Arizona State University

Ira A. Fulton Schools of Engineering

CSE 543 Information Assurance and Security

Portfolio Report

Arizona State University

Hannah Ohunene Ajoge

ASU ID: 1217581417

Date Due: December 14, 2020
Date Submitted: December 14, 2020

Table of Contents

Introduction

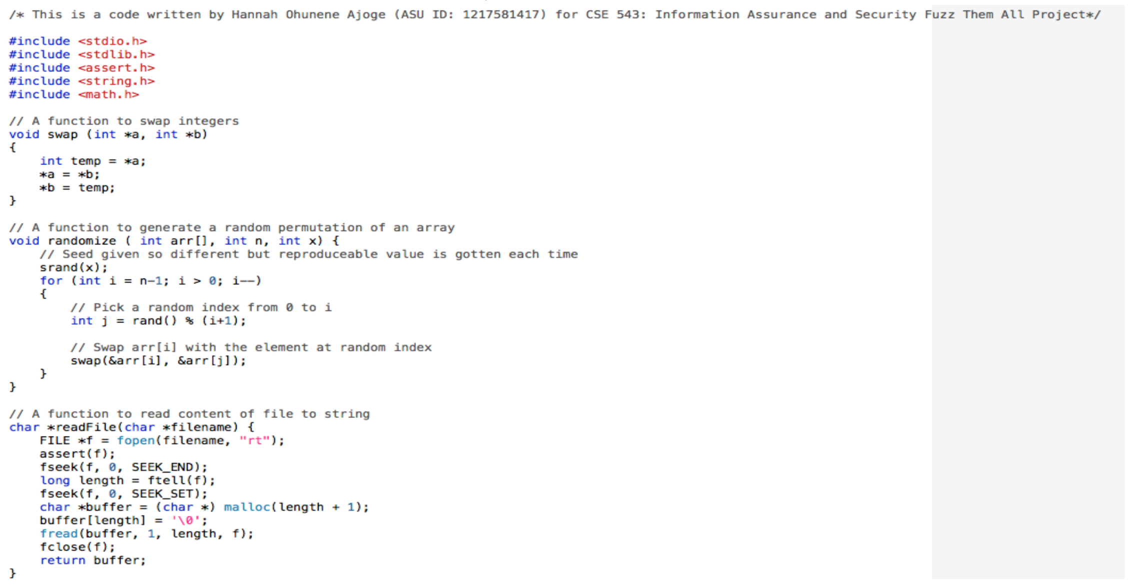
Information assurance and security is an evolving field which is continuously and dynamically changing in response to the contemporary business, society and technological advancements[1]. Information assurance can be divided into two subdomains – information assurance and information security. Cherdantseva and Hilton 2003 gave a clear definition of these two field which are adopted and quoted verbatim as following, viz:

> *Information Security is a multidisciplinary area of study and professional activity which is concerned with the development and implementation of security countermeasures of all available types (technical, organisational, human-oriented and legal) in order to keep information in all its locations (within and outside the organisation's perimeter) and, consequently, information systems, where information is created, processed, stored, transmitted and destructed, free from threats … Information Assurance is a multidisciplinary area of study and professional activity which aims to protect business by reducing risks associated with information and information systems by means of a comprehensive and systematic management of security countermeasures, which is driven by risk analysis and cost-effectiveness* [1]

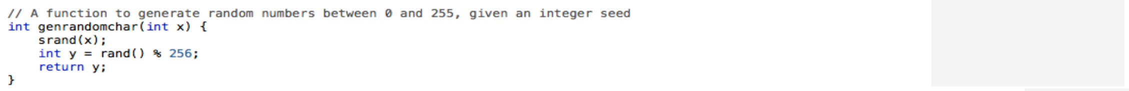
Because of the ambiguous nature of information assurance and security, Cherdantseva and Hilton 2003 proposed a reference model that encompass security development life cycle (SDLC), information taxonomy, security goals and security counter measures. In this project I used a simple and effective technique which based on randomly generated inputs for detecting software bugs[2]. This techniques is known as fuzzing and can be part of SDLC[3]. Fuzzing was specifically used in this project to identify segmentation faults.

Solutions

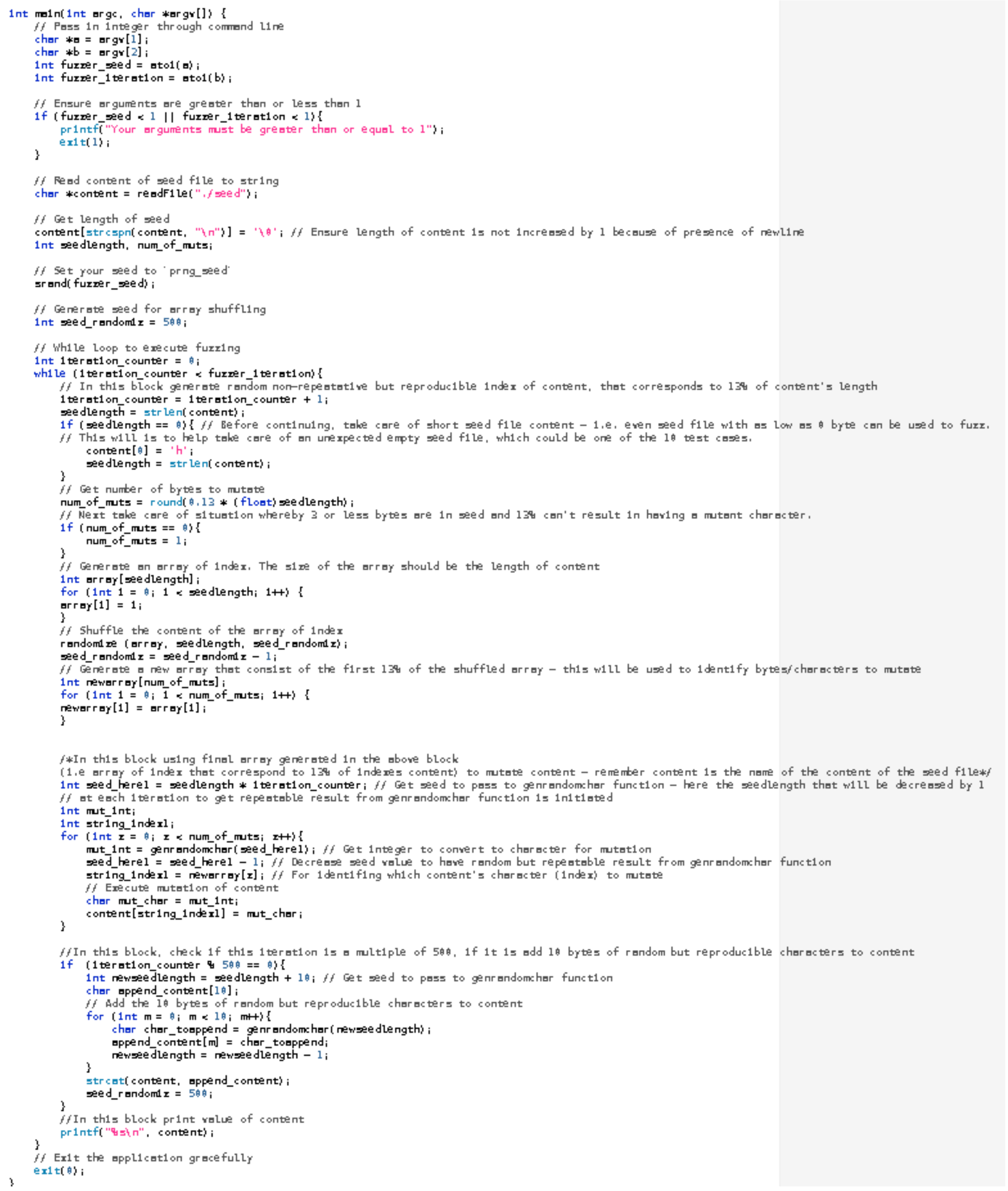
The fuzzer I developed is an executable from a C source code. I wrote the C source code and built the executeable using the command: *gcc fuzzer.c -o fuzzer -lm*
I needed the -lm flag because I included <math.h>
All the libraries I included are <stdio.h>, <stdlib.h>, <assert.h>, <string.h> and <math.h>
I executed all my coding, compiling and fuzzing of the test programs using Visual Studio Code. All my work was done in Ubuntu 20.04.
Screenshots of the C source code, which contains five functions is shown below:



```c
/* This is a code written by Hannah Ohunene Ajoge (ASU ID: 1217581417) for CSE 543: Information Assurance and Security Fuzz Them All Project*/

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include <math.h>

// A function to swap integers
void swap (int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// A function to generate a random permutation of an array
void randomize ( int arr[], int n, int x) {
    // Seed given so different but reproduceable value is gotten each time
    srand(x);
    for (int i = n-1; i > 0; i--)
    {
        // Pick a random index from 0 to i
        int j = rand() % (i+1);

        // Swap arr[i] with the element at random index
        swap(&arr[i], &arr[j]);
    }
}

// A function to read content of file to string
char *readFile(char *filename) {
    FILE *f = fopen(filename, "rt");
    assert(f);
    fseek(f, 0, SEEK_END);
    long length = ftell(f);
    fseek(f, 0, SEEK_SET);
    char *buffer = (char *) malloc(length + 1);
    buffer[length] = '\0';
    fread(buffer, 1, length, f);
    fclose(f);
    return buffer;
}
```



```c
// A function to generate random numbers between 0 and 255, given an integer seed
int genrandomchar(int x) {
    srand(x);
    int y = rand() % 256;
    return y;
}
```



```c
int main(int argc, char *argv[]) {
    // Pass in integer through command line
    char *a = argv[1];
    char *b = argv[2];
    int fuzzer_seed = atoi(a);
    int fuzzer_iteration = atoi(b);

    // Ensure arguments are greater than or less than 1
    if (fuzzer_seed < 1 || fuzzer_iteration < 1){
        printf("Your arguments must be greater than or equal to 1");
        exit(1);
    }

    // Read content of seed file to string
    char *content = readFile("./seed");

    // Get length of seed
    content[strcspn(content, "\n")] = '\0'; // Ensure length of content is not increased by 1 because of presence of newline
    int seedlength, num_of_muts;

    // Set your seed to `prng_seed`
    srand(fuzzer_seed);

    // Generate seed for array shuffling
    int seed_randomiz = 500;

    // While loop to execute fuzzing
    int iteration_counter = 0;
    while (iteration_counter < fuzzer_iteration){
        // In this block generate random non-repeatative but reproducible index of content, that corresponds to 13% of content's length
        iteration_counter = iteration_counter + 1;
        seedlength = strlen(content);
        if (seedlength == 0){ // Before continuing, take care of short seed file content - i.e. even seed file with as low as 0 byte can be used to fuzz.
        // This will is to help take care of an unexpected empty seed file, which could be one of the 10 test cases.
            content[0] = 'h';
            seedlength = strlen(content);
        }
        // Get number of bytes to mutate
        num_of_muts = round(0.13 * (float)seedlength);
        // Next take care of situation whereby 3 or less bytes are in seed and 13% can't result in having a mutant character.
        if (num_of_muts == 0){
            num_of_muts = 1;
        }
        // Generate an array of index. The size of the array should be the length of content
        int array[seedlength];
        for (int i = 0; i < seedlength; i++) {
        array[i] = i;
        }
        // Shuffle the content of the array of index
        randomize (array, seedlength, seed_randomiz);
        seed_randomiz = seed_randomiz - 1;
        // Generate a new array that consist of the first 13% of the shuffled array - this will be used to identify bytes/characters to mutate
        int newarray[num_of_muts];
        for (int i = 0; i < num_of_muts; i++) {
        newarray[i] = array[i];
        }


        /*In this block using final array generated in the above block
        (i.e array of index that correspond to 13% of indexes content) to mutate content - remember content is the name of the content of the seed file*/
        int seed_here1 = seedlength * iteration_counter; // Get seed to pass to genrandomchar function - here the seedlength that will be decreased by 1
        // at each iteration to get repeatable result from genrandomchar function is initiated
        int mut_int;
        int string_index1;
        for (int z = 0; z < num_of_muts; z++){
            mut_int = genrandomchar(seed_here1); // Get integer to convert to character for mutation
            seed_here1 = seed_here1 - 1; // Decrease seed value to have random but repeatable result from genrandomchar function
            string_index1 = newarray[z]; // For identifing which content's character (index) to mutate
            // Execute mutation of content
            char mut_char = mut_int;
            content[string_index1] = mut_char;
        }

        //In this block, check if this iteration is a multiple of 500, if it is add 10 bytes of random but reproducible characters to content
        if  (iteration_counter % 500 == 0){
            int newseedlength = seedlength + 10; // Get seed to pass to genrandomchar function
            char append_content[10];
            // Add the 10 bytes of random but reproducible characters to content
            for (int m = 0; m < 10; m++){
                char char_toappend = genrandomchar(newseedlength);
                append_content[m] = char_toappend;
                newseedlength = newseedlength - 1;
            }
            strcat(content, append_content);
            seed_randomiz = 500;
        }
        //In this block print value of content
        printf("%s\n", content);
    }
    // Exit the application gracefully
    exit(0);
}
```

Results

The executable I implemented which is called fuzzer, crashed three instructor provided test programs p_0, p_1 and p_2 at iterations 755, 1412 and 3842 respectively.

Contributions

This project is an individual project, so I implemented every aspect myself. I designed an algorithm (see below) and then wrote the C source code (fuzzer.c - see above screenshots). I then compiled the source code and utilized the resulting program known as fuzzer to execute segmentation fault on the three given test programs.
Algorithm:
Before I started coding I decided that my code will accomplish the following.
1. Input:
a. My code should be able to take an input see file, read the content and assign the content to a variable named content. But the code must not modify the seed file.
b. My code should accept two arguments which are both int32. These integers should not be less than or equal to 0.
2. Iterations:
a. At each iteration, my code should calculate the length of the content variable, take a rounded up 13% of the length and randomly but reproducibly mutate 13% of the bytes.
b. Still within each iteration check if that iteration is a multiple of 500 (of course not zero as the while loop ensures that at the beginning). If it is, it should add 10 bytes which are randomly but reproducibly generated.
3. My code would ensure at the end of each iteration the content is printed to stdout.
4. Finally my code should ensure that it graciously exit with exit(0).

Testing of fuzzer:
I tested the fuzzer I developed on three test programs provided by the instructor, from within the folders containing the test programs; using the following commands :

```
counter=0; while read -r line; counter=$((counter+1)); do echo $line | ./prog_0; if [[ $? -eq
139 ]]; then echo -e "Iteration: $counter" '\n' $line   > crashinput; break; else echo -e
"Iteration: $counter" '\n'  $line  >> noncrashinput; fi  done < <(./fuzzer 1 100000)

counter=0; while read -r line; counter=$((counter+1)); do echo $line | ./prog_1; if [[ $? -eq
139 ]]; then echo -e "Iteration: $counter" '\n' $line   > crashinput; break; else echo -e
"Iteration: $counter" '\n'  $line  >> noncrashinput; fi  done < <(./fuzzer 1 100000)

counter=0; while read -r line; counter=$((counter+1)); do echo $line | ./prog_2; if [[ $? -eq
139 ]]; then echo -e "Iteration: $counter" '\n' $line   > crashinput; break; else echo -e
"Iteration: $counter" '\n'  $line  >> noncrashinput; fi  done < <(./fuzzer 1 100000)
```

These commands read each print out and attempt to crash the test programs p_0, p_1 and p_2 respectively. A counter is also initiated to know which iteration that each input to the program is generated. The commands will send iteration count and the bytes to a file called crashinput and stop execution once they get the error code 139 - Segmentation fault. Else if no error is encountered, the iteration count and bytes are sent to the file known as noncrashinput. The idea is to crash the program and know which iteration the crash took place.

Lessons Learned

Before doing this project I did not know how to code in C. But because Python and R which I am proficient in were slow/ insufficient to meet the project requirement, I had to learn to code in C. I will find this new skillset useful in subsequent courses and in my professional endeavor.

References

1. Y. Cherdantseva and J. Hilton, "A Reference Model of Information Assurance & Security," 2013 International Conference on Availability, Reliability and Security, Regensburg, 2013, pp. 546-555, doi: 10.1109/ARES.2013.72.

2. H. Peng, Y. Shoshitaishvili and M. Payer, "T-Fuzz: Fuzzing by Program Transformation," 2018 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, 2018, pp. 697-710, doi: 10.1109/SP.2018.00056.

3. D. O. Kengo, T. Fujikura and R. Kurachi, "Shift Left: Fuzzing Earlier in the Automotive Software Development Lifecycle using HIL Systems", escar EU 2018, Nov 2018. https://www.researchgate.net/profile/Toshiyuki_Fujikura2/publication/330384207_Shift_Left_Fuzzing_Earlier_in_the_Automotive_Software_Development_Lifecycle_using_HIL_Systems/links/5c3d92cea6fdccd6b5ada612/Shift-Left-Fuzzing-Earlier-in-the-Automotive-Software-Development-Lifecycle-using-HIL-Systems.pdf