SMT-Based Binary Analysis from the Ground Up

Möbius Strip Reverse Engineering
http://www.msreverseengineering.com

December 2nd, 2014

This course is designed to teach SMT-based binary program analysis from the ground up. Students shall be involved in the construction of a minimal, yet fully-working SMT-based program analysis framework in Python for X86 assembly language. Then, the framework will be put to use in automating reverse engineering tasks, including developing a compiler for return-oriented programming (ROP).

The course consists of lecture material, an implementation manual, source code to be completed by the student, and exercises of the following varieties:

- Hand-written
- Programming
- Using SMT solvers
- Automating reverse engineering tasks

The intended audience is low-level people comfortable programming in Python. For all programming exercises in the course, the uninteresting parts of the code are provided, leaving the most relevant parts for student implementation exercises. Comprehensive test suites ensure that the students' implementions work correctly.

Contents

1 Introduction to Analyzing Programs

As a warm-up for the types of programming we will be doing throughout the course, students are introduced to how programs are represented within program analysis tools, the typical structure of analysis algorithms, and typical algorithms such as interpreters, type-checkers, and translators. We illustrate the concepts using two tiny programming languages. Students code an interpreter, type-checker, bytecode interpreter, and bytecode translator.

2 Computational Logic

This section introduces the students to what SAT and SMT solvers are, how to use them, and how they work internally.

2.1 Propositional Logic and the SAT Problem

Propositional logic is explained by analogy to the C programming language. Students shall implement propositional logic, several algorithms, and a brute-force SAT solver.

2.2 SMT Solvers

Students are introduced to the concept of SMT solvers and their capabilities, with a brief glimpse of how they work. Students will solve a series of exercises with the Z3 SMT solver.

2.3 SMT Bit-Vector Theory

We introduce the bit-vector theory, used to model machine integer operations. Students shall implement translations from bit-vector operations into propositional logic, and hence a basic SMT solver.

2.4 DPLL SAT Solvers

We explain and implement the DPLL algorithm for the SAT problem, which has been crucial to the modern SAT revolution.

2.5 SMT Array Theory

The array theory is used to model memory accesses within SMT formulas. We explain what it is, how it works, and how to use it.

3 X86 Assembly and Disassembly

We discuss how X86 instructions are encoded, and write a library for representing, encoding, and decoding X86 instructions.

4 Intermediate Representations (IRs)

Intermediate representations allow program analysis algorithms to be written generically, in a cross-platform fashion, a vast simplification over writing analyses directly upon assembly-language instructions. We discuss options in the design of IRs, and then describe the IR for our framework.

4.1 IR Translator Synthesis

Implementing IR translators is tedious business, so we construct a tool that automates most of the work for us. Then, we implement the IR translator with the help of this tool. At the conclusion of this section, students have a framework to disassemble X86 instructions, convert them into an IR, and interface them with the Z3 SMT solver.

4.2 SSA Form

This brief section explains why Static Single Assignment (SSA) Form is necessary, and how to implement a SSA translator. Now, the framework is ready for use.

5 Automating Tasks in Reverse Engineering

We give a number of exercises involving applying the newly-constructed framework to reverse engineering problems, such as input generation.

5.1 ROP Compilers

The course concludes by having the students complete the implementation for a ROP compiler inspired by Q from CMU, and ROPC by pa_kt.