Introduction Information Security
Project # Heap Overflow

Index

Goals:

- Understanding the concepts of heap overflow
- Exploiting a heap overflow vulnerability

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

Task 1. Understanding Heap Overflow (40 points)

1) Heap overflow
Write a C/C++ program that contains a heap overflow vulnerability. Explain which kind of heap overflow it is and 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

Deliverable: a PDF file containing your vulnerable program (paste your code in the PDF directly) and your explanation. **Also include you gtid, for example, jdoe123, in the pdf file.**

Grading rubric for Task 1:

1. Vulnerable Program (20 points)

2. Please explain which specific heap buffer overflow their code contains (10 points)
Note: unlike stack buffer overflow, heap overflow happens in different manner (e.g., heap meta data overwriting, user-after-free, or simple out-of-bound write).

3. Explain how to exploit it (10 points)

Task 2. Exploiting Heap Overflow (60 points):

The following C code contains a heap overflow vulnerability. Please write an exploit

(e.g., Python script) to open a shell on Linux. The high level idea is to overwrite the function pointer in heap with the address of a spatial code piece (contained in the *sort* program). Once the code piece is executed, stack pointer is changed to point to your payload that contains the addresses of system() and "sh". Finally, the *ret* instruction in the code piece will call *system()* and open the shell.
Note: To reduce the difficulty, we have directly inserted this special code piece in *sort.c* (see the *sub* and *ret* instructions in assemly() function). In ROP attack, such code piece is called stack pivot ROP gadget, which can adjust stack pointer (esp) to make it point to your payload.

Note: the address of return address may be different within GDB and outside of it

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. The VirtualBox-5.0.4 for Windows on Windows 7 is recommended.
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 folder has been put onto the Desktop of VM. you should conduct your project in the folder (do not change your project files location)
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 function pointer in heap with the address of a spatial code piece (contained in the *sort* program). Once the code piece is executed, stack pointer is changed to point to your

payload that contains the addresses of system() and "sh". Finally, the *ret* instruction in the code piece will call *system()* and open the shell.

9.**Before you exploit the program, please first understand the logic of *sort.c***. Have fun.

Hints:

[1] Why traditional 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] What is Return Oriented Programming (ROP) Attacks (https://www.exploit-db.com/docs/28479.pdf)

[4] GDB is a helpful tool to understand the stack layout when overflow happens and get the addresses of system() and "sh"

Deliverable: the ***data.txt*** file you craft. The exploiting process should be demonstrated with your screenshots (paste the screenshots under each item of Task 2). 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.

Grading rubric for Task 2:

1. Heap overflow detection data craft and overflow proof in GDB (10 points)
   Note: Please prove that a crafted data can overflow the heap of toy program

2. Locate the address of the special code piece (ROP gadget) (10 points)

3. Locate system () and "/bin/sh" in GDB(10 points)

4. Correct exploit payload in data.txt and being able to open the shell in the terminal(30 points)

Appendix A: Toy Program for Task 2

```c
#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 locates maxmun number
 */

long n = 0, c = 0, d = 0, max = 0;
FILE *fp = NULL;

typedef struct chunk {
  long array[10];
  void (*process)(long *);
} chunk_t;

void maxSort(long array[10])
{
  // do bubble sorting
  int i = 0;
  max = array[0];
  for (i = 1 ; i < ( n - 1 ); i++)

  {
    if (array[i] > max)
    {
        max = array[i];
    }
  }
}

void assembly()
{
  long vesp = 0xbffff0ac;  //esp address
  long vchunk = 0x804b170; // address of system() and "sh"
  long offset = 0xB7FB3F3C; //vesp - vchunk;
  asm volatile (
                "sub $0xB7FB3F5C, %esp \n\t"
                "ret \n\t"
        );
}

void max_sort()
{
  //long array[10];
  chunk_t *next;
  next = malloc(sizeof(chunk_t));
  next->process = maxSort;

  // 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", &(next->array[n]));
      printf("0x%lx\n", next->array[n]);
      ++n;
    }
  }


  next->process(next->array);
  fclose(fp);
  // output sorting result
  printf("\nMaxmum number in the list is:\n");

  printf("%lx\n", max);

}

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

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

    return 0;
}
```

Appendix B: Sample Data for Task 2

1
3
5
7
80
a
d0