SAT Solving

Contents

1 Introduction

Microsoft Research's Z3 theorem prover [1] can be used to reverse the keystream generated by a stream cipher function by satisfying a series of boolean constraints.

1.1 Goals

- Express RULE's bit to bit triplet association as a disjunctive normal form (DNF)
- Apply the DNF constraints to each bit in the representation of the output from `super_solver.next()`
- Extract the original seed value used to generate the keystream by solving the boolean constraints

2 Methodology

2.1 RULE's Disjunctive Normal Form (DNF)

The hash function from `super_solver.py` builds the next keystream by associating `RULE` to bit triplets from the current keystream:

```
...
RULE = [86 >> i & 1 for i in range(8)]
...
y |= RULE[(x >> i) & 7] << i
...
```

The following bit triplets are associated with `0b1`:

Next Bit	Previous Triplet
1	001
	010
	100
	110

Expressing this relationship as a DNF, which always returns as `True` with the associated bits at consecutive indexes `p`, `q`, and `r`:

```
(¬p ∧ ¬q ∧ r) ∨ (¬p ∧ q ∧ ¬r) ∨ (p ∧ ¬q ∧ ¬r) ∨ (p ∧ q ∧ ¬r)
```

The DNF was written into `dnf_bit_1()`, expressed in `z3.BoolRef` form:

```
Or(
   And(Not(p), Not(q), r),
   And(Not(p), q, Not(r)),
   And(p, Not(q), Not(r)),
   And(p, q, Not(r))
)
```

2.2 Applying the SAT Constraints

In order to express the output constraints in the `super_cipher.next()` function, the DNF associating bit triplets to the output bit has to be applied to each bit index in the representation of the output.

The current keystream's binary values expressed as boolean are added as further constraints to the solver.

As it is known that the `next()` hash function produces valid output by following these constraints, the model is checked instead for unsatisfiability (`z3.unsat`) in order to pick up any semantic errors in the expression of DNF and constraints.

The Z3 solver is then tasked to build the model of the previous keystream that satisfies all constraints imposed.

2.3 Extracting the Seed

To extract the seed from the keystream, the initial 256-bit keystream block is fed as a list of `bool` into the `sat_solve_prev()` function and iterated 128 (`N//2`) times, the same number of times the seed was iterated to get the initial keystream block. The final `bool` list is then converted to `bin`, then `int`, then bytes before being decoded into a `utf-8` string:

```
INS{Rule86_is_W0lfr4m_Cha0s}
```

3 Conclusion

Using Z3 to iteratively solve sets of DNF boolean constraints is another viable (but computationally slower) approach to extracting the seed for the hash function in `super_cipher.py`.

4 Discussion

SAT (boolean satisfiability problem) is a constraints-based approach to solving problems. By using DNF to represent restrictions to possible solutions, we can make use of Z3 to check if the set of constraints are satisfiable, and if so, present valid assignments.

In this instance of reversing the `next()` function to obtain the seed, the satisfiability of the applied restrictions is already known from deconstructing the function in the previous assignment. The benefits of the SAT approach are more apparent in problems that have unknown satisfiability. If the problems can be restricted to expressions using the disjunctive normal form, their satisfiability can then be checked in linear time. Full DNF expressions can be checked in constant time. The trade-off is from possibly having to use exponential resources to express the logic formulas in the SAT problem using DNF.

5 Source Code

solution.py

The Z3 Python package is required to execute `solution.py`:

```
pip install z3-solver
```

The `keystream` file should be present in the same directory as `solution.py`.

---

1 https://github.com/Z3Prover/z3