SECURE SYSTEMS ENGINEERING

Report on Security Measures used to secure our binary

Soham Tripathy (CS20B073), Arunesh J B (CS20B009)

14-04-2023

CS6570 - LAB 5 Report

Defence Phase

Introduction

Each team was given a secret function and were asked to submit an executable which asks the user for the function to run(sum or hidden). Only super users are allowed to access the hidden function with the key provided in the readme (submitted to the TAs).

Aim

We have to build a executable with minimal vulnerabilities to make it difficult for attackers to find our hidden function.

Methods to secure our binary

1. **Programming Language :** One way to make our binary secure is by using a secure programming language. Languages like Rust, Go, Swift etc provide strong guarantees and memory safety. We used Rust to secure our executable because it prevents common programming errors like null pointer dereferences, buffer overflows, and data races.
2. **Encryption :** We encrypt our function using the SHA-256 encryption algorithm. This takes an input (in our case the key and produces a hashvalue which is mapped to the function pointer of our secret function), performs 12 rounds of salting and returns a 256 bit key which is mapped to the function pointer of our secret function.
3. **Obfuscation :** Intentionally concealing or obscuring important information in our code to mislead, decieve and make it unclear for users to understand. We did not implement sum/hidden function in the most generic way instead we use a while loop (or bit manupliations) to perform the exact same task. Given below is the example of sum function.

```rust
//sum
fn sum(a: i32, b: i32) -> i32 {
    a+b
}
```

```rust
//Obfuscated sum
fn sum(a: i32, b: i32) -> i32 {
    let mut x = a ^ b;
    let mut y = a & b;
    while y != 0 {
        y = y << 1;
        let carry = x & y;
        x = x ^ y;
        y = carry;
    }
    x
}
```

4. **Stripping** : We strip our binary to remove the debug (like function names, global variables name, etc.) info from the binary which not only makes it difficult for attackers to understand the code but also reduces the size of the binary and makes it more compact and more efficient. For example in case of a stripped binary the function names will be replaced with hex to make it difficult for attackers to understand the binary.

```
sum() changes to FUN_1290424(), when decompiled.
```

5. **Miscellaneous** : We added all the given functions in our code to decieve the attackers and mislead them from finding our secret function.