USING LIBFUZZER AND AFL TO FUZZ YOUR OWN C/C++ IMPLEMENTATION

SUBJECT NAME: CRYPTOGRAPHY AND NETWORK SECURITY

SUBJECT CODE: CS6008

MODULE: 4

NAME	BHUVANESHWAR S
REG.NO	2019103513
DATE	05/04/2022

USING LIBFUZZER AND AFL TO FUZZ YOUR OWN C/C++ IMPLEMENTATION

AIM :

To implement a c/c++ code using fuzzer tools.

TOOLS INVOLVED:

- LIBFUZZER
- AFL FUZZER
- VISUAL STUDIO CODE
- KALI LINUX TERMINAL (WSL)
- WINDOWS (OPERATION SYSTEM)

PROBLEM DESCRIPTION:

Fuzzing is to try massive numbers of random inputs to code in order to trigger a vulnerability. You create a testbench for the code of interest, pair it with fuzzing engine that generates random data, and launch it on some server somewhere. Hours days, or weeks later – if your testbench is solid , it comes back with a set of inputs that cause the code to crash. This process may be accelerated by:

- Using a sanitizer: compiler-supported sanitizers instrument binaries with extra code to check for illegal conditions, such as out-of-bounds memory accesses, that may not cause an immediate crash. This makes the code under test more likely to fail and thus reduces the fuzzer running time.
- Using coverage-driven fuzzing: fuzzers can monitor program states reached under different inputs and guide the inputs and guide the inputs in a way that tends to produce new ones.

libFuzzer can be checked out from LLVM's Subversion repository and built using their directions. You supply a test driver as a function called LLVMFuzzerTestOneInput with C linkage. The result is a standalone program that exercises the code inside that function. It uses some Clang compiler-supplied instrumentation, via the -fsanitize-coverage option, to monitor which paths are exercised, so gcc is not an option. It must be compiled with -fsanitize=memory to ensure no initialization is missed.

AFL is a standalone tool that uses binary rewriting to instrument the code being tested. It supplies wrapper compilers that call either Clang or gcc as necessary. The test driver is written as a main program that takes the random string from standard input, which means each run is a separate process. However, if you use Clang, there is a special "fast" mode that instruments your code as a compiler pass, rather than a final object code rewrite. This means the instrumentation itself can be optimized, producing faster binaries. So clang-fast is used.

Also, AFL is more mature and has more sophisticated mutation algorithms, and though its one-process-per-test approach is slower, the special Clang support compensates. Address sanitizing seems much faster than memory sanitizing, and you can always re-run all the (unique path) test cases afterwards.

INPUT:

Getting an input from a files.

OUTPUT:

Given input string is to check whether it is a palindrome or not.

SCREENSHOT:

LIBFUZZER

FILENAME : libfuzz.cc

```
#include <stdint.h>
#include <stddef.h>

bool fuzz(const uint8_t *data, size_t s){
    for(int i = 0 ; i <= s/2 ; i++){
        if(data[i] != data[s-i-1])
            return false;
    }
    return true;
}

extern "C" int LLVMFuzzerTestOneInput(const uint8_t *data, size_t s){
    fuzz(data,s);
    return 0;
}
```

In the function “fuzz”, in the for loop the value of i is equal to half of value of size, it will give error because it not condition for palindrome. Clang+ is used to add libfuzzer to our program. LLVMFuzzerTestOneInput is the driver of libfuzzer to test fuzz() function. We will compile and see what happens. clang++ -g –

fsanitize=address,fuzzer filename is used to compile source file for fuzzing. Max_len=20 is used to run fuzzer with max input size of 5.

```
┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial]
└─$ clang++ -g -fsanitize=address,fuzzer libfuzz.cc -o fuzz

┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial]
└─$ ls -la
total 3272
drwxrwxrwx 1 bhuvan bhuvan    4096 Jun  5 19:29 .
drwxrwxrwx 1 bhuvan bhuvan    4096 Jun  5 19:28 ..
-rwxrwxrwx 1 bhuvan bhuvan 1675136 Jun  5 19:28 a.out
-rwxrwxrwx 1 bhuvan bhuvan 1675136 Jun  5 19:29 fuzz
-rwxrwxrwx 1 bhuvan bhuvan     310 Jun  5 19:28 libfuzz.cc
```

```
┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial]
└─$ ./fuzz -max_len=5
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 360876546
INFO: Loaded 1 modules   (7 inline 8-bit counters): 7 [0x54bee0, 0x54bee7),
INFO: Loaded 1 PC tables (7 PCs): 7 [0x524070,0x5240e0),
===================================================================
==384==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60200000002f at pc 0x00000050cc52 bp 0x7ffde865af70 sp 0x7ffde865af68
READ of size 1 at 0x60200000002f thread T0
    #0 0x50cc51 in fuzz(unsigned char const*, unsigned long) /mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial/libfuzz.cc:6:23
    #1 0x50cd84 in LLVMFuzzerTestOneInput /mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial/libfuzz.cc:13:5
    #2 0x4409b3 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial/fuzz+0
x4409b3)
    #3 0x441f20 in fuzzer::Fuzzer::ReadAndExecuteSeedCorpora(std::vector<fuzzer::SizedFile, fuzzer::fuzzer_allocator<fuzzer::SizedFile> >&) (/mnt/e/clg 6th
sem/crypto&net security/assignment/fuzz/trial/fuzz+0x441f20)
    #4 0x442482 in fuzzer::Fuzzer::Loop(std::vector<fuzzer::SizedFile, fuzzer::fuzzer_allocator<fuzzer::SizedFile> >&) (/mnt/e/clg 6th sem/crypto&net securi
ty/assignment/fuzz/trial/fuzz+0x442482)
    #5 0x42f92d in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) (/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz
/trial/fuzz+0x42f92d)
    #6 0x45b6a2 in main (/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial/fuzz+0x45b6a2)
    #7 0x7fcd95f007fc in __libc_start_main csu/../csu/libc-start.c:332:16
    #8 0x4238d9 in _start (/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial/fuzz+0x4238d9)

0x60200000002f is located 1 bytes to the left of 1-byte region [0x602000000030,0x602000000031)
allocated by thread T0 here:
    #0 0x50a37d in operator new[](unsigned long) (/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial/fuzz+0x50a37d)
    #1 0x4408c2 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial/fuzz+0
x4408c2)
    #2 0x441f20 in fuzzer::Fuzzer::ReadAndExecuteSeedCorpora(std::vector<fuzzer::SizedFile, fuzzer::fuzzer_allocator<fuzzer::SizedFile> >&) (/mnt/e/clg 6th
sem/crypto&net security/assignment/fuzz/trial/fuzz+0x441f20)
    #3 0x442482 in fuzzer::Fuzzer::Loop(std::vector<fuzzer::SizedFile, fuzzer::fuzzer_allocator<fuzzer::SizedFile> >&) (/mnt/e/clg 6th sem/crypto&net securi
ty/assignment/fuzz/trial/fuzz+0x442482)
    #4 0x42f92d in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) (/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz
/trial/fuzz+0x42f92d)
    #5 0x45b6a2 in main (/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial/fuzz+0x45b6a2)
    #6 0x7fcd95f007fc in __libc_start_main csu/../csu/libc-start.c:332:16

SUMMARY: AddressSanitizer: heap-buffer-overflow /mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial/libfuzz.cc:6:23 in fuzz(unsigned char const*, u
nsigned long)
Shadow bytes around the buggy address:
```

Here the error says that heap-overflow has occurred at address 0x6020000002f at instruction pc 0x000000050xx52 bp 0x7ffde865af70 bp 0x7ffde865af68.

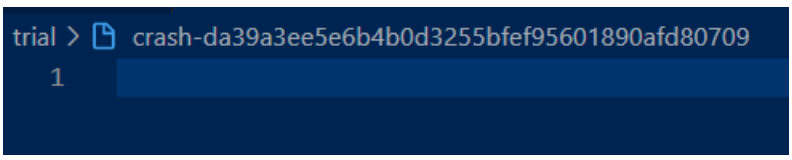
Program after removal of discovered bug:

```
#include <stdint.h>

#include <stddef.h>

#include <signal.h>


bool isPalindrome(const uint8_t *data, size_t s){

    for(int i = 0 ; i < s/2 ; i++){

        if(data[i] != data[s-i-1])
```

```
            return false;
    }


    return true;
}


extern "C" int LLVMFuzzerTestOneInput(const uint8_t *data, size_t s){
    isPalindrome(data,s);
    return 0;
}
```

```
┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial]
└─$ clang++ -g -fsanitize=address,fuzzer libfuzz.cc -o vuln

┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/crypto&net security/assignment/fuzz/trial]
└─$ ./vuln -max_len=3
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 3727234840
INFO: Loaded 1 modules   (7 inline 8-bit counters): 7 [0x54bee0, 0x54bee7),
INFO: Loaded 1 PC tables (7 PCs): 7 [0x524070,0x5240e0),
INFO: A corpus is not provided, starting from an empty corpus
#2      INITED cov: 4 ft: 4 corp: 1/1b exec/s: 0 rss: 29Mb
#6      NEW    cov: 5 ft: 5 corp: 2/3b lim: 3 exec/s: 0 rss: 30Mb L: 2/2 MS: 4 ShuffleBytes-ChangeBit-ChangeByte-CopyPar
t-
#7      NEW    cov: 7 ft: 7 corp: 3/5b lim: 3 exec/s: 0 rss: 30Mb L: 2/2 MS: 1 InsertByte-
#4194304        pulse  cov: 7 ft: 7 corp: 3/5b lim: 3 exec/s: 1398101 rss: 217Mb
#8388608        pulse  cov: 7 ft: 7 corp: 3/5b lim: 3 exec/s: 1398101 rss: 403Mb
#16777216       pulse  cov: 7 ft: 7 corp: 3/5b lim: 3 exec/s: 1290555 rss: 775Mb
#33554432       pulse  cov: 7 ft: 7 corp: 3/5b lim: 3 exec/s: 1118481 rss: 1159Mb
#67108864       pulse  cov: 7 ft: 7 corp: 3/5b lim: 3 exec/s: 1100145 rss: 1159Mb
#134217728      pulse  cov: 7 ft: 7 corp: 3/5b lim: 3 exec/s: 1109237 rss: 1159Mb
```

Crash reported at when the input string is null.



AFL

Palindrome.cpp

```
#include <iostream>

#include <bits/stdc++.h>
```

```cpp
#include <fstream>

using namespace std;

bool isPalindrome(string data, int s)
{
    for (int i = 0; i < s / 2; i++)
    {
        if (data[i] != data[s - i - 1])
            return false;
    }

    return true;
}

void parse_file(char *filename,vector<string>&input){
    string i;

    ifstream input_file(filename);
    if(!input_file.is_open()){
        cerr << "Could not open the file - '"<< filename << "'" << endl;
        exit(-1);
    }
    while(getline(input_file,i)){
        input.push_back(i);
    }
    input_file.close();
}
```

```
int main(int argc, char **argv)
{
    string data;
    if (argc < 2)
    {
        cout << "enter filename";
        return -1;
    }

    vector<string>input;
    parse_file(argv[1],input);

    for(auto i : input){
        if(isPalindrome(i,i.size())){
            cout<<i<<"\t -->  TRUE\n";
        }
        else{
            cout<<i<<"\t -->  FALSE\n";
        }
    }

    return 0;
}
```

Compile and run the program using afl-g++