

D21

Semestral task

peer review

Audit

5th December 2021

by Blockchain student Bc. Jan Bureš from UTB

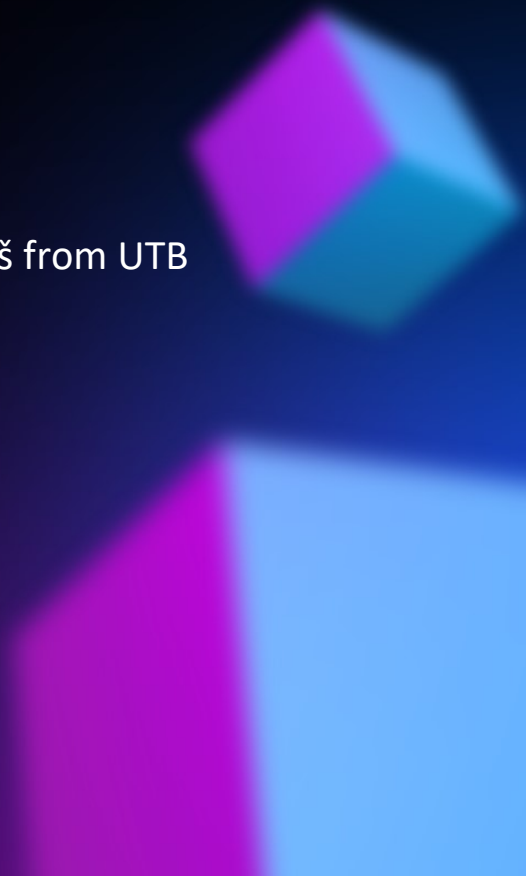
Abstract geometric shapes in the bottom right corner, including a small 3D cube with pink, blue, and teal faces, and larger, blurred pink and blue shapes below it.

Table of Contents

1. Overview	2
2. Scope	4
3. System Overview	5
4. Security Specification	7
5. Findings	9
6. Conclusion	14

Document Revisions

Version	Date	Description
1	2021/12/05	Initial audit

1. Overview

This document presents our findings in reviewed contracts as a semestral project to Blockchain subject at UTB in Zlín.

1.1 Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the recommendations for sparing gas amount.
3. **Tool-based analysis** – automated Solidity tools (Slither – static analyze, Brownie – unit testing to ensure system works as expected).
4. **Local deployment + testing + gas fee optimization** – the project is deployed locally with testing whole spectre of the Use Cases in Remix IDE.

1.2 Review team

The only auditor was me personally (Bc. Jan Bureš).

1.3 Disclaimer

I've put my best effort to find all vulnerabilities in the system. On the other hand, the general aura of testing implies, that the absence of the issue in the audit does not mean, there is none like that.

2. Scope

This chapter describes the audit scope, contains provided specification, used documentation and set main objectives for the audit process.

2.1 Coverage

Object of the testing:

- Contract: Janeček voting method D21
- Author: Bc. David Rábel
- Github repo: <https://github.com/Ackee-Blockchain/NIE-BLO-David.Rabel.git>
- Repository: Private
- Commit: be80d52d1d511fa05656d180f4f93a93f81461c2
- Date of the commit: 17th November 2021
- Files being audited:
 - IvoteD21 – interface of the contract
 - D21 – implementation of the contract

2.2 Supporting Documentation

Developer didn't provide any supporting documentation, however contract implement well-known voting metod „Janečkova metoda D21“, which is specified in many resources (such as <https://www.ih21.org/o-metode>) on the Internet.

2.3 Objectives

I've defined following main objectives of the audit:

- Check the code quality, architecture and best practices
- Check if somebody is able to use methods of the implementation against voting method rules
- Get 100% coverage of the automated test (in Brownie)
- Check other possible ways of implementation that would save gas

3. System Overview

Section below describes audited voting system from my own perspective.

IVoteD21.sol

Interface (abstract contract) used as a template of D21.sol. It Contains only methods' definitions, without implementation

function addSubject(string memory name) external;

- Add a new subject into the voting system using the name.

function addVoter(address addr) external;

- Add a new voter into the voting system.

function getSubjects() external view returns(address[] memory);

- Get addresses of all registered subjects.

function getSubject(address addr) external view returns(Subject memory);

- Get the subject details.

function votePositive(address addr) external;

- Vote positive for the subject.

function voteNegative(address addr) external;

- Vote negative for the subject.

function getRemainingTime() external view returns(uint);

- Get remaining time to the voting end in seconds.

D21

Implementation of the abstract contract IVoteD21.

4.Security Specification

Section specifies roles of the participants.

4.1 Actors

This part describes actors of the system, their roles and permissions.

User

Everybody who participate in the contract is its user. This role can be upgraded to voter, subject or even owner. Each role(s) is defined by user address, thus each user can have multiple roles.

Owner

User who deploy the contract becomes the owner, which is immutable.

Voter

Voter is user who was given the right to vote by the owner.

Subject

User can add add new subject into the voting system. For this subject is assigned address of the calling user.

4.2 Trust model

The contract is based on the trust to the owner, that he gives the right to vote to every suitable user.

5. Contract rules (UC)

This section specify the most important rules for the Janečkova volební metoda D21.

UC1 - Everyone can register a subject (e.g. political party)

UC2 - Everyone can list registered subjects

UC3 - Everyone can see subject's results

UC4 - Only owner can add eligible voters

UC5 - Every voter has 2 positive and 1 negative vote

UC6 - Both positive votes can't be given to the same subject

UC7 - Negative vote can be used only after 2 positive votes

UC8 - Voting ends after 7 days from the contract deployment

6. Findings

This chapter shows detailed output of our analysis and testing.

6.1 General Comments

Contract implementation has a pretty structure, so it is easily understandable. Author avoids every more serious issues (critical or higher).

However I found a few imperfections (most of the to spare some gas), which I recommend to update.

6.2 Issues

Using my toolset, manual code review and unit testing I've identified the following issues.

Low

Low severity issues are more comments and recommendations rather than security issues.

It is mainly about storing data issues.

Solidity stores its values to 32 byte (256 bit) slots, so it is pretty important to store variables as slightly large enough data types and even think about the order of declaring the variables (less slots = less gas), to the case we want to save some gas and not everytime just small amount of the money as the final result.

ID	Code with the issue	File	Line	Short description
L1	Multiple	IvoteD21.sol	-	Missing descriptions of the functions
L2	pragma solidity >=0.4.22 <0.9.0;	D21.sol	3	Accept outdated solidity version
L3	variables, struct Voter	D21.sol	9, 12, 15,17	First letter of the variables/struct is uppercase
L4	struct Voter{}	D21.sol	8-13	Too large data type
L5	uint256 BlockCreated	D21.sol	17	Too large data type

L1

Description:

- Comment declared functions and variables is one of the best-practices and provide much better understanding of the contract

Recommendation:

- Comment every function and variable

Cause of the recommended change:

- Lead to better understanding of the contract

L2Description:

- Although the solidity version is not directly the security issue, the lower versions are not using the newest conventions and techniques, so there is a much higher probability of security leak. Not to speak of that no one want to develop outdated software.

Recommendation:

- Require at least version 0.7.6

Cause of the recommended change:

- Avoidance compatability issues
- Might ease work with the security

L3Description:

- Variables and structures have got first letter as uppercase and should have got lowercase to correspond to the programming conventions and best practices

Recommendation:

- Change first letters of the variables and structs to the lowercase

Cause of the recommended change:

- Lead to common programming conventions and best practices

L4Description:

- Counter stores a very low decimal number (0-3), so there is no need to reserve such a big data type as int256 for this cause. Uint8 would fits better here
- int = int256, so it store value from -2^{255} to $2^{255} - 1$; stored in too many solidity slots
- uint8 stored data 0-256bit, so it is stored in a single slot (unlike int)

Recommendation:

```
struct Voter{
    bool canVote;
    uint8 votes;
    mapping(address => bool) votedSubjects;
}
```

Cause of the recommended change:

- Sparing gas – mainly in combination with issue M1

L5

Description:

- uint = uint256, so it can store decimal numbers from 0 to $2^{255} - 1$, which is unnecessarily high, cause we need just to store decimal numbers up to 604 800 (s) = 1 week
 - uint32 is fits much better for this case

Recommendation:

- uint32 blockCreated;

Cause of the recommended change:

- Sparing gas

Medium

Medium severity issues aren't security vulnerabilities but cause a higher gas fee leak which with a huge amount of the data could cause spending a pretty amount of money spent as the final result. And also system logical issues, which could lead to misunderstandings, which would cause a deviation from the default principles of the contract.

ID	Code with the issue	File	Line	Short description
M1	struct Voter{}	D21.sol	8-13	Unnecessarily used 2 counters
M2	Mutiple	D21.sol	79, 80, 86, 87	The best „++code“
M3	function addSubject(string memory name)	D21.sol	64-67	Able to add subject with blank „name“
M4	function addSubject(string memory name)	D21.sol	64-67	Not handled adding already existing subject
M5	function votePositive(address addr)	D21.sol	77-82	Not handled voting positive for the subject before its creation

M1

Description:

- If address mapping is used, only 1 variable for storing counter (number of votes) for all (positive and negative) votes really small positive decimal number (0-3) would fits better

Recommendation:

```
struct Voter{
    bool canVote;
    uint8 votes;
    mapping(address => bool) votedSubjects;
}
```

Cause of the recommended change:

- Sparing gas

M2

Description:

- Using an increment/decrement like "Voters[msg.sender].votesPositive +=1" or "subjects[addr].votes -= 1" may not seem like a problem. However, these methods cost a lot of gas. So it's better to use the ones that save us gas (and not very little).

Recommendation:

- ++ Voters[msg.sender].votesPositive;
- -- subjects[addr].votes;

Cause of the recommended change:

- Sparing a lot of gas

M3

Description:

- User is able to add a new subject with blank name attribute (name="")

Recommendation:

- Modifier (require) with condition to at least 2 char long attribute Name

Cause of the recommended change:

- Avoidance of no-name subjects, so every voter knows who is his/her votes addressed for and thus avoid waste of voters' votes

M4

Description:

- User is able to add subject with the same name as already existing subject

Recommendation:

- Modifier (require) with condition to check if there is any subject with the given name

Cause of the recommended change:

- Avoidance of duplicit subjects, which could cause to incorrect voting results

M5

Description:

- Voter is able to give a positive vote to the subject even if it does not exist
- This issue in negative voting is resolved thanks to other requirements (allow negative vote only after 2 positives one)

Recommendation:

- Modifier (require) to check whether the subject on the given address exists (attribute name is not blank)

Cause of the recommended change:

- Avoidance of giving votes to the non-existing subject and thus waste of voters' votes

High

High severity issues are security vulnerabilities, which require specific steps and conditions to be exploited. These issues have to be fixed.

ID	Code with the issue	File	Line	Short description
--	-	-	-	--

No high severity issues were found.

Critical

Direct critical security threats, which could be instantly misused to attack the system. These issues have to be fixed.

ID	Code with the issue	File	Line	Short description
-	-	-	-	-

No critical severity issues were found.

6.3 Static testing

All of the tests found in the static analysis can be considered negligible because none of them significantly affect the contract.

```
— Slither: Solidity static analysis framework by Trail of Bits —
Using slither version: 0.8.1
Refreshing explorer...
Loaded 0 issues, displaying 0
— Starting analysis —
✗ D21.votePositive(address) (D21.sol:108-122) compares to a boolean constant:
  • require(bool,string)(Voters[msg.sender].VotedSubjects[addr] != true,U already voted for that subject!) (D21.sol#115-118)

✗ D21.voteNegative(address) (D21.sol:124-139) compares to a boolean constant:
  • require(bool,string)(Voters[msg.sender].VotedSubjects[addr] != true,U already voted for that subject!) (D21.sol#132-135)

✗ D21.isVoter() (D21.sol:64-70) compares to a boolean constant:
  • require(bool,string)(Voters[msg.sender].CanVote == true,U are not in Voter list please contact the owner!) (D21.sol#65-68)

✗ Different versions of Solidity is used:
  • Version used: ['0.8.10', '>=0.4.22<0.9.0']
  • >=0.4.22<0.9.0 (D21.sol:3)
  • 0.8.10 (IVoteD21.sol#2)
  • ABIEncoderV2 (IVotedD21.sol#3)

✗ Pragma version>=0.4.22<0.9.0 (D21.sol:3) is too complex

✗ Pragma version0.8.10 (IVotedD21.sol:2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

✗ solc-0.8.10 is not recommended for deployment

✗ Variable D21.Voters (D21.sol:15) is not in mixedCase

✗ Variable D21.BlockCreated (D21.sol:17) is not in mixedCase

✗ Modifier D21.CanVoteNeg() (D21.sol:24-30) is not in mixedCase

— Analysis: 1 succeeded, 0 failed, 0 skipped —
Refreshing explorer...
Loaded 10 issues, displaying 10
```

6.4 Unit testing

For unit tests was utilize tool Brownie. There was make 27 tests total to cover and covered every line of the code considering every (main) Use Cases. Although I discovered 3 issues, all tests are written to pass. All tests are stored in a single file „**test_D21.py**“.

Test overview

ID	Test name	Issue
1	test_checkCorrectOwner	-
2	test_addVoterAsOwner	-
3	test_addVoterAsNotOwner	-
4	test_addSubject	-
5	test_addTwoSubjectsFromTheSameAddress	-
6	test_addSubjectBlankName	M3
7	test_addExistingSubject	M4

8	test_getRegisteredSubject	-
9	test_getNotRegisteredSubject	-
10	test_getSubjects	-
11	test_getSubjectsWithoutRegistered	-
12	test_votePositiveWithoutPermission	-
13	test_votePositiveOnce	-
14	test_votePositiveTwice	-
15	test_votePositiveTwiceForTheSameSubject	-
16	test_votePositiveThreeTimes	-
17	test_votePositiveNotRegisteredSubject	M5
18	test_voteNegativeAfterNonePositive	-
19	test_voteNegativeAfterOnePositive	-
20	test_voteNegativeAfterTwoPositive	-
21	test_voteNegativeAfterTwoPositiveAndOneNegative	-
22	test_voteNegativeAfterPositiveForTheSameSubject	-
23	test_getRemainingTime	-
24	test_AddSubjectAfterDeadline	-
25	test_AddVoterAfterDeadline	-
26	test_votePositiveAfterDeadline	-
27	test_voteNegativeAfterDeadline	-

7. Conclusion

I started by manually reviewing the code to better understand the author's implementation and uncover the first flaws. Later I used testing tools like Slither for static testing, Brownie for automated testing and Remix IDE with attempts to find the loophole and find a more suitable solution that would save some gas.

No critical issue or high severity problem was found, only a few and mild ones that slightly spoil the good impression of the implementation.

Thank You

Ackee Blockchain a.s.



Prague, Czech Republic



hello@ackeeblockchain.com



<https://discord.gg/z4KDUbuPxq>