

Building a User Mode Debugger for Windows

Offensive and Defensive Tool Construction

Table of Contents

[Objectives 2](#_Toc476921846)

[Background Reading 2](#_Toc476921847)

[Important Information 2](#_Toc476921848)

[Introduction 3](#_Toc476921849)

[Problem 1 3](#_Toc476921850)

[Problem 2 3](#_Toc476921851)

[Problem 3 3](#_Toc476921852)

[Problem 4 4](#_Toc476921853)

Offensive and Defensive Tool Construction

Building a User Mode Debugger for Windows

Objectives

This lab focuses on the following objectives:

* Write debugger code.
* Attach debugged process to debugger.
* Get CPU register state.
* Handle debug events.
* Use breakpoints.

Background Reading

Read chapter 3 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 create several debugging scenarios. First, we will write a debugger which will start the process to be debugged, and then we will explore ways to attach a debugger to a process that is already running. We will get the register state, handle debug events, set breakpoints and inspect the register state at breakpoints.

**Note:** We will build a simple debugger in this class. Each successive problem builds on the code from the previous problem and adds functionality.

# Problem 1

1. Create a file named **my\_debugger\_defines.py**.
2. Create a simple class named **m06runcalc.py** that loads an executable and then exits.
3. Write a simple test harness named **m06runcalc\_test.py** that tests the functionality of the debugger class.

# Problem 2

Create files named **m06attach.py** and **m06attach\_test.py**. Ensure that the m06attach.py file includes steps to:

* Attach to a process specified by PID
* Wait for a debug event and wait for a keystroke
* Detach and exit

# Problem 3

Create a file named **m06cpureg.py** and the associated test harness named **m06cpureg\_test.py** using the files you used in the previous problem. Ensure that your program prompts for the process PID to attach to. Once the PID is entered from the keyboard, ensure that the program dumps registers for each thread of the process.

The output should look similar to this example:

Process PID: 1234

Dumping regs for thread ID: 0x00000c7c

EIP: 0x772670b4

ESP: 0x001aef20

EBP: 0x001aef3c

EAX: 0x00000001

EBX: 0x00000000

ECX: 0x00375ab0

EDX: 0x00000030

# Problem 4

Create a file named **m06events.py** and the associated test harness named **m06events\_test.py**, again from the files used in the previous problem. Ensure that the program prompts for the PID and then shows all debugging events as they arrive. The program should gracefully detach from the process and exit when the letter **Q** (case insensitive) is pressed on the keyboard. Use the **msvcrt** module to get the keystrokes from the keyboard.

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