Runtime Monitoring of Smart Contracts

On the Ethereum network

Lars Stegeman [s1346466]
l.stegeman@student.utwente.nl

March 23, 2018

1 Introduction

Smart contracts are an integral part of the Ethereum network. Every day many new contracts are written and deployed to the network. Vulnerabilities within these contracts are irreversible since they are part of the immutable blockchain. To detect these vulnerabilities different techniques can be used. Formal verification proves a specification against all possible input on a certain contract. While testing only guarantees correct output for a given input. In this research the feasibility of runtime monitoring is analysed.

2 Runtime monitoring

During runtime monitoring a specification is checked at specific times during execution. This means that a specification has to be known before hand. A compiler will translate the solidity code and the given specification to a new solidity contract. This contract behaviour will be exactly the same as the original contract, but it is inlined with extra assertions. These assertions are generated from the specification file. Writing a specification and checking it runtime can have several benefits

- Explicity writing a specification helps understanding the problem. The code usually describes how a contract should behave and do calculations. While the specification should describe what the contract does and what properties should be satisfied.
- Runtime exceptional state. While the contract is active on the main Ethereum network properties can be checked at runtime. If a certain property fails due to an untested case, the program can go into an exceptional state. In this state, functions can be deactivated or the contract can be completely cleared.
- It helps with bug bounty programs. When new contracts are developed they usually first launch a bug bounty program. Vulnerabilities that discovered by users can be rewarded by the creators of the contracts before it goes live on the main network. In most of these programs the specification of what the contract should do is not given. This makes it difficult do decide what is intended behaviour and what counts as a found vulnerability. With runtime verification one could specify properties that the contract should satisfy. If one of these properties fail because of a vulnerability found by the community, it is guaranteed not to be intended. This makes these bug bounty programs more useful for both the creator and the participants.
- The output of the tool is Solidity code which means that it can serve as input to other formal verification tools. For example the KEVM framework, which formally verifies smart contracts at the EVM bytecode level.

3 Research Topics

The result should be a tool that does the items that were described above. Before implementing can start a few aspects have to be studied first:

- How to specify properties? What language/technique must be used, and what properties are needed to specify smart contracts?
- Related work in the field. Other runtime verification tools for Solidity smart contracts? There are runtime verification tools for traditional programming languages and what do they do?
- What are Solidity smart contracts able to do? Analyse the documentation and what are the basic building blocks of a smart contract.
- Which tool/library to use for implementation of the compiler?