Python and SMT Programming Test

Vector Informatik

1 Setup

We have set you up an account on a Linux VM to do some programming tasks. The account details are:

- Host: `10.44.35.3`
- User: `dimitris`
- Password: `password123`

The Windows laptop you've been provided with has a program called `MobaXterm` installed – this application is an easy-to-use way to get SSH access onto a remote Linux box, along with X11 forward.

On the Linux VM, we have provided you with a following:

- Sublime text – `subl`
- PyCharm – `pycharm.sh`
- Binaries for the Z3 solver
- Binaries for the STP solver

You are encouraged to use `vim` as your editor, but this is not mandatory. Everyone in the team works in `vim`, so not using `vim` may make your life hard later on ...

1.1 Challenge 0.01

You should verify the following:

- You can connect to the VM
- You can start `subl` or `PyCharm` (or `vim`!)
- That you can use the Python bindings for Z3:

```
1  import z3
2  print z3.Z3_get_full_version()
```

It is *strongly* recommended that you create yourself a local git repository and that you make separate files for all of the challenges – remember, commit early and commit often!

2 About Z3

Z3 is an open-source SMT solver written by Microsoft research. It comes with an *very* friendly Python API!

For this test, we expect you to use Z3's Python API and *not* by writing SMT2LIB files directly!

Here are some useful links:

1. `https://github.com/Z3Prover/z3/tree/master/examples/python`
2. `https://ericpony.github.io/z3py-tutorial/guide-examples.htm`
3. `https://yurichev.com/writings/SAT_SMT_by_example.pdf`
4. `https://stackoverflow.com/questions/tagged/z3py`
5. `https://theory.stanford.edu/~nikolaj/programmingz3.html`

I would recommend taking a look at the `z3py` tutorials (Link 2) to start with *and trying these out* to make yourself familiar. I would then take a look Section 2.2 in Link 3.

2.1 Challenge 0.01

For your first challenge, try to solve these:

- Print all of the numbers between -10 and 10 that sum to equal 5 – you need to think about how you can get multiple answers out of the solver! Hint: you can call `solver.check()` multiple times!
- Use Z3 to stimulate C-style overflow – what property are you going to use to make this happen?

3 Implement basic reachability

A *transition system* is a tuple $A = (Q, q_0, T)$, where:

- Q is a set of states,
- $q^{init} \in Q$ is the initial state,
- $T \subseteq Q \times Q$ is a transition relation.

Let $n \geq 0$. An *n-path in A* is a sequence of states $(q_0, \ldots, q_n)$ such that $(q_i, q_{i+1}) \in T$ for all $i \in \{0, \ldots, n-1\}$.

We say that a state $q \in Q$ is *n-reachable* iff there exists an n-path $(q_0, \ldots, q_n)$ such that $q_0 = q^{init}$ and $q_n = q$. A state $q \in Q$ is *reachable* iff there exists $n \geq 0$ such that q is n-reachable.

The task is to implement a bounded model checker for reachability in transition systems. The implementation should rely on a translation of the n-reachability problem into SMT. Additionally, the tool should allow for testing if a given state q is reachable, i.e. if there exists $n \geq 0$ such that q is n-reachable. It is okay if the algorithm does not terminate when q is not reachable.

The translation consists of encoding all the possible paths of length n (without enumerating them). Then, to perform the reachability test the formula encoding the paths (f_T) needs to be constrained with the encoding of the initial state (f_{init}) and the encoding of the final state (f_{final}). Therefore, the translation of the problem into SMT should be encoded as a formula in the following form:

$$F = f_{init} \wedge f_T \wedge f_{final}.$$

The translation has the following property: F is satisfiable iff the state is n-reachable.

When the formula is satisfiable you should take the model for the formula and extract the witness for the reachability of the state.

Here are some simple examples of transition systems that can be used for testing:

```
def example_1():
    init = 1
    trans = [(1, 2), (2, 3)]
    final = 3
    return (init, trans, final)

def example_2():
    init = 1
    trans = [(1, 2), (3, 4), (4, 5)]
    final = 5
    return (init, trans, final)
```

The final version of the tool should take an input file (or files) with the description of the transition system and the reachability properties to be tested.

4 Reverse engineer an algorithm

You are given the following snippet of C code, as well as the expected output string of `WIEXST5$`:

```
/* algorithm */
for ( i = 0; i <= 7; ++i )
{
    output_string[i] += output_string[i] % 5;
    if ( output_string[i] > 120 )
        output_string[i] = -100 - output_string[i];
}

/* assert */
strcmp("WIEXST5$", output_string)
```

You should:

- Implement a simple encoder for the above algorithm in Python
- Implement an *decoder* using Z3 that will provide you with the plain text string for a given ciphertext
- Verify that: `ciphertext == enc_str(dec_str(ciphertext))`

5 Implement a password cracker

This is a very similar challenge to the previous one, but where you do not know the length of the input:

- `https://github.com/sectalks/sectalks/blob/master/ctf-solutions/0x00/blinken/writeup`

5.1 Brief

This challenge starts by requiring you to break an insecure encryption algorithm implemented in Javascript:

- The user is prompted for a username and a password
- The username is iterated over:
  - The ordinal value of each character is found and multiplied together to produce a key, we'll call it the user_product
  - The ordinal values of each character are summed to produce a key, the user_sum
  - user_sum += ord(username[i])
- The password is also iterated over to produce a passwd_product and passwd_sum, but for the passwd_sum the square of the ordinal value is used - passwd_sum += ord(passwd[i]*ord(passwd[i])

To break the encryption, you must find what username string(s) produce the provided user_sum and user_product values, and what passwords strings produce the provided passwd_sum and passwd_product values. These are:

- user_sum = 1537
- user_prod = 1172188274400
- passwd_sum = 393644
- passwd_prod = 1842055687879732800