CS 6035 – Introduction to Information Security
Project 1 – Buffer Overflow

Contents

Goal

Students should be able to clearly explain:

1. What a buffer overflow is - understand the concepts of buffer overflow
2. Why a buffer overflow is dangerous
3. How to exploit a buffer overflow. Students are expected to launch an attack that exploits stack buffer overflow vulnerability in the provided toy program.

Task 1 - Understanding Buffer Overflow (40 points)

Stack Buffer Overflow

Write a C/C++ program that contains the stack buffer overflow vulnerability. Show what the stack layout looks like and explain how to exploit it. You are not required to write the real exploit code, but you may *want to use some figures to make your description* clear.

Hints

1. Learn how to write a C/C++ program if you do not know how to do that
2. "Smashing The Stack For Fun And Profit"(http://phrack.org/issues/49/14.html)

Deliverable

An Adobe PDF file containing your vulnerable program (paste your code in the PDF directly) and your explanation. For an example of how your stack layout should look, see chapter 10 in the text Stallings, William, and Lawrie Brown. *Computer Security Principles and Practice.* Boston: Pearson, 2015 text. Also a good example of a stack diagram can be found at: https://en.wikipedia.org/wiki/Stack_buffer_overflow. Make sure to properly cite any references that you use.

Grading rubric for Task 1

1. Vulnerable Program (5 points)
2. Stack Layout (25 points)
   - Note: the stack layout should contain the following contents:
     - Stack layout (5 points)
     - High and low address of stack, stack grows direction (5 points)
     - Size of each part on the stack (5 points)
     - Content of each part of the stack (5 points)
     - Overflowed area of the stack (5 points)
3. Exploiting Explanation (10 points)

Task 2 - Exploiting Buffer Overflow (60 points)

The C code shown in Appendix 1 contains possible stack buffer overflow vulnerability. Your task is to write an exploit (e.g., Python script) to open a shell on Linux. The high level idea is to overwrite the return address with the **libc** address of the function ***system()***, and pass the parameter ***"sh"*** to this function. Once the return instruction is executed, this function will be called to open a shell.

Lab Environment

Students are required to download the VirtualBox appliance (Project1.ova) from the assigned website and following instructions to perform the project.

1. Install VirtualBox on your computer. You can either download it from official website or from Google Drive, if you do not have the software installed already. The appliance will have the guest additions CD already installed *(**version 5.0.14 r105127**)*. If you download a newer VirtualBox or have already have different version installed, you will need to install your own guest additions CD by going to the VM Menu and selecting **Devices → Insert Guest Additions CD Image**. Then follow the screen prompts.
2. Download the VirtualBox appliance *(**Project1.ova**)* from Google Drive. The lab environment has already been created; therefore you can start the lab on it directly.
    a. Download link: https://drive.google.com/open?id=0B_260Z7EWPixMTNmSVRUNjUtUzg
    b. Login Instructions - Username: Ubuntu / Password: 123456
    c. Instructions on importing the Virtual Box Appliance: https://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html
3. Change to the directory of the Project 1 Package. The package has been placed on the Desktop of the VM.
4. Compile the Toy program as follows: ***gcc sort.c -o sort -fno-stack-protector***.
5. To run the program, add hexadecimal integers for sorting in the file ***data.txt*** and execute your sort function by: ***./sort data.txt***
6. You will notice that when you put a very long list of integers in ***data.txt***, that sort crashes with a segment fault error; this is because the return address has been overwritten by your data.
7. Your job is to craft your own shellcode in ***data.txt***. You are to overwrite the return address with the address of function ***system()*** and pass it with the address of string "***sh***".
8. **Warning:** Before you exploit the program, please first understand the logic of ***sort.c***. Also, DO NOT install any updates to the VM. Good Luck!

Hints

1. Why traditional buffer overflow exploits do not work? See DEP and W^X protections
2. How to bypass DEP and W^X? See "The advanced return-into-lib(c) exploits" (http://phrack.org/issues/58/4.html) and "Return-to-libc" (https://www.exploit-db.com/docs/28553.pdf)
3. GDB is a helpful tool to understand the stack layout when buffer overflow happens and get the addresses of system() and "sh"

Deliverable

Deliver either an exploit script (any scripting language is allowed, but your script must contain everything, so that the TA can directly run it by ***./script*** or the ***data.txt*** file you craft. The exploiting results should be demonstrated with your *screenshots* (paste the screenshots in the same PDF file as your Task 1 report).

- **Note:** Ensure that **data.txt** does not have ***CRLF***. To achieve this, if you use winSCP or similar tools to transfer files, make sure the file is transferred as ***binary***. *Do not create the text file with notepad or similar tools in a Windows system (Apple OSX should not have this problem).*

Grading rubric for Task 2

1. Buffer overflow caused by your crafted ***data.txt*** and overflow proof in GDB (10 points)
2. Locate the Libc ***system()*** address in GDB (10 points)
3. Locate ***/bin/sh*** address in GDB (10 points)
4. Correct exploit payload in ***data.txt*** and being able to open the shell in terminal (30 points)

Appendix A – The Toy Program for Task 2

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
 * a toy program for learning stack buffer
 * overflow exploiting
 * It reads a list of hex data from the
 * specified file, and performs bubble sorting
 */

long n = 0, c = 0, d = 0;

FILE *fp = NULL;

void bubble_sort()
{
  long swap = 0;
  long array[12];

  // loading data to array
  printf("Source list:\n");
  char line[sizeof(long) * 2 + 1] = {0};
  while(fgets(line, sizeof(line), fp)) {
    if (strlen((char *)line) > 1) {
      sscanf(line, "%lx", &(array[n]));
      printf("0x%lx\n", array[n]);
      ++n;
    }
  }
  fclose(fp);

  // do bubble sorting
  for (c = 0 ; c < ( n - 1 ); c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
```

```
      if (array[d] > array[d+1])
      {
        swap      = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }

  // output sorting result
  printf("\nSorted list in ascending order:\n");
  for ( c = 0 ; c < n ; c++ )
     printf("%lx\n", array[c]);

}

int main(int argc, char **argv)
{
   if(argc!=2)
   {
      printf("Usage: ./sort file_name\n");
      return -1;
   }

   fp = fopen(argv[1], "rb");
   bubble_sort();

   return 0;
}
```

Appendix B – Sample Data for Task 2

```
1
3
5
7
80
a
d0
```