

A Report on
Assignment - I : Format String Fuzzing
Introduction to Cybersecurity
Name: Padam Jung Thapa (2623560)

Format String Fuzzing:

It is a type of software testing technique used to identify vulnerabilities in a program's handling of formatted strings. A formatted string is a string that contains placeholders for values that are substituted at runtime and is a type of software vulnerability that occurs when a program takes an untrusted user input and uses it as the format string argument for a formatted output function such as `printf()` or `sprintf()`. The placeholders are often specified using special format specifiers, such as `"%d"` for an integer or `"%s"` for a string.

Title: Environment Variable Fuzzer and Modifier

Goal: The purpose of this lab is to demonstrate a format string attack leading to the modification of an environment variable.

Description: The program is written in the C programming language and is divided into three main functions: `'printenv'`, `'fuzzee'`, and `'fuzzer'`.

1. `'printenv'`: This helper function prints all the environment variables passed to the program. It iterates through the `'envp'` array and prints the index, memory address, and value of each environment variable.
2. `'fuzzee'`: This function is provided as-is and should not be modified. It takes a format string input (`'fmt_input'`) and writes it to an output buffer (output). This function is used to demonstrate how the crafted payload is used to modify the `"SECRET_ENV"` variable.
3. `'fuzzer'`: This is the main function where the payload is crafted and the attack is performed. The function searches for the `"SECRET_ENV"` variable in the `'envp'` array (Noted that also modified in the Makefile environment). If found, it crafts a payload to modify the value of the `"SECRET_ENV"` variable to `"hacked"`. The payload is then passed to the `'fuzzee'` function to apply the modification. Finally, the function checks if the attack was successful by comparing the new value of the `"SECRET_ENV"` variable to the desired value (`"hacked"`).

The main function, `'main'`, serves as the entry point of the program. It first calls the `'printenv'` function to print the initial state of the environment variables. Then, it calls the `'fuzzer'` function to perform the attack. Finally, it calls the `'printenv'` function again to print the updated state of the environment variables, showing the result of the attack.

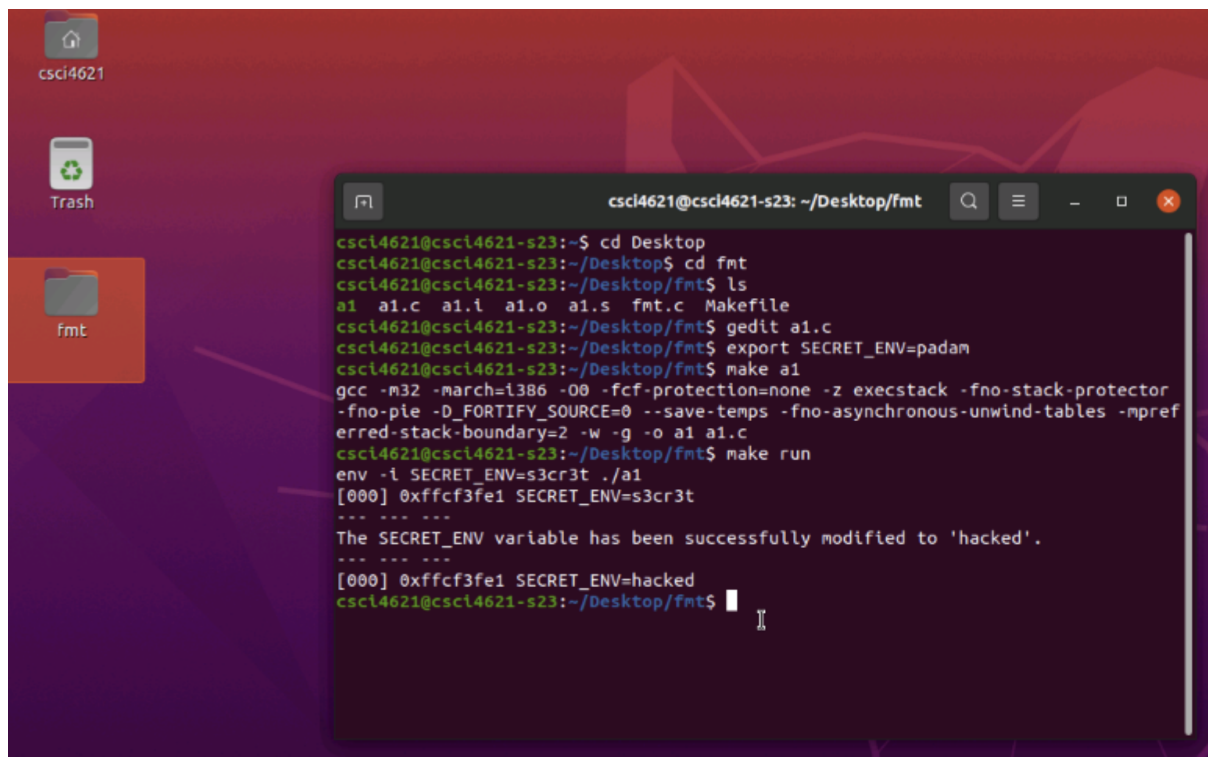
Bonus Part: Being a graduate student, the bonus part of the assignment is also solved which is the payload crafting and modification of the "SECRET_ENV" variable. The payload is crafted using the 'snprintf' function to write the desired value ("hacked") into the 'buffer'. The 'memcpy' function is then used to copy the crafted payload from the buffer to the memory location of the "SECRET_ENV" variable. The attack is successful if the "SECRET_ENV" variable is modified to the value "hacked".

How to run the program?

1. Well, the code file is saved the code named "a1.c". Open the terminal and navigate to the directory where "a1.c" is located in the desktop within the fmt folder.
2. Before running the program, set the 'SECRET_ENV' variable by executing the following command: 'export SECRET_ENV=some_value' (replace 'some_value' with any value you want to assign to the environment variable initially, I have set mine as my name - 'padam').
3. Compile the code by running the command: 'make a1' and finally run the program using the updated run command in the Makefile: 'make run'.

The snapshot of the output and runnable code is attached here down below:

Also, the code solution is on the NEXT PAGE down below:



```
csci4621@csci4621-s23:~$ cd Desktop
csci4621@csci4621-s23:~/Desktop$ cd fmt
csci4621@csci4621-s23:~/Desktop/fmt$ ls
a1 a1.c a1.i a1.o a1.s fmt.c Makefile
csci4621@csci4621-s23:~/Desktop/fmt$ gedit a1.c
csci4621@csci4621-s23:~/Desktop/fmt$ export SECRET_ENV=padam
csci4621@csci4621-s23:~/Desktop/fmt$ make a1
gcc -m32 -march=i386 -O0 -fcf-protection=none -z execstack -fno-stack-protector
-fno-pie -D_FORTIFY_SOURCE=0 --save-temps -fno-asynchronous-unwind-tables -mpref
erred-stack-boundary=2 -w -g -o a1 a1.c
csci4621@csci4621-s23:~/Desktop/fmt$ make run
env -i SECRET_ENV=s3cr3t ./a1
[000] 0xffcf3fe1 SECRET_ENV=s3cr3t
...
The SECRET_ENV variable has been successfully modified to 'hacked'.
...
[000] 0xffcf3fe1 SECRET_ENV=hacked
csci4621@csci4621-s23:~/Desktop/fmt$
```

Solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

// Helper function to print environment variables
void printenv(char *envp[]) {
    for (int i = 0; envp[i]; i++)
        printf("[%03d] %p %s\n", i, envp[i], envp[i]);
}

// Function fuzzee - not to be modified
void fuzzee(int argc, char *argv[], char *envp[], char *fmt_input, char *output) {
    // Use the fmt_input format string to write into the output buffer
    sprintf(output, fmt_input);
}

// Main fuzzer function to find and modify the SECRET_ENV variable
void fuzzer(int argc, char *argv[], char *envp[]) {
    char buffer[512];
    char *secret_env_addr = NULL;
    char *temp_envp[2];

    // Search for the SECRET_ENV variable in the environment variables
    for (int i = 0; envp[i]; i++) {
        if (strncmp(envp[i], "SECRET_ENV=", 11) == 0) {
            secret_env_addr = envp[i] + 11; // Skip the "SECRET_ENV=" part.
            temp_envp[0] = envp[i];
            temp_envp[1] = NULL;
            break;
        }
    }
}
```

```

if (secret_env_addr) {
    // ----- BONUS PART: Craft the payload and modify the SECRET_ENV variable -----
    snprintf(buffer, sizeof(buffer), "hacked");
    memcpy(secret_env_addr, buffer, strlen(buffer) + 1);

    // Call fuzzee() with the crafted payload
    char output[512];
    fuzzee(argc, argv, temp_envp, buffer, output);

    // Check if the attack was successful
    if (strcmp(secret_env_addr, "hacked") == 0) {
        printf("The SECRET_ENV variable has been successfully modified to 'hacked'.\n");
    } else {
        printf("The attack failed. The SECRET_ENV variable is still: %s\n", secret_env_addr);
    }
    // ----- END OF BONUS PART -----
} else {
    printf("SECRET_ENV not found in the environment variables.\n");
}
}

// Main function - not to be modified
int main(int argc, char *argv[], char *envp[]) {
    // Print the initial environment variables
    printenv(envp);
    printf("--- --- ---\n");

    // Run the fuzzer
    fuzzer(argc, argv, envp);
    printf("--- --- ---\n");

    // Print the updated environment variables
    printenv(envp);
}

```