

Building a User Mode Debugger for Windows

Offensive and Defensive Tool Construction

Table of Contents

[Objectives 4](#_Toc66396097)

[Background Reading 4](#_Toc66396098)

[Important Information 4](#_Toc66396099)

[Introduction 5](#_Toc66396100)

[Problem 1 5](#_Toc66396101)

[Questions 6](#_Toc66396102)

[Problem 2 7](#_Toc66396103)

[Questions 7](#_Toc66396104)

[Problem 3 8](#_Toc66396105)

[Questions 8](#_Toc66396106)

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.

Background Reading

Chapters **1-3** in the *Gray Hat Python* textbook will be the material used for this lab. 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>
* https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/

# Important Information

**YOU MUST PRESENT IMAGES OF YOUR CODE BEING EXECUTED. DO NOT SUBMIT YOUR ANSWERS IN THE DOCUMENT. CREATE A BLANK DOCUMENT AND SUBMIT YOUR ANSWERS THERE.**

**YOU WILL LOSE MARKS FOR NOT FOLLOWING THE ABOVE REQUIREMENTS.**

* This Gray Hat Python book originally targeted Windows 7-32bits. You can use a copy of Windows 10 x64 with some adjustments to the Windows environment.
* Install python3+ on Windows
* Much of the lab centers around Microsoft Programming API (Application Programming Interface) which allows a developer access into much of the internals of the Operating System. To get information about these APIs you can use the authoritative site: [***https://docs.microsoft.com***](https://docs.microsoft.com) to explore the many features. Alternatively, ***google*** also provides a shotgun approach to finding a large assortment (good and bad) of resources.

# Introduction

In this lab we will create several debugging scenarios. The objective is to use python to access a process and examine the running process in memory. The, blue team, implications of being able to do this includes a deeper inspection of process memory to potentially extract vital process behavioral data. The, black hat, implications include injecting malicious to hijack process and circumvent security tools used to monitor behavior.

We will write some debugger code with the intention of exploring the following:

1. Confirm the ability to start the process to be debugged.
2. Explore ways to attach a debugger to a running process.
3. Once attached get the register state, handle debug events, set breakpoints and inspect the register state at breakpoints.

**Note:** A simple debugger will be built and each successive problem will build on the code from the previous problem adding new functionality at teach stage.

# Problem 1 - (17pts)

1. Create a working folder on your Windows system. This is where all files for this Lab will be located
   1. **C:\users\<username>\Documents\ITSC203\Lab6**
2. Get **my\_debugger\_defines.py,** **my\_debugger.py** and other code from the official book website. Download the **ghpython\_src.zip** file: ***https://nostarch.com/ghpython.htm***
   1. The original code was built to work with Python2.7 so some modification may be required to work with Python3+
   2. Python2.7 print statement => **print “Hello %s %s” % (“Hello”, “world”)**
   3. Python3+ print function => **print(”Hello %s %s” % (“Hello”, “world”))**
   4. To modify you can use Notepad++, it has regular expression features built-in. Alternatively, you could try the re.sub function or manually change the file.
3. Confirm that the **debugger class** has the **load** and **open\_process** functions. These functions provide access to processes on your windows system. The object also contains the **\_\_init\_\_** method that initializes the debugger object when created.
4. Write a simple test harness named **load\_calc.py** that tests the functionality of the debugger class. The example below shows an example of loading notepad.

import my\_debugger

from my\_debugger\_defines import \*

debugger = my\_debugger.debugger()

exePath2Load = "c:\\windows\\system32\\calc.exe".encode(“utf-8”)

debugger.**load**(exePath2Load)

## Question 1 (12pts)

1. What do the following lines accomplish? (**2pts)**
   1. Kernel32 = WinDLL('kernel32')
   2. Kernel32.GetSystemInfo(byref(system\_info))
2. What is **SYSTEM\_INFO()** where is it defined? (**2pts)**
3. What is the difference between **pointers** and **handles**? (**2pts)**
4. **CreateProcessA** and **OpenProcess** are functions inside the my\_debugger.py files (**2pts)**
   1. What is the purpose of the CreateProcessA function?
   2. What is the purpose of the OpenProcess function?
5. (**2pts)** Why doesn’t the process window show up on the screen? What would you need to modify to make the process show up on the screen.
6. (**2pts)** When you get the calculator application to open up; can you find the PID of that process?

# Problem 2 - (14pts)

In this problem you will create python files that will allow your program to attach to an already running process. To accomplish this task do the following:

1. Create files named **m5attach\_test.py**.
2. You will use the **my\_debugger.py** and **my\_debugger\_defines.py** files.
3. **my\_debugger.py** needs to have the ability to attach to a process which your **m5attach\_test.py** program will ask for:
   1. ***Please enter the PID of the process to attach to:***
      * You can find the PID of the process from **Windows Task Manager**
   2. Your debugger (in my\_debugger.py) will have an attach method. Although much of the code has already been provided, there are some minor modifications that need to be made.
4. Attach to a process specified by the PID
5. The program will continue indefinitely until a debug event **and** a keystroke has occurred. You will perform the keystroke inside the command window.
6. After a debug event and a keystroke, detach from the process and exit.

**HINT: How does the debugger remain active and can the state be changed?**

**Python has a module to help determine if a keypress has occurred.**

## Question 2 - (9 pts)

1. What type of EVENT is triggered after attaching to the process? (**3pts)**
2. Can you access the interface of the process being debugged? (**3pts)**
3. What constitutes a debug event? In other words how do you know when a debug event has occurred? (**3pts)**

# Problem 3 - (17pts)

Create a file named **m5cpureg.py** and the associated test harness named **m5cpureg\_test.py** using the files you used in the previous problem.

1. **Your program should, prompt for the PID to attach to.**
2. **Once the PID is entered, 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

## Question3 (12pts)

1. What is a Thread? **(3pts)**
2. What is a Thread Context? **(3pts)**
3. Will the EIP of all Threads be the same? Why? **(3pts)**
   1. Show screen shots of at least 2 threads if present.
4. Will the ESP be the same for all the Threads? Why? **(3pts)**
   1. Show screen shots of at least 2 threads if present.