Security Review Report
NM-0076 Gyroscope Governance

(August 15, 2023)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind on the Gyroscope's Gorvenance System. Gyroscope's mission is to build robust public infrastructure for DeFi. The central piece is a fully-backed stablecoin with all-weather reserves and autonomous price bounding. The Gyroscope governance system is designed to bring many different stakeholder groups into governance. Voting power is split between different voting vaults, which allocate voting power to different stakeholder groups. Furthermore, the quorum, time delay, and other parameters are set depending on the impact of the proposed changes.

The audited code consists of 3,119 lines of Solidity. The Gyroscope team provided one document to assist the audit presenting an overview of the contracts and how to run the test suite. The audit was supported by the documentation accessible from the Governance's documentation and continuous communication with the Gyroscope's team.

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contracts, and (d) creation of test cases. **Along this document, we report** 21 points of attention, where three are classified as `Critical`, six are classified as `High`, three are classified as `Medium`, one is classified as `Low`, and eight are classified as `Informational` or `Best Practices`. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope of this audit. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology adopted for this audit. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the compilation, tests, and automated tests. Section 9 concludes the document.

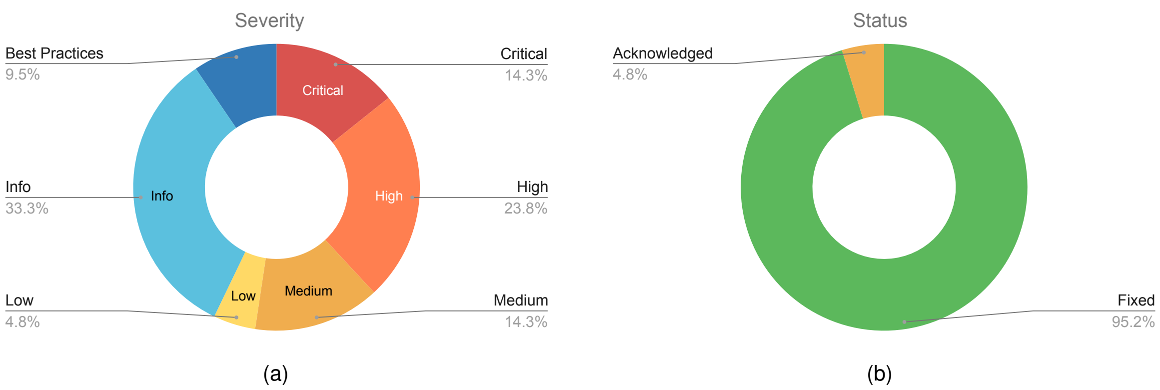

Fig. 1: Distribution of issues: Critical (3), **High** (5), **Medium** (3), **Low** (1), **Undetermined** (0), **Informational** (7), **Best Practices** (2).
Distribution of status: Fixed (20), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Mar. 27, 2023
Final Report	August 15, 2023
Methods	Manual Review, Automated Analysis
Repository	Gyroscope Governance
Commit Hash (Initial Audit)	93d75c7565b02dd53b79dc56e2412472e37ac77c
Documentation	README
Documentation Assessment	Medium
Test Suite Assessment	Medium

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	contracts/WrappedERC20WithEMA.sol	72	1	1.4%	17	90
2	contracts/VotingPowerAggregator.sol	148	25	16.9%	34	207
3	contracts/ActionTierConfig.sol	56	1	1.8%	9	66
4	contracts/LiquidityMining.sol	111	11	9.9%	20	142
5	contracts/RecruitNFT.sol	84	10	11.9%	19	113
6	contracts/GovernanceManager.sol	326	16	4.9%	53	395
7	contracts/EmergencyRecovery.sol	164	2	1.2%	30	196
8	contracts/access/GovernanceOnly.sol	13	2	15.4%	4	19
9	contracts/access/ImmutableOwner.sol	12	2	16.7%	4	18
10	contracts/vaults/RecruitNFTVault.sol	33	2	6.1%	7	42
11	contracts/vaults/FriendlyDAOVault.sol	62	8	12.9%	18	88
12	contracts/vaults/NFTVault.sol	61	1	1.6%	12	74
13	contracts/vaults/FoundingFrogVault.sol	64	7	10.9%	12	83
14	contracts/vaults/LPVault.sol	160	9	5.6%	31	200
15	contracts/vaults/AggregateLPVault.sol	63	1	1.6%	17	81
16	contracts/emergency_recovery_multisig/NoSafeManagementByMultisig.sol	61	12	19.7%	8	81
17	contracts/emergency_recovery_multisig/SafeManagementModule.sol	150	7	4.7%	16	173
18	contracts/tier_strategies/SimpleThresholdStrategy.sol	40	2	5.0%	6	48
19	contracts/tier_strategies/SetAddressStrategy.sol	39	1	2.6%	9	49
20	contracts/tier_strategies/BaseThresholdStrategy.sol	26	1	3.8%	5	32
21	contracts/tier_strategies/SetVaultFeesStrategy.sol	34	3	8.8%	5	42
22	contracts/tier_strategies/SetSystemParamsStrategy.sol	46	1	2.2%	6	53
23	contracts/tier_strategies/StaticTierStrategy.sol	21	1	4.8%	5	27
24	libraries/Errors.sol	11	1	9.1%	2	14
25	libraries/ScaledMath.sol	10	1	10.0%	3	14
26	libraries/DataTypes.sol	88	1	1.1%	15	104
27	libraries/VotingPowerHistory.sol	191	1	0.5%	24	216
28	libraries/Merkle.sol	19	1	5.3%	3	23
29	interfaces/IVotingPowersUpdater.sol	7	1	14.3%	1	9
30	interfaces/ITierer.sol	8	1	12.5%	2	11
31	interfaces/IVotingPowerAggregator.sol	20	1	5.0%	6	27
32	interfaces/IVault.sol	5	1	20.0%	2	8
33	interfaces/ITierStrategy.sol	7	1	14.3%	2	10
34	interfaces/IDelegatingVault.sol	12	1	8.3%	4	17
35	interfaces/ILiquidityMining.sol	24	12	50.0%	12	48
36	interfaces/IWrappedERC20WithEMA.sol	4	1	25.0%	1	6
37	interfaces/ILockingVault.sol	18	1	5.6%	5	24
	Total	**2270**	**151**	**6.7%**	**429**	**2850**

	Contract	LoC	Comments	Ratio	Blank	Total
1	contracts/GydRecovery.sol	275	48	17.5%	62	385
2	contracts/Motherboard.sol	412	26	6.3%	76	514
3	contracts/ReserveStewardshipIncentives.sol	162	21	13.0%	39	222
	Total	**849**	**95**	**11.2%**	**177**	**1121**

3 Summary of Issues

	Finding	Severity	Update
1	Liquidity Pool providers can create an infinite amount of delegated vote power	Critical	Fixed
2	Voting power can be reused for proposal voting	Critical	Fixed
3	Voting power of users is not being correctly stored when votes are cast	Critical	Fixed
4	Any user can mint a RecruitNFT after the first valid mint	High	Fixed
5	Function `claimNFT(...)` can be frontrun	High	Fixed
6	Function `setSchedule(...)` always reverts after it is called once	High	Fixed
7	Underflow in WrappedERC20WithEMA for withdraw transactions	High	Fixed
8	`WrappedERC20WithEMA` can lock underlying tokens if elapsed time windows exceed 41	High	Fixed
9	Function `_updateEMA()` may fail when `windowWidth` is set to a low value	Medium	Fixed
10	Users cannot withdraw all their assets from `GydRecovery`	Medium	Fixed
11	`int256` unsafely casted to `uint256` during EMA calculations	Medium	Fixed
12	Max supply of recruitNFT cannot be reached	Low	Fixed
13	Burn actions affect users with withdrawal waiting time completed	Info	Acknowledged
14	The function `setSchedule(...)` accepts `scheduleEndsAt` equal to `scheduleStartsAt`	Info	Fixed
15	Users are forced to delegate voting power in LPVault	Info	Fixed
16	Using `delete` on a Solidity array won't decrease its length	Info	Fixed
17	Wrong value used in `_getSelector(...)`	Info	Fixed
18	`DataTypes.Status` defaults to `Active`	Info	Fixed
19	`claimNFT(...)` does not properly check `multiplier`	Info	Fixed
20	State variable `owner` is shadowed in the function `claimNFT(...)`	Best Practices	Fixed
21	Transaction status is not checked	Best Practices	Fixed

4 System Overview

The audit is based on two repositories: a) **Gyroscope Governance** is designed to bring different stakeholder groups into governance. The scope of this part consists of 25 contracts. b) **Gyroscope Protocol** contains the core components, such as the primary AMM pricing mechanism and several other core contracts of the protocol. Our scope was limited to 4 (four) contracts: `LiquidityMining`, `Motherboard`, `ReserveStewardshipIncentives`, and `GydRecovery` The directed graph in Fig.2 shows the interaction of the main contracts for the **Governance** repository.

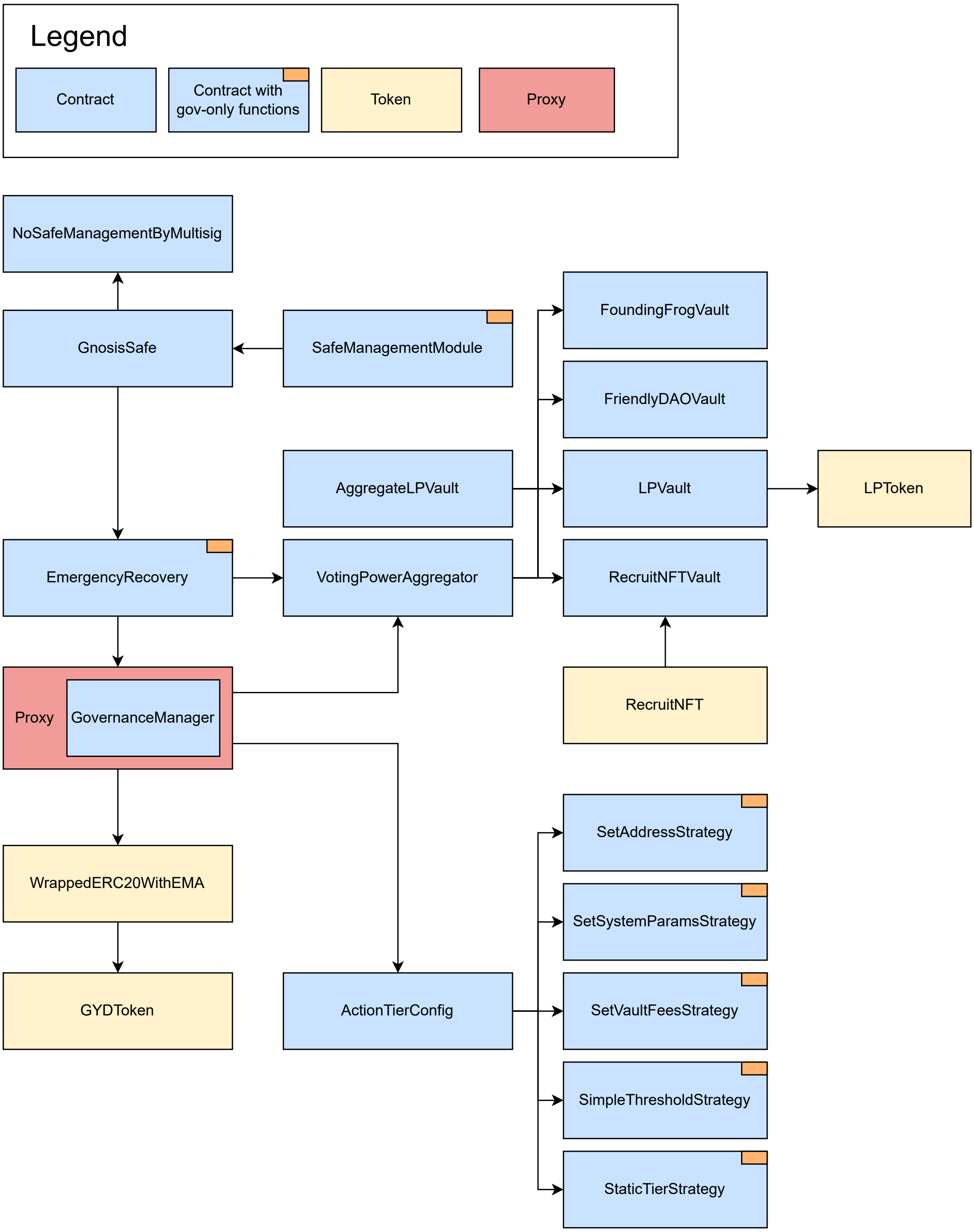

Fig. 2: Directed graph for Governance describing contracts' interactions.

Vaults - The Gyroscope Governance implements 5 (five) vaults: `FoundingFrogVault`, `FriendlyDAOVault`, `LPVault`, `RecruitNFTVault`, and `AggregateLPVault`. In most vaults, the voting power can be delegated. This action decreases the voting power of the account delegating and increases the voting power of the delegated account.

Voting Power Computation - The `VotingPowerAggregator` contract computes the voting power of a user. This contract loops over all vaults and sums up the voting power of the user in each vault weighted with the vault's weight (`getVotingPower`). The owner adds these vaults through the function `setSchedule`.

Action Tiering - The Gyroscope Governance implements 5 (five) tier strategies: `SimpleThresholdStrategy`, `SetVaultFeesStrategy`, `StaticTierStrategy`, `SetSystemParamsStrategy`, and `SetAddressStrategy`. The `StaticTierStrategy` contract always returns the same tier regardless of the passed arguments. `SimpleThresholdStrategy` returns a tier based on whether one of the parameters is above a given threshold. `SetVaultFeesStrategy` contract is similar to `SimpleThresholdStrategy`. The difference is that `SetVaultFeesStrategy` compares two arguments to the threshold. `StaticTierStrategy` contract implements the simplest strategy. It always returns the same tier regardless of the passed arguments. `SimpleThresholdStrategy`, `SetVaultFeesStrategy` are strategies that rely on a given threshold to return a tier. `SimpleThresholdStrategy` check if one of the parameters is above the threshold. On the other hand, `SetVaultFeesStrategy` compares two arguments to the threshold. `SetSystemParamsStrategy` is another strategy that follows similar logic to the `SimpleThresholdStrategy`. They differ from one another in that `SetSystemParamsStrategy` compares several fields of a struct to multiple thresholds.

Emergency Recovery - The Gyroscope Governance includes an emergency recovery mechanism as a fallback. The `EmergencyRecovery` contract implements the following features: i) Multisig signers are assigned by governance and can be changed at any time; ii) The multisig can start an upgrade to the governance contract subject to a timelock. During the timelock, governance can vote to override the upgrade. Emergency recovery has a built-in sunset, which governance can optionally extend.

GYD users' power over upgradeability - GYD stablecoin holders have power over governance regarding how upgradeable the protocol should be. For that, `WrappedERC20WithEMA` contract allows users to choose between holding GYD or the wrapped wGYD (an alternative wrapped form of the GYD that can affect governance settings) and can freely convert between them. If a user chooses to hold wGYD, it means a vote for a more limited upgradeability of the protocol. When a user converts between GYD and wGYD, it calculates the new Exponential Moving Average (EMA) in the wGYD contract.

Reserve stewardship incentives - `ReserveStewardshipIncentives` contract implements a mechanism to be a steward of the GYD reserve structure, adapting it as the DeFi space changes. This contract is in the `Protocol` repository. When governance starts an initiative by calling the function `startInitiative`. It is required that the system has excessive health properties, e.g., that the reserve ratio is above an excess threshold. To specify an initiative requires defining a `rewardPercentage`. Once an initiative is active, several health properties of the system are tracked over a long period, as performed via `_checkpoint`. At the end of the period, governance can call the function `completeInitiative` to verify whether health properties were achieved over the term of the initiative. If so, then it calculates an incentive reward based on `rewardPercentage` and the average GYD supply over the term.

GYD recovery module - The governance can incentivize a backstop to the protocol. This mechanism is implemented in the `GydRecovery` contract in the `Protocol` repository. Basically, any user can deposit GYD to the recovery module to backstop the system. A user can initiate a withdrawal of their staked GYD by calling the function `initiateWithdrawal` subject to an unstacking period (`rewardsEmissionEndTime`).

5 Risk Rating Methodology

The risk rating methodology used by Nethermind follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to Motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploited the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to pass to the client formally;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

6.1 [Critical] Liquidity Pool providers can create an infinite amount of delegated vote power

File(s): LPVault.sol

Description: Users can deposit LP tokens into the LPVault and receive base voting power. Later, they can withdraw their LP tokens through a two-step process that removes their base voting power. Voting power can also be delegated using the function delegateVote(...), which decreases the depositor's voting power and increases the specified address's voting power. This change is recorded in the field Record.netDelegatedVotes and is considered in VotingPowerHistory.total(...).

When a user withdraws LP tokens from LPVault using the function initiateWithdrawal(...), only base voting power is checked, as presented below:

```
function initiateWithdrawal(uint256 _amount, address _delegate) external returns (uint256) {
    require(_amount >= 0, "invalid withdrawal amount");

    VotingPowerHistory.Record memory currentVotingPower = history
        .currentRecord(msg.sender);
    require(
        currentVotingPower.baseVotingPower >= _amount,
        "not enough to undelegate"
    );
    ...
}
```

Then, the base voting power is decreased by the corresponding amount. Undelegating voting power is optional and is omitted if _delegate is address(0) or msg.sender. This creates a situation where a user can deposit LP tokens, delegate all of their voting power to another address and then withdraw all of their LP tokens, leaving the delegated voting power with the other address. For example, consider the following scenario.

- Bob deposits 100 LP tokens into the LPVault contract and receives 100 base voting power;
- Bob delegates 100 voting power to Alice;
- Bob withdraws all 100 LP tokens;
- Alice retains 100 delegated voting power;

A user who delegates votes will end up with negative netDelegatedVotes.

Recommendation(s): To prevent withdrawing LP tokens while voting power is delegated,a check could be added in the initiateWithdraw(...) function to ensure that the withdrawn amount is lower than or equal to the sum of baseVotingPower and netDelegatedVotes.

Status: Fixed.

Update from client: This issue was fixed by using a snapshot of the voting power. The two main commits are: 34a6c1422c7d1f7f5630c955da058bfccb4303 and 84cfe67c9c23ade939a8880eea0fd2a42155f266. For full context, see PR 45.

Update from Nethermind: Not fixed yet. The LP can still add liquidity, delegate, and withdraw without removing the delegated voting power.

Update from the client: Fixed in 96f5a922929ff562e3181c074d74e2b9d3bb6b4e.

Update from Nethermind: The provided change fixes the issue. However, the introduced check

```
currentVotingPower.baseVotingPower -
    history.delegatedVotingPower(msg.sender) >=
_amount
```

is correct only if the multiplier is equal to 1e18 in the LPVault. For multiplier != 1e18 the baseVotingPower must be scaled by multiplier. If the multiplier changes in the future, the current change will be invalid. Therefore we recommend either introducing the scaling of baseVotingPower by multiplier or documenting clearly in the code comments why this multiplication is not currently needed.

Update from the client: Added a comment in 054defa0020dac40bc2872e517bdd9fe283e4c2a. The LP vault purposefully does not provide any way to update multiplier, nor do we have any plans of supporting to change the multiplier in this vault. Therefore, we can safely assume that it is always 1e18.

6.2 [Critical] Voting power can be reused for proposal voting

File(s): `GovernanceManager.sol`

Description: When a user votes for a proposal, their current voting power is recorded in the `_totals` object. However, the user can still use their voting power without any restrictions, allowing malicious users to reuse their voting power by delegating it to other addresses. Below we present an exploit scenario.

Exploit Scenario:

- Address `A` delegates all of its voting power to address `B`. Address `B` then casts a vote for Proposal 1;
- Address `A` revokes its delegation from address `B` and delegates all of its voting power to address `C`. Address `C` then casts a vote for Proposal 1;
- Address `A` revokes its delegation from address `C` and casts its vote for Proposal 1;

In this scenario, the same voting power owned by address `A` was used three times to cast votes for the same proposal. This process can be repeated indefinitely, allowing an infinite number of votes to be generated. A similar exploit is possible by depositing tokens to LPVault, voting, and withdrawing tokens.

Recommendation(s): Consider measuring voting power for a specific proposal at the block the proposal was created. This would freeze voting power for a given proposal, so votes can't be delegated or withdrawn.

Status: Fixed.

Update from the client: This issue was fixed using a snapshot of the voting power. The two main commits are: 34a6c1422c7d1f7f5630c955da058bfccb430 and 84cfe67c9c23ade939a8880eea0fd2a42155f266. For full context, see PR 45.

Update from Nethermind: In the `vote()` function, the current implementation fetches the voting power from `proposal.createdAt`. Nevertheless, the contract currently permits voting immediately after the proposal's creation, creating a potential scenario where users could manipulate the system by adding voting power, voting, and then delegating/removing it all within a single block/TX (like flashloan). To address this concern, it is suggested to use `proposal.createdAt - 1` instead of `proposal.createdAt` to prevent such exploits.

Another potential solution is to introduce a small waiting period between the creation of a proposal and the begin of the voting period.

Update from the client: Fixed in 533a74b34fbd83a28cdc112a19a714537e11e5d1.

6.3 [Critical] Voting power of users is not being correctly stored when votes are cast

File(s): `GovernanceManager.sol`

Description: The function `_copyToStorage(...)` copies all elements of the memory array `vaults` to the storage array `existingVoteVaults`. If the length of `vaults` is smaller than `existingVoteVaults`, it has to remove redundant elements at the end of the storage array using a `for` loop. However, the start index of the loop is incorrect; it starts from `vaults.length - 1` instead of `vaults.length`.

```
function _copyToStorage(
    DataTypes.VaultVotingPower[] storage existingVoteVaults,
    DataTypes.VaultVotingPower[] memory vaults
) internal {
    if (existingVoteVaults.length > vaults.length) {
        for (uint256 i = 0; i < vaults.length; i++) {
            existingVoteVaults[i] = DataTypes.VaultVotingPower({
                vaultAddress: vaults[i].vaultAddress,
                votingPower: vaults[i].votingPower
            });
        }

        //////////////////////////////////////////////////
        // @audit Start index should be vaults.length
        //////////////////////////////////////////////////
        for (
            uint256 i = vaults.length - 1;
            i < existingVoteVaults.length;
            i++
        ) {
            delete existingVoteVaults[i];
        }
    }
    ...
}
```

As a result, the last element of the storage array is deleted. Even deleting the last element, the function `vote(...)` casts a vote for the user correctly because it loops through the memory array. An attacker can exploit this by repeatedly calling the function `vote(...)`. Because the storage array is missing the last element, it will not be reduced from the total vote, and the attacker can repeat this to get an infinite vote for a proposal.

```
function vote(uint24 proposalId, DataTypes.Ballot ballot) external {
    ...

    DataTypes.VaultVotingPower[] memory uvp = votingPowerAggregator
        .getVotingPower(msg.sender);

    DataTypes.Vote storage existingVote = _votes[msg.sender][proposalId];
    // First, zero out the effect of any vote already cast by the voter.
    for (uint256 i = 0; i < existingVote.vaults.length; i++) {
        DataTypes.VaultVotingPower memory vvp = existingVote.vaults[i];
        (, uint256 val) = _totals[proposalId][existingVote.ballot].tryGet(
            vvp.vaultAddress
        );
        _totals[proposalId][existingVote.ballot].set(
            vvp.vaultAddress,
            val - vvp.votingPower
        );
    }

    // Then update the record of this user's vote to the latest ballot and voting power
    existingVote.ballot = ballot;
    // Copy over the voting power
    _copyToStorage(existingVote.vaults, uvp);

    ///////////////////////////////////////////////////////////
    // @audit Even though _copyToStorage(...) is wrong,
    // it still casts the vote correctly by using `uvp` array.
    ///////////////////////////////////////////////////////////
    // And, finally, update running total
    for (uint256 i = 0; i < uvp.length; i++) {
        DataTypes.VaultVotingPower memory vvp = uvp[i];
        (, uint256 val) = _totals[proposalId][existingVote.ballot].tryGet(
            vvp.vaultAddress
        );
        _totals[proposalId][existingVote.ballot].set(
            vvp.vaultAddress,
            val + vvp.votingPower
        );
    }
    ...
}
```

Recommendation(s): Change the start index in function `_copyToStorage(...)` to `vaults.length`.

Status: Fixed.

Update from the client: Fixed in b0a54fa2e6bba1aa11d3b764098695dffe96f94c.

6.4 [High] Any user can mint a RecruitNFT after the first valid mint

File(s): RecruitNFT.sol

Description: To mint a RecruitNFT, the mint(...) function is used. Before minting the NFT, this function performs the following checks: a) the receiver address cannot have been the mint receiver before; b) if the caller differs from the owner, the caller must provide valid Merkle proof and a valid signature. The _requireValidProof(...) function is used to verify whether the provided proof and signature are valid. This function is shown in the following code snippet.

```
function _requireValidProof(...) public {
    if (msg.sender == owner) {
        return;
    }

    bytes32 hash = _hashTypedDataV4(
        keccak256(abi.encode(_TYPE_HASH, to, _encodeProof(proof)))
    );
    address claimant = ECDSA.recover(hash, signature);

    require(claimant == to, "invalid signature");

    bytes32 node = keccak256(abi.encodePacked(owner));
    require(merkleRoot.isProofValid(node, proof), "invalid proof");
}
```

This function should check whether the receiver is a part of the set of valid addresses through the provided proof and whether the receiver is willing to receive the NFT through the provided signature. However, the function always performs an inclusion-proof for the same address, the owner. This means that after a valid proof is submitted, any user can reuse it to prove that the owner is a valid address and mint a token for them.

Recommendation(s): Check whether the to address is a part of the Merkle tree instead of the owner address

```
- bytes32 node = keccak256(abi.encodePacked(owner));
+ bytes32 node = keccak256(abi.encodePacked(to));
```

Status: Fixed.

Update from the client: Fixed in 903a32055fdc01a751e0e4274a4f6ef6b47941cd

6.5 [High] Function `claimNFT(...)` can be frontrun

File(s): `FoundingFrogVault.sol`

Description: The `FoundingFrogVault` contract allows users to gain voting power by calling the function `claimNFT()`. This function first checks if the provided proof is valid given the Merkle root and then increases the base voting power of the caller by one. However, the function does not check if the caller `msg.sender` is equal to the `owner` (who represents the owner of the NFT). As a result, a malicious user may detect that another user is claiming the NFT in `FoundingFrogVault`, and front-run their call with the same data, resulting in an update to the malicious user's base voting power and preventing a valid user from claiming the NFT.

```
function claimNFT(
    address owner,
    uint128 multiplier,
    bytes32[] calldata proof,
    bytes calldata signature
) external {
    require(
        multiplier >= 1e18,
        "multiplier must be greater or equal than 1e18"
    );

    bytes32 hash = _hashTypedDataV4(
        keccak256(
            abi.encode(_TYPE_HASH, owner, multiplier, _encodeProof(proof))
        )
    );
    address claimant = ECDSA.recover(hash, signature);
    require(claimant == owner, "invalid signature");

    require(_claimed[owner] == address(0), "NFT already claimed");

    bytes32 node = keccak256(abi.encodePacked(owner, multiplier));
    require(merkleRoot.isProofValid(node, proof), "invalid proof");

    _claimed[owner] = msg.sender;
    //////////////////////////////////////////////////////////////////////
    // @audit should add voting power to the owner instead of msg.sender
    //////////////////////////////////////////////////////////////////////
    VotingPowerHistory.Record memory current = history.currentRecord(
        msg.sender
    );
    history.updateVotingPower(
        msg.sender,
        current.baseVotingPower + ScaledMath.ONE,
        multiplier,
        current.netDelegatedVotes
    );
}
```

Recommendation(s): Consider checking if `owner` is equal to the `msg.sender` or add an address to the signed message committing to the user that will call `claimNFT(...)`.

Status: Fixed.

Update from the client: Fixed in bef8e4dad77ce21684083a12d8cc303e10c7bca7.

6.6 [High] Function `setSchedule(...)` always reverts after it is called once

File(s): `VotingPowerAggregator.sol`

Description: The function `setSchedule(...)` calls the `internal` function `_removeAllVaults()` to remove all existing vaults before adding the new ones. The `_removeAllVaults()` is described below:

```
function _removeAllVaults() internal {
  uint256 length = _vaultAddresses.length();
  for (uint256 i; i < length; i++) {
      address vaultAddress = _vaultAddresses.at(i);
      _vaultAddresses.remove(vaultAddress);
      delete _vaults[vaultAddress];
  }
}
```

The state variable `_vaultAddresses` implements the OpenZeppeling library `EnumerableSet`. After each call to function `remove(...)`, the `_vaultAddresses`' length is updated. However, the loop in `_removeAllVaults()` increases the `i` value. The error scenario is detailed below. Consider the current state for `_vaultAddresses`:

0	1	2	3
0xd2a..	0x9D7..	0x358A..	0xDA0b..

The next table describes the value for `i` in each iteration and the `_vaultAddresses`' length before and after removing the vault address. When `_removeAllVaults()` is called, we have the following:

`i`	`length` before	`length` after
0	4	3
1	3	2
2	2	`error`

As we can see, the current value for `i` is greater than the current length for `_vaultAddresses`. In addition, not all vaults are removed, as presented below:

- When `i=0`, the last address is moved to the first position and the length is decreased.

0	1	2
0xDA0b..	0x9D7..	0x358A..

- When `i=1`:

0	1
0xDA0b..	0x358A..

- When `i=2`, the transaction is reverted as the index is outside the reachable range at `_vaultAddresses`. Observe that even solving the issue with the range, `_vaultAddresses` can remain with addresses;

Recommendation(s): Make sure the loop variable `i` is accessing a valid range in `_vaultAddresses` and all addresses are removed. One solution would be to remove the last addresses. The loop decrements the value of `i`, where this value starts with `i = length - 1`, and the loop runs while `i > 0`, as described below. As we can observe, to remove all vaults, the function `remove(...)` needs to be called once after the loop.

```
for (i=length-1; i > 0; --i){...}
```

Status: Fixed.

Update from the client: Fixed in 7921df3bd8f0a8ecfe964c63bb19e8fda5cd1db0.

6.7 [High] Underflow in WrappedERC20WithEMA for withdraw transactions

File(s): `WrappedERC20WithEMA`

Description: The function `_updateEMA()` is called by functions `deposit(...)`, `withdraw(...)`, and updateEMA() to calculate the Exponential Moving Average (EMA) indicator. As described below, `_updateEMA()` calculates values for `expMovingAverage.value` and `previousWrappedPctOfSupply.value`.

```
function _updateEMA() internal {
    if (previousWrappedPctOfSupply.blockNb < block.number) {
        ...
        expMovingAverage.value += uint256(
            ((int256(ScaledMath.ONE) - LogExpMath.exp(exponent)) *
                int256(
                    previousWrappedPctOfSupply.value -
                        expMovingAverage.value
                )) / int256(ScaledMath.ONE)
        );
        expMovingAverage.blockNb = previousWrappedPctOfSupply.blockNb;
    }
    previousWrappedPctOfSupply.value = wrappedPctOfSupply();
    previousWrappedPctOfSupply.blockNb = block.number;
}
```

By definition, EMA indicator tracks the price of an asset over time, i.e., its value can `decrease` and `increase`. The way EMA is implemented in `_updateEMA()`, the variable `expMovingAverage.value` should always increase its value, independently of whether the transaction is _deposit_ or _withdraw_. However, its value should decline when withdrawals are executed, indicating a downtrend. On the other hand, when withdraw transactions are executed, `previousWrappedPctOfSupply.value` declines correctly. As withdraw transactions decrease `previousWrappedPctOfSupply.value`, its value tends to be smaller than `expMovingAverage.value` (described below).

```
previousWrappedPctOfSupply.value -
                        expMovingAverage.value
```

Consider the formula to calculate EMA:

`EMA = (weightingFactor * (currentPrice - previousEMA)) + previousEMA` Where:

`weightingFactor = ((int256(ScaledMath.ONE) - LogExpMath.exp(exponent))` `currentPrice = previousWrappedPctOfSupply.value` `previousEMA = expMovingAverage.value`

When `previousWrappedPctOfSupply.value < expMovingAverage.value`, the result of `(currentPrice - previousEMA)` is smaller than `zero`. The attempt to convert this negative value (`int256`) to `uint256` results in an underflow.

Exploit Scenario:

- Consider the following initial state: `underlying token = 10000, windowWidth = 2e18, totalSupply = 0;`
- Now, suppose the sequence of operations described in the table below:

User	Operation	expMovingAverage value	previousWrappedPctOfSupply value	totalSupply
Alice	deposit(5000)	0	500000000000000000	5000
Bob	deposit(1000)	47581290982020213	600000000000000000	6000
Alice	withdraw(5000)	100150881657413113	100000000000000000	1000

As we can observe, when Alice and Bob deposit, both `expMovingAverage.value` and `previousWrappedPctOfSupply.value` increase. However, when Alice withdraws her balance, the `previousWrappedPctOfSupply.value` results in a value smaller than `expMovingAverage.value`. Then, when Bob tries to withdraw, the transaction reverts. This happens because the following operation results in underflow:

```
int256(
    previousWrappedPctOfSupply.value -
        expMovingAverage.value
))
```

Recommendation(s): Ensure that the subtraction operation is applied between two `int256` and the add operation is also applied between two `int256` values **before converting** to `uint256`, for example:

```
int256 partialEMA =  (int256(ScaledMath.ONE) - LogExpMath.exp(exponent)) *
          ( int256(previousWrappedPctOfSupply.value) -
              int256(expMovingAverage.value)
          )) / int256(ScaledMath.ONE);

expMovingAverage.value = uint256(partialEMA + int256(expMovingAverage.value))
```

Status: Fixed.

Update from the client: Fixed in dbb48bc87db0a0095cb961292c518c3be7e5f739.

Response from Nethermind: Why the following check exists in _updateEMA(...):

```
int256 discount = LogExpMath.exp(exponent);
multiplier -= discount < 0 ? 0 : uint256(discount);
```

The output for `exp(...)` function can't ever be negative, so why is it checked for values less than `0`?

Update from the client: Fixed in ce1657b1e45d72fde15fbb567389a651cc594f71

6.8 [High] `WrappedERC20WithEMA` can lock underlying tokens if elapsed time windows exceed 41

File(s): `WrappedERC20WithEMA.sol`

Description: The `WrappedERC20WithEMA` contract updates its exponential moving average through the function `_updateEMA(...)`. A part of the calculations involves the function `exp(...)` from the library `LogExpMath`. A code snippet from the calculations is shown below.

```
expMovingAverage.value += uint256(
    ((int256(ScaledMath.ONE) - LogExpMath.exp(exponent)) *
        int256(
            previousWrappedPctOfSupply.value -
                expMovingAverage.value
        )) / int256(ScaledMath.ONE)
);
```

The `LogExpMath.exp(...)` call is done with the argument `exponent`, representing the negated number of EMA time windows between `expMovingAverage` and `previousWrappedPctOfSupply` scaled to `1e18`. Inside the `exp(...)` function, we see the following requirement:

```
require(
    x >= MIN_NATURAL_EXPONENT && x <= MAX_NATURAL_EXPONENT,
    BalancerErrors.INVALID_EXPONENT
);

// Where...

int256 constant MAX_NATURAL_EXPONENT = 130e18;
int256 constant MIN_NATURAL_EXPONENT = -41e18;
```

This requirement can be broken when the number of elapsed time windows exceeds 41. If this were to occur, the `WrappedERC20WithEMA` contract would revert on any call which involves an EMA update. This would prevent deposits, withdrawals, and direct EMA updates. The test files indicate that the time window should be two blocks, so if the contract is not interacted with in 82 blocks (16 minutes), then it is effectively unusable.

Recommendation(s): Refactor the `_updateEMA(...)` function to avoid calling `exp(...)` with an invalid exponent. One possible solution is to use constant weight to compute the new average when a specific amount of time has passed.

Status: Fixed.

Update from the client: Fixed in dbb48bc87db0a0095cb961292c518c3be7e5f739.

6.9 [Medium] Function `_updateEMA()` may fail when `windowWidth` is set to a low value

File(s): `WrappedERC20WithEMA.sol`

Description: The state variable `windowWidth` is set during deployment and cannot be changed later. The function `_updateEMA()` uses it to calculate the `exponent`. When computing `deltaBlockNb`, the values of the block numbers are brought to 18 decimals, as described below.

```
function _updateEMA() internal {
...
    uint256 deltaBlockNb = (previousWrappedPctOfSupply.blockNb -
            expMovingAverage.blockNb) * ScaledMath.ONE;
    int256 exponent = -int256(deltaBlockNb.divDown(windowWidth));
    expMovingAverage.value += uint256(
        ((int256(ScaledMath.ONE) - LogExpMath.exp(exponent)) *
            int256(
                previousWrappedPctOfSupply.value -
                    expMovingAverage.value
            )) / int256(ScaledMath.ONE)
    );
...
}
```

After some calls to `_updateEMA()`, the function may revert with the message `"INVALID_EXPONENT"`. For example, suppose `windowWidth=10` and the initial block number equals `78`. In the first call to `_updateEMA()`, the `LogExpMath.exp` is successfully calculated because `exponent=0`. However, in the next call to `_updateEMA()`, `deltaBlockNb` is greater than `zero`, resulting in `exponent=-100000000000000000000000000000000000`, which is smaller than `MIN_NATURAL_EXPONENT = -41e18`. The function will then revert since the `exponent` value is invalid.

`expMovingAverage.` blockNb	`previousWrappedPctOfSupply.` blockNb	`exponent`	`LogExpMath.` exp
78	79	0	1000000000000000000

Recommendation(s): Consider checking that the variable `windowWidth` is set to a safe value, potentially a value bigger than `1e18`.

Status: Fixed.

Update from the client: Fixed in 290f0944b684aee89fda1d49626aeb202e64b585.

Response from Nethermind: The commit above checked `windowWidth >= 1e18`. However, the check was changed to `windowWidth >= 0.01e18` in the final commit.

Update from the client: Everything works fine with this being less than one. If we would like the moving average to be extremely fast, we could, for example, set this to 0.5.

6.10 [Medium] Users cannot withdraw all their assets from `GydRecovery`

File(s): `GydRecovery.sol`

Description: As noted in the codebase, the `initiateWithdrawalAdjusted(...)` function includes a redundant check that provides a better error message. However, this check is incorrect as `adjustedAmount` is the amount divided by `adjustmentFactor`, whereas `balanceOf(msg.sender)` is not an adjusted amount.

Furthermore, we know that $adjustmentFactor \leq 1$, which means $adjustedBalance \geq balanceOf(msg.sender)$. As a result, users cannot withdraw the full `adjustedBalance` because the check limits the maximum amount to only `balanceOf(msg.sender)`.

Consider the following scenario.

- The adjustment factor is decreased from `1e18` to `5e17`;
- Bob deposits `10 gyd`, his `realBalance` is `10`, and his `adjustedBalance` is `20`;
- Bob wants to withdraw his `10 gyd`. He will call the function `initiateWithdrawal(...)` with `10` as the amount or the function `initiateWithdrawalAdjusted(...)` with `20` as the amount. In both cases, the requirement will fail, and Bob won't be able to withdraw all his balance because $20 \geq 10$;

```
function initiateWithdrawalAdjusted(uint256 adjustedAmount)
    public
    returns (uint256 withdrawalId)
{
    // redundant with _unstake() but we want a better error message.
    /////////////////////////////////////////////////////////////////////////////////////////
    // @audit wrong check, since maximum amount in _unstake is just balanceAdjustedOf
    /////////////////////////////////////////////////////////////////////////////////////////
    require(adjustedAmount <= balanceOf(msg.sender), "not enough to withdraw");

    _unstake(msg.sender, adjustedAmount); // This also handles full burns, which is important below.
    ...
}
```

Note that users can still withdraw most of their balances, creating multiple withdrawals.

Recommendation(s): Consider correcting the check in the function `initiateWithdrawalAdjusted(...)` as suggested.

Status: Fixed.

Update from the client: Fixed in 30cda8992712e484c992a99a84f93422f7813398.

6.11 [Medium] `int256` unsafely casted to `uint256` during EMA calculations

File(s): `WrappedERC20WithEMA.sol`

Description: After calculating the expected EMA change in `_updateEMA(...)`, the change is cast from `int256` to `uint256` and then added to `expMovingAverage.value`. However, if the `int256` value is negative when cast to `uint256`, it will turn into a very large positive number that is added to the EMA, rather than reducing the EMA. A code snippet with these calculations is shown below:

```
expMovingAverage.value += uint256(
    ((int256(ScaledMath.ONE) - LogExpMath.exp(exponent)) *
        int256(
            previousWrappedPctOfSupply.value -
                expMovingAverage.value
        )) / int256(ScaledMath.ONE)
);
```

It is also worth noting that the `expMovingAverage.value` is always incremented, meaning it is impossible for the exponential moving average ever to decrease.

Recommendation(s): Consider adding a check to see if the `int256` value is positive or negative first, and then decrease or increase `expMovingAverage.value` depending on if the EMA change is positive or negative.

Status: Fixed.

Update from the client: Fixed in dbb48bc87db0a0095cb961292c518c3be7e5f739.

6.12 [Low] Max supply of recruitNFT cannot be reached

File(s): RecruitNFT.sol

Description: The `mint(...)` function ensures that no more than `maxSupply` tokens are minted. This check is shown in the next snippet of code.

```
function mint(...) public {
    _requireValidProof(to, proof, signature);

    require(!_claimed[to], "user has already claimed NFT");

    _mint(to, tokenId);
    tokenId++;

    _claimed[to] = true;

    require(
        tokenId < maxSupply,
        "mint error: supply cap would be exceeded"
    );
    vault.updateBaseVotingPower(to, 1e18);
}
```

However, because the check that `tokenId` is lower than `maxSupply` is done for the next `tokenId` that will be used, the check will fail one mint before the maximum valid `tokenId` (`maxSupply - 1`) is reached.

Recommendation(s): Modify the function to check if the `tokenId` used for the minted token is a valid one.

Status: Fixed.

Update from the client: Fixed in df51e0b25177f40b4d04b9d7d84574c65fb64c00.

6.13 [Info] Burn actions affect users with withdrawal waiting time completed

File(s): GydRecovery.sol

Description: In `GydRecovery`, users cannot withdraw their stake immediately. Instead, they have to call the `initiateWithdrawal(...)` function and wait for a certain amount before withdrawing to their accounts. During this waiting period, the recovery module can burn `GYD` from stakers. However, if users do not call the `withdraw(...)` function immediately after the waiting period, they could still be affected by burn events and potentially lose their funds.

```
function withdraw(uint256 withdrawalId) external returns (uint256 amount) {
    PendingWithdrawal memory pending = pendingWithdrawals[withdrawalId];
    require(pending.to == msg.sender, "matching withdrawal does not exist");
    require(pending.withdrawableAt <= block.timestamp, "not yet withdrawable");

    /////////////////////////////////////////////////////////////////
    // @audit if users withdraw late, they could lose all
    /////////////////////////////////////////////////////////////////
    if (pending.createdFullBurnId < nextFullBurnId) {
        delete pendingWithdrawals[withdrawalId];
        userPendingWithdrawalIds[pending.to].remove(withdrawalId);
        emit WithdrawalCompleted(withdrawalId, pending.to, 0, 0);
        return 0;
    }

    positions[pending.to].adjustedAmount -= pending.adjustedAmount;

    amount = pending.adjustedAmount.mulDown(adjustmentFactor);
    gydToken.safeTransfer(pending.to, amount);

    delete pendingWithdrawals[withdrawalId];
    userPendingWithdrawalIds[pending.to].remove(withdrawalId);

    emit WithdrawalCompleted(withdrawalId, pending.to, pending.adjustedAmount, amount);
}

```

Recommendation(s): Consider reviewing the withdrawal mechanism. If a burn occurs after the withdrawal request is available, the available withdrawal requests should not be affected.

Status: Acknowledged.

Update from the client: We accept the risk here, and this will be documented appropriately.

6.14 [Info] The function `setSchedule(...)` accepts `scheduleEndsAt` equal to `scheduleStartsAt`

File(s): `VotingPowerAggregator.sol`

Description: The `setSchedule(...)` function sets the `scheduleStartsAt` and `scheduleEndsAt` state variables. However, the current implementation allows for the `scheduleEndsAt` value to be equal to `scheduleStartsAt`, which contradicts the error message stating that the schedule must end after it begins.

Recommendation(s): To resolve this issue, consider applying the change below.

```
- require(
-     _scheduleEndsAt >= _scheduleStartsAt,
-     "schedule must end after it begins"
- );
+ require(
+     _scheduleEndsAt > _scheduleStartsAt,
+     "schedule must end after it begins"
+ );
```

Status: Fixed.

Update from the client: Fixed in 232e5cedafa01872235382c8eb387256f876294e.

6.15 [Info] Users are forced to delegate voting power in LPVault

File(s): `LPVault.sol`

Description: Users can deposit LP tokens to `LPVault` to stake LP, and get voting power. However, the function `deposit(...)` does not allow for `_delegate` to be `address(0)`. In effect calling `deposit(...)` requires passing an address other than `address(0)`. It is still possible to not delegate voting power by providing `msg.sender` as the `_delegate`.

Recommendation(s): Consider removing the check for `_delegate` being `address(0)` to avoid passing an unnecessary argument in case the user doesn't want to delegate voting power.

Status: Fixed.

Update from the client: Fixed in c6abf7ddfe657105cec9404642fa03f979a7ca09.

6.16 [Info] Using `delete` on a Solidity array won't decrease its length

File(s): `GovernanceManager.sol`

Description: In function `_copyToStorage(...)`, the function `delete` is used to remove redundant elements. However, `delete` in Solidity only sets the value to `zero` and does not decrease the array's length. As a result, the storage array will retain redundant elements and cost more gas for users when they try to vote.

```
for (
    uint256 i = vaults.length - 1;
    i < existingVoteVaults.length;
    i++
) {
    ///////////////////////////////////////////////////////////
    // @audit delete not decrease length, use pop() instead
    //////////////////////////////////////////////////////////
    delete existingVoteVaults[i];
}
```

Recommendation(s): Use `pop()` instead of `delete()` and cache the array length before the loop.

Status: Fixed.

Update from the client: Fixed in c1cbddac8031dd22e5a6f5cd655cee002b559d8f.

6.17 [Info] Wrong value used in `_getSelector(...)`

File(s): `ActionTierConfig.sol`, `NoSafeManagementByMultisig.sol`

Description: For getting the function selector, the function `_getSelector(...)` is used, it returns a `bytes4` output. The code used for the same is shown below.

```
function _getSelector(
    bytes memory _calldata
) internal pure returns (bytes4 out) {
    assembly {
       ////////////////////////////////////////////////////////////////////////////
       // @audit For bytes4, they have an extra F here (9 F's instead of 8 F's)
       ////////////////////////////////////////////////////////////////////////////
        out := and(
            mload(add(_calldata, 32)), 0xFFFFFFFFF0000000000000000000000000000000000000000000000000000000)
    }
}
```

The output data uses an `and` operation with the value `0xFFFFFFFFF0000000000000000000000000000000000000000000000000000000`, which should only contain 8 F's, instead of 9. The same problem can be seen in `NoSafeManagementByMultisig` contract as well.

Recommendation(s): Change the value from:

```
- 0xFFFFFFFFF0000000000000000000000000000000000000000000000000000000
+0xFFFFFFFF00000000000000000000000000000000000000000000000000000000
```

Status: Fixed.

Update from the client: Fixed in 55c8ac615e266f94c33aeb3c47c04248916f0aff.

6.18 [Info] `DataTypes.Status` defaults to `Active`

File(s): `DataTypes.sol`

Description: The enum `Status` declared in the contract `DataTypes` contains four possible states: "Active", "Rejected", "Queued", and "Executed". The enum definition is shown below.

```
enum Status {
    Active,
    Rejected,
    Queued,
    Executed
}
```

The Solidity compiler will treat these possible values as integers, where the first enum value is `0`, the second is `1`, and so on. Since Ethereum state defaults to zero, all uninitialized proposals will be considered "Active" because it is the first enum value. This does not affect the protocol but may affect UI or protocol monitoring software.

Recommendation(s): Consider adding a new first entry to the `Status` enum named `Undefined`, similar to the approach taken for the `ProposalOutcome` and `Ballot` enums.

Status: Fixed.

Update from the client: Fixed in eb89183c9979ab5efe3949e0ee575d494720635c.

6.19 [Info] `claimNFT(...)` does not properly check `multiplier`

File(s): `FoundingFrogVault.sol`

Description: The function `claimNFT(...)` in contract `FoundingFrogVault` only checks that the `multiplier` is at least `1e18`. However, the function `updateMultiplier(...)` also retrains the multiplier to be not larger than `20e18`. It is possible for a user to call `claimNFT(...)` with a multiplier higher than `20e18`, and since the multiplier can only be increased, it couldn't be changed.

Recommendation(s): Consider checking if the `multiplier` used in `claimNFT(...)` is in the range `[1e18; 20e18]`.

Status: Fixed.

Update from the client: Fixed in 12fd96390171075c311de2544d1cb5ec12e8cf0f.

6.20 [Best Practice] State variable `owner` is shadowed in the function `claimNFT(...)`

File(s): `FoundingFrogVault.sol`

Description: The function `claimNFT(...)` receives four parameters: `owner`, `multiplier`, `proof`, and `signature`. This function can be found in the `FoundingFrogVault` contract, which indirectly inherits from the `ImmutableOwner` contract. The `ImmutableOwner` contract has only one state variable called `owner`, which is shadowed in this function by the argument `owner` received as an argument. Shadowed variables can be confusing and make the code error-prone.

Recommendation: Consider changing the name of the argument received by this function to avoid shadowing the `owner` state variable.

Status: Fixed.

Update from the client: Fixed in ba613d2dc810e4cf67fedf1e28ff12e1031d4693.

6.21 [Best Practice] Transaction status is not checked

File(s): `SafeManagementModule.sol`

Description: The `execTransactionFromModule(...)` function is inherited from the contract `ModuleManager`. It returns a boolean value, which contains the status of the function call.

```
function execTransactionFromModule(
    address to,
    uint256 value,
    bytes memory data,
    Enum.Operation operation
) public virtual returns (bool success) {}
```

In the contract `SafeManagementModule`, there are multiple instances where the function `execTransactionFromModule(...)` is used without the check for the status of that particular call.

```
function _swapOwner(address prevOwner, address oldOwner, address newOwner) internal {
    bytes memory data = abi.encodeCall( OwnerManager.swapOwner, (prevOwner, oldOwner, newOwner) );
    /////////////////////////////////////////////////////////////////////////
    // @audit The status of the transaction is not checked.
    /////////////////////////////////////////////////////////////////////////
    safe.execTransactionFromModule( address(safe), 0, data, Enum.Operation.Call );
}
```

As these calls are done through governance, we expect maximum scrutiny of the parameters in these calls. However, considering human error while creating proposals, the best way is to ensure the finality of these calls through code.

Recommendation(s): Consider checking the status of these function calls.

Status: Fixed.

Update from the client: Fixed in f7929a9f7c645619c376e5d8bc68cba83cd7e6b6.

7 Documentation Evaluation

Technical documentation is created to explain what the software product does. This way, developers and stakeholders can easily follow the purpose and the underlying functionality of each file/function/line. Documentation can come not only in the form of a README.md but also using code as documentation (to write clear code), diagrams, websites, research papers, videos, and external documentation. Besides being a good programming practice, proper technical documentation improves the efficiency of audits. Less time can be spent understanding the protocol, and more time can be put towards auditing, improving the audit's efficiency and overall output.

The `Gyroscope` team provided two documents to assist the audit process: (a) **Gyroscope governance and protocol** - this documentation specifies on high-level abstraction the proposal lifecycle (proposal creation, voting phase, and how proposals are concluded and executed). The documentation also describes in finer granularity features, such as: how voting power is computed, the particularities of each implemented vault, the tier strategies, the emergency recovery mechanism, Governance checks and balances (e.g., the power of GYD users to limit upgradeability, GYD recovery module, etc.). (b) **instructions for running the test suite**. These documents covered the most common terms used in the source code, explanation for the core business logic, and guides for running tests.

7.1 Documentation inconsistencies

During the auditing process, we noticed some parts in the code where the implementation differs from the specification. In the code, all vaults inherit the abstract contract `ImmutableOwner`, directly or through `NFTVault`. **Inconsistencies.** The `FoundingFrogVault` contract inherits the `NFTVault`. The specification on the provided documentation describes that the *governance* can increase the voting power of some users by calling the function `NFTVault.updateMultiplier`. Similarly, the documentation also specifies that the `FriendlyDAOVault` allows *governance* to arbitrarily assign voting power to any address by calling the function `updateDAOAndTotalWeight`. We checked this inconsistency with the client, who explained that the `ImmutableOwner` and `GovernanceOnly` contracts represent the same functionality. They reported that the existence of both contracts is an error. **Recommendation.** Consider updating the documentation to avoid code misconceptions.

8 Test Suite Evaluation

8.1 Contracts Compilation Output

```
$ make compile
yarn run hardhat compile
yarn run v1.22.17
$ .../Audits/Gyroscope/governance/node_modules/.bin/hardhat compile
Downloading compiler 0.8.17
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing
↪ "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for
↪ non-open-source code. Please see https://spdx.org for more information.
--> libraries/LogExpMath.sol


Warning: Unreachable code.
  --> contracts/testing/RaisingERC20.sol:11:9:
   |
11 |         return 0;
   |         ^^^^^^^^


Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/emergency_recovery_multisig/NoSafeManagementByMultisig.sol:38:9:
   |
38 |         uint256 value,
   |         ^^^^^^^^^^^^^


Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/emergency_recovery_multisig/NoSafeManagementByMultisig.sol:40:9:
   |
40 |         Enum.Operation operation,
   |         ^^^^^^^^^^^^^^^^^^^^^^^^


Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/testing/MockVault.sol:15:32:
   |
15 |     function getRawVotingPower(address user) external view returns (uint256) {
   |                                ^^^^^^^^^^^^


Warning: Function state mutability can be restricted to view
   --> contracts/GovernanceManager.sol:226:5:
    |
226 |     function _toVoteTotals(
    |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Function state mutability can be restricted to view
  --> contracts/RecruitNFT.sol:65:5:
   |
65 |     function _requireValidProof(
   |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Function state mutability can be restricted to view
  --> contracts/emergency_recovery_multisig/NoSafeManagementByMultisig.sol:36:5:
   |
36 |     function checkTransaction(
   |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Function state mutability can be restricted to pure
  --> contracts/emergency_recovery_multisig/NoSafeManagementByMultisig.sol:67:5:
   |
67 |     function checkAfterExecution(bytes32, bool) external {
   |     ^ (Relevant source part starts here and spans across multiple lines).
```

```
Warning: Function state mutability can be restricted to pure
  --> contracts/testing/MockVotingAggregator.sol:51:5:
   |
51 |     function getVaultWeight(address) external view returns (uint256) {
   |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Function state mutability can be restricted to pure
  --> contracts/testing/MockVotingAggregator.sol:55:5:
   |
55 |     function listVaults()
   |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Function state mutability can be restricted to pure
  --> contracts/testing/MockVotingAggregator.sol:63:5:
   |
63 |     function setSchedule(
   |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Function state mutability can be restricted to pure
 --> contracts/testing/RaisingERC20.sol:9:5:
  |
9 |     function totalSupply() public view override returns (uint256) {
  |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Contract code size is 24740 bytes and exceeds 24576 bytes (a limit introduced in Spurious Dragon). This
→  contract may not be deployable on Mainnet. Consider enabling the optimizer (with a low "runs" value!), turning off
→  revert strings, or using libraries.
  --> @gnosis.pm/safe-contracts/contracts/GnosisSafe.sol:19:1:
   |
19 | contract GnosisSafe is
   | ^ (Relevant source part starts here and spans across multiple lines).


Compiled 89 Solidity files successfully
  Done in 20.38s.
```

8.2 Tests Output

```
$ brownie test
Brownie v1.19.2 - Python development framework for Ethereum

.../Audits/Gyroscope/governance/.venv/lib/python3.9/site-packages/brownie/project/scripts.py:174: ImportWarning:
↪ test_founding_frog_vault.py, unable to determine import spec for 'conftest', the --update flag may not work
↪ correctly with this test file
  warnings.warn(
=================== test session starts =====================
platform darwin -- Python 3.9.7, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: .../Audits/Gyroscope/governance
plugins: eth-brownie-1.19.2, forked-1.4.0, web3-5.31.1, xdist-1.34.0, hypothesis-6.27.3
collected 127 items
This version of µWS is not compatible with your Node.js build:

Error: Cannot find module './uws_darwin_arm64_88.node'
Falling back to a NodeJS implementation; performance may be degraded.



Launching 'ganache-cli --chain.vmErrorsOnRPCResponse true --server.port 8545 --miner.blockGasLimit 12000000
↪ --wallet.totalAccounts 10 --hardfork london --wallet.mnemonic brownie'...

tests/test_action_tier_config.py ...
↪ [  2%]
tests/test_emergency_recovery.py ...........
↪ [ 11%]
tests/test_emergency_recovery_safe.py EEEEEEEEEEEEEE
↪ [ 22%]
tests/test_governance_manager.py .................
↪ [ 37%]
tests/test_liquidity_mining.py ...
↪ [ 39%]
tests/test_recruit_nft.py ....
↪ [ 42%]
tests/test_voting_power_aggregator.py ......
↪ [ 47%]
tests/test_voting_power_history.py ..
↪ [ 48%]
tests/test_wrapped_erc20_with_ema.py ...
↪ [ 51%]
tests/tier_strategies/test_set_address_strategy.py ..
↪ [ 52%]
tests/tier_strategies/test_set_system_params_strategy.py ..
↪ [ 54%]
tests/tier_strategies/test_set_vault_fees_strategy.py ...
↪ [ 56%]
tests/tier_strategies/test_simple_threshold_strategy.py .....
↪ [ 60%]
tests/tier_strategies/test_static_tier_strategy.py .
↪ [ 61%]
tests/vaults/test_aggregate_lp_vault.py .
↪ [ 62%]
tests/vaults/test_founding_frog_vault.py .......
↪ [ 67%]
tests/vaults/test_friendly_dao_vault.py ...
↪ [ 70%]
tests/vaults/test_lp_vault.py .........
↪ [ 77%]
tests/vaults/test_nft_vault.py ...........................
↪ [ 99%]
tests/vaults/test_recruit_nft_vault.py .
↪ [100%]
```

8.3 Slither

All the relevant issues raised by Slither have been incorporated into the issues described in this report.

9 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

Disclaimer

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.