Auditing report – Jérôme Verstraeten

1. Document Revisions

Version	Date	Description
1.0	2022-06-04	Initial audit

2. Overview

2.1 Audit Methodology

1. **Technical specification/documentation** – a brief overview of the system is given.
2. **Tool-based analysis** – check with automated Solidity analysis tools and woke is performed.
3. **Manual code review** – the code is checked line by line.
4. **Local deployment** – the contracts are deployed locally.

2.2 Finding classification

The following categories will be used based on the level of severity impact:

- **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.

The following categories will be used based on the level of likelihood:

- **High** – The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** – Exploiting the issue currently requires non-trivial preconditions.
- **Low** – Exploiting the issue requires strict preconditions.

2.3 Review team

Member's name	Position
Jérôme Verstraeten	Student, Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.4 Disclaimer

We've put our best effort to find all vulnerabilities in the system, however, our findings shouldn't be considered a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Summary of findings

The following table summarizes the findings we identified during our review.

	Severity	Reported	Status
P1: use of block.timestamp can be manipulated.	Low	1.0	Acknowledged
P2: Gas limit may be exceeded.	Medium	1.0	Acknowledged
P3: Access control could be improved	High	1.0	Acknowledged

4. Report revision 1.0

System overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

The goal of the contracts is to provide the "Janečkova metoda D21" voting system. More about this specific voting system can be found here: https://www.ih21.org/o-metode. The following use cases must be achieved:

- 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 cannot 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.

Contracts

Contracts we find important for better understanding are described in the following section.

IVoteD21.sol

Contract IVoteD21.sol serves as an interface for other contracts. The contract helps with correct implementation of D21.sol. It is strictly necessary to follow this interface.

D21.sol

This contract implements 8 functions from IVoteD21.sol by using inheritance. The goal of the contract is to actually prove the implementation of the voting system.

Actors

Owner

The owner is the address that deploys the contract. Only the owner can add eligible voters. Furthermore, the owner can register, list and see the subjects (results).

Voters

Voters are the addresses that can register, list and see the subjects (results). They can give 2 positive and 1 negative vote to the different subjects.

Trust model

Voters will have to trust the owner. They have to be confident that the owner has no malicious intent. For example: they must trust the owner to register all the different subjects in a right and honest way.

Audit

Woke output

```
Using the following detectors:

• axelar-proxy-contract-id
  Detects incorrect use of the contractId function in Axelar proxy and upgradeable contracts.
• bug-empty-byte-array-copy
  Detects empty array copy bug for solidity versions < 0.7.14 (https://blog.soliditylang.org/2020/10/19/empty-byte-array-copy-bug/)
• 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}.
• missing-return
  Detector that checks if all possible paths have a return or revert statement or have all return values set
• msg-value-nonpayable-function None
• overflow-calldata-tuple-reencoding-bug
  Detects Head Overflow Calldata Tuple Reencoding compiler bug
• proxy-contract-selector-clashes
  Detects selector clashes in proxy and implementation contracts. Proxy contracts are detected based on fallback function and usage of slot variables and implementation contracts that use
  same slots as proxy contracts
• reentrancy
  Detects re-entrancy vulnerabilities.
• unchecked-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.
• unsafe-tx-origin
  Detects unsafe usage of tx.origin.
  Every usage of tx.origin us unsafe expect for the following:
    • tx.origin == msg.sender and it's varieties
    • timestamp[tx.origin] < block.number
• unused-contract
  Detects abstract contracts, interfaces and libraries that are not used.
```

Figure 1: Output woke

Problem 1: use of block.timestamp can be manipulated

Impact:	Low	Likelihood:	Medium
Target:	D21.sol	Type:	Timestamp manipulation

Description

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block. In a timestamp manipulation attack, a miner intentionally manipulates the timestamp value of a block to influence the outcome of a transaction or to exploit time-dependent functionalities in smart contracts. By adjusting the timestamp, the miner can potentially change te conditions under which certain operations are executed, such as time-based unlocks, voting periods, or other time-sensitive functionalities.

Exploitation

In a voting system like this, manipulating the timestamp could extend or shorten the voting period. This allows the miner to influence the results by altering the eligibility of voters or the timing of votes.

Solution

Consider alternatives for block.timestamp.

Problem 2: Gas limit may be exceeded

Impact:	Medium	Likelihood:	High
Target:	D21.sol	Type:	Denial-Of-Service

Description

Depending on the number subjects and voters, the code may exceed the gas limit for certain operations, such as sorting the array of subjects in the ‘getResults’ function. Gas limits can prevent the execution of transactions, leading to denial-of-service attacks.

Solution

A possible solution would be pagination. Instead of sorting and returning the entire array of subjects in the ‘getResults’ function, we could implement pagination to return results in smaller batches. This way, the gas cost of the operation can be reduced, ensuring it stays within the gas limit.

Problem 3: Access control could be improved

Impact:	High	Likelihood:	Medium
Target:	D21.sol	Type:	Weak Access Control

Description

The contract uses a basic access control mechanism where only the contract owner can add voters. However, this implementation assumes that the owner’s address is secure and cannot be compromised. If the owner’s account or private key is compromised, an attacker could gain unauthorized access to the contract and perform actions reserved for the owner, such as adding voters or manipulating voting outcomes.

Solution

Implement a Role-Based Access Control system. RBAC allows for fine-grained control over different roles and their associated permissions. This would provide a more secure way to manage sensitive operations and data access.

5. Executive summary

Janečkova metoda D21 is a modern voting and electoral method.

Revision 1

CTU Prague engaged Jérôme Verstraeten to perform a security review of the voting system coded by Oleh Kuznetsov, also a student at CTU Prague. The review consisted of manual code review and the use of Solidity static analysis in Remix IDE and the tool Woke. Slither was not used due to technical issues.
The review resulted in 3 findings. The full report can be consulted under 3. Report revision 1.0.

6. Conclusion

The code exhibits both strengths and potential security issues that should be addressed to ensure the reliability and security of the smart contract.

The code provides a basic implementation of the specified voting system. Several security issues should be considered. Firstly, the reliance on block.timestamp introduces the risk of a timestamp manipulation attack. Secondly, a potential gas limit issue could occur if the number of subjects grow significantly. Thirdly, the weak access control mechanism could cause manipulation of the voting process.

It is important to note that the security audit is performed by a student, and thus, that several security issues probably went unnoticed.