Research Proposal

Detecting Smart Contracts Vulnerabilities in Blockchain Software

STUDENT NAME: Siyu Luan
COURSE NAME: Doctor of Philosophy
DEPARTMENT: Faculty of Engineering and Information Technology
COURSE CODE:
SUPERVISOR: Yulei Sui
DATE OF SUBMISSION: 27/3/2018

1. Introduction:

Smart contract[1] is a computer program that can be automatically executed. The execution of the contract does not require third parties to supervise it. It can effectively simplify the transaction process and has many advantages such as security and reliability.

With the development of digital currency, blockchain[2] technology is being applied more and more in areas such as finance and logistics. Its core is a distributed database, with decentralization, transparency, openness, autonomy, information can not be modified, and anonymity.

Because of the above characteristics, blockchain technologies are instrumental for delivering the trust model envisaged by smart contracts.

2. Problem statement

Due to the features of the blockchain, traditional software engineering can not be able to effectively guide blockchain programming, in recent years, several detrimental software misbehaviors, which caused significant monetary loss and community splits, have posed the problem of the correct design, validation and execution of smart contracts. In 2017 a bug discovered in a smart contract library used by the Parity application [3-4], vulnerability makes it possible for hackers to become the owner of the library through library functions. Then the suicide function is called to cause the entire contract library to be destroyed. About 500K Ethers in the wallet was frozen.

3. Aims and objectives

Compared to traditional Software Engineering, a discipline of Smart Contract and Blockchain programming, with standardized best practices that can help in solving mentioned problems and conflicts, is not yet sufficiently developed.

The Parity wallet case of study clearly showed that in order to drive blockchain applications to the next reliable level， a Blockchain-Oriented Software Engineering (BOSE) is needed to define new directions to allow effective software development.

The aim of BOSE is to create a bridge between traditional software engineering and blockchain software development, addressing the issues posed by smart contract programming and other applications running on blockchain, defining new ad-hoc methodologies, quality metrics, security strategies and testing approaches capable to guarantee the novel and disciplined area of software engineering and paving the way for a testable, disciplined and verifiable smart contract software development.

4. Research Methodology

The development of bug-free source code is still a utopia for the traditional software development, after decades of analysis and development of engineering approaches. Error freedom is even more daunting for blockchain software development, BOSE approaches that could have helped in mitigating the effects of the attack, drawing from

accepted best practices in traditional software engineering.

At least two main areas to start addressing have been highlighted by analysis of a specific case of study:

• Design patterns

An anti-pattern is a common response to a recurring problem that is usually ineffective and risks being highly counterproductive. Therefore, we should review the source code, summarize the current blockchain programming vulnerabilities and anti-patterns, propose the better design patterns

• Testing

Testing is another very important technique to detect bugs in blockchain programming. Testing smart contract is challenging and critical, because once deployed on the blockchain they become immutable, not allowing for further testing or upgrading.

There are currently two test technology directions.The first one is using robust testing techniques. Manual Test generation is likely to form an important component but inevitably is limited; there is a need for effective automation test generation (and execution) techniques[5-6].

The other is the approach which is to base test automation on a model; an approach that is typically called model-based testing (MBT) [7]. We have seen that smart contracts are state-based and so it would be natural to use state-based models in MBT, allowing the use of a wide range of test

automation techniques. Naturally, the effectiveness of MBT depends on the model and also the fault model (or test hypotheses) used. An interesting challenge is to explore smart contracts and their faults in order to derive appropriate fault models or test hypotheses.

5. Expected outcomes

The expected outcomes of the project are 1) an open-source tool for automatically detecting bugs of the blockchain applications; 2) high-quality publications in the area of software engineering, blockchain and artificial intelligence.

6. Research Timeline

Months 1-6	● Conduct continuous, through literature review to identify gaps in knowledge and experts. ● Identify specific aims of project based on research vision, plan, data results literature review results.
Months 7-11	● Investigate the existing implementation and how they relate to my specific project. ● Design and implement program.
Months 12-16	● Obtain advice/guidance from colleagues and sponsor sources (e.g., research supervisors). ● Know potential reviewers (my audience). ● Compiler implementation and performance improvements (in both design and implementation).
Months 17-27	● Write research paper draft. ● Put research paper draft aside for a time, then edit. ● Ask external reviewers to review draft and provide comments.
Months 28-33	● Try to put everything together and investigating other directions which may appear. ● Rewrite research paper based on external reviewers comments(this process may continue until close to proposal submission).
Months 33-36	● Submit Ph.D. paper to UTS graduate school and review.

7. References:

1. Szabo N. Smart contracts: building blocks for digital markets[J]. EXTROPY: The Journal of Transhumanist Thought, (16), 1996.
2. Nakamoto S. Bitcoin: A peer-to-peer electronic cash system[J]. Consulted, 2008.
3. “Ethereum foundation. the solidity contract-oriented language.” https://github.com/ethereum/solidity., 2014.
4. “Ethereum foundation. ethereum original white paper.” https://github.com/ethereum/wiki/wiki/White-Paper, 2014.
5. C. Cadar and K. Sen, “Symbolic execution for software testing: three decades later,” Commun. ACM, vol. 56, no. 2, pp. 82–90, 2013. [Online]. Available: http://doi.acm.org/10.1145/2408776.2408795
6. M. Harman and P. McMinn, “A theoretical and empirical study of search-based testing: Local, global, and hybrid search,” IEEE Trans. Software Eng., vol. 36, no. 2, pp. 226–247, 2010. [Online]. Available: https://doi.org/10.1109/TSE.2009.71
7. R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. J. Krause, G. Luttgen, A. J. H. Simons, S. A. Vilkomir, M. R. Woodward, ¨ and H. Zedan, “Using formal specifications to support testing,” ACM Comput. Surv., vol. 41, no. 2, pp. 9:1–9:76, 2009. [Online]. Available: http://doi.acm.org/10.1145/1459352.1459354