Applying Debugger for Windows

Lab Manual

Objectives

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

Table of Contents

1.

Background Reading

Textbook:

Grey Hat Python, chapter 5

Other resources:
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

Notes common to all lab and home assignment problems

For every lab and home assignment, all work should go into your personal repository, subdirectory named mXX, where XX stands for the module number. For each problem, carefully name the program as described. The programs are extracted from your repository by a Python script, and errors in the program name will result in the instructor never seeing your program, and your mark for it will be ZERO!

Anything to record and report in this lab is to be written in plain text file (Word document is not a plain text file). The file MUST be named mXXlabrep.txt. Mis-naming this file, or not having it in the proper location will result in mark ZERO for anything to be recorded or reported.

There are always many ways how to solve a programming problem, and usually one or two ways which are fast, compact and elegant.

Make sure to push your work to the server often, and have pushed the working version of the program by the deadline specified. The script extracting your programs from your repository will be run at any time after the deadline.

Lab specific intro

In this lab we will explore the Immunity Debugger. You need to have C development environment installed for this lab, for example “mingw”. It is open source, you can download it and install it from http://www.mingw.org/. After installation, you can add **`c:\mingw\bin`** to your system path so you can use the gcc compiler without typing in the full path.

Problem 1

Install Immunity debugger:

download from http://debugger.immunityinc.com,

install in default location.

The PyCommands are found in:

"/c/Program Files/Immunity Inc/Immunity Debugger/PyCommands"

Problem 2

Write PyCommand script, name it m07p02.py, which will output a single line consisting of your sait email address, like this:

fred.flintstone@edu.sait.ca

Problem 3

Write C program 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.");
}
```

Compile it:

gcc -g -O0 -o callprintf callprintf.c

Debug the program:

locate the call to print_all() function. Observe how the parameters are passed on the stack. Set up a breakpoint on entry to the print_all() function, then single-step through the function up to the prinf() function call. Record the stack frame after the print_all() function has been entered. Record the value of

the two parameters passed.

Problem 4

Write simple PyCommand file, **`m07p02.py`** which will search for the instruction "`jmp esp`" and print all adresses where the instruction is found and can be executed.

Problem 5

Create shellcode which will execute windows calculator, calc.exe. Use the WinExec() API call, hard-code its address. Hint: You can write the code in C, then compile to assembler, then adapt.

Problem 6

Modify the task of problem 5: instead of hard-coding the WinExec() call, locate its address at runtime.

Problem 7

Write C program which will generate buffer overflow, name it **`overflow.c`** Find suitable address of a "jmp esp" instruction and insert it before the shellcode. Apply the shellcode and copy it into the buffer. Execute the program and verify that the calc.exe is spawned.

Problem 8

Review the code in the Chapter 5, badchar.py, and fix it so it performs as intended.

Problem 9

Bypass DEP on windows. Reuse the code in overflow.c and modify it for this purpose. Utilize the immunity debugger script **`findantidep.py`**. As a last step, verify that the DEP has been bypassed.

Applying Debugger for Windows

Home Assignemnt Manual

Objectives

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

Table of Contents

1.

Background Reading

Textbook:

Grey Hat Python, chapter 5

Other resources:
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

Notes common to all lab and home assignment problems

For every lab and home assignment, all work should go into your personal repository, subdirectory named mXX, where XX stands for the module number. For each problem, carefully name the program as described. The programs are extracted from your repository by a Python script, and errors in the program name will result in the instructor never seeing your program, and your mark for it will be ZERO!

Anything to record and report in this lab is to be written in plain text file (Word document is not a plain text file). The file MUST be named mXXlabrep.txt. Mis-naming this file, or not having it in the proper location will result in mark ZERO for anything to be recorded or reported.

There are always many ways how to solve a programming problem, and usually one or two ways which are fast, compact and elegant.

Make sure to push your work to the server often, and have pushed the working version of the program by the deadline specified. The script extracting your programs from your repository will be run at any time after the deadline.

Problem 5

Modify the task of problem 5: instead of hard-coding the WinExec() call, locate its address at runtime.

Problem 6

Write C program which will generate buffer overflow, name it **`overflow.c`** To accomplish this, declare a local variable in your main() of size at least 512 bytes. Have your main() call a function fun(), declare local stack variable in that function. Utilize shellcode which will execute calc.exe.

Find suitable address of a “jmp esp” instruction and insert it before the shellcode. Apply the shellcode and copy it into the buffer. Execute the program and verify that the calc.exe is spawned.

Problem 7

Review the code in the Chapter 5, badchar.py, and fix it so it performs as intended.

Problem 8

Bypass DEP on windows. Reuse the code in overflow.c and modify it for this purpose. Utilize the immunity debugger script **`findantidep.py`**. As a last step, verify that the DEP has been bypassed.