Tony Maldonado
September 16, 2020
CS 471

Programming #3 – Digging into the Runtime Stack

```
/*
Purpose: This program will demonstrate how to
overwrite the return address inside a function.
We will use a global variable to store the address
we want to go on return, and we will use an array
in the function to seek the location and replace
with the new value.

Name: Tony Maldonado
Date: September, 2020

*/

#include <stdio.h>

// dummy function which makes one important change

void f() {

  unsigned int *A;
  int i;
  int a;
  int b;
  int c;
  //int d;
  //int e;
  //int f;

  a = 6;
  b = 12;
  c = 13;
  //d = 14;
  //e = 21;
  //f = 32;

  // odd nunmber of vars it breaks
  // even it doesnt

  //printf("a is %u\n",&a);
  //printf("b is %u\n",&b);
  //printf("c is %u\n",&c);
  //printf("d is %u\n",&d);
```

```
  //printf("e is %u\n",&e);

  A = (unsigned int *) &A;

  for (i = 0; i <= 12; i++)
    printf("%d %u\n",i,A[i]);

  // you need to change the A reference in order to unbreak it
  A[8] = A[8] + 10;
  printf("A is %u \n",A);

  for (i = -4; i <= 10; i++)
    printf("%d %u\n",i,A[i]);
}

int main() {

   int A[100];
   unsigned int L[4];

   L[0]=100;
   L[1]=200;
   L[2]=300;
   L[3]=400;

   for (int i=0; i < 100; i++)
      A[i]=i;

   printf("\n\n\nmain is at %lu \n",main);

   printf("f is at %lu \n",f);
   printf("I am about to call f\n");
   f();
   printf("I called f\n");

out: printf("I am here\n\n\n");

}
```

Results:

- So, after experimenting a few different things with the program, I found that once you break the program, you need to change the ‘A’ reference in order to unbreak it.

Tony Maldonado
September 16, 2020
CS 471

- As we add variables one at a time, since they're 32 bits long, when you add them each it's going to take up half of the 64 bits of the architecture memory location. So that's why it takes an even number of variables to work correctly, since an even number would fill out both halves of the 64 bits.
- I found that as you add two variables, you need to increment the A reference register by two otherwise you get some weird behavior, or the expected behavior, I guess.
- I also found that if you add -10 instead of +10 to 'A', f just calls itself over and over again. If you do -20 instead, it'll print the printf before the call for 'f' and then do that over and over again.
- Overall, the experimenting helped me to understand what is going on behind the scenes in the memory runtime stack.