ENGR-E 399/599: Embedded Systems Reverse Engineering
Indiana University — Spring 2021
Prepared by: Austin Roach — Assigned: January 28, 2021

Assignment 1: ARM disassembly

Due date: February 19, 2021

This assignment is all about analyzing ARM assembly language, primarily focusing on the contents of the `binary_water_balloon_arm` Linux ELF. Two additional sections of the assignment focus on the identification of standard library functions.

Since `binary_water_balloon_arm` is compiled for ARM, you will not be able to execute it natively on an x86 system or x86 virtual machine. Instead, you will need to emulate the ARM instruction set architecture. Once you have installed the `qemu-user` package, you can execute the binary with:

```
qemu-arm ./binary_water_balloon_arm
```

In the `resources` directory of this assignment you can find a copy of the ARM v7A-R architecture reference manual. To allow easy access from Ghidra's "Processor Manual" command, copy the reference manual PDF to the `Ghidra/Processors/ARM/data/manuals/` directory of your Ghidra installation.

0.1 Program interaction

There are six independent stages to the binary water balloon, each of which requires a different input to pass. You can type in each of these inputs manually each time the program executes. Or you can pass a filename as an argument when you run the program. The program will read each line from that file as an input for a separate stage. If you have fewer than six lines in your file, the program will request input for later stages on standard input.

If you want to skip a stage in order to attempt a solution on a later stage, you can provide the following input for the phase that you wish to skip:

```
PASS
```

0.2 General program flow

The program fetches a line of input before each stage. It passes a pointer to the input string as the sole argument to the function for each stage (`phase_1()`, `phase_2()`, `phase_3()`, etc.) Invalid inputs will result in `balloon_splat()` being called within the relevant stage's function. You won't need to reverse-engineer any of the details of `main()` or its supporting functions to complete the assignment, though you are free to do so if you wish.

0.3 Saving Ghidra program for export

To share your marked-up Ghidra program, select "File→Export Program" from the Code-Browser menu. Choose "Ghidra Zip File" as the export format. Submit this file along with your write-up, and include your name in the filename that you choose.

0.4 Dynamic analysis

You may find it helpful to use dynamic analysis techniques to improve your understanding of specific points of the program. To do this, you will want to install the `gdb-multiarch` package in Ubuntu. You can have QEMU launch a GDB server by specifying a port number with the `-g` command line switch, for example:

```
qemu-arm -g 1234 ./binary_water_balloon_arm
```

In a different terminal, you can then run

```
gdb-multiarch binary_water_balloon_arm
```

and connect to the server with the command

```
(gdb) target remote localhost:1234
```

From there, you can interact with the program using standard GDB commands. It is strongly recommended that you use dynamic analysis techniques judiciously. They are very powerful, but it would be very time consuming to step through an entire program instruction-by-instruction.

0.5 References

Azeria Labs ARM assembly overview
Maria Markstedter
https://azeria-labs.com/writing-arm-assembly-part-1/

Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation
Chapter 2: ARM
Bruce Dang, Alexandre Gazet, Elias Bachaalany, and Sébastian Josse
John Wiley & Sons
ISBN-13: 978-1-118-78731-1
https://iu.skillport.com/skillportfe/main.action?assetid=62680

1 Binary water balloon general program information (1 point)

First, document some general information about the binary. From the output of the `file` command:

- What is the target architecture of the binary?
- Is the binary statically or dynamically linked?
- Are symbols stripped?

Run the `strings` command. Do you see some strings that look relevant to your interaction with the program?

Note some general details about this architecture:

- What is the register size of this architecture?
- What is the endianness of this architecture?
- How are arguments passed to functions? (We discussed this in class, but you can also see the Procedure Call Standard for the ARM Architecture for offical documentation. This document is included in the `resources` directory.)
- How is an integer return value from a function passed back to the calling function?

2 Binary water balloon analysis (2 points/stage; 12 points total)

Now load the program into Ghidra. The default analysis settings will be sufficient. Analyze the functions `phase_1()` through `phase_6()`. Within the assembly listing, add comments noting:

- Information about how arguments are passed to each function called from the function that you are analyzing: (contents, types, and storage location of arguments)
- Information about values returned from each of those function calls (contents, types, and storage location of returned values)
- The conditions associated with branch points

You can expect to have one comment for every 4-5 instructions in your assembly listing. The `resources/demo` directory contains a Ghidra file where the `evaluate_input()` file has been commented as a demonstration. *Note that this documentation is the most important part of this assignment.* The goal is to develop a systematic approach to understanding how information flows through the program and for how control flow decisions are made. Even if you have difficulty putting the pieces together to generate a valid input for a stage, having good documentation of these details will earn you partial credit for that stage.

In your write-up for each stage, document the set of valid inputs that would pass each stage. For some stages, there is a single unique input. For others, there is a family of inputs that have some relationship that you should describe.

3 Standard library function identification #1 (1 point)

Below you will find the ARM assembly listing for a single C standard library function. For this function list:

- How many arguments does this function take?
- What are the types of the arguments? (Pointer, integer, etc.)
- What is the type of the returned value?
- Describe what the function does. If you recognize this standard library function, you can simply list the name. If you do not, describe the functionality as best you can.

```
00000000 02 20 80 e0     add             r2,r0,r2
                     LAB_00000004
00000004 02 00 50 e1     cmp             r0,r2
00000008 05 00 00 0a     beq             LAB_00000024
0000000c 01 30 d0 e4     ldrb            r3,[r0],#0x1
00000010 01 c0 d1 e4     ldrb            r12,[r1],#0x1
00000014 0c 30 53 e0     subs            r3,r3,r12
00000018 f9 ff ff 0a     beq             LAB_00000004
                     LAB_0000001c
0000001c 03 00 a0 e1     cpy             r0,r3
00000020 1e ff 2f e1     bx              lr
                     LAB_00000024
00000024 00 30 a0 e3     mov             r3,#0x0
00000028 fb ff ff ea     b               LAB_0000001c
```

4 Standard library function identification #2 (1 point)

Below you will find the ARM assembly listing for a single C standard library function. For this function list:

- How many arguments does this function take?
- What are the types of the arguments? (Pointer, integer, etc.)
- What is the type of the returned value?
- Describe what the function does. If you recognize this standard library function, you can simply list the name. If you do not, describe the functionality as best you can.

```
00000000 00 30 a0 e1     cpy              r3,r0
00000004 02 20 80 e0     add              r2,r0,r2
                     LAB_00000008
00000008 02 00 53 e1     cmp              r3,r2
0000000c 1e ff 2f 01     bxeq             lr
00000010 00 c0 d1 e5     ldrb             r12,[r1,#0x0]
00000014 01 c0 c3 e4     strb             r12,[r3],#0x1
00000018 00 00 5c e3     cmp              r12,#0x0
0000001c 01 10 81 12     addne            r1,r1,#0x1
00000020 f8 ff ff ea     b                LAB_00000008
```