

Ida Pro

1. program3

2. c program

The c program does the exact same thing as program3, except that it is written in c, so it will look different compiled.

Program 3 thorough explanation:

Here is the source code, for reference purposes.

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
using namespace std;
int main() {
    string m;
    char c;

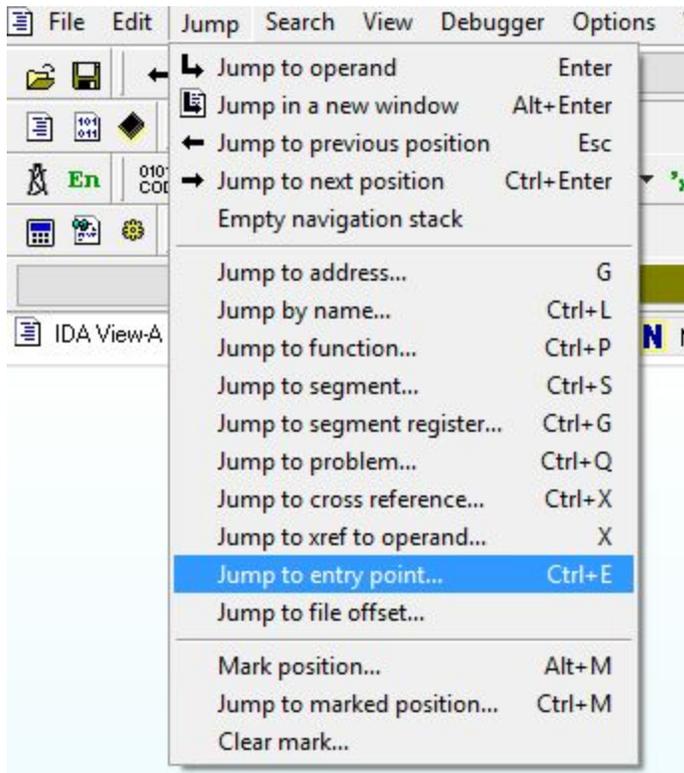
    ifstream f("example");
    getline (f,m);
    char k = 's';
    f.close();
    for (int t = 0; t < m.size(); t++)
        m[t] ^= k;

    string o = "bitsadmin.exe /transfer 'JobName' ";
    o.append(m);
    o.append(" C:/Users/student/Desktop/code.txt");

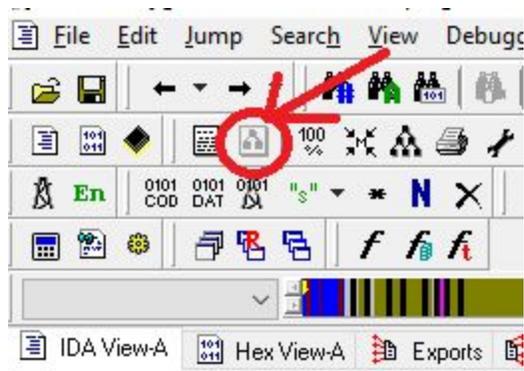
    cout << o;
    system(o.c_str());
    cin >> c;

    return 0;
}
```

1) Go to entry point.



You can change to **graph view** (the blue sections in the bar at the top) or you can also see the plain assembly code from top to bottom. You will spend most of the time in IDA-View-A.

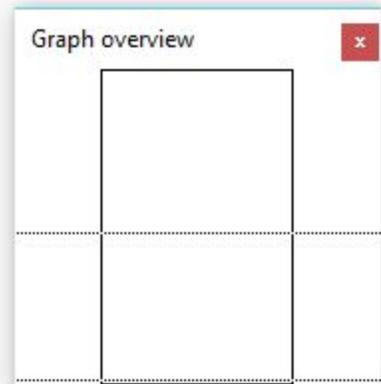


2) Go to the main function by double clicking on the first function that will run. In this case, double click `sub_4011D0`. Sub refers to the user defined functions, so there is only one function in this program. The null function calls are used by the compiler for garbage collection, debug functions, etc.

```
public start
start proc near

var_18= dword ptr -18h
var_14= dword ptr -14h
var_10= dword ptr -10h

push    ebp
mov     ebp, esp
sub     esp, 18h
and     esp, 0FFFFFF0h
mov     [esp+18h+var_18], offset sub_4011D0
call    sub_4014E0
mov     [esp+18h+var_10], 0
mov     [esp+18h+var_14], 0
mov     [esp+18h+var_18], 0
call    nullsub_1
mov     [esp+18h+var_10], 0
mov     [esp+18h+var_14], 0
mov     [esp+18h+var_18], 0
call    nullsub_2
mov     [esp+18h+var_10], 0
mov     [esp+18h+var_14], 0
mov     [esp+18h+var_18], 0
call    nullsub_3
mov     [esp+18h+var_10], 0
mov     [esp+18h+var_14], 0
mov     [esp+18h+var_18], 0
call    nullsub_4
leave
retn
start endp
```



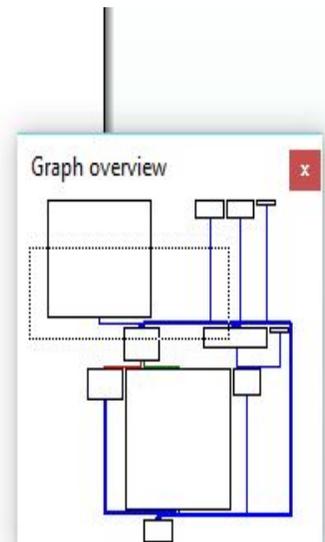
3) Once in main, you can see the allocation for a char pointer, and the string m which we can see is `var_10`. The first function call is a generic function by the compiler to declare the string.

The second call is `basic ifstream`. A quick google search

(http://en.cppreference.com/w/cpp/io/basic_ifstream) tells us this:

1. This is a cpp generated file and now we know the language this was written in.
2. This program is opening a file, since the string “example” was loaded into the `eax`, this is the file name.
3. It gets exactly 1 line from the file, then closes it. (the `getline` function call).

```
push    ebp
mov     ebp, esp
push    ebx
and     esp, 0FFFFFF0h
sub     esp, 130h      ; char *
call    __main
lea     eax, [esp+134h+var_10]
mov     [esp+134h+var_134], eax
call    __ZNStC1Ev
mov     [esp+134h+var_12C], 8
mov     [esp+134h+var_130], offset aExample ; "example"
lea     eax, [esp+134h+var_118]
mov     [esp+134h+var_134], eax
call    __ZNSt14basic_ifstreamIcSt11char_traitsIcEEC1EPKcSt13_ Ios_0penmode
lea     eax, [esp+134h+var_10]
mov     [esp+134h+var_130], eax
lea     eax, [esp+134h+var_118]
mov     [esp+134h+var_134], eax
call    __ZSt7getlineIcSt11char_traitsIcESaIcEEERSt13basic_istreamIT_T0_ES7_RSbIS4_S5_T1_E
mov     [esp+134h+var_9], 73h
lea     eax, [esp+134h+var_118]
mov     [esp+134h+var_134], eax
call    __ZNSt14basic_ifstreamIcSt11char_traitsIcEE5closeEv
mov     [esp+134h+var_8], 0
```

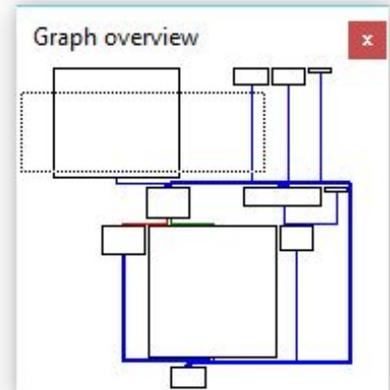


loc 401243:

4) The line gets saved to `var_10`, a string.

```
var_12C= dword ptr -12Ch
var_11C= dword ptr -11Ch
var_118= dword ptr -118h
var_10= dword ptr -10h
var_9= byte ptr -9
var_8= dword ptr -8
var_4= dword ptr -4

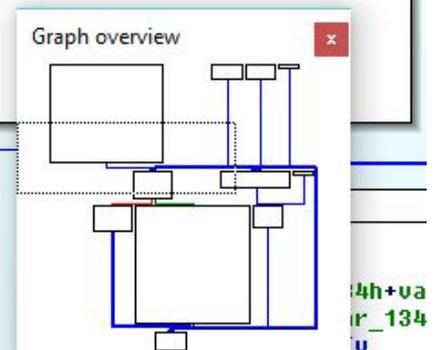
push    ebp
mov     ebp, esp
push    ebx
and     esp, 0FFFFFF0h
sub     esp, 130h      ; char *
call    __main
lea     eax, [esp+134h+var_10]
mov     [esp+134h+var_134], eax
call    _ZN5Sc1Ev
mov     [esp+134h+var_12C], 8
mov     [esp+134h+var_130], offset aExample ; "example"
lea     eax, [esp+134h+var_118]
mov     [esp+134h+var_134], eax
call    _ZNSt14basic_ifstreamIcSt11char_traitsIcEEC1EPKcSt13_Ios_Openmode
lea     eax, [esp+134h+var_10]
mov     [esp+134h+var_130], eax
lea     eax, [esp+134h+var_118]
mov     [esp+134h+var_134], eax
call    _ZSt7getlineIcSt11char_traitsIcESaIcEESt13basic_istreamIT_T0_ES7_RSbIS4_S5
mov     [esp+134h+var_9], 73h
lea     eax, [esp+134h+var_118]
mov     [esp+134h+var_134], eax
```



5) After the getline function, the hex value 73h was moved to `var_9`. 73h is the letter s.

After the file was closed, a variable `var_8` (an int) was set to 0. It was compared over and over (when following the arrows in the graph), we can safely assume this is a while or for loop. (See the add 1 to `var_8`).

```
call    _ZSt7getlineIcSt11char_traitsIcESaIcEESt13basic_istreamIT_T0_ES7_RSbIS4_S5_T1_E
mov     [esp+134h+var_9], 73h
lea     eax, [esp+134h+var_118]
mov     [esp+134h+var_134], eax
call    _ZNSt14basic_ifstreamIcSt11char_traitsIcEE5closeEv
mov     [esp+134h+var_8], 0
```



6) The loop takes the string's size, compares it with var_8 (look closely at the function call, you can see it is **size**). **Red** arrows mean failure, so it loops again. **Green** means success.

If you follow the **red path**, you will see **var_10**, the string, being **xor**'ed with the letter s, byte by byte (byte pointer),

Following the blue arrow after that leads back to the loop.

Graph overview

```
loc_401243:  
lea    eax, [esp+134h+var_10]  
mov    [esp+134h+var_134], eax  
call   _ZNKs4sizeEv  
mov    edx, eax  
mov    eax, [esp+134h+var_8]  
cmp    edx, eax  
setnbe al  
test   al, al  
jz     short loc_401294
```

```
mov    eax, [esp+134h+var_8]  
mov    [esp+134h+var_130], eax  
lea    eax, [esp+134h+var_10]  
mov    [esp+134h+var_134], eax  
call   _ZNs6ixEj  
movzx  edx, byte ptr [eax]  
xor    dl, [esp+134h+var_9]  
mov    [eax], dl  
add    [esp+134h+var_8], 1  
jmp    short loc_401243
```

```
loc_401294:  
lea    eax, [esp+12Ah]  
mov    [esp+134h+var_134], eax  
call   _ZNSaIcEC1Ev  
lea    eax, [esp+12Ah]  
mov    [esp+134h+var_12C], eax  
mov    [esp+134h+var_130], offset aBitsadmin_ex  
lea    eax, [esp+134h+var_11C]  
mov    [esp+134h+var_134], eax  
call   _ZNSsC1EPKcRKSaIcE
```

7) **Green path**, you can see lots of things happen:

1. A long string (**var_11C**) containing the command “bitsadmin.exe” ([https://msdn.microsoft.com/en-us/library/aa362813\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa362813(v=vs.85).aspx))
2. The var_10 string was appended to it.
3. Another string with a “location” was appended as well.

```
setnbe al
test  al, al
jz    short loc_401294

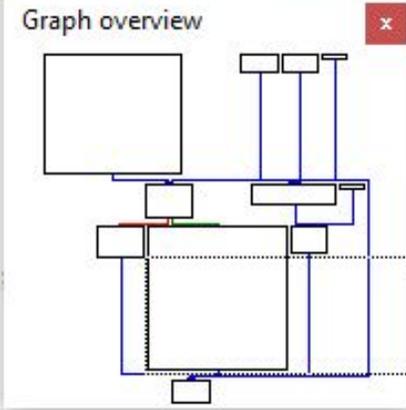
loc_401294:
lea  eax, [esp+12Ah]
mov  [esp+134h+var_134], eax
call _ZNSaIcEC1Ev
lea  eax, [esp+12Ah]
mov  [esp+134h+var_12C], eax
mov  [esp+134h+var_130], offset aBitsadmin_exeT ; "bitsadmin.exe /transfer 'JobName' "
lea  eax, [esp+134h+var_11C]
mov  [esp+134h+var_134], eax
call _ZNSsC1EPKcRKSaIcE
lea  eax, [esp+12Ah]
mov  [esp+134h+var_134], eax
call _ZNSaIcED1Ev
lea  eax, [esp+134h+var_10]
mov  [esp+134h+var_130], eax
lea  eax, [esp+134h+var_11C]
mov  [esp+134h+var_134], eax
call _ZNSs6appendERKSS
mov  [esp+134h+var_130], offset aCUsersStudentD ; " C:/Users/student/Desktop/code.txt"
lea  eax, [esp+134h+var_11C]
mov  [esp+134h+var_134], eax
call _ZNSs6appendEPKc
lea  eax, [esp+134h+var_11C]
mov  [esp+134h+var_130], eax
```

You can see the 2 **append** functions:

```
loc_401294:
lea  eax, [esp+12Ah]
mov  [esp+134h+var_134], eax
call _ZNSaIcEC1Ev
lea  eax, [esp+12Ah]
mov  [esp+134h+var_12C], eax
mov  [esp+134h+var_130], offset aBitsadmin_exeT ; "bitsadmin.exe /transfer 'JobName' "
lea  eax, [esp+134h+var_11C]
mov  [esp+134h+var_134], eax
call _ZNSsC1EPKcRKSaIcE
lea  eax, [esp+12Ah]
mov  [esp+134h+var_134], eax
call _ZNSaIcED1Ev
lea  eax, [esp+134h+var_10]
mov  [esp+134h+var_130], eax
lea  eax, [esp+134h+var_11C]
mov  [esp+134h+var_134], eax
call _ZNSs6appendERKSS
mov  [esp+134h+var_130], offset aCUsersStudentD ; " C:/Users/student/Desktop/code.txt"
lea  eax, [esp+134h+var_11C]
mov  [esp+134h+var_134], eax
call _ZNSs6appendEPKc
lea  eax, [esp+134h+var_11C]
mov  [esp+134h+var_130], eax
```

8) The full string was printed (cout). Then it was converted into a pointer of the string:

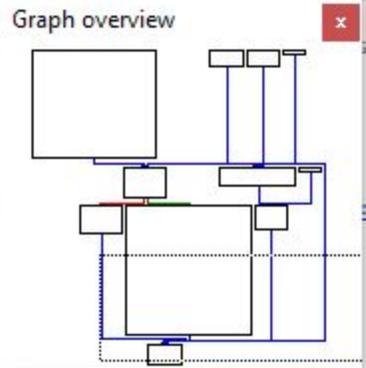
```
nov [esp+134h+var_134], eax
call _ZNSSc1EPKcRKSaIcE
lea  eax, [esp+12Ah]
nov  [esp+134h+var_134], eax
call _ZNSaIcE01Ev
lea  eax, [esp+134h+var_10]
nov  [esp+134h+var_130], eax
lea  eax, [esp+134h+var_11C]
nov  [esp+134h+var_134], eax
call _ZNSS6appendERKSS
nov  [esp+134h+var_130], offset aCUsersStudentD
lea  eax, [esp+134h+var_11C]
nov  [esp+134h+var_134], eax
call _ZNSS6appendEPKc
lea  eax, [esp+134h+var_11C]
nov  [esp+134h+var_130], eax
nov  [esp+134h+var_134], offset _ZSt4cout
call _ZSt15IcSt11char_traitsIcESaIcEERSt13basic_ostreamIT_0_ES7_RKSbIS4_S5_T1_E
lea  eax, [esp+134h+var_11C]
```



The diagram, titled "Graph overview", shows a control flow graph. A large box on the left represents the main code block. A smaller box on the right is labeled "code.txt". Arrows indicate the flow of execution between various points in the code, including a loop structure at the top right.

This string (var_11C)'s pointer was placed into a system call as a parameter,

```
nov [esp+134h+var_130], offset a
call _ZNSS6appendERKSS
nov  [esp+134h+var_130], offset a
lea  eax, [esp+134h+var_11C]
nov  [esp+134h+var_134], eax
call _ZNSS6appendEPKc
lea  eax, [esp+134h+var_11C]
nov  [esp+134h+var_130], eax
nov  [esp+134h+var_134], offset a
call _ZSt15IcSt11char_traitsIcESaIcE
lea  eax, [esp+134h+var_11C]
nov  [esp+134h+var_134], eax
call _ZNKSS5c_strEv
nov  [esp+134h+var_134], eax
call system
lea  eax, [esp+123h]
```



The diagram, titled "Graph overview", shows a control flow graph similar to the one above. It illustrates the flow of execution through various code blocks, with a call to the "system" function highlighted in the code. The graph shows a loop structure at the top right and a flow to a block labeled "code.txt".

9) After that the program pauses until user input (cin).

The program does nothing with the input, does memory clean up (unwind resume).

```
nov     ebx, 0
lea     eax, [esp+134h+var_11C]
mov     [esp+134h+var_134], eax
call    _ZNSSD1Ev
lea     eax, [esp+134h+var_118]
mov     [esp+134h+var_134], eax
call    _ZNSt14basic_ifstreamIcSt11char_traitsIcEED1Ev
lea     eax, [esp+134h+var_10]
mov     [esp+134h+var_134], eax
call    _ZNSSD1Ev
mov     eax, ebx
jmp     short loc_4013B9
```

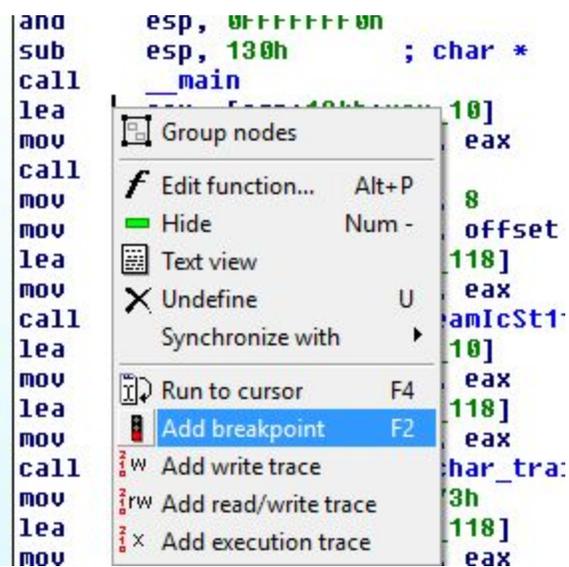
```
loc_4013B9:
mov     ebx, [ebp+var_4]
leave
retn
sub_4011D0 endp
```

```
loc_4013A0:
lea     eax, [esp+124h]
mov     [esp], eax
call    _ZNSSD1Ev
mov     eax, ebx
mov     [esp], eax
call    _Unwind_Resume
```

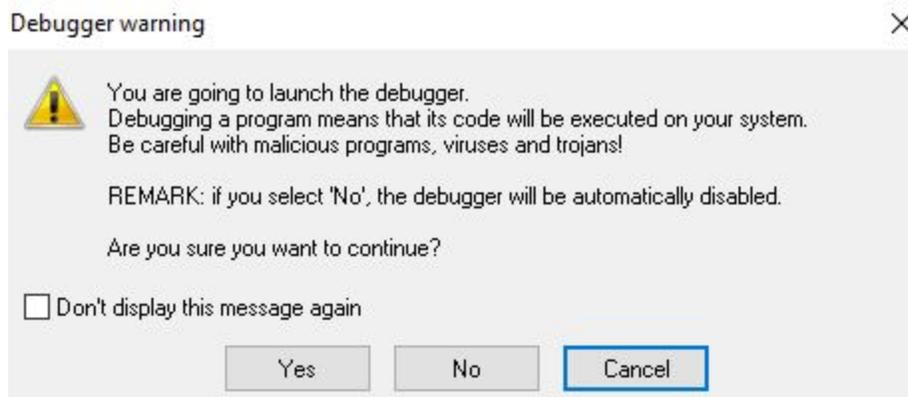
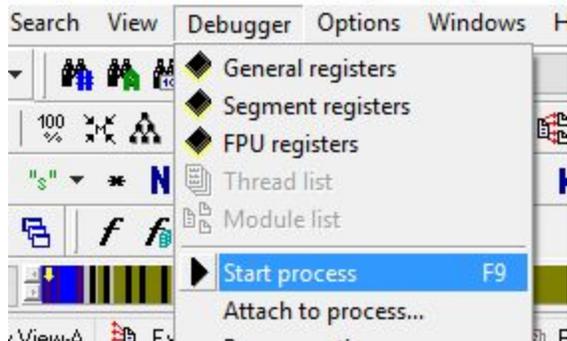
The program returns, runs the destructor functions, and exits.

Tracing with debugger:

Add a breakpoint right when main runs



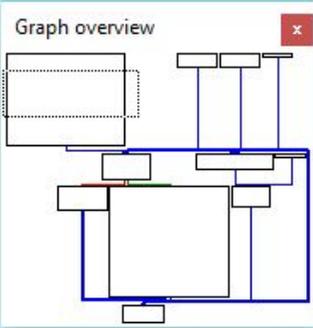
Go to start process and you will warnings, click yes and ok.



If you step into (F7) you can see the register values and a flow chart view of the code, you can disable the graph view by pressing spacebar.

IDA View-EIP

```
004011D0 var_134= dword ptr -134h
004011D0 var_130= dword ptr -130h
004011D0 var_12C= dword ptr -12Ch
004011D0 var_11C= dword ptr -11Ch
004011D0 var_118= dword ptr -118h
004011D0 var_10= dword ptr -10h
004011D0 var_9= byte ptr -9
004011D0 var_8= dword ptr -8
004011D0 var_4= dword ptr -4
004011D0
004011D0 push    ebp
004011D1 mov     ebp, esp
004011D3 push    ebx
004011D4 and     esp, 0FFFFFF0h
004011D7 sub     esp, 130h      ; char *
004011DD call   main
004011E2 lea    eax, [esp+134h+var_134]
004011E9 mov     [esp+134h+var_134], eax
004011EC call   _ZN5sC1Ev
004011F1 mov     [esp+134h+var_12C], 8
004011F9 mov     [esp+134h+var_130], offset aExample ; "example"
```



100.00% (-32,108) (880,43) 000005D4 004011D4: sub_4011D0+4

General registers

EAX	00000001		CF	0
EBX	0061CC5C	Stack[00001CFC]:0061CC5C	PF	1
ECX	00000000		AF	0
EDX	00000000		ZF	1
ESI	20041647	debug027:20041647	SF	0
EDI	611D3FB2	cygwin1.dll:611D3FB2	TF	0
EBP	0061CC38	Stack[00001CFC]:0061CC38	IF	1
ESP	0061CB00	Stack[00001CFC]:var_134	DF	0
EIP	004011E2	sub_4011D0+12	OF	0
EFL	00000246			

Threads

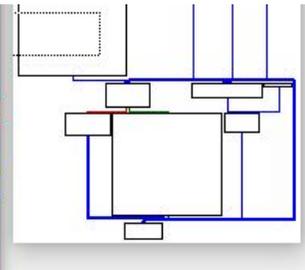
Edit Search

Thread

- 00001CFC
- 00002608

Make sure you press F8 to step over function calls, or else you will go into the functions the compiler did to add strings together, or open files, etc.

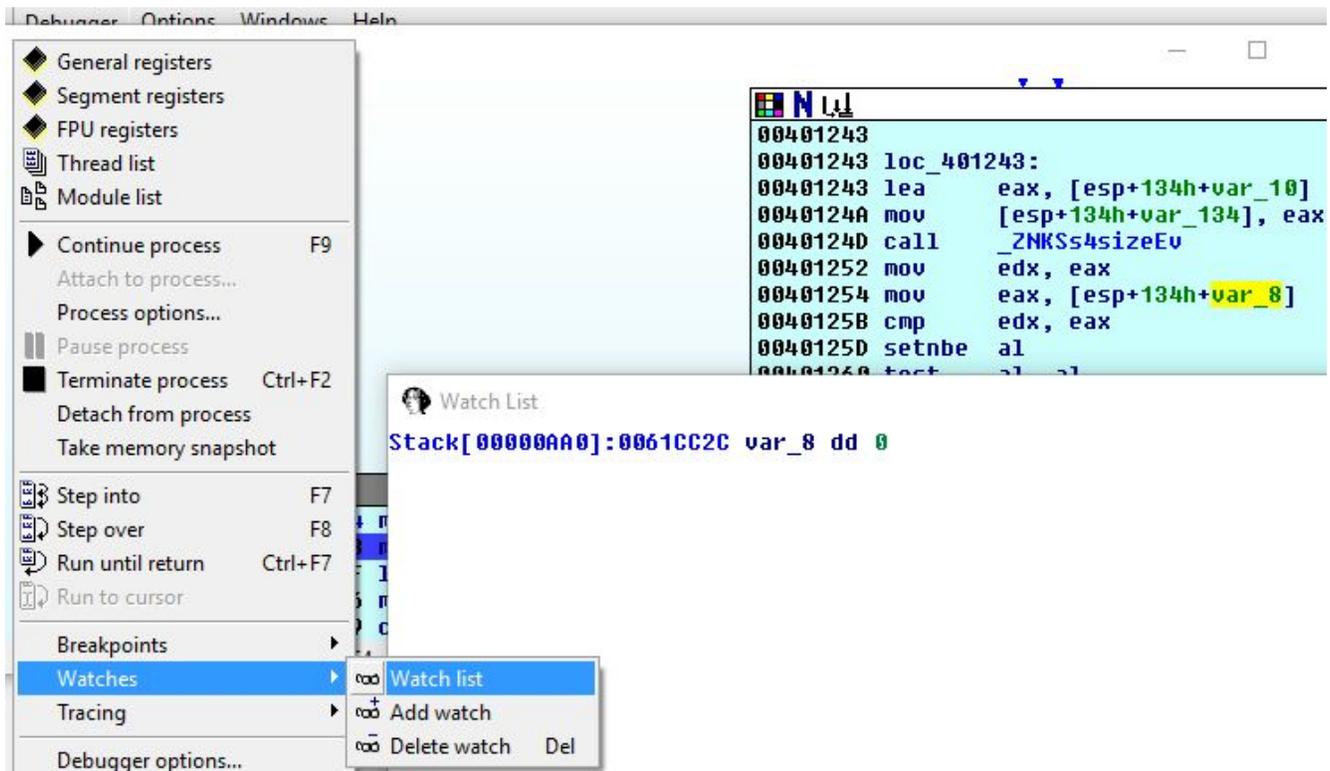
```
004011D3 push ebx
004011D4 and esp, 0FFFFFF0h
004011D7 sub esp, 130h ; char *
004011DD call main
004011E2 lea eax, [esp+130h+var_10]
004011E9 mov [esp+134h+var_134], eax
004011EC call _ZNSsC1Ev
004011F1 mov [esp+134h+var_12C], 0
004011F9 mov [esp+134h+var_130], offset aExample ; "example"
00401201 lea eax, [esp+134h+var_118]
00401205 mov [esp+134h+var_134], eax
```



You can add specific variables to the watch list and see how they change as the program runs. The variable var_8 was the counter for a loop, and you can see it increase by 1 each time.

var_10 was a string that was being xor'ed, you can see it change byte by byte.

var_11C was the string that was being appended, and you can see the values change as well.



C program

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int main(char *argc, char* argv[]){
    char buffer[200];
    buffer[0] = '\0';

    FILE* file = fopen("example", "r");
    char line[46];
    fgets(line, sizeof(line), file);
    line[45]='\0';

    char k = 's';
    fclose(file);

    int t;
    for (t = 0; t < sizeof(line)-1; t++){
        line[t] ^= k;
    }

    strcpy(buffer,"powershell Invoke-WebRequest ");
    strcat(buffer,line);

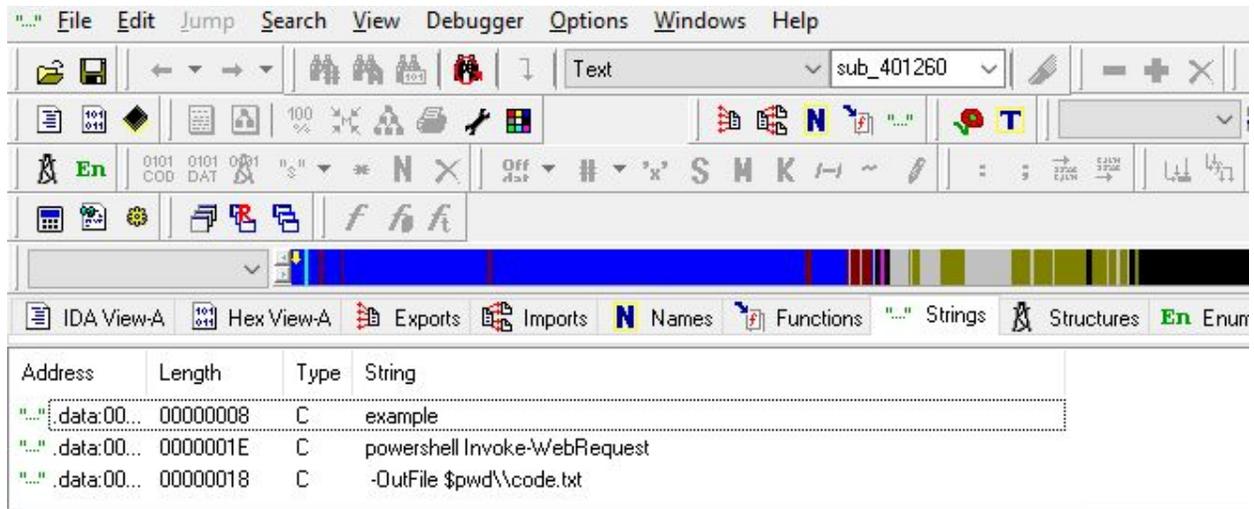
    strcat(buffer," -OutFile $pwd\code.txt");

    system(buffer);

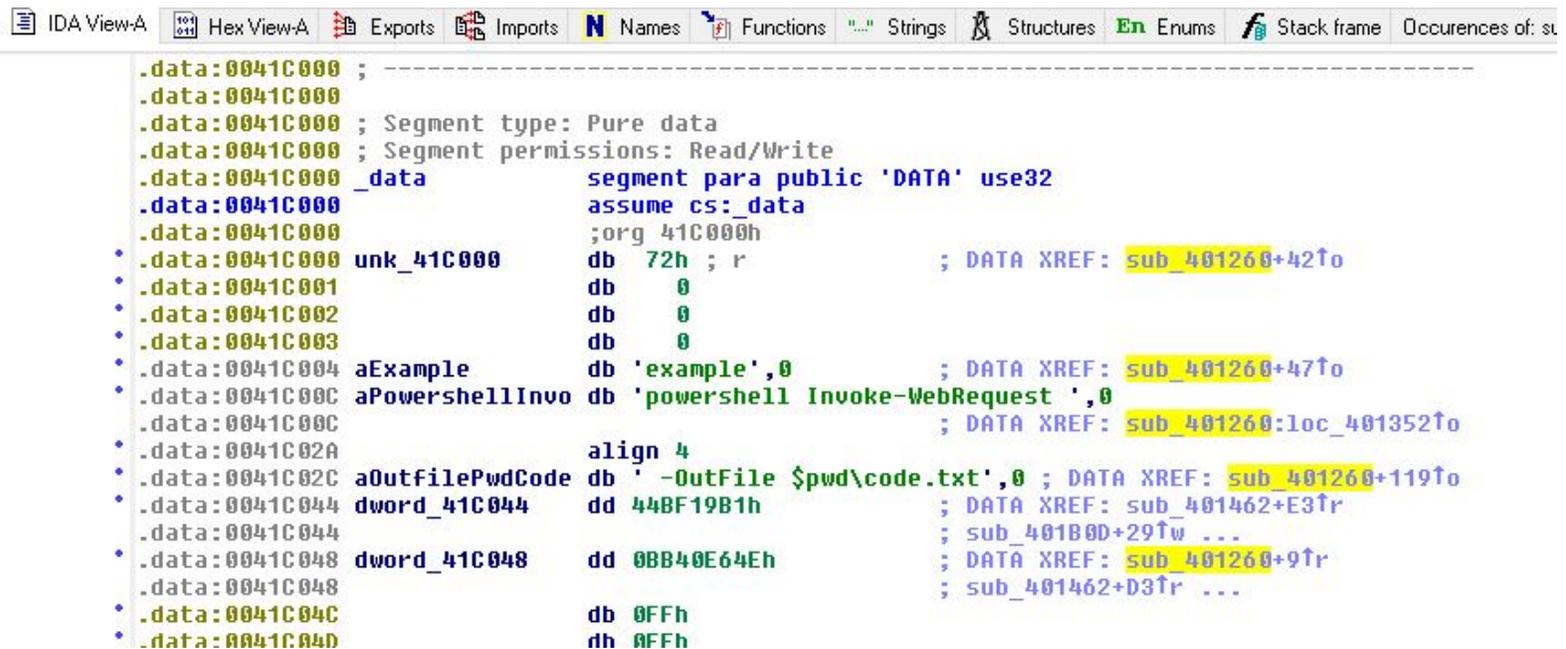
    printf(buffer);
    getch();

    return 0;
}
```

The first thing you should do is investigate the strings tab:



You can see where strings are used by double clicking:



Double clicking `sub_401260` will take you to the function where the strings are being used.

You can also see `sub_401260` being called when you “jump to entry point”.

Between `hModule` because it is using a Windows system call to call a powershell.

```
loc_4017DB:                ; hModule
push    edi
call    sub_402079
call    sub_405207
mov     edi, eax
call    sub_405201
mov     esi, eax
call    sub_404D6D
push    eax
push    dword ptr [edi]
push    dword ptr [esi]
call    sub_401260
mov     esi, eax
push    0                    ; hModule
call    sub_402115
```

It is the same as `program3`, in this case, the counter for the loop is `var_104`.

