Week 5

Help

Static Analysis and Symbolic Execution for Security

Last week we looked at how security fits into the various activities of the development process. Up to this point we have generally looked at particular software flaws and bugs and how to avoid them through good design and implementation practices.

This week we will look at the testing and validation phases of software development, and in particular how automated tools can assist developers and testers in finding important security bugs. We will focus on two technologies: *static analysis*, and *symbolic execution.*

Learning Objectives

After the completion of this week's material, you will:

- Know what static analysis (SA) and symbolic execution (SE) are, how they compare, and why they are hard
- Understand the basics of each approach
- Understand how to improve the precision and scalability of each approach
- Be aware of several commercial and open-source SA and SE tools, and how they compare

Video Lectures

This week we cover two code analysis technologies: static analysis, and symbolic execution. The latter is often used as the core of a penetration testing technology called *whitebox* fuzz testing; we will discuss other fuzz testing techniques next week.

Static Analysis

- Static Analysis: Introduction part 1 (5:10)
- Static Analysis: Introduction part 2 (8:12)
- Flow Analysis (8:50)
- Flow Analysis: Adding Sensitivity (8:57)
- Context Sensitive Analysis (8:53)
- Flow Analysis: Scaling it up to a Complete Language and Problem Set (11:40)
- Static Analysis: Challenges and Variations (8:01)

Symbolic Execution

- Introducing Symbolic Execution (10:52)
- Symbolic Execution: A Little History (3:05)
- Basic Symbolic Execution (14:17)
- Symbolic Execution as Search, and the Rise of Solvers (12:45)
- Symbolic Execution Systems (8:26)

Break out: Interview with Andy Chou

In August 2014, Mike had the pleasure of interviewing Andy Chou. In 2003, Andy co-founded Coverity, a company that develops static analysis (and other) tools for finding software defects. Coverity's products started from Andy's Ph.D. research at Stanford, and grew as the company grew; the organization was at 300 employees when it was acquired by SynopSys in 2014. In this interview we discussed principles and practice of static analysis, and the path towards making a practical tool that is now in wide use. Like last week, the interview is *optional* from an assessment perspective -- there will no quiz questions on it. We hope you find it interesting!

Mike Hicks interviews Andy Chou (32:31). Highlights, indexed by time:

- start - Intro to static analysis, and different sorts of analysis
- 5:52 - Static analysis's place in secure development practice; tradeoffs
- 9:33 - Coverity Scan open source analysis: what it is, and its impact
- 11:07 - Static analysis challenges with particular languages
- 13:11 - Dynamic analysis: a complement to static analysis
- 17:35 - The process of choosing which analyses go into your tools

- 21:16 - How the market for static analysis has matured over the last 15 years
- 25:08 - More challenges for use and development of static analysis; looking ahead
- 29:19 - Free service Code Spotter; eating your own dogfood; closing thoughts

---

Supplemental Links

Here is some additional reading material, if you are interested in learning more.

- A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World - An article by the Coverity team on their experience writing and deploying a commercial static analysis tool
- All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (but might have been afraid to ask) - formal presentation of these analyses and their variations

Here are some links to static analysis and symbolic execution tools you might find interesting.

Static analyzers

- Coverity's free Code Spotter and Coverity Scan initiatives
- Clang Analyzer - A static analyzer built on the LLVM framework for C and C++ programs
- Joern - A static analyzer for C and C++ programs
- FindBugs - A static analyzer for Java programs

Symbolic executors

- KLEE - A symbolic executor for LLVM bitcode (primarily supporting C and C++)
- Otter - A source-level symbolic executor for C code
- Pex - A symbolic executor for .NET
- SAGE - A 10-minute video on the SAGE whitebox fuzz tester (based on symbolic execution)

SMT solvers (used by various of the above tools)

- Z3 - developed at Microsoft Research
- Yices - developed at SRI

- STP - developed by Vijay Ganesh, now @ Waterloo
- CVC3 - developed primarily at NYU

Quiz

The quiz for this week covers all of the material for this week. You must submit the quiz no later than the very end of week 6. You will have three attempts to complete the quiz, at two hours per attempt. It consists of 15 questions, and if you are well versed in the material it should take no more than 30 minutes (but longer if you have to go back and look things up, obviously).

Project

For this week's project you will get your feet wet using a symbolic execution tool called KLEE, and comparing its performance to a black box fuzzing tool called radamsa (which we will learn more about next week). When you finish it, take the on-line assessment, due at 8am EST on December 8.

Created Wed 9 Apr 2014 6:16 AM PDT

Last Modified Fri 21 Nov 2014 7:14 AM PST