CS220 - Computer System II
Lab 6

Due: 10/05/2017, 11:59pm

1 Introduction

In this lab, you will play with pointers to modify contents of the stack to corrupt data and control flow.

2 Data Corruption: The Problem

In Prog.c given below, you are to implement `bad` function such that the print statement prints the name of the student as "Quick brown fox jumped over the lazy dog" and age as 1000.

```
/* Prog.c */
#include <stdio.h>
void bad()
{
  /* Your implementation goes here */
}

int main() {
  struct {
    char *name;
    int age;
  } student = {.name = "John", .age = 22 };
  bad();
  printf("student.name = %s\nstudent.age = %d\n", student.name, student.age);
  return 0;
}
```

RULES:

1. You are not allowed to change main. All your changes must be confined in the bad function.
2. `bad` function is not allowed to accept any input from user. That is, you are not allowed to manually pass the address of `student` to `bad` function.

3 The Solution

3.1 Strategy

We know that the local variables are stored on the stack. If we can locate the student structure on the stack, then, we can modify the "name" field to point to any string. Similarly, we can modify the age field to a value 1000.

3.2 Step 1: Creating the new name

First, let us create a new string in `bad` with the new value. We will also create a pointer called temp that we will use. We use type unsigned char * for temp to allow us to perform pointer arithmetic easily. We initialize it to the address of new_name.

```
void bad() {
  char *new_name = "Quick brown fox jumped over the lazy dog";
  unsigned char *temp;
  temp = (unsigned char *) (&new_name);
  ...
}
```

Note that although name is local to bad, because it is a string literal, it is created in the data section, so its scope is not limited to the scope of `bad` function.

3.3 Step 2: Locating address of `student` and `new_name`

If the student structure is in main's stack frame, the new_name variable is located in `bad` function's stack frame, which is on top of main's stack frame. So, `&student` is at positive offset from `&new_name`. The offset is positive because the stack grows from higher to lower address. That is, the top of the stack is at a lower address than bottom of the stack. Let us use gdb to find the some addresses. We are interested in address of student.name, student.age and new_name. We are also interested in finding the location where our string is stored, which is nothing but the value of new_name.

```
$ gcc -g prog.c -o prog -O0 /* -O0 will turn off optimizations */
$ gdb ./prog
(gdb) b main
```

```
(gdb) r
(gdb) p &student.name
(gdb) p &student.age
(gdb) b bad
(gdb) c
(gdb) p &new_name
(gdb) n /* This will initialize new_name with the string */
```

Record the addresses of student.name, student.age, and new_name. We also need the location where the new string is stored, but we don't need to separately record it.

3.4 Step 3: Computing the offset

Compute the offset of student.name from new_name, which is nothing but &student.name - &new_name. Record it as Offset_name. Similarly, record Offset_age as the offset from &student.age and &new_name.

3.5 Step 4: Modifying the values

The final step is to make student.name to point to the new string and change the value at student.age to 1000.

```
void bad()
{
  char *new_name = "Quick brown fox jumped over the lazy dog";
  unsigned char *temp;
  temp = (unsigned char *) (&new_name);

  /* We are telling the compiler: Treat as if temp+offset_name contains an
    address to a string and set the address of the new_string as the contents
     of the location */
  *((char **) (temp + Offset_name)) = new_name;

  /* Treat as if temp+offset_age is a pointer to an integer and store the
    value 1000 in the location pointed to by that integer. */
  *((int *) (temp + Offset_age)) = 1000;
}
```

4 Altering control-flow

Going one step further, because the return address is located on the stack, the problem to ask is: can we modify the return address to return to some other location. The answer is yes. In the above program, the return from `bad` function returns control to the instruction/statement immediately after the call to `bad` function. Now, let us change that such that the print statement in main that prints the students name and age is skipped!

4.1 Strategy

Just like how we altered the name and age fields, we first want to find the location where the return address is stored on stack. Then, given the address where the return statement in main is, we want to change the return address from bad to point to the return statement in main so that the print statement in main is skipped.

4.2 Step 1: Finding the address that `bad` returns to

Use gdb and find the address `bad` *should* return to.

```
$ gdb ./prog
(gdb) b main
(gdb) r
(gdb) disas main
```

Locate the call instruction to bad. It looks like this:

```
addr <+offset from start of function>: callq <bad>
```

Note that address of instruction immediately after call to bad. Record it as `ret_orig`.

4.3 Step 2: Find the address print statement in main returns to

Next, we identify the new value of the return address. That is, we want the address that follows after the call to printf. Using similar steps as above, locate the address in main that follows call to printf. Call it `ret_new`. Compute the distance/offset from ret_orig and ret_new. Call it ret_offset.

4.4 Step 3: Compute offset from new_name to return address on stack

Use gdb to intercept control flow in the bad function and print the contents of the stack. Locate ret_orig. Record the address where ret_orig is stored. This is the address where return address is located.

```
$ gdb ./prog
(gdb) b bad
(gdb) r
(gdb) x/16gx $rsp
```

Here, rsp is the address of the top of the stack. We are printing 16 64bit values in hex starting from the top of the stack. This prints the first 16 64bit values on the stack. Search for `ret_orig` and record the address where it is located. Stack addresses typically start with 0x7ffff. Call it `addr_ret_orig`. Next, given the address of new_name, compute the distance to `addr_ret_orig`. Call it stack_offset_ret_addr.

4.5 Step 4: Finally modify the return address to point to `ret_new`

```
void bad() {
...
temp = (unsigned char *) (&new_name);
*((unsigned long *) (temp + stack_offset_ret_addr)) += ret_offset;
}
```

4.6 Step 5: Testing

Finally, compile and run the program. If all went well, you should see main return without printing student's name and age.