BLO

CTU FIT
Andrey Bortnikov

Security audit

D21 Voting system by Martin Ondejka

8. December 2022

Contents:

1. Executive summary

This report contains the results of the assessment of the D21 voting method implemented by Martin Ondejka.
The audited project can be found on GitHub:
https://github.com/Ackee-Blockchain-Education/NIE-BLO-Martin-Ondejka

A manual review of the contract was performed according to the voting method specifics. The audit resulted in only one warning finding and a couple of informational ones. No risky security vulnerabilities were found. Audit recommendations contain notes on code improvement and gas optimizations.

2. System overview

"Janečkova metoda D21" is a modern voting system, which allows more accurate voting. You can learn more about it here: https://www.ih21.org/o-metode.The system implemented on Solidity as an assessment for Blockchain course at FIT CTU.

Contract D21.sol implements the IVoteD21.sol interface.

State:
The D21 smart contract contains the state of the owner, end time, voters and subjects. Voter is represented as a structure, as following:

```
struct Voter {
  bool isVoter;
  uint256 positiveVotes;
  uint256 negativeVotes;
  address firstVotedSubject;
  address secondVotedSubject;
}
```

Voters are stored as a mapping, where an address points to a specific Voter. Subjects are stored both as a mapping and an array of subject addresses, there address of a subject creator points to a Subject. Contract also has 2 uint variables, storing the maximums of possible votes for each Voter.

Owner and remaining time are being set during the deployment of the contract in constructor.

Actors:

1. Owner. The owner of D21 contract. He/she can add voters to the voting system.
2. Voter. The address is considered as a voter, who can vote positive or negative for an existing subject.

3. Non-voter. The user is allowed to add a subject, list existing subjects, get subject details, check remaining time and get results. The owner and the voters can use these functions too.

Contract Interface IVoteD21.sol

Structs:

The interface contains a structure Subject. It consists of a name of type string and an integer storing a number of votes.

```
struct Subject {
  string name;
  int votes;
 }
```

Functions:

1. addSubject - Add a new subject into the voting system using the name.
2. addVoter - Add a new voter into the voting system.
3. getSubjects - Get addresses of all registered subjects.
4. getSubject - Get the subject details.
5. votePositive - Vote positive for the subject.
6. voteNegative - Vote negative for the subject.
7. getRemainingTime - Get the remaining time to the voting end in seconds.
8. getResults - Get the voting results, sorted descending by votes.

3. Trust Model

Users have to trust the owner to add them to the voting system. Could be potentially exploited if the owner does not add voters, who would vote against his/her preferences.

The following rules should also be met:

- UC1 - Everyone can register a subject (e.g. political party)
- UC2 - Everyone can list registered subjects
- UC3 - Everyone can see the subject's results
- UC4 - Only the owner can add eligible voters
- UC5 - Every voter has 2 positive and 1 negative vote
- UC6 - Voter can not give more than 1 vote 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

4. 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. Tool-based analysis - deep check with automated Solidity analysis tools and Woke is performed.

3. Manual code review - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.

4. Local deployment + hacking - the contracts are deployed locally and we try to attack the system and break it.

5. Unit and fuzzy testing - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests

3.1 Findings classification

• **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.

• **Medium** - Code that activates the issue will result in consequences of serious substance.

• **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.

• **Warning** - The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.

• **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change

5. Tool-based analysis

The following detectors were tested by the woke tool:

• axelar-proxy-contract-id
Detects incorrect use of the contractId function in Axelar proxy and upgradeable contracts.

• function-call-options-not-called
Function with gas or value set actually is not called, e.g. this.externalFunction.value(targetValue) or this.externalFunction{value: targetValue}.

• overflow-calldata-tuple-reencoding-bug
Detects Head Overflow Calldata Tuple Reencoding compiler bug

• reentrancy
Detects re-entrancy vulnerabilities.

• unchecked-function-return-value
Return value of a function call is ignored.

• unsafe-address-balance-use
Address.balance is either written to a state variable or used in a strict comparison (== or !=).

• unsafe-delegatecall
Delegatecall to an untrusted contract.

• unsafe-selfdestruct
Selfdestruct call is not protected.

6. Findings

Woke tool didn't find any vulnerabilities. All of the findings are found by manual code review, related to gas optimizations and code quality.

W1. Owner variable is private.

Description:

Owner variable is declared as private. In the contract scope, the voting system should be transparent for users to trust it and to be assured of the origin of the contract.

Exploit scenario:
Alice copies the contract and deploys her own. Bob doesn't know the contract is original and performs tasks on the fake instance.

Recommendation:

Make the owner variable public and immutable.

I1. Unnecessary storage variables.

Description:

Function votePositive() and voteNegative() contain storage variable for a voter:

```
    Voter storage voter = _voters[msg.sender];
```

This will cost more gas, and we are able to access the voter without the storage variable declaration. Moreover, a code after this declaration has following requirements:

```
require(voter.firstVotedSubject != addr);

require(voter.secondVotedSubject != addr);

require(voter.positiveVotes == MAX_POSITIVE_VOTES);

require(voter.negativeVotes < MAX_NEGATIVE_VOTES);
```

So, even if some condition is not met and the function stops, we still declare a storage variable, which is not optimal.

Recommendation:
We can use _voters[msg.sender] without the new variable and still get access to the voter and save some gas.

I2. Unnecessary big uint variables in voter structure.

Description:
Voter structure contains attributes to store counters for positive and negative votes and their type is uint256. Due to rules, these counters cannot be negative and cannot be bigger than 2 for positive votes and 1 for negative ones.

Recommendation:
Declare these attributes as int8, which will be enough for the contact purposes. It also can save some gas.

I3. Unnecessary big uint state variables.

Description:

Store variables MAX_POSITIVE_NOTES and MAX_NEGATIVE_VOTES are declared as uint256.

```
uint256 private constant MAX_POSITIVE_VOTES = 2;

uint256 private constant MAX_NEGATIVE_VOTES = 1;
```

These variables are constant and store “2” and “1” respectively.

Recommendation:
There are 2 options in this case.
First: We can get rid of these variables as voter structure contains counters for positive and negative votes. We can check counters explicitly in the code by comparing them to 2 or 1 ints. It will decrease code readability and possible further code modifications, but it will save memory and, therefore, some gas.
Second: Declare these attributes as int8, which will be enough and can save some gas.