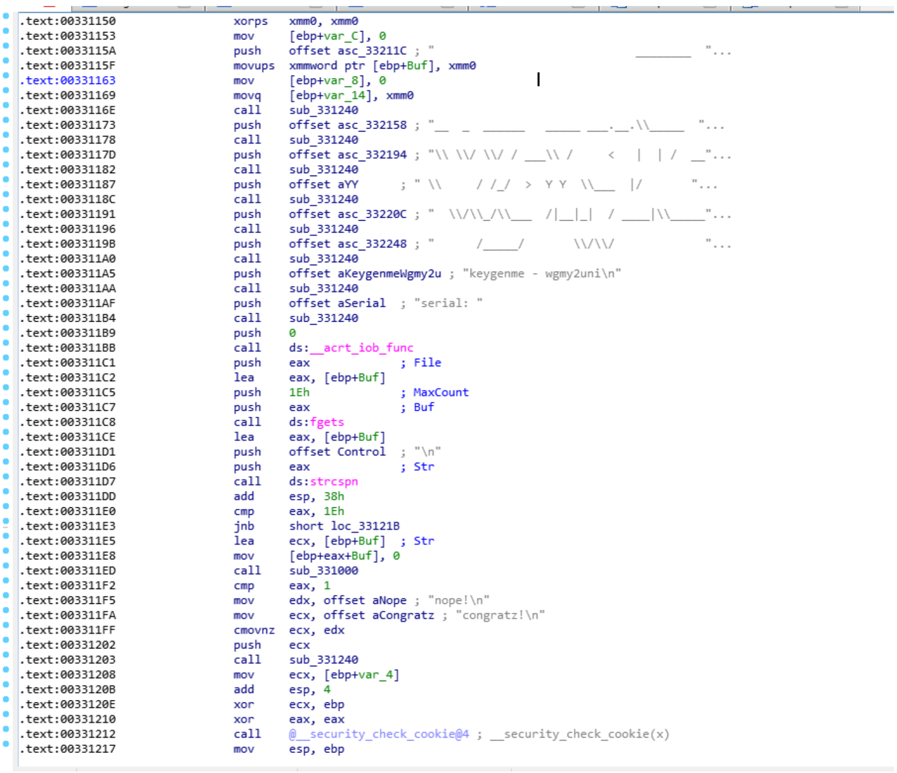
3. Double click in any interested string in Figure 1 and this will lead to data section of the executable. Refer Figure 2, click on the where the data is being called.



```
.rdata:00332281                     align 4
.rdata:00332284 aKeygenmeWgmy2u db 'keygenme - wgmy2uni',0Ah,0
.rdata:00332284                                         ; DATA XREF: sub_331140+65↑o
.rdata:00332299                     align 4
.rdata:0033229C aSerial         db 'serial: ',0         ; DATA XREF: sub_331140+6F↑o
.rdata:003322A5                     align 4
.rdata:003322A8 ; char Control[]
.rdata:003322A8 Control         db 0Ah,0                ; DATA XREF: sub_331140+91↑o
.rdata:003322AA                 align 4
.rdata:003322AC aCongratz       db 'congratz!',0Ah,0    ; DATA XREF: sub_331140+BA↑o
.rdata:003322B7                 align 4
.rdata:003322B8 aNope           db 'nope!',0Ah,0        ; DATA XREF: sub_331140+B5↑o
.rdata:003322BF                 align 10h
```

Figure 2 IDA data section

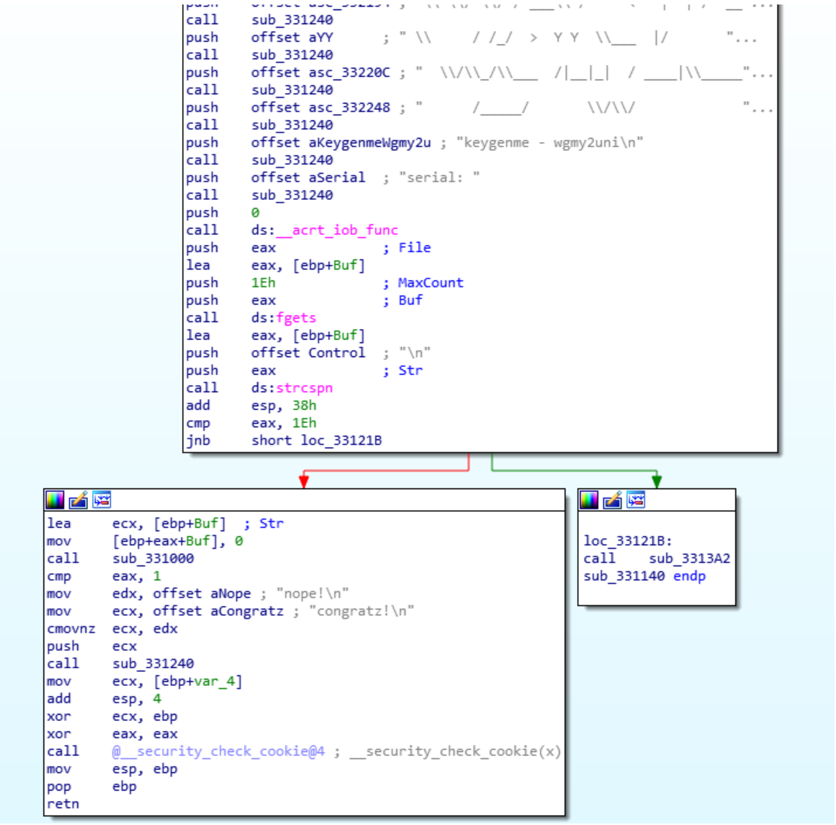
4. Figure 3 shows the main function which ask user to input the serial key. Figure 4 shows the graph view (SPACEBAR) on the function.
5. To understand some of the line, fgets and strcspn are C++ functions.



```
.text:00331150                 xorps   xmm0, xmm0
.text:00331153                 mov     [ebp+var_C], 0
.text:0033115A                 push    offset asc_33211C ; "                        ________    "...
.text:0033115F                 movups  xmmword ptr [ebp+Buf], xmm0
.text:00331163                 mov     [ebp+var_8], 0
.text:00331169                 movq    [ebp+var_14], xmm0
.text:0033116E                 call    sub_331240
.text:00331173                 push    offset asc_332158 ; "__  _  ______    _____ ___.__.\\_____   "...
.text:00331178                 call    sub_331240
.text:0033117D                 push    offset asc_332194 ; "\\ \\/ \\/ / __\\ /     <   |  | /  _"...
.text:00331182                 call    sub_331240
.text:00331187                 push    offset aYY      ; " \\     / /_/  >  Y Y  \\___  |/       "...
.text:0033118C                 call    sub_331240
.text:00331191                 push    offset asc_33220C ; "  \\/\\_/\\___  /|__|_|  / ____|\\_____"...
.text:00331196                 call    sub_331240
.text:0033119B                 push    offset asc_332248 ; "      /_____/        \\/\\/            "...
.text:003311A0                 call    sub_331240
.text:003311A5                 push    offset aKeygenmeWgmy2u ; "keygenme - wgmy2uni\n"
.text:003311AA                 call    sub_331240
.text:003311AF                 push    offset aSerial  ; "serial: "
.text:003311B4                 call    sub_331240
.text:003311B9                 push    0
.text:003311BB                 call    ds:__acrt_iob_func
.text:003311C1                 push    eax             ; File
.text:003311C2                 lea     eax, [ebp+Buf]
.text:003311C5                 push    1Eh             ; MaxCount
.text:003311C7                 push    eax             ; Buf
.text:003311C8                 call    ds:fgets
.text:003311CE                 lea     eax, [ebp+Buf]
.text:003311D1                 push    offset Control  ; "\n"
.text:003311D6                 push    eax             ; Str
.text:003311D7                 call    ds:strcspn
.text:003311DD                 add     esp, 38h
.text:003311E0                 cmp     eax, 1Eh
.text:003311E3                 jnb     short loc_33121B
.text:003311E5                 lea     ecx, [ebp+Buf]  ; Str
.text:003311E8                 mov     [ebp+eax+Buf], 0
.text:003311ED                 call    sub_331000
.text:003311F2                 cmp     eax, 1
.text:003311F5                 mov     edx, offset aNope ; "nope!\n"
.text:003311FA                 mov     ecx, offset aCongratz ; "congratz!\n"
.text:003311FF                 cmovnz  ecx, edx
.text:00331202                 push    ecx
.text:00331203                 call    sub_331240
.text:00331208                 mov     ecx, [ebp+var_4]
.text:0033120B                 add     esp, 4
.text:0033120E                 xor     ecx, ebp
.text:00331210                 xor     eax, eax
.text:00331212                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:00331217                 mov     esp, ebp
```

Figure 3 IDA the main function

6. Look at Figure 4, graph on bottom left. The **CMP eax, 1** and **CALL sub_331000.** This function calls another function. Double click on it to view the graph view of the called function.



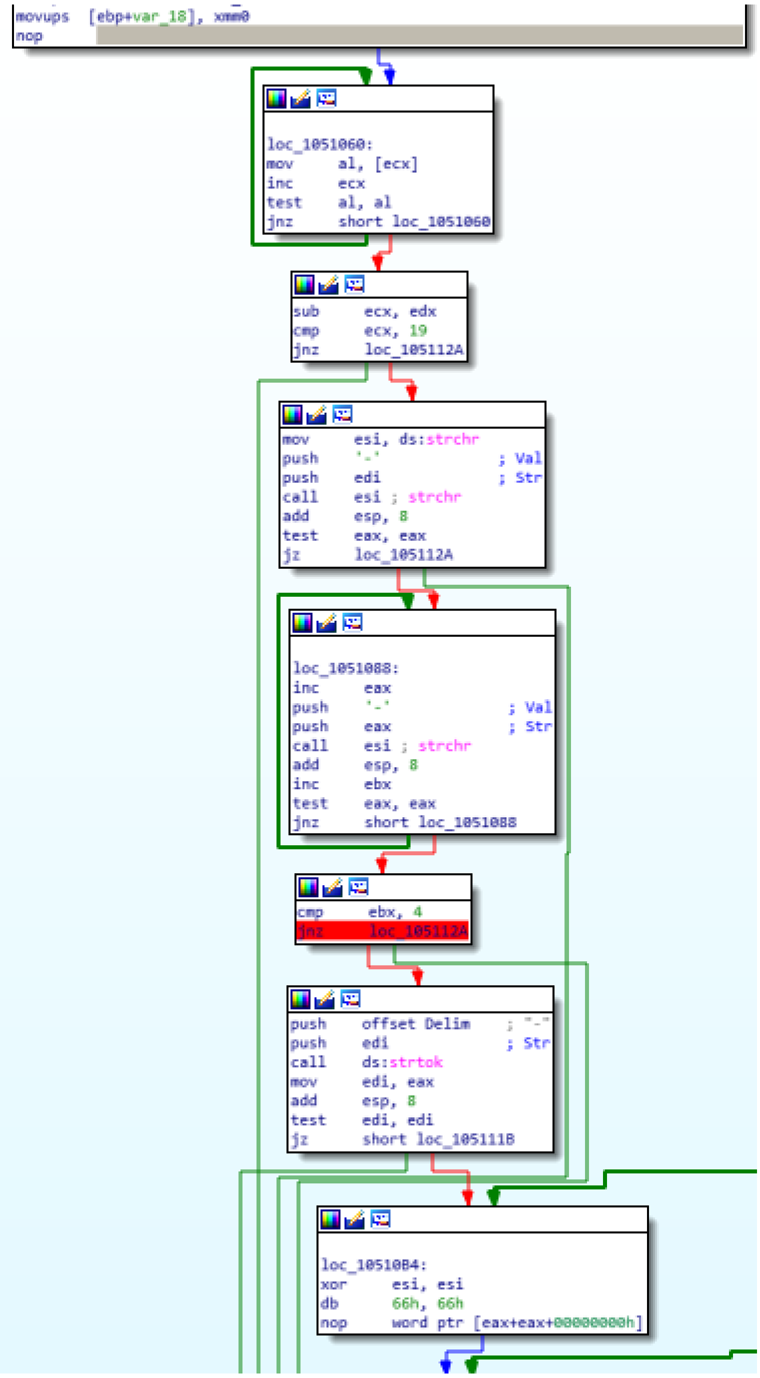
```
call    sub_331240
push    offset aYY      ; " \\     / /_/  >  Y Y  \\___  |/       "...
call    sub_331240
push    offset asc_33220C ; "  \\/\\_/\\___  /|__|_|  / ____|\\_____"...
call    sub_331240
push    offset asc_332248 ; "      /_____/        \\/\\/            "...
call    sub_331240
push    offset aKeygenmeWgmy2u ; "keygenme - wgmy2uni\n"
call    sub_331240
push    offset aSerial  ; "serial: "
call    sub_331240
push    0
call    ds:__acrt_iob_func
push    eax             ; File
lea     eax, [ebp+Buf]
push    1Eh             ; MaxCount
push    eax             ; Buf
call    ds:fgets
lea     eax, [ebp+Buf]
push    offset Control  ; "\n"
push    eax             ; Str
call    ds:strcspn
add     esp, 38h
cmp     eax, 1Eh
jnb     short loc_33121B
```

```
lea     ecx, [ebp+Buf]  ; Str
mov     [ebp+eax+Buf], 0
call    sub_331000
cmp     eax, 1
mov     edx, offset aNope ; "nope!\n"
mov     ecx, offset aCongratz ; "congratz!\n"
cmovnz  ecx, edx
push    ecx
call    sub_331240
mov     ecx, [ebp+var_4]
add     esp, 4
xor     ecx, ebp
xor     eax, eax
call    @__security_check_cookie@4 ; __security_check_cookie(x)
mov     esp, ebp
pop     ebp
retn
```

```
loc_33121B:
call    sub_3313A2
sub_331140 endp
```

Figure 4 IDA graph view on main function

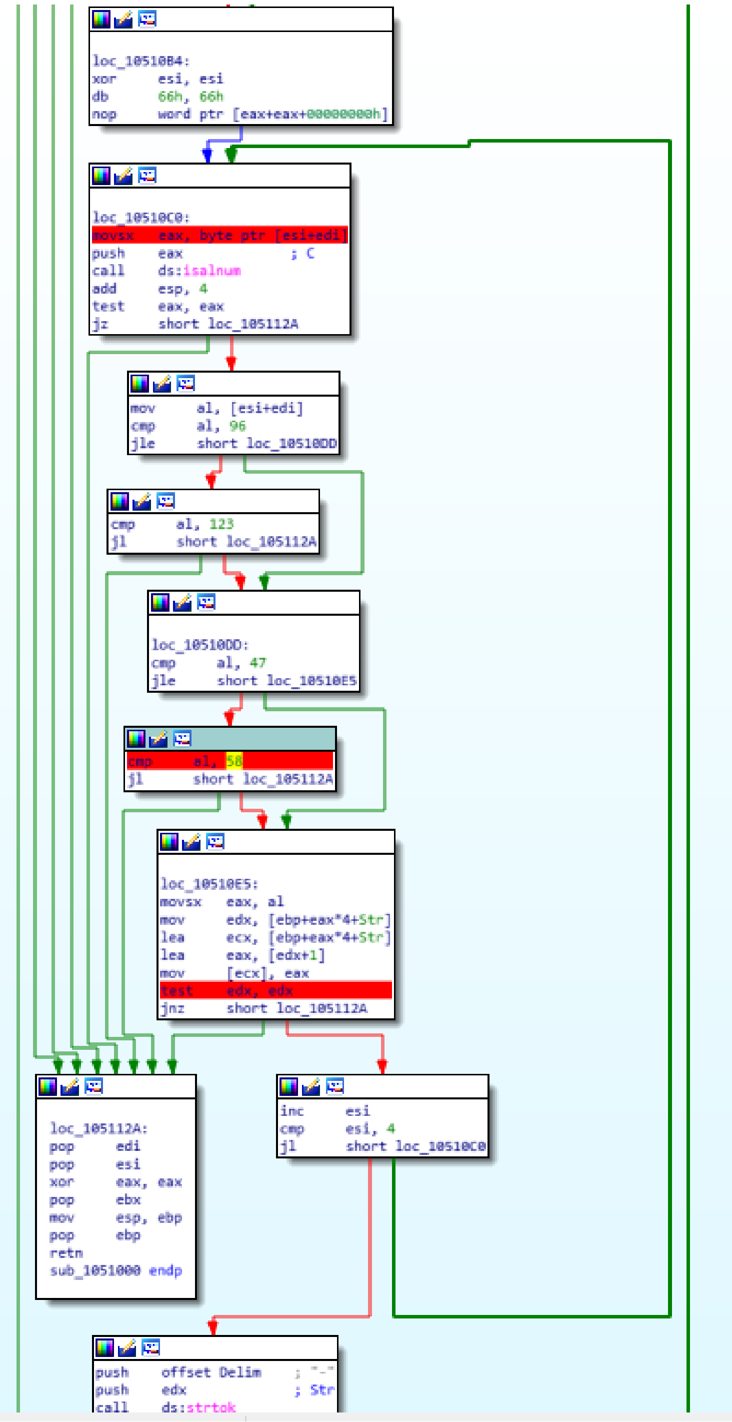
7. Refer to Figure 5, the **loc_331060**. It keeps on looping until the **al** register is 0 while increasing **ecx** register value. Once **al** is 0, it compares the **ecx** register if it is equal to 19 in decimal. Set a breakpoint and execute line by line show the program is comparing the serial key length (19 characters). The 3rd to 6th box is where the program check for serial key format (ABCD-EFGH-1234-5678)



```
movups  [ebp+var_18], xmm0
nop

loc_1051060:
mov     al, [ecx]
inc     ecx
test    al, al
jnz     short loc_1051060

sub     ecx, edx
cmp     ecx, 19
jnz     loc_105112A

mov     esi, ds:strchr
push    '-'                     ; Val
push    edi                     ; Str
call    esi ; strchr
add     esp, 8
test    eax, eax
jz      loc_105112A

loc_1051088:
inc     eax
push    '-'                     ; Val
push    eax                     ; Str
call    esi ; strchr
add     esp, 8
inc     ebx
test    eax, eax
jnz     short loc_1051088

cmp     ebx, 4
jnz     loc_105112A

push    offset Delim            ; "-"
push    edi                     ; Str
call    ds:strtok
mov     edi, eax
add     esp, 8
test    edi, edi
jz      short loc_105111B

loc_10510B4:
xor     esi, esi
db      66h, 66h
nop     word ptr [eax+eax+00000000h]
```

Figure 5 IDA graph view sub_331000 function. Pt.1

8. Refer to Figure 6, the 2nd box is to check each character in serial key is either digits or alphabets. The 3rd to 6th box is to check if the character is outside range of 96-123 (lowercase alphabets) and 47-58 (all the digits). The number range is the integer value of alphabets, e.g. A in integer is 65. Using python to see what character is allowed for the serial key, refer to Figure 7. Figure 8 shows the code in high-level language.



```
loc_1051084:
xor     esi, esi
db      66h, 66h
nop     word ptr [eax+eax+00000000h]
```

```
loc_10510C0:
movsx   eax, byte ptr [esi+edi]
push    eax             ; C
call    ds:isalnum
add     esp, 4
test    eax, eax
jz      short loc_105112A
```

```
mov     al, [esi+edi]
cmp     al, 96
jle     short loc_10510DD
```

```
cmp     al, 123
jl      short loc_105112A
```

```
loc_10510DD:
cmp     al, 47
jle     short loc_10510E5
```

```
cmp     al, 58
jl      short loc_105112A
```

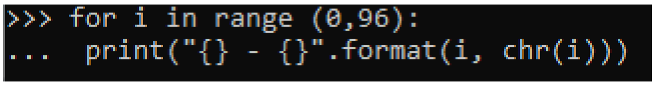
```
loc_10510E5:
movsx   eax, al
mov     edx, [ebp+eax*4+Str]
lea     ecx, [ebp+eax*4+Str]
lea     eax, [edx+1]
mov     [ecx], eax
test    edx, edx
jnz     short loc_105112A
```

```
loc_105112A:
pop     edi
pop     esi
xor     eax, eax
pop     ebx
mov     esp, ebp
pop     ebp
retn
sub_1051000 endp
```

```
inc     esi
cmp     esi, 4
jl      short loc_10510C0
```

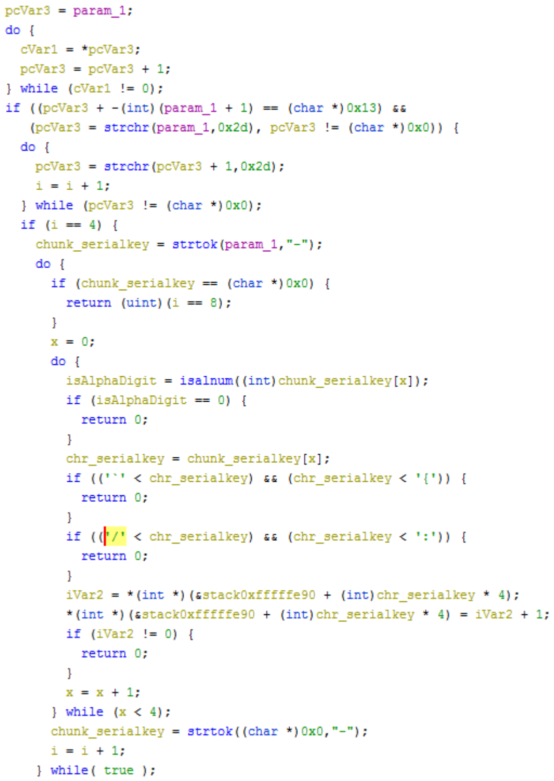
```
push    offset Delim    ; "-"
push    edx             ; Str
call    ds:strtok
```

Figure 6 IDA graph view sub_331000 function. Pt.2



```
>>> for i in range (0,96):
...   print("{} - {}".format(i, chr(i)))
```

Figure 7 python to view allowed characters



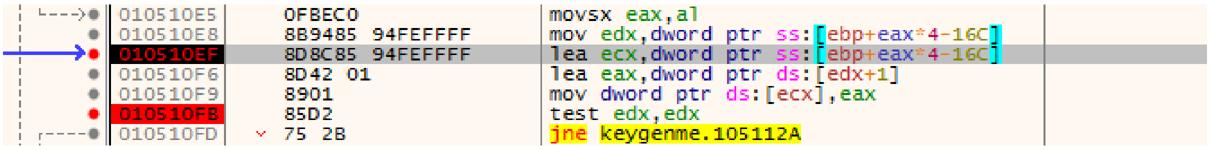
```
pcVar3 = param_1;
do {
  cVar1 = *pcVar3;
  pcVar3 = pcVar3 + 1;
} while (cVar1 != 0);
if ((pcVar3 + -(int)(param_1 + 1) == (char *)0x13) &&
   (pcVar3 = strchr(param_1,0x2d), pcVar3 != (char *)0x0)) {
  do {
    pcVar3 = strchr(pcVar3 + 1,0x2d);
    i = i + 1;
  } while (pcVar3 != (char *)0x0);
  if (i == 4) {
    chunk_serialkey = strtok(param_1,"-");
    do {
      if (chunk_serialkey == (char *)0x0) {
        return (uint)(i == 8);
      }
      x = 0;
      do {
        isAlphaDigit = isalnum((int)chunk_serialkey[x]);
        if (isAlphaDigit == 0) {
          return 0;
        }
        chr_serialkey = chunk_serialkey[x];
        if (('`' < chr_serialkey) && (chr_serialkey < '{')) {
          return 0;
        }
        if (('/' < chr_serialkey) && (chr_serialkey < ':')) {
          return 0;
        }
        iVar2 = *(int *)(&stack0xfffffe90 + (int)chr_serialkey * 4);
        *(int *)(&stack0xfffffe90 + (int)chr_serialkey * 4) = iVar2 + 1;
        if (iVar2 != 0) {
          return 0;
        }
        x = x + 1;
      } while (x < 4);
      chunk_serialkey = strtok((char *)0x0,"-");
      i = i + 1;
    } while( true );
```

Figure 8 GHIDRA decompile function

FINDING THE SERIAL KEY

TL; DR: brute forcing manually. Flag is **FLPO-ZKJN-XBCD-QIVH**

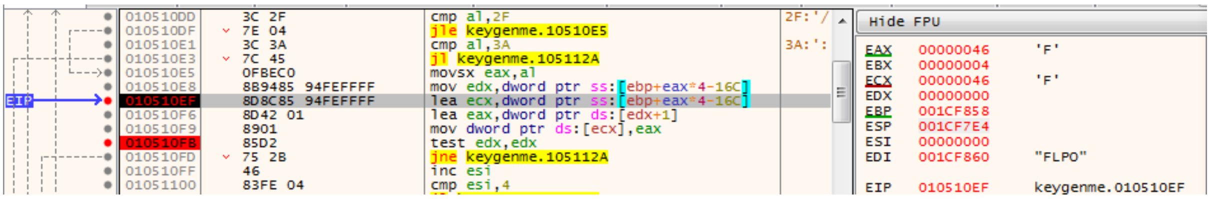
1. After running the program countless time looking for how to program validate the key and where the key is stored, the answer is at the Figure 9.



```
010510E5   0FBEC0          movsx eax,al
010510E8   8B9485 94FEFFFF mov edx,dword ptr ss:[ebp+eax*4-16C]
010510EF   8D8C85 94FEFFFF lea ecx,dword ptr ss:[ebp+eax*4-16C]
010510F6   8D42 01         lea eax,dword ptr ds:[edx+1]
010510F9   8901            mov dword ptr ds:[ecx],eax
010510FB   85D2            test edx,edx
010510FD   75 2B           jne keygenme.105112A
```

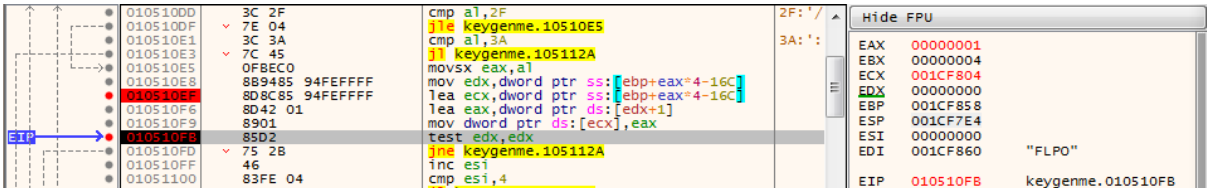
Figure 9 Validating the key

2. Brute force the key, **edx** register must equal to 0. **mov edx, dword ptr ss:[ebp+eax*4-16c]**, moving the value from memory pointer based on the calculation. Refer to Figure 10, where **EAX** = 46, **EBP** = 1CF858, and the pointer is 0x1cf804. Hence, with the correct character, the **EDX** register value is equal to 0 as shown in Figure 11.



```
     010510DD   3C 2F           cmp al,2F                              2F:'/'
     010510DF   7E 04           jle keygenme.10510E5
     010510E1   3C 3A           cmp al,3A                              3A:':'
     010510E3   7C 45           jl keygenme.105112A
     010510E5   0FBEC0          movsx eax,al
     010510E8   8B9485 94FEFFFF mov edx,dword ptr ss:[ebp+eax*4-16C]
EIP  010510EF   8D8C85 94FEFFFF lea ecx,dword ptr ss:[ebp+eax*4-16C]
     010510F6   8D42 01         lea eax,dword ptr ds:[edx+1]
     010510F9   8901            mov dword ptr ds:[ecx],eax
     010510FB   85D2            test edx,edx
     010510FD   75 2B           jne keygenme.105112A
     010510FF   46              inc esi
     01051100   83FE 04         cmp esi,4

Hide FPU

EAX   00000046   'F'
EBX   00000004
ECX   00000046   'F'
EDX   00000000
EBP   001CF858
ESP   001CF7E4
ESI   00000000
EDI   001CF860   "FLPO"

EIP   010510EF   keygenme.010510EF
```

Figure 10 x32dbg - checking correct char

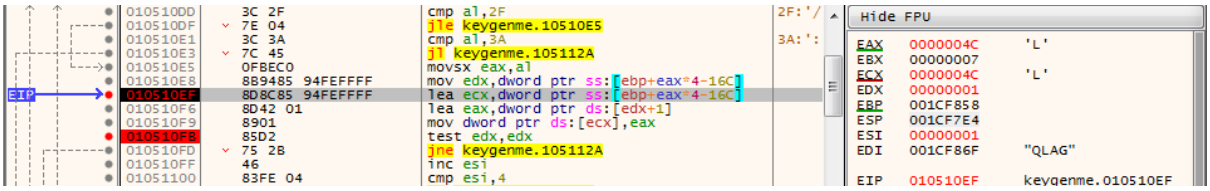


```
010510DD    3C 2F            cmp al,2F                          2F:'/'
010510DF    7E 04            jle keygenme.10510E5
010510E1    3C 3A            cmp al,3A                          3A:':'
010510E3    7C 45            jl keygenme.105112A
010510E5    0FBEC0           movsx eax,al
010510E8    8B9485 94FEFFFF  mov edx,dword ptr ss:[ebp+eax*4-16C]
010510EF    8D8C85 94FEFFFF  lea ecx,dword ptr ss:[ebp+eax*4-16C]
010510F6    8D42 01          lea eax,dword ptr ds:[edx+1]
010510F9    8901             mov dword ptr ds:[ecx],eax
010510FB    85D2             test edx,edx
010510FD    75 2B            jne keygenme.105112A
010510FF    46               inc esi
01051100    83FE 04          cmp esi,4
```

```
Hide FPU

EAX   00000001
EBX   00000004
ECX   001CF804
EDX   00000000
EBP   001CF858
ESP   001CF7E4
ESI   00000000
EDI   001CF860      "FLPO"

EIP   010510FB      keygenme.010510FB
```

Figure 11 x32dbg - EDX register is equal to 0 after checking char 'F'

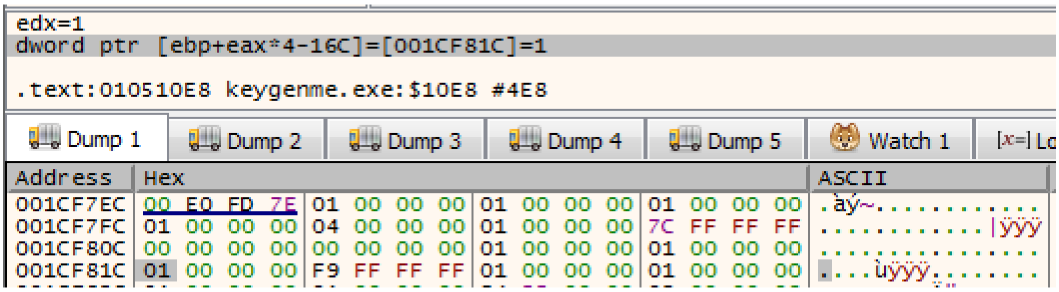
3. FLPO-ZKJN-XBCD-QLAG is the serial key, where the green zone is the correct section. Figure 12 shows that char ‘L’ is incorrect because of **EDX** register is not equal to 0. By calculation, edx register get the value from 0x1CF81C, refer to Figure 13.



```
010510DD    3C 2F            cmp al,2F                          2F:'/'
010510DF    7E 04            jle keygenme.10510E5
010510E1    3C 3A            cmp al,3A                          3A:':'
010510E3    7C 45            jl keygenme.105112A
010510E5    0FBEC0           movsx eax,al
010510E8    8B9485 94FEFFFF  mov edx,dword ptr ss:[ebp+eax*4-16C]
010510EF    8D8C85 94FEFFFF  lea ecx,dword ptr ss:[ebp+eax*4-16C]
010510F6    8D42 01          lea eax,dword ptr ds:[edx+1]
010510F9    8901             mov dword ptr ds:[ecx],eax
010510FB    85D2             test edx,edx
010510FD    75 2B            jne keygenme.105112A
010510FF    46               inc esi
01051100    83FE 04          cmp esi,4
```

```
Hide FPU

EAX   0000004C      'L'
EBX   00000007
ECX   0000004C      'L'
EDX   00000001
EBP   001CF858
ESP   001CF7E4
ESI   00000001
EDI   001CF86F      "QLAG"

EIP   010510EF      keygenme.010510EF
```

Figure 12 x32dbg - checking incorrect char

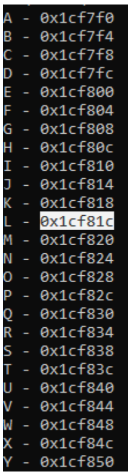


```
edx=1
dword ptr [ebp+eax*4-16C]=[001CF81C]=1

.text:010510E8 keygenme.exe:$10E8 #4E8
```

```
Address   Hex                                              ASCII
001CF7EC  00 E0 FD 7E 01 00 00 00 01 00 00 00 01 00 00 00  .àý~............
001CF7FC  01 00 00 00 04 00 00 00 01 00 00 00 7C FF FF FF  ............|ÿÿÿ
001CF80C  00 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00  ................
001CF81C  01 00 00 00 F9 FF FF FF 01 00 00 00 01 00 00 00  ....ùÿÿÿ........
```

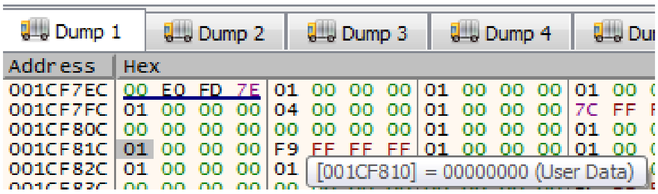
Figure 13 x32dbg - memory pointer

4. To make the guessing is bit easier, using the EBP value can calculate it with all the alphabet and using the result (refer to Figure 14), manually check it at the memory (refer to Figure 15) which shows that the correct character is ‘I’ at 0x1CF810.



```
A - 0x1cf7f0
B - 0x1cf7f4
C - 0x1cf7f8
D - 0x1cf7fc
E - 0x1cf800
F - 0x1cf804
G - 0x1cf808
H - 0x1cf80c
I - 0x1cf810
J - 0x1cf814
K - 0x1cf818
L - 0x1cf81c
M - 0x1cf820
N - 0x1cf824
O - 0x1cf828
P - 0x1cf82c
Q - 0x1cf830
R - 0x1cf834
S - 0x1cf838
T - 0x1cf83c
U - 0x1cf840
V - 0x1cf844
W - 0x1cf848
X - 0x1cf84c
Y - 0x1cf850
```

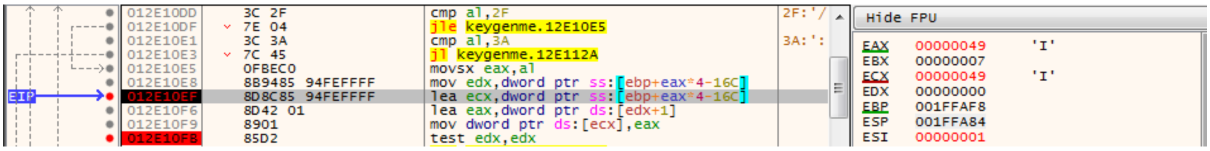
Figure 14 calculated address



```
Address   Hex
001CF7EC  00 E0 FD 7E  01 00 00 00  01 00 00 00  01 00
001CF7FC  01 00 00 00  04 00 00 00  01 00 00 00  7C FF
001CF80C  00 00 00 00  00 00 00 00  01 00 00 00  01 00
001CF81C  01 00 00 00  F9 FF FF FF  01 00 00 00  01 00
001CF82C  01 00 00 00  01 [001CF810] = 00000000 (User Data)
```

Figure 15 value in memory address

5. FLPO-ZKJN-XBCD-QIAG is the serial key. After checking at character ‘I’, notice that the EDX register value is equal to 0. Hence, it’s correct.



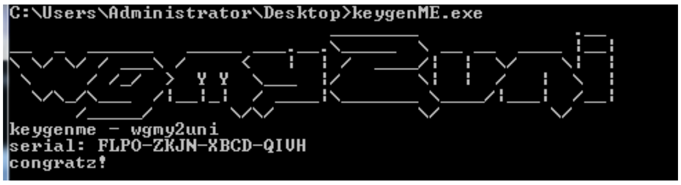
```
    012E10DD   3C 2F            cmp al,2F                          2F:'/'
    012E10DF   7E 04            jle keygenme.12E10E5
    012E10E1   3C 3A            cmp al,3A                          3A:':'
    012E10E3   7C 45            jl keygenme.12E112A
    012E10E5   0FBEC0           movsx eax,al
    012E10E8   8B9485 94FEFFFF  mov edx,dword ptr ss:[ebp+eax*4-16C]
EIP 012E10EF   8D8C85 94FEFFFF  lea ecx,dword ptr ss:[ebp+eax*4-16C]
    012E10F6   8D42 01          lea eax,dword ptr ds:[edx+1]
    012E10F9   8901             mov dword ptr ds:[ecx],eax
    012E10FB   85D2             test edx,edx
```

```
Hide FPU

EAX   00000049   'I'
EBX   00000007
ECX   00000049   'I'
EDX   00000000
EBP   001FFAF8
ESP   001FFA84
ESI   00000001
```

Figure 16 x32dbg - char I is correct

CONCLUSION

FLAG: **FLPO-ZKJN-XBCD-QIVH**



```
C:\Users\Administrator\Desktop>keygenME.exe

keygenme - wgmy2uni
serial: FLPO-ZKJN-XBCD-QIVH
congratz!
```

I find this challenge is difficult because of lack of knowledge and experience. I took almost 2 days to complete it. I feel like there is an easy way to get the serial key without manually brute force it. Below are the tools used in this challenge,

- VMware
- FlareVM
- IDAfree70
- x32dbg
- google
- Ghidra.