Decent Labs - Fractal Contracts

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 17th, 2023 - April 28th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/10/2023	István Böhm
0.2	Document Updates	04/19/2023	István Böhm
0.3	Document Updates	04/24/2023	Roberto Reigada
0.4	Document Updates	04/28/2023	István Böhm
0.5	Draft Review	04/28/2023	Ataberk Yavuzer
0.6	Draft Review	04/28/2023	Piotr Cielas
0.7	Draft Review	04/28/2023	Gabi Urrutia
1.0	Remediation Plan	05/03/2023	István Böhm
1.1	Remediation Plan Review	05/04/2023	Ataberk Yavuzer
1.2	Remediation Plan Review	05/04/2023	Piotr Cielas
1.3	Remediation Plan Review	05/05/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com
István Böhm	Halborn	Istvan.Bohm@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

The Decent Labs's Fractal Contracts is a collection of smart contracts that allow for composable governance.

Decent Labs engaged Halborn to conduct a security audit on their smart contracts beginning on April 17th, 2023 and ending on April 28th, 2023. The security assessment was scoped to the smart contracts provided in the decent-dao/fractal-contracts GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 AUDIT SUMMARY

The team at Halborn was provided 9 days for the engagement and assigned two full-time security engineers to audit the security of the smart contracts in scope. Security engineers are blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the audits is to:

- Identify potential security issues within the smart contracts

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which have been addressed and acknowledged by Decent Labs.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet analysis (Brownie, Foundry Remix IDE)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 Exploitability

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 Impact

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 Severity Coefficient

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. Fractal Contracts:

- Repository: decent-dao/fractal-contracts
- Initial Commit ID: e3c4132
- LinearERC20 Wrapped Voting Update Commit ID: f113f23
- Fixed Commit ID: 54290ad

Out-of-scope:
- Third-party libraries and dependencies.
- Economic attacks.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF DISABLEINITIALIZERS CALL TO PREVENT UNINITIALIZED CONTRACTS	Informational (1.7)	SOLVED - 05/03/2023
SINGLE STEP OWNERSHIP TRANSFER PROCESS	Informational (1.7)	ACKNOWLEDGED
MISSING ACCESS CONTROL IN EVENT EMITTING FUNCTIONS	Informational (1.0)	ACKNOWLEDGED
UNUSED VARIABLES IN LINEARERC20VOTING STRATEGY	Informational (0.0)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) LACK OF DISABLEINITIALIZERS CALL TO PREVENT UNINITIALIZED CONTRACTS - INFORMATIONAL (1.7)

Description:

It was identified that the initializer functions are not disabled in the Fractal implementation contracts. Uninitialized contracts can be initialized by someone else to take over them.

In the latest version (4.8.0), this is done by calling the _disableInitializers function in the constructor.

Note that the Fractal contracts are designed to be deployed using the ERC-1167 Minimal Proxy Contract (Cloning) standard, where the implementation contracts are only utilized through the proxies. However, it is still considered to be a best practice to disable the initializer functions to prevent any accidental misuse.

Code Location:

For example, before initialization, the setUp function of an Azorius contract can be called by anyone to take over the contract:

Listing 1: contracts/azorius/Azorius.sol

```
    function setUp(bytes memory initializeParams) public override
 ↳ initializer {
        (
            address _owner,
            address _avatar,
            address _target,
            address[] memory _strategies,    // enabled
 ↳ BaseStrategies
            uint32 _timelockPeriod,         // initial
 ↳ timelockPeriod
```

```
            uint32 _executionPeriod        // initial executionPeriod
        ) = abi.decode(
            initializeParams,
            (address, address, address, address[], uint32, uint32)
        );
        __Ownable_init();
        avatar = _avatar;
        target = _target;
        _setUpStrategies(_strategies);
        transferOwnership(_owner);
        _updateTimelockPeriod(_timelockPeriod);
        _updateExecutionPeriod(_executionPeriod);

        emit AzoriusSetUp(msg.sender, _owner, _avatar, _target);
    }
```

The Azorius contract in the test files is deployed through a two-step process. The first step deploys the contract, while the second step initializes it. However, this approach creates an opportunity for a malicious user to frontrun the second step and initialize the contract themselves, effectively taking control of it.

Listing 2: test/test/Azorius-LinearERC20Voting.test.ts (Lines 151,165)

```
    azorius = await new Azorius__factory(deployer).deploy();

    const azoriusSetupData = abiCoder.encode(
      ["address", "address", "address", "address[]", "uint32", "uint32"],
      [
        gnosisSafeOwner.address,
        gnosisSafe.address,
        gnosisSafe.address,
        [],
        60, // timelock period in blocks
        60, // execution period in blocks
      ]
    );

    await azorius.setUp(azoriusSetupData);
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (1.7)

Recommendation:

Consider adding a constructor to the initializable contracts and calling the _disableInitializers function in them:

Listing 3: Initialization Example

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();
}
```

Remediation Plan:

SOLVED: The Decent Labs team solved the issue in commit 54290ad by modifying the constructors to automatically mark the contracts as initialized when they are deployed.

4.2 (HAL-02) SINGLE STEP OWNERSHIP TRANSFER PROCESS - INFORMATIONAL (1.7)

Description:

It was identified that ownable Fractal contracts are inherited from OpenZeppelin's OwnableUpgradeable contract. Ownership of the contracts that are inherited from the OwnableUpgradeable contract can be lost, as their ownership can be transferred in a single-step process. The address that the ownership is changed to should be verified to be active or willing to act as the owner.

Code Location:

Single step ownership transfer process in the OwnableUpgradeable contract:

Listing 4: node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol

```
    function transferOwnership(address newOwner) public virtual
 ↳ onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the
 ↳  zero address");
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account
 ↳ (`newOwner`).
     * Internal function without access restriction.
     */
    function _transferOwnership(address newOwner) internal virtual
 ↳  {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (1.7)

Recommendation:

Consider using the Ownable2StepUpgradeable library over the OwnableUpgradeable library.

Alternatively, it is recommended to split the current ownership transfer process into two steps. For example, the first one is to call the requestTransferOwnership function, which proposes a new owner for the protocol, and the second, the new owner accepts the proposal by calling the acceptsTransferOwnership function.

Note that for contracts intended to be owned by other contracts, additional logic is required to be implemented in the parent contracts to be compatible with the two-step ownership transfer process.

Remediation Plan:

ACKNOWLEDGED: The Decent Labs team acknowledged this finding. Contract owners in the Azorius system are intended to be other contracts, and implementing a two-step ownership process on the parent contract would not be feasible.

4.3 (HAL-03) MISSING ACCESS CONTROL IN EVENT EMITTING FUNCTIONS - INFORMATIONAL (1.0)

Description:

It was identified that the updateDAOName and declareSubDAO functions within the FractalRegistry contract, as well as the updateValues function in the KeyValuePairs contract, can be called by anyone. Although these functions do not alter the state variables of the contracts directly, this behavior places the responsibility to validate the msg.sender to the services listening to these events, A malicious actor may also be able to disrupt the services by generating large amounts of events.

Code Location:

Listing 5: fractal-contracts/contracts/FractalRegistry.sol

```
    function updateDAOName(string memory _name) external {
        emit FractalNameUpdated(msg.sender, _name);
    }

    /** @inheritdoc IFractalRegistry*/
    function declareSubDAO(address _subDAOAddress) external {
        emit FractalSubDAODeclared(msg.sender, _subDAOAddress);
    }
```

Listing 6: fractal-contracts/contracts/KeyValuePairs.sol

```
    function updateValues(string[] memory _keys, string[] memory
↳ _values) external {

        uint256 keyCount = _keys.length;

        if (keyCount != _values.length)
            revert IncorrectValueCount();

        for (uint256 i; i < keyCount; ) {
            emit ValueUpdated(msg.sender, _keys[i], _values[i]);
            unchecked {
                ++i;
            }
        }
    }
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:L/A:N/D:N/Y:N/R:N/S:C (1.0)

Recommendation:

It is recommended to implement access control mechanisms for the functions to ensure that only authorized callers can emit events.

Remediation Plan:

ACKNOWLEDGED: The Decent Labs team acknowledged this finding. The msg.sender of the events will be validated off-chain.

4.4 (HAL-04) UNUSED VARIABLES IN LINEARERC20VOTING STRATEGY - INFORMATIONAL (0.0)

Description:

In the Azorius contract’s submitProposal function, the values of the txHashes and _data variables are passed to the configured voting strategy’s initializeProposal call. However, it was identified that these values are not used in the LinearERC20Voting contract, which is currently the only voting strategy of the Fractal Contracts. Simplifying the code can reduce the contract’s size, leading to potential gas cost savings and decreased complexity.

Code Location:

The values of the txHashes and _data variables passed to the configured voting strategy’s initializeProposal call:

Listing 7: contracts/azorius/Azorius.sol (Lines 149-161,169)

```
    function submitProposal(
        address _strategy,
        bytes memory _data,
        Transaction[] calldata _transactions,
        string calldata _metadata
    ) external {
        if (!isStrategyEnabled(_strategy)) revert StrategyDisabled
 ↳ ();
        if (!IBaseStrategy(_strategy).isProposer(msg.sender))
            revert InvalidProposer();

        bytes32[] memory txHashes = new bytes32[](_transactions.
 ↳ length);
        uint256 transactionsLength = _transactions.length;
        for (uint256 i; i < transactionsLength; ) {
            txHashes[i] = getTxHash(
                _transactions[i].to,
                _transactions[i].value,
                _transactions[i].data,
                _transactions[i].operation
            );
            unchecked {
                ++i;
            }
        }

        proposals[totalProposalCount].strategy = _strategy;
        proposals[totalProposalCount].txHashes = txHashes;
        proposals[totalProposalCount].timelockPeriod =
 ↳ timelockPeriod;
        proposals[totalProposalCount].executionPeriod =
 ↳ executionPeriod;

        IBaseStrategy(_strategy).initializeProposal(
            abi.encode(totalProposalCount, txHashes, _data)
        );
```

These values are not used in the initializeProposal function of the LinearERC20Voting contract:

Listing 8: contracts/azorius/LinearERC20Voting

```
    function initializeProposal(bytes memory _data) external
 ↳ virtual override onlyAzorius {
        uint32 proposalId = abi.decode(_data, (uint32));
        uint32 _votingEndBlock = uint32(block.number) +
 ↳ votingPeriod;

        proposalVotes[proposalId].votingEndBlock = _votingEndBlock
 ↳ ;
        proposalVotes[proposalId].votingStartBlock = uint32(block.
 ↳ number);

        emit ProposalInitialized(proposalId, _votingEndBlock);
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider reviewing the function and removing any unnecessary functionalities.

Remediation Plan:

ACKNOWLEDGED: The Decent Labs team acknowledged this finding. The current logic is an intentional choice to enable compatibility with different governance extensions in the future. Added additional documentation to the source code on this behavior in commit f113f23.

MANUAL TESTING

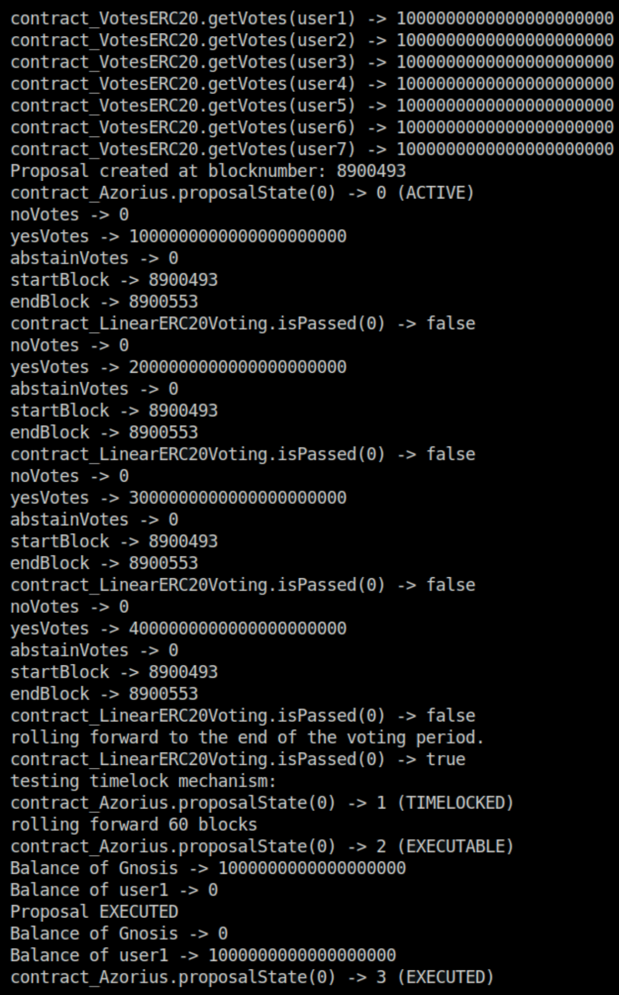
5.1 INTRODUCTION

Halborn conducted a comprehensive manual assessment of the smart contracts in scope in a local test environment, examining them for potential logic flaws and vulnerabilities. The Fractal contracts were deployed in a local Foundry test environment to facilitate this evaluation.

5.2 EXAMPLE SCENARIOS

EXECUTED PROPOSAL:

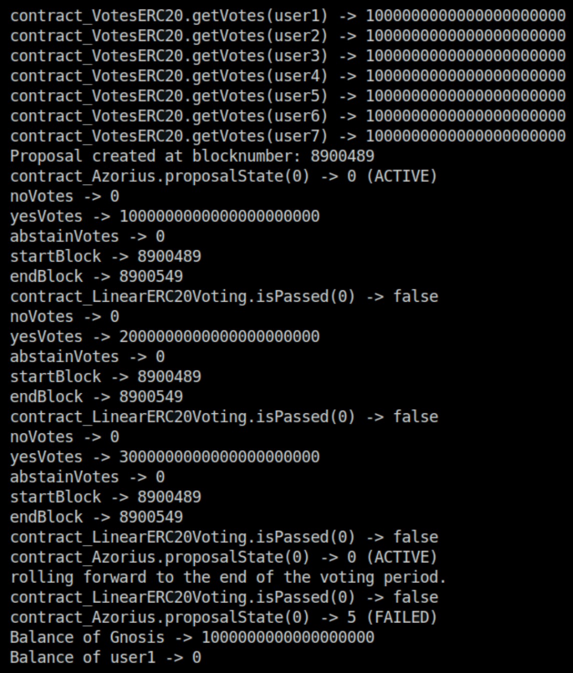
The state changes of a proof of concept “send one ether” proposal are inspected from the start of the proposal to its successful execution:



```
contract_VotesERC20.getVotes(user1) -> 1000000000000000000000
contract_VotesERC20.getVotes(user2) -> 1000000000000000000000
contract_VotesERC20.getVotes(user3) -> 1000000000000000000000
contract_VotesERC20.getVotes(user4) -> 1000000000000000000000
contract_VotesERC20.getVotes(user5) -> 1000000000000000000000
contract_VotesERC20.getVotes(user6) -> 1000000000000000000000
contract_VotesERC20.getVotes(user7) -> 1000000000000000000000
Proposal created at blocknumber: 8900493
contract_Azorius.proposalState(0) -> 0 (ACTIVE)
noVotes -> 0
yesVotes -> 1000000000000000000000
abstainVotes -> 0
startBlock -> 8900493
endBlock -> 8900553
contract_LinearERC20Voting.isPassed(0) -> false
noVotes -> 0
yesVotes -> 2000000000000000000000
abstainVotes -> 0
startBlock -> 8900493
endBlock -> 8900553
contract_LinearERC20Voting.isPassed(0) -> false
noVotes -> 0
yesVotes -> 3000000000000000000000
abstainVotes -> 0
startBlock -> 8900493
endBlock -> 8900553
contract_LinearERC20Voting.isPassed(0) -> false
noVotes -> 0
yesVotes -> 4000000000000000000000
abstainVotes -> 0
startBlock -> 8900493
endBlock -> 8900553
contract_LinearERC20Voting.isPassed(0) -> false
rolling forward to the end of the voting period.
contract_LinearERC20Voting.isPassed(0) -> true
testing timelock mechanism:
contract_Azorius.proposalState(0) -> 1 (TIMELOCKED)
rolling forward 60 blocks
contract_Azorius.proposalState(0) -> 2 (EXECUTABLE)
Balance of Gnosis -> 1000000000000000000
Balance of user1 -> 0
Proposal EXECUTED
Balance of Gnosis -> 0
Balance of user1 -> 1000000000000000000
contract_Azorius.proposalState(0) -> 3 (EXECUTED)
```

FAILED PROPOSAL:

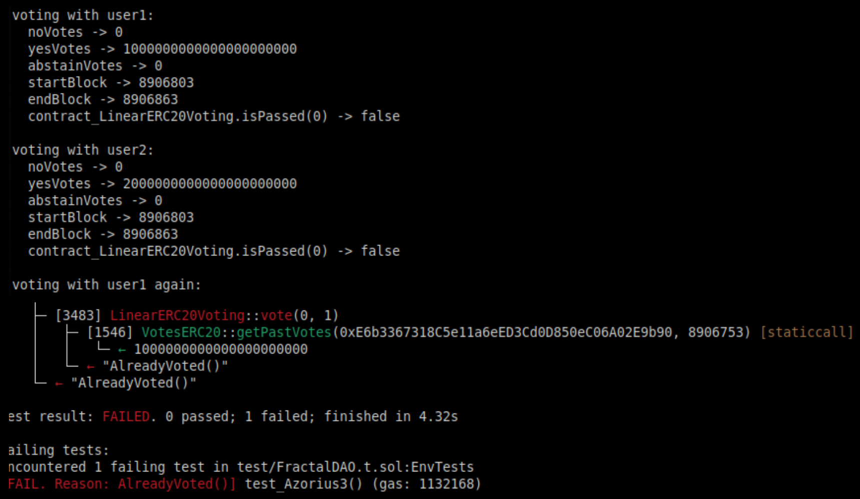
Without enough upvotes until expiration, the same proposal failed, and the transaction did not execute:



```
contract_VotesERC20.getVotes(user1) -> 1000000000000000000000
contract_VotesERC20.getVotes(user2) -> 1000000000000000000000
contract_VotesERC20.getVotes(user3) -> 1000000000000000000000
contract_VotesERC20.getVotes(user4) -> 1000000000000000000000
contract_VotesERC20.getVotes(user5) -> 1000000000000000000000
contract_VotesERC20.getVotes(user6) -> 1000000000000000000000
contract_VotesERC20.getVotes(user7) -> 1000000000000000000000
Proposal created at blocknumber: 8900489
contract_Azorius.proposalState(0) -> 0 (ACTIVE)
noVotes -> 0
yesVotes -> 1000000000000000000000
abstainVotes -> 0
startBlock -> 8900489
endBlock -> 8900549
contract_LinearERC20Voting.isPassed(0) -> false
noVotes -> 0
yesVotes -> 2000000000000000000000
abstainVotes -> 0
startBlock -> 8900489
endBlock -> 8900549
contract_LinearERC20Voting.isPassed(0) -> false
noVotes -> 0
yesVotes -> 3000000000000000000000
abstainVotes -> 0
startBlock -> 8900489
endBlock -> 8900549
contract_LinearERC20Voting.isPassed(0) -> false
contract_Azorius.proposalState(0) -> 0 (ACTIVE)
rolling forward to the end of the voting period.
contract_LinearERC20Voting.isPassed(0) -> false
contract_Azorius.proposalState(0) -> 5 (FAILED)
Balance of Gnosis -> 1000000000000000000
Balance of user1 -> 0
```

VOTING TWICE ON THE SAME PROPOSAL:

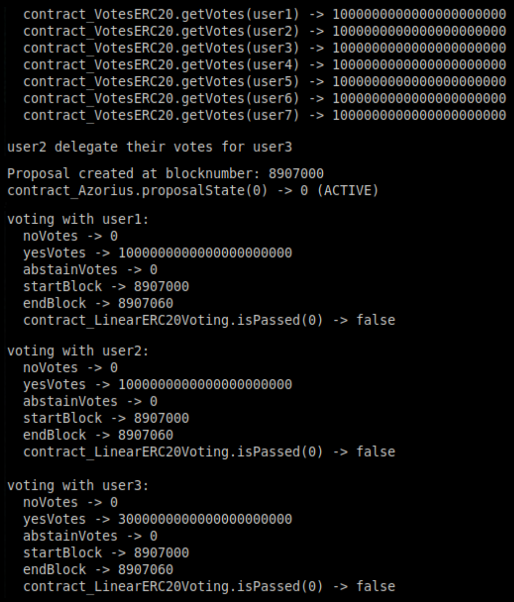
The audit also included manual testing of the strategies. For example, it verified that users cannot vote twice with the same account:



```
voting with user1:
  noVotes -> 0
  yesVotes -> 1000000000000000000000
  abstainVotes -> 0
  startBlock -> 8906803
  endBlock -> 8906863
  contract_LinearERC20Voting.isPassed(0) -> false

voting with user2:
  noVotes -> 0
  yesVotes -> 2000000000000000000000
  abstainVotes -> 0
  startBlock -> 8906803
  endBlock -> 8906863
  contract_LinearERC20Voting.isPassed(0) -> false

voting with user1 again:

    ├─ [3483] LinearERC20Voting::vote(0, 1)
    │   ├─ [1546] VotesERC20::getPastVotes(0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90, 8906753) [staticcall]
    │   │   └─ ← 1000000000000000000000
    │   └─ ← "AlreadyVoted()"
    └─ ← "AlreadyVoted()"

est result: FAILED. 0 passed; 1 failed; finished in 4.32s

ailing tests:
ncountered 1 failing test in test/FractalDAO.t.sol:EnvTests
FAIL. Reason: AlreadyVoted()] test_Azorius3() (gas: 1132168)
```

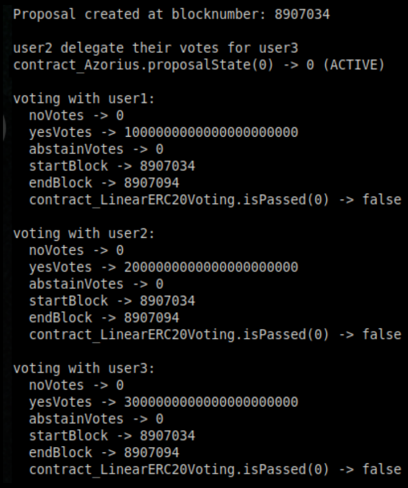
DELEGATING VOTES:

The delegation of voting power underwent testing as well. It was observed that if user2 delegated their voting tokens before the proposal submission, they were still able to execute the voting function without any error. However, their action did not change the voting position, and user3 was able to cast a vote using both their tokens and those previously delegated by user2:



```
  contract_VotesERC20.getVotes(user1) -> 1000000000000000000000
  contract_VotesERC20.getVotes(user2) -> 1000000000000000000000
  contract_VotesERC20.getVotes(user3) -> 1000000000000000000000
  contract_VotesERC20.getVotes(user4) -> 1000000000000000000000
  contract_VotesERC20.getVotes(user5) -> 1000000000000000000000
  contract_VotesERC20.getVotes(user6) -> 1000000000000000000000
  contract_VotesERC20.getVotes(user7) -> 1000000000000000000000

user2 delegate their votes for user3

Proposal created at blocknumber: 8907000
contract_Azorius.proposalState(0) -> 0 (ACTIVE)

voting with user1:
  noVotes -> 0
  yesVotes -> 1000000000000000000000
  abstainVotes -> 0
  startBlock -> 8907000
  endBlock -> 8907060
  contract_LinearERC20Voting.isPassed(0) -> false

voting with user2:
  noVotes -> 0
  yesVotes -> 1000000000000000000000
  abstainVotes -> 0
  startBlock -> 8907000
  endBlock -> 8907060
  contract_LinearERC20Voting.isPassed(0) -> false

voting with user3:
  noVotes -> 0
  yesVotes -> 3000000000000000000000
  abstainVotes -> 0
  startBlock -> 8907000
  endBlock -> 8907060
  contract_LinearERC20Voting.isPassed(0) -> false
```

It was also verified that delegating of voting tokens after the submission of the proposal did not affect the voting:



```
Proposal created at blocknumber: 8907034

user2 delegate their votes for user3
contract_Azorius.proposalState(0) -> 0 (ACTIVE)

voting with user1:
  noVotes -> 0
  yesVotes -> 1000000000000000000000
  abstainVotes -> 0
  startBlock -> 8907034
  endBlock -> 8907094
  contract_LinearERC20Voting.isPassed(0) -> false

voting with user2:
  noVotes -> 0
  yesVotes -> 2000000000000000000000
  abstainVotes -> 0
  startBlock -> 8907034
  endBlock -> 8907094
  contract_LinearERC20Voting.isPassed(0) -> false

voting with user3:
  noVotes -> 0
  yesVotes -> 3000000000000000000000
  abstainVotes -> 0
  startBlock -> 8907034
  endBlock -> 8907094
  contract_LinearERC20Voting.isPassed(0) -> false
```

AUTOMATED TESTING

6.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

contracts/ERC20Claim.sol

```
ERC20Claim.setUp(bytes)._childTokenFunder (contracts/ERC20Claim.sol#70) lacks a zero-check on :
                - funder = _childTokenFunder (contracts/ERC20Claim.sol#76)
ERC20Claim.setUp(bytes)._childERC20 (contracts/ERC20Claim.sol#72) lacks a zero-check on :
                - childERC20 = _childERC20 (contracts/ERC20Claim.sol#78)
ERC20Claim.setUp(bytes)._parentERC20 (contracts/ERC20Claim.sol#71) lacks a zero-check on :
                - parentERC20 = _parentERC20 (contracts/ERC20Claim.sol#79)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in ERC20Claim.claimTokens(address) (contracts/ERC20Claim.sol#100-110):
        External calls:
        - IERC20(childERC20).safeTransfer(claimer,amount) (contracts/ERC20Claim.sol#107)
        Event emitted after the call(s):
        - ERC20Claimed(parentERC20,childERC20,claimer,amount) (contracts/ERC20Claim.sol#109)
Reentrancy in ERC20Claim.setUp(bytes) (contracts/ERC20Claim.sol#66-97):
        External calls:
        - snapShotId = VotesERC20(_parentERC20).captureSnapShot() (contracts/ERC20Claim.sol#82)
        - IERC20(_childERC20).safeTransferFrom(_childTokenFunder,address(this),_parentAllocation) (contracts/ERC20Claim.sol#84-88)
        Event emitted after the call(s):
        - ERC20ClaimCreated(_parentERC20,_childERC20,_parentAllocation,snapShotId,_deadlineBlock) (contracts/ERC20Claim.sol#90-96)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

VotesERC20._burn(address,uint256) (contracts/VotesERC20.sol#70-75) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

contracts/ERC20FreezeVoting.sol

```
ERC20FreezeVoting.castFreezeVote() (contracts/ERC20FreezeVoting.sol#56-93) uses a dangerous strict equality:
        - userVotes == 0 (contracts/ERC20FreezeVoting.sol#69)
ERC20FreezeVoting.castFreezeVote() (contracts/ERC20FreezeVoting.sol#56-93) uses a dangerous strict equality:
        - userVotes == 0 (contracts/ERC20FreezeVoting.sol#85)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
```

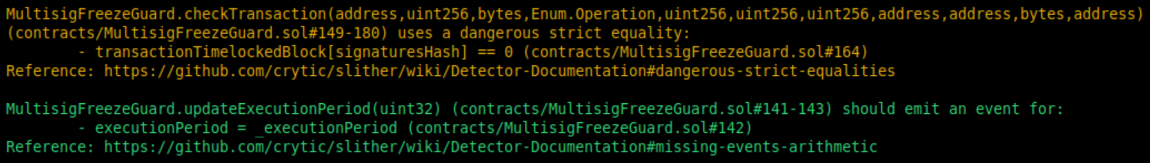
contracts/FractalModule.sol

```
FractalModule.removeControllers(address[]).i (contracts/FractalModule.sol#61) is a local variable never initialized
FractalModule.addControllers(address[]).i (contracts/FractalModule.sol#84) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

contracts/KeyValuePairs.sol

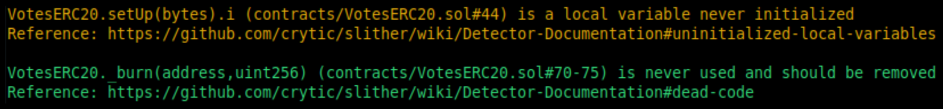
```
KeyValuePairs.updateValues(string[],string[]).i (contracts/KeyValuePairs.sol#24) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

contracts/MultisigFreezeGuard.sol



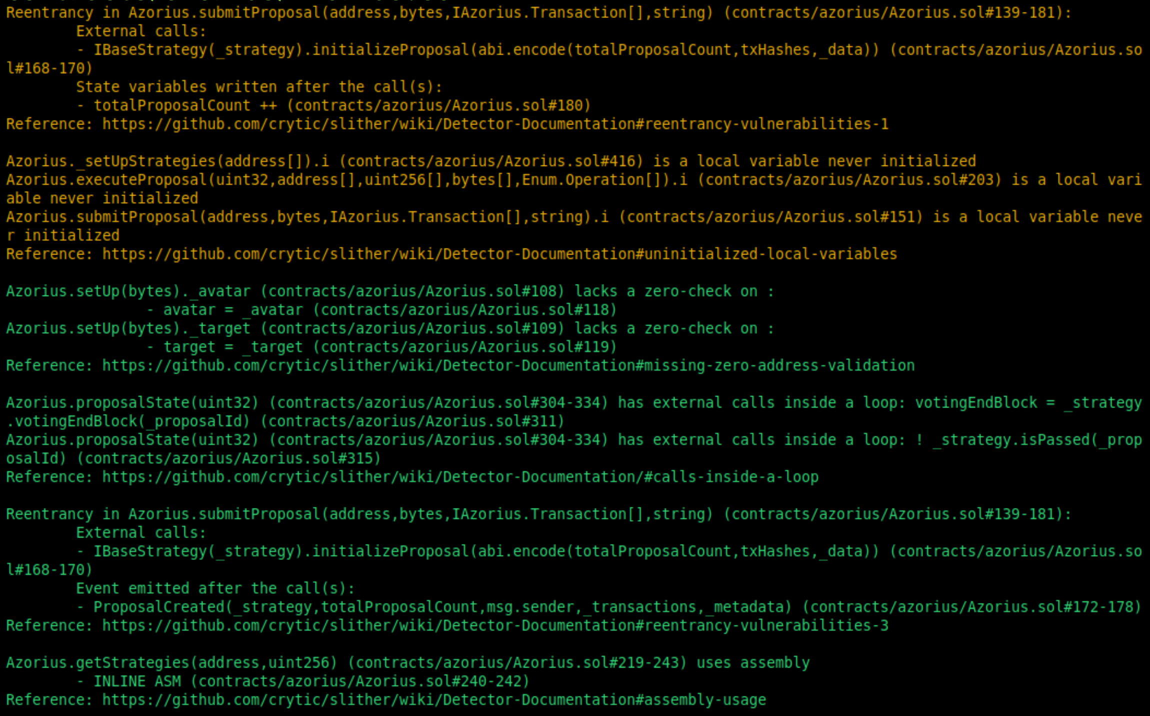
```
MultisigFreezeGuard.checkTransaction(address,uint256,bytes,Enum.Operation,uint256,uint256,uint256,address,address,bytes,address)
(contracts/MultisigFreezeGuard.sol#149-180) uses a dangerous strict equality:
        - transactionTimelockedBlock[signaturesHash] == 0 (contracts/MultisigFreezeGuard.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

MultisigFreezeGuard.updateExecutionPeriod(uint32) (contracts/MultisigFreezeGuard.sol#141-143) should emit an event for:
        - executionPeriod = _executionPeriod (contracts/MultisigFreezeGuard.sol#142)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
```

contracts/VotesERC20.sol



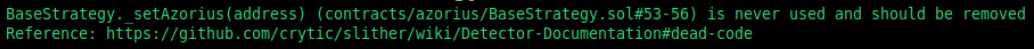
```
VotesERC20.setUp(bytes).i (contracts/VotesERC20.sol#44) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

VotesERC20._burn(address,uint256) (contracts/VotesERC20.sol#70-75) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

contracts/azorius/Azorius.sol



```
Reentrancy in Azorius.submitProposal(address,bytes,IAzorius.Transaction[],string) (contracts/azorius/Azorius.sol#139-181):
        External calls:
        - IBaseStrategy(_strategy).initializeProposal(abi.encode(totalProposalCount,txHashes,_data)) (contracts/azorius/Azorius.so
l#168-170)
        State variables written after the call(s):
        - totalProposalCount ++ (contracts/azorius/Azorius.sol#180)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

Azorius._setUpStrategies(address[]).i (contracts/azorius/Azorius.sol#416) is a local variable never initialized
Azorius.executeProposal(uint32,address[],uint256[],bytes[],Enum.Operation[]).i (contracts/azorius/Azorius.sol#203) is a local vari
able never initialized
Azorius.submitProposal(address,bytes,IAzorius.Transaction[],string).i (contracts/azorius/Azorius.sol#151) is a local variable neve
r initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

Azorius.setUp(bytes)._avatar (contracts/azorius/Azorius.sol#108) lacks a zero-check on :
                - avatar = _avatar (contracts/azorius/Azorius.sol#118)
Azorius.setUp(bytes)._target (contracts/azorius/Azorius.sol#109) lacks a zero-check on :
                - target = _target (contracts/azorius/Azorius.sol#119)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Azorius.proposalState(uint32) (contracts/azorius/Azorius.sol#304-334) has external calls inside a loop: votingEndBlock = _strategy
.votingEndBlock(_proposalId) (contracts/azorius/Azorius.sol#311)
Azorius.proposalState(uint32) (contracts/azorius/Azorius.sol#304-334) has external calls inside a loop: ! _strategy.isPassed(_prop
osalId) (contracts/azorius/Azorius.sol#315)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Reentrancy in Azorius.submitProposal(address,bytes,IAzorius.Transaction[],string) (contracts/azorius/Azorius.sol#139-181):
        External calls:
        - IBaseStrategy(_strategy).initializeProposal(abi.encode(totalProposalCount,txHashes,_data)) (contracts/azorius/Azorius.so
l#168-170)
        Event emitted after the call(s):
        - ProposalCreated(_strategy,totalProposalCount,msg.sender,_transactions,_metadata) (contracts/azorius/Azorius.sol#172-178)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Azorius.getStrategies(address,uint256) (contracts/azorius/Azorius.sol#219-243) uses assembly
        - INLINE ASM (contracts/azorius/Azorius.sol#240-242)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

contracts/azorius/BaseStrategy.sol



```
BaseStrategy._setAzorius(address) (contracts/azorius/BaseStrategy.sol#53-56) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

As a result of the tests carried out with the Slither tool, some results were obtained and these results were reviewed by Halborn. Based on the results reviewed, vulnerabilities were determined to be false positives and these results were not included in the report.

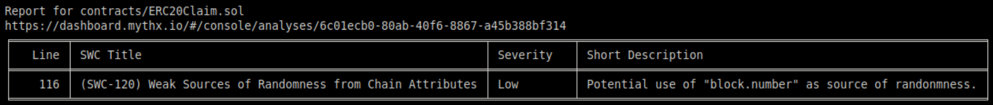
6.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:

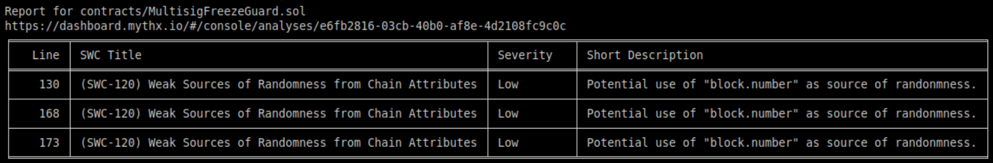
contracts/ERC20Claim.sol



```
Report for contracts/ERC20Claim.sol
https://dashboard.mythx.io/#/console/analyses/6c01ecb0-80ab-40f6-8867-a45b388bf314
```

Line	SWC Title	Severity	Short Description
116	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

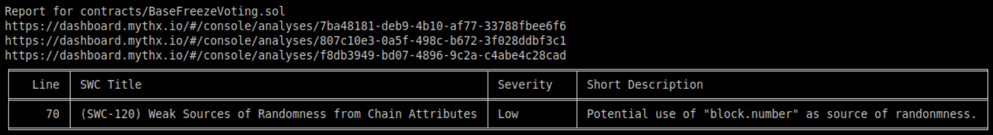
contracts/MultisigFreezeGuard.sol



```
Report for contracts/MultisigFreezeGuard.sol
https://dashboard.mythx.io/#/console/analyses/e6fb2816-03cb-40b0-af8e-4d2108fc9c0c
```

Line	SWC Title	Severity	Short Description
130	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
168	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
173	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

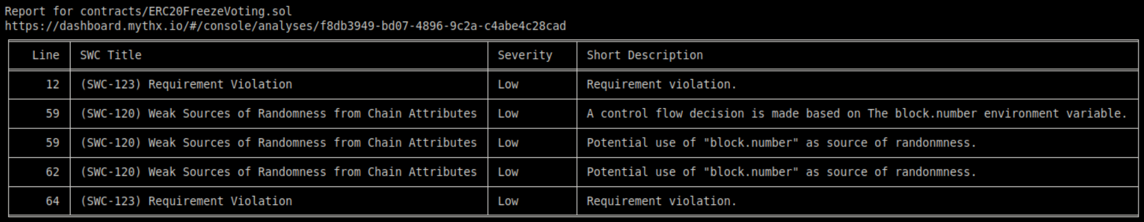
contracts/BaseFreezeVoting.sol



```
Report for contracts/BaseFreezeVoting.sol
https://dashboard.mythx.io/#/console/analyses/7ba48181-deb9-4b10-af77-33788fbee6f6
https://dashboard.mythx.io/#/console/analyses/807c10e3-0a5f-498c-b672-3f028ddbf3c1
https://dashboard.mythx.io/#/console/analyses/f8db3949-bd07-4896-9c2a-c4abe4c28cad
```

Line	SWC Title	Severity	Short Description
70	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

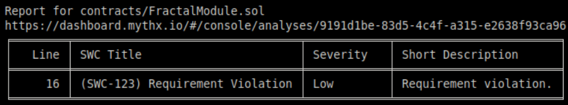
contracts/ERC20FreezeVoting.sol



```
Report for contracts/ERC20FreezeVoting.sol
https://dashboard.mythx.io/#/console/analyses/f8db3949-bd07-4896-9c2a-c4abe4c28cad
```

Line	SWC Title	Severity	Short Description
12	(SWC-123) Requirement Violation	Low	Requirement violation.
59	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	A control flow decision is made based on The block.number environment variable.
59	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
62	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
64	(SWC-123) Requirement Violation	Low	Requirement violation.

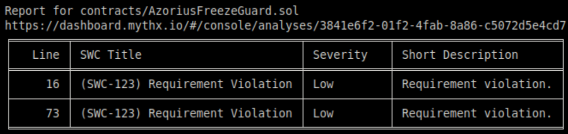
contracts/FractalModule.sol



Report for contracts/FractalModule.sol
https://dashboard.mythx.io/#/console/analyses/9191d1be-83d5-4c4f-a315-e2638f93ca96

Line	SWC Title	Severity	Short Description
16	(SWC-123) Requirement Violation	Low	Requirement violation.

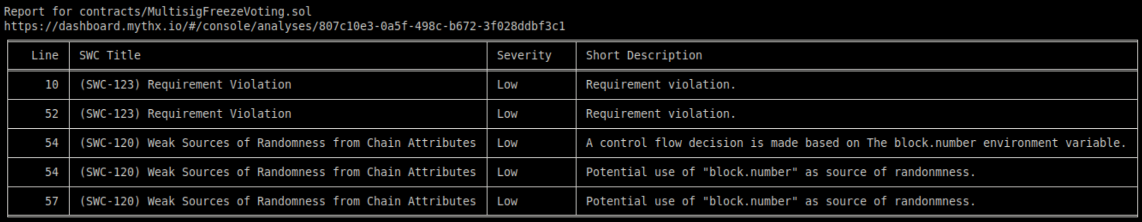
contracts/AzoriusFreezeGuard.sol



Report for contracts/AzoriusFreezeGuard.sol
https://dashboard.mythx.io/#/console/analyses/3841e6f2-01f2-4fab-8a86-c5072d5e4cd7

Line	SWC Title	Severity	Short Description
16	(SWC-123) Requirement Violation	Low	Requirement violation.
73	(SWC-123) Requirement Violation	Low	Requirement violation.

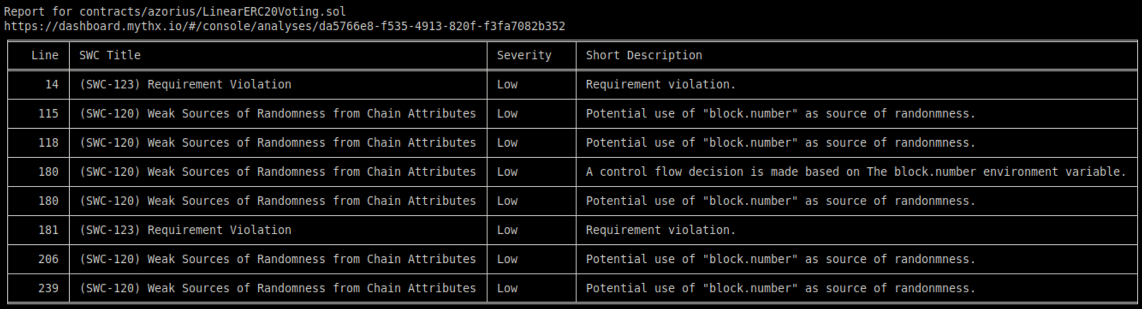
contracts/MultisigFreezeVoting.sol



Report for contracts/MultisigFreezeVoting.sol
https://dashboard.mythx.io/#/console/analyses/807c10e3-0a5f-498c-b672-3f028ddbf3c1

Line	SWC Title	Severity	Short Description
10	(SWC-123) Requirement Violation	Low	Requirement violation.
52	(SWC-123) Requirement Violation	Low	Requirement violation.
54	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	A control flow decision is made based on The block.number environment variable.
54	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
57	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

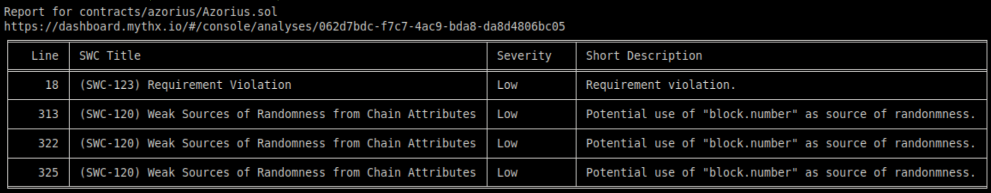
contracts/azorius/LinearERC20Voting.sol



Report for contracts/azorius/LinearERC20Voting.sol
https://dashboard.mythx.io/#/console/analyses/da5766e8-f535-4913-820f-f3fa7082b352

Line	SWC Title	Severity	Short Description
14	(SWC-123) Requirement Violation	Low	Requirement violation.
115	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
118	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
180	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	A control flow decision is made based on The block.number environment variable.
180	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
181	(SWC-123) Requirement Violation	Low	Requirement violation.
206	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
239	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

contracts/azorius/Azorius.sol



Report for contracts/azorius/Azorius.sol
https://dashboard.mythx.io/#/console/analyses/062d7bdc-f7c7-4ac9-bda8-da8d4806bc05

Line	SWC Title	Severity	Short Description
18	(SWC-123) Requirement Violation	Low	Requirement violation.
313	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
322	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
325	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

The findings obtained as a result of the MythX scan were examined, and the findings were not included in the report because they were false positive.

THANK YOU FOR CHOOSING

// HALBORN