Assignment 3, Fall 2018
CS4630, Defense Against the Dark Arts
Reverse Engineering

Purpose

This assignment will help you understand the process of reverse engineering as well as deepen your understanding of x86 stack operations and the x86 calling convention.

In the first part of the assignment, you will analyze the relationship between a program written in C (high-level programming language) to its corresponding assembly code equivalent. In the second part of the assignment, you will reverse engineer the functionality of a program by disassembling an x86 32-bit ELF binary executable with the `objdump` utility and then analyzing the results.

Due

This assignment is due on **Tuesday, 2-OCT-2018 at 11:59 pm**

Prerequisites

You should understand the basic operation of the x86 hardware stack and the instructions that affect it.

Assignment Details

Examine the following C code:

```
#include <stdio.h>
#define BUFSIZE 16

int baz(int value, int vector[], int len) {
    int i;
    for (i = 0; i < len; i++)
       vector[i] = value;
    return len;
}

int main() {
    int x, i, sum;
    int buffer[BUFSIZE];
    x = baz(8, buffer, BUFSIZE);
    sum = 0;
    for (i = 0; i < BUFSIZE; i++)
       sum += buffer[i];
```

```
    printf ("Sum is %d\n", sum);
    return 0;
}
```

The assembly code that follows was produced for these two functions by *gcc* on Ubuntu. The code was produced using the following command:

`gcc -m32 -c -S -fno-stack-protector -fno-pie -no-pie -fno-asynchronous-unwind-tables code.c`

For answering the questions in this assignment, do not compile this source code, as the results are compiler-version dependent. Instead, use the copy of this assembly code, `code.s`, provided in Collab as an attachment.

```
        .file   "code.c"
        .text
        .globl  baz
        .type   baz, @function
baz:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    $0, -4(%ebp)
        jmp     .L2
.L3:
        movl    -4(%ebp), %eax
        leal    0(,%eax,4), %edx
        movl    12(%ebp), %eax
        addl    %eax, %edx
        movl    8(%ebp), %eax
        movl    %eax, (%edx)
        addl    $1, -4(%ebp)
.L2:
        movl    -4(%ebp), %eax
        cmpl    16(%ebp), %eax
        jl      .L3
        movl    16(%ebp), %eax
        leave
        ret
        .size   baz, .-baz
        .section        .rodata
.LC0:
        .string "Sum is %d\n"
        .text
        .globl  main
        .type   main, @function
main:
        leal    4(%esp), %ecx
```

```
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $84, %esp
        pushl   $16
        leal    -84(%ebp), %eax
        pushl   %eax
        pushl   $8
        call    baz
        addl    $12, %esp
        movl    %eax, -20(%ebp)
        movl    $0, -16(%ebp)
        movl    $0, -12(%ebp)
        jmp     .L6
.L7:
        movl    -12(%ebp), %eax
        movl    -84(%ebp,%eax,4), %eax
        addl    %eax, -16(%ebp)
        addl    $1, -12(%ebp)
.L6:
        cmpl    $15, -12(%ebp)
        jle     .L7
        subl    $8, %esp
        pushl   -16(%ebp)
        pushl   $.LC0
        call    printf
        addl    $16, %esp
        movl    $0, %eax
        movl    -4(%ebp), %ecx
        leave
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (Ubuntu 7.3.0-16ubuntu3) 7.3.0"
        .section        .note.GNU-stack,"",@progbits
```

Examine the source code and the assembly language. Relate the assembly code back to the source code. For example, can you identify the section of the assembly code that corresponds to the `for` loop in function `main`?

After you have developed an understanding of the assembly code answer the following questions. Please use a text editor to edit the text file template (called `answers.txt`) to include:

- Your name

- Your UVA computing ID
- Honor code, and
- Your answers to the questions

The text file containing your answers is what you will submit.
NOTE: When a question asks for the address of a variable, your answer should be of the form of the effective/register-relative address of the variable. For example, `55(%ebp)`, `25(%esp)`, `10(%eax)`.

Questions to Answer

Part 1 - Assembly Code Analysis

Answer the following questions based on your analysis of the assembly code in the file `code.s`.

1. What is the address of local variable `i` of function `main`?
2. What is the address of local variable `sum` of function `main`?
3. What is the address of local variable `x` of function `main`?
4. What is the address of local variable `buffer` of function `main`?
5. What is the address of the address of the parameter `vector` of function `baz`?
6. What is the address of parameter `len` of function `baz`?
7. What is the address of parameter `value` of function `baz`?
8. What is the address of local variable `i` of function `baz`?

Part 2 - Analyzing Disassembled Code

Answer the following questions based on your analysis of the disassembled code in `funcs.dis.txt` that you generate (on your Ubuntu 18.04.1 LTS VM) for `funcs.exe`.

The commandline you might want to use is:

```
objdump -d funcs.exe > funcs.dis.txt
```

Examine the disassembly and answer the following questions. You can use the debugger `gdb` to set breakpoints, examine memory, and observe the execution of the program. We have provided a `gdb` cheatsheet to help you. Also, remember that you can search the web for help. In particular, you may want to search for the IA32 Software Developer's Manual which describes each assembly code instruction.

1. List the names of the functions called in `main`.
2. How many parameters does the function `f1` take?
3. How many parameters does the function `f2` take?

4. How many parameters does the function `f3` take?
5. Does `f1` have any local variables? If so, how many and at what memory addresses?
6. Does `f2` have any local variables? If so, how many and at what memory addresses?
7. Does `f3` have any local variables? If so, how many and at what memory addresses?
8. Describe the calculation that function `f1` performs.
9. Describe the calculation that function `f2` performs.
10. Describe the calculation that function `f3` performs.

Items to Submit

The due date for this assignment is: **Tuesday, 2-OCT-2018 at 11:59 pm** . Please submit your completed `answers.txt` file to Assignment 3 on Collab.