A Visual Approach to Symbolic Execution

Nicholas Pfister
Santa Barbara City College ▪ UCSB SecLab

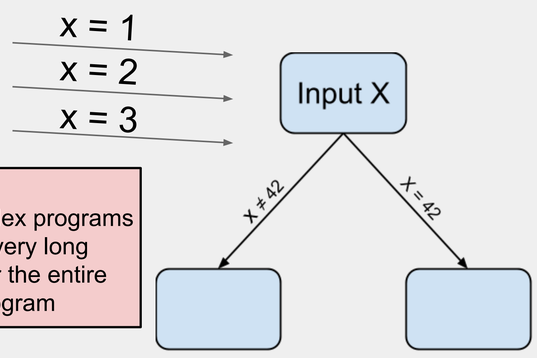
Abstract

In the modern world we live in, we are surrounded by devices that use software in some way, shape, or form. Many of these devices such as power plants, airplanes, and medical equipment are safety critical and could cause significant harm if the software controlling them was to fail. Hence, we can use a type of static software analysis called symbolic execution in order to determine the full scope of operation and detect any vulnerabilities of said software. In order to fully understand the results of this analysis it would be incredibly useful to have a means of visualizing the scope of a given program as a control flow graph (CFG), or collection of code blocks with control flow transitions. Thus we present a versatile web-based visual platform for displaying CFGs, as well as some other valuable information during symbolic execution, that is capable of running on any given web-browser. This interface is meant to be user-friendly and interactive in order to give the user a very clear image of the results of symbolic execution and the scope of a particular program. With this powerful tool, computer security professionals will be able to better assess the vulnerabilities of software and therefore maintain the integrity of safety-critical devices.

Symbolic Execution

2 Types of Software Analysis

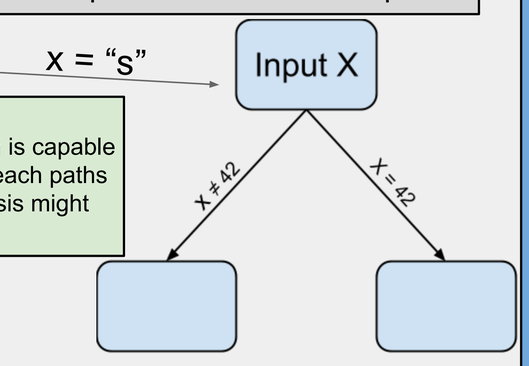
Dynamic Analysis runs a program many times, sending new inputs every time to try and find every possible path in a program



The Problem
In more complex programs it may take a very long time to test for the entire scope of a program

Static Analysis examines the source code of a program without executing it

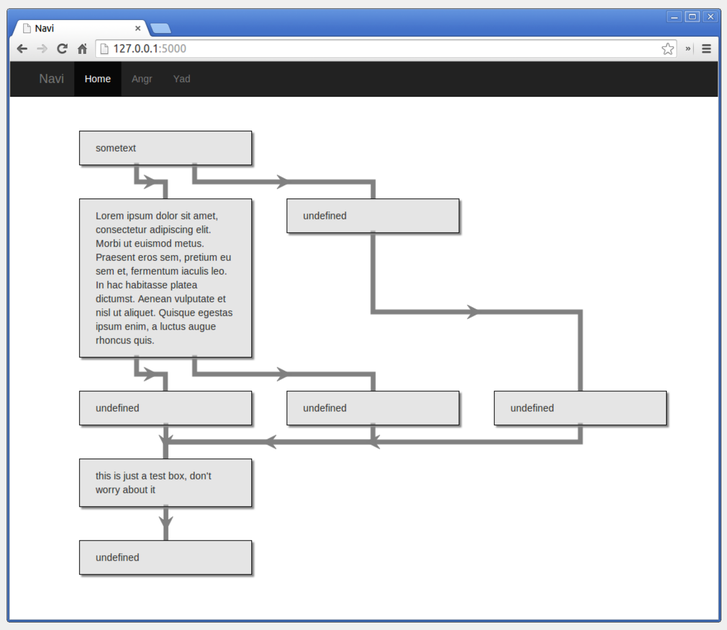
Symbolic Execution is a type of static analysis that runs a "virtual" version of the program and inputs abstract values. It then uses constraint solvers to determine mathematically the value that the input must equal to follow a certain path



The Advantage
Symbolic execution is capable of finding hard-to-reach paths that dynamic analysis might not find

The Interface

Sample Output
(dummy text)



- Launches in Google Chrome, Internet Explorer, Mozilla Firefox, and most other common browsers
- Links directly to symbolic execution system to obtain program analysis results
- Boxes hold useful information about the function they represent
- Lines represent the possible paths a program might take
- Graph generation is completely automated to require minimum effort on the user's behalf

Usability

The **usability** of our software is highly dependent on the loading speed of the webpage and the clear, logical layout of the visualizations

Average loading time (20 trials)
5.15ms*

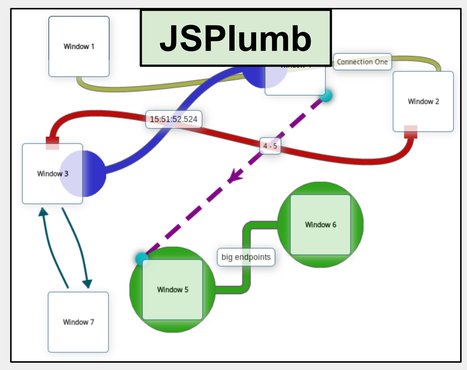
Layout
- Connections between object are clearly defined and easy to understand
- Boxes resize automatically so the user can easily discern important function information

*average loading time of a simple graph with 8 functions and 3 paths

Coding

Languages Used
- **HTML/CSS** - used to determine the webpage layout, as well as the styling of the graph
- **JavaScript/jQuery** - used to devise the logic behind the graphing algorithms
- **Python** - used to communicate with and obtain results from the symbolic execution system

Important Libraries
- **JSPlumb** - JavaScript library used to create logical connections and add interactivity to the graph
- **Tornado Web Server** - Python library used to launch the website directly from Python code



Current/Future Plans

This software was implemented by the SecLab team at DEFCON 22, a hacking convention in Las Vegas, to visualize CFGs

Because this is part of an ongoing symbolic execution project, our software will continue to be improved to handle larger and more complex graphs

Mentors
Fish Wang, Christophe Hauser, Yan Shoshitaishvili
Faculty Advisor
Christopher Kruegel ▪ Department of Computer Science

Funding
Defense Advanced Research Projects Agency (DARPA)