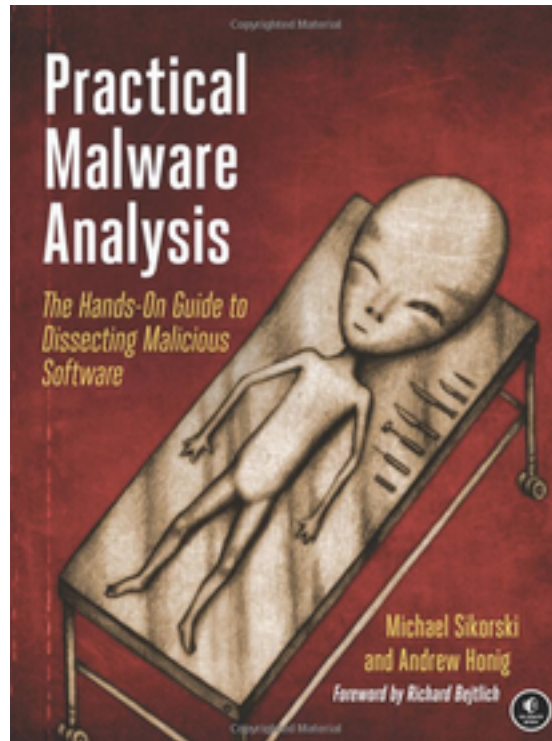


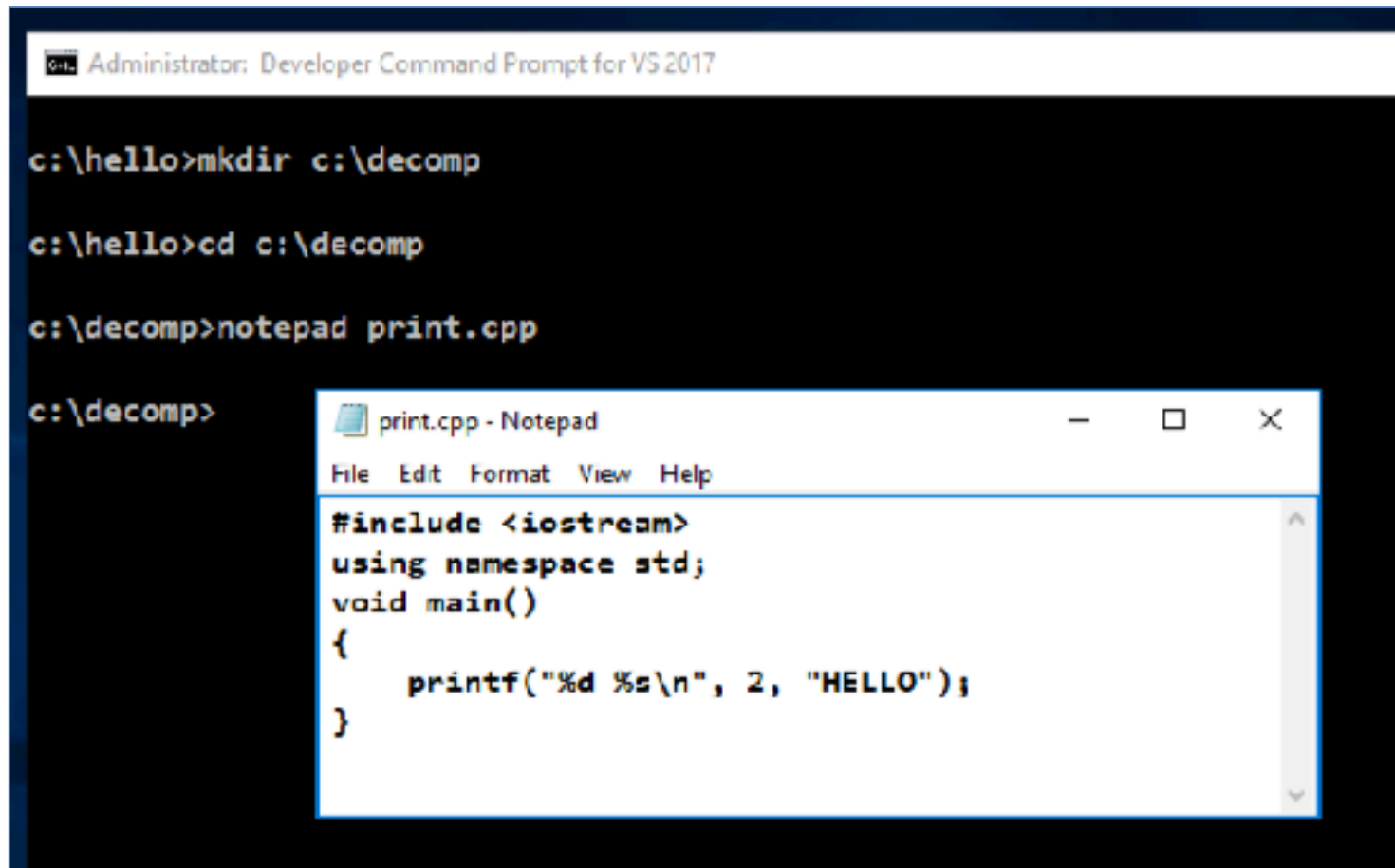
# Practical Malware Analysis

## Ch 6: Recognizing C Constructs in Assembly



Updated 10-2-18

# Function Call



The image shows a Windows Developer Command Prompt window titled "Administrator: Developer Command Prompt for VS 2017". The command history is as follows:

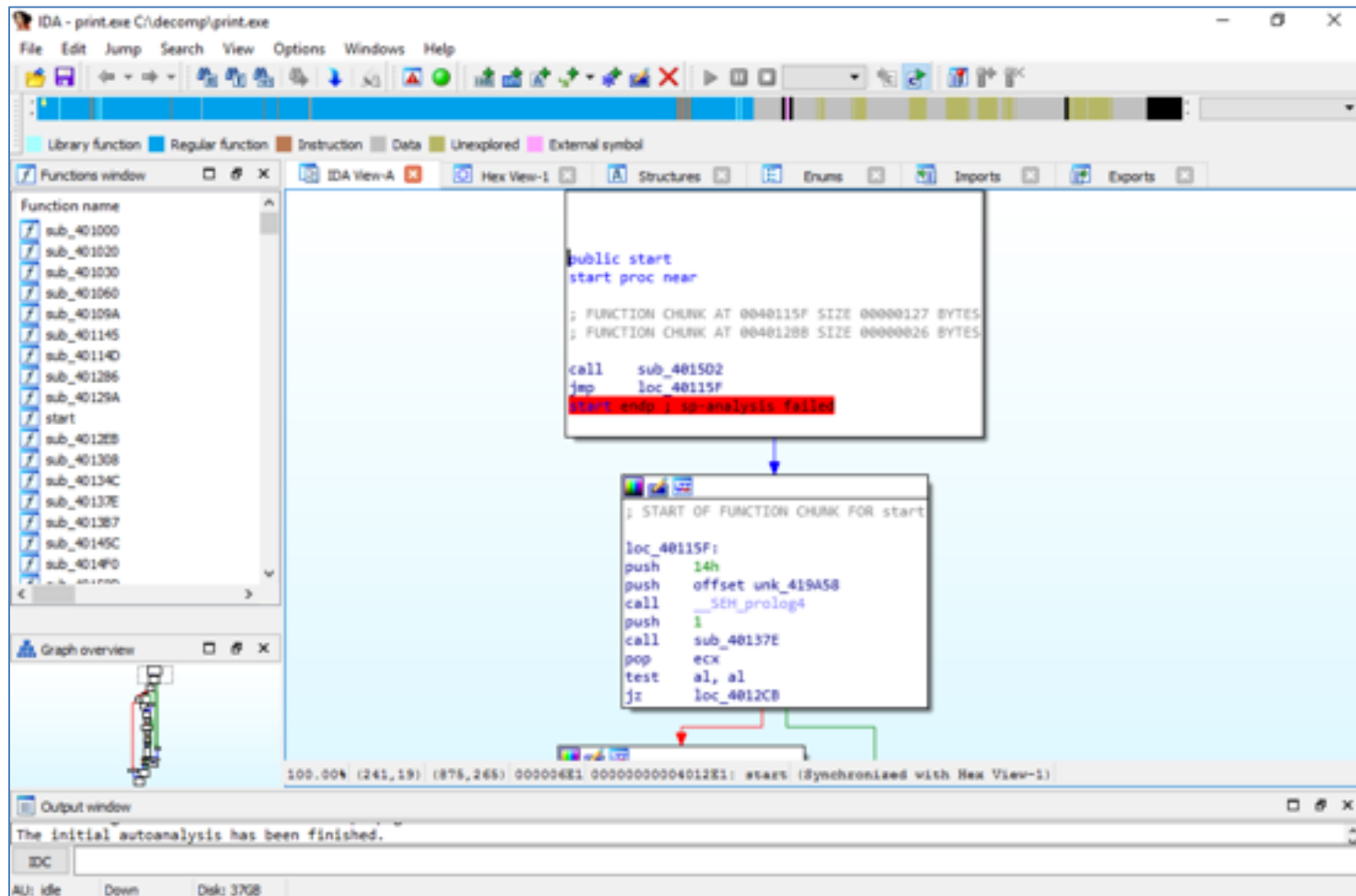
```
c:\hello>mkdir c:\decomp  
c:\hello>cd c:\decomp  
c:\decomp>notepad print.cpp  
c:\decomp>
```

An instance of Notepad is open, titled "print.cpp - Notepad". The menu bar includes "File", "Edit", "Format", "View", and "Help". The code in the editor is:

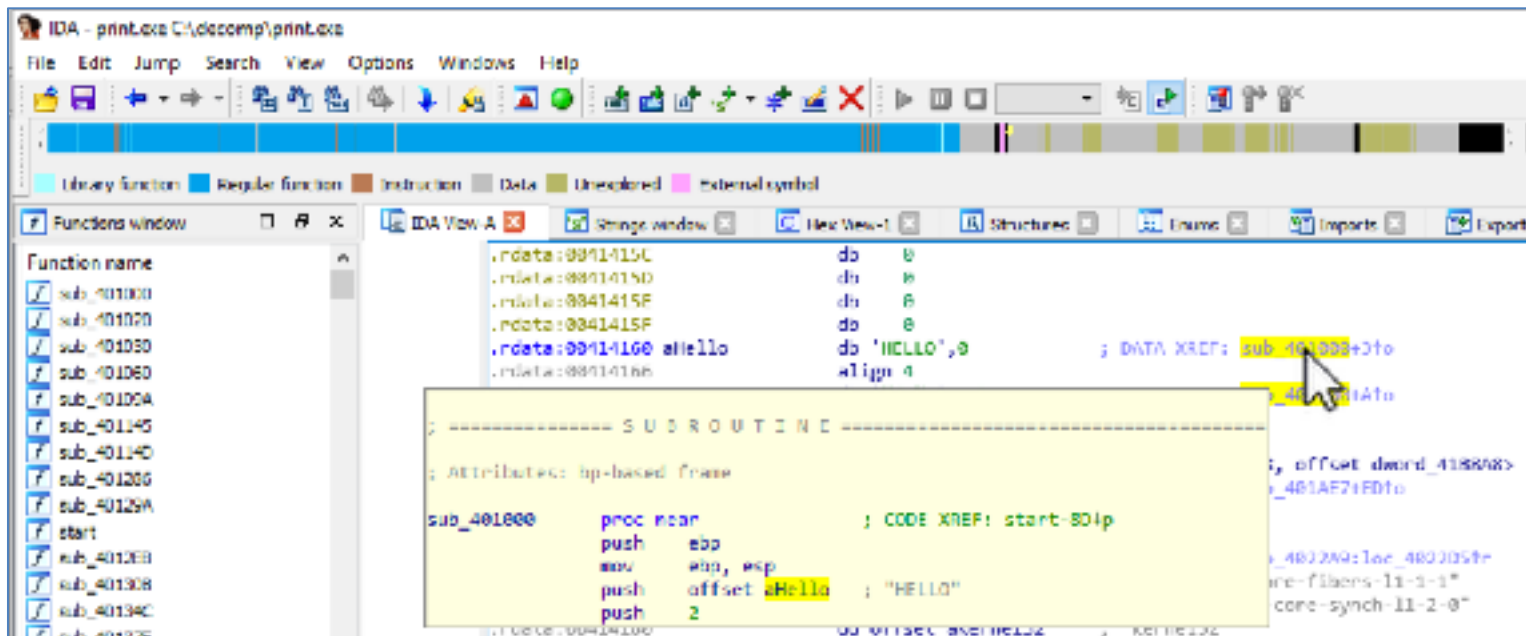
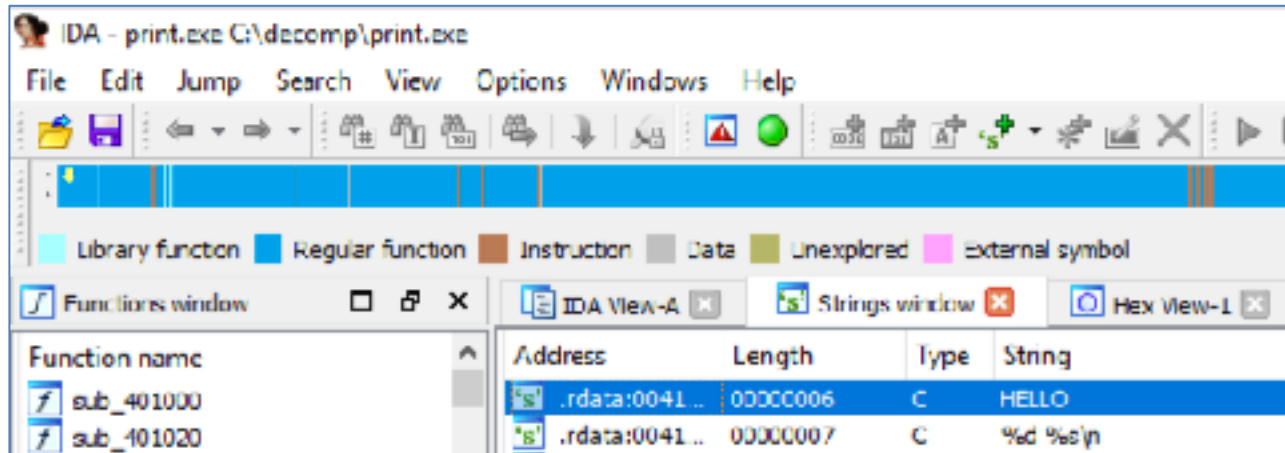
```
#include <iostream>  
using namespace std;  
void main()  
{  
    printf("%d %s\n", 2, "HELLO");  
}
```

# Finding the Code in IDA Pro

- IDA shows only the entry point



# Use Strings, then XREF



# Disassembly in IDA Pro

- Arguments for printf() function
- Pushed onto stack
- Reverse order
- **call** launches function

```
printf("%d %s\n", 2, "HELLO");
```

```
sub_401000 proc near
push     ebp
mov      ebp, esp
push     offset aHello    ; "HELLO"
push     2
push     offset aDS       ; "%d %s\n"
call     sub_401060
add      esp, 0Ch
xor      eax, eax
pop      ebp
retn
sub_401000 endp
```

# Global vs. Local Variables

- Global variables
  - Available to any function in the program
  - Stored outside all functions
- Local variables
  - Defined in a function and only available to that function
  - Stored on the stack

# Global vs. Local Variables

```
#include <iostream>
using namespace std;

int g=2; // GLOBAL VARIABLE

void main()
{
    int l = 3; // LOCAL VARIABLE
    printf("%d %d\n", g, l);
}
```

```
sub_401000 proc near
var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+var_4], 3
mov     eax, [ebp+var_4]
push    eax
mov     ecx, dword_41B000
push    ecx
push    offset aDD          ; "%d %d\n"
call    sub_401070
add     esp, 0Ch
xor     eax, eax
mov     esp, ebp
pop     ebp
retn
sub_401000 endp
```

# Arithmetic Operations

```
#include <iostream>
using namespace std;

void main()
{
    int i = 3;
    int j = 6;
    int k = i + j;
    int l = j - i;
    int m = j * i;
    float n = (float)j / (float)i;
    printf("%d %d %d %d %d %f\n", i, j, k, l, m, n);
}
```

```
push    ebp
mov     ebp, esp
sub     esp, 18h
mov     [ebp+var_8], 3
mov     [ebp+var_4], 6
mov     eax, [ebp+var_8]
add     eax, [ebp+var_4]
mov     [ebp+var_18], eax
mov     ecx, [ebp+var_4]
sub     ecx, [ebp+var_8]
mov     [ebp+var_14], ecx
mov     edx, [ebp+var_4]
imul    edx, [ebp+var_8]
mov     [ebp+var_10], edx
cvttsi2ss xmm0, [ebp+var_4]
cvttsi2ss xmm1, [ebp+var_8]
divss   xmm0, xmm1
movss   [ebp+var_C], xmm0
cvtss2sd xmm0, [ebp+var_C]
sub     esp, 8
movsd   [esp+20h+var_20], xmm0
```



# Branching (if)

```
#include <iostream>
using namespace std;

void main()
{
    int i = 3;
    if (i > 0) {
        printf("i is positive\n");
    } else {
        printf("i is not positive\n");
    }
}
```

```
sub_401000 proc near
var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+var_4], 3
cmp     [ebp+var_4], 0
jle     short loc_401020
```

```
push    offset aIIsPositive ; "i is positive\n"
call    sub_401080
add     esp, 4
jmp     short loc_40102D
```

```
loc_401020:
push    offset aIIsNotPositive ; "i is not positive\n"
call    sub_401000
add     esp, 4
```

# Finding for Loops

*Example 7-12. C code for a for loop*

```
int i;  
  
for(i=0; i<100; i++)  
{  
    printf("i equals %d\n", i);  
}
```

- Four components
  - **Initialization:** i starts at 0
  - **Comparison:** is i<100 ?
  - **Execution:** printf
  - **Increment/decrement:** i++

*Example 7-13. Assembly code for the for loop example in Example 7-12*

00401004	mov	[ebp+var_4], 0	1	Initialization
0040100B	jmp	short loc_401016	2	
0040100D	loc_40100D:			
0040100D	mov	eax, [ebp+var_4]	3	Increment
00401010	add	eax, 1		
00401013	mov	[ebp+var_4], eax	4	
00401016	loc_401016:			
00401016	cmp	[ebp+var_4], 64h	5	Comparison
0040101A	jge	short loc_40102F	6	
0040101C	mov	ecx, [ebp+var_4]		Execution
0040101F	push	ecx		
00401020	push	offset aID ; "i equals %d\n"		
00401025	call	printf		
0040102A	add	esp, 8		
0040102D	jmp	short loc_40100D	7	

# Arrays

*Example 7-24. C code for an array*

```
int b[5] = {123,87,487,7,978};  
void main()  
{  
    int i;  
    int a[5];  
  
    for(i = 0; i<5; i++)  
    {  
        a[i] = i;  
        b[i] = i;  
    }  
}
```

*Example 7-25. Assembly code for the array in [Example 7-24](#)*

```
00401006      mov     [ebp+var_18], 0
0040100D      jmp     short loc_401018
0040100F loc_40100F:
0040100F      mov     eax, [ebp+var_18]
00401012      add     eax, 1
00401015      mov     [ebp+var_18], eax
00401018 loc_401018:
00401018      cmp     [ebp+var_18], 5
0040101C      jge     short loc_401037
0040101E      mov     ecx, [ebp+var_18]
00401021      mov     edx, [ebp+var_18]
00401024      mov     [ebp+ecx*4+var_14], edx 1
00401028      mov     eax, [ebp+var_18]
0040102B      mov     ecx, [ebp+var_18]
0040102E      mov     dword_40A000[ecx*4], eax 2
00401035      jmp     short loc_40100F
```

**Initialization**

**Increment**

**Comparison**

**Assign value to  
Element in b  
(base is var\_14)**

**Assign value to  
Element in a  
(base is dword\_40A000)**

# Summary

- Finding the Code
  - Strings, then XREF
- Function Call
  - Arguments pushed onto stack
  - Reverse order
  - call
- Variables
  - Global: in memory, available to all functions
  - Local: on stack, only available to one function

# Summary

- Arithmetic
  - Move variables into registers
  - Perform arithmetic (**add**, **sub**, **idiv**, etc.)
  - Move results back into variables
- Branching
  - Compare (**cmp**, **test**, etc.)
  - Conditional jump (**jz**, **jnz**, etc.)
  - Red arrow if false, green arrow if true

**Kahoot!**