

Applying Fuzzing Techniques

Offensive and Defensive Tool Construction

Table of Contents

[Objectives 4](#_Toc477850579)

[Background Reading 4](#_Toc477850580)

[Important Information 4](#_Toc477850581)

[Introduction 5](#_Toc477850582)

[Problem 1 5](#_Toc477850583)

[Problem 2 5](#_Toc477850584)

[Problem 3 5](#_Toc477850585)

[Problem 4 5](#_Toc477850586)

Offensive and Defensive Tool Construction

Applying Fuzzing Techniques

Objectives

This lab focuses on the following objectives:

* Explain the concept of “fuzzing.”
* Distinguish bug classes.
* Write code to implement buffer overflows.
* Use integer overflows.
* Use format string attacks.
* Create several types of fuzzers.

Background Reading

Read chapter 8 in the *Gray Hat Python* textbook. The following links are also useful:

* <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 concept of fuzzing. We will generate malformed data, send it to the application under attack, and then observe the application’s behaviour. The application under attack should crash repeatedly, and we will learn what data patterns make it crash.

We will also explore several bug classes: buffer overflows, integer overflows and format string attacks. We will create a program named **file\_fuzzer.py** to run as a simple fuzzer.

# Problem 1

In Module 7, we implemented a buffer overflow on the stack using C, and then exploited that code to start the **calc.exe** executable file.

1. Write a code in C that generates heap overflow and name it **heapover.c**.
2. Run the code under debugger control and observe where it crashes. Observe what caused the crash, and report annotated screenshots of the debugger.
3. Save your screenshots in files named **heapoverXXX.png**, where XXX is the sequence number of the screenshot.

# Problem 2

1. Read section 8.1.2 in the *Grey Hat Python* textbook.
2. Describe the sequence of events that lead to integer overflow and the compromising of the heap.

# Problem 3

1. Find calls to malloc().
2. Inspect them and then select the one that allocates the most memory.
3. Report the location of the call and the amount of memory requested.

# Problem 4

1. Explore format string attacks.
2. Write a source file in C named **m10p04.c** that misuses the format specifier string and dumps the content of its stack instead.

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