System Overview

The system is a modern voting system called "**Janečkova metoda D21**", which allows for more accurate voting. It is implemented in Solidity on the Ethereum blockchain. The system allows for anyone to register a subject (e.g. political party) and list registered subjects. It also allows for every voter to have two positive and one negative vote, which can be used only after two positive votes have been given. Voting ends after 7 days contract is deployed. Vojtech Drska's implementation is composed of the following components:

1. **Structs**:

The D21 contract uses two structs: **Voter** and **Subject**. **Voter** stores a boolean indicating whether the voter has registered, an address array storing the subjects the voter has positively voted on, and an address array storing the subjects the voter has negatively voted on. **Subject** stores a string containing the name of the subject, and an integer indicating the number of votes the subject has received.

2. **State variables**:

The contract defines several variables. **_startDate** is a uint storing the date the contract was deployed. **_owner** is an address storing the address of the contract owner. **_subjectKeys** is an address array storing the keys of the registered subjects. **_subjects** is a mapping containing the Subject structs for each registered subject. And **_voters** is a mapping containing the Voter structs for each registered voter.

3. **Modifiers**:

The contract defines two modifiers, **onlyOwner** and **votingInProgress**. **onlyOwner** checks if the sender is the owner of the contract, while **votingInProgress** checks if the voting period has not ended yet.

4. **Functions**:

The contract defines following public functions: **addSubject** is used to register a new subject, **addVoter** is used to add a new voter, **getSubjects** is used to return the subject keys, **getSubject** is used to retrieve the data of a subject, **votePositive** and **voteNegative** are used to cast a positive or negative vote, and **getRemainingTime** is used to retrieve the remaining time before the voting period ends, **getResults** is used to get the voting results, sorted descending by votes.

It also defines an internal function, **checkIfCanVote**, which is used to check if a voter can cast a vote. Moreover the contract includes a **quicksort** function, which sorts an array of subjects based on the number of votes. This is used to sort the subjects in

order to return the results in the **getResults** function. The **partition** function is a helper function used in the quicksort algorithm.

5. **Constructor**:

The constructor sets the **_owner** address to the sender of the contract deployment, and sets the **_startDate** variable to the current block timestamp.

Slyther summary:

```
(base) herman@herman-GL502VMK:~/NI-BLO/du2/NIE-BLO-Vojtech-Drska-main/contracts$ slither . --print contract-summary

+ Contract D21 (Most derived contract)
  - From D21
    - addSubject(string) (external)
    - addVoter(address) (external)
    - checkIfCanVote(address,bool) (private)
    - constructor() (public)
    - getRemainingTime() (external)
    - getResults() (external)
    - getSubject(address) (external)
    - getSubjects() (external)
    - partition(IVoteD21.Subject[],uint256,uint256) (private)
    - quicksort(IVoteD21.Subject[],uint256,uint256) (private)
    - voteNegative(address) (external)
    - votePositive(address) (external)

+ Contract IVoteD21
  - From IVoteD21
    - addSubject(string) (external)
    - addVoter(address) (external)
    - getRemainingTime() (external)
    - getResults() (external)
    - getSubject(address) (external)
    - getSubjects() (external)
    - voteNegative(address) (external)
    - votePositive(address) (external)
```

```
(base) herman@herman-GL502VMK:~/NI-BLO/du2/NIE-BLO-Vojtech-Drska-main/contracts$ slither . --print human-summary

Compiled with solc
Number of lines: 169 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 2 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 5
Number of low issues: 1
Number of medium issues: 0
Number of high issues: 0


+------+-------------+------+------------+--------------+----------+
| Name | # functions | ERCS | ERC20 info | Complex code | Features |
+------+-------------+------+------------+--------------+----------+
| D21  |      21     |      |            |     Yes      |          |
+------+-------------+------+------------+--------------+----------+

Compiled with solc
Number of lines: 25 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 1 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 2
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0


+----------+-------------+------+------------+--------------+----------+
|   Name   | # functions | ERCS | ERC20 info | Complex code | Features |
+----------+-------------+------+------------+--------------+----------+
| IVoteD21 |      8      |      |            |      No      |          |
+----------+-------------+------+------------+--------------+----------+
. analyzed (3 contracts)
```

```
(base) herman@herman-GL502VMK:~/NI-BLO/du2/NIE-BLO-Vojtech-Drska-main/contracts$ slither . --print modifiers

Contract D21
+-------------------------------------+----------------------+
|               Function              |      Modifiers       |
+-------------------------------------+----------------------+
|              addSubject             |          []          |
|               addVoter              |          []          |
|             getSubjects             |          []          |
|              getSubject             |          []          |
|             votePositive            |          []          |
|             voteNegative            |          []          |
|           getRemainingTime          |          []          |
|              getResults             |          []          |
|             constructor             |          []          |
|              addSubject             |          []          |
|               addVoter              |    ['onlyOwner']     |
|             getSubjects             |          []          |
|              getSubject             |          []          |
|             votePositive            | ['votingInProgress'] |
|             voteNegative            | ['votingInProgress'] |
|           getRemainingTime          |          []          |
|              getResults             |          []          |
|            checkIfCanVote           |          []          |
|              quicksort              |          []          |
|              partition              |          []          |
| slitherConstructorConstantVariables |          []          |
+-------------------------------------+----------------------+
```

```
(base) herman@herman-GL502VMK:~/NI-BLO/du2/NIE-BLO-Vojtech-Drska-main/contracts$ slither . --print require

Contract D21
+-------------------------------------+-----------------------------------------------------------------------------------------------------+
|               Function              |                                          require or assert                                          |
+-------------------------------------+-----------------------------------------------------------------------------------------------------+
|              addSubject             |                                                                                                     |
|               addVoter              |                                                                                                     |
|             getSubjects             |                                                                                                     |
|              getSubject             |                                                                                                     |
|             votePositive            |                                                                                                     |
|             voteNegative            |                                                                                                     |
|           getRemainingTime          |                                                                                                     |
|              getResults             |                                                                                                     |
|             constructor             |                                                                                                     |
|              addSubject             |                          require(bool,string)(bytes(name).length != 0,Empty name)                   |
|                                     |  require(bool,string)(bytes(_subjects[msg.sender].name).length == 0,Owner with given name exists)  |
|               addVoter              |              require(bool,string)(! _voters[addr].exists,Voter already registered)                  |
|                                     |                  require(bool,string)(msg.sender == _owner,Not owner)                               |
|             getSubjects             |                                                                                                     |
|              getSubject             |       require(bool,string)(bytes(_subjects[addr].name).length != 0,Subject doesn't exist)           |
|             votePositive            |  require(bool,string)(bytes(_subjects[subject_address].name).length != 0,Subject doesn't exist)     |
|                                     |           require(bool,string)(block.timestamp <= _startDate + DURATION,Voting ended)               |
|                                     |           require(bool,string)(_voters[msg.sender].exists == true,Voter doesn't exist)              |
|             voteNegative            |  require(bool,string)(bytes(_subjects[subject_address].name).length != 0,Subject doesn't exist)     |
|                                     |           require(bool,string)(block.timestamp <= _startDate + DURATION,Voting ended)               |
|                                     |           require(bool,string)(_voters[msg.sender].exists == true,Voter doesn't exist)              |
|           getRemainingTime          |                                                                                                     |
|              getResults             |                                                                                                     |
|            checkIfCanVote           |  require(bool,string)(bytes(_subjects[subject_address].name).length != 0,Subject doesn't exist)     |
|                                     |           require(bool,string)(_voters[msg.sender].exists == true,Voter doesn't exist)              |
|              quicksort              |                                                                                                     |
|              partition              |                                                                                                     |
| slitherConstructorConstantVariables |                                                                                                     |
+-------------------------------------+-----------------------------------------------------------------------------------------------------+
```

```
(base) herman@herman-GL502VMK:~/NI-BLO/du2/NIE-BLO-Vojtech-Drska-main/contracts$ slither . --print variable-order

D21:
+-----------------+--------------------------------------+------+--------+
|       Name      |                 Type                 | Slot | Offset |
+-----------------+--------------------------------------+------+--------+
| D21._subjectKeys |              address[]               |  0   |   0    |
|  D21._subjects   | mapping(address => IVoteD21.Subject) |  1   |   0    |
|   D21._voters    |    mapping(address => D21.Voter)     |  2   |   0    |
+-----------------+--------------------------------------+------+--------+
```

Trust Model

The trust model of this contract starts with the assumption that the contract creator, the _owner, and the voters are all valid and trustworthy actors. The owner of the contract is responsible for managing and administering the voting process. The owner is assumed to be valid and trusted, and is the only user who can set up and register voters.

The trust model of this contract also assumes that the voters will act according to the rules of the contract, and that they will not try to subvert the results of the voting process. The contract enforces this by restricting the type of votes that can be cast and the number of votes that can be cast.

Therefore, the contract does not provide any way for users to prove their identity or to verify that their votes are genuine.

Methodology

Both manual code and tool-based analysis to assess the security of the contract were used. First, a code review was conducted to investigate potential security vulnerabilities. Then, combination of automated and manual tests was used to detect any potential issues. Finally, the contract was deployed locally for further testing and validation.

1. **Technical specification/documentation** - The audit was done in accordance with the task use cases
2. **Tool-based analysis** -code examined with automated Solidity analysis tools and Woke.
3. **Manual code review** - the code is reviewed meticulously for security issues, redundancy, guidelines and the code architecture is analysed.
4. **Local deployment + hacking** - the contract was deployed locally and attempted to break the system.
5. **Unit and fuzzy testing** - carried out tests to ensure the system works correctly, and potentially compose additional unit or fuzzy tests.

Summary of findings

1. Tool-based analysis

Woke:

woke detect did not find any vulnerabilities.

Slyther detector:

L1:
Severity: Low
Confidence: Medium

D21.getRemainingTime() (D21.sol#75-82) uses timestamp for comparisons
Dangerous comparisons:
- time <= 0 (D21.sol#77)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Description:
Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.

Exploit Scenario:
An attacker could exploit this vulnerability by manipulating block.timestamp.

Recommendation:
Avoid relying on block.timestamp.

Solution:
Developers should use alternative techniques such as checking the block's number or the block's difficulty to ensure the code is not vulnerable to manipulation. Also use a trusted source of data for timestamp verification or use the new oracle services for timestamp verification.

I1:
Severity: Informational
Confidence: High

D21.checkIfCanVote(address,bool) (D21.sol#96-120) compares to a boolean constant:
-require(bool,string)(_voters[msg.sender].exists == true,Voter doesn't exist) (D21.sol#97)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Description
Detects the comparison to boolean constants.

Exploit Scenario:
Boolean constants can be used directly and do not need to be compare to true or false.

Recommendation and Solution:
Remove the equality to the boolean constant.

I2:

Severity: Informational
Confidence: High

Parameter D21.checkIfCanVote(address,bool).subject_address (D21.sol#96) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Description:
Solidity defines a naming convention that should be followed.

Rule exceptions:
Allow constant variable name/symbol/decimals to be lowercase (ERC20).
Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

Recommendation and Solution:
Follow the Solidity naming convention.

2. Local deployment

Contract was deployed locally using the Truffle Suite and tested it using the Remix debugger. This allowed to validate the contract's functionality, as well as test it for potential security issues.

3. Manual code review

A manual code review was conducted to identify any potential security vulnerabilities in the code. During this process, I looked for common vulnerabilities such as re-entrancy, overflow, and denial-of-service attacks. The GPT3 (Generative Pre-trained Transformer) algorithm was also used as an experiment to find vulnerabilities in the process.

M1:

Description:

The **getSubject** function does not check if the address passed to it is empty or not.

Exploit scenario:

The attacker can exploit this vulnerability by passing an empty address to the getSubject function. This will cause the require statement in the function to fail and the function will abort.

Recommendation:

Always check the data passed to the contracts to protect from malicious intent.

Solution:

The solution to this vulnerability is to check if the address passed to the getSubject function is empty or not. This can be done by adding an additional require statement to the function. For example:

require(addr != address(0), “Invalid address”);

M2:

Description:

The contract has an overflow vulnerability in the **quicksort**() function. This function performs partitioning of an array with no checks for overflow and underflow when manipulating the array, resulting in a potentially exploitable overflow issue when the array has more than 2^256 elements.

Exploit Scenario:

An attacker can exploit this overflow vulnerability by adding an array of more than 2^256 elements to the array being sorted by the quicksort() function, which would cause the function to overflow and may allow the attacker to gain control over the execution of the contract.

Recommendation:

It is recommended that the quicksort() function be modified to check for overflows and underflows when manipulating the array.

Solution:

The quicksort() function can be modified to use the SafeMath library to ensure that overflows and underflows are avoided when manipulating the array. Additionally, the code can be refactored to use a more efficient algorithm to avoid the need to sort such a large array.

M3:

Description:

The Denial of Service (**DoS**) vulnerability in the above code is due to the usage of the quicksort algorithm which has a worst-case time-complexity of O(n^2). This means if the number of subjects for voting grows large enough, the quicksort algorithm will take a long time to finish sorting the array, causing the contract to be unresponsive for a long time.

Exploit Scenario:

An attacker can add a large number of subjects for voting, causing the quicksort algorithm to take a long time to sort the array. This will render the contract unresponsive for a long time, resulting in a Denial of Service (DoS) attack.

Recommendation:

The best way to avoid DoS attacks is to use a more efficient sorting algorithm, preferably one with a time complexity of O(n log n).

Solution:

Using an efficient sorting algorithm such as Merge Sort, Heap Sort or Quick Sort with a good pivot selection algorithm can help reduce the time complexity of the quicksort algorithm and thus prevent DoS attacks.

M2:

Description:

Re-entrancy vulnerability occurs when a function calls another function, and the called function calls the original function again before it is finished. This can cause an infinite loop of calls that depletes the original caller's resources or leads to unintended behavior.

Exploit Scenario:

A malicious user could exploit the re-entrancy vulnerability by calling the function **votePositive** or **voteNegative**. This would cause the function to call itself again and again until the malicious user’s resources are depleted, or the contract runs out of gas. An attacker calls the **votePositive** or **voteNegative** functions, which in turn calls the checkIfCanVote function. Within the **checkIfCanVote** function, the attacker calls the votePositive or voteNegative functions again, resulting in an infinite loop. The malicious user could also exploit the vulnerability by calling the function with a maliciously crafted address that points to a vulnerable contract.

Recommendation:

To prevent a re-entrancy attack, it is recommended to always use the "check-effects-interactions" pattern when making a call to another contract, and to lock the caller’s resources to prevent them from running out of gas.

Solution:

A contract can be made re-entrancy-safe by using the “checks-effects-interactions” pattern. This pattern requires that the contract checks the input and data, makes all the necessary calculations, and only then proceeds with the modification of the state

variables. This way, if any recursive calls are made, the vulnerable functions will not be executed until all the input checks and calculations have been completed. Additionally, it is best to use the “mutex” pattern when modifying the state variables, which prevents any other transactions from being executed until the current one is completed.

```
modifier noReentrant() {
        require(!locked, "No re-entrancy");
        locked = true;
        _;
        locked = false;
    }
```

And then added to functions:

```
function votePositive(address addr) external noReentrant votingInProgress;
function voteNegative(address addr) external noReentrant votingInProgress;
```

M5:

Description:

The **addSubject** function may be vulnerable to **re-entrancy** attacks. The function allows attackers to call the **addSubject** function within the **addSubject** function, effectively calling it twice and creating multiple subjects with the same name.

Exploit Scenario:

An attacker will first call the addSubject function to create a new subject with a name of their choosing. They will then call the addSubject function again with the same name, this time within the original function call, which will create a duplicate subject. This creates a situation in which the addSubject function fails to keep track of valid subjects and allows attackers to create multiple subjects with the same name.

Recommendation:

In order to protect against re-entrancy attacks, it is recommended to implement measures such as the "checks-effects-interactions" pattern, as well as check for already existing subjects with the same name before allowing the creation of a new subject.

Solution:

In order to fix this vulnerability, the following change can be made to the addSubject function:

```
function addSubject(string memory name) external {
    require(bytes(name).length != 0, "Empty name");
    require(bytes(_subjects[msg.sender].name).length == 0, "Owner with given name exists");
    // Check that a subject with this name does not already exist
    for (uint i = 0; i < _subjectKeys.length; i++) {
        require(name != _subjects[_subjectKeys[i]].name, "Subject with this name already exists");
    }
    _subjects[msg.sender] = Subject(name, 0);
    _subjectKeys.push(msg.sender);
}
```

4. Fuzz tests

Fuzz tests were written to verify that the system could safely handle unexpected input. These tests were written in Python and simulated various malicious user behaviors and input, such as trying to register a subject multiple times, giving more than two positive votes to the same subject, etc. The author of the code wrote some tests that are listed in the repository.

5. Executive summary

This report presents an audit of the Janečkova metoda D21 voting system implemented on the Ethereum blockchain. The code was analyzed using a combination of manual code analysis, tool-based analysis, and local deployment tests. This audit identified several security vulnerabilities, including an overflow vulnerability, re-entrancy vulnerability, and denial-of-service vulnerability. Additionally, the audit found that the contract was not following the Solidity naming conventions and that the getSubject function did not check for empty addresses. Recommendations for mitigating these vulnerabilities and improving the security of

the contract were provided. Finally, fuzz tests were written to verify that the system could safely handle unexpected input.