Introduction to Symbolic Execution with angr

h4tum Presentation

Fabian Kilger

Technische Universität München

2025-05-21

Overview

- Use-Cases
- Symbolic Execution
- `angr`
- Basic Example #1
- Limitations of Symbolic Execution
- Example #2 - Path Explosion
- Example #3 - Symbolic Pointers

Use-Cases

- In CTFs, especially reversing, we often have **Code Snippets** that perform some **logic**
  - $\Rightarrow$ We often need to **manually extract** the logic and re-implement it or solve it (e.g. with **Z3**)
- What if we could do that **automatically**?
- This is what **Symbolic Execution** is used for in CTFs

Symbolic and Concolic Execution

- Work with symbolic instead of concrete values
- Symbolic values are formulas

Symbolic and Concolic Execution

- Work with symbolic instead of concrete values
- Symbolic values are formulas
- Conditions on these symbolic values result in SMT formulas
- Branch on conditions into multiple **state**s, appending branch condition to **state**.

Symbolic and Concolic Execution

- Work with symbolic instead of concrete values
- Symbolic values are formulas
- Conditions on these symbolic values result in SMT formulas
- Branch on conditions into multiple **state**s, appending branch condition to **state**.
- Problem: Not everything is or can be modeled and not everything can be dealt with efficiently

Symbolic and Concolic Execution

- Work with symbolic instead of concrete values
- Symbolic values are formulas
- Conditions on these symbolic values result in SMT formulas
- Branch on conditions into multiple **state**s, appending branch condition to **state**.
- Problem: Not everything is or can be modeled and not everything can be dealt with efficiently
  e.g. symbolic pointers, hash computations
- Solution – **concolic** execution: Mix symbolic and concrete execution

Symbolic and Concolic Execution

- Work with symbolic instead of concrete values
- Symbolic values are formulas
- Conditions on these symbolic values result in SMT formulas
- Branch on conditions into multiple **state**s, appending branch condition to **state**.
- Problem: Not everything is or can be modeled and not everything can be dealt with efficiently
  e.g. symbolic pointers, hash computations
- Solution – **concolic** execution: Mix symbolic and concrete execution
- In the context of symbolic formulas, concrete values are *constants*.

Intro to angr

- **Binary Analysis** framework with support for **Symbolic Execution**
- Nowadays also has a somewhat buggy graphical UI and decompiler.
- Install in venv via `pip install angr==9.2.154`
- Important concepts:
  - **Project** is the base class that manages all analysis and binary content
  - **State** refers to an intermediate or final state of the execution, manages information about a single path execution.
  - **SimulationManager** manages **state**s and execution of these states
  - **SimulationManager.explore** performs a search for a state

Basic Example #1

- Use template code `angr_basic.py` and binary `lineq1`
- Challenge was part of European Cyber Week CTF 2024
- **SimulationManager.explore** supports two important arguments:
  - **`find`**: Search for a path/state that fulfills a condition
  - **`avoid`**: Avoid paths/states that fulfill a condition
- Briefly open the binary in Ghidra (or similar tool to your choosing) to figure out what conditions you may want to choose.

Limitations of Symbolic Execution

- Path Explosion; Example - how many paths?

```
int foo() {
  for (int i = 0; i < 50; ++i) {
    if (getc(stdin) % 2)  puts("uneven");
    else                  puts("even");
  }
}
```

Limitations of Symbolic Execution

- Path Explosion; Example - how many paths?

```
int foo() {
  for (int i = 0; i < 50; ++i) {
    if (getc(stdin) % 2)  puts("uneven");
    else                  puts("even");
  }
}
```

- Constraints too complex
  - Constraints are solved with **SMT** solvers (commonly **Z3**)
  - Not all problems can be solved efficiently (but in CTFs most can)

Limitations of Symbolic Execution

- Path Explosion; Example - how many paths?

```
int foo() {
  for (int i = 0; i < 50; ++i) {
    if (getc(stdin) % 2)  puts("uneven");
    else                  puts("even");
  }
}
```

- Constraints too complex
  - Constraints are solved with **SMT** solvers (commonly **Z3**)
  - Not all problems can be solved efficiently (but in CTFs most can)
- Symbolic pointers:

```
int foo = arr[symbolic_index];
int (*fooptr)() = funcptr_arr[symbolic_index];
```

Limitations of Symbolic Execution

- Path Explosion; Example - how many paths?

```
int foo() {
  for (int i = 0; i < 50; ++i) {
    if (getc(stdin) % 2)  puts("uneven");
    else                  puts("even");
  }
}
```

- Constraints too complex
  - Constraints are solved with **SMT** solvers (commonly **Z3**)
  - Not all problems can be solved efficiently (but in CTFs most can)
- Symbolic pointers:

```
int foo = arr[symbolic_index];
int (*fooptr)() = funcptr_arr[symbolic_index];
```

- Note: CTF challenges use such things to prevent **plain angr** solutions.

Basic Example #2 - Path Explosion

- Use template code `angr_basic.py` and binary `path_explosion`
- Use **`avoid`** to restrict the paths executed and lead the symbolic execution!
- Again, open the binary in a tool of your choosing to figure out what you need.

Basic Example #3 - Symbolic Pointers

- Use template code `angr_basic.py` and binary `lineq2`
- Challenge was part of European Cyber Week CTF 2024
- This time, mathematical operations are obfuscated with symbolic hlarray accesses!
- Again, open the binary in a tool of your choosing to figure out what you need.
- Make sure to check out the **cheatsheet**.