Introduction Information Security
Project #1 Buffer Overflow

Index

Goals:

- Understanding the concepts of buffer overflow
- Exploiting a stack buffer overflow vulnerability

Students should be able to clearly explain: 1) what a buffer overflow is; 2) why a buffer overflow is dangerous; and 3) how to exploit a buffer overflow. Students are expected to launch an attack that exploits a stack buffer overflow vulnerability in the provided toy program.

Task 1. Understanding Buffer Overflow (40 points)

1) Stack buffer overflow

Write a C/C++ program that contains a 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: a PDF file containing your vulnerable program (paste your code in the PDF directly) and your explanation.

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 following C code contains a stack buffer overflow vulnerability. Please 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 address of 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 (Ubuntu.ova) from the assigned website and following instructions to perform the project

1. Install VirtualBox on your computer.
Note: You can either download it from official website or from Google Drive, if you do not have the software installed already.
2. Download the VirtualBox appliance (Ubuntu.ova) from Google Drive. The lab environment has been created, therefore you can start the lab on it (Username: Ubuntu / Password: 123456)
(Download Link: https://drive.google.com/folderview?id=0B6NEF8Y9uP9OallsOWVVTkNNdlE&usp=sharing)
3. Import the VirtualBox appliance
https://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html
4. Go to the directory of the project package
Note: The project package (not compressed) has been put onto the Desktop of VM, so you do not have to download it
5. Compile the provided C code: *gcc sort.c -o sort -fno-stack-protector.*
6. To run this program, put some hexadecimal integers for sorting in the file: *data.txt*, and execute *sort* by: *./sort data.txt*
7. When you put a very long list of integers in *data.txt*, you will notice *sort* crashes with a memory error segment fault; this is because the return address has been overwritten by your data.
8. Now you can craft your shellcode in *data.txt*. Again, your goal is to overwrite the return address with the address of function system() and pass it with the address of string "sh".
9.**Before you exploit the program, please first understand the logic of *sort.c***. Have fun.

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: 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 task 1).Note: Please 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. Also you shouldn't create the text file with notepad or similar tools in a Windows system (OSX should not have this problem).

Grading rubric for Task 2:

1. Buffer overflow detection data craft and overflow proof in GDB (10 points)

2. Locate 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: 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, swap = 0;
FILE *fp = NULL;

void bubble_sort()
{
  long array[10];

  // 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