

Debugging Theory

Offensive and Defensive Tool Construction

Table of Contents

[Objectives 2](#_Toc476902716)

[Background Reading 2](#_Toc476902717)

[Important Information 2](#_Toc476902718)

[Introduction 3](#_Toc476902719)

[Problem 1 3](#_Toc476902720)

[Problem 2 3](#_Toc476902721)

[Problem 3 4](#_Toc476902722)

Offensive and Defensive Tool Construction

Debugging Theory

Objectives

This lab focuses on the following objectives:

* Explain general-purpose CPU registers.
* Explain a “stack.”
* Explain debug events.
* Explain soft breakpoints.
* Explain hardware breakpoints.
* Explain memory access faults.

Background Reading

Read chapters 1 and 2 in the *Gray Hat Python* textbook. The following links are also useful:

* <https://docs.python.org/3/library/pdb.html>
* <http://www.gnu.org/software/gdb/documentation/>
* <http://sourceware.org/gdb/current/onlinedocs/gdb.pdf.gz>
* <https://docs.python.org/2/extending/extending.html>

# Important Information

* For *every* lab and home assignment, store all your work in your personal repository in a subdirectory named **mXX**, where XX is the module number. Carefully name the program as described in each problem.
* Your programs are extracted from your repository by a Python script. If there are any errors in the program name, then your instructor will never see your program, and you will receive a mark of zero.
* Push your work to the server often, and ensure that you push the final version of a program by the deadline specified, because the script extracting them can be run at any time after the deadline.

# Introduction

In this lab, we will explore basic debugging theory.

# Problem 1

1. Download the **pp\_driver.c** program from the class Git repository.
2. Write a C program named **printproc.c** that contains a single procedure named **printproc()**.
3. Compile the program and create a shared library named **libprintproc.so**.
4. Have the procedure get the parameters as specified in the pp\_driver.c program, and then output the following:

State: S

ParentPid: 123

ParentGid: 456

StartCode: 0x0000000000400000

EndCode: 0x0000000000067a3c

StartStack: 0x00007fff287e9300

ESP: 0x00007fff287e8818

EIP: 0xffffffff811e74b9

1. Write a Python program named **m05p01.py** to explore the Python to C calling convention, using the Python ctypes module.
2. Have the program get the proper parameters about itself (its own process), load the library **libprintproc.so** you just created, and then use the **printproc()** procedure to output the parameters.

# Problem 2

1. Write a short program in C named **m05p02.c** that contains three functions with the following prototypes:

* int main(int argc, char \*\*argv[]);
* int funtop(int a, int b, char \*str);
* int funbot(int a, int b, char \*str);

1. Ensure that main() will call funtop(). The funtop() function will multiply its integer parameters by some constant you choose, and call funbot() will simply return the result of funbot(). Ensure that funbot() prints the str, adds the two integer parameters together and returns the result.
2. Ensure that each function prints its location in memory. The innermost function prints the contents of the stack (everything from the current stack position to the top of the stack).

**Hint:** To find the top of the stack, read file */proc/[pid]/maps* and look for *[stack]*.

# Problem 3

1. Using the program you created in Problem 2, place a breakpoint just past where you read the current stack pointer and display the value of registers there.
2. Clear the breakpoint.
3. Disassemble the compiled code using the objdump -d command or the GDB disassemble command.
4. Locate the instructions which pass the parameters to the funbot() function and append them to a file named **m05report.txt**.

**Note:** The rest of the problems for this module are available in the homework assignment. See your course schedule for details.