

# Malware Analysis

## Reverse Engineering

Created By: -

Rohan Goswami

### AIM:

Consider yourself as a MA team member in the MMC company, they have given you a task to bypass the license key for Some executable files. You have to do reverse engineering on the given executable files and bypass the license key or try to get unauthorized access.

### Exe File 1: ReverseMe.exe

So first of all We use Ollydbg, which is a debugging tool used for binary analysis.

So it converts binary language into assembly language.

We basically take things apart, figuring it out how it works and keeping it together.

So when we open ReverseMe.exe in ollydbg the first thing we see is an assembly language instructions given to execute the exe file now we have to analyze how it works and crack it.

So it calls GetModuleHandleA.

```
00401000 $ 6A 00 PUSH 0
00401002 . E8 64020000 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 . A3 77214000 MOV DWORD PTR DS:[402177],EAX
0040100C . C705 97214000 MOV DWORD PTR DS:[402197],4003
00401016 . C705 9B214000 MOV DWORD PTR DS:[40219B],reverseM.00401
00401020 . C705 9F214000 MOV DWORD PTR DS:[40219F],0
0040102A . C705 A3214000 MOV DWORD PTR DS:[4021A3],0
00401034 . A1 77214000 MOV EAX,DWORD PTR DS:[402177]
00401039 . A3 A7214000 MOV DWORD PTR DS:[4021A7],EAX
0040103E . 6A 04 PUSH 4
00401040 . 50 PUSH EAX
00401046 . E8 3F030000 CALL <JMP.&USER32.LoadIconA>
0040104B . A3 AB214000 MOV DWORD PTR DS:[4021AB],EAX
00401050 . 68 007F0000 PUSH 7F00
00401052 . E8 C8020000 CALL <JMP.&USER32.LoadCursorA>
00401057 . A3 AF214000 MOV DWORD PTR DS:[4021AF],EAX
0040105C . 6A 00 PUSH 0
0040105E . 68 6F214000 PUSH reverseM.0040216F
00401063 . 6A 03 PUSH 3
00401065 . 6A 00 PUSH 0
00401067 . 6A 03 PUSH 3
00401069 . 68 000000C0 PUSH C0000000
00401077 . 68 000000C0 PUSH C0000000
```

Registers (FPU)  
EAX 008B067D  
ECX 00331F29  
EDX 00400000 ASCII "MZP"  
ESP 0019FF78  
ESI 00401000 reverseM.<  
EDI 00401000 reverseM.<  
EIP 00401046 reverseM.0  
ES 002B 32bit 0(FF)  
CS 0023 32bit 0(FF)  
SS 002B 32bit 0(FF)  
DS 002B 32bit 0(FF)  
FS 0053 32bit 37E0  
GS 002B 32bit 0(FF)  
LastErr ERROR\_SUCC  
TL 00000246 (NO,NB,E,B  
empty 0.0  
empty 0.0

Now you can see it creates a file named Keyfile.dat.

```
0040102A . C705 A3214000 MOV DWORD PTR DS:[4021A3],0
00401034 . A1 77214000 MOV EAX,DWORD PTR DS:[402177]
00401039 . A3 A7214000 MOV DWORD PTR DS:[4021A7],EAX
0040103E . 6A 04 PUSH 4
00401040 . 50 PUSH EAX
00401046 . E8 3F030000 CALL <JMP.&USER32.LoadIconA>
0040104B . A3 AB214000 MOV DWORD PTR DS:[4021AB],EAX
00401050 . 68 007F0000 PUSH 7F00
00401052 . E8 C8020000 CALL <JMP.&USER32.LoadCursorA>
00401057 . A3 AF214000 MOV DWORD PTR DS:[4021AF],EAX
0040105C . 6A 00 PUSH 0
0040105E . 68 6F214000 PUSH reverseM.0040216F
00401063 . 6A 03 PUSH 3
00401065 . 6A 00 PUSH 0
00401067 . 6A 03 PUSH 3
00401069 . 68 000000C0 PUSH C0000000
00401077 . 68 79204000 PUSH reverseM.00402079
00401079 . E8 0B020000 CALL <JMP.&KERNEL32.CreateFileA>
0040107B . 83F8 FF CMP EAX,-1
0040107D . 75 1D JNZ SHORT reverseM.0040109A
0040107F . 6A 00 PUSH 0
00401081 . 68 00204000 PUSH reverseM.00402000
```

Registers (FPU)  
EAX FFFFFFFF  
ECX 005E0000  
EDX 005E0000  
EBX 0038A000  
ESP 0019FF78  
EBP 0019FF84  
ESI 00401000 reverseM  
EDI 00401000 reverseM  
EIP 00401078 reverseM  
C 0 ES 002B 32bit 0(FF)  
P 1 CS 0023 32bit 0(FF)  
A 0 SS 002B 32bit 0(FF)  
Z 1 DS 002B 32bit 0(FF)  
S 0 FS 0053 32bit 38  
T 0 GS 002B 32bit 0(FF)  
D 0  
0 0 LastErr ERROR\_FI  
EFL 00000246 (NO,NB,E  
ST0 empty 0.0  
ST1 empty 0.0

Address Hex dump ASCII  
00402000 20 4B 65 79 20 46 69 6C Key Fil  
00402008 65 20 52 65 76 65 72 73 e Revers  
00402010 65 40 65 00 00 00 20 45 eMe... E  
00402018 76 61 6C 75 61 74 69 6F valuat  
00402020 6E 20 70 65 72 69 6F 64 n period  
00402028 20 6F 75 74 20 6F 66 20 out of  
00402030 64 61 74 65 2E 20 50 75 date. Pu  
00402038 72 63 68 61 73 65 20 6E rchase n  
00402040 65 77 20 6C 69 63 65 6E em lisen  
00402048 73 65 00 00 00 00 00 00 se.....  
00402050 00 00 00 00 00 00 00 00 .....  
00402058 00 00 00 00 00 00 00 00 .....  
00402060 00 00 00 00 00 00 00 00 .....  
00402068 00 00 00 00 00 00 00 00 .....  
00402070 00 00 00 00 00 00 00 00 .....

0019FF78 755E6739 RETURN to KERNEL32.755E6739  
0019FF7C 0038A000  
0019FF80 755E6720 KERNEL32.BaseThreadInitThunk  
0019FF84 0019FFDC  
0019FF88 77008FD2 RETURN to ntdll.77008FD2  
0019FF8C 0038A000  
0019FF90 C5E4C8C9  
0019FF94 00000000  
0019FF98 00000000  
0019FF9C 0038A000  
0019FFA0 00000000  
0019FFA4 00000000  
0019FFA8 00000000  
0019FFAC 00000000  
0019FFB0 00000000  
0019FFB4 00000000

Now if in JNZ we set Z = 0, then it will go to a specific address, it will only go to specific location if Z = 0 so the problem was that if we set Z = 1, it will execute code until it exits the process.

Registers (FPU)

EAX	FFFFFFFF
ECX	006F0000
EDX	006F0000
EBX	0030B000
ESP	0019FF78
EBP	0019FF84
ESI	00401000 reverseM.
EDI	00401000 reverseM.
EIP	0040107B reverseM.
C 0	ES 002B 32bit 0(F
P 1	CS 0023 32bit 0(F
A 0	SS 002B 32bit 0(F
Z 0	DS 002B 32bit 0(F
S 0	FS 0053 32bit 30E
T 0	GS 002B 32bit 0(F
D 0	
0 0	LastErr ERROR_FIL
EFL	00000206 (NO, NB, NE
ST0	empty 0.0
ST1	empty 0.0

Now it reads the file and tests the extended accumulator.

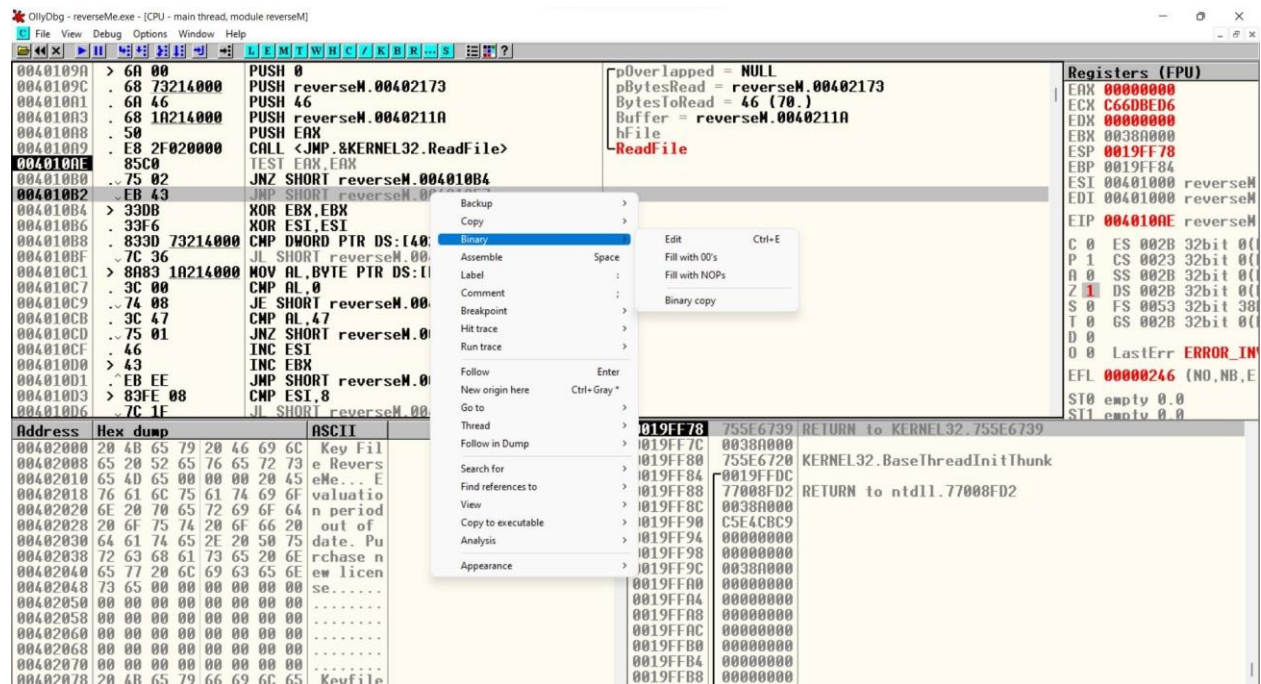
Registers (FPU)

EAX	FFFFFFFF
ECX	005E0000
EDX	005E0000
EBX	0030B000
ESP	0019FF74
EBP	0019FF84
ESI	00401000 reverseM.
EDI	00401000 reverseM.
EIP	0040109C reverseM.
C 0	ES 002B 32bit 0(F
P 1	CS 0023 32bit 0(F
A 0	SS 002B 32bit 0(F
Z 0	DS 002B 32bit 0(F
S 0	FS 0053 32bit 38E
T 0	GS 002B 32bit 0(F
D 0	
0 0	LastErr ERROR_FIL
EFL	00000206 (NO, NB, NE
ST0	empty 0.0
ST1	empty 0.0

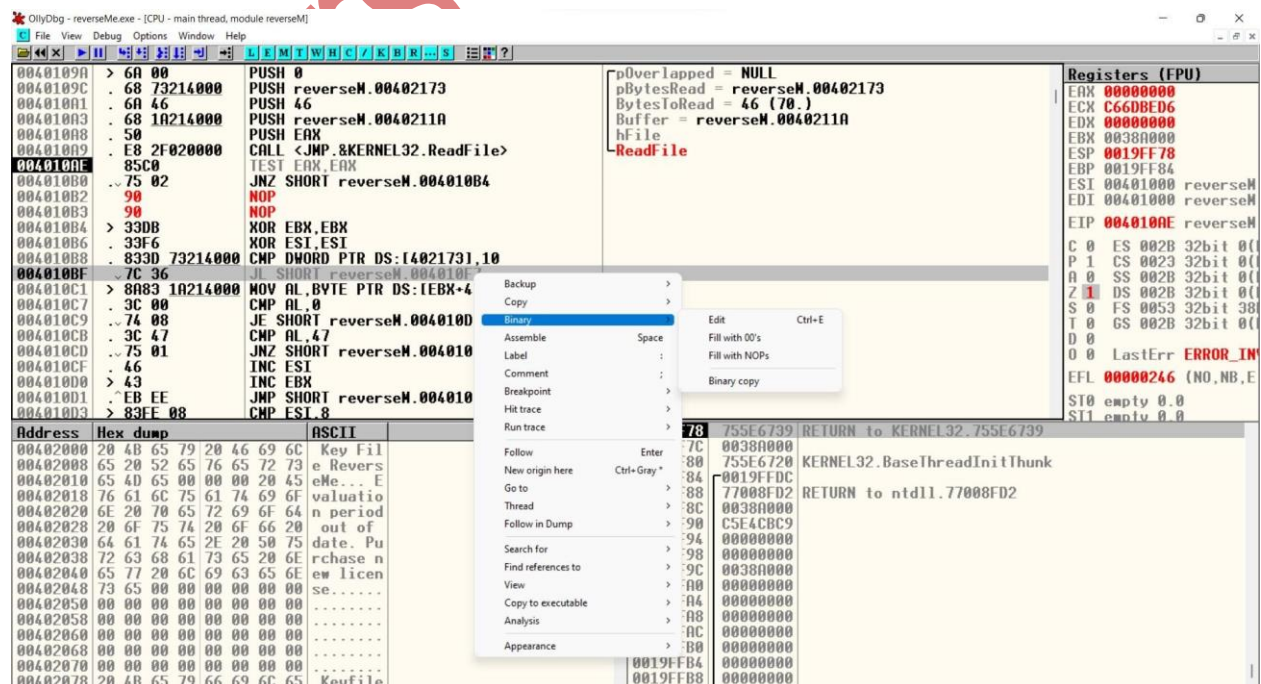


Here as you can see after the TEST is done it goes to address so if it goes to this address as 004010F7 then i think it will give us as the key file is not valid.

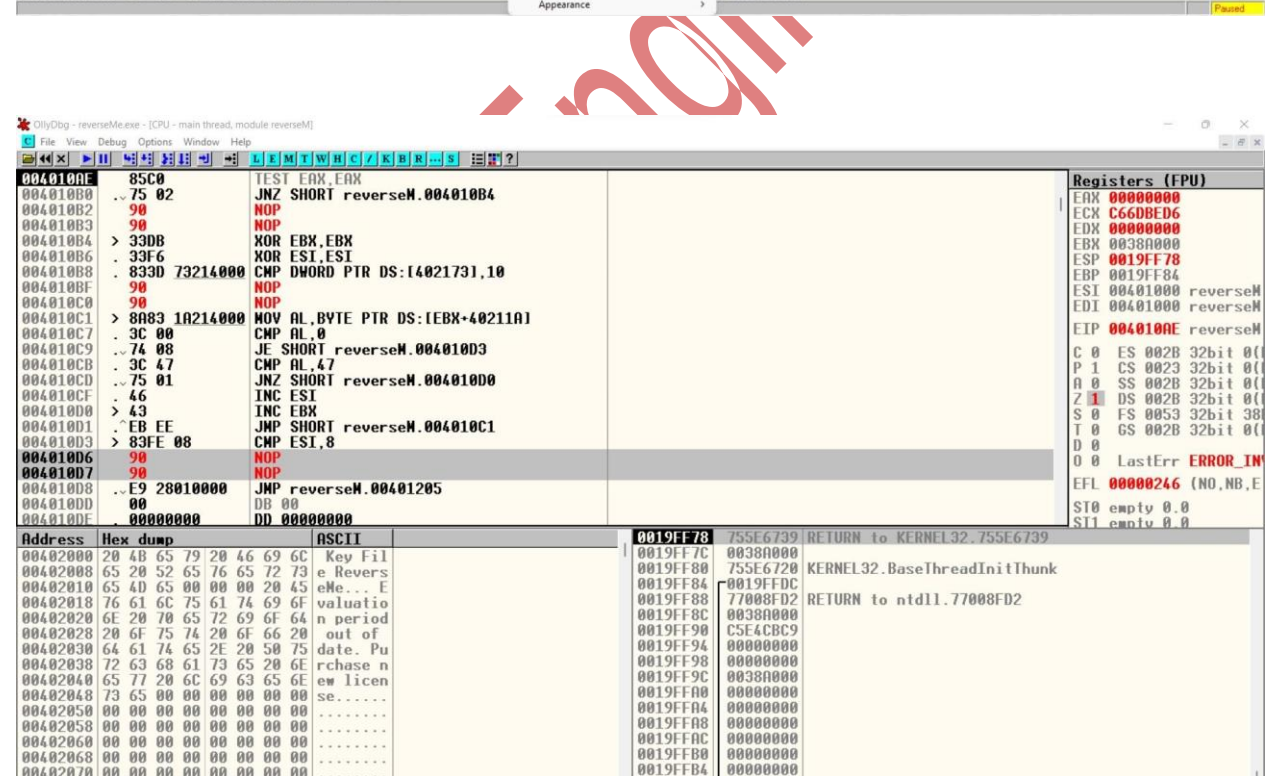
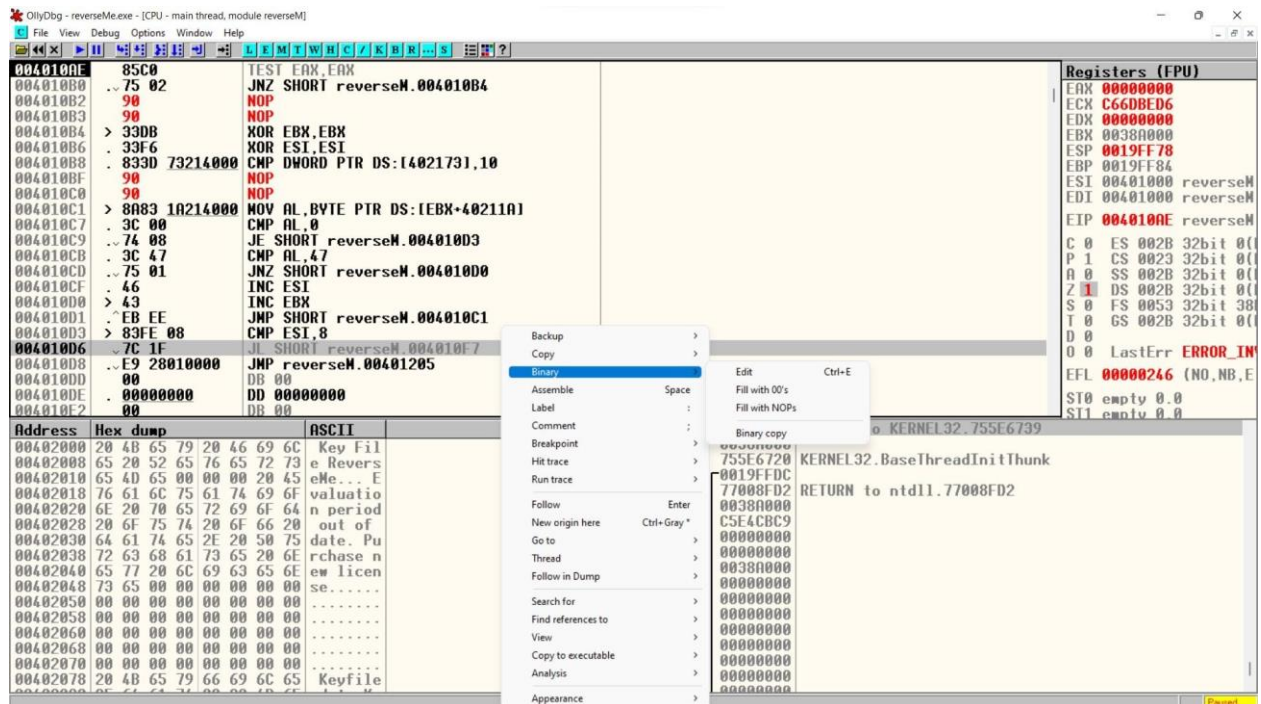
So what we have to do is whenever 004010F7 is inputted we just have to fill it as NOP.



After that in 10BF it gives 004010F7 address we also have to fill with NOP.



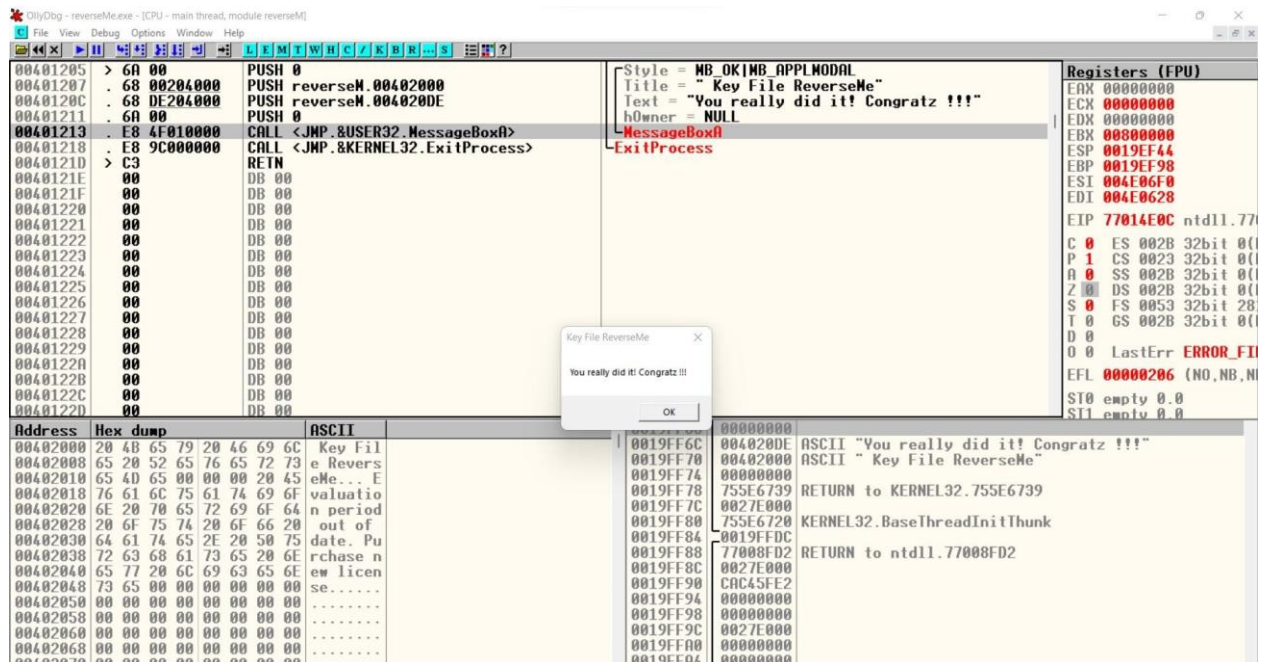
After that in 10D6 it gives 004010F7 addresses we also have to fill with NOP.



After that we executed it.

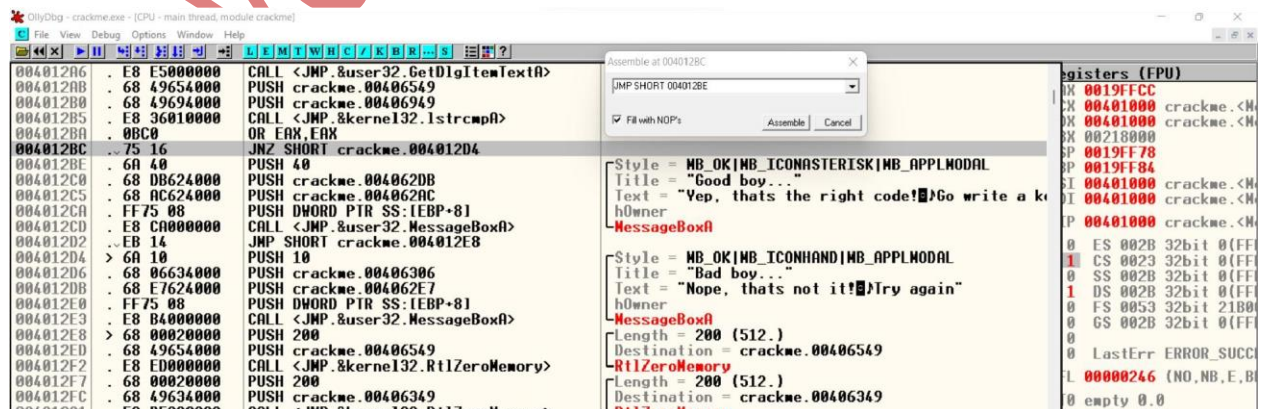


You can see it goes to 00401205 and when it goes to MessageBoxA a message box will appear and it will give us you really did it.



## Exe File 2: Crackme.exe

So what happens here when you execute the code, it will give us a dialogue box about giving username and reg code, so when we write 2 characters then will say minimum 4 characters required.



```
00401116 C1E8 10 SHR EDX,10
00401119 66:0BD2 OR DX,DX
0040111C 0F85 5B020000 JNZ crackme.0040137D
00401122 66:3D EC03 CMP AX,3EC
00401126 0F85 00020000 JNZ crackme.0040132C
0040112C 50 PUSH EAX
0040112D 53 PUSH EBX
0040112E 55 PUSH EBP
0040112F 68 00020000 PUSH 200
00401134 68 49634000 PUSH crackme.00406349
00401139 68 E8030000 PUSH 3E8
0040113E FF75 08 PUSH DWORD PTR SS:[EBP+8]
00401141 E8 4A020000 CALL <JMP.&user32.GetDlgItemTextA>
00401146 83F8 03 CMP EAX,3
00401149 77 18 JA SHORT crackme.00401163
0040114B 6A 10 PUSH 10
0040114D 68 06634000 PUSH crackme.00406349
00401152 68 0A624000 PUSH crackme.00406240
00401157 FF75 08 PUSH DWORD PTR SS:[EBP+8]
0040115A E8 3D020000 CALL <JMP.&user32.MessageBoxA>
0040115F C9 LEAVE
00401160 C2 1000 RETN 10
00401163 8D15 49634000 LEA EDI,DWORD PTR DS:[406349]
```

Registers (FPU)  
EAX 0019FFCC  
ECX 00401000 crackme.<M  
EDX 00401000 crackme.<M  
EBX 00209000  
ESP 0019FF78  
EBP 0019FF84  
ESI 00401000 crackme.<M  
EDI 00401000 crackme.<M  
EIP 00401000 crackme.<M  
C 0 ES 002B 32bit 0(F  
P 1 CS 0023 32bit 0(F  
A 0 SS 002B 32bit 0(F  
Z 0 DS 002B 32bit 0(F  
S 0 FS 0053 32bit 21B0  
T 0 GS 002B 32bit 0(F  
D 0  
I 0 LastErr ERROR\_SUC  
EFL 00000246 (NO,NB,E,  
ST0 empty 0.0  
ST1 empty 0.0

Now here in you can see that it compares the Mem 00406549 = Mem 00406949 so what is going on is its jumps to 12D4 location which will give us output as a try again.

```
00401286 75 F0 JNZ SHORT crackme.00401278
00401288 68 49674000 PUSH crackme.00406749
0040128D 68 49654000 PUSH crackme.00406549
00401292 E8 5F010000 CALL <JMP.&kernel32.lstrcpA>
00401297 68 00020000 PUSH 200
0040129C 68 49694000 PUSH crackme.00406949
004012A1 6A 64 PUSH 64
004012A3 FF75 08 PUSH DWORD PTR SS:[EBP+8]
004012A6 E8 E5000000 CALL <JMP.&user32.GetDlgItemTextA>
004012AB 68 49654000 PUSH crackme.00406549
004012B0 68 49694000 PUSH crackme.00406949
004012B5 E8 36010000 CALL <JMP.&kernel32.lstrcpA>
004012BA 0BC0 OR EAX,EAX
004012BC EB 00 JMP SHORT crackme.004012BE
004012BE 6A 40 PUSH 40
004012C0 68 DB624000 PUSH crackme.004062DB
004012C5 68 AC624000 PUSH crackme.004062AC
004012CA FF75 08 PUSH DWORD PTR SS:[EBP+8]
004012CD E8 C8000000 CALL <JMP.&user32.MessageBoxA>
004012D2 EB 14 JMP SHORT crackme.004012E8
004012D4 6A 10 PUSH 10
004012D6 68 06634000 PUSH crackme.00406349
004012DB 68 E7624000 PUSH crackme.004062E7
```

String2 = ""  
String1 = crackme.00406549  
lstrcpA  
Count = 200 (512.)  
Buffer = crackme.00406949  
ControlID = 64 (100.)  
hWnd  
GetDlgItemTextA  
String2 = ""  
String1 = "1234"  
lstrcpA  
Style = MB\_OK|MB\_ICONASTERISK|MB\_APPLMODAL  
Title = "Good boy..."  
Text = "Yep, thats the right code! Go write a k  
hOwner  
MessageBoxA  
Style = MB\_OK|MB\_ICONHAND|MB\_APPLMODAL  
Title = "Bad boy..."  
Text = "Nope, thats not it! Try again"

Registers (FPU)  
EAX 00000000  
ECX 75891D80 user32.758  
EDX 00000000  
EBX 00218000  
ESP 0019FF78  
EBP 0019FF84  
ESI 00401000 crackme.<M  
EDI 006216A8  
EIP 004010A2 crackme.00  
C 0 ES 002B 32bit 0(F  
P 1 CS 0023 32bit 0(F  
A 0 SS 002B 32bit 0(F  
Z 1 DS 002B 32bit 0(F  
S 0 FS 0053 32bit 21B0  
T 0 GS 002B 32bit 0(F  
D 0  
I 0 LastErr ERROR\_SUCC  
EFL 00000246 (NO,NB,E,B  
ST0 empty -2560.0000000  
ST1 empty -1024.0000000

So what we will do is to modify the address and give it to a specific address which will give us successful output.

```
004012D6 E8 E5000000 CALL <JMP.&user32.GetDlgItemTextA>
0040128B 68 49654000 PUSH crackme.00406549
004012B0 68 49694000 PUSH crackme.00406949
004012B5 E8 36010000 CALL <JMP.&kernel32.lstrcpA>
004012BA 0BC0 OR EAX,EAX
004012BC 75 16 JNZ SHORT crackme.004012D4
004012BE 6A 40 PUSH 40
004012C0 68 DB624000 PUSH crackme.004062DB
004012C5 68 AC624000 PUSH crackme.004062AC
004012CA FF75 08 PUSH DWORD PTR SS:[EBP+8]
004012CD E8 C8000000 CALL <JMP.&user32.MessageBoxA>
004012D2 EB 14 JMP SHORT crackme.004012E8
004012D4 6A 10 PUSH 10
004012D6 68 06634000 PUSH crackme.00406349
004012DB 68 E7624000 PUSH crackme.004062E7
004012E0 FF75 08 PUSH DWORD PTR SS:[EBP+8]
004012E3 E8 B4000000 CALL <JMP.&user32.MessageBoxA>
004012E8 68 00020000 PUSH 200
004012ED 68 49654000 PUSH crackme.00406549
004012F2 E8 ED000000 CALL <JMP.&kernel32.RtlZeroMemory>
004012F7 68 00020000 PUSH 200
004012FC 68 49634000 PUSH crackme.00406349
00401301 E8 DE000000 CALL <JMP.&kernel32.RtlZeroMemory>
```

GetDlgItemTextA  
String2 = ""  
String1 = ""  
lstrcpA  
Style = MB\_OK|MB\_ICONASTERISK|MB\_APPLMODAL  
Title = "Good boy..."  
Text = "Yep, thats the right code! Go write a k  
hOwner  
MessageBoxA  
Style = MB\_OK|MB\_ICONHAND|MB\_APPLMODAL  
Title = "Bad boy..."  
Text = "Nope, thats not it! Try again"  
Length = 200 (512.)  
Destination = crackme.00406549  
RtlZeroMemory  
Length = 200 (512.)  
Destination = crackme.00406349  
RtlZeroMemory

Address Hex dump ASCII  
00406000 A2 68 3A FE 02 4E 83 oh:\*.i.Na  
00406008 E2 2F 37 90 AB 7C 88 0/7E%LZC  
00406010 61 23 C6 45 52 8C 84 6E a@aERi#n  
00406018 3F 4C 6D D1 55 99 92 56 7LIDU6v  
00406020 B5 95 ED B1 EB A5 8E 24 Aov@unA\$  
00406028 69 18 15 DF 1C DB D7 0F i\$N.ito  
00406030 E7 EC F9 5D AA 1E 47 9E tv-l.\*Gx  
00406038 5B 1A 78 D3 C7 B7 4D 41 f-xEaNA  
00406040 D6 94 D2 CE 3D 32 5C 1B i6E4-2+  
00406048 36 C8 C3 C1 77 7F AC 5F 61=-wX  
00406050 3C 19 1D 0A B2 8B 5B 26 <1=.iP8  
00406058 29 3B 64 6F 9B EE 59 72 );d8 Vv  
00406060 D0 40 F2 22 9A DA 71 44 s8="uqD  
00406068 86 F7 F8 AD C0 F0 DE 42 a-"i-lB  
00406070 2A 96 C2 6A BE 8A B0 76 =u-ivv

0019FF78 755E6739 RETURN to KERNEL32.755E6739  
0019FF7C 00218000  
0019FF80 755E6720 KERNEL32.BaseThreadInitThunk  
0019FF84 0019FFDC  
0019FF88 77008FD2 RETURN to ntdll.77008FD2  
0019FF8C 00218000  
0019FF90 0FC18FA  
0019FF94 00000000  
0019FF98 00000000  
0019FF9C 00218000  
0019FFA0 00000000  
0019FFA4 00000000  
0019FFA8 00000000  
0019FFAC 00000000  
0019FFB0 00000000  
0019FFB4 00000000



So when we execute the code again it will give us a dialogue box and when we write the username and Reg code it will give us a message box as that's the right code.

The screenshot shows the OllyDbg interface with the following components:

- Assembly View:** Disassembled code starting at address 0040105D. Instructions include `CALL <JMP.&kernel32.LockResource>`, `MOV EAX, 41634000`, `ADD EAX, 4`, `PUSH EAX`, `CALL <JMP.&kernel32.GlobalAlloc>`, `MOV DWORD PTR DS:[40633D], EAX`, `MOV ECX, DWORD PTR DS:[406341]`, `ADD EAX, 4`, `MOV EDI, EAX`, `REP MOVSB BYTE PTR ES:[EDI], BYTE PTR DS:[EAX]`, `POP ESI`, `PUSH 0`, `PUSH crackme.004010B4`, `PUSH 0`, `PUSH crackme.00406200`, `PUSH DWORD PTR DS:[406206]`, `CALL <JMP.&user32.MessageBoxParam>`, `PUSH DWORD PTR DS:[40633D]`, `CALL <JMP.&kernel32.GlobalFree>`, and `PUSH 0`.
- Registers (FPU):** Shows the state of various registers, including `EAX: 00000000`, `ECX: 00000000`, `EDX: 00000000`, `ESP: 0019F49C`, `EBP: 0019F4F0`, `ESI: 00659800`, `EIP: 00659498`, and `EIP: 77014E0C` (ntdll.77014E0C).
- Memory Dump:** Shows a hex dump of memory starting at address 00406000. The ASCII column shows the string `oh: *i. Nâ`.
- Dialog Box:** A small dialog box titled "Lafarge's crackme #2" is visible in the center. It contains the text "Good boy..." and "Yep, that's the right code! Go write a keygen!" with an "OK" button.