Symbolic Execution Tools

22572201 Matthew Tam

September 26, 2022

1 Manticore

1.1 Background

Manticore [4] is a symbolic execution tool for Solidity Smart contract. Manticore was chosen because I previously had seen it while finding similar tools as my honours project, which is a static analysis tool for Solidity Smart contracts. Thus, I was intrigued to test this tool, see what vulnerabilities it could find, and experience the usability of the tool.

1.2 Experience Using Manticore

Manticore has good documentation in comparison to many symbolic analysis tools, however, it was frustrating to get the tool working. The frustration was due to not much clarification on how to run the tool, and the demo video showed the tool working, however, when running the exact same file and commands many errors occurred and searching these errors meant that the tool was not as straight forward to use as initially thought. This meant searching and finding ways to fix the tool before even using the tool.

1.3 Amount of effort expended to run Manticore

A few issues occurred when running Manticore. The first issue was a dependency that was not up-to-date when installing the package, this is stated in ticket [6]. This meant removing the previous package and finding the newest version of that package, thus `capstone` package was updated from version `4.0.2` to `5.0.0`. The next obstacle was that another error occurred, which gave the error output " `ERROR: Exception in state 1: ManticoreError('Forking on unfeasible constraint set')`". A corresponding ticket was found at ticket [5]. As from when this report has been published, there is no fix for this error. Therefore, to get past this hurdle, I instead of using `pip3 install manticore` version, I cloned the master branch of manticore and ran `pip install -e ".[native]"`. The next step was using Manticores functions `manticore` and `manticore-verify`.

1.4 Example and Results

The `manticore` function automatically detects if a file is of type "`.sol`" or "`.vy`" and tests the file using symbolic execution. The `manticore-verify` function simplifies contract testing and allows writing properties methods in the same high-level language as the code given to the function.

Input : ‘`manticore ManticoreExamples/Reentrancy.sol --contract Reentrance`‘

Output :

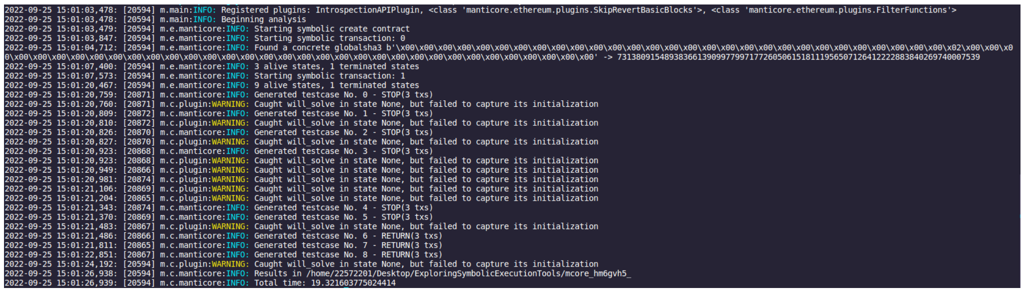
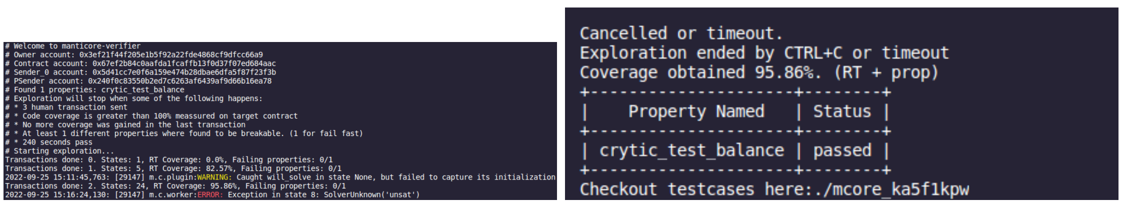


Figure 1: Manticore on reentrancy contract

Input : ‘`manticore-verifier ManticoreExamples/Reentrancy.sol --contract Reentrance`‘

Output :



(a) Section 1

(b) Section 2

Figure 2: Manticore-Verify

I tested a Reentrancy vulnerable [1] contract in figure 3a and the output is shown in figure 1, this means that a contract’s balance is vulnerable to being drained, however, Manticore did not pick up on this vulnerability. The outcome may have occurred because warnings were stated in the output that needed to be rectified, but there was no fix for these warnings, or Manticore cannot find Reentrancy vulnerable contracts.

The next contract, tested is displayed in 3c and the output is displayed in figure 2, is a contract provided in Manticore. The contract is assumes that no account can have more than 100% of the currency in the contract, however, that is not true because the transfer function allows that to be possible. However, because of the bug found when running the tool this means that the coverage obtained from `manticore-verify` is only 95.89% and should be 100% to cover the basis, thus, this function does not throw an error but rather a time-out.

(a) Reentrency.sol

(b) Test.sol

(c) Property.sol

Figure 3: Smart Contracts

2 CrossHair

2.1 Background

CrossHair [2] is a symbolic execution tool for Python code. CrossHair tool was chosen because the documentation and API looked great and well documented. Additionally, I wanted to see a symbolic execution tool in Python, this is because every other module this semester uses python because of their vast libraries. Therefore, I wanted to see if a symbolic analysis tool was easily accessible in python.

2.2 Experience Using CrossHair

The CrossHair documentation was the best documentation I have seen out of about 10 static analysis tools, however, even CrossHair documentation was not very specific. The documentation introduces the function `watch`, however, does not explain how watch worked and when to use it. Reading further into the documentation, I was introduced to the `check` function and some examples of the use of the function. I then started to understand how the tool worked. CrossHair functions look at `assert`, `PEP316`, `icontract`, `deal`, and `Hypothesis` types of python. The purpose of these types of code is to demonstrate what a method is intended to achieve, and CrossHair attempts to find a counter-example.

2.3 Amount of effort expended to run CrossHair

Since the documentation shown in reference [3] is very good, there was minimal effort expended to run CrossHair. The ease of using CrossHair was due to the simplicity of the tool only using a minimum amount of external libraries and keeping the libraries up to date, thus there were no issues in version control when using the tool. The documentation was good, and the examples helped my understanding of what each function of the tool does.

2.4 Example and Results

Input : ‘`crosshair check CrossHairExample/Test.py`‘

Output :

```
/CrossHairExample/Test.py:5: error: false when calling sumNumbers([-36979.0, -175.0, -175.0, 0.0, 1.0]) (which returns -74656.0)
/CrossHairExample/Test.py:16: error: false when calling getDifferenceOfMaxAndMin([0.0]) (which returns 0.0)
```

Figure 4: CrossHair Test using PEP316 notation

Input : ‘`crosshair check --analysis_kind=asserts CrossHairExample/Test2.py`‘

Output :

```
CrossHairExample/Test2.py:21: error: AssertionError:  when calling isUsable(Vehicle(type=<VehicleType.BICYCLE: 4>, fuel_level=None))
```

Figure 5: CrossHair Test using assert notation

In figure 6 the two python files that were analysed with CrossHair are displayed. The first file 6a's assumptions of each function are in 3 single quotes at the start and 3 single quotes at the end. In the single quotes, there are pre- and post-assertions. CrossHair attempts to find counter examples for the post-assertions given the pre-assertion. Thus, when connecting figures 4 and 6a, CrossHair tool found errors in `sumNumber` function and `getDifferenceOfMaxAndMin` function. These errors occur because in `sumNumber` function, the assumption that if given an array that has at least 1 element, the function should return the sum of the numbers in the array. However, this is not the case when given the input `[-35878.0, -175.0, -175.0, 0.0, 1.0]`, thus CrossHair generated

a counter example for the assumption in the `sumNumber` function. This is done in the same way for `getDifferenceOfMaxAndMin` function to get a counter-example.

In figure 6b the assumptions are in the form of `assert` statements. The first assert statement is the pre-assertion and the second assert statement is the post-assertion. Therefore, connecting figures 5 and 6b, there is an error that is found when the function `isUsable` takes in an object `Vehicle(type = <VehicleType.BICYCLE>, fuel_level = None)`. This error occurs because the post-assertion assumes that only a `VehicleType` of `truck`, `motocycle`, and `car` are possible after the if statement before the assertion takes place, however, this is not the case because `VehicleType bycicle` is still possible. Therefore, the error has been thrown. The `isUsableFixed` function is the fixed version of `isUsable`, and CrossHair does not find any errors with this function.

(a) Example 1

(b) Example 2

Figure 6: Python Files

References

[1] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe. Reguard: finding reentrancy bugs in smart contracts. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, pages 65–68. IEEE, 2018.

[2] P. Schanely. Crosshair, 2022. URL https://github.com/pschanely.

[3] P. Schanely. Crosshair docs, 2022. URL https://crosshair.readthedocs.io/en/latest/contracts.html.

[4] TailOfBits. Manticore, 2022. URL https://github.com/trailofbits/manticore.

[5] TailOfBits. Manticoreerror1, 2022. URL https://github.com/trailofbits/manticore/issues/2492.

[6] TailOfBits. Manticoreerror2, 2022. URL https://github.com/trailofbits/manticore/issues/1291.