In the name of God

Sharif University of Technology
Department of Computer Engineering

CE 815: Secure Software Systems

Mehdi Kharrazi *Azar 27^{th}, 1400*

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. Then, simply convert dot files to PNG images. Your callgraph should consists of **function names** and their relationships.

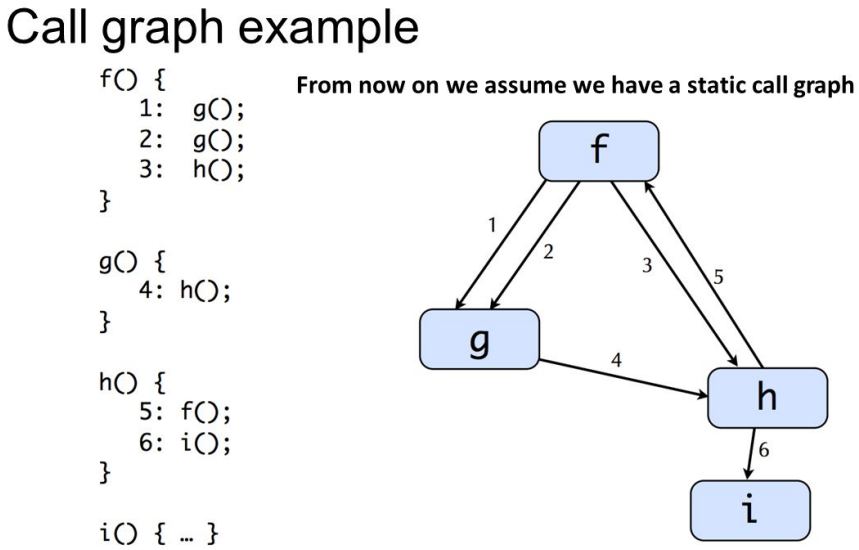

Figure 1: Call Graph

*Acknowledgement: This homework was developed by Iman Hosseini and Solmaz Salimi, edited by Razieh Eskandari and Parisa EzzatPanah

1.2 Syscalls

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.*

Hint: How are syscalls called? You can check out this blog post for a good guide on how syscalls are called from assembly.

1.3 Static Detection of Vulnerabilities

Suppose you receive a complete call graph that is provided by static binary code analysis of the new application program A and its third-party libraries. Also, you receive one or more stack traces that are provided based on dynamic code analysis of known vulnerable program B during execution of the application program , which contains the one or more vulnerable functions included in the one or more third - party libraries.

You should write a program which takes a set of stack traces and check a binary to see if these traces could happen in a binary.

Actually, situations like these is an anti-virus program which has a big database of rules. Each rule is derived from stack trace of vulnerable program and shows which sequence of calls will lead to vulnerability. Hence, the antivirus checks a new binary against those rules and flags the binary as a potential threat if it contains any of those malicious rules.

A simple use-case for these rule is related to double free bugs: if you detect that a resource is freed twice, that means trouble.

This method, used for detecting suspicious programs, is called *policy based detection.*

Your task is to implement policy-checking in BAT: your program should take a set of rules and check a binary to see if those rules are occurred. No mater how these rules are provided, you should only check the presence/absence of these rules in the call-graph of given program.
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)

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 Tool

2.1 Instructions

In these series of assignment, you will implement some simple static analyses tools for Java Program enabling vulnerability detection.

Part II: More Advanced Integer Analysis: Test Input Generation

In the previous HW, you had implemented a simple **constant integer** analyzer to detect: division by zero, negative array index and bad shift. In this assignment, you should first add another analyzer to detect integer overflow. As you see in the handout examples, you should also handle the cases in which a statement may execute zero or more times, using control-flow structures (i.e., if, for and while) .

Actually, you should statically parse the code and find a **Symbolic formula** for each integer variable. Then, you can decide how the value of this variable changes based on the given input arguments. Based on extracted formula, you can report a domain in which the variable is vulnerable to either **Division by zero**, **Negative array index**, **Bad shift** or **Integer overflow**.

A) suppose that the main function calls other function with some concrete values. In this case, your task is very straigh-forward:just replace the concrete values in your symbolic formula (i.e., symbolic execution) of each statement to check whether the aforementioned errors/vulnerabilities may occur in the program.

B) Now, suppose a more realistic case : the main function calls other functions with the user provided input. Hence, you should solve the symbolic formula for these input and calculate the exact value for each variable which certainly leads to the aforementioned vulnerabilities in the program.

Hint1: A key piece of machinery used by symbolic execution is an SMT solver. For part b, you could use the Z3 solver from Microsoft Research. Z3 has bindings for various programming languages, including .Net, C, C++, Java, Python etc. You will invoke Z3 using its API.

Hint2: You can use any available Java parser, or develop your own Java parser. One possible solution could be deriving an AST first, and then, add your own code in order to modify some nodes of AST so that it could handle the desired symbolic formula parsing. You can find sample Java files in handouts repos.

2.2 Delivery

You should submit your code which takes a Java file as input and prints the symbolic execution of statements. At the end of each function, It should print the final symbolic formulas for every variable (sorted alphabetically based on variables names) and show their final exact values.

In addition, for *part a*, during symbolic execution of every statement, if you detect any vulnerabilities, you should report and print the variable's name and the statement related to it. For *part b*, report test inputs which leads to error/vulnerability.