Lab2: Characterize a memory corruption vulnerability

Given information is

1. We are provided with an Application based upon a data model stored in ABaseClass
2. It was written for a 16-bit 80286-based workstation
3. After update, the new code was stored in ANewClass
4. The program intended to run the service is expl1.c

Testing:

First let us try running the program and see the result.

From the terminal in a Linux machine type the command “make” to create object files for all .cpp files.

```
root@kali:~/Downloads/lab2# make
g++ -c -o expl1.o -fno-stack-protector -g2 -O0 expl1.cpp
expl1.cpp:8:29: warning: integer constant is too large for its type
    ABaseClass a = ANewClass(0x40404040404040404040404040404040);
                             ^
g++ -c -o ABaseClass.o -fno-stack-protector -g2 -O0 ABaseClass.cpp
g++ -c -o ANewClass.o -fno-stack-protector -g2 -O0 ANewClass.cpp
g++ -o expl1 -fno-stack-protector -g2 -O0 expl1.o ABaseClass.o ANewClass.o
```

This creates 3 object files and an executable as below.

```
root@kali:~/Downloads/lab2# ls
ABaseClass.cpp  ANewClass.cpp  expl1      expl1.o  Makefile
ABaseClass.o    ANewClass.o    expl1.cpp  include
```

Also we can observe that it created a warning:

integer constant is too large for its type

ABaseClass a = ANewClass(0x40404040404040404040404040404040);

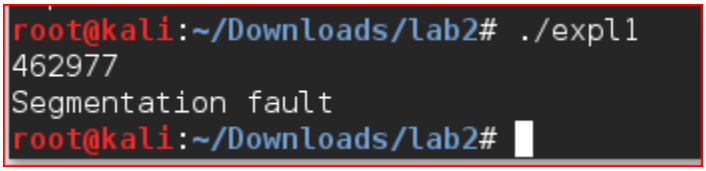
We can find the above line of code in expl1.cpp file. In this line a new variable “a” is being declared as type ‘ABaseCLass’ while passing an object of length 14 bytes into it. From file ANewClass.cpp, we can see that the variable declared is of type “unsigned long long” whose size is 8 bytes.

In C++ size of different datatypes is listed in the below table:

Type	Size
bool, char, unsigned char, signed char, __int8	1 byte

__int16, short, unsigned short, wchar_t, __wchar_t	2 bytes
float, __int32, int, unsigned int, long, unsigned long	4 bytes
double, __int64, long double, long long	8 bytes

By running the executable file expl1, we can see that there is Segmentation fault.



What is a segmentation fault?

When your program runs, it has access to certain portions of memory. First, you have local variables in each of your functions; these are stored in the stack. Second, you may have some memory, allocated during runtime (using either malloc, in C, or new, in C++), stored on the heap (you may also hear it called the "free store"). Your program is only allowed to touch memory that belongs to it -- the memory previously mentioned. Any access outside that area will cause a segmentation fault.

There are four common mistakes that lead to segmentation faults: dereferencing NULL, dereferencing an uninitialized pointer, dereferencing a pointer that has been freed (or deleted, in C++) or that has gone out of scope (in the case of arrays declared in functions), and writing off the end of an array.

A fifth way of causing a segfault is a recursive function that uses all of the stack space. On some systems, this will cause a "stack overflow" report, and on others, it will merely appear as another type of segmentation-fault.

The strategy for debugging all of these problems is the same: load the core file into GDB, do a backtrace, move into the scope of your code, and list the lines of code that caused the segmentation fault.

Source: http://www.cprogramming.com/debugging/segfaults.html

From the code snippet exp1.cpp:

```
#include <cstdio>
#include <cstring>

#include "include/ANewClass.h"

int
main(int argc, char **argv) {
    ABaseClass a = ANewClass(0x4040404040404040404040404040);

    a.show_data();
}
```

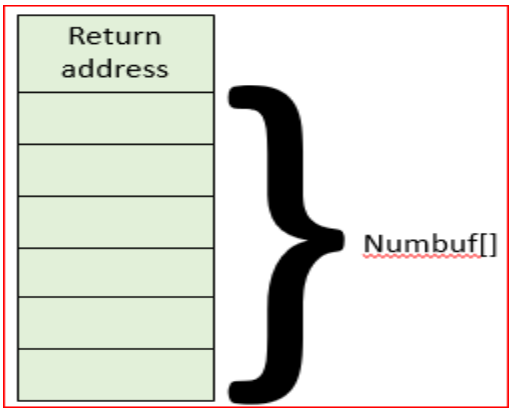
From the above object declaration, object “a” is being declared as:

ABaseClass a = ANewClass(0x4040404040404040404040404040);

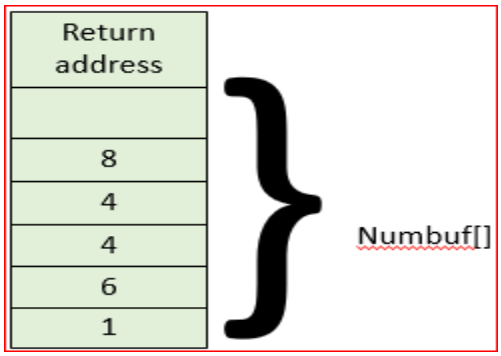
Also, from ABaseClass.h, functions show_data() and *get_buf() are virtual functions as below.

```
#ifndef _C6056_ABASECLASS_H_
#define _C6056_ABASECLASS_H_

class ABaseClass {
 protected:
  char numbuf[6];

 public:
  ABaseClass(unsigned short);
  virtual void show_data(void);
  virtual char *get_buf(void);
};

#endif /* _C6056_ABASECLASS_H_ */
```

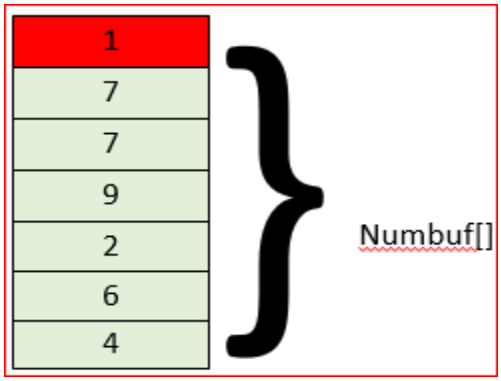
From my analysis, I found that while writing the decimal conversion of input to the buffer numbuf[], it is overwriting the return address which causes segmentation fault. It is well picturized below.



Numbuf before any constructor has run



Numbuf after ABaseClass constructor has run



Numbuf after ANewClass constructor has run

It can be seen that the return address has been overwritten by the use of sprint in ANewClass constructor. As there is no return address, Segmentation fault occurs.

Proposed solution: The main problem here is that the size of the numbuf is not sufficient to store the result. So, I created another new buffer in ANewClass.cpp with a size of 20. Also I used function over-riding to prevent the execution of normal show_data() and *get_buf() functions from ABaseClass.cpp. To make ANewClass.cpp functions overwrite ABaseClass.cpp functions, modify object a's declaration as

ANewClass a = ANewClass(0x40404040404040404040404040404040);

Also, I used **snprintf** instead of sprint to prevent any function exhausting the size of my new buffer "nbuf". Now the new code snippets in ANewClass.cpp, ANewClass.h and expl1.cpp look as below:

expl1.cpp × ABaseClass.h × ANewClass.cpp

```
#include "include/ANewClass.h"
#include <cstdio>
#include <stdint.h>

ANewClass::ANewClass(unsigned long long input) : ABaseClass((unsigned short)input) {
snprintf(this->get_buf(),sizeof(nbuf), "%llu", input);
}
char*
ANewClass::get_buf(void) {
  return &(this->nbuf[0]);
 }
void
ANewClass::show_data(void) {
  printf("%s\n", this->nbuf);
}
```

New code in ANewClass.cpp

expl1.cpp × ABaseClass.h × ANewClass.cpp × ANewClass.h ×

```
#ifndef _C6056_ANEWCLASS_H_
#define _C6056_ANEWCLASS_H_

#include "ABaseClass.h"

/*
 * Implement a version which can accept new 64-bit type values,
 * but remains backward compatible!
 */

class ANewClass : public ABaseClass {
protected:
  char nbuf[20];
  public:
    ANewClass(unsigned long long input);
    void show_data(void);
    char *get_buf(void);
};

#endif /* _C6056_ANEWCLASS_H_ */
```

New code in ANewClass.h

expl1.cpp × ABaseClass.h × ANewClass.cpp

This file "/root/Downloads/lab2/expl1.cpp" is already open in another window.

Do you want to edit it anyway?

```
#include <cstdio>
#include <cstring>

#include "include/ANewClass.h"

int
main(int argc, char **argv) {
    ABaseClass a = ANewClass(0x4040404040404040404040404040);

    a.show_data();
}
```

New code in expl1.cpp

Now, regardless of the size of the input from expl1.cpp, it is truncated to 8 byte data and gives a correct result **without** Segmentation fault.

The out will look something like this:

```
root@kali:~/Downloads/lab2_new# ./expl1
18446744073709551615
root@kali:~/Downloads/lab2_new#
```

Submitted by,
Pavan Ulichi