CIS 452

Lab 9 Report

Ashley Hendrickson

Muna Gigowski

Fall 2019

Memory Bound Protection

1. Describe the error precisely (nature of problem, location). Fix the problem and submit your corrected source code.

After compiling the program again, using the linked Electric Fence library, on the second run of the program when “notarealusername” was entered it caused a segmentation fault (core dumped) error. This is because Electric Fence detected a memory error within the sampleProgram1.c program, so it errored out. In stepping through the program, we discovered that the error occurred on this line:

data1 = malloc (SIZE);

The issue here is that SIZE is a constant that is defined above as 16 bytes. With each character being 1 byte in size, this means that the program will work perfectly for any user input that is 15 characters or less, but will “overflow” its allocated memory for any user input that is 16 characters or more - which “notarealusername” is. To correct this, we implemented a new version of the program that reads in each character from the user’s input one at a time, and puts them into a character array. Once all of the user’s characters have been read in, we look at the total size of the array and take “arraySize%16,” then use the resulting number to tell us how many additional blocks of 16-bytes we need to add to the total size of our constant SIZE. This way, SIZE gets determined dynamically based on the size of the user’s input, and we can ensure that their input will never be larger than the amount of memory we have allocated.

Corrected SampleProgram1 Code:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  int main()  {      size\_t len=16;      printf ("Please input username: ");      char c;      char\* str=malloc(sizeof(char)\*len);      len=0;      while(EOF!=(c=fgetc(stdin)) && c !='\n'){      str[len++]=c;      if(len%16==0)      str=realloc(str, sizeof(char)\*(len+16));      }      str[len++]='\0';      printf(" you entered %s\n",str);      free(str);      return 0;  } |

1. What's going on here, and why?

What is happening is that the compiler is intentionally obfuscating the addresses of the dynamic segments, which it does to make it more difficult for malicious code injection into the stack.