angr – The Next Generation of Binary Analysis

Length

3 hours in total.

Description

This tutorial will introduce a binary analysis platform, angr, to attendees, and demonstrate ways to perform some common tasks in binary analysis with angr, including dynamic symbolic execution, static CFG recovery and querying, extracting data dependency information, automatically generating ROP chains, etc.

Prerequisites

The tutorial is designed for people who has basic understandings of or some experience on program analysis and/or binary reversing, but are new to angr. This tutorial assumes some basic Python programming skills and the basic ability to read X86/AMD64 assembly snippets. Students must bring a personal laptop that can connect to the internet.

Textbook

None

Speakers

Ruoyu (Fish) Wang, UC Santa Barbara
Yan Shositaishvili, Arizona State University

Schedule

1. Binary analysis 101 (20 min)
   - Goals of binary analysis
   - Introduction to binary analysis techniques

2. Introduction to angr (1 hour 10 min)
   - Fundamentals of angr
   - Installing angr
   - Using angr from a Python interactive interface
   - Loading a binary with angr
   - Loading a binary with libraries
   - Basic static analysis: CFG recovery
• Introduction to symbolic execution
• Basic symbolic execution: a symbolic execution demo
• Q&A

3. **Symbolic execution in a nutshell (50 min)**
• Exploring a binary program
• Customizing the exploration with exploration techniques
• Program exploration in the GUI
• Using symbolic execution to solve CTF challenges
• Effectively using symbolic execution to audit functions
• Automatic ROP chain generation
• Q&A

4. **Static analysis (40 min)**
• CFG recovery
• Querying the CFG
• Exploring CFG in the GUI
• Obtaining data dependence information
• Variable recovery and basic type inference
• Writing your own data-flow analysis routine
• Q&A

**Things we will briefly cover during the tutorial (but we won’t have time to go into details):**
• What can be solved with symbolic execution? What is symbolic execution good for?
• Dealing with hard transformation routines
• Performing symbolic execution efficiently