In the name of God

Sharif University of Technology
Department of Computer Engineering

CE 874: Secure Software Systems

*Mehdi Kharrazi* *dey $1^{th}$, 1399*

# Homework 2[*]

## 1 BAT: Binary Analysis Tool

**Part III: Disassembling elves with Capstone**

### 1.1 Instructions

In the previous assignment, we made a disassembler for executables in Linux, now we are going to make use of that to plot the callgraph of a program. To do this, we need to track "call" and "ret" instructions. Is there anyway this would change? (Try playing with optimization levels to see if it makes a difference).
The task expected of you in this assignment, is to add a module to BAT, which gets a binary as input and outputs the callgraph in DOT format. Your callgraph should consists of **function names** and their relationships.

### 1.2 Syscalls

How are syscalls called? You can check out this blog post for a good guide on how syscalls are called from assembly. Detect all the syscalls of a program, in each function. So that in your callgraph, each node, which represents a function also holds this piece of data: the list of syscalls called from that functions.

### 1.3 Policy Based Vulnerability Detection

One of the methods for detecting suspicious programs, is policy based detection. An anti-virus has a big database of rules, and checks a binary against those rules and flags the binary as a threat if it violates those rules. Implement policy-checking in BAT: your program should take a set of rules and check a binary to see if those rules are violated.

The rules we want to handle are simple and are only of the following types: 1) Checking whether a sequence of functions can be called. (e.g. f1() -> f2() -> f3() can happen in the program) 2) Checking whether a sequence of syscalls can happen. (e.g. syscall 101 -> syscall 202)

A simple use-case is double free bugs, if you detect that a resource is freed twice, that means trouble.

[*] Acknowledgement: This homework was developed by Iman Hosseini and Solmaz Salimi

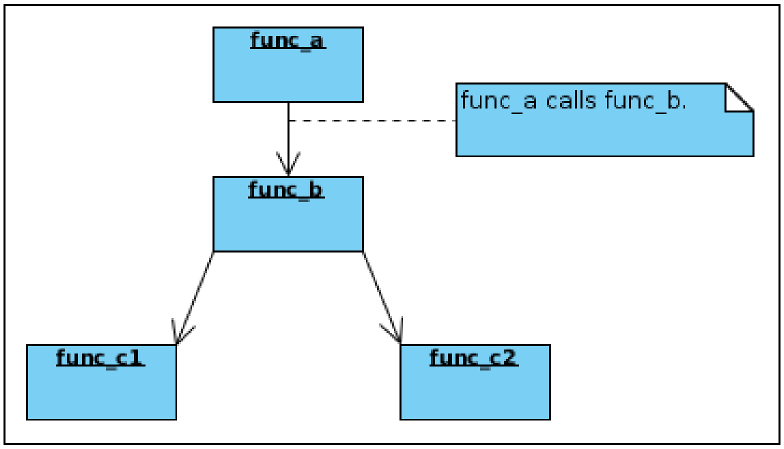

Figure 1: Call Graph

## 1.4 Delivery

You should submit a report, explaining your program. You should also submit your code and explain how to run it.

# 2 Source Code Analysis

## 2.1 Instructions

In this part of assignment, you get familiar with **Source Code Analysis** and its application in Vulnerability Detection.

Imagine, that you are given the following code:

```
#include <stdioh.h>
int  random_func (int arg0, int arg1, ..., int argn) {
int var0 = random_number;
int var1 = random_number;
.
.
.
int varm = random_number;
/*
aritmetics with + - / *
*/
return return_value;;
}
int main (int argc , char** argv) {
int data = random_func (random_number0 , ..., random_numbern);
printf("%d",data);
return 0;
 }
```

We can statically parse this code and find a **Symbolic formula** for each variable. Then we can decide how the value of this varaible changes based on input arguments.

In this part of assignment, first you parse the source code and find symbolic formula for each variable. Based on this formula you can report a domain in which the variable is vulnerable to either **Division by zero** or **Integre overflow**.

Then you can check how the main calls the random_function and solve the formula with the given argument during function call, you can solve the symbolic formula for these input and calculate the exact value for each varible and report **Division by zero** and **Integre overflow** vulnerabilities. You can use any availble parser, or deveope your own *C* parser.

## 2.2 Delivery

You should submit your code which takes a C file as input and prints variables' symbolic formulas (sorted alphabetically based on variables names) and exact values. If you detect any vulnerability you should print the variable's name related to it.