


**Author: Muhammad Aiman bin Shamshuri (Poji)**

**Universiti Teknologi MARA**

## **WGMY Fortnite Challenge #0402 WriteUp**

--- #0402 Keygenme We need the key badly but too bad, the server is no longer up. Can you help us to get the key? Even happier of you can give us a keygen. ---

```
PS G:\WGMY STUDENT> .\keygenME.exe
```

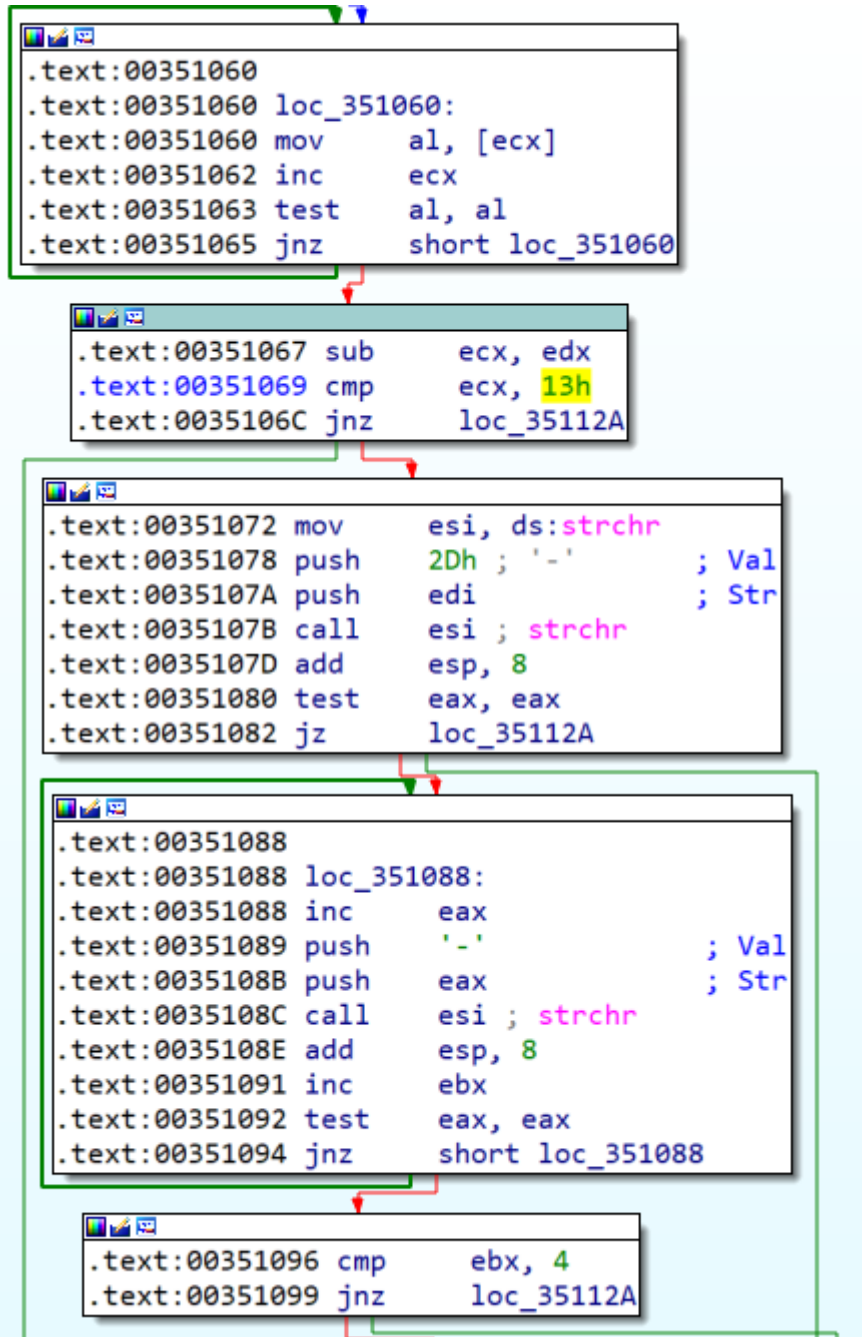


```
keygenme - wgmy2uni
serial:
```

In the main function we see a function named sub\_351000

```
.text:003511E5 lea     ecx, [ebp+Buf] ; Str
.text:003511E8 mov     [ebp+eax+Buf], 0
.text:003511ED call    sub_351000
.text:003511F2 cmp     eax, 1
.text:003511F5 mov     edx, offset aNope ; "nope!\n"
.text:003511FA mov     ecx, offset aCongratz ; "congratz!\n"
.text:003511FF cmovnz  ecx, edx
.text:00351202 push    ecx
.text:00351203 call    sub_351240
.text:00351208 mov     ecx, [ebp+var_4]
.text:0035120B add     esp, 4
.text:0035120E xor     ecx, ebp
.text:00351210 xor     eax, eax
.text:00351212 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:00351217 mov     esp, ebp
.text:00351219 pop     ebp
.text:0035121A retn
```

Let us see:



```

.text:0035109F push    offset Delim    ; "" - ""
.text:003510A4 push    edi                ; Str
.text:003510A5 call     ds:strtok
.text:003510AB mov     edi, eax
.text:003510AD add     esp, 8
.text:003510B0 test    edi, edi
.text:003510B2 jz      short loc_35111B

```

```

.text:003510B4
.text:003510B4 loc_3510B4:
.text:003510B4 xor     esi, esi
.text:003510B6 db      66h, 66h
.text:003510B6 nop     word ptr [eax+eax+00000000h]

```

```

.text:003510C0
.text:003510C0 loc_3510C0:
.text:003510C0 movsx   eax, byte ptr [esi+edi]
.text:003510C4 push    eax                ; C
.text:003510C5 call     ds:isalnum
.text:003510CB add     esp, 4
.text:003510CE test    eax, eax
.text:003510D0 jz      short loc_35112A

```

```

.text:003510D2 mov     al, [esi+edi]
.text:003510D5 cmp     al, 60h ; ''
.text:003510D7 jle     short loc_3510DD

```

```

.text:003510D9 cmp     al, 7Bh ; '{'
.text:003510DB jl      short loc_35112A

```

```

.text:003510DD
.text:003510DD loc_3510DD:
.text:003510DD cmp     al, 2Fh ; '/'
.text:003510DF jle     short loc_3510E5

```

```

.text:003510E1 cmp     al, 3Ah ; ':'
.text:003510E3 jl      short loc_35112A

```

```

.text:003510E5
.text:003510E5 loc_3510E5:
.text:003510E5 movsx   eax, al
.text:003510E8 mov     edx, [ebp+eax*4+Str]
.text:003510EF lea     ecx, [ebp+eax*4+Str]
.text:003510F6 lea     eax, [edx+1]
.text:003510F9 mov     [ecx], eax
.text:003510FB test    edx, edx
.text:003510FD jnz     short loc_35112A

```

```

.text:0035112A
.text:0035112A loc_35112A:
.text:0035112A pop     edi
.text:0035112B pop     esi
.text:0035112C xor     eax, eax
.text:0035112E pop     ebx
.text:0035112F mov     esp, ebp
.text:00351131 pop     ebp
.text:00351132 retn

```

```

.text:003510FF inc     esi
.text:00351100 cmp     esi, 4
.text:00351103 jl      short loc_3510C0

```

Using the magic key (f5) in ida, we get the source code for this function.

```
if ( strlen(Str) == 19 )
{
    v3 = strchr(v2, 45);
    if ( v3 )
    {
        do
        {
            v3 = strchr(v3 + 1, '-');
            ++v1;
        }
        while ( v3 );
        if ( v1 == 4 )
        {
            v4 = strtok(v2, "-");
            if ( !v4 )
                return v1 == 8;
        }
    }
}
```

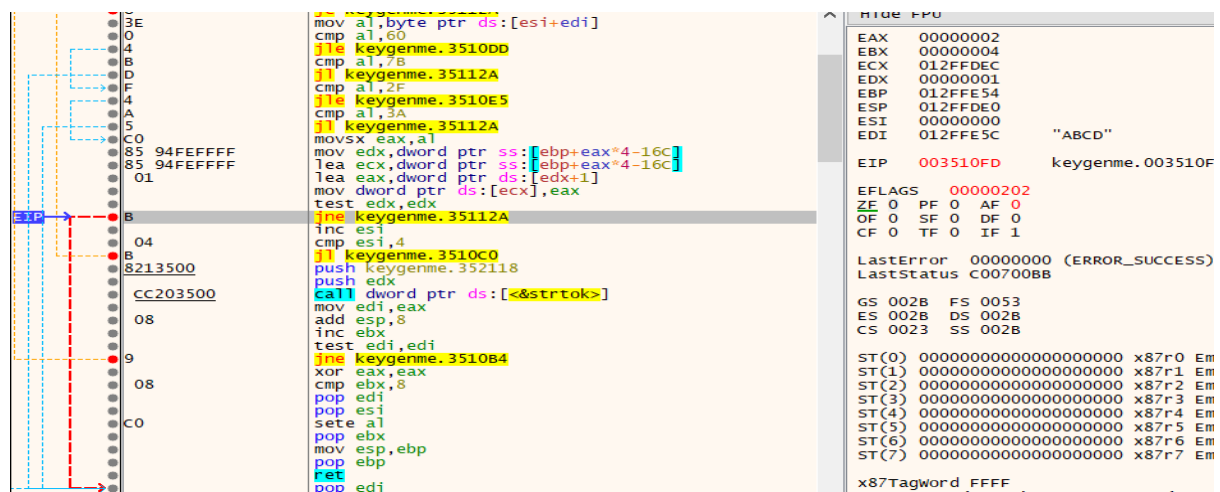
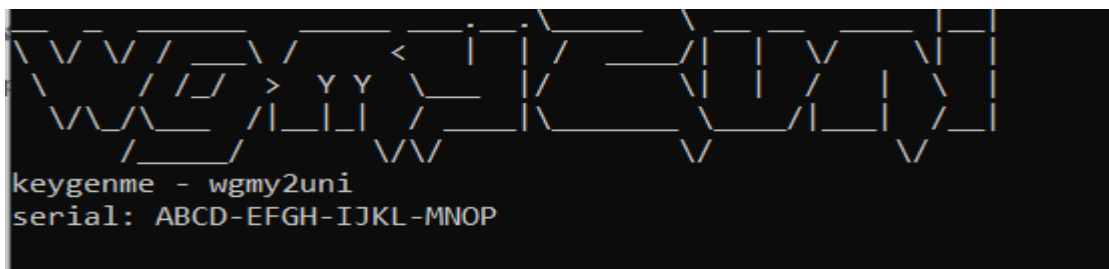
```
LABEL_6:
    v5 = 0;
    while ( 1 )
    {
        v10 = v4[v5];
        if ( !isalnum(v10) )
            break;
        v6 = v4[v5];
        if ( v6 > '`' && v6 < '{' )
            break;
        if ( v6 > '/' && v6 < ':' )
            break;
        v7 = Stra[v6];
        Stra[v6] = v7 + 1;
        if ( v7 )
            break;
        if ( ++v5 >= 4 )
        {
            v4 = strtok(0, "-");
            ++v1;
            if ( v4 )
                goto LABEL_6;
            return v1 == 8;
        }
    }
```

From the source code we can see there is a check for string length == 19. Then it splits our input with “-” as a delimiter and checks whether there are 4 parts.

Also it breaks when it finds a digit or lowercase character. So we can only use uppercase and we cannot use the same character again. It increases the byte at that location in that string by one once we use a character. For example, if we input A = 0x41 then Stra[0x41] is checked and later incremented. It compares the characters of our input with a string taking our character as index. If at that location 0 exists .. Then it is fine.

So to check which character is passed, I brute force each character and use x64dbg to see the values.

I try to brute force the input from A-Z.



We can see that if we input “A”, it will break which means that A is not a valid character. So I keep trying for other characters.



After brute forcing the input, I get which characters that are valid.



```
keygenme - wgmy2uni
serial: BCDF-HIJK-LOPQ-TUVX
congratz!
```

VOILA!!