Embedded Systems Lab

CPE 325-02

Intro to Software Reverse Engineering

By: David Thornton

Lab Date: November 5, 2020

Lab Due: November 16, 2020

Demonstration Due: November 16, 2020

Introduction

This lab introduces basic concepts, tools, and techniques in software reverse engineering.

Lab Assignment

1. There are at least 5 questions embedded inside the tutorial for Lab 10. Answer at least 4 of them.

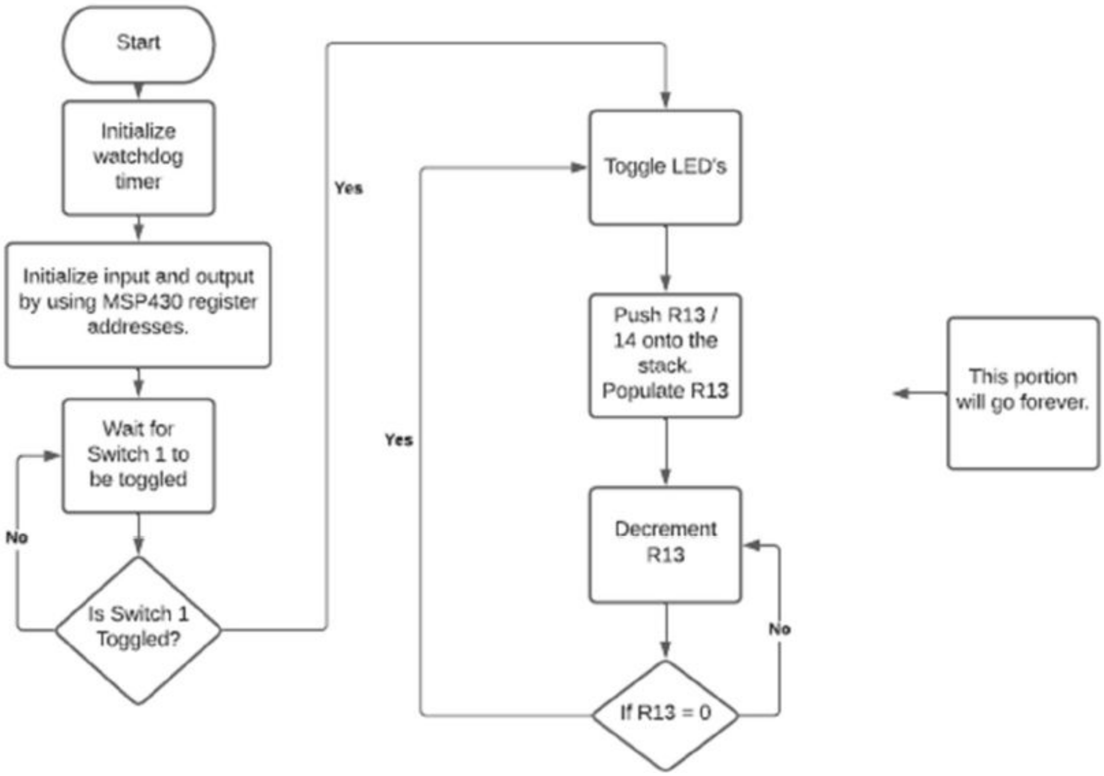
2. Please find the executable file crack_me.out. This executable file has a bunch of usernames and passwords.
   a. Find as many usernames and passwords that you can find. Document the entire process in your report and elaborate to your instructor during demonstration.
   b. Connect your MSP-EXP430F5529 board to your computer. Make the UART connection at 115200 baud rate.
   c. Program the board with the provided .out file and try to guess the correct password. Upon successfully guessing the password, you will see `"and it's CORRECT!!!!!!!!!!!!!"` message. Take a screenshot and put it in your report.

3. From the same .out file from Q2, find the following relevant information. What tool did you use? Take a screenshot and put in your report.
   a. What is the magic number used?
   b. What is the class of this .out file?
   c. What machine was this file built for?
   d. What is the size of the header?
   e. How many section headers are there? Please verify. You many need to run the command again.

4. You are given an executable file reverse_me.txt. This is a hex file generated using the process described in Section 5.1 in the tutorial.
   a. Program the given hex file to your microcontroller using MSP430Flasher tool, and paste the output in your report. In your demonstration, you must show the whole process.
   b. Guess from observation on the board what the program does?
   c. Using the naken utility and the steps shown in Section 5.2 of the tutorial, reverse engineer the hex file to assembly code.
   d. Comment on each line of the assembly code generated from Q4c above to describe what each line is doing.
   e. Describe what the program is doing in a neat flowchart. You can also write a paragraph to describe in addition to the flowchart.

Flow Chart

Figure 1: Question 4.e Flow Chart



Observations

1. After searching the tutorial document, I found and answered the following questions.
    a. Do you think Figure 14 belongs to the same microcontroller as in Figure 13?
        i. Figure 13 and 14 most likely belong to different microcontrollers due to the fact that the same section starts at different addresses in memory. For example the .data section starts at address 0x2400 in figure 13, but .data starts at 0x1100 in figure 14.

    b. Where do you find this information about register addresses?
        i. You can find this information in the linker command file. For example it may be in the directory "C:\ti\ccs1010\ccs\ccs_base\msp430\include\lnk_msp430f5529.cmd." This location will vary based on your installation of CCS. Once the file is opened, you can search it for the desired address.

c. Similarly, the next instruction ANDs content of 0x0223 with 127. Can you guess what this statement does?
    i. This instruction ANDs the last bits of 0x0223 with 127 (decimal). This will likely turn the LED on port 0x0223 off. This is because in the cmd file, 0x0223 is PBOUT_H.

d. The next instruction moves 0x4432 to R0 (PC). This means the PC points to 0x4432. Where command is executed after this?
    i. xor.b #1, &0x0202 will be executed. 0x0202 is PAOUT. This toggles the other LED. When one LED turns off, the other LED is toggled.

2. The output of the command `.\msp430-elf-objdump.exe -s crack_me.out` is shown below. This command must be run from the directory (containing crack_me.out): `C:\ti\ccs1010\ccs\tools\compiler\msp430-gcc-9.2.0.50_win64\bin`
    a. Username: `Abraham_uname`, password: `lincoln_pass`. To find this output, you must `CTRL + C` after a short time (~5s) to terminate the execution, and find the `.const` section of the output. This command will infinitely output all of the different sections and their data in strings and in hex along with their addresses.

```
80ee 00000000 00000000 00000000 00000000  ................
80fe 00000d0a 53747564 656e7420 22257322  ....Student "%s"
810e 20656e74 65726564 20746865 20677565   entered the gue
811e 73736564 20706173 73776f72 642e0a0d  ssed password...
812e 00002061 6e642069 74732057 524f4e47  .. and its WRONG
813e 21212121 21212121 21212121 2121210a  !!!!!!!!!!!!!!!.
814e 0d002061 6e642069 74732043 4f525245  .. and its CORRE
815e 43542121 21212121 21212121 21212121  CT!!!!!!!!!!!!!!
816e 210a0d00 30313233 34353637 38394142  !...0123456789AB
817e 43444546 00003031 32333435 36373839  CDEF..0123456789
818e 61626364 65660000 25000a57 68617420  abcdef..%..What
819e 69732079 6f757220 67756573 73656420  is your guessed
81ae 70617373 776f7264 3f0a0d00 57686174  password?...What
81be 20697320 796f7572 20636861 72676572   is your charger
81ce 2049443f 0a0d0000 2a2a2a2a 2a2a2a2a   ID?....********
81de 2a2a2a2a 2a2a2a2a 0a0d0000 41627261  ********....Abra
81ee 68616d5f 756e616d 65006c69 6e636f6c  ham_uname.lincol
81fe 6e5f7061 737300                      n_pass.
```

    b. The following output from PuTTY displays both successful and unsuccessful attempts at logging in.

```
****************
What is your charger ID?
abc123
What is your guessed password?
*******
Student "abc123" entered the guessed password.
 and its WRONG!!!!!!!!!!!!!!
****************
What is your charger ID?
Abraham_uname
What is your guessed password?
*************
Student "Abraham_uname" entered the guessed password.
 and its CORRECT!!!!!!!!!!!!!!!
```

3. The output of the command `.\msp430-elf-readelf.exe -h crack_me.out` is shown below. This command must be executed from the directory (containing crack_me.out): `C:\ti\ccs1010\ccs\tools\compiler\msp430-gcc-9.2.0.50_win64\bin`
    a. Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
    b. Class: ELF32
    c. Machine: Texas Instruments msp430 microcontroller
    d. Size of this header:: 52 (bytes)
    e. Number of section headers: 99

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Texas Instruments msp430 microcontroller
  Version:                           0x1
  Entry point address:               0x7e7c
  Start of program headers:          184384 (bytes into file)
  Start of section headers:          184544 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         5
  Size of section headers:           40 (bytes)
  Number of section headers:         99
  Section header string table index: 98
```

4.
    a. The output of the command `C:\ti\MSPFlasher_1.3.20\MSP430Flasher.exe -n MSP430F5529 -w reverse_me.txt -v -z [Vcc]` is shown below. This command must be executed from the directory (containing reverse_me.txt): `C:\ti\ccs1010\ccs\tools\compiler\msp430-gcc-9.2.0.50_win64\bin`.

This command also assumes that the MSP430Flasher tool is installed in the corresponding directory, `C:\ti\MSPFlasher_1.3.20`.

```
* -----/|-------------------------------------------------------------------- *
*     / |__                                                                   *
*    /_   /   MSP Flasher v1.3.20                                             *
*      | /                                                                    *
* -----|/-------------------------------------------------------------------- *
*
* Evaluating triggers...done
* Checking for available FET debuggers:
* Found USB FET @ COM4 <- Selected
* Initializing interface @ COM4...done
* Checking firmware compatibility:
* FET firmware is up to date.
* Reading FW version...
* Debugger does not support target voltages other than 3000 mV!
* Setting VCC to 3000 mV...done
* Accessing device...done
* Reading device information...done
* Loading file into device...done
* Verifying memory (reverse_me.txt)...done
*
* ----------------------------------------------------------------------------
* Arguments   : -n MSP430F5529 -w reverse_me.txt -v -z [Vcc]
* ----------------------------------------------------------------------------
* Driver      : loaded
* Dll Version : 31400000
* FwVersion   : 31200000
* Interface   : TIUSB
* HwVersion   : E 3.0
* JTAG Mode   : AUTO
* Device      : MSP430F5529
* EEM         : Level 7, ClockCntrl 2
* Erase Mode  : ERASE_ALL
* Prog.File   : reverse_me.txt
* Verified    : TRUE
* BSL Unlock  : FALSE
* InfoA Access: FALSE
* VCC ON      : 3000 mV
* ----------------------------------------------------------------------------
* Starting target code execution...done
* Disconnecting from device...done
*
* ----------------------------------------------------------------------------
* Driver      : closed (No error)
* ----------------------------------------------------------------------------
*/
```

b. From observation and experimentation, this code will blink both LEDs infinitely after S1 is pressed (P2.1).

c. The output of the command `C:\ti\MSPFlasher_1.3.20\MSP430Flasher.exe -n MSP430F5529 -r [reverse_me.txt, MAIN]` is shown below. This command must be executed from the directory (containing reverse_me.txt): `C:\ti\ccs1010\ccs\tools\compiler\msp430-gcc-9.2.0.50_win64\bin`. This command also assumes that the MSP430Flasher tool is installed in the corresponding directory, `C:\ti\MSPFlasher_1.3.20`.

After this command is executed, execute `C:\ti\naken_asm-2020-04-25\naken_util.exe -msp430 -disasm reverse_me_stripped.txt > reverse_me.txt`. The result of this command is that the asm code will be contained in `reverse_me.txt`.

```
* -----/|-------------------------------------------------------------- *
*     / |__                                                            *
*    /_   /   MSP Flasher v1.3.20                                      *
*      | /                                                             *
* -----|/--------------------------------------------------------------- *
*
* Evaluating triggers...done
* Checking for available FET debuggers:
* Found USB FET @ COM4 <- Selected
* Initializing interface @ COM4...done
* Checking firmware compatibility:
* FET firmware is up to date.
* Reading FW version...done
* Setting VCC to 3000 mV...done
* Accessing device...done
* Reading device information...done
* Dumping memory from MAIN into reverse_me.txt...done
*
* ----------------------------------------------------------------------
* Arguments   : -n MSP430F5529 -r [reverse_me.txt MAIN]
* ----------------------------------------------------------------------
* Driver      : loaded
* Dll Version : 31400000
* FwVersion   : 31200000
* Interface   : TIUSB
* HwVersion   : E 3.0
* JTAG Mode   : AUTO
* Device      : MSP430F5529
* EEM         : Level 7, ClockCntrl 2
* Read File   : reverse_me.txt (memory segment = MAIN)
* VCC OFF
* ----------------------------------------------------------------------
* Powering down...done
* Disconnecting from device...done
*
* ----------------------------------------------------------------------
* Driver      : closed (No error)
* ----------------------------------------------------------------------
*/
```

d. Assembly code generated from Q4.c with comments on every instruction

```
Addr Opcode Instruction Cycles
------- ------ ---------------------------------- ------
0x4400: 0x40b2 mov.w #0x5a80, &0x015c 5 // Setting the watchdog timer
0x4402: 0x5a80
0x4404: 0x015c
0x4406: 0xd3d2 bis.b #1, &0x0204 4 // Setting PADIR
0x4408: 0x0204
0x440a: 0xd0f2 bis.b #0x80, &0x0225 5 // Setting PBDIR_H
0x440c: 0x0080
0x440e: 0x0225
0x4410: 0xc3e2 bic.b #2, &0x0204 4 // Clearing PADIR
0x4412: 0x0204
0x4414: 0xd3e2 bis.b #2, &0x0202 4 // Setting output direction P2.2 for
PAOUT
0x4416: 0x0202
```

```
0x4418: 0xd3e2 bis.b #2, &0x0206 4 // Setting PAREN
0x441a: 0x0206
0x441c: 0xc3e2 bic.b #2, &0x0205 4 // Clearing PADIR_H
0x441e: 0x0205
0x4420: 0xd3e2 bis.b #2, &0x0203 4 // Setting PAOUT_H
0x4422: 0x0203
0x4424: 0xd3e2 bis.b #2, &0x0207 4 // Setting PAREN_H
0x4426: 0x0207
// What's about to happen below is the auto generation waiting for an
input, but it does it I think 3 times. As in it waits for input 3 times,
maybe debouncing? Not sure.
0x4428: 0xb3e2 bit.b #2, &0x0200 4 // Anding with PAIN, a test. Most likely
for the button press.
0x442a: 0x0200
0x442c: 0x23fd jne 0x4428 (offset: -6) 2 // And then if the switch has not
been pressed, wait for the button to be pressed.
0x442e: 0x120d push.w r13 3 // If the button is pressed, push R13 onto the
stack
0x4430: 0x403d mov.w #0x031d, r13 2 // Populate a large value
0x4432: 0x031d
0x4434: 0x831d sub.w #1, r13 1 // Sub R13
0x4436: 0x23fe jne 0x4434 (offset: -4) 2 // Loop until 0
0x4438: 0x413d pop.w r13 -- mov.w @SP+, r13 2 // Then pop R13
0x443a: 0x3c00 jmp 0x443c (offset: 0) 2 // And jump to 443c, which is the
next line? This is funny how it generates.
0x443c: 0xb3e2 bit.b #2, &0x0200 4 // Same thing as 0x4428. It waits for a
button to be pressed or some tinput.
0x443e: 0x0200
0x4440: 0x23f3 jne 0x4428 (offset: -26) 2 // If it is not set jump to 4428
to restart (loop).
0x4442: 0xb3e2 bit.b #2, &0x0201 4 // Test the input high value
0x4444: 0x0201
0x4446: 0x23fd jne 0x4442 (offset: -6) 2 // If it is not set go back to
4442 ti wait for the high value to be set.
0x4448: 0x120d push.w r13 3 // Push r13
0x444a: 0x403d mov.w #0x031d, r13 2 // Populate a value
0x444c: 0x031d
0x444e: 0x831d sub.w #1, r13 1 // Sub 1
0x4450: 0x23fe jne 0x444e (offset: -4) 2 // Until 0, loop.
0x4452: 0x413d pop.w r13 -- mov.w @SP+, r13 2 // Pop R13 and push it onto
the stack.
0x4454: 0x3c00 jmp 0x4456 (offset: 0) 2 // Jump to next line lol
```

```
0x4456: 0xb3e2 bit.b #2, &0x0201 4 // Test the high state of the input
again
0x4458: 0x0201
0x445a: 0x23f3 jne 0x4442 (offset: -26) 2 // The following code below is a
loop to toggle the LED by decrementing
0x445c: 0xe3d2 xor.b #1, &0x0202 4 // the register values and then toggling
when they're 0.
0x445e: 0x0202
0x4460: 0xe0f2 xor.b #0x80, &0x0223 5 // Toggle the LED
0x4462: 0x0080
0x4464: 0x0223
0x4466: 0x120d push.w r13 3 // Push R13 and R14 onto the stack.
0x4468: 0x120e push.w r14 3 //
0x446a: 0x403d mov.w #0x2844, r13 2 // Move 2844 into R13 to reset its
value.
0x446c: 0x2844
0x446e: 0x431e mov.w #1, r14 1 // Move #1 into register R14
0x4470: 0x831d sub.w #1, r13 1 // Take 1 off of register R13
0x4472: 0x730e subc.w #0, r14 1 // Subtract with carry off of R14
0x4474: 0x23fd jne 0x4470 (offset: -6) 2 // If the Z flag is not set, keep
decrementing.
0x4476: 0x930d cmp.w #0, r13 1 // Compare 0 to R13.
0x4478: 0x23fb jne 0x4470 (offset: -10) 2 // If the Z flag is not set, keep
decrementing
0x447a: 0x413e pop.w r14 -- mov.w @SP+, r14 2 // If they are 0, pop R14 and
R13 off of the stack
0x447c: 0x413d pop.w r13 -- mov.w @SP+, r13 2 // Increment the stack
pointer and put it into R13 and R14 presumably to restart.
0x447e: 0x3c00 jmp 0x4480 (offset: 0) 2 // Jump to 4480? Weird...
0x4480: 0x4303 nop -- mov.w #0, CG 1 // No operation, move 0 into CG?
0x4482: 0x3fec jmp 0x445c (offset: -40) 2 // Jump to 445c.
0x4484: 0x4303 nop -- mov.w #0, CG 1 // Move 0 into CG?
0x4486: 0x4031 mov.w #0x4400, SP 2 // Go back to the start of the program.
0x4488: 0x4400
0x448a: 0x12b0 call #0x44a0 5 // Weird... Just go to the next line.
0x448c: 0x44a0
0x448e: 0x430c mov.w #0, r12 1 // Move 0 into R12
0x4490: 0x12b0 call #0x4400 5 // Restart program
0x4492: 0x4400
0x4494: 0x431c mov.w #1, r12 1 // Move 1 into R12
0x4496: 0x12b0 call #0x449a 5 // Goes to the next line...
0x4498: 0x449a
```

```
// Maybe what's going on here is the program is just infinite looping as an
interrupt. It's a little obscure.
0x449a: 0x4303 nop -- mov.w #0, CG 1 // No operation
0x449c: 0x3fff jmp 0x449c (offset: -2) 2 // jump to next line
0x449e: 0x4303 nop -- mov.w #0, CG 1 // no operation
0x44a0: 0x431c mov.w #1, r12 1 // Move 1 into r12, Already done though???
0x44a2: 0x4130 ret -- mov.w @SP+, PC 3 // Return from interrupt
0x44a4: 0xd032 bis.w #0x0010, SR 2 // Set 10 as the status register
0x44a6: 0x0010
0x44a8: 0x3ffd jmp 0x44a4 (offset: -6) 2 // Jump to 44a4
0x44aa: 0x4303 nop -- mov.w #0, CG 1 // No operation wait for end of
program
```

e. See Flow Chart section.

Conclusion

This lab was a lot of fun, and I wish more of our labs were like this.

Demo link