

PE Code Injection

Implement your own Windows code to display simple Hello Text in the body of the Application. Now hijack control of this program and inject MessageBox code into it, Caption should show your Name_RollNumber and text in the message box should be "You have been Hacked". After the initial display of this MessageBox, retrieve back the original entry point. Create an answer document listing all the steps with appropriate screen shots.

A simple hello world windows program was written using codeblocks IDE in win xp.

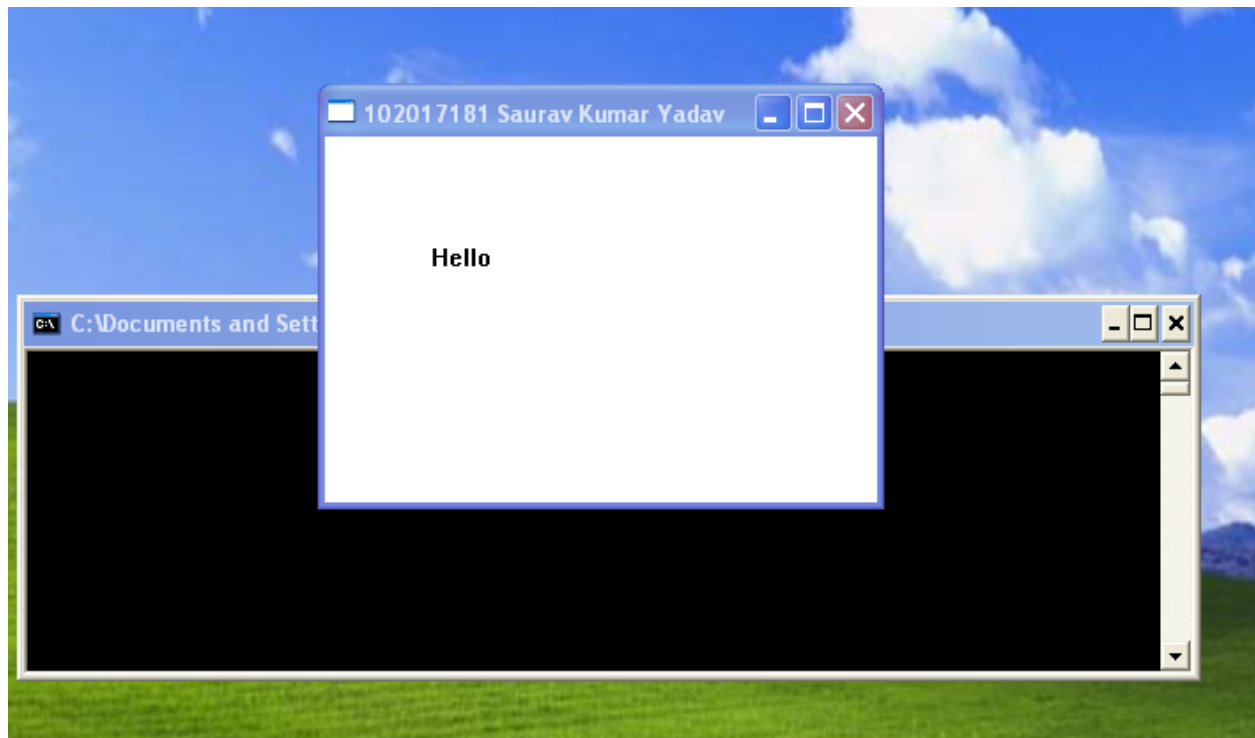
```
1  #include <windows.h>
2
3  LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
4
5  static char gszClassName[] = "102017181 Saurav Kumar Yadav";
6  static HINSTANCE ghInstance = NULL;
7
8  int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdS
9      WNDCLASSEX WndClass;
10     HWND hwnd;
11     MSG Msg;
12
13     ghInstance = hInstance;
14
15     WndClass.cbSize = sizeof(WNDCLASSEX);
16     WndClass.style = NULL;
17     WndClass.lpfnWndProc = WndProc;
18     WndClass.cbClsExtra = 0;
19     WndClass.cbWndExtra = 0;
20     WndClass.hInstance = ghInstance;
21     WndClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
22     WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
23     WndClass.hbrBackground = (HBRUSH) (COLOR_WINDOW+1);
24     WndClass.lpszMenuName = NULL;
25     WndClass.lpszClassName = gszClassName;
26     WndClass.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
27
28     if(!RegisterClassEx(&WndClass)) {
29         MessageBox(0, "Error Registering Window!", "Error!", MB_ICONSTOP | MB_OK);
30         return 0;
31     }
32
33     hwnd = CreateWindowEx(
34         WS_EX_STATICEDGE,
35         gszClassName,
36         "102017181 Saurav Kumar Yadav",
37         WS_OVERLAPPEDWINDOW,
38         CW_USEDEFAULT, CW_USEDEFAULT,
39         320, 240,
40         NULL, NULL,
41         ghInstance,
42         NULL);
```

```

40         NULL, NULL,
41         ghInstance,
42         NULL);
43
44     if (hwnd == NULL) {
45         MessageBox(0, "Window Creation Failed!", "Error!", MB_ICONSTOP | MB_OK);
46         return 0;
47     }
48
49     ShowWindow(hwnd, nCmdShow);
50     UpdateWindow(hwnd);
51
52     while (GetMessage(&Msg, NULL, 0, 0)) {
53         TranslateMessage(&Msg);
54         DispatchMessage(&Msg);
55     }
56     return Msg.wParam;
57 }
58
59 LRESULT CALLBACK WndProc(HWND hwnd, UINT Message, WPARAM wParam, LPARAM lParam) {
60     HDC hdc;
61     PAINTSTRUCT ps;
62     LPSTR szMessage = "Hello!";
63
64     switch (Message) {
65         case WM_PAINT:
66             hdc = BeginPaint(hwnd, &ps);
67             TextOut(hdc, 70, 50, szMessage, strlen(szMessage));
68             EndPaint(hwnd, &ps);
69             break;
70         case WM_CLOSE:
71             DestroyWindow(hwnd);
72             break;
73         case WM_DESTROY:
74             PostQuitMessage(0);
75             break;
76         default:
77             return DefWindowProc(hwnd, Message, wParam, lParam);
78     }
79     return 0;
80 }
81

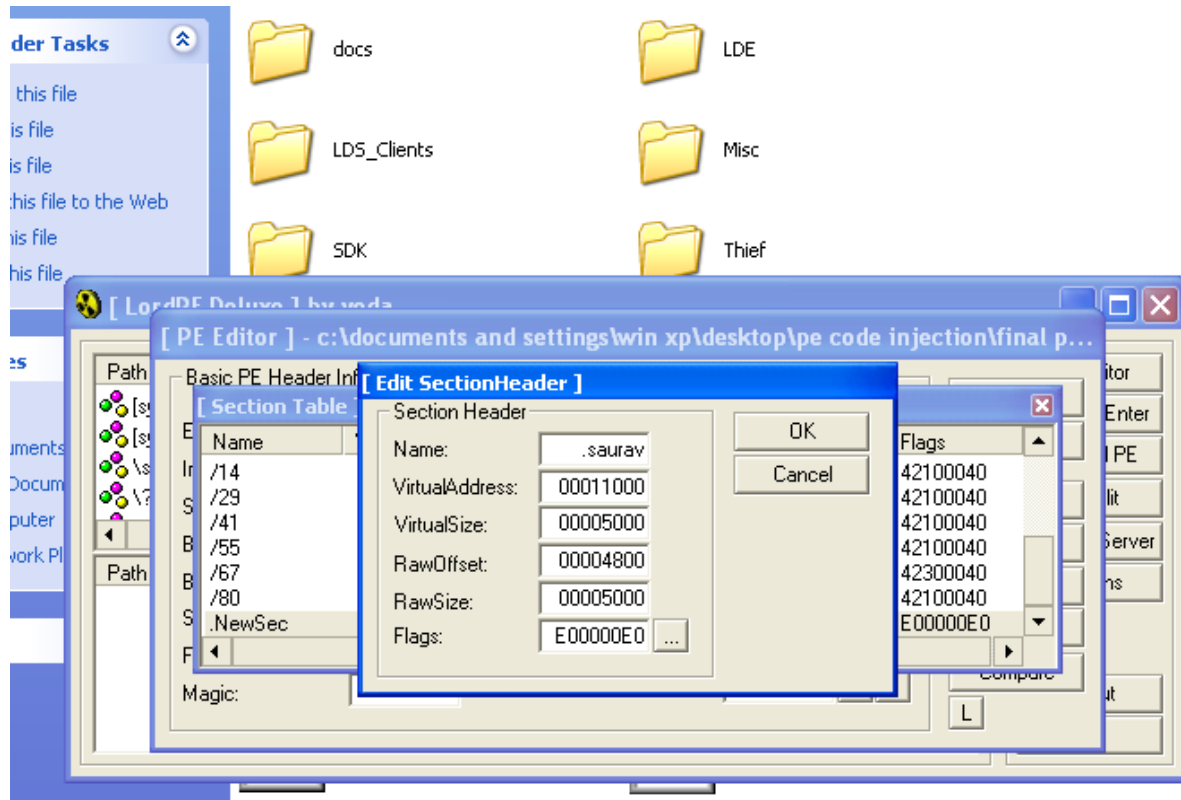
```

The app displays following window

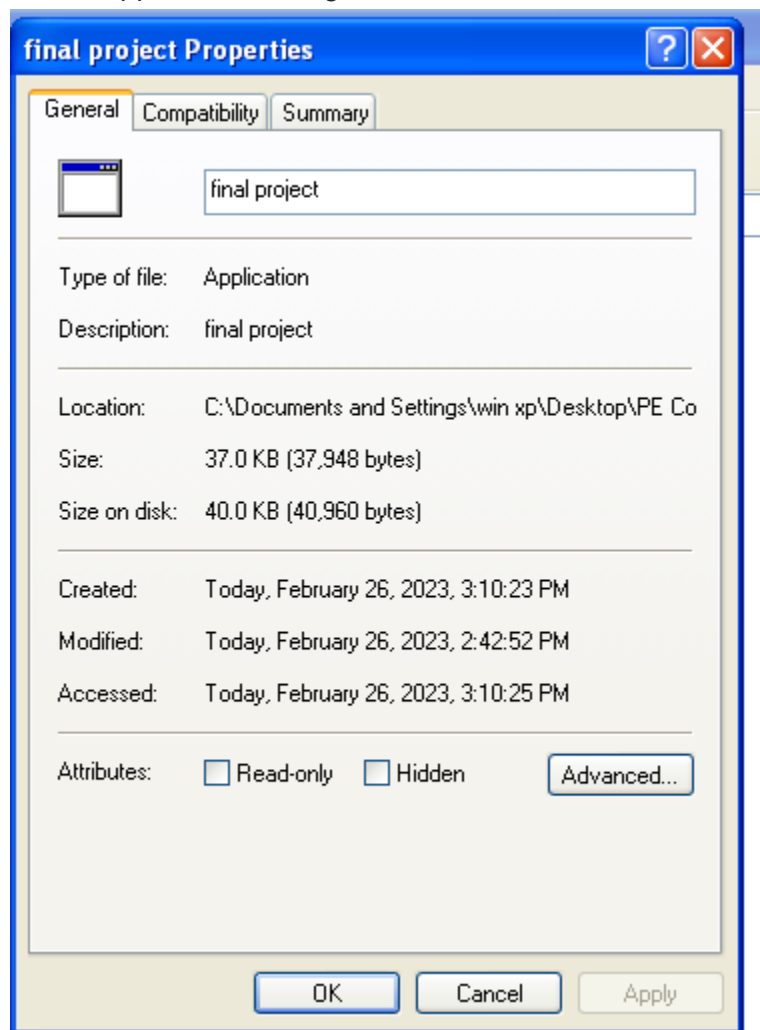


Now we implement PE code Injection .The steps are as follows:

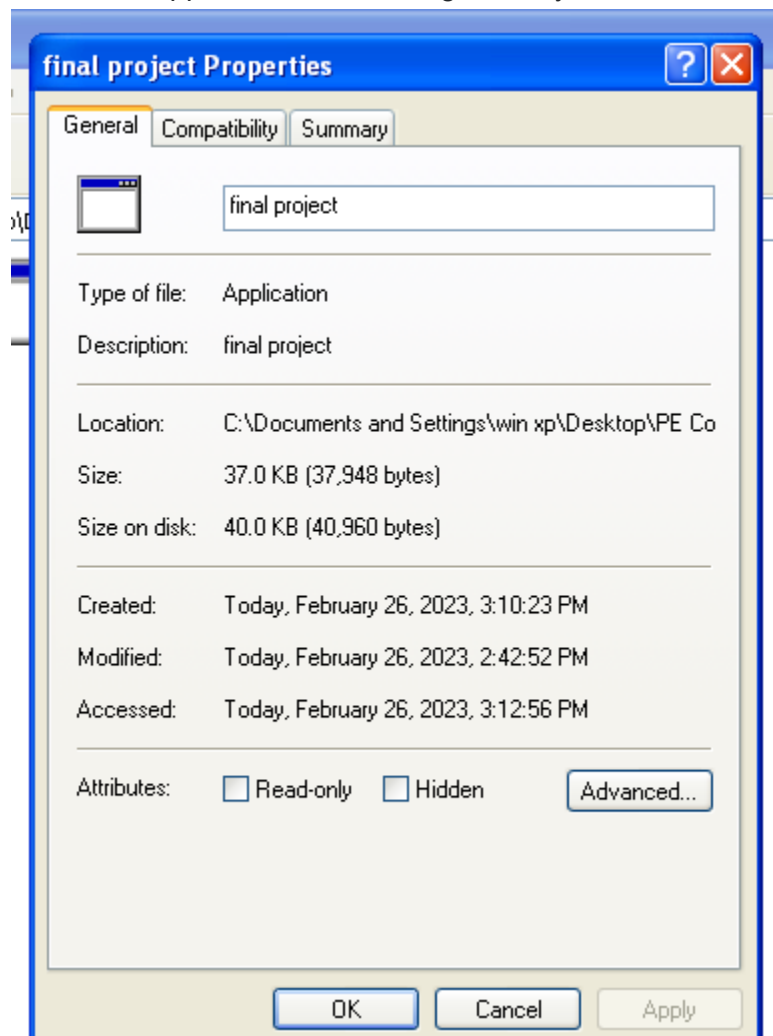
Using LordPE a new section of 5000 bytes is added to the above program.



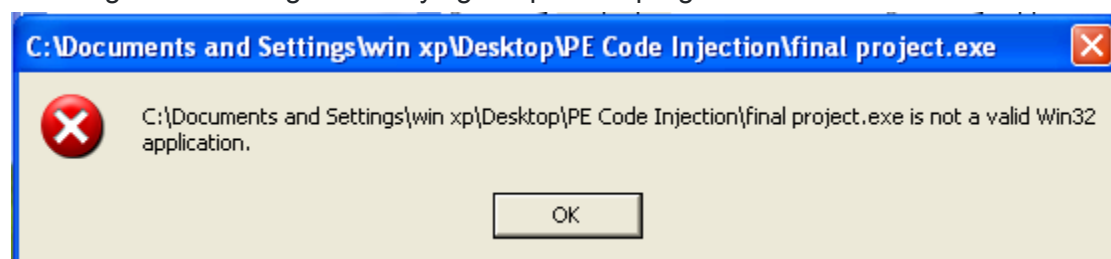
Size of app before adding a new section:



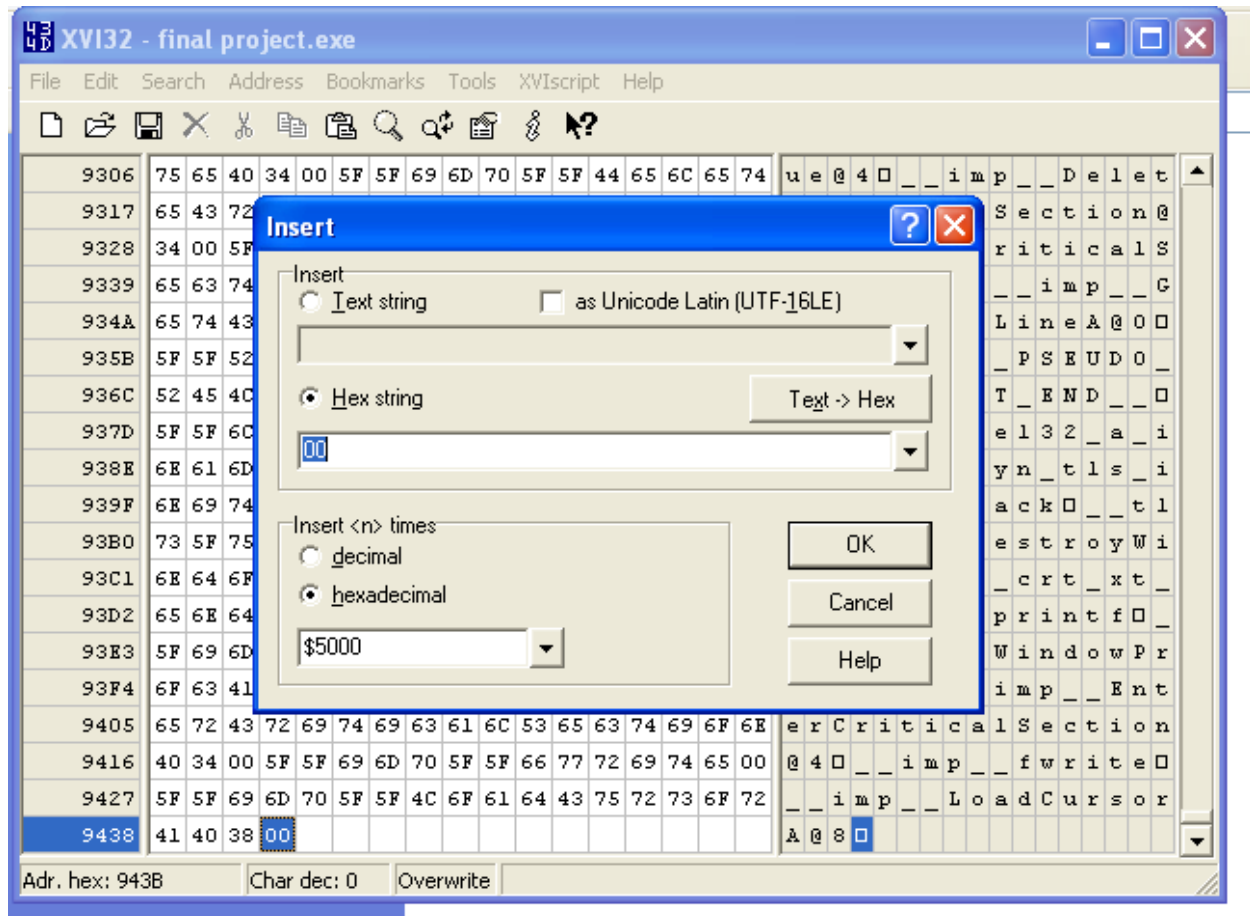
Size of the application after adding 5000 bytes:



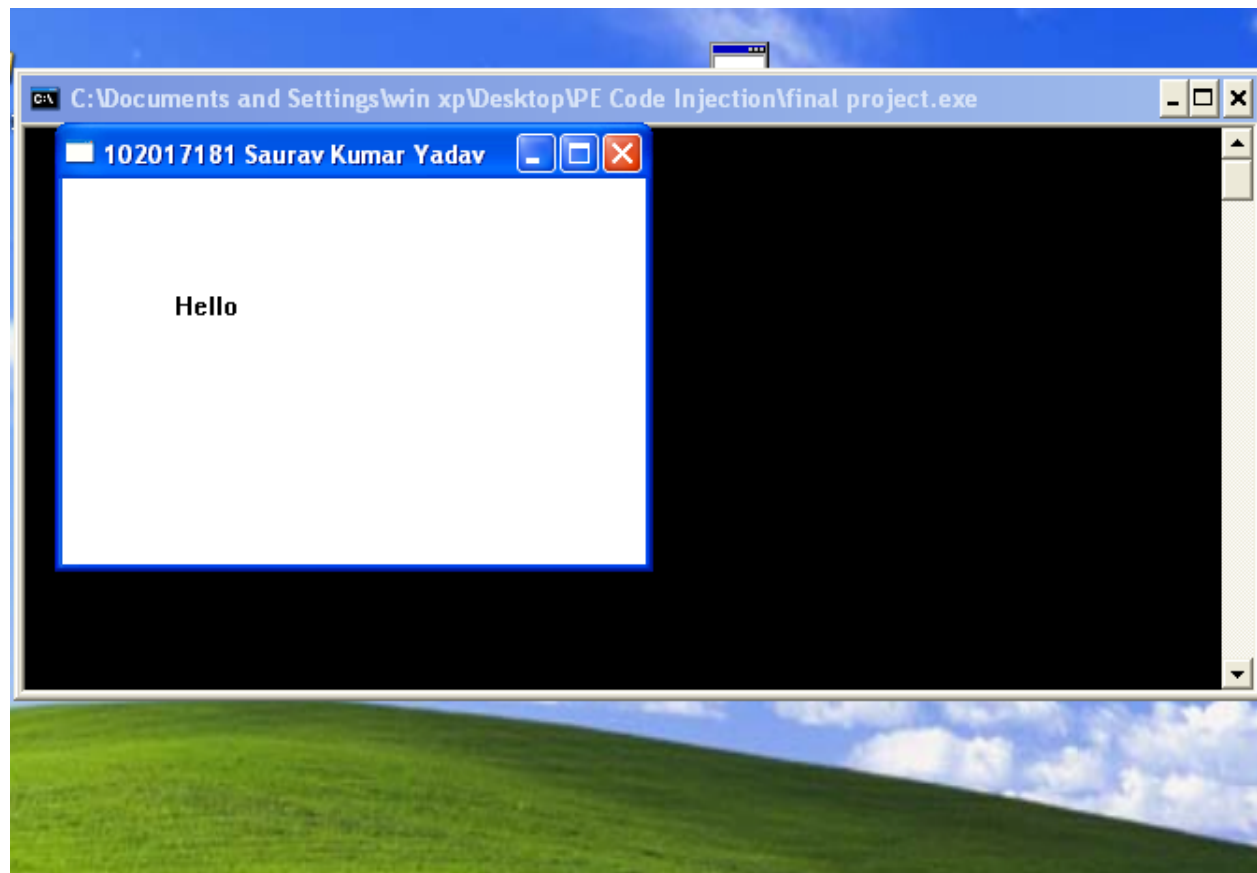
Since the size of application doesnot match the size of the header,Windows displays the following error message while trying to open the program.



XVI32 is used to add 5000 bytes of zeros to the program so that the total size of header match the total size of the program.



The program is then saved. Now the program works as before.



Now using ollydbg PE Code injection is done.

From the memory map, The address of the header which we added is found.

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
00022000	00001000				Priv	RW	Guar	RW
00022000	00003000				Priv	RW	Guar	RW
00023000	00003000			stack of na	Map	R	R	
00024000	00004000				Priv	RW	RW	
00034000	00006000				Priv	RW	RW	
00035000	00003000				Map	RW	RW	
00036000	00016000				Map	R	R	
00038000	00041000				Map	R	R	
0003D000	00006000				Map	R	R	
0003E000	00001000				Priv	RW	RW	
0003F000	00001000				Priv	RW	RW	
00040000	00001000	final_pr		PE header	Imag	R	RWE	
000401000	00002000	final_pr	.text	code	Imag	R	RWE	
000403000	00001000	final_pr	.data	data	Imag	R	RWE	
000404000	00001000	final_pr	.rdata		Imag	R	RWE	
000405000	00001000	final_pr	/4		Imag	R	RWE	
000406000	00001000	final_pr	.bss		Imag	R	RWE	
000407000	00001000	final_pr	.idata	imports	Imag	R	RWE	
000408000	00001000	final_pr	.CRT		Imag	R	RWE	
000409000	00001000	final_pr	.tls		Imag	R	RWE	
00040A000	00001000	final_pr	/14		Imag	R	RWE	
00040B000	00002000	final_pr	/29		Imag	R	RWE	
00040D000	00001000	final_pr	/41		Imag	R	RWE	
00040E000	00001000	final_pr	/55		Imag	R	RWE	
00040F000	00001000	final_pr	/67		Imag	R	RWE	
000410000	00001000	final_pr	/80		Imag	R	RWE	
000411000	00005000	final_pr	.saurav		Imag	R	RWE	
000420000	00041000				Map	R	R	\Device\HarddiskVolume1\WINDOWS\system32\sortkey.nls
000470000	00008000				Map	R	E	R
000530000	00002000				Map	R	E	R
000540000	00103000				Map	R	R	R
000550000	00076000				Map	R	E	R

The address of new header is 0X00411000. This address is where the MessageBoxA() code will be injected to display message box.

The Program has the following instructions in main module:

The screenshot displays a debugger interface with two main panes. The left pane shows assembly instructions with their addresses, hex values, and mnemonics. The right pane shows the state of the CPU registers.

Assembly Instructions (Left Pane):

- 00401280: 83EC 1C SUB ESP,1C
- 00401283: C70424 010000 MOV DWORD PTR SS:[ESP],1
- 00401284: FF15 80714000 CALL DWORD PTR DS:[&msvcrt.__set_app_type] msvcrt.__set_app_type
- 00401286: 58 68FDFFFF CBL final_pr.00401000
- 00401288: A1 807426 00 LEA ESI,DWORD PTR DS:[ESI]
- 00401289: 80BC27 000000 LEA EDI,DWORD PTR DS:[EDI]
- 0040128A: 83EC 1C SUB ESP,1C
- 0040128B: C70424 020000 MOV DWORD PTR SS:[ESP],2
- 0040128C: FF15 80714000 CALL DWORD PTR DS:[&msvcrt.__set_app_type] msvcrt.__set_app_type
- 0040128E: E8 48FDFFFF CALL final_pr.00401000
- 0040128F: 807426 00 LEA ESI,DWORD PTR DS:[ESI]
- 00401290: 80BC27 000000 LEA EDI,DWORD PTR DS:[EDI]
- 00401291: A1 80714000 MOV EAX,DWORD PTR DS:[&msvcrt.atexit]
- 00401292: FFE0 JMP EAX
- 00401293: 89F6 MOV ESI,ESI
- 00401294: 80BC27 000000 LEA EDI,DWORD PTR DS:[EDI]
- 00401295: A1 8C714000 MOV EAX,DWORD PTR DS:[&msvcrt._onexit]
- 00401296: FFE0 JMP EAX
- 00401297: 90 NOP
- 00401298: 90 NOP
- 00401299: 90 NOP
- 0040129A: 90 NOP
- 0040129B: 90 NOP
- 0040129C: 90 NOP
- 0040129D: 90 NOP
- 0040129E: 90 NOP
- 0040129F: 90 NOP
- 004012A0: 55 PUSH EBP
- 004012A1: 89E5 MOV EBP,ESP
- 004012A2: 83EC 18 SUB ESP,18
- 004012A3: A1 80304000 MOV EAX,DWORD PTR DS:[4030181]
- 004012A4: 85C0 TEST EAX,EAX
- 004012A5: 74 3A JE SHORT final_pr.00401329
- 004012A6: C70424 00404000 MOV DWORD PTR SS:[ESP],final_pr.00404000
- 004012A7: E8 89030000 CALL <JMP.&KERNEL32.GetModuleHandleA> ASCII "libgcoj-16.dll" GetModuleHandleA
- 004012A8: 83EC 04 SUB ESP,4
- 004012A9: 85C0 TEST EAX,EAX
- 004012AA: 8A 00000000 MOV EDI,0
- 004012AB: 74 15 JE SHORT final_pr.0040131C
- 004012AC: C74424 04 0E4 MOV DWORD PTR SS:[ESP+4],final_pr.00404000
- 004012AD: 890424 MOV DWORD PTR SS:[ESP],EAX
- 004012AE: E8 05030000 CALL <JMP.&KERNEL32.GetProcAddress> ASCII "_Jv_RegisterClasses" GetProcAddress
- 004012AF: 83EC 08 SUB ESP,8
- 004012B0: 89C2 MOV EDX,EAX
- 004012B1: 5502 TEST EDX,EDX
- 004012B2: 74 09 JE SHORT final_pr.00401329
- 004012B3: C70424 18304000 MOV DWORD PTR SS:[ESP],final_pr.00403018
- 004012B4: FFD2 CALL EDI
- 004012B5: C70424 40134000 MOV DWORD PTR SS:[ESP],final_pr.00401340
- 004012B6: E8 8BFFFFFF CALL final_pr.004012C0
- 004012B7: C9 LEAVE
- 004012B8: C3 RETN
- 004012B9: 89F6 MOV ESI,ESI
- 004012BA: 80BC27 000000 LEA EDI,DWORD PTR DS:[EDI]
- 004012BB: 55 PUSH EBP
- 004012BC: 89E5 MOV EBP,ESP
- 004012BD: 5D POP EBP
- 004012BE: C3 RETN
- 004012BF: 90 NOP

Registers (FPU) (Right Pane):

- EAX: 00000000
- ECX: 0022FF00
- EDX: 7C90E4F4 ntdll.KiFastSystemCallRet
- EBX: 7FFD7000
- ESP: 0022FFC4
- EBP: 0022FF00
- ESI: FFFFFFFF
- EDI: 7C910208 ntdll.7C910208
- EIP: 00401280 final_pr.<ModuleEntryPoint>
- EAX: 00000000
- ECX: 00000000
- EDX: 00000000
- EBX: 00000000
- ESP: 00000000
- EBP: 00000000
- ESI: 00000000
- EDI: 00000000
- EIP: 00000000
- EAX: 00000000
- ECX: 00000000
- EDX: 00000000
- EBX: 00000000
- ESP: 00000000
- EBP: 00000000
- ESI: 00000000
- EDI: 00000000
- EIP: 00000000

Hex dump (Bottom Pane):

Address Hex dump ASCII

00403000: 64 51 72 68 62 6C 75 65 darkblue

00403008: FF FF FF FF

00403010: 00 40 00 00 00 20 40 00 .@.S@.

00403018: 00 00 00 00 00 00 00 00

00403020: 00 00 00 00 00 00 00 00

00403028: 00 00 00 00 00 00 00 00

00403030: 00 00 00 00 00 00 00 00

00403038: 00 00 00 00 00 00 00 00

00403040: 00 00 00 00 00 00 00 00

00403048: 00 00 00 00 00 00 00 00

00403050: 00 00 00 00 00 00 00 00

00403058: 00 00 00 00 00 00 00 00

00403060: 00 00 00 00 00 00 00 00

00403068: 00 00 00 00 00 00 00 00

00403070: 00 00 00 00 00 00 00 00

00403078: 00 00 00 00 00 00 00 00

00403080: 00 00 00 00 00 00 00 00

00403088: 00 00 00 00 00 00 00 00

00403090: 00 00 00 00 00 00 00 00

00403098: 00 00 00 00 00 00 00 00

004030A0: 00 00 00 00 00 00 00 00

004030A8: 00 00 00 00 00 00 00 00

004030B0: 00 00 00 00 00 00 00 00

004030B8: 00 00 00 00 00 00 00 00

004030C0: 00 00 00 00 00 00 00 00

004030C8: 00 00 00 00 00 00 00 00

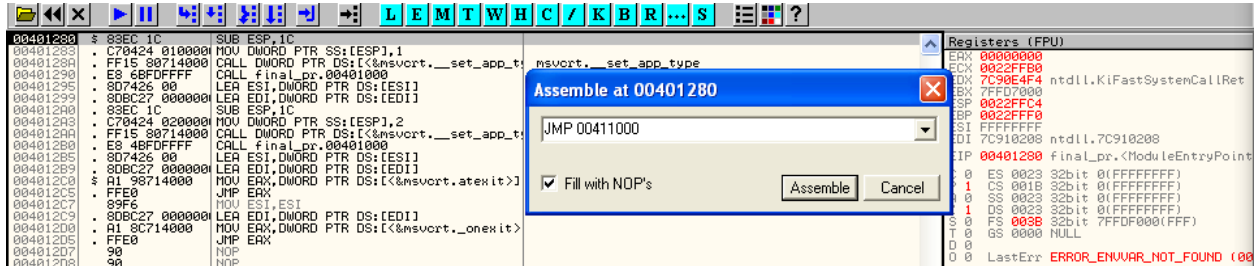
004030D0: 00 00 00 00 00 00 00 00

004030D8: 00 00 00 00 00 00 00 00

004030E0: 00 00 00 00 00 00 00 00

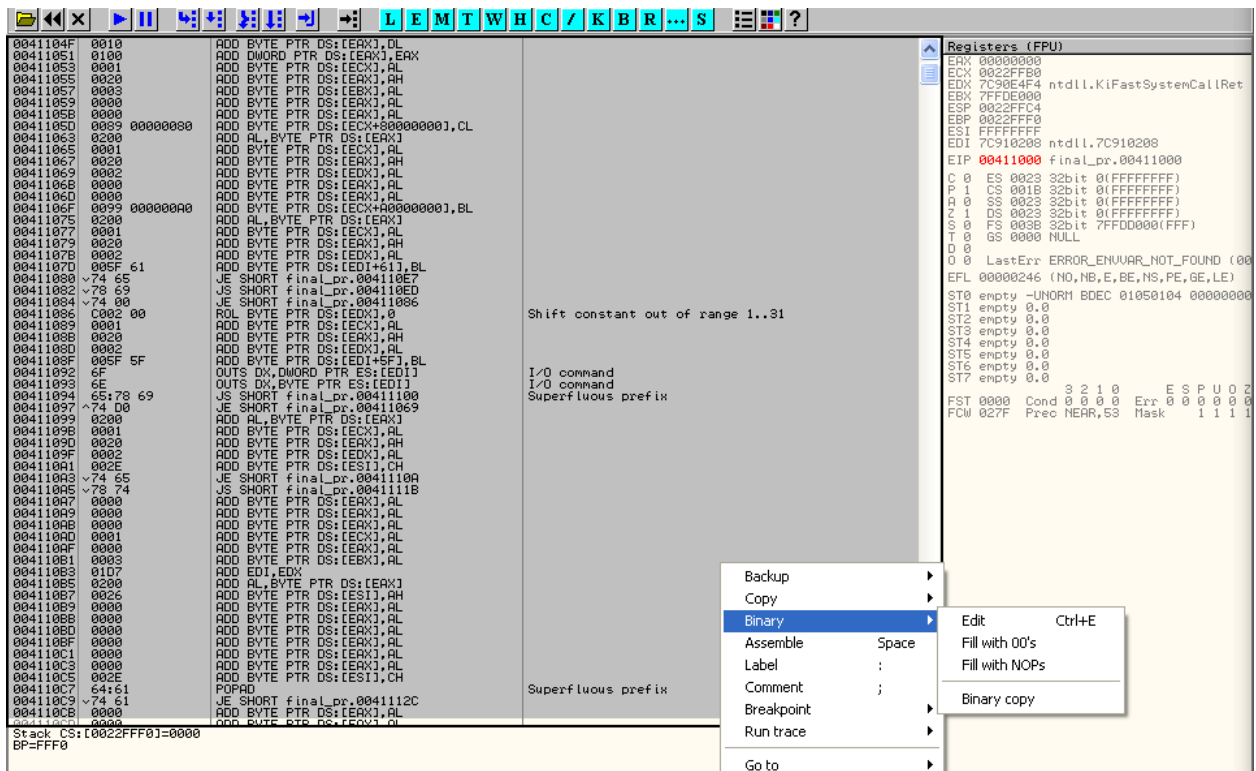
004030E8: 00 00 00 00 00 00 00 00

From the main thread, the first few instructions are copied for later reference. At the entry point of the program the instruction is changed from "SUB ESP,1C" to "JUMP 00411000" due to which there is change in the flow of the program, and the program jump to 0X00411000 and instructions from address starts getting executed.



The changes are then copied to the executables and saved to the program.

From the memory address 0X00411000 some portion is selected and filled with 00 using Ollydbg. This is where we will be injecting new code.



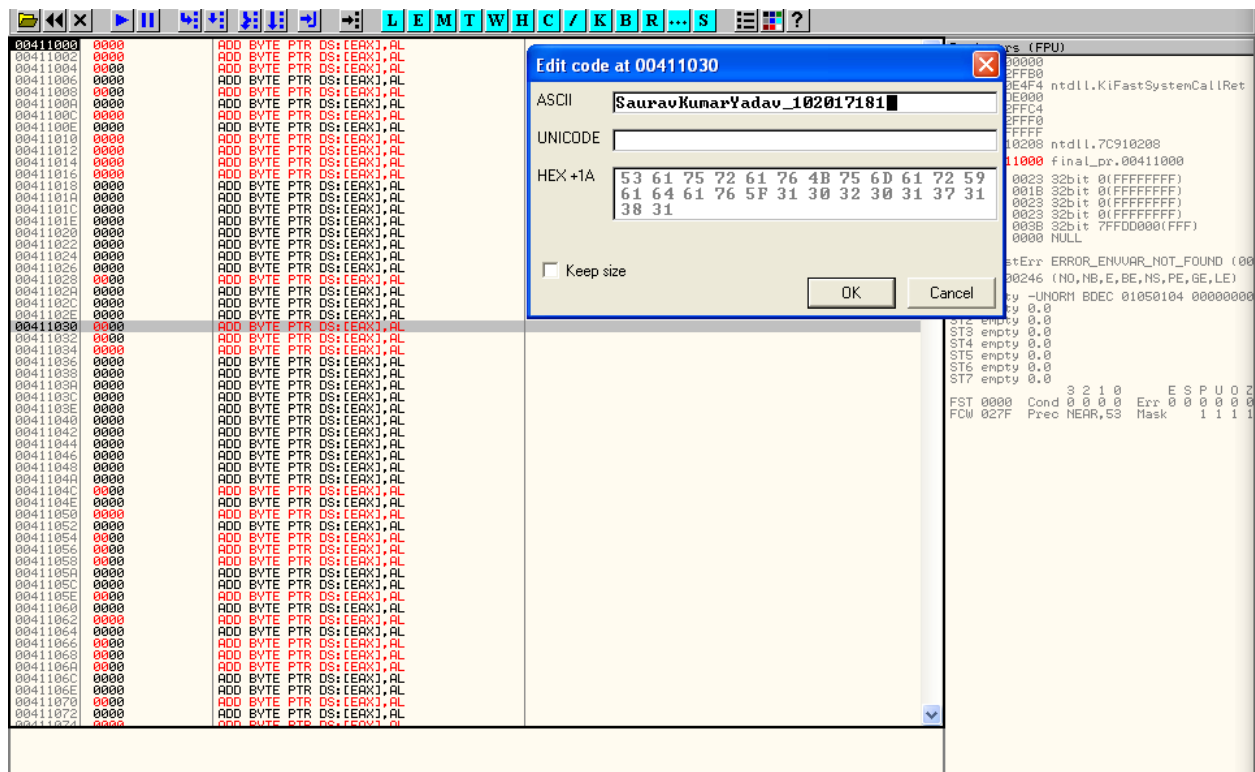
We will be injecting MessageBoxA() into the app. The syntax of message box is as follow
int MessageBoxA(

```
[in, optional] HWND hWnd,  
[in, optional] LPCSTR lpText,  
[in, optional] LPCSTR lpCaption,  
[in]      UINT uType  
);
```

Where in our program hWnd will be '0' signifying no owner for handle to owner window value, lpText will be "You have been Hacked", lpCaption will be "SauravKumarYadav_102017181", and uType value will be '0' for one push button OK.

Two strings are added into the memory , First string is SauravKumarYadav_102017181. Which will be pointed by lpCaption and second string is "You have been Hacked" which will be pointed by lpText.

At 0x00411030 , "SauravKumarYadav_102017181" is added.



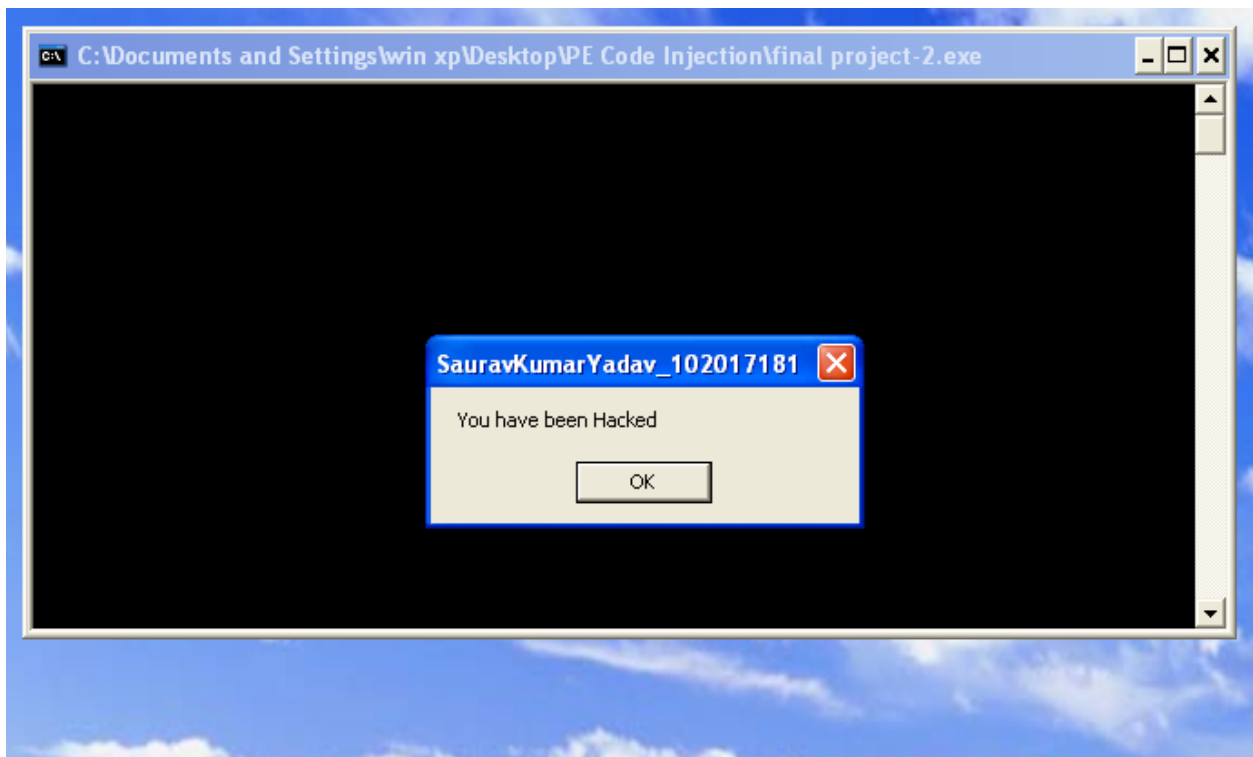
At 0x00411065, "You have been Hacked" is added.

The screenshot shows a debugger window with assembly code on the left and a code edit window on the right. The assembly code is for a function named `final_pr.00411000`. The code starts with a `JBE SHORT final_pr.00411002` instruction at address 00411035. The code then enters a loop that adds bytes from a memory location to the `ESI` register. The code edit window is titled "Edit code at 00411065" and shows the ASCII string "You have been Hacked" being added to the code at address 00411065. The code edit window also shows the hexadecimal representation of the string: 59 6F 75 20 68 61 76 65 20 62 65 65.

At the 00411000 the parameters for `MessageBoxA()` are added and a call is made to `MessageBoxA`.

The screenshot shows a debugger window with assembly code on the left and a registers window on the right. The assembly code is for a function named `final_pr.00411000`. The code starts with a `PUSH 0` instruction at address 00411000. The code then enters a loop that adds bytes from a memory location to the `ESI` register. The code then calls `MessageBoxA` with the parameters `0`, `0`, `0`, and `0`. The registers window shows the state of the registers, including `EAX`, `ECX`, `EDX`, `EBX`, `ESP`, `EBP`, `ESI`, `EDI`, `EIP`, `EAX`, `ECX`, `EDX`, `EBX`, `ESP`, `EBP`, `ESI`, `EDI`, `EIP`, `EAX`, `ECX`, `EDX`, `EBX`, `ESP`, `EBP`, `ESI`, `EDI`, `EIP`.

The changes are then copied to executable and then saved to the file.
On running the newly created app after injecting the code. The following window is displayed.

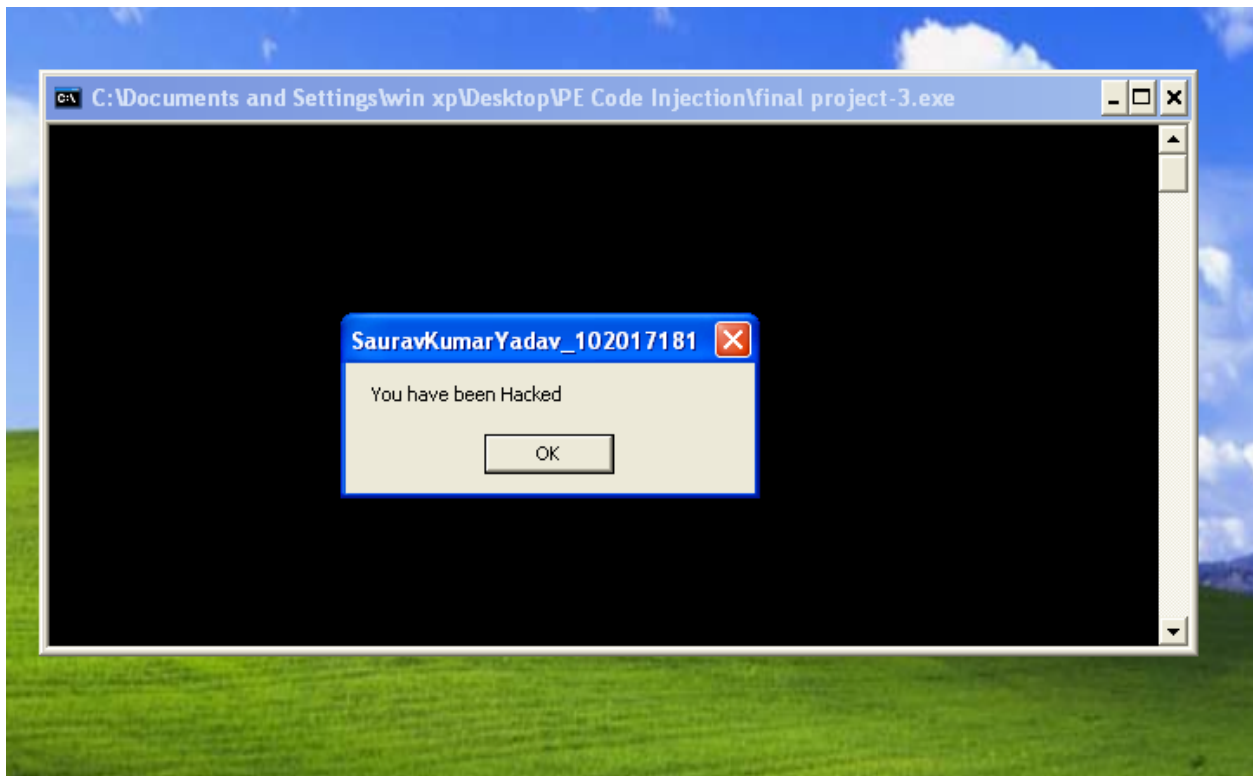


The insertion of MessageBox is successful .Since we have not returned the program to its normal flow yet. It breaks down when ok is pressed. So now the program is returned to it's old flow using ollygdbg.

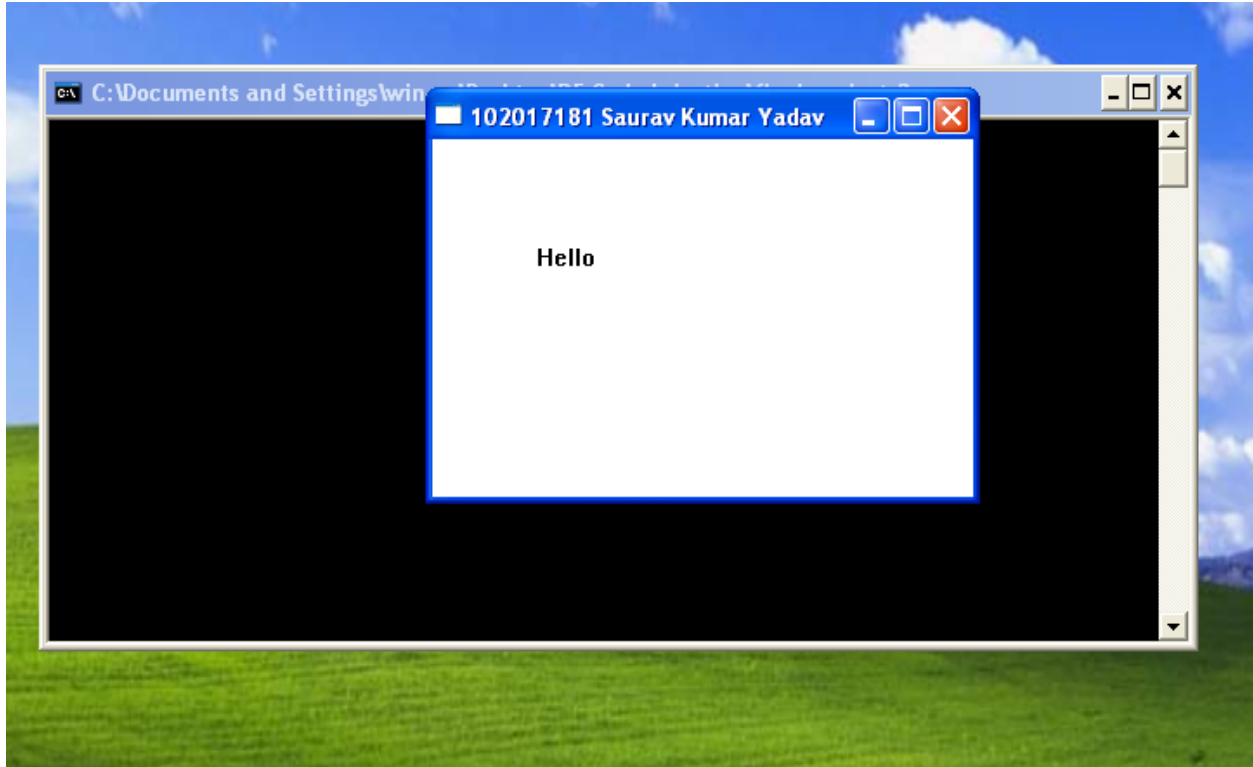
After the MessageBoxA() is called, we add the instructions that we had overwritten with “JMP 00411000”. Then we make the jump to the origin entrypoint using “JMP 00401290” which is the address from where original instructions follow. The selection is then copied to the executable and saved.

Address	Disassembly	Comment
00411000	6A 00	PUSH 0
00411002	68 30104100	PUSH final_pr.00411030
00411007	68 65104100	PUSH final_pr.00411065
0041100C	6A 00	PUSH 0
0041100E	E8 D7F7037E	CALL USER32.MessageBoxA
00411013	90	NOP
00411014	90	NOP
00411015	83EC 1C	SUB ESP,1C
00411018	C70424 01000000	MOV DWORD PTR SS:[ESP],1
0041101F	FF15 80714000	CALL DWORD PTR DS:[<&msvcrt.__set_app_type
00411025	90	NOP
00411026	90	NOP
00411027	E9 6402FFFF	JMP final_pr.00401290
0041102C	90	NOP
0041102D	90	NOP
0041102E	0000	ADD BYTE PTR DS:[EAX],AL
00411030	53	PUSH EBX
00411031	61	POPAD
00411032	75 72	JNZ SHORT final_pr.004110A6
00411034	61	POPAD
00411035	76 4B	JBE SHORT final_pr.00411082
00411037	75 6D	JNZ SHORT final_pr.004110A6
00411039	61	POPAD
0041103A	72 59	JB SHORT final_pr.00411095
0041103C	61	POPAD
0041103D	64:61	POPAD
0041103F	76 5F	JBE SHORT final_pr.004110A0
00411041	3130	XOR DWORD PTR DS:[EAX],ESI
00411043	3230	XOR DH,BYTE PTR DS:[EAX]
00411045	3137	XOR DWORD PTR DS:[EDI],ESI
00411047	3138	XOR DWORD PTR DS:[EAX],EDI
00411049	3100	XOR DWORD PTR DS:[EAX],EAX
0041104B	0000	ADD BYTE PTR DS:[EAX],AL
0041104D	0000	ADD BYTE PTR DS:[EAX],AL
0041104F	0000	ADD BYTE PTR DS:[EAX],AL
00411051	0000	ADD BYTE PTR DS:[EAX],AL
00411053	0000	ADD BYTE PTR DS:[EAX],AL
00411055	0000	ADD BYTE PTR DS:[EAX],AL
00411057	0000	ADD BYTE PTR DS:[EAX],AL
00411059	0000	ADD BYTE PTR DS:[EAX],AL
0041105B	0000	ADD BYTE PTR DS:[EAX],AL
0041105D	0000	ADD BYTE PTR DS:[EAX],AL
0041105F	0000	ADD BYTE PTR DS:[EAX],AL
00411061	0000	ADD BYTE PTR DS:[EAX],AL
00411063	0000	ADD BYTE PTR DS:[EAX],AL
00411065	59	POP ECX
00411066	6F	OUTS DX,DWORD PTR ES:[EDI]
00411067	75 20	JNZ SHORT final_pr.00411089
00411069	68 61766520	PUSH 20657661
0041106E	6265 65	BOUND ESP,QWORD PTR SS:[EBP+65]
00411071	6E	OUTS DX,BYTE PTR ES:[EDI]
00411072	2048 61	AND BYTE PTR DS:[EAX+61],CL
00411075	636B 65	ARPL WORD PTR DS:[EBX+65],BP
00411078	64:0000	ADD BYTE PTR FS:[EAX],AL
0041107B	0000	ADD BYTE PTR DS:[EAX],AL
0041107D	0000	ADD BYTE PTR DS:[EAX],AL
0041107F	0000	ADD BYTE PTR DS:[EAX],AL
00411081	0000	ADD BYTE PTR DS:[EAX],AL
00411083	0000	ADD BYTE PTR DS:[EAX],AL

The new program behaves as follow:



After ok is pressed following appear



Which is the original program.

Hence a program was created using Win32 to display "Hello". This program was hijacked and MessageBox was successfully injected in the program to display "Saurav Kumar Yadav_102017181" as caption and "You have been Hacked" as text. Then the original entry point of the program was retrieved and the flow of program was returned to original.

Code used to create the app:

```
#include <windows.h>
```

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

```
static char gszClassName[] = "102017181 Saurav Kumar Yadav";
```

```
static HINSTANCE ghInstance = NULL;
```

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,  
int nCmdShow) {
```

```
    WNDCLASSEX WndClass;
```

```
    HWND hwnd;
```

```
    MSG Msg;
```

```
    ghInstance = hInstance;
```

```
    WndClass.cbSize = sizeof(WNDCLASSEX);
```

```
    WndClass.style = NULL;
```

```
    WndClass.lpfnWndProc = WndProc;
```

```
    WndClass.cbClsExtra = 0;
```

```
    WndClass.cbWndExtra = 0;
```

```
    WndClass.hInstance = ghInstance;
```

```
    WndClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
```

```
    WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
```

```
    WndClass.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
```

```
    WndClass.lpszMenuName = NULL;
```

```
    WndClass.lpszClassName = gszClassName;
```

```
    WndClass.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
```

```
    if(!RegisterClassEx(&WndClass)) {
```

```
        MessageBox(0, "Error Registering Window!", "Error!", MB_ICONSTOP | MB_OK);
```

```
        return 0;
```

```
    }
```

```
    hwnd = CreateWindowEx(
```

```
        WS_EX_STATICEDGE,
```

```
        gszClassName,
```

```
        "102017181 Saurav Kumar Yadav",
```

```
        WS_OVERLAPPEDWINDOW,
```

```
        CW_USEDEFAULT, CW_USEDEFAULT,
```

```
        320, 240,
```

```
        NULL, NULL,
```

```
        ghInstance,
```

```
        NULL);
```

```

    if(hwnd == NULL) {
        MessageBox(0, "Window Creation Failed!", "Error!", MB_ICONSTOP | MB_OK);
        return 0;
    }

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    while(GetMessage(&Msg, NULL, 0, 0)) {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    return Msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT Message, WPARAM wParam, LPARAM
IParam) {
    HDC hdc;
    PAINTSTRUCT ps; SECURE CODING-UCS638
    LPSTR szMessage = "Hello!";

    switch(Message) {
        case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps);
            TextOut(hdc, 70, 50, szMessage, strlen(szMessage));
            EndPaint(hwnd, &ps);
            break;
        case WM_CLOSE:
            DestroyWindow(hwnd);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, Message, wParam, IParam);
    }
    return 0;
}

```