Secure and Dependable Systems
Constructor University
Dr. Jürgen Schönwälder

Module: CO-566
Date: 2023-02-16
Due: 2023-02-23

Problem Sheet #2

Problem 2.1: *test coverages* (1+1+1+1+1 = 5 points)

The following Rust function calculates the greatest common divisor of two integers.

```rust
pub fn gcd(mut a: i64, mut b: i64) -> i64 {
    while (a > 0) && (b > 0) {
        if a > b {
            a -= b;
        } else {
            b -= a;
        }
    }
    a + b
}
```

Here is a `main()` program exposing this function to the command line. The `main()` function returns a result in order to deliver errors (note how the `?` operator is used do propagate parsing errors).

```rust
use gcd::gcd;
use std::error::Error;

fn main() -> Result<(), Box<dyn Error>> {
    let args: Vec<String> = std::env::args().collect();
    if args.len() > 1 {
        let mut g: i64 = args[1].parse()?;
        for arg in args.iter().skip(2) {
            let y: i64 = arg.parse()?;
            g = gcd(g, y);
        }
        println!("{g}");
    }
    Ok(())
}
```

Lets assume the program has been compiled into the executable `gcd`. Your task is to write down a *minimal* number of shell commands that achieve different code coverages.

a) Which commands are necessary to achieve function coverage? Explain.

b) Which commands are necessary to achieve statement coverage? Explain.

c) Which commands are necessary to achieve branch coverage? Explain.

d) Which commands are necessary to achieve path coverage? Explain.

e) Which commands are necessary to achieve condition coverage? Explain.

Problem 2.2: *clang libfuzzer* (3+2 = 5 points)

The `clang` compiler support a fuzzying API, which makes it very easy to fuzz C functions. Below is a simple example:

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

static int memcmp(void *s1, const void *s2, size_t n)
{
    unsigned char *a = (unsigned char *) s1;
    unsigned char *b = (unsigned char *) s2;

    for (int i = 0; i < n; i++) {
        if (a[i] < b[i]) {
            return -1;
        }
        if (a[i] > b[i]) {
            return 1;
        }
    }
    return 0;
}

int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size)
{
    char *msg = "FUZZ";
    (void) memcmp(msg, data, size);
    return 0;
}
```

By compiling the code with `-fsanitize=fuzzer`, you obtain an executable that will feed fuzzed inputs to the function `LLVMFuzzerTestOneInput()`, from where you can call any function you want to test. It is usually a good idea to enable additional `clang` sanitizers by compiling the code with `-fsanitize=fuzzer,address,undefined`.

a) Fuzz the example shown above. What is the test case found by the fuzzer that causes the implementation of `memcmp()` to fail? What is the problem here? Explain.

b) Take a function of medium complexity that you wrote in the past and which is processing strings. (In the operating systems course you likely wrote a function (as part of the word guessing game) that selects a random word in a text string, which is then replaced by underscore characters and the word is returned as an allocated copy, `char* hide_word(char *text)`.) Implement a suitable fuzzying wrapper and report which bugs were found (if any).