Practical Foundations of Debugging

Chapter 8

Function Pointer Parameters - Part 2

Designed, written and illustrated by Dmitry Vostokov

Patterns (functions)

Function prolog

```
push    ebp
mov     ebp,esp
```

Function epilog

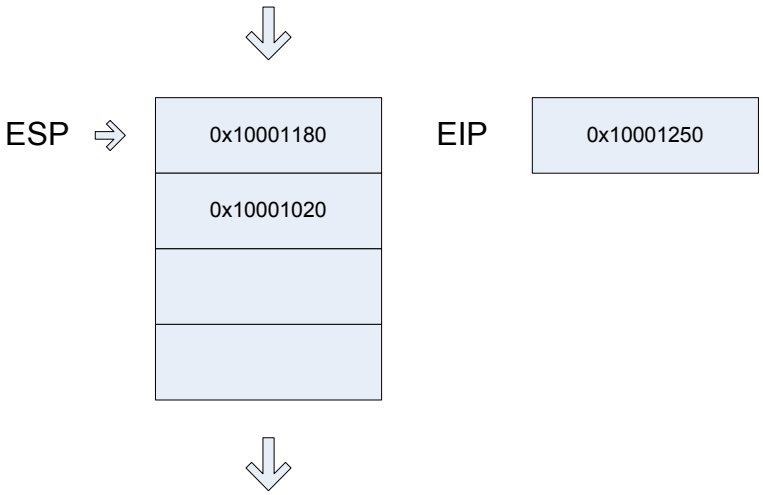
```
mov     esp,ebp
pop     ebp
ret     [number]
```

or

```
leave
ret     [number]
```

Call stack

```
.PDB file
func    0x10001000 – 0x10001100
func2   0x10001101 – 0x10001200
func3   0x10001201 – 0x10001300
```



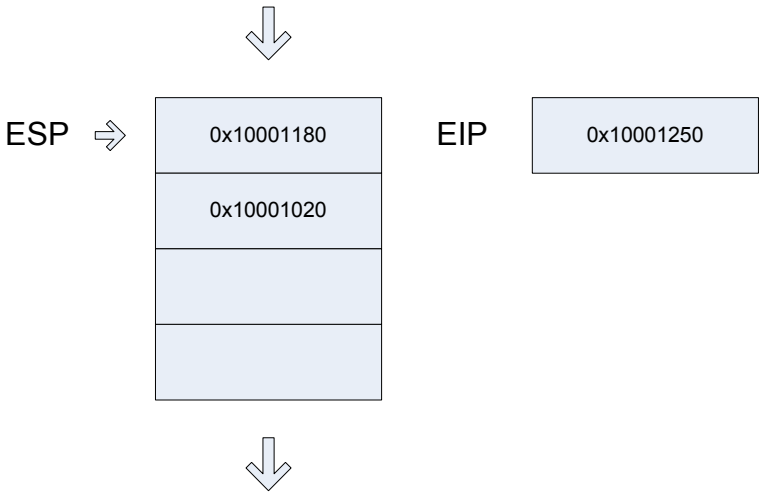

```
func3+0x4F
func2+0x7F
func+0x20
```

```
➢u func2 func2+0x7F

push ebp
mov  ebp, esp
…
…
call func3
```

Wrong symbol file

```
.PDB file
func    0x10001000 – 0x10001090
func2   0x10001091 – 0x10001190
func3   0x10001191 – 0x10001290
```




```
func3+0x5F
func2+0x8F
func+0x20
```

```
u func2 func2+0x8F

add  ecx, 10
mov  eax, [ebx+10]
push ebp
mov  ebp, esp
…
…
call func3+0x10
```

Patterns (passing parameters)

```
[ebp-XXX]       local variable
[ebp+XXX]       function parameter
```

Mnemonic: first push parameters (+, up) then use local variables (-, down)

```
push  0x427034
Address of static/global variable. But more likely string constant
   (db 0x427034)

mov     reg,[ebp-XXX]
push    reg                     ; local variable
…
call    func


lea     reg,[ebp-XXX]
push    reg                     ; address of local variable
…
call    func
```

LEA (load effective address)

```
lea  eax,[ebp-0x8]

is equivalent to:

mov  eax,ebp
sub  eax,0x8
```

Patterns (accessing parameters and local variables)

```
mov     eax,[ebp+0x8]   accessing DWORD parameter
add     eax,0x1

mov     eax,[ebp+0x8]   accessing a pointer
mov     eax,[eax]
add     eax,0x1

mov     eax,[ebp-0x8]   accessing DWORD local
add     eax,0x1

mov     eax,[ebp-0x8]   accessing a pointer to local
mov     eax,[eax]
add     eax,0x1
```

“Arithmetical” project

```
// FunctionParameters.cpp
#include <stdio.h>
#include "Arithmetic.h"

int main(int argc, char* argv[])
{
    int a, b;

    printf("Enter a and b: ");
    scanf("%d %d", &a, &b);

    if (arithmetic (a, &b))
    {
         printf("Result = %d", b);
    }

    return 0;
}
```

```
// Arithmetic.h
#ifndef __ARITHMETIC_H__
#define __ARITHMETIC_H__

bool arithmetic (int a, int *b);

#endif
```

```
// Arithmetic.cpp
#include "Arithmetic.h"

bool arithmetic (int a, int *b)
{
        if (!b)
        {
                return false;
        }

        *b = *b + a;
        ++a;
        *b = a * *b;

        return true;
}
```

```
FunctionParameters!main:
push    ebp                             ; establishing stack frame
mov     ebp,esp                         ;
sub     esp,0xd8                        ; creating stack frame for locals
push    ebx                             ; saving registers that might be used
push    esi                             ;   outside
push    edi                             ;
lea     edi,[ebp-0xd8]                  ; getting lowest address of stack frame
mov     ecx,0x36                        ; filling stack frame with 0xCC
mov     eax,0xcccccccc                  ;
rep     stosd                           ;
push    0x427034                        ; address of "Enter a and b: " string
call    FunctionParameters!ILT+1285(_printf) (0041150a)
add     esp,0x4                         ; adjust stack pointer (1 parameter)
lea     eax,[ebp-0x14]                  ; address of b
push    eax                             ;
lea     ecx,[ebp-0x8]                   ; address of a
push    ecx                             ;
push    0x42702c                        ; address of "%d %d" string
call    FunctionParameters!ILT+990(_scanf) (004113e3)
add     esp,0xc                         ; adjust stack pointer (3 parameters,
…                                       ;   3*4 = 12 bytes, 0xc in hexadecimal)
```

```
lea     eax,[ebp-0x14]                  ; address of b
push    eax                             ;
mov     ecx,[ebp-0x8]                   ; value of a
push    ecx                             ;
call    FunctionParameters!ILT+535(?arithmeticYA_NHPAHZ) (0041121c)
add     esp,0x8                         ; adjust stack pointer (2 parameters)
movzx   edx,al                          ; bool result from arithmetic
test    edx,edx                         ; testing for zero
jz      FunctionParameters!main+0x68 (00411bf8)
mov     eax,[ebp-0x14]                  ; value of b
push    eax                             ;
push    0x42701c                        ; address of "Result = %d" string
call    FunctionParameters!ILT+1285(_printf) (0041150a)
add     esp,0x8                         ; adjust stack pointer (2 variables)
00411bf8:
xor     eax,eax                         ; return result 0
push    edx                             ; saving register ?
mov     ecx,ebp                         ; pasing parameter via ecx
push    eax                             ; saving register ?
lea     edx,[FunctionParameters!main+0x8f (00411c1f)] ; probably address of info about
                                        ;  stack frame
call    FunctionParameters!ILT+455(_RTC_CheckStackVars (004111cc)
pop     eax                             ; restoring register
pop     edx                             ;
pop     edi                             ;
pop     esi                             ;
pop     ebx                             ;
add     esp,0xd8                        ; adjusting stack pointer
cmp     ebp,esp                         ; ESP == EBP ?
call    FunctionParameters!ILT+1035(__RTC_CheckEsp) (00411410)
mov     esp,ebp                         ; restoring previous stack pointer
pop     ebp                             ; restoring previous stack frame
ret                                     ; return
```

```
FunctionParameters!arithmetic:
push    ebp
mov     ebp,esp
sub     esp,0xc0
push    ebx
push    esi
push    edi
lea     edi,[ebp-0xc0]
mov     ecx,0x30
mov     eax,0xcccccccc
rep     stosd
cmp     dword ptr [ebp+0xc],0x0          ; &b == 0 ?
jnz     FunctionParameters!arithmetic+0x28 (00411b48)
xor     al,al                            ; return bool value false (0)
jmp     FunctionParameters!arithmetic+0x4e (00411b6e)
00411b48:
mov     eax,[ebp+0xc]                    ; eax := address of b
mov     ecx,[eax]
add     ecx,[ebp+0x8]                    ; ecx := ecx + [a]  (in C: t = *b + a)
mov     edx,[ebp+0xc]                    ; edx := address of b
mov     [edx],ecx                        ;                   (in C: *b := t)
mov     eax,[ebp+0x8]                    ; eax := [a]        (in C: ++a)
add     eax,0x1
mov     [ebp+0x8],eax                    ; [a] := eax
mov     eax,[ebp+0xc]                    ; eax := address of b
mov     ecx,[ebp+0x8]                    ; ecx := [a]
imul    ecx,[eax]                        ; ecx := ecx * [b]  (in C: t = a * *b)
mov     edx,[ebp+0xc]                    ; edx := address of b
mov     [edx],ecx                        ;                   (in C: *b = t)
mov     al,0x1                           ; return bool value true (0)
00411b6e:
pop     edi
pop     esi
pop     ebx
mov     esp,ebp
pop     ebp
ret
```

With FPO (dynamic addressing of local variables)

```
FunctionParameters!main:
sub     esp,0x8           ; allocating room for local variables
push    0x408110          ; address of "Enter a and b: "
call    FunctionParameters!printf (00401085)
lea     eax,[esp+0x4]     ; address of b   ([ESP + 0 + 4])
push    eax
lea     ecx,[esp+0xc]     ; address of a   ([ESP + 4 + 8])
push    ecx
push    0x408108          ; address of "%d %d"
call    FunctionParameters!scanf (0040106e)
mov     eax,[esp+0x14]    ; value of a     ([ESP + 4 + 10])
lea     edx,[esp+0x10]    ; address of b   ([ESP + 0 + 10])
push    edx
push    eax
call    FunctionParameters!arithmetic (00401000)
add     esp,0x18          ; adjusting stack after all pushes
test    al,al             ; al == 0 ?
jz      FunctionParameters!main+0x48 (00401068)
mov     ecx,[esp]         ; address of b   ([ESP + 0])
push    ecx
push    0x4080fc          ; address of "Result = %d"
call    FunctionParameters!printf (00401085)
add     esp,0x8           ; adjust stack pointer (2 parameters)
00401068:
xor     eax,eax           ; return value 0
add     esp,0x8           ; adjust stack pointer (local variables)
ret
```

With FPO

```
FunctionParameters!arithmetic:
mov     eax,[esp+0x8]       ; address of b
test    eax,eax             ; &b == 0 ?
jnz     FunctionParameters!arithmetic+0xb (0040100b)
xor     al,al               ; return value false (0)
ret
0040100b:
mov     edx,[eax]           ; edx := [b]   (in C: *b)
mov     ecx,[esp+0x4]       ; ecx := [a]
add     edx,ecx
inc     ecx
imul    edx,ecx
mov     [eax],edx           ; [b] := edx
mov     al,0x1              ; return value true (1)
ret
```

Stack overflow

```
STACK_TEXT:
f3fbfffc f3c83163 85216000 00000001 00000001 drv+0x13bf2
f3fc0020 f3c83a05 85216000 85e60424 00000000 drv+0x13163
f3fc0048 f3c831c2 85216000 00000000 000000a8 drv+0x13a05
f3fc006c f3c83a05 00000002 85e60424 00000000 drv+0x131c2
f3fc0094 f3c831c2 85216000 00000000 000000a8 drv+0x13a05
f3fc00b8 f3c83a05 00000002 85e60424 00000000 drv+0x131c2
f3fc00e0 f3c831c2 85216000 00000000 000000a8 drv+0x13a05
f3fc0104 f3c83a05 00000002 85e60424 00000000 drv+0x131c2
…
…


FOLLOWUP_IP:
drv+13bf2
f3c83bf2 55                push    ebp
```

What's next?

- Virtual memory
- Processes and threads
- Calling Windows functions
  (stdcall vs. cdecl)
- Strings
- Pointers to pointers (LPSTR *)
- Structures in memory