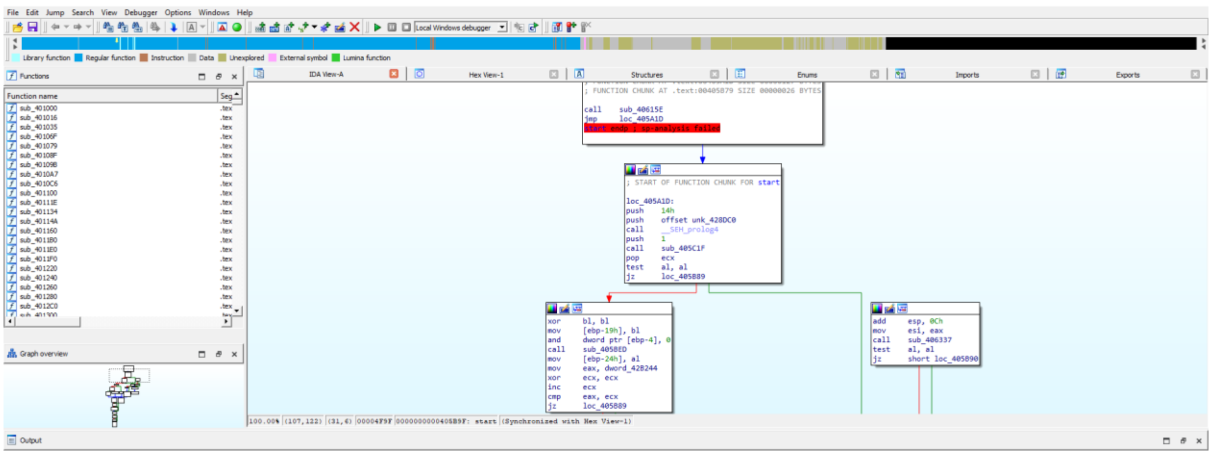
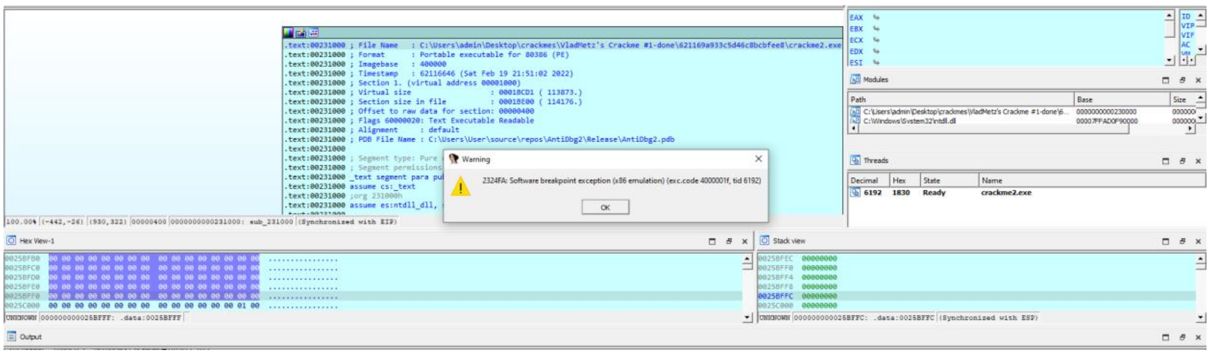
In this write up I will be using IDA Freeware 7.7 to solve this crackme.

Let's open this crackme in IDA and see what we have.



```
; FUNCTION CHUNK AT .text:00405B79 SIZE 00000026 BYTES

call    sub_40615E
jmp     loc_405A1D
start endp ; sp-analysis failed

; START OF FUNCTION CHUNK FOR start

loc_405A1D:
push    14h
push    offset unk_428DC0
call    __SEH_prolog4
push    1
call    sub_405C1F
pop     ecx
test    al, al
jz      loc_405B89

xor     bl, bl
mov     [ebp-19h], bl
and     dword ptr [ebp-4], 0
call    sub_4058ED
mov     [ebp-24h], al
mov     eax, dword_42B244
xor     ecx, ecx
inc     ecx
cmp     eax, ecx
jz      loc_405B89

add     esp, 0Ch
mov     esi, eax
call    sub_406337
test    al, al
jz      short loc_405B90
```

When tried to run the crackme in IDA I get this warning:



```
.text:00231000 ; File Name   : C:\Users\admin\Desktop\crackmes\VladMetz's Crackme #1-done\621169a933c5d46c8bcbfee8\crackme2.exe
.text:00231000 ; Format      : Portable executable for 80386 (PE)
.text:00231000 ; Imagebase   : 400000
.text:00231000 ; Timestamp   : 62116646 (Sat Feb 19 21:51:02 2022)
.text:00231000 ; Section 1. (virtual address 00001000)
.text:00231000 ; Virtual size                  : 0001BCD1 ( 113873.)
.text:00231000 ; Section size in file          : 00018E00 ( 114176.)
.text:00231000 ; Offset to raw data for section: 00000400
.text:00231000 ; Flags 60000020: Text Executable Readable
.text:00231000 ; Alignment     : default
.text:00231000 ; PDB File Name : C:\Users\User\source\repos\AntiDbg2\Release\AntiDbg2.pdb
.text:00231000
.text:00231000 ; Segment type: Pure
.text:00231000 ; Segment permissions
.text:00231000 _text segment para pu
.text:00231000 assume cs:_text
.text:00231000 ;org 231000h
.text:00231000 assume es:ntdll_dll,
```

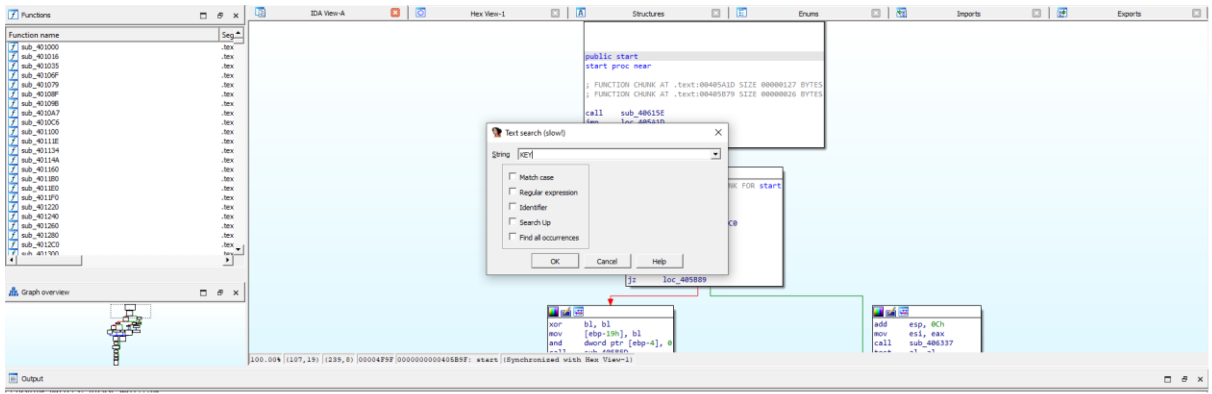
Warning

2324FA: Software breakpoint exception (x86 emulation) (exc.code 4000001f, tid 6192)

It looks like some anti-debugging algorithms are preventing the crackme from running in IDA.

My first try was to patch the crackme in order to bypass the anti-debugging algorithms but I failed and couldn't do it so I thought about other alternatives, my second plan was to try to run the crackme in command line and attach IDA to it and start debugging from there but another problem will arise which is to make a break point in the program so after entering the key the program flow needs to stop so we can debug it.

Now the first thing we'll have to do to achieve our goal is to locate the checking key algorithm and make a break point there, I used the "search for text" feature to try to search for some keywords that are used by the key checking algorithm, so when running the crackme in a command line we can see that there a "Input key:" message so let's search for the word "key", In IDA go to Search > Text:



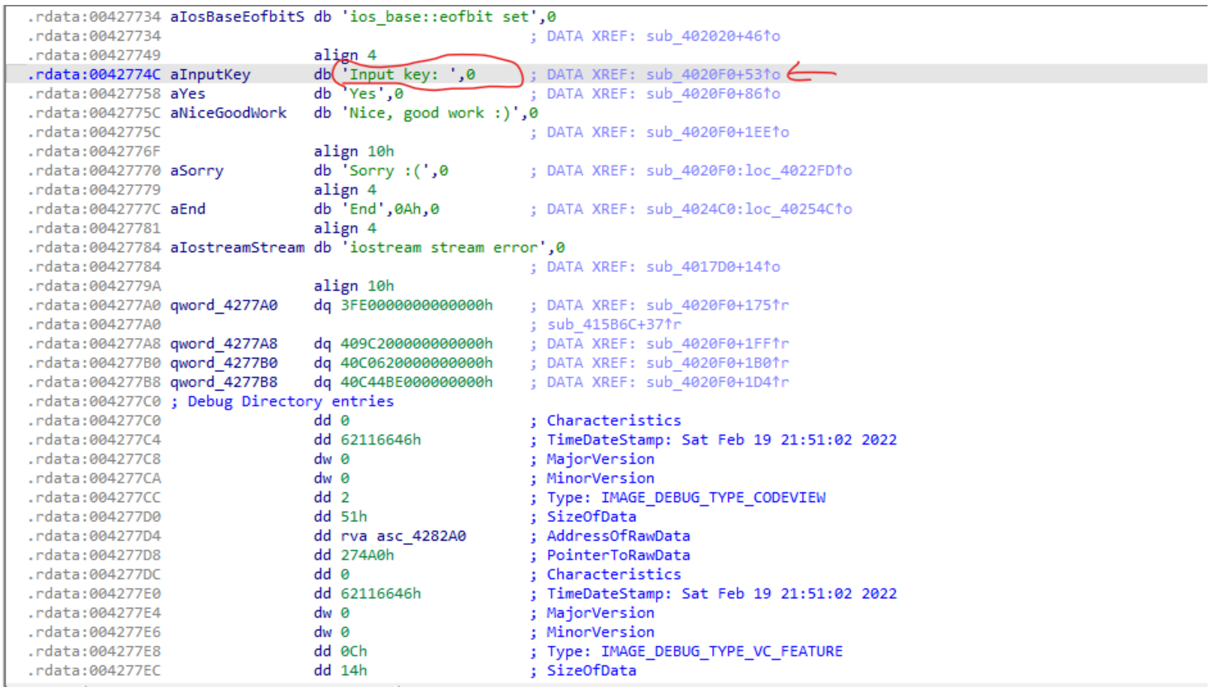
Text search (slow!)

String: KEY

☐ Match case
☐ Regular expression
☐ Identifier
☐ Search Up
☐ Find all occurrences

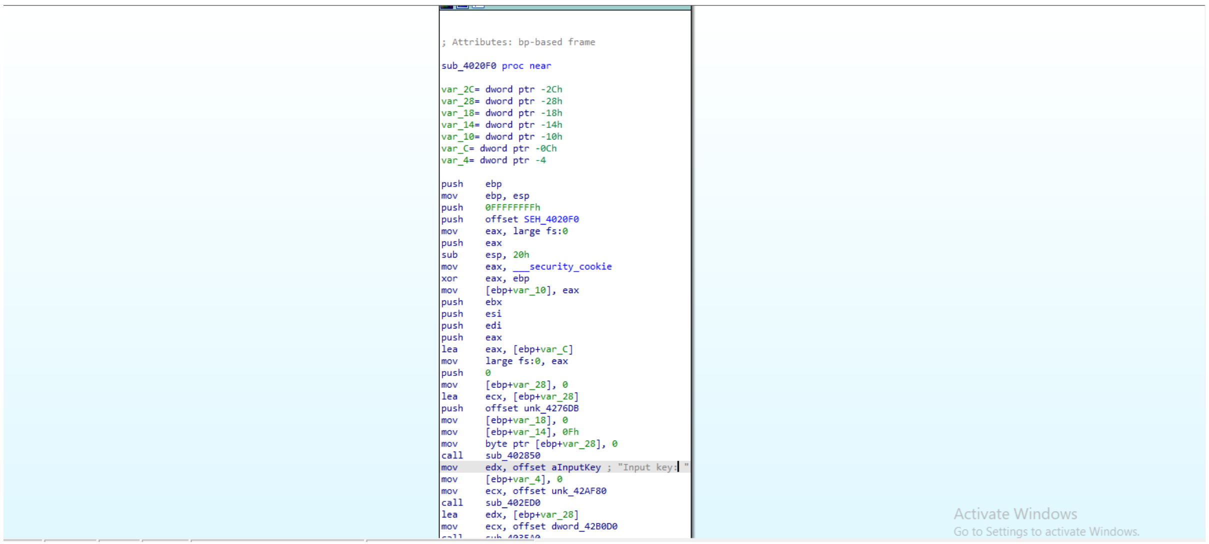
```
public start
start proc near

; FUNCTION CHUNK AT .text:00405A1D SIZE 00000127 BYTES
; FUNCTION CHUNK AT .text:00405B79 SIZE 00000026 BYTES

call    sub_40615E
```

We will be presented with the following code area:



```
.rdata:00427734 aIosBaseEofbitS db 'ios_base::eofbit set',0
.rdata:00427734                                         ; DATA XREF: sub_402020+46↑o
.rdata:00427749                 align 4
.rdata:0042774C aInputKey       db 'Input key: ',0      ; DATA XREF: sub_4020F0+53↑o
.rdata:00427758 aYes            db 'Yes',0              ; DATA XREF: sub_4020F0+86↑o
.rdata:0042775C aNiceGoodWork   db 'Nice, good work :)',0
.rdata:0042775C                                         ; DATA XREF: sub_4020F0+1EE↑o
.rdata:0042776F                 align 10h
.rdata:00427770 aSorry          db 'Sorry :(',0         ; DATA XREF: sub_4020F0:loc_4022FD↑o
.rdata:00427779                 align 4
.rdata:0042777C aEnd            db 'End',0Ah,0          ; DATA XREF: sub_4024C0:loc_40254C↑o
.rdata:00427781                 align 4
.rdata:00427784 aIostreamStream db 'iostream stream error',0
.rdata:00427784                                         ; DATA XREF: sub_4017D0+14↑o
.rdata:0042779A                 align 10h
.rdata:004277A0 qword_4277A0    dq 3FE0000000000000h    ; DATA XREF: sub_4020F0+175↑r
.rdata:004277A0                                         ; sub_415B6C+37↑r
.rdata:004277A8 qword_4277A8    dq 409C200000000000h    ; DATA XREF: sub_4020F0+1FF↑r
.rdata:004277B0 qword_4277B0    dq 40C0620000000000h    ; DATA XREF: sub_4020F0+1B0↑r
.rdata:004277B8 qword_4277B8    dq 40C44BE000000000h    ; DATA XREF: sub_4020F0+1D4↑r
.rdata:004277C0 ; Debug Directory entries
.rdata:004277C0                 dd 0                    ; Characteristics
.rdata:004277C4                 dd 62116646h            ; TimeDateStamp: Sat Feb 19 21:51:02 2022
.rdata:004277C8                 dw 0                    ; MajorVersion
.rdata:004277CA                 dw 0                    ; MinorVersion
.rdata:004277CC                 dd 2                    ; Type: IMAGE_DEBUG_TYPE_CODEVIEW
.rdata:004277D0                 dd 51h                  ; SizeOfData
.rdata:004277D4                 dd rva asc_4282A0       ; AddressOfRawData
.rdata:004277D8                 dd 274A0h               ; PointerToRawData
.rdata:004277DC                 dd 0                    ; Characteristics
.rdata:004277E0                 dd 62116646h            ; TimeDateStamp: Sat Feb 19 21:51:02 2022
.rdata:004277E4                 dw 0                    ; MajorVersion
.rdata:004277E6                 dw 0                    ; MinorVersion
.rdata:004277E8                 dd 0Ch                  ; Type: IMAGE_DEBUG_TYPE_VC_FEATURE
.rdata:004277EC                 dd 14h                  ; SizeOfData
```

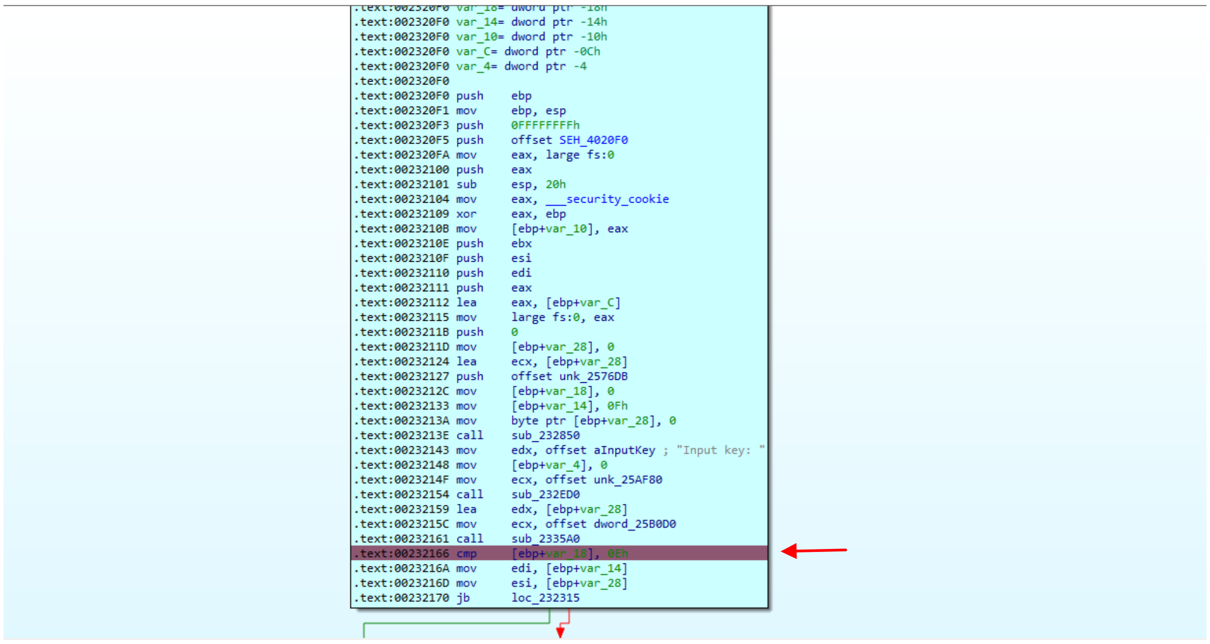
Double click on the “sub_4020F0+53↑o” to go to the subroutine where this variable is referenced:



```
; Attributes: bp-based frame

sub_4020F0 proc near

var_2C= dword ptr -2Ch
var_28= dword ptr -28h
var_18= dword ptr -18h
var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    0FFFFFFFFh
push    offset SEH_4020F0
mov     eax, large fs:0
push    eax
sub     esp, 20h
mov     eax, ___security_cookie
xor     eax, ebp
mov     [ebp+var_10], eax
push    ebx
push    esi
push    edi
push    eax
lea     eax, [ebp+var_C]
mov     large fs:0, eax
push    0
mov     [ebp+var_28], 0
lea     ecx, [ebp+var_28]
push    offset unk_4276DB
mov     [ebp+var_18], 0
mov     [ebp+var_14], 0Fh
mov     byte ptr [ebp+var_28], 0
call    sub_402850
mov     edx, offset aInputKey ; "Input key: "
mov     [ebp+var_4], 0
mov     ecx, offset unk_42AF80
call    sub_402ED0
lea     edx, [ebp+var_28]
mov     ecx, offset dword_42B0D0
```

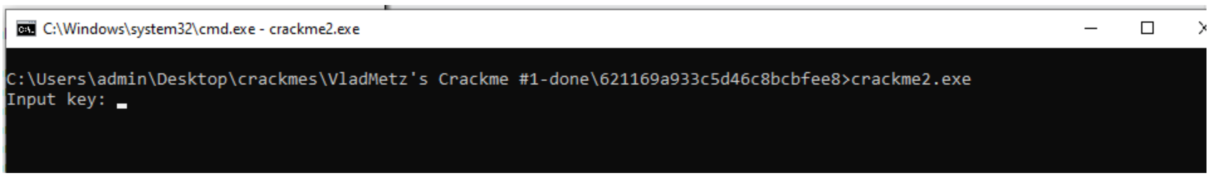
Now we have the algorithm responsible for processing the key.

Let’s make a break point exactly after the code that takes the user input:



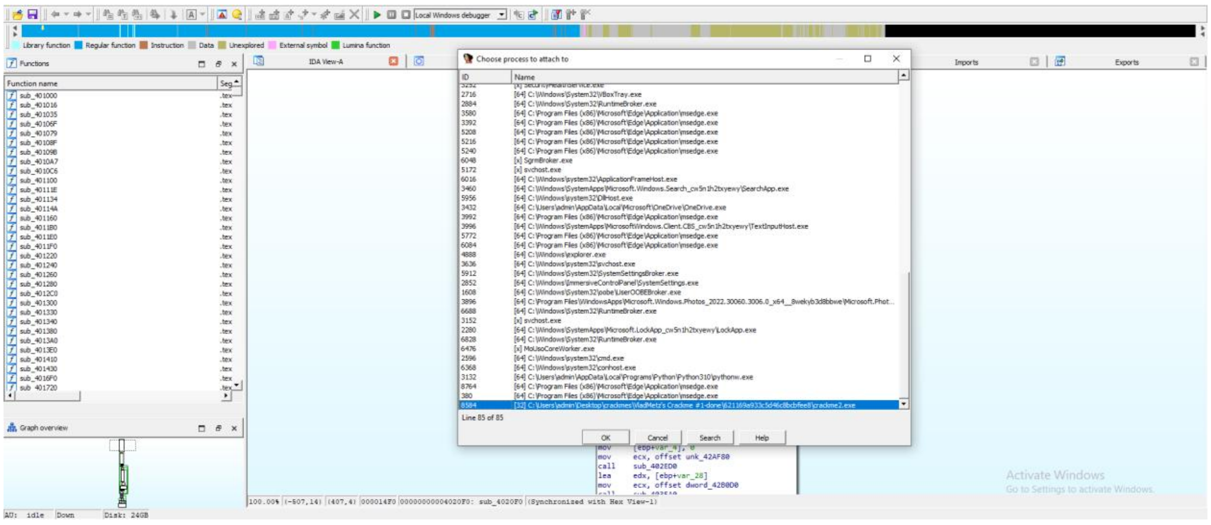
```
.text:002320F0 var_14= dword ptr -14h
.text:002320F0 var_10= dword ptr -10h
.text:002320F0 var_C= dword ptr -0Ch
.text:002320F0 var_4= dword ptr -4
.text:002320F0
.text:002320F0 push    ebp
.text:002320F1 mov     ebp, esp
.text:002320F3 push    0FFFFFFFFh
.text:002320F5 push    offset SEH_4020F0
.text:002320FA mov     eax, large fs:0
.text:00232100 push    eax
.text:00232101 sub     esp, 20h
.text:00232104 mov     eax, ___security_cookie
.text:00232109 xor     eax, ebp
.text:0023210B mov     [ebp+var_10], eax
.text:0023210E push    ebx
.text:0023210F push    esi
.text:00232110 push    edi
.text:00232111 push    eax
.text:00232112 lea     eax, [ebp+var_C]
.text:00232115 mov     large fs:0, eax
.text:0023211B push    0
.text:0023211D mov     [ebp+var_28], 0
.text:00232124 lea     ecx, [ebp+var_28]
.text:00232127 push    offset unk_2576DB
.text:0023212C mov     [ebp+var_18], 0
.text:00232133 mov     [ebp+var_14], 0Fh
.text:0023213A mov     byte ptr [ebp+var_28], 0
.text:0023213E call    sub_232850
.text:00232143 mov     edx, offset aInputKey ; "Input key: "
.text:00232148 mov     [ebp+var_4], 0
.text:0023214F mov     ecx, offset unk_25AF80
.text:00232154 call    sub_232ED0
.text:00232159 lea     edx, [ebp+var_28]
.text:0023215C mov     ecx, offset dword_25B0D0
.text:00232161 call    sub_2335A0
.text:00232166 cmp     [ebp+var_18], 0Eh
.text:0023216A mov     edi, [ebp+var_14]
.text:0023216D mov     esi, [ebp+var_28]
.text:00232170 jb      loc_232315
```

Now let's run the crackme in the command line :



```
C:\Windows\system32\cmd.exe - crackme2.exe

C:\Users\admin\Desktop\crackmes\VladMetz's Crackme #1-done\621169a933c5d46c8bcbfee8>crackme2.exe
Input key: _
```

Let's attach IDA to this crackme, go to Debugger > Attach to process, and choose the running crackme:

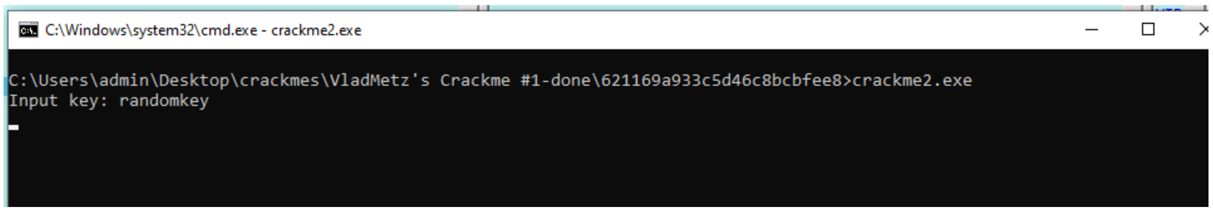


```
Functions
Function name      Segment
sub_401000         .tex
sub_401016         .tex
sub_401035         .tex
sub_40106F         .tex
sub_401079         .tex
sub_40108F         .tex
sub_40109B         .tex
sub_4010A7         .tex
sub_4010C6         .tex
sub_401100         .tex
sub_40111E         .tex
sub_401134         .tex
sub_40114A         .tex
sub_401160         .tex
sub_4011B0         .tex
sub_4011E0         .tex
sub_4011F0         .tex
sub_401220         .tex
sub_401240         .tex
sub_401260         .tex
sub_401280         .tex
sub_4012C0         .tex
sub_401300         .tex
sub_401330         .tex
sub_401340         .tex
sub_401380         .tex
sub_4013A0         .tex
sub_4013E0         .tex
sub_401410         .tex
sub_401430         .tex
sub_4016F0         .tex
sub_401720         .tex
```

```
Choose process to attach to
ID      Name
2716    [64] C:\Windows\System32\VBoxTray.exe
2884    [64] C:\Windows\System32\RuntimeBroker.exe
3580    [64] C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
3392    [64] C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
5208    [64] C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
5216    [64] C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
5240    [64] C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
6048    [x] SgrmBroker.exe
5172    [x] svchost.exe
6016    [64] C:\Windows\system32\ApplicationFrameHost.exe
3460    [64] C:\Windows\SystemApps\Microsoft.Windows.Search_cw5n1h2txyewy\SearchApp.exe
5956    [64] C:\Windows\system32\DllHost.exe
3432    [64] C:\Users\admin\AppData\Local\Microsoft\OneDrive\OneDrive.exe
3992    [64] C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
3996    [64] C:\Windows\SystemApps\MicrosoftWindows.Client.CBS_cw5n1h2txyewy\TextInputHost.exe
5772    [64] C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
6084    [64] C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
4888    [64] C:\Windows\explorer.exe
3636    [64] C:\Windows\system32\svchost.exe
5912    [64] C:\Windows\System32\SystemSettingsBroker.exe
2852    [64] C:\Windows\ImmersiveControlPanel\SystemSettings.exe
1608    [64] C:\Windows\System32\oobe\UserOOBEBroker.exe
3896    [64] C:\Program Files\WindowsApps\Microsoft.Windows.Photos_2022.30060.3006.0_x64__8wekyb3d8bbwe\Microsoft.Phot...
6688    [64] C:\Windows\System32\RuntimeBroker.exe
3152    [x] svchost.exe
2280    [64] C:\Windows\SystemApps\Microsoft.LockApp_cw5n1h2txyewy\LockApp.exe
6828    [64] C:\Windows\System32\RuntimeBroker.exe
6476    [x] MoUsoCoreWorker.exe
2596    [64] C:\Windows\system32\cmd.exe
6368    [64] C:\Windows\system32\conhost.exe
3132    [64] C:\Users\admin\AppData\Local\Programs\Python\Python310\pythonw.exe
8764    [64] C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
380     [64] C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
8584    [32] C:\Users\admin\Desktop\crackmes\VladMetz's Crackme #1-done\621169a933c5d46c8bcbfee8\crackme2.exe
Line 85 of 85
```

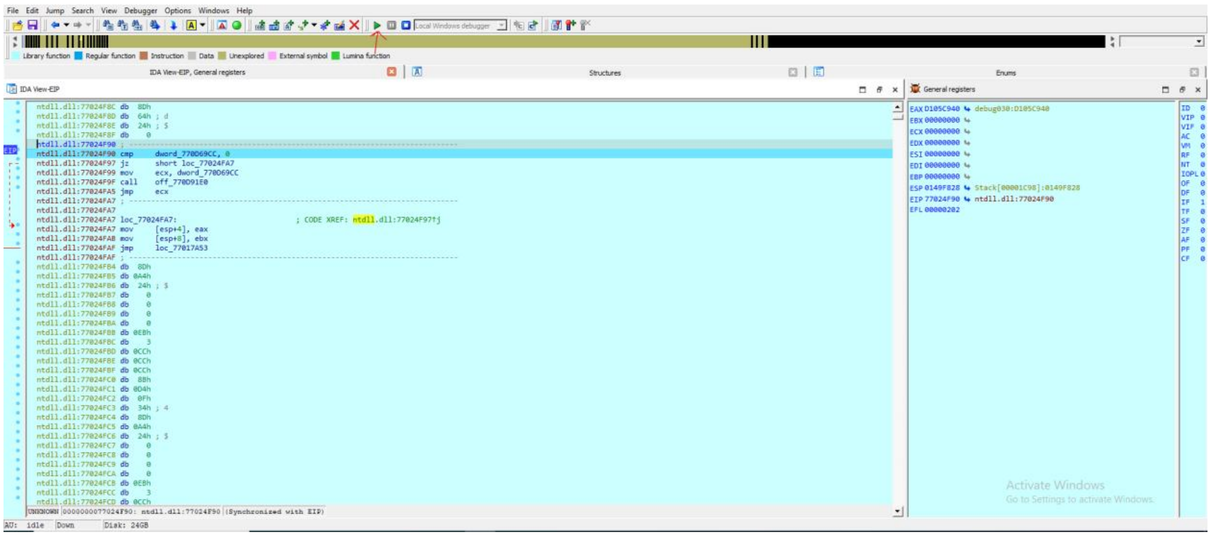
```
mov     ecx, offset unk_42AF80
call    sub_402ED0
lea     edx, [ebp+var_28]
mov     ecx, offset dword_42B0D0
```

Input a random key in the command line and hit enter:



```
C:\Windows\system32\cmd.exe - crackme2.exe

C:\Users\admin\Desktop\crackmes\VladMetz's Crackme #1-done\621169a933c5d46c8bcbfee8>crackme2.exe
Input key: randomkey
_
```

Go back to IDA and hit continue:

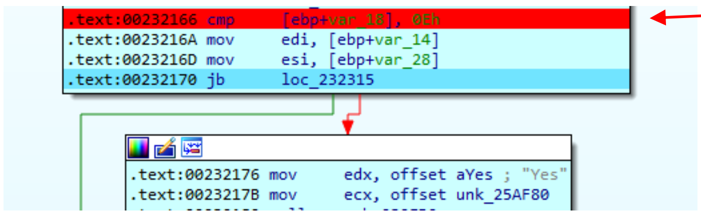


```
IDA View-EIP
ntdll.dll:77024F8C db  8Dh
ntdll.dll:77024F8D db  64h ; d
ntdll.dll:77024F8E db  24h ; $
ntdll.dll:77024F8F db    0
ntdll.dll:77024F90 ; ---------------------------------------------------------------
ntdll.dll:77024F90 cmp     dword_770D69CC, 0
ntdll.dll:77024F97 jz      short loc_77024FA7
ntdll.dll:77024F99 mov     ecx, dword_770D69CC
ntdll.dll:77024F9F call    off_770D91E0
ntdll.dll:77024FA5 jmp     ecx
ntdll.dll:77024FA7 ; ---------------------------------------------------------------
ntdll.dll:77024FA7
ntdll.dll:77024FA7 loc_77024FA7:                          ; CODE XREF: ntdll.dll:77024F97↑j
ntdll.dll:77024FA7 mov     [esp+4], eax
ntdll.dll:77024FAB mov     [esp+8], ebx
ntdll.dll:77024FAF jmp     loc_77017A53
ntdll.dll:77024FAF ; ---------------------------------------------------------------
ntdll.dll:77024FB4 db  8Dh
ntdll.dll:77024FB5 db 0A4h
ntdll.dll:77024FB6 db  24h ; $
ntdll.dll:77024FB7 db    0
ntdll.dll:77024FB8 db    0
ntdll.dll:77024FB9 db    0
ntdll.dll:77024FBA db    0
ntdll.dll:77024FBB db 0EBh
ntdll.dll:77024FBC db    3
ntdll.dll:77024FBD db 0CCh
ntdll.dll:77024FBE db 0CCh
ntdll.dll:77024FBF db 0CCh
ntdll.dll:77024FC0 db  8Bh
ntdll.dll:77024FC1 db 0D4h
ntdll.dll:77024FC2 db  0Fh
ntdll.dll:77024FC3 db  34h ; 4
ntdll.dll:77024FC4 db  8Dh
ntdll.dll:77024FC5 db 0A4h
ntdll.dll:77024FC6 db  24h ; $
ntdll.dll:77024FC7 db    0
ntdll.dll:77024FC8 db    0
ntdll.dll:77024FC9 db    0
ntdll.dll:77024FCA db    0
ntdll.dll:77024FCB db 0EBh
ntdll.dll:77024FCC db    3
ntdll.dll:77024FCD db 0CCh
UNKNOWN 0000000077024F90: ntdll.dll:77024F90 (Synchronized with EIP)
```

```
General registers
EAX D105C940 ↳ debug030:D105C940
EBX 00000000 ↳
ECX 00000000 ↳
EDX 00000000 ↳
ESI 00000000 ↳
EDI 00000000 ↳
EBP 00000000 ↳
ESP 0149F828 ↳ Stack[00001C98]:0149F828
EIP 77024F90 ↳ ntdll.dll:77024F90
EFL 00000202

ID   0
VIP  0
VIF  0
AC   0
VM   0
RF   0
NT   0
IOPL 0
OF   0
DF   0
IF   1
TF   0
SF   0
ZF   0
AF   0
PF   0
CF   0
```

The program flow will stop and we can start debugging now.

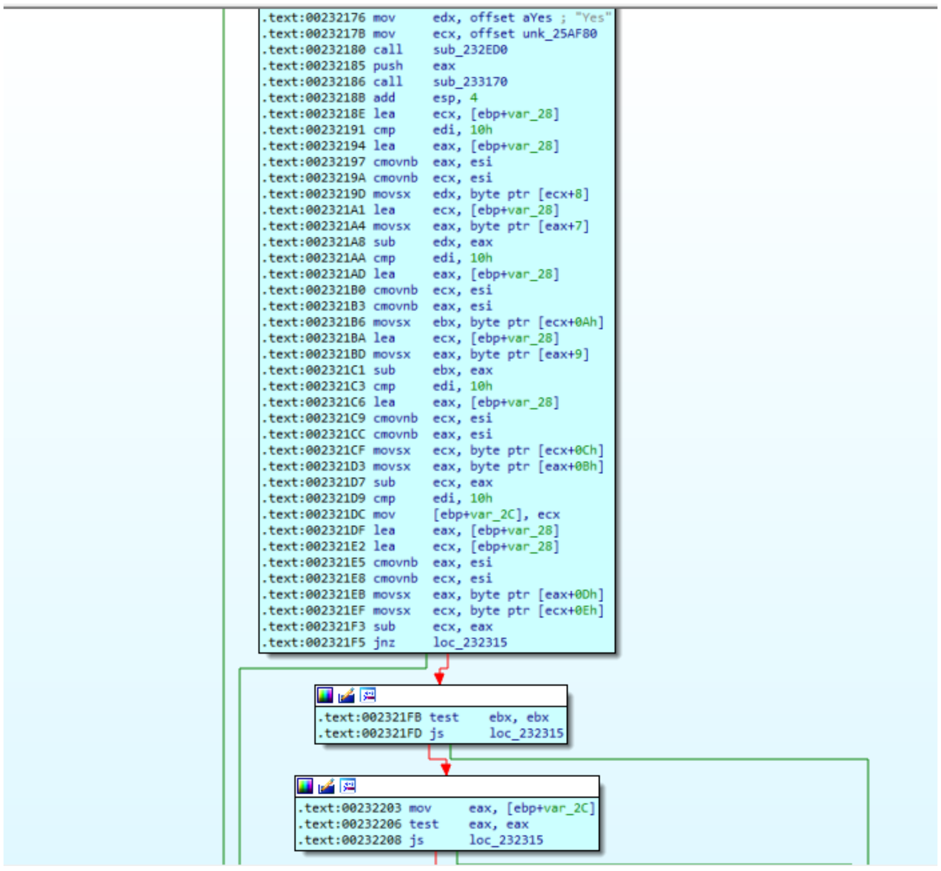
The first check this crackme does after entering a key is to check the length of the key:



```
.text:00232166 cmp     [ebp+var_18], 0Eh
.text:0023216A mov     edi, [ebp+var_14]
.text:0023216D mov     esi, [ebp+var_28]
.text:00232170 jb      loc_232315

.text:00232176 mov     edx, offset aYes ; "Yes"
.text:0023217B mov     ecx, offset unk_25AF80
```

The code above checks the length of the key and to pass this check the key length must be greater than or equal to 14.

The second check:

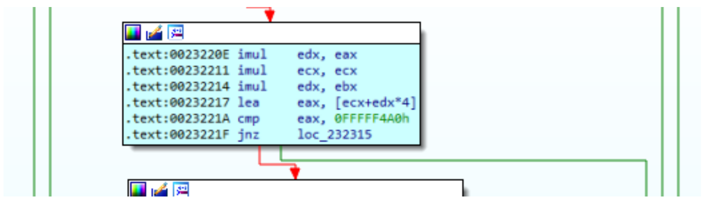


```
.text:00232176 mov     edx, offset aYes ; "Yes"
.text:0023217B mov     ecx, offset unk_25AF80
.text:00232180 call    sub_232ED0
.text:00232185 push    eax
.text:00232186 call    sub_233170
.text:0023218B add     esp, 4
.text:0023218E lea     ecx, [ebp+var_28]
.text:00232191 cmp     edi, 10h
.text:00232194 lea     eax, [ebp+var_28]
.text:00232197 cmovnb  eax, esi
.text:0023219A cmovnb  ecx, esi
.text:0023219D movsx   edx, byte ptr [ecx+8]
.text:002321A1 lea     ecx, [ebp+var_28]
.text:002321A4 movsx   eax, byte ptr [eax+7]
.text:002321A8 sub     edx, eax
.text:002321AA cmp     edi, 10h
.text:002321AD lea     eax, [ebp+var_28]
.text:002321B0 cmovnb  ecx, esi
.text:002321B3 cmovnb  eax, esi
.text:002321B6 movsx   ebx, byte ptr [ecx+0Ah]
.text:002321BA lea     ecx, [ebp+var_28]
.text:002321BD movsx   eax, byte ptr [eax+9]
.text:002321C1 sub     ebx, eax
.text:002321C3 cmp     edi, 10h
.text:002321C6 lea     eax, [ebp+var_28]
.text:002321C9 cmovnb  ecx, esi
.text:002321CC cmovnb  eax, esi
.text:002321CF movsx   ecx, byte ptr [ecx+0Ch]
.text:002321D3 movsx   eax, byte ptr [eax+0Bh]
.text:002321D7 sub     ecx, eax
.text:002321D9 cmp     edi, 10h
.text:002321DC mov     [ebp+var_2C], ecx
.text:002321DF lea     eax, [ebp+var_28]
.text:002321E2 lea     ecx, [ebp+var_28]
.text:002321E5 cmovnb  eax, esi
.text:002321E8 cmovnb  ecx, esi
.text:002321EB movsx   eax, byte ptr [eax+0Dh]
.text:002321EF movsx   ecx, byte ptr [ecx+0Eh]
.text:002321F3 sub     ecx, eax
.text:002321F5 jnz     loc_232315

.text:002321FB test    ebx, ebx
.text:002321FD js      loc_232315

.text:00232203 mov     eax, [ebp+var_2C]
.text:00232206 test    eax, eax
.text:00232208 js      loc_232315
```

The above code can be abstracted into the following python code:

```python
if (ord(key[14])-ord(key[13])==0) and (ord(key[10])-ord(key[9]) > 0) and (ord(key[12])-ord(key[11])) > 0:
```

The third check:

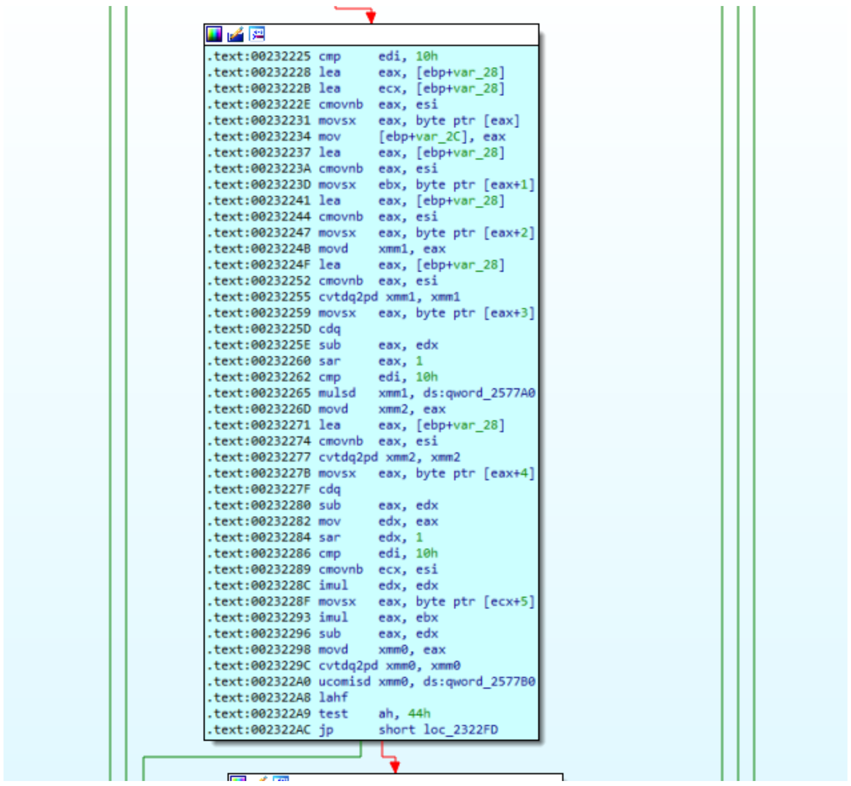


```
.text:0023220E imul    edx, eax
.text:00232211 imul    ecx, ecx
.text:00232214 imul    edx, ebx
.text:00232217 lea     eax, [ecx+edx*4]
.text:0023221A cmp     eax, 0FFFFF4A0h
.text:0023221F jnz     loc_232315
```

Python abstraction for the code above:

```python
edx=  (ord(key[8]) - ord(key[7]) ) * (ord(key[12]) - ord(key[11]) )

ecx= (ord(key[14])-ord(key[13])) * (ord(key[14])-ord(key[13])) *  (ord(key[14])-ord(key[13])) * (ord(key[14])-ord(key[13]))

edx=edx*(ord(key[10])-ord(key[9]))

eax=ecx+edx*4

if eax == -2912:
```

Fourth check:

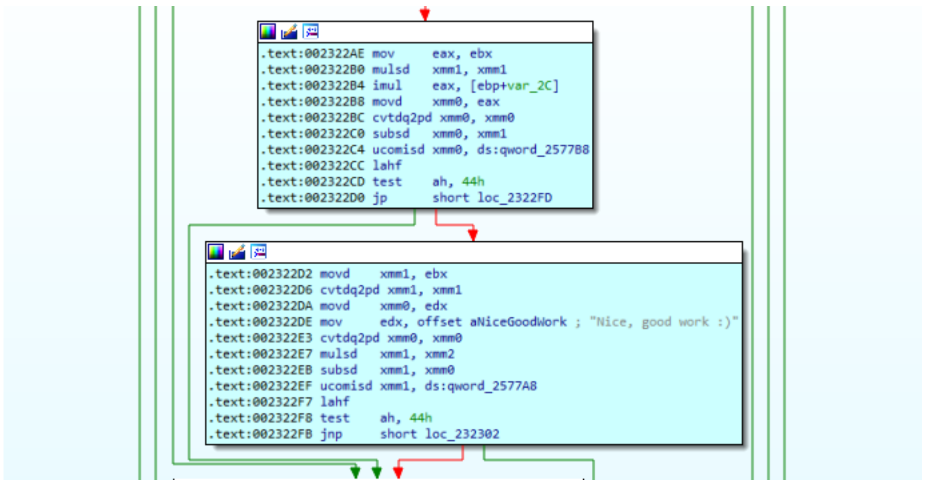


```
.text:00232225 cmp      edi, 10h
.text:00232228 lea      eax, [ebp+var_28]
.text:0023222B lea      ecx, [ebp+var_28]
.text:0023222E cmovnb   eax, esi
.text:00232231 movsx    eax, byte ptr [eax]
.text:00232234 mov      [ebp+var_2C], eax
.text:00232237 lea      eax, [ebp+var_28]
.text:0023223A cmovnb   eax, esi
.text:0023223D movsx    ebx, byte ptr [eax+1]
.text:00232241 lea      eax, [ebp+var_28]
.text:00232244 cmovnb   eax, esi
.text:00232247 movsx    eax, byte ptr [eax+2]
.text:0023224B movd     xmm1, eax
.text:0023224F lea      eax, [ebp+var_28]
.text:00232252 cmovnb   eax, esi
.text:00232255 cvtdq2pd xmm1, xmm1
.text:00232259 movsx    eax, byte ptr [eax+3]
.text:0023225D cdq
.text:0023225E sub      eax, edx
.text:00232260 sar      eax, 1
.text:00232262 cmp      edi, 10h
.text:00232265 mulsd    xmm1, ds:qword_2577A0
.text:0023226D movd     xmm2, eax
.text:00232271 lea      eax, [ebp+var_28]
.text:00232274 cmovnb   eax, esi
.text:00232277 cvtdq2pd xmm2, xmm2
.text:0023227B movsx    eax, byte ptr [eax+4]
.text:0023227F cdq
.text:00232280 sub      eax, edx
.text:00232282 mov      edx, eax
.text:00232284 sar      edx, 1
.text:00232286 cmp      edi, 10h
.text:00232289 cmovnb   ecx, esi
.text:0023228C imul     edx, edx
.text:0023228F movsx    eax, byte ptr [ecx+5]
.text:00232293 imul     eax, ebx
.text:00232296 sub      eax, edx
.text:00232298 movd     xmm0, eax
.text:0023229C cvtdq2pd xmm0, xmm0
.text:002322A0 ucomisd  xmm0, ds:qword_2577B0
.text:002322A8 lahf
.text:002322A9 test     ah, 44h
.text:002322AC jp       short loc_2322FD
```

Python abstraction for the code above:

```
eax=ord(key[0])
var_2c = eax
ebx=ord(key[1])
eax=ord(key[2])
xmm1 = eax
eax = ord(key[3])
eax = eax >> 1
xmm1 = xmm1 * 0.5
xmm2 = eax
eax = ord(key[4])
edx = eax
edx = edx >> 1
edx = edx * edx
eax = ord(key[5])
eax = eax * ebx
eax = eax - edx
xmm0 = eax

if xmm0 == 8388:
```

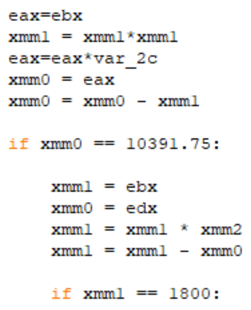
Fifth check:



```
.text:002322AE mov      eax, ebx
.text:002322B0 mulsd    xmm1, xmm1
.text:002322B4 imul     eax, [ebp+var_2C]
.text:002322B8 movd     xmm0, eax
.text:002322BC cvtdq2pd xmm0, xmm0
.text:002322C0 subsd    xmm0, xmm1
.text:002322C4 ucomisd  xmm0, ds:qword_2577B8
.text:002322CC lahf
.text:002322CD test     ah, 44h
.text:002322D0 jp       short loc_2322FD
```

```
.text:002322D2 movd     xmm1, ebx
.text:002322D6 cvtdq2pd xmm1, xmm1
.text:002322DA movd     xmm0, edx
.text:002322DE mov      edx, offset aNiceGoodWork ; "Nice, good work :)"
.text:002322E3 cvtdq2pd xmm0, xmm0
.text:002322E7 mulsd    xmm1, xmm2
.text:002322EB subsd    xmm1, xmm0
.text:002322EF ucomisd  xmm1, ds:qword_2577A8
.text:002322F7 lahf
.text:002322F8 test     ah, 44h
.text:002322FB jnp      short loc_232302
```

Python abstraction for the code above:



```
eax=ebx
xmm1 = xmm1*xmm1
eax=eax*var_2c
xmm0 = eax
xmm0 = xmm0 - xmm1

if xmm0 == 10391.75:

    xmm1 = ebx
    xmm0 = edx
    xmm1 = xmm1 * xmm2
    xmm1 = xmm1 - xmm0

    if xmm1 == 1800:
```

After passing all previous checks we get a valid key.

A keygen is provided along with this write up.