

Applying a Debugger for Windows

Offensive and Defensive Tool Construction

Table of Contents

[Objectives 4](#_Toc477255276)

[Background Reading 4](#_Toc477255277)

[Important Information 4](#_Toc477255278)

[Introduction 5](#_Toc477255279)

[Problem 1 5](#_Toc477255280)

[Problem 2 5](#_Toc477255281)

[Problem 3 5](#_Toc477255282)

[Problem 4 6](#_Toc477255283)

[Problem 5 6](#_Toc477255284)

[Problem 6 6](#_Toc477255285)

[Problem 7 6](#_Toc477255286)

[Problem 8 6](#_Toc477255287)

[Problem 9 7](#_Toc477255288)

Offensive and Defensive Tool Construction

Applying a Debugger for Windows

Objectives

This lab focuses on the following objectives:

* Outline the basics of the Immunity Debugger.
* Find exploit-friendly instructions.
* Describe bad character filtering.
* Defeat anti-debugging code in malware.

Background Reading

Read chapter 5 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>
* <https://support.microsoft.com/en-us/kb/875352>
* <https://www.sans.org/reading-room/whitepapers/malicious/basic-reverse-engineering-immunity-debugger-36982>
* <https://sgros-students.blogspot.ca/2014/05/immunity-debugger-basics-part-1.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 the Immunity Debugger. Note that you need to have the C development environment installed on your computer for this lab. For example, mingw is open source and you can download it from <http://www.mingw.org/>. After you install it, add **c:\mingw\bin** to your system path so you can use the gcc compiler without typing in the full path.

# Problem 1

1. Download Immunity Debugger from <http://debugger.immunityinc.com>.
2. Install it in a default location.
3. Locate the PyCommands in **/c/Program Files/Immunity Inc/Immunity Debugger/PyCommands**.

# Problem 2

Write a PyCommand script named **m07p02.py** that outputs a single line consisting of your SAIT email address in this format:

fred.flintstone@edu.sait.ca

# Problem 3

1. Write a C program named **callprintf.c**:

#include <stdio.h>

char label[]="The address of main is ";

void

print\_all (char \*label,void \*addr)

{

printf ("%s: 0x%08x\n", label,addr);

}

int

main (int argc, char \*\*argv)

{

void \*addr\_of\_main;

addr\_of\_main = (void \*) &main;

print\_all (label,addr\_of\_main);

printf ("Done.");

}

1. Compile it:

gcc -g -O0 -o callprintf callprintf.c

1. Debug the program:
2. Locate the call to the **print\_all()** function. Observe how the parameters are passed on the stack.
3. Set up a breakpoint on the entry to the print\_all() function, and then single-step through the function up to the prinf() function call.
4. Record the stack frame after the print\_all() function has been entered.
5. Record the value of the two parameters that passed.

# Problem 4

Write a simple PyCommand file named **m07p02.py** that searches for the instruction “jmp esp” and prints all addresses where the instruction is found and can be executed.

# Problem 5

Create shellcode that executes the Windows calculator, **calc.exe**. Use the **WinExec()** API call and hard-code its address.

**Hint:** You can write the code in C, compile to the assembler, and then adapt it.

# Problem 6

Modify Problem 5 such that instead of hard-coding the WinExec() call, you locate its address at runtime.

# Problem 7

1. Write a C program named **overflow.c** that generates a buffer overflow.
2. Find a suitable address for a “jmp esp” instruction and insert it before the shellcode.
3. Apply the shellcode and copy it into the buffer.
4. Execute the program and verify that the calc.exe is spawned.

# Problem 8

Review the code in the **badchar.py** program in chapter 5 and fix it so it performs as intended.

# Problem 9

1. Bypass DEP on Windows.

**Hint:** Reuse the code in **overflow.c** and modify it for this purpose.

1. Use the immunity debugger script **findantidep.py**.
2. Verify that the DEP has been bypassed.