Iteration-1 Report

Symbolic Execution Of Smart Contracts Using Manticore

Advanced Topics in Software Engineering

CSE 6324

GitHub link: https://github.com/PavanKumarChaparla/ADV-TOPICS-IN-SE-CSE-6324

Team 8:

Pavan Kumar Chaparla-1001865675

Keerthi Chittineni-1001967726

Anupama Rani Neerukonda-1002007833

Lokesh Bogam-1001946861

Jai Venkata Sai Sriram Batchu-1001949158

1. Project Plan

Introduction

There are a lot of open issues with the Manticore. The main goal of the project is to improvise the Manticore performance on the smart contracts by working on the internal Manticore API's.

In the iteration 1:

With the same motive, we have looked into the open issues that the Manticore tools currently have, in which we were able to successfully resolve one such issue #2577 by modifying the Manticore API as it was failing with the Python versions greater than 3.6.

In the same way, we are looking into the issues related to time complexity, code coverage, etc. that can help in improving the performance of the manticore when tested with smart contracts of Ethereum and WASP etc.

Project plan Features:

1. Compile and run Solidity Code
   The IDE will allow the user to write the solidity code that they would like to analyse.
2. Analysing the smart contract with Manticore
   The user can compile the smart contract code using the Manticore API.
3. Generating the results
   The tool, once finished with the analysis, will produce the results into reports and save them in the specified directory.

2. Design and Specifications

Use Case Diagrams:

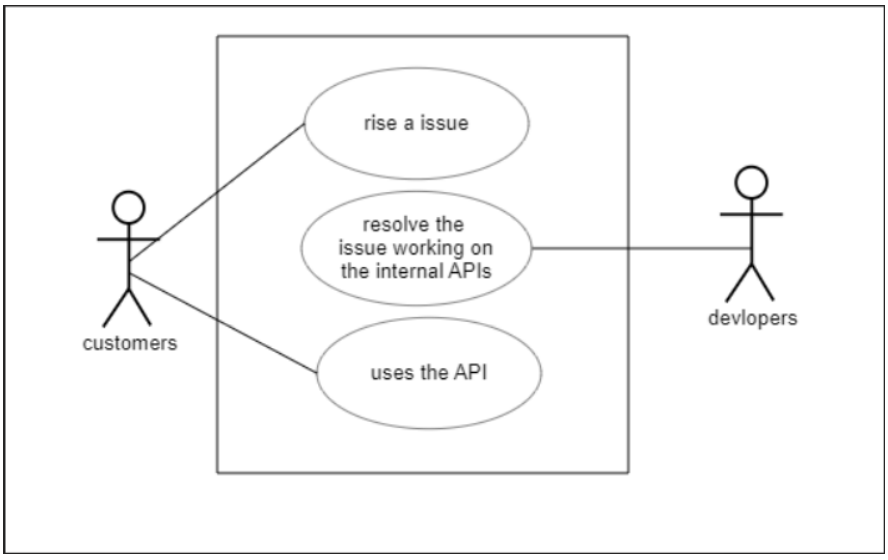

Fig 1: Use case diagram for customers and Developers

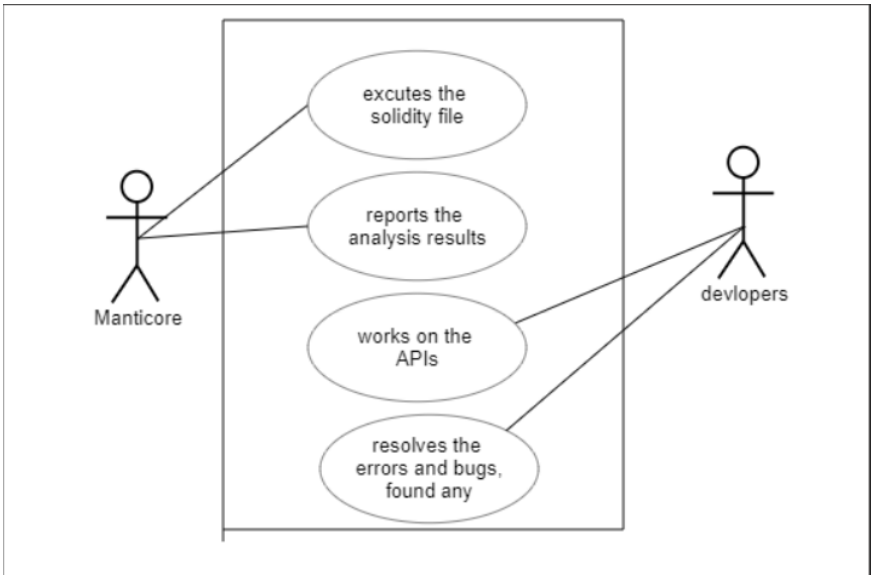

Fig 2: Use case Diagram for Manticore and Developers

Issue #2577

We have identified an open issue in the manticore development repository which was occurring with the newer versions of python i.e. >3.6, which was because the API's that were developed for the manticore were based using the python versions <= 3.6 and so, it was failing to work with python 3.10.

```
ubuntu@ubuntu:~/manticore/examples/evm$ manticore
Traceback (most recent call last):
  File "/usr/local/bin/manticore", line 5, in <module>
    from manticore.__main__ import main
  File "/home/ubuntu/.local/lib/python3.10/site-packages/manticore/__main__.py", line 13, in <module>
    from .wasm.cli import wasm_main
  File "/home/ubuntu/.local/lib/python3.10/site-packages/manticore/wasm/__init__.py", line 1, in <module>
    from .manticore import ManticoreWASM
  File "/home/ubuntu/.local/lib/python3.10/site-packages/manticore/wasm/manticore.py", line 9, in <module>
    from .types import I32, I64, F32, F64
  File "/home/ubuntu/.local/lib/python3.10/site-packages/manticore/wasm/types.py", line 5, in <module>
    import wasm
  File "/home/ubuntu/.local/lib/python3.10/site-packages/wasm/__init__.py", line 5, in <module>
    from .decode import (
  File "/home/ubuntu/.local/lib/python3.10/site-packages/wasm/decode.py", line 5, in <module>
    from .modtypes import ModuleHeader, Section, SEC_UNK, SEC_NAME, NameSubSection
  File "/home/ubuntu/.local/lib/python3.10/site-packages/wasm/modtypes.py", line 4, in <module>
    from .wasmtypes import *
  File "/home/ubuntu/.local/lib/python3.10/site-packages/wasm/wasmtypes.py", line 4, in <module>
    from .types import UIntNField, UnsignedLeb128Field, SignedLeb128Field
  File "/home/ubuntu/.local/lib/python3.10/site-packages/wasm/types.py", line 296, in <module>
    class Structure(WasmField):
  File "/home/ubuntu/.local/lib/python3.10/site-packages/wasm/compat.py", line 26, in wrapper
    return metaclass(cls.__name__, cls.__bases__, orig_vars)
  File "/home/ubuntu/.local/lib/python3.10/site-packages/wasm/types.py", line 264, in __new__
    isinstance(cur_field, collections.Callable) or
AttributeError: module 'collections' has no attribute 'Callable'
ubuntu@ubuntu:~/manticore/examples/evm$
```

Fig 3: open issue error in manticore

So, for the same reason we have modified the structure Meta class that was defined in the one of the manticore API types.py file. By this modification we were able to resolve the issue and successfully compile the smart contract solidity code with python 3.10 version.

Below is the snippet for the same API that we have modified

```
class StructureMeta(type):
    """
    Metaclass used to create `Structure` classes,
    populating their `_meta` field and performing sanity checks.
    """
    def __new__(mcs, name, bases, cls_dict):
        # Inject _meta.
        meta = cls_dict['_meta'] = MetaInfo()

        # Iterate over fields, move relevant data to meta.
        for cur_field_name, cur_field in list(cls_dict.items()):
            # Is callable, property, private or magic? We don't touch those.
            if (
                isinstance(cur_field, collections.abc.Callable) or
                isinstance(cur_field, property) or
                cur_field_name.startswith('_')
            ):
                pass

            # Is one of our types? Metafy.
            elif isinstance(cur_field, WasmField):
                meta.fields.append(FieldMeta(cur_field_name, cur_field))

            # Unknown type, print warning.
            else:
                logger.warning(
                    'Non-WasmField typed field "{}" found on type "{}". '
                    'Ignoring.'.format(cur_field_name, name)
                )

        # Order fields by type ID (see `WasmField` for the "why").
        meta.fields = sorted(meta.fields, key=lambda x: x.field._type_id)

        # Create data class type for "instances".
        class GeneratedStructureData(StructureData):
            __slots__ = [x for x, _ in meta.fields]
            _meta = meta
        meta.data_class = GeneratedStructureData

        # Create class, saving type ref in meta.
        meta.structure = type.__new__(mcs, name, bases, cls_dict)
        return meta.structure
```

Fig 4: updated API code for Manticore

3. Code and Test:

We have executed the following solidity code with our manticore tool with the Python 3.10 version and we were able to obtain the analysis results generated by manticore.

```
contract SymExExample {


    function test_me(int a, int b, int c) public pure {
        int x = 0;
        int y = 0;
        int z = 0;

        if (a != 0) {
            x = -2;
        }

        if (b < 5) {
            if (a == 0 && c != 0) {
                y = 1;
            }
            z = 2;
        }

        // will fail when: a == 0 && b < 5 && c != 0
        assert(x + y + z != 3);
    }

}
```

Fig 5: Sample Smart contract code umd_example.sol

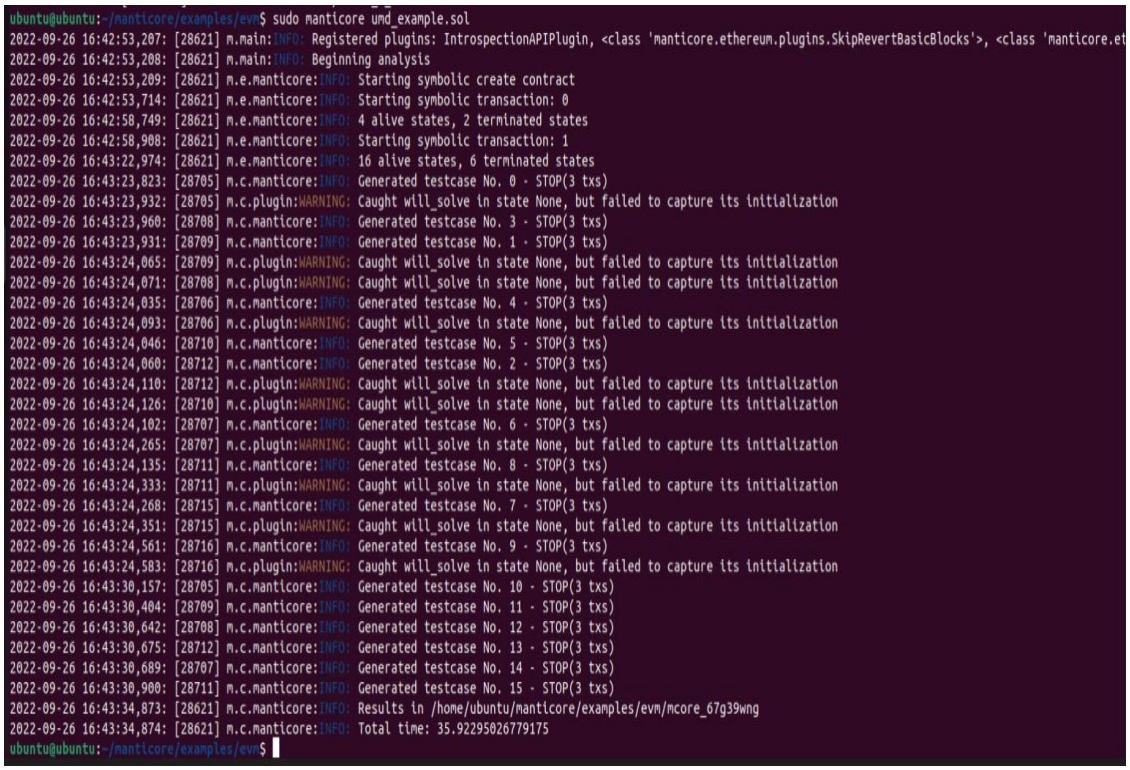

Fig 6: Sample Smart Contract code output

```
1   contract Overflow {
2       uint private sellerBalance=0;
3
4       function add(uint value) returns (bool, uint){
5           sellerBalance += value; // complicated math with possible overflow
6
7           // possible auditor assert
8           assert(sellerBalance >= value);
9       }
10  }
```

Fig 7: Sample Smart Contract code simple_value_check.sol

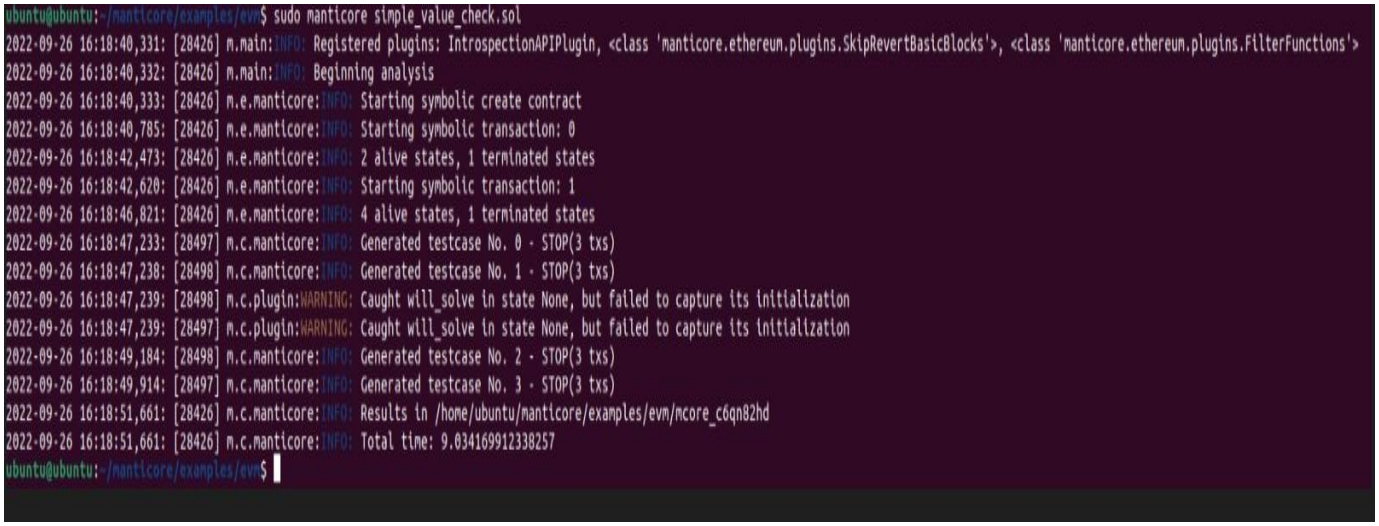

Fig 8: Sample contract execution result.

4. Customers and Users:

Usually, Symbolic Execution for Smart Contracts are used by the Smart Contract Engineers and Smart Contract Developers. However general customers involve those who would like to perform symbolic testing for scripts written in various languages like python, solidity, etc as the Manticore tool supports those languages mentioned. This is an open-source dynamic framework and the end user, or the customer could be the general audience who need the convenience of having the automated analysis for binaries and Ethereum smart contracts.

The Symbolic Execution for Analysis of Smart Contracts using Manticore can be helpful in the following ways: [1]

- Create inputs automatically for a variety of special code paths.
- Trace the inputs that caused the program to crash.
- Keep track of the execution of each instruction.
- Make its analytical engine accessible via the Python API.

The end users can use the feedback returned by the manticore symbolic analysis results in finding the unique code paths and inputs that crashes the programs. The Manticore which after the API's being modified performs better with regards to the results and also has the bugs resolved which are identified as the open issues by the developers.

References:

- **Git hub issue Link:** https://github.com/trailofbits/manticore/issues/2577
- **Fig:5 Img Refernce:** https://github.com/trailofbits/manticore/blob/master/examples/evm/umd_example.sol
- **Fig:7 Img Refernce:** https://github.com/trailofbits/manticore/blob/master/examples/evm/simple_value_check.sol
- [1] https:/medium.com/haloblock/introduction-to-manticore-a-symbolic-analysis-tool-for-smart-contract-9de08dae4e1e