

jz_crackme1 solution

by skalk

Tools: IDA, Ollydbg

Sorry for PDF but I had to include some graph routine from IDA...

1. First of all run crackme, write any string (eg "anystring") and we see badboy: "Wrong!"
So Open Ollydbg -> "Search for all referenced text strings" and we see

Address=0040146A Text string=ASCII "wrong!\n"

Now Ctrl+G, 0040146A, and
we are at

```
0040146A . 68 58104100    PUSH jz_crack.00411058                ; ASCII "wrong!\n"
```

That's out badboy message.

Few lines before we have a dialog which gets password:

```
00401332 . 68 64104100    PUSH jz_crack.00411064                ; ASCII "input password : "  
00401337 . E8 B70C0000    CALL jz_crack.00401FF3                ;  
0040133C . 83C4 04        ADD ESP,4  
0040133F . 8D85 E0FAFFFF  LEA EAX,DWORD PTR SS:[EBP-520]  
00401345 . 50            PUSH EAX  
00401346 . 68 60104100    PUSH jz_crack.00411060                ; ASCII "%s"  
0040134B . E8 8C0C0000    CALL jz_crack.00401FDC                ;
```

And later starting with 0040140E there's a subroutine which calculates some checksum

```
0040140E . 52            PUSH EDX                               ; /Arg2 = 00000009  
0040140F . 8D85 E0FAFFFF  LEA EAX,DWORD PTR SS:[EBP-520]  
00401415 . 50            PUSH EAX                               ; |Arg1 = 0012FA60 ASCII  
"anystring"  
00401416 . E8 A5000000    CALL jz_crack.004014C0                ; \jz_crack.004014C0
```

and then checksum is compared with 0xAF006DC3 and if equal then good boy, if not - badboy...

```
0040145E . 81BD A8FAFFFF  C36>CMP DWORD PTR SS:[EBP-558],AF006DC3  
00401468 . 74 0F         JE SHORT jz_crack.00401479  
0040146A . 68 58104100    PUSH jz_crack.00411058                ; ASCII "wrong!\n"  
0040146F . E8 7F0B0000    CALL jz_crack.00401FF3                ;
```

I have to say that there's also some kind of protection (I assume anti-debug), which on my system didn't work ...

```
00401452 . 330D E0414100  XOR ECX,DWORD PTR DS:[4141E0]
```

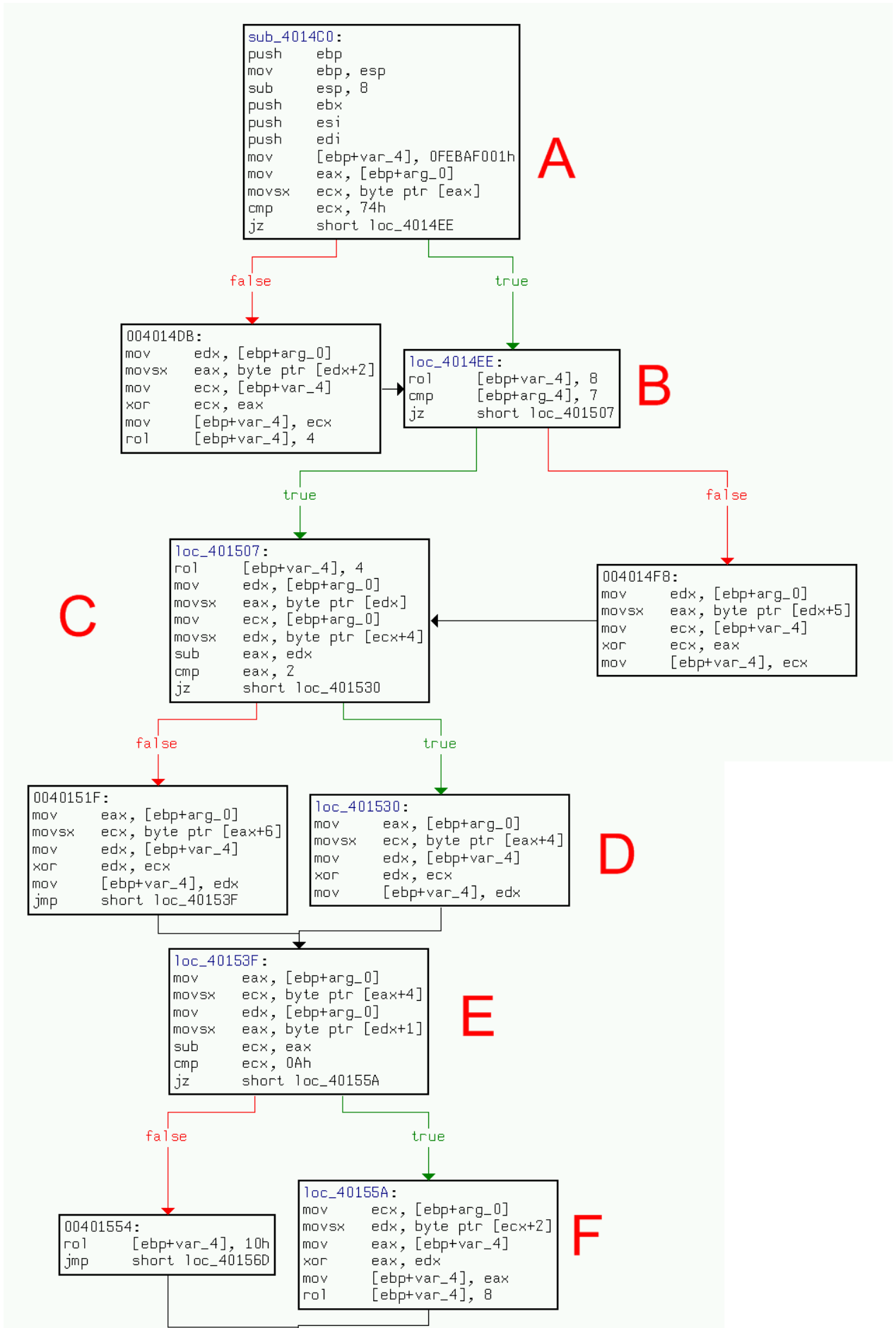
to DWORD PTR DS:[4141E0] is somewhere written something in the code ... in my case it was always 0 (0 XOR checksum = checksum) so I didn't bother with that ...

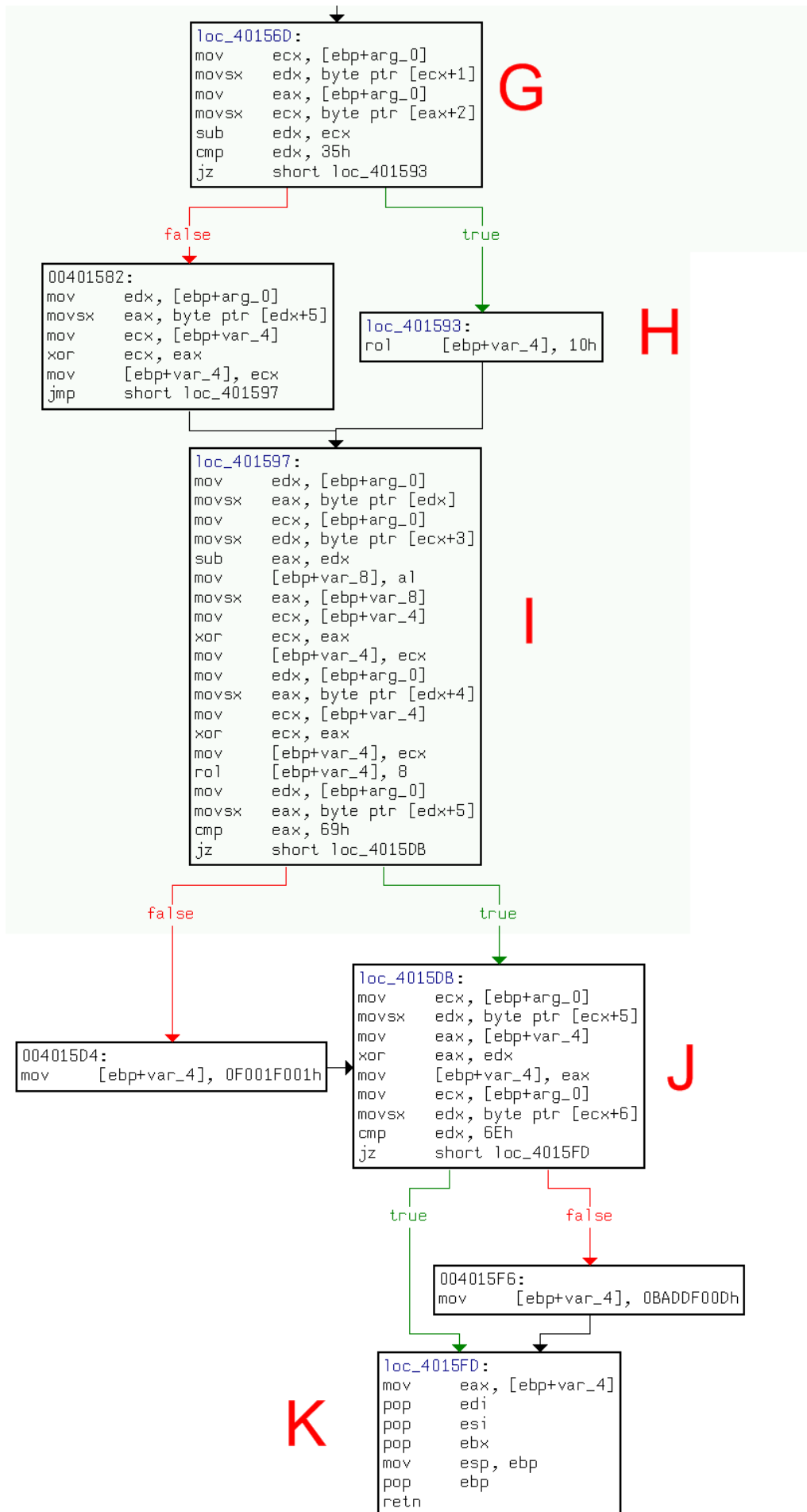
2. Let's analyse routine which calculates our checksum:

We see that it takes as parameters password and its length ...

```
0040140E . 52            PUSH EDX                               ; /Arg2 = 00000009  
0040140F . 8D85 E0FAFFFF  LEA EAX,DWORD PTR SS:[EBP-520]  
00401415 . 50            PUSH EAX                               ; |Arg1 = 0012FA60 ASCII "anystring"  
00401416 . E8 A5000000    CALL jz_crack.004014C0                ; \jz_crack.004014C0
```

I like to analyse algos in IDA, because there is nice routine graph viewer so we get graph {scroll} down ...





I will write here what's going on in the algo , but only most important things...

First of all I assumed (and that was correct assumption) that after all parts of algo we go through conditions to TRUE , so:

| Algorithm section | Whats going on | conclusions | Decoded password |
|-------------------|--|---|------------------|
| A | Checksum = 0xFEBAF001; if 1st_letter_of_password= 0x74 ('t') then True and go to B; | First letter is 0x78't' Checksum (dword[ebp+var4]):0xFEBAF001 | t |
| B | Roll left Checksum by 8 bits XXXXXXXX; If length of password=7 Then True and go to C | Password length=7 Checksum (dword[ebp+var4]):0xBAF001FE | t..... |
| C | Roll left Checksum by 4 bits XXXXXXXX; If 1st_letter_of_password MINUS 5th_letter_of_password =2 Then True and go to D | (0x74 - x = 2) so Fifth letter is 0x72 = 'r' Checksum (dword[ebp+var4]):0xAF001FEB | t...r.. |
| D | Get 5th_letter_of_password and xor it with Checksum ; store checksum ; | Checksum (dword[ebp+var4]):0xAF001F99 | |
| E | If 5th_letter_of_password MINUS 2th_letter_of_password= 0x0A Then TRUE and go to F | 0x72-0x0A=0x68 so 2nd letter is 'h' | th...r.. |
| F | Get 3rd_letter_of_password and xor it with Checksum , store checksum | we dont know 3 rd letter so checksum cant't be decoded now but after part G of algo and running crackme with partially decoded password we 'll have Checksum (dword[ebp+var4]):0x001FD8AF | |
| G | If 2nd_letter_of_password MINUS 3rd_letter_of_password= 0x35 Then True and go to H | 0x68-0x35=0x33 ='3', so 3 rd letter is '3' | th3...r.. |
| H | Roll left Checksum by 0x10 ; | Checksum (dword[ebp+var4]):0xAAF001F | |
| I | Now something like: x= 4th - 1st_letter_of_password ; checksum = x XOR checksum; checksum = 5th_letter_of_password XOR checksum; Roll left Checksum by 0x08 ; If 6th_letter_of_password= 0x69 ('i') Then True and go to J | 6th letter is 0x69 : 'i' | th3...ri.. |
| J | Checksum = 6th_letter_of_password XOR checksum If 7th_letter_of_password= 0x6E ('n') Then True and go to end. | 7th_letter_of_password = 'n' | th3...rin |

Now, we still dont know 4th letter .
To find out it we have to assume checksum result after this routine as 0xAF006DC3 which comes from comparing bad/good boy...

```
-> 0040145E . 81BD A8FAFFFF C36>CMP DWORD PTR SS:[EBP-558],AF006DC3
```

And then reverse routine until part „I” taking all xor-s ... and changing rols to rors

```
As w know xor has
if          A XOR B = C
then       C XOR A = B
and also   C XOR B = A
```

So when reversing we have

```
J) 0xAF006DC3 XOR 6th_letter_of_password = 0xAF006DC3 XOR 0x69 = 0xAF006DAA
I) ROR 0xAF006DAA by 0x08 = 0xAAF006D
   0xAAF006D XOR 5th_letter_of_password = 0xAAF006D XOR 0x72 = 0xAAF001F
```

```
And now we have
checksum after part H = 0xAAF001F
```

```
so looking at part I we have
```

```
0xAAF001F XOR (4th - 2nd_letter_of_password) = 0xAAF001F
```

```
it means that
4th - 1st_letter_of_password = 0
```

```
and that means that
```

```
4th letter = 1st letter = 't'
```

PASSWORD IS 'th3trn'

greetz
skalk