Robin Ward
COMP 7720/7726/4970
Summer 2018
7/4/2018
Auburn University

1. There is more than one clue that i could tell by looking at the assembly code.
    a. The first and most obvious would be the value of retn at the end of the function assembly.
        i. RtlUpperString
            1. .text:7C95F82A retn 8
            2. The value 8 is hex which in decimal is equal to 8.
            3. We are assuming that every element is 4 bytes, so 8/4=2 arguments
        ii. RtlEqualString
            1. .text:7C9135F4 retn 0Ch
            2. The value 0c is hex which in decimal is equal to 12.
            3. We are assuming that every element is 4 bytes, so 12/4=3 arguments
    b. The second way to find out in IDA the number of arguments is to look at the following line in the assembly. These x's represent the number of arguments
        i.
```
.text:7C95F7E9 ; Attributes: bp-based frame
.text:7C95F7E9
.text:7C95F7E9 ; __stdcall RtlUpperString(x, x)
.text:7C95F7E9                 public _RtlUpperString@8
.text:7C95F7E9 _RtlUpperString@8 proc near             ; DATA XREF: .text:off
.text:7C95F7E9
.text:7C95F7E9 arg_0           = dword ptr  8
.text:7C95F7E9 arg_4           = dword ptr  0Ch
```
        ii.
```
C9135D6
C9135D6 ; Attributes: bp-based frame
C9135D6
C9135D6 ; __stdcall RtlEqualString(x, x, x)
C9135D6                 public _RtlEqualString@12
C9135D6 _RtlEqualString@12 proc near           ; CODE XREF: RtlEqualDomainNam
C9135D6                                        ; DATA XREF: .text:off_7C90342
C9135D6
C9135D6 arg_0           = dword ptr  8
C9135D6 arg_4           = dword ptr  0Ch
C9135D6 arg_8           = byte ptr  10h
```
2. On x86, jl and jg do the signed compare, while ja and jb do the unsigned compare. **.text:7C95F803 cmp ax, dx** is performing a jump if below or equal so this would be an **unsigned comparison**.
3. The reason the function is using MOVZX is because it needs to increase the unsigned integer size. MOVZX copies a smaller operand into a larger one and zero extends it on the way. Remember, we are comparing unsigned integers from the previous question.

4. ax=0; dx=8
5. The jump to loc_7C95F827 will happen when the zero flag is set to 1. This will occur if the value of eax is equal to 0. Test eax, eax is testing to see if the eax register is equal to zero. If it is, then perform the jump operation.
6. At the beginning of the loop we are defining our variables. Xor eax, eax means that we are setting eax=0. We are then moving the pointer of EDI into AL. the purpose of this loop is to change the source string to uppercase. It does this by calling the RtlUpperChar to each Char in the source string. It will exit the loop once the end of the source string has been reached. This is the purpose of the dec ebx.
7. Assuming that str1 is the destination string, the value of str1= str1{length=8, MaximumLength=8, Buffer="GOOD BYE"}
8. There are two circumstances that the jump to loc_7c92930D will be taken
    a. The jump to loc_7c9135f7 has to trigger and not be skipped
    b. The comparison at line .text:7C9135FF has to trigger the jnz to loc_7C92930D
9. There are many differences between these three cmp instructions. I thought it would be easier to present them in a table.

	.text:7C913616	.text:7C92931B	.text:7C929337
cmp = cl,dl?	yes	yes	No, cmp = al,bl
Does the cmp follow with a jz jump?	yes	yes	no
Does the cmp follow with a jnz jump?	no	no	yes
Are there other instructions before the jump?	no	Yes, there are 2 mov instructions	no
Is there a possible loop?	yes	yes	no
Is the jump creating a loop?	yes	yes	n/a

10. As mentioned in the question, we are assuming that the cmp condition at address 7C9293CC is satisfied and the jg jump to 7C94C431 is taken. Below are the three main cases that will control the branching within the code based on arg1

a. .text:7C94C4AC - cmp word ptr [edx+ecx*2], 0 ; ecx has al, which has arg1
b. .text:7C94C51D - cmp byte ptr [ebp+arg_0+1], 0 ; this is a direct compare of arg1
c. .text:7C9293D4 - cmp al, 61h ; al has arg1