Electronics and Computer Science
Faculty of Engineering and Physical Sciences
University of Southampton

German Nikolishin

November 29, 2023

Folidity - Safe Functional Smart Contract Language

Project Supervisor: Prof. Vladimiro Sassone
Second Examiner: Dr Butthead

A project progress report submitted for the award of
BSc Computer Science

Contents

1. Introduction

The idea of "smart contract" [SC] was first coined by Nick Szabo as a computerised transaction protocol [1]. He later defined smart contracts as observable, verifiable, privity-applicable, and enforceable programs. [2]. In other words, it was envisioned for smart contracts to inherit the natural properties of the traditional "paper-based" contracts.

It was only in 2014 when SCs were technically formalised at the protocol level, as an arbitrary program written in some programming language (Solidity) and executed in the blockchain's virtual machine (EVM) [3].

Ethereum Virtual Machine (EVM) iterated over the idea of Bitcoin Scripting allowing developers to deploy general-purpose, Turing-Complete programs that can have own storage, hence state. This enabled the development of more sophisticated applications that spanned beyond the simple transfers of funds among users.

Overall, SC can be summarised as an *immutable*, *permissionless*, *deterministic* computer programs that are executed as part of state transition in the blockchain system. At the time of writing, Solidity is still the most widely used SC language (SCL) [4].

After a relatively short time, SCs have come a long way and allowed users to access different online services in a completely trustless and decentralised way. The applications have spanned across financial, health, construction and many other sectors.

2. Security of Smart Contracts

2.1 Overview

With the increased adoption of decentralised applications (DApps) and the increased total value locked in DApps, there has been evidence of numerous attacks and exploits focused on extracting funds from SCc in an unconventional way. Due to the permissionless nature of SCs, the most common way of attacks is by exploiting the mistakes in the SC's source code. Specifically, the attacker can not tamper with the protocol code due to consensus mechanisms. Instead, they can cleverly tamper with the publicly accessible parameters to force the SC into an unexpected state, essentially gaining partial control of it.

A notorious example of such attacks is the DAO hack when hackers exploited unprotected re-entrance calls to withdraw **$50 million worth of ETH**. This event forced the community to hard-fork the protocol to revert the transaction provoking a debate on the soundness of the action [5].

Another less-known example is the "King of the Ether" attack which was caused by the unchecked low-level Solidity `send` call to transfer funds to some account [6].

2.2 Vulnerability classification

There has been an effort in both academia and industry to classify common vulnerabilities and exploits in SCs in blockchain systems [7][8][9]. Some of the work has been recycled by bug bounty platforms growing the community of auditors and encouraging peer-review of SCs such as "code4rena"[1], "Solodit"[2], and many others.

Analysing the work mentioned above, SCs vulnerabilities can be categorised into the 6 general groups that are outlined in Table 2.1

[1] https://code4rena.com
[2] https://solodit.xyz

Code	Title	Summary
SCV1	Timestamp manipulation	Timestamp used in control-flow, randomness and storage, can open an exploit due to an ability for validator to manipulate the timestamp
SCV2	Pseudo-randomness	Using block number, block hash, block timestamp are not truly random generated parameters, and can be manipulated by the adversary validator
SCV3	Invalidly-coded states	When coding business logic, control-flow checks can be incorrectly coded resulting the SC entering into invalid state
SCV4	Access Control exploits	This is a more categorisation of vulnerabilities. It occurs when an adversary calls a restricted function. This is specifically present in *upgradeability* and *deleteability* of SCs
SCV5	Arithmetic operations	SCs are suspected to the same arithmetic bugs as classic programs. Therefore, unchecked operations can result in underflow/overflow or deletion by zero
SCV6	Unchecked externall calls	Unchecked re-entrant, forward, delegate calls can result in the contract entering into unexpected state

Table 2.1: classification of SC Vulnerabilities

Note, for the sake of this paper, we do not evaluate the listed vulnerabilities based on their severity. As far as this paper is concerned, all vulnerabilities are considered to be of equal weight for the reasons described in Section 2.3

2.3 Setting the scene

3. Current solution

4. Proposed Solution

5. Conclusions

It works.

References

[1] Nick Szabo. Smart contracts. In *Nick Szabo's Papers and Concise Tutorials*, 1994.

[2] Nick Szabo. Smart contracts: Building blocks for digital markets. In *Nick Szabo's Papers and Concise Tutorials*, 1996.

[3] Dr. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. https://gavwood.com/paper.pdf, 2014.

[4] Mohammad Hamdaqa Majd Soud, Gísli Hjálmtýsson. Dissecting smart contract languages: A survey. arXiv:2310.02799v2, 2023.

[5] Xiangfu Zhao, Zhongyu Chen, Xin Chen, Yanxia Wang, and Changbing Tang. The dao attack paradoxes in propositional logic. pages 1743–1746, 11 2017.

[6] King of the Ether. Post-mortem investigation. https://www.kingoftheether.com/postmortem.htm, 2016. Accessed: 28/11/2023.

[7] Jinson Varghese Behanan. Owasp smart contract top 10. https://owasp.org/www-project-smart-contract-top-10/. Accessed: 28/11/2023.

[8] et al Stefano De Angelis. Security and dependability analysis of blockchain systems in partially synchronous networks with byzantine faults, 2023.

[9] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts sok. In *Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204*, page 164–186, Berlin, Heidelberg, 2017. Springer-Verlag.