Practical Reverse Engineering Exercises - Write Ups
Chapter 1 – Exercise 3 – part5 (question 9) (23th of July 2014)

TASK

"9. Sample L. Explain what function sub_1000CEA0 does and then decompile it back to C."

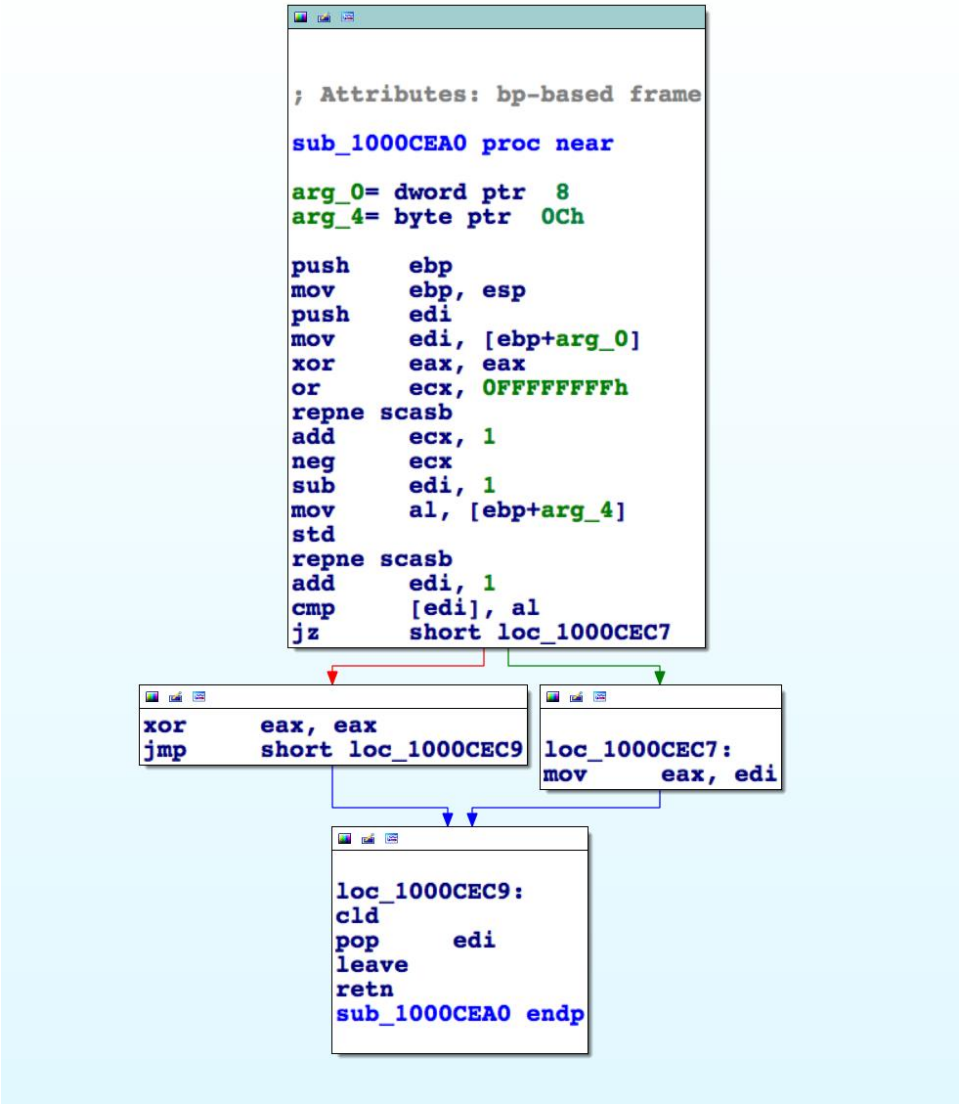
Excerpt from: *"Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation"*,

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

EXERCISE 3.9 – SAMPLE L – FUNCTION SUB_1000CEA0 ANALYSIS

SUB_1000CEA0 is a fairly short function and is quite similar to the code from Chapter 1 – Exercise 1.

Here is how the function looks in IDA disassembler.

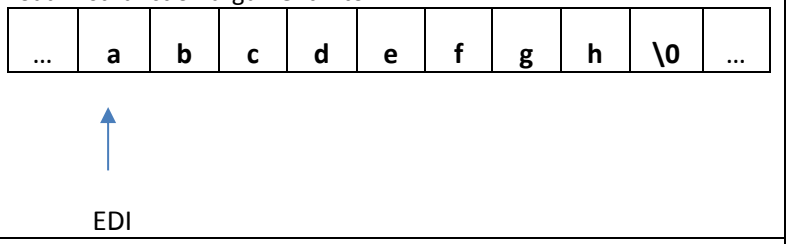
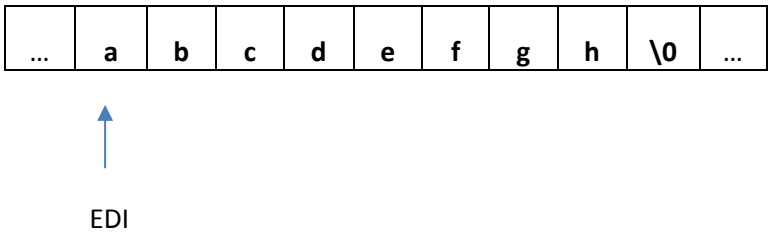
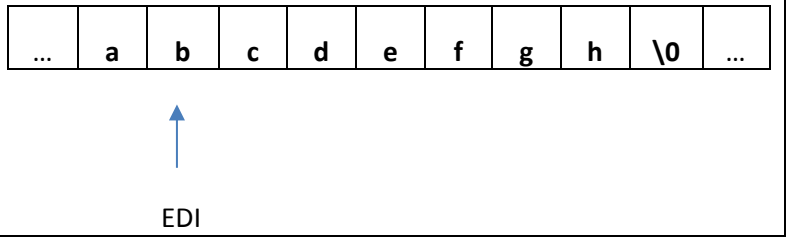


Author: ePsiLoN (info at epsilon-labs dot com)

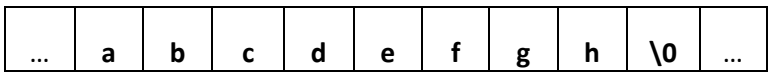
Practical Reverse Engineering Exercises - Write Ups

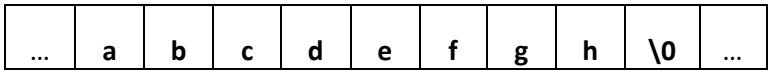
Chapter 1 – Exercise 3 – part5 (question 9) (23th of July 2014)

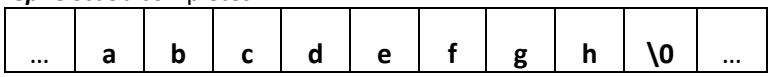
Here is a detailed function walk through:

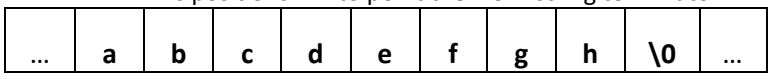
Instructions	Explanation
sub_1000CEA0 proc near arg_0 = dword ptr 8 arg_4 = byte ptr 0Ch	IDA helps out here with meaningful function argument names and sizes. Conclusion about the arguments could be drawn from the code as well
push ebp mov ebp, esp	Function prologue
push edi	Preserve EDI
mov edi, [ebp+arg_0]	Load first function argument into EDI 
xor eax, eax	EAX=0
or ecx, 0FFFFFFFFh	ECX=-1
repne scasb	Search EDI for value in EAX(0). This in fact traverses the string pointed by EDI byte by byte till the null terminator is reached. The operation decreases ECX with each iteration. At the end of the instruction **ECX = -1 -n** (n is the length of the string pointed by EDI including the NULL terminator). At the end of the instruction **EDI** points to byte right after the Null terminator **Before the 1st iteration**  **After the 1st iteration** 

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

<table>
<tr><td></td><td>
After the 2nd iteration



EDI
...
After the 8th iteration



EDI
After the 9th iteration – we have read Null terminator (EAX value) and repne scasb completes


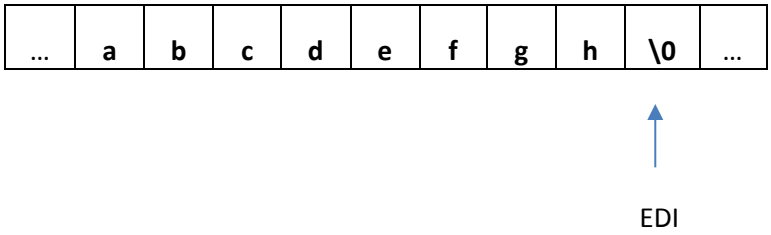

EDI
</td></tr>
<tr><td>add ecx, 1</td><td>ECX = -1 –n +1 = -n</td></tr>
<tr><td>neg ecx</td><td>ECX = n (n is the length of the string pointed by EDI including the NULL terminator)</td></tr>
<tr><td>sub edi, 1</td><td>EDI = EDI -1. This positions EDI to point the NULL string terminator


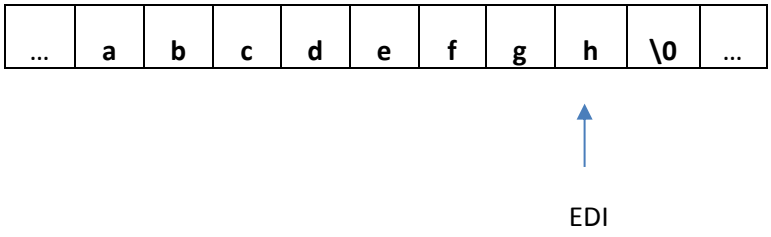

EDI</td></tr>
<tr><td>mov al, [ebp+arg_4]</td><td>Load AL the second function argument. Considering the size of the AL (1 byte) we can assume that the argument is a character

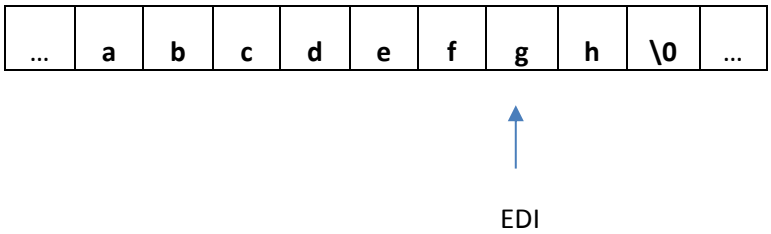
Let say the we load AL with 66h ('f')</td></tr>
<tr><td>std</td><td>DF=1

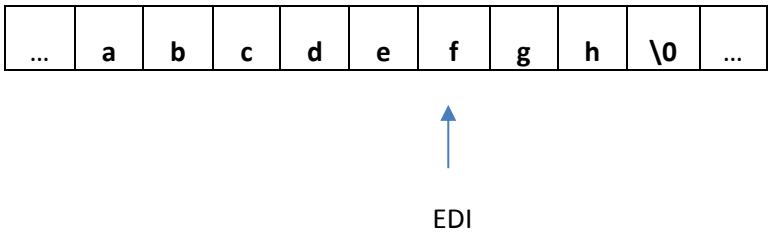
The instruction sets the direction flag. DF determines if EDI registers will be incremented or decremented during the scasb operation. In case DF=1 the EDI will be decremented on every step</td></tr>
</table>

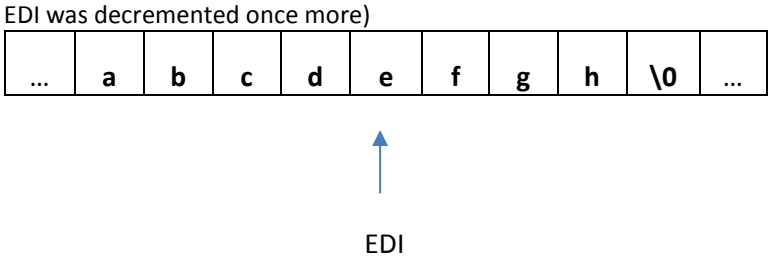
<table>
<tr>
<td>repne scasb</td>
<td>Search the string pointed by EDI for the character loaded in AL starting from the NULL terminator and moving byte by byte towards the beginning of the string. The instruction decrements ECX on every step and will complete either when ECX becomes 0 or when a match is found.

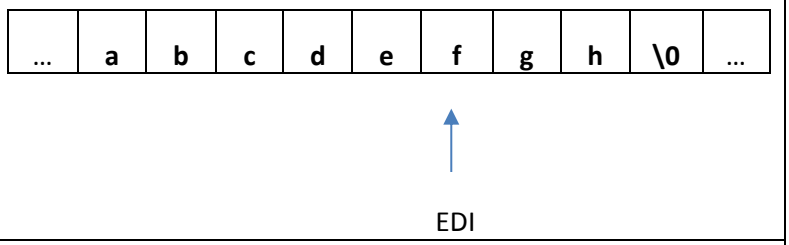
Before the 1st iteration
 

After the 1st iteration
 

After the 2nd iteration
 

After the 3rd iteration
 

After the 4th iteration (we have found f but during that operation the EDI was decremented once more)
 
</td>
</tr>
<tr>
<td>add edi, 1</td>
<td>EDI = EDI + 1</td>
</tr>
</table>

	In case repne scasb has found a match the EDI has been already decreased so we need to return EDI one stap back so it points to the position of in the string with symbol matching AL. 
cmp [edi], al	Check if EDI points to byte equal to the one stored in AL. If so that would mean that a match is found and if not that means that ECX became 0 before finding match i.e the string does not contain the character stored in AL
jz short loc_1000CEC7	If EDI points to the same character as the one stored in AL jumpt to loc_1000CEC7
xor eax, eax jmp short loc_1000CEC9	EAX=0 Result will be 0 - no match was found
loc_1000CEC7: mov eax, edi	This is the case where a matching symbol is found. The value of the EDI is moved to EAX i.e. the result of the function execution will be a pointer to the position in the string where a matching symbol was found
loc_1000CEC9: cld	This is where the common exit code for both program branches starts. Clear the DF DF=0
pop edi	Restore EDI
leave retn sub_1000CEA0 endp	Function epilogue

After analysing the function behaviour we could conclude that it searches for the last occurrence of a character in a string. It receives two parameters – char* and char (which is equivalent to int in ANSI C).

EXERCISE 3.9 – SAMPLE L – FUNCTION SUB_1000CEA0 – C EQUVALENT

There is a standard function ***strrchr*** included in the **string.h** library which has the same behaviour as FUNCTION SUB_1000CEA0 and which has the following prototype:

char *strchr(const char *s, int c);

Here are two possible implementations of the function. The functions behaviour could be tested using sample code from my github (ex3_9-c.c)

strtchrA() and strrchrB()

```
char *strrchrA(const char *string, char c) {
        char *result=0;
        do {
                if (*string==c) {
                        result = (char*) string;
                }
        } while (*string++);
        return result;
}



char *strrchrB(const char *string, char c) {
        if (!string)
                return 0;
        char *result=0;
        while(*string!='\0'){
                if(*string==c)
                        result = (char*)string;
                *string++;
        }
        return result;
}
```

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