International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 07 | July 2022 www.irjet.net p-ISSN: 2395-0072

Upgradability of smart contracts: A Review

Girish N M[1], Sharadadevi Kaganurmath[2]

[1]UG Student, RV College of Engineering, Bengaluru, Karnataka, India
[2]Professor, Dept of Information Science and Engineering, RV College of Engineering, Bengaluru, Karnataka, India

---

Abstract - *Smart contracts have been developed and employed in both permissioned and permissionless blockchains recently, mainly to enforce agreements among parties without the need for intermediaries. This achievement is the result of blockchain immutability which guarantees that no party can alter the conditions of an already deployed contract. However, immutability also makes patching or updating contracts impossible even when incorrectness, unfairness, or security flaws are spotted in them. So far, researchers in academia and industry have developed two main methods, data segregation and proxy storage, with six patterns to make deployed contracts upgradable. However, until now, there has been no comprehensive framework that can simultaneously offer upgradability, security resilience, and scalability features. For example, none of the existing solutions have implemented any security mechanism that can resist attacks such as the DAO one. Through extensive analysis and implementation of all these patterns, and taking state-of-the-art attacks on the Ethereum network into consideration, we review framework, "Comprehensive-Data-Proxy pattern" which uses data segregation on the top of proxy pattern, that can completely defend against any types of Reentrancy attacks. Additionally, this solution mitigates the scalability issue of the proxy pattern. Our experiments show that the framework can address these two issues with negligible impact on performance.*

Key Words: **Blockchain, Smart Contract, Ethereum, Distributed platform, DAO**

I. INTRODUCTION

Due to the immutability of blockchains, smart contracts [26] are replacing regular contracts since they can do away with middlemen. This immutable quality guarantees that deployed contracts are not changed by any participating parties, fostering confidence between them. With the exception of Bitcoin [12], many blockchains have been built to accommodate smart contracts. Smart contracts are written in languages like Go-Lang [1] and Solidity [19] on some platforms, such as Ethereum [7]. Once deployed, they can be executed by the parties concerned at specific addresses within the Ethereum network [17].

Anyone can access any address or smart contract on a permissionless blockchain like Ethereum and start its execution if the necessary criteria are satisfied. In actuality, anyone can create and use contract. Making sure that the deployed smart contracts are accurate, equitable, and secure is crucial [11]. Writing fair and error-free contracts is far from simple, as shown by the numerous attacks being attempted against the Ethereum platform [4]. This kind of security vulnerability requires the targeted smart contract to be patched immediately, or attackers exploit it again. However, patching or upgrading a contract is in contrast with the immutability feature of blockchain. We need to find a way to patch or upgrade smart contracts if we aim to develop this new technology and prepare it for mass adoption.

So far, there have been two categories of upgradable patterns developed by researchers to achieve upgradability on the Ethereum platform. The technique used in all these solutions is to map the states of storage data in the newly deployed contracts to the original versions. By doing so, the states of data can be updated and manipulated by the new contracts without making any changes to the previous versions, something which was supposedly impossible to do directly before. For instance, in [25], the authors introduce a data segregation pattern, which suggests that a contract is written as two separate (sub)contracts; a data contract and a logic contract. In this case, the upgradability is achieved through upgrading the logic contract without touching the data in the data contract. In another approach, OpenZeppelin [9] developed a novel method that uses a proxy contract to overtake the ownership of storage addresses of all versions of a given contract. Any call to the target contract would be redirected to the proxy contract which in turn, will send all transactions to the same address it controls, and through this, achieves the upgradability goal. To sum up, mapping the states of new version of the deployed contract to its original storage address is the key to any state-of-the-art solution.

While these solutions can bring upgradability to smart contracts in the immutable environment of blockchain, what is missing in them is resilience to attacks, which was one of the reasons behind the need for updating the contract in the first place. Indeed, none of the existing patterns implement any control to prevent critical attacks, such as Reentrancy attacks [16], in Ethereum. For instance, protecting the Ethers held by logic contracts in data segregation patterns is vital, but none of sub-patterns in this category can handle it. Thus, it remains an open question how we can apply a security mechanism to improve updatable contracts resilience to attacks.

In this paper, we first introduce six main state-of-the-art approaches and then comprehensively analyse four of them that address the upgradability problem in Ethereum. Three of these use data segregation pattern: basic data segregation, satellite data segregation and register pattern [25]. The remaining three methods that employ the proxy pattern approach are Inherited Storage Proxy [14], Eternal Storage Proxy [13] and Unstructured Storage Proxy [15]. After introducing the six approaches, we deeply examine three kinds of popular attacks on Ethereum which were studied in [16], [8] and [21], namely, cross-function Reentrancy, typical Reentrancy and DAO attacks. After analysing and implementing these attacks, we then review mechanism to enhance the current upgrading approaches in terms of resilience to such attacks. Next, we thoroughly analyse the proxy patterns and specifically, we study the function signatures [19] of smart contracts in order to determine how to improve the scalability of this pattern. We they implement the design of multi-contracts proxy with a single contract so that this implementation requires only one proxy for better scalability.

To this end, we formulate and implement the comprehensive combined-model, Comprehensive-data-proxy pattern [3], to fully achieve upgradability as well as security resilience and scalability. Under this model, the design smart contracts in two layers as proposed by typical data segregation patterns, but with the distinctive characteristics. First, they embed one extra authentication and data verification component between data layer and its logic contract layer, to rule out the possibility of DAO attacks. Second, we implement a proxy pattern at the data layer to fully achieve upgradability.

The first contribution is the analysing of state-of-the-art methods that have been developed to upgrade smart contracts, in which we shows that while proxy patterns can fully achieve upgradability, they raise concerns about memory handling and scalability. We also found that none of the existing patterns have any security control to defend against typical attacks. Then, we make enhancements to each pattern based on our evaluation in the previous step. We show that with the enhancement, i.e. Ether-Transfer-Verification, all the existing data segregation patterns would completely thwart well-known Re-entrancy and DAO attacks. The new model of upgradable smart contract, i.e. Comprehensive-dataproxy pattern, which can simultaneously provide upgradability and resilience to typical DAO and Re-entrancy attacks. The rest of the paper is organised as follows: In Section II, we introduce the general background of smart contracts and the problems that justify the need for upgradability feature. Discussion on the results will be presented in Section III. We wrap up the paper in Section IV with our conclusion and further work.

II. BACKGROUND

A. The Concept of Upgradable Smart Contract

If smart contracts are immutable, how can we update them like conventional software? This is the main research question. It is crucial to understand that an already deployed contract or opcode cannot be changed in any way. However, we can upgrade contracts by creating a storage mapping from the new contract to the old one. Several sorts of patterns are now used to deal with contract upgradeability. The six most common upgradable contract patterns will be examined in the remaining paragraphs of this section. Additionally, we separate them into two groups: proxy and data segregation.

B. Data Segregation Patterns

In this section, we discuss the first upgradable pattern of smart contract, i.e. Data Segregation. The core idea behind data segregation is to separate data from the contract logic. In this way, a given contract is written in two separate (sub)contracts, a data contract and a logic or business contract. The logic

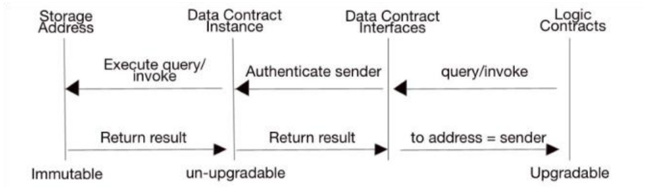

Fig. 1.Upgradability by Data Segregation.

contract interacts and manipulates the data in the data contract through some interfaces provided by the data contract (see Fig. 1). By doing so, it brings the upgradability to the contract as the logic contract can be upgraded and redeployed, while the data contract which holds all the states of for the contract remains the same on the chain. However, one disadvantage of this simple pattern is that, the data contract in this case does not have any upgradable capacity like its counterpart.

There are currently three patterns in this category: basic data segregation, satellite pattern, and register contract pattern. In the basic data segregation approach, an original contract is simply separated into two child contracts; a data contract and a logic contract. For instance, an ERC20 [23] standard contract is realised by a data contract which contains the states such as accounts and balance of account, and a logic contract which covers operations like mint and transfer methods. Similarly, Satellite Contract also contains a data contract and satellite contracts which works similarly to the logic contract mentioned before. What makes this pattern different from the previous one is that each of the satellite contracts is responsible for one functionality only. Register contract pattern is additionally used to update the logic contract address to the data contract. Although this pattern can refer to the latest copy of a contract, it cannot import the existing data into the new version of the contract. Therefore, it should be used with other types of upgradable pattern, such as basic segregation or satellite contract pattern.

C. Proxy Patterns

Another upgradable category is about proxy contracts, as Zeppolinos [10] has suggested for the Ethereum platform. There are currently three types of upgradable proxy patterns: Inherited Proxy Storage, Eternal Proxy Storage and Unstructured Proxy Storage. They all share the same foundation that the proxy contract takes control of all the states of contracts by handling every call to them though delegatecalls [6]. By doing this, the sender's original address is preserved while its call is passed through the proxy contract to the original storage address, as shown in Fig. 2.

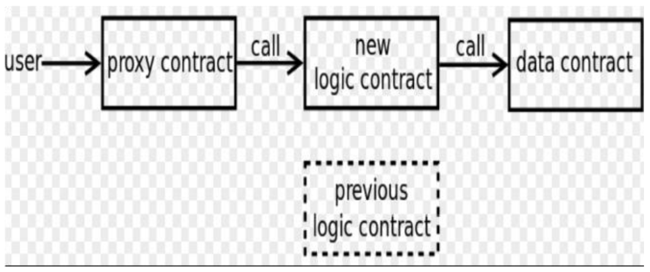

Fig. 2.Upgradability by using the Inherited Storage.

As it can be seen, all the calls to the contract are handled by the proxy contract. It forwards all these calls to the execution storage area of the contract by using delegatecall. Through this, the proxy is in control of the storage, and all states of the contract, since the delegatecall does not change the sender origin. It is also worth noticing that there is no need for separation of data and logic contracts in this solution. The target contract will update itself with its storage being controlled by the proxy contract, thus, it can access all the states of its original contract.

III. REVIEW RESULTS

In this section, we describe the key results of the experiment regarding to the performance of upgradable patterns, resilience to DAO and re-entrance attacks and upgradability capacity.

A. Performance Indicators

In order to evaluate the gas used and TPS for each pattern, we perform the experiment with two standard transactions; the mint (top up) and transfer transactions. Fig. 5. Gas used for deployment, mint and transfer (gas used for deployment was presented by 3 *percent* of actual value for presenting purpose

We also measured the gas consumed in the process of deploying smart contracts for each pattern as shown in Fig. 5. As it can be seen from the these above tables, and TPS in table 6, the amount of Gas

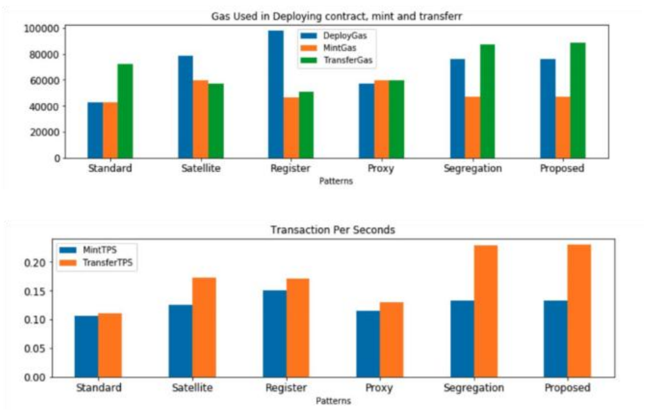

Fig. 6.The TPS performance indicator.

used for the deployment, mint and transfer method as well as the TPS are consistent with our analyses in section III. It also proved that these two indicators of proposed pattern, are not considerable more than these existing methods.

B. Resilience to DAO Attacks

In order to evaluate the resilience of patterns against two common attacks on the Ethereum Public Chain, we launched the attacks on each of them. Out result is consistent with our analysis. Without a proper protection mechanism, all patterns are vulnerable to DAO and Reentrancy attacks. On the contrary, the proposed pattern completely blocks these attacks even though the above-mentioned vulnerabilities exist in the logic contract. In the other words, DAT and any types of Reentrancy attacks are eliminated by the use of data segregation technique introduced.

C. Upgradable Capacity

As we stated before, none of the data segregation patterns can provide full upgradability, since only the logic contract of this category is upgradable while its counterpart (the data contract) is not. On the other hand, all the three proxy patterns as well as the pattern have the capacity for complete upgradability. In the case of proxy patterns, the target contact can be upgraded freely regardless of the proxy contract. The pattern also achieves this with both logic contract and data contract, plus multi-proxy contract capacity.

CONCLUSION

In this paper, we analysed the need for upgradability in smart contracts, state-of-the-art upgradable patterns, then we thoroughly analysed those patterns to show the main differences, also the limitations.

REFERENCES

[1] Golang documentation. [Online]. Available: https://golang.org/doc/

[2] N. Atzei, M. Bartoletti, and T. 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*. New York, NY, USA: Springer-Verlag New York, Inc., 2017, pp. 164–186. [Online]. Available: https://doi.org/10.1007/978-3-662-54455-6 _8

[3] C. Bui. (2019, May) Upgradable smart contract patterns. [Online]. Available: https://github.com/SWlabs/dao-attacks

[4] Consensys, "Known attacks," May 2016. [Online]. Available: https: //consensys.github.io/smart-contract-best-practices/known attacks/

[5] P. Daian. (2016, Jun) Analysis of the dao exploit. [Online]. Available: http://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/

[6] Ethereum, "Ethereum virtual machine opcodes," Sep 2018. [Online]. Available: https://ethervm.io/#CALLDATALOAD

[7] "What is ethereum," July 2019. [Online]. Available: http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html

[8] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: Analyzing safety of smart contracts," 01 2018.

[9] Z. Labs, "Upgradeability using eternal storage," April 2018. [Online]. Available: https://github.com/zeppelinos/labs/blob/master/upgradeabilityusingeternalstorage/contracts/Proxy.sol

[10] "Upgradeability using eternal storage," April 2018. [Online]. Available: https://github.com/zeppelinos/labs/tree/master/upgradeabilityusingeternalstorage

[11] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 254–269. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978309

[12] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Dec 2008, accessed: 2015-07-01. [Online]. Available: https://bitcoin.org/ bitcoin.pdf

[13] OpenZeppelin. (2018, April) Upgradeability using eternal storage. [Online]. Available: https://github.com/OpenZeppelin/ openzeppelin-labs/tree/ff479995ed90c4dbb5e32294fa95b16a22bb99c8/ upgradeability using eternal storage

[14] Upgradeability using inherited storage. [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-labs/tree/ ff479995ed90c4dbb5e32294fa95b16a22bb99c8/upgradeability _using inherited storage

[15] Upgradeability using unstructured storage. [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-labs/tree/ff479995ed90c4dbb5e32294fa95b16a22bb99c8/upgradeabilityusingunstructuredstorage

[16] M. Rodler, W. Li, G. Karame, and L. Davi, "Sereum: Protecting existing smart contracts against re-entrancy attacks," in *Proceedings of the*

Network and Distributed System Security Symposium (NDSS'19), 2019.

[17] F. Schar, "Decentralized finance: On blockchain- and smart contract-¨ based financial markets," 03 2020.

[18] D. Siegel. (2016, June) Understanding the dao attack. [Online]. Available: https://www.coindesk.com/understanding-dao-hack-journalists

[19] Solidity. (2019, Mar) Solidity assembly. [Online]. Available: https://solidity.readthedocs.io/en/v0.5.5/assembly.html

[20] Solidity assembly. [Online]. Available: https://solidity.readthedocs.io/en/v0.5.3/assembly.html

[21] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bunzli,¨ and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 67–82. [Online]. Available:

http://doi.acm.org/10.1145/3243734.3243780

[22] D. Wesley. (2017, Oct) Reentrancy attack on a smart contract. [Online]. Available: https://medium.com/JusDev1988/Reentrancy-attack-on-a-smart-contract-677eae1300f2

[23] E. Wiki, “Erc20 token standard,” Dec 2018. [Online]. Available:

https://theethereum.wiki/w/index.php/ERC20 Token Standard

[24] M. Wohrer and U. Zdun, “Smart contracts: security patterns in the ethereum ecosystem and solidity,” in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, March 2018, pp. 2–8.

[25] M. Wohrer and U. Zdun, “Design patterns for smart contracts in the¨ ethereum ecosystem,” 2018.

[26] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger eip-150 revision (759dccd - 2017-08-07),” 2017, accessed: 201801-03. [Online]. Available:

https://ethereum.github.io/yellowpaper/paper.pdf

[27] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba, “A taxonomy of blockchain-based systems for architecture design,” in *2017 IEEE International Conference on Software Architecture (ICSA)*, April 2017, pp. 243–252.

[28] X. Xu, C. Pautasso, L. Zhu, Q. Lu, and I. Weber, “A pattern collection for blockchain-based applications,” 07 2018.