Static Analysis Tools for Detecting Security Vulnerabilities

Shir Landau-Feibish[1] and Noam Rinetzky[2]

[1]*lfshir@gmail.com*
[2]*maon@cs.tau.ac.il*

1 General Information

1.1 Workshop Spirit

In this workshop you will implement and experiment with tools in the area of program analysis and their application for detecting bugs that might lead to security vulnerabilities. The general idea is to take a known result from the program analysis literature, and implement a variant of this result in your own framework. Technically, you will start with a research paper or an open-source project, and work your way to a reasonable (often simplified) implementation of the ideas in the paper or improvements to the existing project.

Simplifying Assumptions The projects listed below provide a lot of freedom and you are *encouraged* to make intelligent choices of simplifying assumptions. Our only requirements are that your simplifying assumptions will be *justifiable*, *consistent*, and *documented*.
Example: assuming that there is no need to deal with pointer arithmetic may be *justifiable* if you find that your technique is useful in a reasonable class of benchmarks despite not handling this feature.

Getting Started The input to your program analysis is going to be a program. All of our projects rely on existing frameworks for the fairly standard problem of parsing a program and transforming it to some intermediate representation that is designed for analysis and transformation. Therefore, the first step in your project should be to create a few example programs and make sure that you can run the basic framework on them to produce the intermediate representation.

Working on the Project For your own sake, you should use a revision control system. You can ask the system staff (*system@cs.tau.ac.il*) to set up an SVN repository for your project.

Like any programming project, this will involve some setup pains and getting lost in code jungles. Use the course forum to get help from the course staff and from your friends if you get stuck. You may share ideas openly in the course forum, you **may not** copy code from each other. We will have zero tolerance towards any form of cheating.

Project Presentation When the project is completed, you will be required to present it in a short evaluation meeting. While you will be running the project for the demonstration, we should be able to run your project independently, so please provide the appropriate documentation for doing so as part of your submission.

1.2 Administration

The projects will be done in groups of 2-4 students. The meetings will be held on Monday 4pm-6pm in Kaplun 324. We will have only a few meetings during the semester. The tentative schedule for the project is:

- 4/March/2013: Project description
- 18/March/2013: Project selection + tool installation.
- 29/April/2013: Progress report (design).
- 17/June/2013: Progress report (initial implementation).
- 1/Sep/2013: project submission.
- Around 10/Sep/2013: Individual evaluation meetings with the different teams.

This tentative schedule is subject to changes. The exact date of the project presentation will be set after the project submission.

Requirements:

- You are required to provide 5 example programs that your analysis can handle.
- You are required to provide a script for running your analyzer from the command line.
- You are required to provide a **short** description of your implementation.
- If your program is for Java program, you are **not required** to handle Java exceptions correctly and may ignore exceptional flow.

2 How to Submit

Projects should be submitted electronically as a single zip file containing a directory with all project files. Directory structure:

- src - source files of your project
- doc - documentation. The directory should contain a README.txt file that explains how to run your project.
- bin - binary files (classes) of your project
- scripts - any scripts for running/testing your project
- examples - example inputs

3 Available Topics

You may choose between the following projects, or suggest one of your own.

Project 1:Buffer Overrun Analysis

Goal Given a C program that performs buffer manipulations, identify statically (at compile time) whether the program may perform array access out of the array bounds.

Main Idea The main idea is to statically track the possible values of integer variables that can be used as array indices in the program. Since integer variables may (conceptually) take unbounded values, and the number of program states may (conceptually) be unbounded, we will employ abstraction to represent the potentially unbounded program state space in a bounded way.

Infrastructure The infrastructure that you will use for this project is the LLVM low-level virtual machine. You will implement the buffer overrun detection as an LLVM analysis pass.

References

- CSSV: towards a realistic tool for statically detecting all buffer overflows in C.
- Buffer overrun detection using linear programming and static analysis

Project 2: Race Detection

Goal Implement a static race detection analysis (e.g., some version of static lockset) as either an LLVM pass or in a Java analysis framework.

Main Idea Static race detection would have to identify when a memory location may be accessed by two threads where at least one of the accesses is a write, and the accesses may happen in parallel. This requires statically capturing two kinds of information: temporal information on what accesses may happen in parallel, and spatial information on what accesses may be accessing the same memory location.

Infrastructure This project can be done in C, based on the LLVM infrastructure, or in Java, based on the Soot infrastructure.

References *General*

- Concurrency attacks

For C:

- RacerX: effective, static detection of race conditions and deadlocks
- LLVM Alias Analysis Infrastructure

For Java:

- Effective static race detection for Java
- The Chord project

Project 3: Pointer Errors Detection

Goal Detect the following programming errors: dereferences to a NULL-valued pointer, access to a pointer after the location that it points to had been freed, and double freeing the same memory address.

Main Idea This project will be done using LLVM. LLVM already has a static analyzer that can catch the above errors in some cases. You should use the analyzer, identify at least 2 cases in which the analyzer misses an error, and extend it to handle these errors too.

References *For C:*

- The static checker of LLVM can be found in LLVM CLang static analyzer

Project 4: Pattern-based Bug Detection

Goal Syntactic detection of bugs by detecting suspicious code patterns.

Main Idea Find coding patterns corresponding to at east 5 possible bugs in Java, write plugins in FindBugs that detect them. Then, write your own analysis in Soot that detect these bugs, and evaluate both analyses over a substantial code base.

References *For Java:*

- FindBugs Homepage