Practical Reverse Engineering Exercises - Write Ups

Chapter 1 – Exercise 2 (15th of July 2014)

TASK

"1. Given what you learned about CALL and RET, explain how you would read the value of EIP? Why can't you just do MOV EAX, EIP?

2. Come up with at least two code sequences to set EIP to 0xAABBCCDD.

3. In the example function, addme, what would happen if the stack pointer were not properly restored before executing RET?

4. In all of the calling conventions explained, the return value is stored in a 32-bit register (EAX). What happens when the return value does not fit in a 32-bit register? Write a program to experiment and evaluate your answer. Does the mechanism change from compiler to compiler?"

Excerpt from: *"Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation",*

Bruce Dang, Alexandre Gazet, Elias Bachaalany, Sebastien Josse, ISBN: 978-1-118-78731-1

MY SHORT ANSWERS

1. According to the "Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 1" the EIP register is not designed to be accessed directly by the software and could be affected implicitly only by handful of control flow instructions. In order to read the value of the EIP register one needs to execute CALL instruction to a function. CALL saves the EIP value in the stack as the function return address. While in the function body one could read the return address saved in the stack.
2. Here are some possible options for setting EIP to 0xAABBCCDD

   Version A

   …
   MOV EAX, AABBCCDDh
   JMP EAX

   Version B

   …
   MOV EAX, AABBCCDDh
   CALL EAX
   …

   Version C

   ...
   MOV EAX, AABBCCDDh
   PUSH EAX
   RET

3. The execution will NOT continue from the saved return address but from completely different address
4. In case a function return value exceeds 32 bits, a space is allocated to accommodate the function result and EAX is initialised with pointer to that memory.

Author: ePsiLoN (info at epsilon-labs dot com)

To verify this answer I used a C program which defines a function concatenating 2 strings with result string bigger than 32 bit. I used 2 compilers – GCC under Linux and LCC under Windows. Both compilers implemented the same mechanism.

CODE SNIPPETS

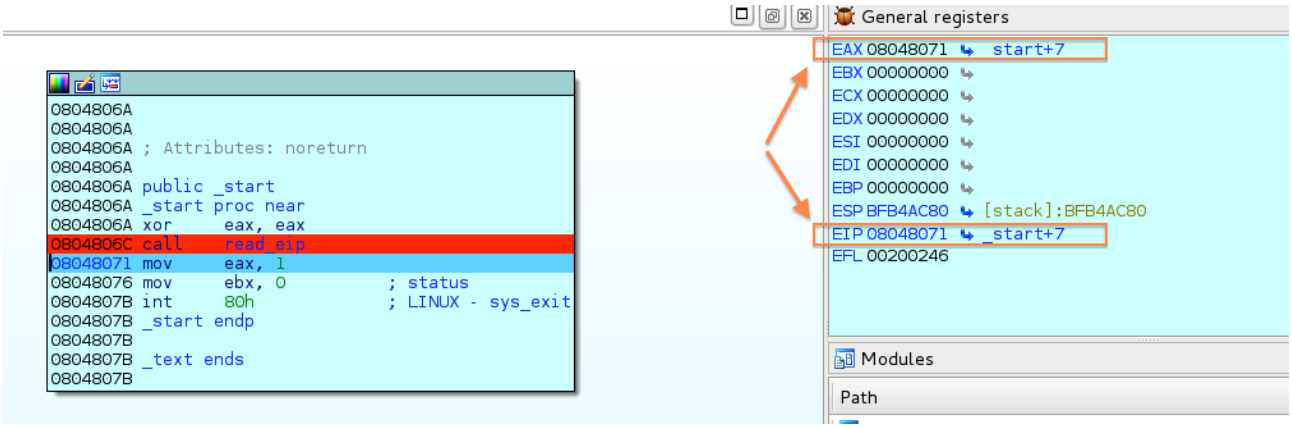
READ EIP VALUE

Here is my suggestion for reading EIP value

ex2_1-intel.asm listing

```
        global _start

section .text

read_eip:

        push ebp

        mov ebp, esp

        mov eax, [ebp+4] ; move the return value in the EAX register

        mov esp, ebp

        pop ebp

        ret

_start:

        xor eax, eax

        call read_eip ; after returning from this function the EAX contains the EIP value

        ; exit gracefully

        mov eax, 1

        mov ebx, 0

        int 080h
```

I used IDA Debuger to trace the program execution and monitor the registers. The following screenshot illustrates a step after the ***call read_eip*** instruction.



SET EIP TO 0XAABBCCDD - VERSION A

ex2_2a.asm listing

```
        global _start

section .text

_start:

        mov eax, 0AABBCCDDh

        jmp eax
```

ex2_2b.asm listing

```
        global _start

section .text

_start:

        mov eax, 0AABBCCDDh

        call eax
```

ex2_2c.asm listing

```
        global _start

section .text

_start:

        mov eax, 0AABBCCDDh
```

Author: ePsiLoN (info at epsilon-labs dot com)

```
        push eax
        ret
```

HANDLING RETURN VALUES BIGGER THAN EAX SIZE

I used the following C program to verify my theory regarding hot function return values bigger than 32b are handled. After compiling it with 2 different compilers I disassembled and traced the executables with IDA. The highlighted lines in the disassembled function clearly indicate that the return value stored in EAX is a pointer to a memory location holding the function result.

ex2_3.c listing

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

char* stringConcat(char* s1, char* s2) {
        char* result = malloc(strlen(s1)+strlen(s2)+1);
        strcpy(result, s1);
    strcat(result, s2);
        return result;
}

int main() {
        char* r;
        char* a = "Test ";
        char* b = "string";
        r = stringConcat(a, b);
        return 0;
```

Disassembled *stringConcat* function (part of the ELF executable generated with GCC)

```
.text:080484BC ; int __cdecl stringConcat(char *s, char *src)
.text:080484BC              public stringConcat
```

```
.text:080484BC stringConcat    proc near            ; CODE XREF: main+28p
.text:080484BC
.text:080484BC dest            = dword ptr -0Ch
.text:080484BC s               = dword ptr  8
.text:080484BC src             = dword ptr  0Ch
.text:080484BC
.text:080484BC                 push    ebp
.text:080484BD                 mov     ebp, esp
.text:080484BF                 push    ebx
.text:080484C0                 sub     esp, 24h
.text:080484C3                 mov     eax, [ebp+s]
.text:080484C6                 mov     [esp], eax      ; s
.text:080484C9                 call    _strlen
.text:080484CE                 mov     ebx, eax
.text:080484D0                 mov     eax, [ebp+src]
.text:080484D3                 mov     [esp], eax      ; s
.text:080484D6                 call    _strlen
.text:080484DB                 add     eax, ebx
.text:080484DD                 add     eax, 1
.text:080484E0                 mov     [esp], eax      ; size
.text:080484E3                 call    _malloc
.text:080484E8                 mov     [ebp+dest], eax
.text:080484EB                 mov     eax, [ebp+s]
.text:080484EE                 mov     [esp+4], eax    ; src
.text:080484F2                 mov     eax, [ebp+dest]
.text:080484F5                 mov     [esp], eax      ; dest
.text:080484F8                 call    _strcpy
.text:080484FD                 mov     eax, [ebp+src]
.text:08048500                 mov     [esp+4], eax    ; src
.text:08048504                 mov     eax, [ebp+dest]
```

```
.text:08048507              mov     [esp], eax      ; dest
.text:0804850A              call    _strcat
.text:0804850F              mov     eax, [ebp+dest]
.text:08048512              add     esp, 24h
.text:08048515              pop     ebx
.text:08048516              pop     ebp
.text:08048517              retn
```

Disassembled *stringConcat* function (part of the PE executable generated with LCC)

```
.text:004012D4 ; int __cdecl stringConcat(char *,char *)
.text:004012D4               public _stringConcat
.text:004012D4 _stringConcat   proc near            ; CODE XREF: _main+2Ep
.text:004012D4
.text:004012D4 var_4         = dword ptr -4
.text:004012D4 arg_0         = dword ptr  8
.text:004012D4 arg_4         = dword ptr  0Ch
.text:004012D4
.text:004012D4               push    ebp
.text:004012D5               mov     ebp, esp
.text:004012D7               push    ecx
.text:004012D8               mov     ecx, 1
.text:004012DD
.text:004012DD loc_4012DD:                          ; CODE XREF: _stringConcat+11j
.text:004012DD               dec     ecx
.text:004012DE               mov     [esp+ecx*4+4+var_4], 0FFFA5A5Ah
.text:004012E5               jnz     short loc_4012DD
.text:004012E7               push    esi
.text:004012E8               push    edi
.text:004012E9               push    [ebp+arg_0]     ; char *
.text:004012EC               call    _strlen
```

Author: ePsiLoN (info at epsilon-labs dot com)

```
.text:004012F1              add     esp, 4
.text:004012F4              mov     edi, eax
.text:004012F6              push    [ebp+arg_4]     ; char *
.text:004012F9              call    _strlen
.text:004012FE              add     esp, 4
.text:00401301              mov     esi, eax
.text:00401303              lea     edi, [edi+esi+1]
.text:00401307              push    edi             ; size_t
.text:00401308              call    _malloc
.text:0040130D              add     esp, 4
.text:00401310              mov     [ebp+var_4], eax
.text:00401313              push    [ebp+arg_0]     ; char *
.text:00401316              push    [ebp+var_4]     ; char *
.text:00401319              call    _strcpy
.text:0040131E              add     esp, 8
.text:00401321              push    [ebp+arg_4]     ; char *
.text:00401324              push    [ebp+var_4]     ; char *
.text:00401327              call    _strcat
.text:0040132C              add     esp, 8
.text:0040132F              mov     eax, [ebp+var_4]
.text:00401332              pop     edi
.text:00401333              pop     esi
.text:00401334              leave
.text:00401335              retn
.text:00401335 _stringConcat   endp
```