Introduction to Z3

h4tum Presentation

Fabian Kilger

Technische Universität München

2025-02-26

Contents

- What is a **SMT** solver (and therefore Z3)?
- How to use Z3
- Hands-On Examples from real CTF challenges

What is a SMT Solver?

- Satisfiability Modulo Theories is like SAT, but with non-boolean values
- Encompasses most mathematical formulas we can think of.
- SMT solvers try to solve the following problems:
  - Find a solution to a constraint system
  - Find the maximum/minimum of a formula in a constraint system
  - Simplify formulas
- Keep in mind: These problems are NP-hard in general.
- These solvers can solve many problems, but often not optimal (in terms of performance)

What is Z3

- Z3 is a ''theorem prover'' by Microsoft Research
- Bundles several solvers including **SAT** and **SMT**
- State-of-the-art tool used by security researchers and CTF-players
- Can be relevant for any CTF category
- Includes bindings for 10 programming languages.
- Python API is easy-to-use. `import z3`

How to use Z3? - The Solver object

- The base object for solving is the Solver object.
- Create solver: `s = z3.Solver()`
- Add constraints: `s.add(constraint)`
- Check for satisfiability: `s.check()`
- Get model (after check): `m = s.model()`
- Query model: `m[var].as_long()`

Creating formulas in Z3

- Creating variables:
  - Arbitrary-precision integer: `z3.Int("v")`
  - Bitvector: `z3.BitVec("v", bitsize)`
- Creating formulas:
  - With normal python operators: v * 0x42 == 1
    Be aware: Normal operators are signed versions
  - For unsigned: `z3.{ULT,ULT,ULE,UGT,UGE,UDiv,URem,LShR}`
  - Further logical operators:
    - If-Then-Else: `z3.If(cond, then, else)`
    - Variables must be different: `z3.Distinct(list_of_vars)`
    - $A \implies B$: `z3.Implies(A,B)`

Common Pitfalls

- Bitvectors are big-endian
- Mix up signed and unsigned operations (especially bitshifts)
- Query model before sat check

Chall1 - ROT128

- crypto Challenge from HTB Cyber Apocalypse CTF 2024
- Official difficulty: insane
- We are asked to solve the following problem:

Given $h1$ and $h2$, find $s1, s2, r1, r2, r3, r4$ s.t.:

$$\mathrm{ROL}(s1, r1) \oplus \mathrm{ROL}(s2, r2) == h1$$
$$\mathrm{ROL}(s1, r3) \oplus \mathrm{ROL}(s2, r4) == h2$$

- Using some mathematical reasoning, we can solve this if we can solve:

Given x, find y, c s.t.:

$$y \oplus \mathrm{ROL}(y, c) == x$$

- Task: Find a solution with Z3 - template code given

Chall2 - dreamer

- pwn/misc Challenge from GPN CTF 2024
- Points: 313
- We can execute 4 bytes of arbitrary shellcode followed by 100 bytes of shellcode generated via a bad **PRNG**
- We created a payload with 9 bytes of shellcode
  $\rightarrow$ we need to make the **PRNG** generate 5 specific bytes
- Task: Find a solution with Z3 - template code given
  - `test.py` for testing the implementation
  - `exploit.py` to get the flag
  - Implement the function **`get_seed_for_sequence`**

Chall3 - archventure time

- rev Challenge from GPN CTF 2024
- Points: 135
- License key checker with basically 5 binaries of different architecture (main + 4 sub-binaries)
- Each binary checks some constraints on the license key
- Reversing is already performed and text-descriptions given
- License is a string of form XXXXX-XXXXX-XXXXX-XXXXX where Xs are variables
- Task: Model all constraints with Z3 and retrieve the license key
- Note: template code given again