Using SAT solvers for security related problems

Pierre Bourdon

delroth@lse.epita.fr
`http://lse.epita.fr`

February 8, 2013

Quick example

- You are trying to analyze a program to understand how it encrypts message and how to decrypt these messages
- The program contains only the encryption algorithm, no decryption code
- You possess an encrypted message and the encryption key
- How to decrypt that message?

Quick Example


```python
# Encrypts dw1 and dw2 (32 bits) with the constant key 0x63737265
def encrypt(dw1, dw2):
    sum = 0
    for i in range(32):
        dw1 += (sum + 0x63737265) ^ (dw2 + ((dw2 << 4) ^ (dw2 >> 5)))
        sum -= 0x61C88647
        dw2 += (sum + 0x63737265) ^ (dw1 + ((dw1 << 4) ^ (dw1 >> 5)))
    return dw1, dw2
```

Quick Example

- You might not recognize the algorithm at first
- Inverting this encryption algorithm to get the decryption algorithm is not trivial
- Let's use some magic! PySolver to the rescue

Quick Example

```python
problem = pysolver.Problem()
dw1 = dw1_in = pysolver.Int(problem, 32)
dw2 = dw2_in = pysolver.Int(problem, 32)

dw1, dw2 = encrypt(dw1, dw2)

dw1.must_be(0x131af1be)
dw2.must_be(0x4bb34049)

problem.solve()
print(hex(dw1_in.model), hex(dw2_in.model))
# Prints 0x615f7a6e, 0x645f6572
```

Boolean Satisfiability Problem

Finding a set of values for boolean variables that satisfy a formula.

$$SAT((a \vee b) \wedge (\neg a \vee b)) = \{\neg a, b\}$$

$$SAT(a \wedge \neg a) = UNSAT$$

Hard to solve

- NP-complete problem: no polynomial algorithm exists to solve SAT
- Lots of applications in constraint solving
- People wrote programs called SAT solvers to find solution to the SAT problem
- Very optimized, "fast enough" for most cases but some formulas need a very long time to solve or are reported as false negatives
- No false positives

Applications to security

- A bit is a boolean variable, an integer is a set of bits
- Most operations on integers can be represented as a logic formula operating on the bits
- Write a big formula representing your encryption function, add clauses to "force" the output to some values, use SAT to find satisfying input values
- Also some applications in static analysis (finding input values which will take a certain code path, etc.)

DIMACS and CNF

- SAT solvers use a common input format: DIMACS
- DIMACS represents a CNF boolean formula
- Conjunctive Normal Form, product of boolean sums
- Variables are represented by a simple integer

$$(a \vee \neg b) \wedge (\neg a \vee b \vee \neg c)$$

Forcing an output value

- Let's start with a simple function that checks if a number is equal to a constant
- The formula must be satisfied if and only if each input bit has the same value as our constant
- $b \Leftrightarrow 1 \equiv b$
- $b \Leftrightarrow 0 \equiv \neg b$
- Example: we want to check if a 4 bits number is equal to 11
- $b_0 \wedge \neg b_1 \wedge b_2 \wedge b_3$

AND between two values

- AND between two bits, repeated for every bit in the numbers
- $c_i \Leftrightarrow a_i \wedge b_i$
- $\equiv (a_i \vee \neg c_i) \wedge (b_i \vee \neg c_i) \wedge (c_i \vee \neg a_i \vee \neg b_i)$

ADD between two values

- A bit more complex: we can't just ADD two bits together without keeping a carry
- We'll do it exactly like it's done in circuit design: chained 1 bit adders
- A 1 bit adder has three inputs: a_i, b_i, c_i and two outputs: r_i, c_{i+1}
- Hard to represent as CNF clauses "manually", we can use Sage to convert any boolean formula to (potentially unoptimized) CNF

Easy CNF generation with Pysolver

- Python library to easily generate CNF from "natural" code
- Interfaces with CryptoMiniSAT, a fast and efficient SAT solver
- About 200 lines of Python, improving when I need new features
- `http://code.delroth.net/pysolver`

TODO

- Variable shifts: implement a simple barrel shifter
- Take more advantage of CryptoMiniSAT features (XOR clauses)
- Implement mappings: optimize with a Karnaugh map to minimize the number of clauses

Questions?

- `@delroth_`
- `http://code.delroth.net/pysolver`