Security issues in Governance Tokens for Acentrik Data Marketplace

Singh Kirath  
U1822329J   
*School of Computer Science & Engineering*  
*Nanyang Technological University*Singapore  
kirath001@e.ntu.edu.sg

Arya Shashwat  
U1822436J   
*School of Computer Science & Engineering*  
*Nanyang Technological University*Singapore  
shashwat002@e.ntu.edu.sg

Bairi Sahitya  
U1922676B  
*School of Computer Science & Engineering*  
*Nanyang Technological University*Singapore  
sahitya002@e.ntu.edu.sg

*Abstract*— In this paper, we investigate the various security issues associated with our decentralized project “Governance Tokens for Acentrik Data Marketplace”. We discuss three different kinds of attacks – Sybil attacks, Front-running attacks, Majority attacks, and Smart Contract attacks and how they can be mounted on the application by a malicious party, furthermore, we dig deep into the major concerns and threats associated with those attacks. In the end, we come up with some detailed strategies and mechanisms regarding the mitigation of the security concerns and attacks mentioned earlier and state further improvements that can be made in the system to make it more secure.

Keywords—blockchain, Ethereum, smart contracts, Sybil attack, Governance tokens, Proof of Stake

# Introduction

Since its conceptualization in 2008, numerous applications for the Blockchain technology in various fields are on the rise. Its first use was in the cryptocurrency bitcoin, where the blockchain technology was used for distributed bookkeeping. Blockchain allows for decentralization and tamper-resistance. Furthermore, the rise of Ethereum and the Ethereum Virtual Machine (EVM) have made smart contracts possible. Smart contracts in Ethereum are immutable self-executing contracts that can be called by externally owned accounts or contract accounts in the network. Contract code can update the storage or balance of an account and send messages as necessary.

Given the age of the technology and that it is constantly changing, issues with regards to Security, Privacy and Scalability are persistent and widely evaluated for all block-chain applications. Malicious actors can attack a system in any of the major layers of the blockchain - peer-to-peer network layer, consensus layer or the application layer. Furthermore, as there is open access to all information on-chain, privacy also becomes an issue as transactions can be mapped to physical users. With replicated computation, replicated storage and consensus overhead, transactions on the blockchain have much more overhead than a traditional transaction, restricting the scalability of blockchain.

Our project focused on developing an application that facilitates Governance tokens distribution and serves as a community forum where proposals can be submitted by token holders and approved by council members. The final product is an application, with authentication features and two different access modes. All platform users can add a proposal, vote for proposals in the current epoch (voting period) and view past proposals. Council members, who are owners, investors or partners can additionally approve and decide which proposals go into the development stage, mint, and distribute governance tokens, and reward winning proposals.

The focus of this paper will be on the Security issues present in the application, namely four main types of attacks, specifically Sybil Attacks, Front running attacks, Majority attacks and Attacks making use of smart contract vulnerabilities. We discuss how each of these attacks persist in a governance tokens distribution application and propose possible solutions that can address each of these attacks. The attacks discussed focus on those affecting the application directly, and security concerns of the general blockchain network and mining process are not discussed in detail in this paper.

# Motivation and Literature Survey

## Motivation

Privacy is concerned with maintaining public verifiability while shielding transaction information. It is not as prevalent in the governance token application as no confidential or competing information is shared in the transactions. The transactions on the application are mainly concerned with proposal voting and token distribution, both of which are information that is beneficial to be open. Furthermore, Scalability is also not as big of a concern as the application is not time critical. Voting periods can be up to 2 to 3 weeks, after which proposals are reviewed and approved during council meetings.

However, it is of high importance that malicious attackers are prevented from destroying the integrity of the governance tokens and the voting process. Attacks may result in undesired proposals being accepted, stealing of reward tokens or users losing trust in the platform, all of which are detrimental to the platform for which the governance tokens are being used for.

## Literature Survey

Term Paper submitted for CE/CZ4153 Blockchain Technology course, NTU.

We analyze security attacks that have taken place in the past in Initial Coin Offerings (ICOs) and other Decentralized Application (DApps) to get a better understanding of the potential security threats and their impact on our application.

1. **Fig. 6.BECToken Attack** – The BECToken contract uses the ERC-20 token standard, similar to our development project. It was attacked in 2018, when the attacker exploited the **integer overflow vulnerability** [10]. As a result, many tokens were stolen, and the token trading was shut down temporarily. The vulnerability can be seen in Figure 1, where the batchTransfer() function allowed users to transfer tokens to multiple recipients. An integer overflow occurred at line 3 when the attacker set the variable \_value to a large value, causing the variable amount to be 0. This allowed the attack to bypass the require checks in line 4 and send large amounts to the receivers.

Fig. 1. The vulnerable batchTransfer function in BECToken

1. Status.im ICO Attack – An Initial Coin Offering (ICO) is a fundraising method where an interested investor can receive a new cryptocurrency token issues by the company when they buy into the offering. The token may have some utility or may just represent stake in the company. The attack on Status.im in 2017 was a front-running attack that caused the Ethereum network to be unusable and transactions to not confirm.

To prevent wealthy investors from buying all the tokens after the ICO is open, Status.im used a Dynamic Ceiling Algorithm to ensure a fairer opportunity for smaller investors. The algorithm implemented multiple ceilings on the maximum amount that can be deposited in, and exceeding amount was refunded to the sender.

The attack was found to be carried out by the mining pool, F2Pool, which held 23% of the mining hash rate. F2Pool sent 100 Ether to multiple new Ethereum addresses prior to the ICO. On the time of the ICO, F2Pool constructed transactions from these addressed to the Status.im ICO and mined them themselves (possibly without broadcasting to the network). The remaining funds according to the Dynamic Ceiling, and the tokens were aggregated to the F2Pool main address later.

Analysis showed that F2Pool’s successful transactions were only 10% of their failed transactions [9]. This allowed to pool to maximize their mining reward compared to the gas paid on the successful transaction and at the same time, censor other transactions from the Status.im contract. This serves as an example of bulk displacement, to front-run other transactions.

# Observations and Analysis

Here we are going to discuss the three different kinds of attacks that can be mounted on our system by a malicious party and we also have deep walkthrough on how those attacks can affect the security aspect of our application.

## Sybil attack in proposal voting

The term sybil attack is used to denote a type of attack wherein an adversary maneuvers multiple accounts or identities, in a system, in order to attain a substantial authority in the peer-to-peer network [11]. Sybil attacks can grant the malicious party, the overall control of the network which could prove disastrous especially in a system where all the parties must reach a consensus for them to decide.

Our decentralized application comes bundled with a registration feature where anyone in the community, who possesses a web3 wallet, can register as a community member and gain 5 Governance tokens in their rinkeby account address. There can be multiple profiles associated with a council member’s address as a council member is used to essentially denote a company which has multiple employees, while on the other hand, there can only be one profile associated with an address for community members. The governance tokens enable a user holding N tokens to cast N cumulative votes on several proposals proposed in the present epoch (2 weeks duration). The tokens are deducted from the user’s account according to the number of votes he/she wishes to cast. These tokens are refunded back to the respective users at the end of the epoch.

Any member in the community is free to create a proposal in the system for everyone in the community to vote. A council meeting takes place at the end of the epoch to deliberate on the proposals for the present epoch. The proposals selected by the council members, at the end of the epoch, with a consensus, are awarded 100 governance tokens, while on the other hand, the proposal that attains the maximum number of votes, in the present epoch, is awarded 25 tokens.

The cumulative votes for a proposal help the council members in judging the popularity of a proposal in the community. Proposals with a higher vote count have a higher probability of getting selected in the epoch since the council members use voting as one of the key parameters for judgement.

The potential issue in this decentralized application is that a malicious party can create a plethora of rinkeby account addresses and use them to register on our platform. This will enable the malicious party to gain 5 tokens in each address they possess in their wallet. Moreover, this gives the malicious party a front-door to gain an enormous supply of governance tokens for them to vote. Thereby giving the malicious party a free license to firstly, cast a colossal number of votes for their own proposal, thereby enabling the malicious party to always attain the reward for maximum number of votes in every epoch and secondly, to gain an upper hand of getting their proposals accepted by the council members at the end of the epoch, because according to the council members, a greater vote count denotes greater popularity among the community for that proposal.

## Front-running attacks

A front-running attack is when a malicious node can observe a user transaction, identify its purpose, and construct its own similar request based on the user’s transaction.

The way our voting application is designed and implemented; it can potentially be vulnerable to front-running attack that can compromise the democratic nature of the voting mechanism. Some possible ways an attacker can front-running attack on the network-

1. **Displacement scenario** –

This is the scenario when a malicious node steals a user’s transaction and published his transaction with the same call and value before the user’s, essentially making the user’s request meaningless.

Our system can be liable to a displacement attack if a node decides to steal other users’ proposal submission requests. Since there is a 100 token reward for the best proposal, the malicious node can steal user proposals and publish the same before their submissions. They then have a higher probability of receiving the reward at the expense of the user’s hard work.

1. **Insertion scenario –**

In this attack, the malicious node changes the state of the contract to which the user is trying to send a transaction to. Then when the user’s transaction finally gets executed, it would be run using the contract’s new state as updated by the node.

In our system, an attacker might use this strategy to get their proposal the highest number of votes using minimum number of tokens to vote. They can do this by keeping track of all proposals and their state and when a transaction comes in raising the vote count of a particular proposal to become the highest (highest\_votes), the node can create another transaction setting the vote count of their proposal as *highest\_votes + 1*.

1. **Transaction suppression scenario –**

Using this attack, a malicious node can view all incoming transaction, can prioritize some over others. It will then essentially only run the transactions it deems desirable and suppress the others.

This attack is very detrimental to the democratic mechanism of our application. Using this, an attacker can essentially only allow transaction that are voting for the proposals submitted by the malicious node and discard the rest, ensuring a higher probability of winning the reward when the voting period ends.

## Majority Attacks

Majority attacks can happen when a single user or a group of users hold more than 50% of the power in the system, and specifically we refer to the voting power in this case. The number of governance tokens a user holds is directly proportional to the voting power that they have. Should this exceed 50%, the malicious party can wrongly dictate the proposals chosen for development and take advantage of the rewards. Majority attack can take place in our project in two forms – 51% of council members colluding, 51% of platform users colluding, and collusion of miner in blockchain.

1. **Collusion of Council Members** – Council members make the final decision of choosing which proposals are approved to go into development. All council members need submit their choice of proposals for a particular epoch (decision period) before a decision is made according to the smart contract rules. A proposal requires >50% of the council member’s approval to be finally chosen. Malicious council members can collude to get an unfavorable proposal to be approved. Such a proposal may bring in changes that may not be in the best interest of its users or the platform’s future.
2. **Collusion of Platform Users** – Malicious Platform users can collude and vote for unfavorable proposals or one of their own proposals to gain the associated rewards. If the number of tokens belonging to the malicious group is greater than 50%, this may wrongly direct the council members to believe that this proposal is desired by the users and pressure them to approve the proposal. Since reward is in the form of additional government tokens, this further increases the voting power of the malicious group for future rounds.

## Smart Contract Vulnerabilities

One of the most common ways for a malicious party to attack the governance platform is by exploiting the vulnerabilities in the smart contracts, specifically application bugs or flaws in smart contract code logic. This is especially prevalent as deployed code is permanent in the blockchain and would be costly to repair damages. The most famous attack is the DAO (decentralized autonomous organization) attack in 2016, in which an attacked managed to steal $150 million worth of ether. While the smart contracts in our development project do not deal with cryptocurrencies directly, the currency of concern here is the government tokens. Specifically, the following functions would need to highly secure to ensure that the governance tokens and governance system is not compromised.

1. Authentication Flow
2. Voting
3. Council Approval
4. Minting and Transferring.

A list of Smart Contract Weaknesses can be found at https://swcregistry.io, which includes some of the most common vulnerabilities found in Ethereum smart contracts [6]. Examples of these that our code may be susceptible to are reentrancy attacks, integer overflow and underflow, Denial of Service with block gas limit, transaction order dependance, among others.

# Proposed Solutions

## Solution to Sybil attack

We came up with several methods to mitigate Sybil attack in our decentralized application –

1. **Proof of Stake** – We can make use of the concept of Proof of stake [12], wherein a potential community member must put some initial security deposit in the form of ether, thereby making it less economically viable for the community member to create multiple accounts and launch a Sybil attack. This solution assumes that the incentive for a community member to get their proposal approved is significantly less than the security deposit they must commit in order to become a community member
2. **Hide vote count of proposals** - Hide the total number of votes on the front-end and only display the votes after the proposals have been judged by the council member in the council meeting. This will partially mitigate our problem as the council members will not be able to view the votes, hence the total number of votes received by the proposal will not have an influence on the council member’s judgment. However, the malicious party can still plant a Sybil attack to guarantee themselves the highest vote count reward in every epoch. We can further counter this by scraping the highest vote count for the epoch reward.
3. **Restricted usage** - Make the application restricted by having a set of whitelisted community member address known by the driver of project and the council members. This will prevent any untrusted parties from being a community member and launching a sybil attack. The issue with this solution is that the community members will have to manually whitelist the addresses of new users if they are willing to make them a part of the community of our application.
4. **Track IP address** – In this solution, we keep a mapping of the ip address as key and the address of the user as the value in order to keep track of the ip addresses of the community members. Moreover, when the user tries to register in our system, we can extract the ip address of the user by making a call to <https://geolocation-db.com/json/> from the front-end. This call will help us in attaining the ip address as well other details in json format. Furthermore, while calling the contract function for registration, we can check if there already exists a community member using the same ip address, if found we deny registration of this user who is trying to use another address for registering in our system and potentially mount a Sybil attack. The application, however, must ask the user to agree to a clause stating the collection of their ip addresses for security purposes before they register as a community member.

This solution will help us in mitigating Sybil attacks, however, since this solution involves collecting part of a user’s identity, it can potentially raise concerns for privacy in the user’s mind making our application less trustworthy to the end users. The other drawback of this solution is that it will prevent different people (using the same WIFI network) to register as community members due to the fact that they will share the same external ip address which will not be permitted by our decentralized application.

We can successfully tackle Sybil attacks by using all or a subset of the above-mentioned solutions, but every solution has its own tradeoffs which we need to consider before applying them in our decentralized application.

## Solution to front-running attacks

Our application is vulnerable to potential front-running attacks mainly because of two reasons – **(1) Lack of transaction confidentiality and (2) Miner’s ability to arbitrarily reorder transactions at will.** There are separate strategies to address each of these issues in turn. These include-

1. **CodeChain DEX Protocol (solves confidentiality leak) –**

Using this protocol, the trader encrypts the order with a publicly known encryption key. There is an entity called reveal agent (to be discussed in detail in the next chapter) that reveals the decryption key after a period. CodeChain makes sure that reveal agents reveal their decryption keys on time with the reward system and using Shamir’s Secret Sharing Scheme to recover the secret key that is required to reveal the committed order with a predefined `minimum` number of reveal agents. This idea decentralizes the classical commit/reveal scheme, which is not bound to traders’ decisions. Traders are not entities that can decide whether to reveal their commitments or not. Instead, reveal agents take responsibility and reveal with regards to the Shamir’s Secret Sharing Scheme. Besides, to have less transaction size, traders can choose not to encrypt the transaction if they believe it is safe to trade. Using this protocol, we would be able to remedy the block suppression front-running attack described above, as the malicious node would not be able to read the encrypted transaction.

1. **Injective protocol (solves transaction reordering)** **–**

This protocol is a possible solution to solve the transaction re-ordering issue. It requires the user to provide a verifiable PoEL (proof of elapsed time) using a VDF (verifiable delay function). PoEL is a measurement to choose the Taker (which is part of the blockchain that inject orders into a smart contract for settlement). A user who wants to take the order begins to solve the VDF. The longer it takes, the higher the chance that the user must become the Taker. If a person starts to solve the VDF first, he’s had more time. Although useful, this protocol adds an overhead of transaction size and computation resources while adding additional complexity to the consensus algorithm. This protocol would ensure that the malicious node is unable to re-order transactions and essentially remedy the displacement and insertion front-running attacks described above.

## Solution to Majority Attacks

The solution to the majority attack can take the form of either increasing the benefit for the council member or user to not collude or increasing the cost of collusion.

1. **Address Council Members Collusion –** The potential collusion of council members may be decreased by increasing the threshold for approval. Increasing the threshold to 70% for instance would require 70% of the council members to approve a proposal. While this increases the number of malicious council members needed to put thorough an unfavorable proposal, it might introduce the problem of favorable proposals not being approved. Additionally, functionality can be added to allow the driver, with the support of the honest council members to blacklist malicious addresses. However, it can be rationally assumed that a council member is unlikely to become malicious as they are either partners or investors in the platform and thus hold the biggest stake in the platform. Since all transactions (voting and approvals) are visible to all platform users, inappropriate approvals by the council can cause distrust among the platform users. This increases the cost for council members.
2. **Address Platform User Collusion –** Collusion among the platform users can be reduced by using a similar proof of stake solution, as proposed under the solutions for sybil attacks. Platform users would need to have stake in the platform before they can participate in the proposing and voting. This solves the 51% attack by making it disadvantageous for a user or group with 51% stake to attack and act maliciously in the voting process. Furthermore, the reward system can also be modified. Instead of getting token rewards or in addition to getting token rewards, users could get reward in the form of increased stake. This would increase the benefit to the user to act rationally and favorably to the platform.

## Solution to Smart Contract Vulnerabilities

Smart Contracts are susceptible to errors just like any code. It is important that manual audits are heavily performed to ensure code accuracy.

There are protocols in place because of previous attacks that can allow to learn and improve smart contracts. For instance, reentrancy attacks can be reduced by ensuring all internal states have been updated before external function calls are made (including calling functions that have external function calls). Reentrancy could also be addressed with the use of a mutex. This serves a lock that allows only one process to update a particular state at any time, thus ensuring proper sequence of operations [5].

It is also important that proper access privileges are in place to prevent undesired outcomes. This is especially important for voting and approval. Users should not be able to vote or approve without the proper checks. To prevent any illegal logins, passwords should also not be stored on-chain without proper encryption. The registration process can be improved by adding an additional verification step before tokens are transferred to registered users.

The use of **MultiSig** protocol is in place in the development project to offer better protection to the minting and transferring functions for the council members. These transactions require signatures from all the council members before they can be processed. This prevents users from acting maliciously and minting arbitrarily large number of tokens into the system.

There are also many security analysis tools currently available in the market to find vulnerabilities in smart contracts. One such tool is **Oyente** [4]. It is a smart contract analyzer tool that uses symbolic execution to find bugs in Ethereum smart contracts. The nature of an execution path is represented as a mathematical formula, which is then compared with formulas containing ordinary bugs [8]. While Oyente can detect severe smart contract bugs, it still poses some challenges and needs more time to be improved. For instance, it was found that the tool has a higher than desired false positive rate and may underestimate a severe bug.

# Conclusion

In this paper, we mainly covered the security aspect of our application where we discussed the various security threats related to our application as well as identified the various attacks that can mounted on our decentralized application mainly – Sybil attack, Front-running attack and Majority attack. We went one step further and discussed the various other vulnerabilities associated with our smart contracts. Furthermore, we proposed solutions to mitigate the various security threats we discussed earlier. Additionally, we went through the implementation of those solutions and discussed any potential drawbacks associated with them.

##### References

1. A. Narayanan, J. Bonneau, E. W. Felten, A. Miller, and S. Goldfeder, “Bitcoin and Cryptocurrency Technologies – A Comprehensive Introduction,” Princeton University Press 2016.
2. J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, “SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies,” IEEE Symp. on Security and Privacy 2015: 104-121.
3. I. Eyal and E. Gun Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” Financial Cryptography, 2014, pp. 436–454.

[4] S. Sayeed, H. Marco-Gisbert and T. Caira, "Smart Contract: Attacks and Protections," in IEEE Access, vol. 8, pp. 24416-24427, 2020, doi: 10.1109/ACCESS.2020.2970495.

[5] ConsenSys, “Known attacks,” *Known Attacks - Ethereum Smart Contract Best Practices*. [Online]. Available: <https://consensys.github.io/smart> contract-best-practices/known\_attacks/#timestamp-dependence. [Accessed: 20-Nov-2021].

[6] “Overview · smart contract weakness classification and test cases,” *Overview · Smart Contract Weakness Classification and Test Cases*. [Online]. Available: https://swcregistry.io/. [Accessed: 20-Nov-2021].

[7] H. Wang, Y. Wang, Z. Cao, Z. Li, and G. Xiong, “An overview of blockchain security analysis,” *Communications in Computer and Information Science*, pp. 55–72, 2019.

[8] L. Luu, D.-H. Chu, H. Olickel, P. Saxena and A. Hobor, "Making smart contracts smarter", Proc. ACM SIGSAC Conf. Comput. Commun. Secur.-CCS, pp. 254-269, 2016.

[9] Eskandari, Shayan & Moosavi, Mahsa & Clark, Jeremy. (2019). SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. 170-189. 10.1007/978-3-030-43725-1\_13.

[10] H. Chen, M. Pendleton, L. Njilla, and S. Xu, “A survey on Ethereum Systems Security,” *ACM Computing Surveys*, vol. 53, no. 3, pp. 1–43, 2020.

[11] J. R. Douceur, “John R. Douceur microsoft research ... - cornell university.” [Online]. Available: http://www.cs.cornell.edu/people/egs/714-spring05/sybil.pdf. [Accessed: 21-Nov-2021].

[12] S. SEANG and D. TORRE, “Proof of work and proof of stake ... - sciencesconf.org,” Feb-2018. [Online]. Available: https://gdre-scpo-aix.sciencesconf.org/195470/document. [Accessed: 21-Nov-2021].