Mon 30 May 2016

misc200 - Setting the Table

Posted by Ingmar ‘doskop’ Steen under HiTB 2016 CTF with tag(s) CTF

Introduction

> Presentation of a meal is an important part of the restaurant experience. Show your table setting skills here: 145.111.225.61:37373. If you feel lost, you can find the basic table setting manual here: download

TL;DR

Disclaimer

I didn’t actually solve this during the CTF: I looked at the challenge and somehow forgot it existed.

Analysis

The source code we have access to starts off with a huge hint: `Soup Meat Tea (SMT) (TM)`. SMT in this case refers to a solver which uses satisfiability modulo theories, also known as constraint-based programming.

Looking at the source code we can see an initial state being defined and then some input being iterated which is used to manipulate the state. It can be reduced to this:

```
uint32_t state = 42;
for (uint32_t idx = 0; idx < 32; idx++)
    state = ((state + input[idx]) * 3294782) ^ 3159238819
```

Then it shows a textual representation of your input and finally it compares the state to `0xde11c105` and if that condition is met it prints the flag:

```
if(state == 0xde11c105) {
    system("/bin/cat flag.txt");
}
```

Exploitation

To find a solution, I used the z3 theorem solver. I started by importing z3, instantiating the solver and setting up a variable for the initial state which is 42:

```
from z3 import *

# Create new z3 solver instance.
solver = Solver()

# Create the initial state, which starts at 42.
state = BitVec('i', 32)
solver.add(state == 42)
```

Next, I’m going to create 32 bitvectors and constrain them to the input the program expects (actually, I’ll restrict it even more as I really dislike having \t or \n in input):

```python
values = []
for i in range(32):
    # Create next byte of input.
    value = BitVec('i%d' % i, 32)

    # Define restrictions if you want the dishes to render.
    solver.add(value > 0, value < 9)

    # Add the variable to the tracking array.
    values.append(value)

    # Update the global state.
    state = ((state + value) * 3294782) ^ 3159238819
```

Next, I constrain the final state to the value we need it to be:

```python
# Add restriction for the final state.
solver.add(state == 0xde11c105)
```

Now that the model is complete, I let z3 check if it’s satisfiable:

```python
assert solver.check() == sat
```

Finally, I get the model from the solver and print a bash compatible representation of the input values:

```python
model = solver.model()
print '$' + repr(b''.join(chr(model[v].as_long()) for v in values))
```

Running this will print:

```
$'\x04\x03\x02\x03\x07\x03\x03\x07\x04\x07\x08\x08\x02\x04\x07\x04\x04\x02\x02\x04\x04\x08\x0
```

Sending that to the server should present me with the flag. That is, it would have assuming I thought of doing all this during the actual CTF.