



**PALADIN**  
BLOCKCHAIN SECURITY

# Smart Contract Security Assessment

Preliminary Report

For Lynex

25 April 2024



[paladinsec.co](https://paladinsec.co)



[info@paladinsec.co](mailto:info@paladinsec.co)

# Table of Contents

Table of Contents	2
Disclaimer	4
1 Overview	5
1.1 Summary	5
1.2 Contracts Assessed	6
1.3 Findings Summary	7
1.3.1 VoterV5	8
1.3.2 GaugeLogic	8
1.3.3 VoterV5_Storage	9
1.3.4 VotingEscrowV2Upgradeable	9
1.3.5 GaugeV2	9
1.3.6 GaugeV2_CL	10
1.3.7 BribeV2	10
1.3.8 OptionTokenV3	11
1.3.9 RewardsDistributorV2	11
1.3.10 MinterUpgradeableV2	12
1.3.11 CLFeesVault	12
1.3.12 AlgebraV1Twap	12
1.3.13 DataStorageLibrary	12
2 Findings	13
2.1 Voter/VoterV5	13
2.1.1 Privileged Functions	14
2.1.2 Issues & Recommendations	15
2.2 Voter/VoterV5_GaugeLogic	28
2.2.1 Issues & Recommendations	29
2.3 Voter/VoterV5_Storage	30
2.3.1 Issues & Recommendations	30
2.4 VotingEscrowV2Upgradeable	31
2.4.1 Issues & Recommendations	33
2.5 Gauge/GaugeV2	38
2.5.1 Privileged Functions	38
2.5.2 Issues & Recommendations	39
2.6 Gauge/GaugeV2_CL	47

2.6.1 Privileged Functions	47
2.6.2 Issues & Recommendations	48
2.7 Gauge/BribeV2	49
2.7.1 Privileged Functions	50
2.7.2 Issues & Recommendations	51
2.8 Gauge/OptionTokenV3	56
2.8.1 Privileged Functions	57
2.8.2 Issues & Recommendations	58
2.9 Gauge/RewardsDistributorV2	65
2.9.1 Issues & Recommendations	66
2.10 Gauge/MinterUpgradeableV2	70
2.9.1 Privileged Functions	71
2.9.2 Issues & Recommendations	72
2.11 Gauges/CLFeesVault	75
2.11.1 Privileged Functions	75
2.11.2 Issues & Recommendations	76
2.12 TWAP/AlgebraV1Twap	79
2.12.1 Issues & Recommendations	79
2.13 TWAP/DataStorageLibrary	80
2.13.1 Issues & Recommendations	80



# Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocacy for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

# 1 Overview

This report has been prepared for Lynex on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

## 1.1 Summary

<b>Project Name</b>	Lynex
<b>URL</b>	<a href="https://www.lynex.fi/">https://www.lynex.fi/</a>
<b>Platform</b>	Ethereum
<b>Language</b>	Solidity
<b>Preliminary Contracts</b>	<a href="https://github.com/Lynexfi/lynex-contracts/pull/39/commits/d2eac416d902970ad6edf09375ad637c13c7a2bf">https://github.com/Lynexfi/lynex-contracts/pull/39/commits/d2eac416d902970ad6edf09375ad637c13c7a2bf</a> <a href="https://github.com/cryptoalgebra/AlgebraV1/blob/main/src/periphery/contracts/libraries/DataStorageLibrary.sol">https://github.com/cryptoalgebra/AlgebraV1/blob/main/src/periphery/contracts/libraries/DataStorageLibrary.sol</a>
<b>Resolution 1</b>	<a href="https://github.com/Lynexfi/lynex-contracts/tree/e2e32c66f5b7c4c2b0ddc5fc12220e92202cc241">https://github.com/Lynexfi/lynex-contracts/tree/e2e32c66f5b7c4c2b0ddc5fc12220e92202cc241</a>

## 1.2 Contracts Assessed

Name	Contract	Live Code Match
VoterV5		
VoterV5_Gauge Logic		
VotingEscrowV 2Upgradeable		
GaugeV2		
GaugeV2_CL		
BribeV2		
OptionTokenV3		
RewardsDistri butorV2		
MinterUpgrade ableV2		
CLFeesVault		
AlgebraV1Twap		
DataStorageLi brary		



## 1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● Governance	5	-	-	5
● High	9	5	-	4
● Medium	10	5	1	4
● Low	23	14	1	8
● Informational	16	7	-	6
Total	63	31	2	27

### Classification of Issues

Severity	Description
● Governance	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
● High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
● Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
● Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
● Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

## 1.3.1 VoterV5

ID	Severity	Summary	Status
01	GOV	Governance Privilege: Non-immutable parameters	ACKNOWLEDGED
02	HIGH	Reward mechanism is broken if gauges remain undistributed during one period	ACKNOWLEDGED
03	HIGH	Kill and revive will break accounting	✓ RESOLVED
04	HIGH	Gauge distribution will incorrectly allocate shares to killed gauges	✓ RESOLVED
05	MEDIUM	Whitelisted pools cannot be removed from the whitelist	✓ RESOLVED
06	MEDIUM	Lack of time-restriction makes voting process highly manipulatable by whales	ACKNOWLEDGED
07	MEDIUM	Malicious gauges cannot be killed as long as oLynx address is address(0)	✓ RESOLVED
08	LOW	The _voteDelay function will not work as intended	✓ RESOLVED
09	LOW	Gauge reward periods may not align with voting periods	ACKNOWLEDGED
10	LOW	Change towards 0Lynx or vice-versa can alter distribution method retroactively	ACKNOWLEDGED
11	LOW	Change to 0Lynx is immutable	✓ RESOLVED
12	LOW	Violation of checks-effects-interactions pattern	PARTIAL
13	INFO	Incompatibility with tokens that have a fee on transfer	ACKNOWLEDGED
14	INFO	Typographical issues	✓ RESOLVED

## 1.3.2 GaugeLogic

ID	Severity	Summary	Status
15	INFO	Typographical issues	✓ RESOLVED



### 1.3.3 VoterV5\_Storage

ID	Severity	Summary	Status
16	INFO	Typographical issues	

### 1.3.4 VotingEscrowV2Upgradeable

ID	Severity	Summary	Status
17	HIGH	Reactivation of claimed tokenId will result in stuck funds	✓ RESOLVED
18	HIGH	split and claim can trick users in NFT sales	✓ RESOLVED
19	MEDIUM	Reentrancy vulnerability allows delegation with parameter endTime = 0	✓ RESOLVED
20	MEDIUM	Token split will render delegation useless	ACKNOWLEDGED
21	LOW	getAccountDelegates can run out of gas	ACKNOWLEDGED
22	INFO	Incorrect dependency usage for upgradeable contract	ACKNOWLEDGED

### 1.3.5 GaugeV2

ID	Severity	Summary	Status
23	GOV	Governance Privilege: Parameter change	ACKNOWLEDGED
24	HIGH	Malicious users can permanently lock a balanceWithLock position	ACKNOWLEDGED
25	MEDIUM	Emergency withdraw will result in permanently locked rewards	✓ RESOLVED
26	MEDIUM	Users can spam the reward array with pointless tokens	PARTIAL
27	LOW	Reward rate can be intentionally decreased	ACKNOWLEDGED
28	LOW	Unbound length for addRewardToken can result in DoS	✓ RESOLVED
29	INFO	updateRewardToken may add the same rewardToken to the rewards array	✓ RESOLVED
30	INFO	emergencyWithdraw allows locked funds to be withdrawn	ACKNOWLEDGED
31	INFO	Unused variables and events	ACKNOWLEDGED

## 1.3.6 GaugeV2\_CL

ID	Severity	Summary	Status
32	GOV	Governance can lock the _claimFees function	ACKNOWLEDGED

## 1.3.7 BribeV2

ID	Severity	Summary	Status
33	GOV	Governance Privilege: Change of privileged addresses	ACKNOWLEDGED
34	HIGH	Burn will permanently lock rewards	ACKNOWLEDGED
35	MEDIUM	Users can bait other users by selling NFT with unclaimed rewards	ACKNOWLEDGED
36	LOW	Delegation logic can result in no rewards	ACKNOWLEDGED
37	LOW	balanceOf and balanceOfOwner are incorrect	✓ RESOLVED
38	INFO	Inconsistency in rewardPerToken function	✓ RESOLVED

## 1.3.8 OptionTokenV3

ID	Severity	Summary	Status
39	GOV	Governance Privilege: Full control over contract functionality	ACKNOWLEDGED
40	MEDIUM	getLockDurationForVeDiscount will not work if discount is increased	✓ RESOLVED
41	LOW	setLockDurationForMinVeDiscount can only be used to reduce lockDurationForMinVeDiscount	✓ RESOLVED
42	LOW	The contract may use an outdated settings if they are not updated accordingly	✓ RESOLVED
43	LOW	Insufficient slippage parameters for exerciseLp	✓ RESOLVED
44	LOW	veMaxDiscount can be set above discount	✓ RESOLVED
45	LOW	Lack of safeguard for setTwapSeconds	✓ RESOLVED
46	LOW	Lack of slippage parameters for exerciseLp	✓ RESOLVED
47	INFO	exerciseExternal seems unfinalized	✓ RESOLVED
48	INFO	TWAP oracle weakness can result in paymentAmount being exploited	ACKNOWLEDGED

## 1.3.9 RewardsDistributorV2

ID	Severity	Summary	Status
49	HIGH	Loop break within _claim can result in side-effects	✓ RESOLVED
50	HIGH	Burning of tokenId will result in permanently stuck rewards	ACKNOWLEDGED
51	MEDIUM	Accrued rewards can serve as bait if tokenId is unlocked	ACKNOWLEDGED
52	LOW	Off-by-one error will temporarily prevent claiming	✓ RESOLVED
53	INFO	Unreachable code-section within _checkpoint_token	✓ RESOLVED
54	INFO	Gas optimizations	✓ RESOLVED

### 1.3.10 MinterUpgradeableV2

ID	Severity	Summary	Status
55	LOW	Lack of epoch update may result in skipped epoch	ACKNOWLEDGED
56	LOW	_initialize can result in undistributed epoch rewards	ACKNOWLEDGED
57	LOW	Emission increase can happen retroactively	ACKNOWLEDGED
58	LOW	Share configuration variables are insufficiently guarded	✓ RESOLVED

### 1.3.11 CLFeesVault

ID	Severity	Summary	Status
59	LOW	Potential underflow in share calculation if MAX_REFERRAL_FEE is in a different denomination	✓ RESOLVED
60	LOW	Incorrect configuration can left partial fees unclaimed	✓ RESOLVED
61	INFO	Several events are not emitted	
62	INFO	Unused imports	

### 1.3.12 AlgebraV1Twap

No issues found.

### 1.3.13 DataStorageLibrary

ID	Severity	Summary	Status
63	INFO	TWAP only works for 136 years	ACKNOWLEDGED

## 2 Findings

---

### 2.1 Voter/VoterV5

VoterV5 is the entry contract for users to use their voting power. Once users either delegate voting power to themselves through their VE positions or receive voting power from another address, they are able to vote for gauges.

Users can externally invoke the following functions:

- `reset`: Users can reset their vote state for the current epoch
- `poke`: Users can repeat their most recent vote
- `vote`: Users can vote for gauges with their VP
- `claimRewards`: Users can claim rewards from a gauge
- `claimBribes`: Users can claim rewards from a bribe
- `claimFees`: Users can claim fees
- `createGauge`: Users can create one gauge
- `createGauges`: Users can create multiple gauges

VE owners will not only be able to vote for gauge allocation but they will also be generally incentivized to vote. They will receive LP Fees (`BribeV2`), general bribes (`BribeV2`) and even Lynex tokens (not directly for votes but rather for the VE position ownership). It is important to note that even if the delegates are responsible for voting, the incentives will still flow towards the original VE position owners.

Whenever an epoch is updated, VoterV5 calculates how much of the minted rewards flow to each gauge based on the previously determined weights during the voting process.

A novel concept is the introduction of the OptionsTokenV3 represented as oLynx. Any address with the VOTER\_ADMIN role can set a new oLynx token, which then will result in the distribution of oLynx to gauges instead of Lynex tokens.

## 2.1.1 Privileged Functions

- `setVoteDelay`
- `setMinter`
- `setOptionsToken`
- `refreshApprovals`
- `setGaugeDepositor`
- `setBribeFactory`
- `setPermissionsRegistry`
- `setNewBribes`
- `setInternalBribesFor`
- `setExternalBribeFor`
- `addFactory`
- `replaceFactory`
- `removeFactory`
- `whitelist`
- `whitelistPool`
- `blacklist`
- `killGauge`
- `reviveGauge`

## 2.1.2 Issues & Recommendations

<b>Issue #01</b>	<b>Governance Privilege: Non-immutable parameters</b>
<b>Severity</b>	<div><div></div> GOVERNANCE</div>
<b>Description</b>	<p>The VoterV5 contract handles several different functionalities, such as allocating votes to a bribe, claiming from bribes, allocating votes to gauges and much more.</p> <p>Any address with the VOTER_ADMIN role can change many important parameters. This can significantly alter the contract's behavior and result in many side-effects.</p>
<b>Recommendation</b>	Consider incorporating a Gnosis multi-signature contract as owner and ensuring that the Gnosis participants are trusted entities.
<b>Resolution</b>	<div><div></div> ACKNOWLEDGED</div> <p>The contracts will be owned by a multi-signature wallet.</p>

**Description**

Whenever an epoch has passed, the index variable is increased by `amount/totalWeight`. The epoch update can be either directly invoked within the Minter contract or within this contract using the `distribute(All)` function which then invokes the epoch update within the minter. Therefore, within `_updateForAfterDistribution`, the respective amount of tokens are allocated to the vault depending on the index increase and the vault's weight for this epoch: <https://github.com/Lynexfi/lynex-contracts/blob/d2eac416d902970ad6edf09375ad637c13c7a2bf/contracts/VoterV5/VoterV5.sol#L784>

This mechanism will not work if there is no distribution towards a gauge within one epoch and the index is increased afterwards. This is a valid scenario because it is not necessary that the `distribute` function is invoked for all gauges before the epoch is updated. Consider the following scenario:

1) There are currently two gauges:

- WETH/USDC
- WBTC/USDC

Both gauges have received a VP of 50.

2) `update_period` is invoked which then updates the index based on the `totalWeight` and the amount of rewards (L250).

For simplicity's sake, let's consider that there are  $100e18$  reward tokens and a `totalWeight` of  $100e18$ , this will set index to  $1e18$  (L690). Therefore, if `_distribute` is invoked, both gauges would receive  $50e18$  tokens (L784) and `supplyIndex[gauge]` is updated afterwards

3) The WBTC/USDC gauge is not distributed and therefore the `supplyIndex` of this gauge is not updated and is still zero.

4) For the next period, there are  $100e18$  reward tokens and only the WBTC/USDC vault got an allocation of  $100e18$  VP. Therefore, index is set to  $1e18+1e18$  (L690)

5) The `distribute` function is now invoked for the WBTC/USDC vault (L776).



---

Remember how the gauge got an allocation of 100e18 for the second period and how the index = 2e18 but supplyIndex for this gauge is zero. This will effectively attempt to distribute 200e18 tokens to the WBTC/USDC gauge, while the gauge should effectively only receive 50e18 (epoch 1) and 100e18 (epoch 2), thus breaking the whole mechanism.

This can also be exploited by users to create a new gauge at the very end of an epoch, vote for it in the hope that no one will update this gauge and then within the next epoch vote a large amount. This large amount will then additionally consume the index delta from the previous round.


This issue will inevitably occur sooner or later if the pool's array grows too large. At some point, users will only invoke `distribute` on their desired gauge to save gas because there is literally no incentive for anyone to invoke the `distributeAll` function.

---

<b>Recommendation</b>	Consider not allowing <code>update_period</code> to be called before all gauges have been marked as distributed for the previous epoch.
-----------------------	---

---

**Resolution**

 ACKNOWLEDGED

The team stated:

"We decided to go with a middleground and exteriorize that responsibility to a contract <https://lineascan.build/address/0xd405a345bb0f2b620a81e0fe77962b3d9ed84d3a#readProxyContract>, after all we have full 7 days to execute the distribution of all Gauges."

---

## Severity



HIGH SEVERITY

## Description

Consider the scenario where we are in a running epoch and votes have already been casted to different gauges, this will have the following storage impact:

```
weightsPerEpoch[time][gauge] = VP  
votes[voter][gauge] = VP  
totalWeightsPerEpoch[time] = VPAggregated
```

Of note is the `totalWeightsPerEpoch` mapping, as this will be used for the index calculation (L684).

If a gauge is now killed, the `totalWeightsPerEpoch` mapping is decreased by the corresponding amount (L379). This is correct because this gauge should not get an allocation.

Several problems will now arise if a gauge is revived because the `totalWeightsPerEpoch` mapping will not increase back to the old value:

- 1) Upon distribution (after the epoch update), the gauge will still get an allocation (due to the still existing `weightsPerEpoch` mapping) but the index variable was not correctly adjusted due to the missing increased `totalWeightsPerEpoch` when the gauge is revived.
- 2) If users `_reset` their votes (in the same epoch), this will decrease the `totalWeightsPerEpoch` mapping, when it actually already was decreased due to the kill interaction. This could also prevent `_reset` due to an underflow revert.

## Recommendation

Consider preventing the revival of gauges once they have been killed, at least for one period.

## Resolution



RESOLVED

A gauge cannot be killed and revived in the same epoch.

## Severity



## Description

Consider the following scenario:



1. Gauge XY is alive in period 604800 and receives votes.
2. Gauge XY is killed in the middle of the epoch, `isAlive` is set to false and `totalWeightsPerEpoch[time]` is decreased by the gauge's votes.
3. `distributeAll` is called and `index` is increased — note that the denominator `totalWeightsPerEpoch[time]` does not include the votes of killed vaults, hence `index` does not include the killed gauge.
4. `_updateForAfterDistribution` is invoked. Due to the fact that `weightsPerEpoch[time][pool]` (supplied) was not reset during `killGauge`, the gauge will still receive a share, which is transferred to the minter. This means that there are insufficient rewards for the other gauges and this will result in a revert.

## Recommendation

Consider resetting `weightsPerEpoch[time][pool]` whenever a gauge is killed.

## Resolution



<b>Issue #05</b>	<b>Whitelisted pools cannot be removed from the whitelist</b>
<b>Severity</b>	 MEDIUM SEVERITY
<b>Description</b>	<p>Privileged addresses can add pools to the <code>isWhitelistedPool</code> mapping. They can then deploy a gauge for this pool via the <code>createGauge</code> function using <code>gaugeType = 1</code>.</p> <p>A problem will arise if a pool needs to be removed from the <code>isWhitelistedPool</code> mapping, as such an operation simply does not exist, rendering such a setting immutable.</p>
<b>Recommendation</b>	Consider implementing a function that allows the removal of pools from the <code>isWhitelistedPool</code> mapping.
<b>Resolution</b>	 RESOLVED
	A <code>blacklistPool</code> function has been added.



## Severity

 MEDIUM SEVERITY

## Description

Users can vote within a running epoch and receive rewards once this epoch has passed. Users can vote and then revoke their votes again. This practice in itself is not necessarily an issue, however becomes one due to the lack of time-restrictions.

Consider the following scenario:

- 1) Alice is a VeLynex whale and allocates her votes to the WETH/USDC gauge – she already knows this gauge will receive a lot of LP Fees and bribes.
- 2) Other users can see that this Gauge already has a large voting allocation and because rewards are distributed based on the percentage of votes in the whole gauge, there is not much economical incentive for users to vote in the already diluted gauge. Therefore, it is likely that Alice stays the majority owner of the pool, it might actually even be the case that no one else votes for this pool.
- 3) Only a few blocks are left until the epoch ends and there are still not many votes besides Alice's vote for the gauge. Alice now calls vote and allocates her votes to a completely different gauge, while keeping a majority stake in the WETH/USDC pool.
- 4) Alice has now effectively received the majority of the bribes for the WETH/USDC pool but also bribes for whatever pool she decides to vote in the last block.

The root-cause of this issue is that users can change their votes up to the last block in the current epoch, which makes this mechanism highly vulnerable for manipulation.

## Recommendation



Consider not allowing users to change their vote for certain amount of time before the end of the period.



## Resolution

 ACKNOWLEDGED

The team stated:


"We will be using the Vote Delay function as a partial protection against this."

<b>Issue #07</b>	<b>Malicious gauges cannot be killed as long as oLynx address is address(0)</b>
<b>Severity</b>	 MEDIUM SEVERITY
<b>Location</b>	<u>L364 - 382</u> function killGauge(address _gauge) external {
<b>Description</b>	<p>When calling killGauge, the contract will try to reset oLynx approval <code>IERC20(oLynx).approve(_gauge, 0)</code>. However, if oLynx is still unset and is <code>address(0)</code>, this will revert, preventing any gauge from being killed as long as the oLynx token is not set.</p> <p>The same issue is present within <code>reviveGauge</code>, but currently oLynx cannot be set back to <code>address(0)</code> so this should not be an issue. However, for better code quality, the check should be added in case the code is updated and allow to unset the oLynx token again.</p>
<b>Recommendation</b>	Consider checking that oLynx is not <code>address(0)</code> before resetting its approval.
<b>Resolution</b>	 RESOLVED

Issue #08	The <code>_voteDelay</code> function will not work as intended
Severity	 LOW SEVERITY
Location	<u>L556-558</u> function <code>_voteDelay(address _voter)</code> internal view {
Description	<p>The <code>lastVoted[_voter]</code> value is set to <code>_epochTimestamp() + 1;</code> (L410, L460, 470) but the <code>_voteDelay</code> check uses <code>block.timestamp</code>.</p> <p>This means that if the <code>VOTE_DELAY</code> is set to 1 day, the delay will only work during the first day of the <code>activePeriod</code>. Once the first day is over, the delay will not work anymore as the <code>lastVoted</code> value will always be set to the same value.</p>
Recommendation	Consider setting the <code>lastVoted[_voter]</code> value to <code>block.timestamp + 1</code> instead of <code>_epochTimestamp() + 1</code> unless this behaviour is the expected one.
Resolution	 RESOLVED



## Severity

 LOW SEVERITY

## Description

Users can vote within a running period. After a period has passed, these votes are used to determine the reward distribution to vaults via `update_period` within the `MinterUpgradeableV2` contract (L218). This will then transfer reward tokens towards the `VoterV5` contract which are then distributed amongst the gauges.

The problem is the fact that the `distributeAll` function must be manually invoked and only once that has happened, then rewards are actually allocated to gauges. This can result in a slight inconsistency for reward allocations if for example a gauge is only notified at a later timestamp, effectively creating a gap between gauge reward start and epoch periods.

## Recommendation

This is a design choice due to the architectural environment — a fix for this issue would require large adjustments to several contracts



## Resolution



 ACKNOWLEDGED



The team stated:



"Refactoring is not practical in this context due to it requiring a major overhaul of the system."



Issue #10	Change towards 0Lynx or vice-versa can alter distribution method retroactively
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	<p>It is possible for privileged addresses to invoke the setOptionsToken function which will then set a 0Lynx address. This means all gauge rewards will be distributed as 0Lynx instead of Lynex.</p> <p>This is a significant disadvantage for users. The problem is that rewards can be already allocated but not distributed, which would then mean the reward mechanism is altered retroactively and gauges would receive 0Lynx instead of Lynex.</p>
<b>Recommendation</b>	Consider ensuring that all outstanding distributions have occurred before this function is triggered.
<b>Resolution</b>	 ACKNOWLEDGED <p>The team stated:          "We made a note about this in the contract and docs."</p>

Issue #11	Change to 0Lynx is immutable
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	<p>It is possible for privileged addresses to invoke the setOptionsToken function which will then set an 0Lynx address. This means all gauge rewards will be distributed as 0Lynx instead of Lynex.</p> <p>This configuration is immutable, which means once the 0Lynx address is set, there is no way to switch back to normal Lynex distribution.</p>
<b>Recommendation</b>	Consider if this is a desired design decision, if not, consider allowing this change to be reverted and set the 0Lynx address to zero.
<b>Resolution</b>	 RESOLVED

<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	<p>Throughout the contract, there several sections which violate the checks-effects-interactions pattern. To ensure a protection against invalid states, all external calls should strictly be implemented after any checks and effects (state variable changes).</p> <p><a href="https://github.com/Lynexfi/lynex-contracts/blob/d2eac416d902970ad6edf09375ad637c13c7a2bf/contracts/VoterV5/VoterV5.sol#L682">https://github.com/Lynexfi/lynex-contracts/blob/d2eac416d902970ad6edf09375ad637c13c7a2bf/contracts/VoterV5/VoterV5.sol#L682</a></p>
<b>Recommendation</b>	Consider following the CEI pattern.
<b>Resolution</b>	 PARTIALLY RESOLVED <p>There are still instances where external calls happen before state updates. e.g. killGauge</p>

<b>Issue #13</b>	<b>Incompatibility with tokens that have a fee on transfer</b>
<b>Severity</b>	 INFORMATIONAL
<b>Location</b>	<p><a href="https://github.com/Lynexfi/lynex-contracts/blob/d2eac416d902970ad6edf09375ad637c13c7a2bf/contracts/VoterV5/VoterV5.sol#L688">https://github.com/Lynexfi/lynex-contracts/blob/d2eac416d902970ad6edf09375ad637c13c7a2bf/contracts/VoterV5/VoterV5.sol#L688</a></p>
<b>Description</b>	<p>This contract is not compatible with tokens that have a fee on transfer. If these token types are used for any purpose within the contract, there will be down-stream issues and inherently break the accounting.</p>
<b>Recommendation</b>	Consider not using these tokens.
<b>Resolution</b>	 ACKNOWLEDGED <p>The team stated: "Noted in the comments at the top of the contract."</p>

Issue #14	Typographical issues
-----------	----------------------

Severity	 INFORMATIONAL
----------	---

Description	<p><u>L251-255</u></p> <pre>function _setExternalBribe(address _gauge, address _external) private {     require(_external.code.length &gt; 0, "!contract");     emit SetBribeFor(false, internal_bribes[_gauge], _external, _gauge);     external_bribes[_gauge] = _external; }</pre> <p>The event uses <code>internal_bribes[_gauge]</code>, but it should use <code>external_bribes[_gauge]</code> instead. This could confuse third party toolings that might use this value for indexing purposes.</p> <p>—</p> <p>The Attach/Detach events are not used.</p>
-------------	---

Recommendation	Consider fixing the typographical issues.
----------------	---

Resolution	 RESOLVED
------------	--



---

## 2.2 Voter/VoterV5\_GaugeLogic



VoterV5\_GaugeLogic is solely meant as an implementation contract and used for delegatecalls by the VoterV5 contract. It incorporates logic for deploying a gauge and corresponding bribe contracts.

Anyone can invoke the createGauge function which creates gauges for two types of pools:

- 1) Gauge for standard LP tokens
- 2) Gauge for Hypervisor positions: "A Uniswap V2-like interface with fungible liquidity to Uniswap V3"

Upon the creation of a gauge for a specific pool, two BribeV2 contracts are deployed which serve as incentive contracts for voters to vote for a specific gauge.

## 2.2.1 Issues & Recommendations

Issue #15	Typographical issues
Severity	 INFORMATIONAL
Description	<p><u>L23-25</u></p> <pre>/// @notice This contract contains the logic for creating gauges in the VoterV4 system. It is used to save contract /// size in VoterV4 by separating out expensive logic. /// @dev This contract MUST be called from VoterV4 through delegatecall().</pre> <p>The natspec comments are outdated as they mention VoterV4 instead of the new VoterV5. Consider updating the natspec comment of the contract.</p>
Recommendation	Consider fixing the typographical issues.
Resolution	 RESOLVED

## 2.3 Voter/VoterV5\_Storage

VoterV5\_Storage is solely meant as a storage contract that keeps all the storage variables for the Voter.

### 2.3.1 Issues & Recommendations

Issue #16	Typographical issues
Severity	<span>INFORMATIONAL</span>
Description	<p><u>L 23-25</u></p> <pre>/// @title VoterV4_Storage /// @notice This contract contains the storage variables for VoterV4. /// @dev This contract is used to ensure both VoterV4 and VoterV4_GaugeLogic have access to the same storage variables /// in the correct slots. They MUST both extend this.</pre> <p>The natspec comments are outdated as they mention VoterV4 instead of the new VoterV5. Consider updating the natspec comment of the contract.</p>
Recommendation	Consider fixing the typographical issues.
Resolution	

---

## 2.4 VotingEscrowV2Upgradeable

VotingEscrowV2Upgradeable is the backbone of the architecture. It allows users to wrap/lock their Lynex token and receive an NFT. Depending on the locked amount and time, users will gain voting power (VP) which they can use for voting purposes within the VoterV5 contract.

It is important to note that the voting power algorithm is out of scope and handled in external contracts. It is therefore assumed that users cannot manipulate their voting power. The same applies to the delegation algorithm. It is expected that delegating and the removal of a delegate works as expected without space for manipulation.

Users can execute the following actions:

- `createLock`: A user can create a simple lock with the desired amount and duration. This lock is inherently granted to the user without any delegation.
- `createLockFor`: A user can create a simple lock with the desired amount and duration for another user. This lock is inherently granted to the recipient without any delegation.
- `createDelegatedLockFor`: A user can create a simple lock with the desired amount and duration for another user. This lock is inherently granted to the recipient but with an optional delegation possibility.
- `increaseAmount`: A user can increase the lock amount of any `tokenId`, by depositing on behalf of it.
- `increaseUnlockTime`: A user can increase the `unlockTime` of any approved `tokenId`.
- `unlockPermanent`: A user can remove the permanent status of any approved `tokenId`. This will set the `unlockTime` to `block.timestamp + 2 years`.



- **claim:** A user can claim any approved and expired lock. This will transfer out the amount and set the tokenId states to zero.
- **merge:** A user can merge one tokenId to another tokenId. This will simply merge amounts and use the longer of both endTimes. The user must be approved for both tokenIds.
- **split:** A user can split an approved tokenId to multiple new tokenIds. This will simply decrease the amount of the original tokenId and mint new tokenIds with the same unlockTime and the corresponding amounts.
- **burn:** A user can burn their own tokenId if the amount is zero.
- **delegate:** A user can delegate any approved tokenId to a delegatee.

**Note:** We highly recommend including the ERC5725Upgradeable and EscrowDelegateStorage contract and all dependencies into the scope for the audit.





## 2.4.1 Issues & Recommendations

<b>Issue #17</b>	<b>Reactivation of claimed tokenId will result in stuck funds</b>
<b>Severity</b>	 HIGH SEVERITY
<b>Description</b>	<p>It is possible for a claimed tokenId to be reactivated by simply invoking <code>increaseUnlockTime</code> with <code>permanent = true</code> (L366, L305)</p> <p>Afterwards, the <code>increaseAmount</code> function can be invoked and the amount for the corresponding tokenId is simply increased (L299).</p> <p>If now the user wants to claim the amount once the tokenId is unlocked, this will not work because of L412 and ERC5725Upgradeable.sol::L110.</p> <pre>claimablePayout = vestedPayout(tokenId) - _payoutClaimed[tokenId]</pre> <p><code>vestedPayout</code> in this scenario is simply the amount for tokenId. However, due to the fact that <code>_payoutClaimed</code> was already increased in the first claim, this will result in either less tokens being claimable or an underflow.</p>
<b>Recommendation</b>	Consider not allowing the reactivation of a claimed tokenId.
<b>Resolution</b>	 RESOLVED
	Increasing a claimed token will now fail.

## Severity

 HIGH SEVERITY

## Description

The `split` function allows the `tokenId` owner to split one `tokenId` into multiple `tokenIds`. This is done by simply deducting the amount from the original `tokenId` and allocating it to new `tokenIds` based on the provided `_weight` parameter. The `unlockTime` will be the same as the original `tokenId` and the `startTime` will be set to the current `block.timestamp`.

If such a `tokenId` is listed on a marketplace for a specific value and the seller frontruns the purchase transaction by splitting the `tokenId` such that the original `tokenId` has an amount of zero, the buyer was effectively tricked into buying a worthless NFT.

## Recommendation

Consider implementing a grace period between token splits and transfers. For example a transfer should not be allowed if a `tokenId` was split in the last 60 seconds. The same approach can be used for the `claim` and `increaseUnlockTime` functions.

It might actually make the most sense to implement a `nftModified` mapping with the timestamp of the last modification and a check within the `_beforeTokenTransfer` function that ensures the `tokenId` was not modified recently.

## Resolution

 RESOLVED

A check that the token has not been modified in the last 60 seconds was added.

## Severity



## Description

Within the `_createLock` function, `_safeMint` is invoked which mints the NFT to the recipient. The `_safeMint` vulnerability is clear - it invokes a hook upon receipt, which allows the recipient to execute arbitrary logic. Upon inspection, the following functions can be invoked:

- 1) `claim`
- 2) `burn`
- 3) `delegate`
- 4) `transferFrom`
- 5) `safeTransferFrom`

After the consideration of these functions, we came to the conclusion that a potential attack-vector can come up if the `delegate` function is invoked right after the mint has happened. This allows the recipient to delegate the `tokenId` with `endTime = 0` since this variable is not set yet. It is clear that a `delegate` happens afterwards as well. However, it will return early because there is no change in the delegatee: <https://github.com/Lynexfi/lynex-contracts/blob/d2eac416d902970ad6edf09375ad637c13c7a2bf/contracts/VoterV5/VotingEscrow/libraries/EscrowDelegateCheckpoints.sol#L318>

Due to the fact that the `EscrowDelegateCheckpoints` logic is out of scope and non-trivial, we did not further look into the impact of this issue. It can be a non-issue but can also be a critical one, depending on the usage of `endTime`. Remember, `endTime = 0` represents a permanently locked position, which is not true in that scenario.

## Recommendation

Consider looking further into this issue. Consider including the whole scope into the audit process.

## Resolution



`safeMint` was removed and the functions are now all protected with a non-reentrant check.

## Severity

 MEDIUM SEVERITY

## Description

One novel feature of VotingEscrowV2Upgradeable is the optional delegation of their own lock to a delegate. This means that users can create a lock for themselves but delegate the voting power to another address. With that, we have identified an issue within the `split` function: the creation of the new `tokenIds` will not account for the delegatee of the original `tokenId`, effectively removing the voting power.


This can become problematic if for example Alice has a `tokenId` which is delegated to Bob. Alice now splits the `tokenId` on Wednesday at 11.59PM. Once a new epoch begins on Thursday at 12:00 AM, Bob will not have the voting power he used to have. Alice has unintentionally removed Bob's voting power.

The problem is that the timestamp of the start of the epoch has passed and even if Alice sees the issue, delegating to Bob will only allow Bob to consume the voting power in the subsequent epoch.

## Recommendation

Consider delegating the newly created `tokenIds` to the previously corresponding delegatee. This can be trivially done by invoking the `getLockDelegatee` function for the original `lockId` and passing the result parameter as fourth argument to the `_createLock` function.

## Resolution

 ACKNOWLEDGED

The side effect was added to the documentation of the function.

Issue #21 <b>getAccountDelegates can run out of gas</b>	
Severity	<span>LOW SEVERITY</span>
Description	getAccountDelegates loops over all tokenIds of the account. In the scenario where the array becomes unreasonably large, this loop will consume excessive gas which can result in a function revert due to the out of gas issue.
Recommendation	Consider incorporating a paginated approach.
Resolution	<span>ACKNOWLEDGED</span> The side effect was added to the documentation of the function.

Issue #22 <b>Incorrect dependency usage for upgradeable contract</b>	
Severity	<span>INFORMATIONAL</span>
Location	<u>L7</u>
Description	Dependencies which are used for a proxy implementation should always incorporate the <code>_gap[ ]</code> variable which reserves space for future upgrades. While this contract is meant to be an implementation for a proxy contract, it does not follow this important standard.
Recommendation	Consider incorporating <code>ReentrancyGuardUpgradeable</code> .
Resolution	<span>ACKNOWLEDGED</span> The side effect was added to the documentation of the function. Note that any upgrade should be done very carefully to not corrupt the storage of the proxy.

---

## 2.5 Gauge/GaugeV2

GaugeV2 is a simple staking contract where users can stake their V1 LP tokens or Hypervisor positions. It is built on top of the Synthetix StakingRewards contract and incorporates its reward algorithm with multiple reward tokens. As an additional layer of rewards, a gaugeRewarder can be added on top which is triggered on each balance changing interaction and presumably distributes additional rewards.

In addition to the standard `deposit` and `withdraw` functions, the contract includes a `depositWithLock` function which locks the deposit for a certain time. It is notable that this feature does not offer any benefits compared to a normal deposit and is meant to be used by the `OptionsTokenV3`.

Disclaimer: GaugeRewarder is out of scope, we highly recommend including it into the audit scope.

### 2.5.1 Privileged Functions

- `setDistribution`
- `setGaugeRewarder`
- `setInternalBribe`
- `activateEmergencyMode`
- `stopEmergencyMode`
- `updateRewardToken`
- `addRewardToken`

## 2.5.2 Issues & Recommendations

Issue #23	Governance Privilege: Parameter change
Severity	<div><div></div> GOVERNANCE</div>
Description	The governance of this contract has several privileges to call certain functions that can drastically alter the contracts behavior. For instance, the gaugeRewarder contract can be set to an incompatible contract, which then could block withdrawals.
Recommendation	Consider incorporating a Gnosis multisignature contract as owner and ensuring that the Gnosis participants are trusted entities.
Resolution	<div><div></div> ACKNOWLEDGED</div> <p>The contract will be owner by a multi-signature wallet.</p>

## Severity

 HIGH SEVERITY

## Description

The depositWithLock function allows the OptionTokenV3 to deposit for a user with the side-effect that this position is locked for a certain time frame.


Within OptionTokenV3, this function can be invoked with an arbitrary recipient. This means any user can literally deposit 1 WEI on behalf of any user before the lockEnd has been reached. This will then re-set lockEnd to the new lock duration, effectively increasing the locking period. This can be repeated over and over to permanently lock these positions.

This will also have other side-effects like extending initially short periods or frontrunning and DoSing interactions with a short \_lockDuration.

## Recommendation

Consider not allowing deposits on behalf of an arbitrary recipient.

## Resolution

 ACKNOWLEDGED

The team stated: "There is only one strategy that 'is connected' and that is LYNX/USDC vAMM. This is a feature we are not currently using or promoting, so it is unlikely users have existing locks vulnerable to this attack.

We have no plans to use such a feature in the near term and its scope is a single pool connected to the options contract. Therefore we acknowledge the issue and mitigate it by not promoting its usage."



## Severity

 MEDIUM SEVERITY

## Description

The Synthetix reward mechanism increases the `rewardPerTokenStored` mapping upon every interaction. This means that the rewards are distributed on every occasion based on the staked supply.

Consider a scenario where Alice and Bob have both deposited 100 tokens, with a `rewardRate` of  $1e18$  and 100 seconds passed. A third address deposits next, which results in `rewardPerTokenStored` being  $50e18$  after 100 seconds, allowing both Alice and Bob to claim  $100e18$  tokens each. At this point, Alice and Bob's rewards are not updated yet.

If Alice calls `emergencyWithdraw`, this will not alter `rewardPerTokenStored` but Bob can only claim  $100e18$  tokens. Alice's reward tokens will be permanently stuck in the contract.

This is a fundamental difference to the masterchef mechanism as within the masterchef algorithm, the reward update is handled differently and lost rewards due to emergency withdrawals are simply allocated amongst the leftover stakers.

## Recommendation

Consider incorporating a function which allows a privileged address to withdraw these locked funds.

## Resolution

 RESOLVED

A sweep function was added.

**Issue #26****Users can spam the reward array with pointless tokens****Severity** MEDIUM SEVERITY**Description**

GaugeV2 can handle multiple reward tokens which can be added via the `addRewardToken` function without any limitation. Interestingly, `notifyRewardAmount` can also be invoked with previously unused reward tokens. The clue is here that the reward token must pass the following check:

```
require(IVoterV5(DISTRIBUTION).isWhitelisted(rewardAddress))
```

This also includes any tokens for gauge creation, hence `token0/` `token1` from all LP pairs can be used as reward tokens.



Users can therefore spam the list with pointless tokens until the queue length is exhausted.



**Recommendation**

Consider not allowing users to manually invoke the `notifyRewardAmount` function.

**Resolution** PARTIALLY RESOLVED



The owner can now remove any reward token. This issue is only partially resolved because the malicious user can still re-add other undesirable tokens later, possibly spamming the protocol with fake tokens which will make it hard for the Admin to remove all of them.

Issue #27 Reward rate can be intentionally decreased	
Severity	 LOW SEVERITY
Description	<p>Normally, notifyRewardAmount is invoked whenever reward tokens are distributed to the gauge. This will determine the reward rate over the next 7 days.</p> <p>If the rewardRate is now 1e18 and 3.5 days have passed, a user can simply invoke notifyRewardAmount with 1 wei, which will decrease the reward rate and increase the reward period.</p>
Recommendation	Consider not allowing unprivileged addresses to invoke the notifyRewardAmount function.
Resolution	 ACKNOWLEDGED

Issue #28 Unbound length for addRewardToken can result in DoS	
Severity	 LOW SEVERITY
Description	<p>The addRewardToken function allows the contract owner to add reward tokens to the rewards array. This array is then used to determine the loop length within the _updateRewardForAllTokens function. If this becomes too large, the reward algorithm will not work anymore.</p> <p>The real issue here is that the rewards array can never be decreased.</p>
Recommendation	Consider setting an upper boundary for this function similar to the boundary within the _notifyRewardAmount function.
Resolution	 RESOLVED The owner can now remove tokens.

Issue #29	updateRewardToken may add the same rewardToken to the rewards array
Severity	<div> <div></div> INFORMATIONAL </div>
Location	<p><u>L156-168</u></p> <pre> function updateRewardToken() external onlyOwner { // @audit-ok     isReward[address(rewardToken)] = false;     address rewardAddress = IVoterV5(DISTRIBUTION).oLynx();     if (rewardAddress == address(0)) {         rewardAddress = IVoterV5(DISTRIBUTION).base();     }      if (!isReward[rewardAddress]) {         isReward[rewardAddress] = true;     }     rewards.push(rewardAddress);     rewardToken = IERC20(rewardAddress); } </pre>
Description	<p>updateRewardToken adds rewardAddress even if it has already been added.</p>
Recommendation	<p>Consider adding the rewardAddress to the array only if it has not already been added, i.e., only when isReward[rewardAddress] is false.</p>
Resolution	<div> <div></div> RESOLVED </div>



<b>Issue #30</b>	<b>emergencyWithdraw allows locked funds to be withdrawn</b>
<b>Severity</b>	 INFORMATIONAL
<b>Description</b>	emergencyWithdraw can be invoked when the contract is in the "emergency" state. This allows users to withdraw their stake without caring about the rewards. It has to be noted that this will allow users to withdraw their locked balance as well.
<b>Recommendation</b>	This is likely a design choice — in the scenario where it is undesired, we recommend incorporating the freeBalance check.
<b>Resolution</b>	 ACKNOWLEDGED This is desired behavior.



Issue #31	Unused variables, events and imports
<b>Severity</b>	<div data-bbox="451 165 480 199"></div> INFORMATIONAL
<b>Location</b>	<p data-bbox="451 248 501 282"><u>L32</u></p> <pre data-bbox="451 293 932 327">address public external_bribe;</pre> <p data-bbox="451 383 501 416"><u>L45</u></p> <pre data-bbox="451 427 1270 506">mapping(address =&gt; mapping(address =&gt; uint)) public lastEarn;</pre> <p data-bbox="451 562 501 595"><u>L57</u></p> <pre data-bbox="451 607 995 640">event RewardAdded(uint256 reward);</pre> <p data-bbox="451 696 501 730"><u>L74</u></p> <pre data-bbox="451 741 772 775">error OnlyAllowed();</pre>
<b>Description</b>	<p data-bbox="451 808 1426 943">Variables and events which are unused will unnecessarily increase the contract size for no reason and will confuse third-party reviewers.</p> <p data-bbox="451 976 1251 1010">The variables and events in the lines above are not used.</p>
<b>Recommendation</b>	Consider removing the unused variables, events and imports.
<b>Resolution</b>	<div data-bbox="451 1122 480 1155"></div> ACKNOWLEDGED



---



## 2.6 Gauge/GaugeV2\_CL

GaugeV2\_CL is a simple extension for the GaugeV2 contract. It claims fees from the feeVault contract and distributes them as reward.

### 2.6.1 Privileged Functions

- `setFeeVault`

## 2.6.2 Issues & Recommendations

<b>Issue #32</b>	<b>Governance can lock the <code>_claimFees</code> function</b>
<b>Severity</b>	 GOVERNANCE
<b>Description</b>	<p><code>_claimFees</code> claims fees from the corresponding <code>feeVault</code> contract. The contract owner can change the <code>feeVault</code> contract to an arbitrary address.</p> <p>This privilege could be abused to temporarily block the <code>_claimFees</code> function until sufficient rewards have been accumulated and then set it back to the real <code>feeVault</code> such that a large sum of rewards is distributed at once, where the owner can profit from.</p>
<b>Recommendation</b>	Consider incorporating a Gnosis multi-signature contract as owner and ensuring that the Gnosis participants are trusted entities.
<b>Resolution</b>	 ACKNOWLEDGED
	The owner will be a multi-signature contract.



---

## 2.7 Gauge/BribeV2

BribeV2 is an incentive distribution contract for voters. Once users use their voting power to vote for a specific gauge within the VoterV5 contract, the nominal voting power amount is then allocated to the voter within the BribeV2 contract for the corresponding period.

After each period, users will receive rewards based on their provided VP, the total VP for this epoch and the aggregated reward allocation during this epoch.

Each gauge has two bribes:

- 1) LP Fees (internal)
- 2) External bribes (external)

While the first bribe simply distributes the LP fees from the pair, the second bribe allows third-parties to allocate rewards which incentivizes users to vote for a specific gauge. This is a popular mechanism for protocols to provide incentives and grow their liquidity.



The owner of the contract or the BribeFactory can whitelist tokens which can then be distributed as rewards. While for Bribe 1), these will likely be only the tokens which form the LP Pair, for Bribe 2) these can be many different tokens.

BribeV2 remains unaffected from any period updates. Aside from the deposit and withdraw functions that are invoked by the VoterV5 contract, it forms a closed system. Users are only allowed to claim their rewards once an epoch has been updated. This ensures a fair and correct distribution.

## 2.7.1 Privileged Functions

- `addRewardTokens`
- `addRewardToken`
- `recoverERC20AndUpdateData`
- `emergencyRecoverERC20`
- `setVoter`
- `setMinter`
- `setOwner`

## 2.7.2 Issues & Recommendations

<b>Issue #33</b>	<b>Governance Privilege: Change of privileged addresses</b>
<b>Severity</b>	 GOVERNANCE
<b>Description</b>	The governance of this contract has several privileges for invoking certain functions that can drastically alter the contracts behavior. For instance, the voter address could be changed which would then allow for manipulating the supply or breaking cross-contract interactions.
<b>Recommendation</b>	Consider incorporating a Gnosis multi-signature contract as owner and ensuring that the Gnosis participants are trusted entities.
<b>Resolution</b>	 ACKNOWLEDGED The owner will be a multi-signature contract.



## Severity

 HIGH SEVERITY

## Description

Users receive rewards in the form of LP Fees or Bribes as lock incentive. These rewards can always be claimed for the corresponding epoch when it has ended (L157).

To claim rewards, users need to claim for all `tokenIds` that they currently own, which is done via a pro-rata approach on a `tokenId`'s weight to the user's overall weight on the start of the corresponding epoch (L198).

Therefore, it is mandatory for a user to claim for all `tokenIds` that he currently owns to achieve an aggregated weight of  $1e18$ , such that the whole balance will be claimed (simplified) (L204).

This logic will result in an issue — assume the case where a user has `tokenId` 1, 2 and 3 allocated to the epoch starting at 604800 with weights of 5,5,90 and corresponding a full weight of 100 at this `block.timestamp`. A lot of time has passed and the user has not claimed any rewards yet, then the user realizes that `tokenId` 3 is unlocked and claims his underlying LYNEX tokens and burns the token (VotingEscrowV2.sol::L381, VotingEscrowV2.sol::L504).


The problem is that after the `tokenId` has been burned, the user is no longer the owner of the `tokenId` and hence if the user now wants to claim for epoch 1, he can only claim `Ids` 1 and 2, because all `getReward` functions only allow claiming for currently owned `tokenIds` (L271).

The weights however, are calculated based on the `block.timestamp` 604800 where the user had a total VP of 100. He effectively lost 90% of his rewards.

## Recommendation

Consider not allowing tokens to be burned before these have been used to claim rewards.

## Resolution

 ACKNOWLEDGED

The side effect was added to the documentation of the function.

## Severity

 MEDIUM SEVERITY

## Description

The owner of the tokenId can claim tokens via the getReward function (L272).

These rewards are then calculated based on the to-be-claimed epoch and the delegatee's allocation in this epoch. In the best scenario, the owner will receive 100% of the tokenIds VP as allocation — this however depends on the delegatee.


This means that each tokenId has a very specific inherent value determined by how much rewards were allocated towards it within all epochs. The owner of this NFT can now list it for sale on various different platforms, which includes the unclaimed rewards.

The current owner now detects a buy transaction from the buyer and frontruns this by calling getReward. He now successfully claims all rewards which have been allocated to that tokenId. The new buyer expected this NFT to still have unclaimed rewards and therefore paid more, however, he ends up without these rewards.



## Recommendation



Consider not allowing transfers if the NFT has still unclaimed rewards. This will require refactoring of certain functionalities and a careful second audit round of these functionalities.



## Resolution

 ACKNOWLEDGED

The side effect was added to the documentation of the function.

Issue #36 Delegation logic can result in no rewards	
Severity	 LOW SEVERITY
Description	<p>Within the getRewards logic, owners of tokenId can claim rewards by using their tokenId. The clue here is that the tokenId might have been delegated to another address. We need to put into context that the _earnedTokenId function calculates the rewards based on the delegatee's VP but claims the rewards to the tokenId owner.</p> <p>If now for example Alice delegates her tokenId to Bob but Bob did not vote at all for a designated epoch, Bob's balance will be zero and Alice will not receive any rewards.</p> <p>This is likely a design choice, however, this inconsistency between the delegatee and tokenId owner may result in issues.</p>
Recommendation	Consider if that is desired, if not, consider just claiming strictly towards the delegatee.
Resolution	 ACKNOWLEDGED The side effect was added to the documentation of the function.

Issue #37 balanceOf and balanceOfOwner are incorrect	
Severity	 LOW SEVERITY
Description	Both functions return the balance (used VP for a specific gauge) for the next epoch (getNextEpochStart). This is incorrect and will always be zero as users can only deposit for the current epoch and never for the next epoch.
Recommendation	Consider using the current active epoch as time denomination.
Resolution	 RESOLVED

<b>Issue #38</b>	<b>Inconsistency in rewardPerToken function</b>
<b>Severity</b>	 INFORMATIONAL
<b>Description</b>	rewardPerToken returns rewardsPerEpoch if the supply is zero. This is correct but inconsistent because it is not multiplied with 1e32. This is just an informational issue because no rewards can be claimed for this epoch if there is no supply.
<b>Recommendation</b>	Consider multiplying by 1e32.
<b>Resolution</b>	 RESOLVED



---

## 2.8 Gauge/OptionTokenV3

OptionTokenV3 is a simple option token which is meant to be compatible with the Lynex token. Users may receive this token as rewards for staking within gauges and then will have several options to receive the underlying Lynex token.

At a high-level overview, users will need to provide an amount of the paymentToken, which is likely USDC, to then receive the underlying Lynex token. There are three possible scenarios to gather the underlying Lynex:

- 1) `exercise`: Allows users to burn 0Lynx, provide a specific amount of paymentToken and receive Lynex.
- 2) `exerciseVe`: Allows users to burn 0Lynx, provide a specific amount of paymentToken and receive a locked NFT. `_discount` can be chosen and will influence the lock duration.
- 3) `exerciseLp`: Allows users to burn 0Lynx, provide a specific amount of paymentToken and create a LP pair which is deposited with a lock into the corresponding gauge. `_discount` can be chosen and will influence the lock duration.

For the paymentAmount calculation, the discount is used which is then applied on the exchange rate provided by Algebra's TWAP oracle.





## 2.8.1 Privileged Functions

- `setTwapOracleAndPaymentToken`
- `setFeeDistributor`
- `setDiscount`
- `setVeDiscount`
- `setTwapSeconds`
- `setPairAndPaymentToken`
- `setGauge`
- `setRouter`
- `setMinLPDiscount`
- `setMaxLPDiscount`
- `setLockDurationForMaxLpDiscount`
- `setLockDurationForMinVeDiscount`
- `setLockDurationForMinLpDiscount`
- `burn`
- `unPause`
- `togglePermissionedMint`
- `toggleOption`
- `pause`



## 2.8.2 Issues & Recommendations

<b>Issue #39</b>	<b>Governance Privilege: Full control over contract functionality</b>
<b>Severity</b>	 GOVERNANCE
<b>Description</b>	<p>The governance of this contract has several privileges for invoking certain functions that can drastically alter the contracts behavior.</p> <p>For instance, the feeDistributor address can be changed to a non-compatible contract which then effectively locks all Lynex tokens or paymentToken can be set to Lynex which then allows a malicious feeDistributor to consume all Lynex tokens within the contract.</p>
<b>Recommendation</b>	Consider incorporating a Gnosis multi-signature contract as owner and ensuring that the Gnosis participants are trusted entities.
<b>Resolution</b>	 ACKNOWLEDGED
	The owner will be a multi-signature contract.

**Description**

getLockDurationForVeDiscount is responsible for returning the lockDuration for a corresponding \_discount. The larger the \_discount variable, the more the user will have to provide in paymentToken. For instance, if \_discount is 80, the user will have to provide 80%. On the other hand, if \_discount is 20, the user will only have to provide 20%.

Therefore it can be stated: the lower the \_discount, the less the user will pay.

The functionality therefore aims to return a large duration for a low \_discount and vice-versa. This means users are incentivized to pay more for a shorter lock. The duration is then calculated as follows:  $\text{slope} * \text{\_discount} + \text{intercept}$

With an increased \_discount, the result is decreased, which perfectly satisfies the condition of a lower lock duration for a larger \_discount.

The idea behind this logic is to just set an intercept (start point) and then decrease the intercept based on the slope multiplied with \_discount — the higher \_discount, the higher the intercept decrease. This mechanism works perfectly fine as long as users the provided discount is not too large. For example:

```
_discount = 70  
-1 546 560 * 70 + 63 072 000  
-> abs(45 187 200)
```

```
_discount = 20  
-1546560 * 20 + 63072000  
-> abs(32 140 800)
```

As above, the duration is shorter with a lower \_discount.

The problem therefore is the fact that discount is not hard-coded as 40 but can be changed by the owner and since discount is also used for the standard exercise function there might be valid scenarios where discount will be set to for example 70.

In such a scenario, this algorithm will not work anymore because  $\text{discount} * \text{slope}$  will become extremely large (negative), resulting in a large duration.

Also notable for this issue, discount can be set up to 100 (L411).

**Recommendation** Consider using a separate variable as `veMinDiscount` cannot be set too high and only serves the VE calculation purpose.

**Resolution**  RESOLVED

#### Issue #41 `setLockDurationForMinVeDiscount` can only be used to reduce `lockDurationForMinVeDiscount`

**Severity**  LOW SEVERITY



**Location** L498-499  





```
if (_duration > lockDurationForMinVeDiscount) revert  
OptionToken_InvalidLockDuration();  
lockDurationForMinVeDiscount = _duration;
```



**Description** Due to an erroneous check, only `lockDurationForMinVeDiscount` can be reduced. The new `_duration` should be checked using the `FULL_LOCK` value.



**Recommendation** Consider fixing the check to allow `lockDurationForMinVeDiscount` to be set as desired.



**Resolution**  RESOLVED

Issue #42	The contract may use an outdated settings if they are not updated accordingly
<b>Severity</b>	 LOW SEVERITY
<b>Location</b>	<p><u>L388-397</u></p> <pre>function setTwapOracleAndPaymentToken(IDynamicTwapOracle _twapOracle, address _paymentToken) external onlyAdmin {</pre> <p><u>L434-444</u></p> <pre>function setPairAndPaymentToken(IPair _pair, address _paymentToken) external onlyAdmin {     (address token0, address token1) = _pair.tokens();</pre>
<b>Description</b>	<p>Both functions update paymentToken, however setTwapOracleAndPaymentToken verifies TWAP's token using the paymentToken, and within the setPairAndPaymentToken function, the new payment token is not checked to be the same as the TWAP.</p>
<b>Recommendation</b>	<p>Consider refactoring the function to avoid misconfiguration, for example by using a single function to set all those variables as they depend on each other.</p>
<b>Resolution</b>	<p> RESOLVED</p> <p>The function was removed and was refactored into the setPaymentConfiguration function.</p>

Issue #43      Insufficient slippage parameters for exerciseLp	
Severity	 LOW SEVERITY
Description	The exerciseLp function allows a user to create a LP position with WETH/Lynex. This function exposes a _maxLPAmount parameter which ensures that the LP creation will not take more WETH than desired. It however lacks a minimum parameter such that a position may result in way less WETH than anticipated.
Recommendation	Consider incorporating a _minPaymentAmount parameter.
Resolution	 RESOLVED Approvals are not reset before setting the new approvals which might lead to unwanted dangling approvals.
Issue #44      veMaxDiscount can be set above discount	
Severity	 LOW SEVERITY
Description	setVeDiscount can be set up to 100, potentially exceeding discount. This will falsify the result of getSlopeInterceptForVeDiscount.
Recommendation	Consider executing the same safeguard checks as here:  <a href="https://github.com/Lynexfi/lynex-contracts/blob/d2eac416d902970ad6edf09375ad637c13c7a2bf/contracts/OptionToken/OptionTokenV3.sol#L471">https://github.com/Lynexfi/lynex-contracts/blob/d2eac416d902970ad6edf09375ad637c13c7a2bf/contracts/OptionToken/OptionTokenV3.sol#L471</a>  <a href="https://github.com/Lynexfi/lynex-contracts/blob/d2eac416d902970ad6edf09375ad637c13c7a2bf/contracts/OptionToken/OptionTokenV3.sol#L481">https://github.com/Lynexfi/lynex-contracts/blob/d2eac416d902970ad6edf09375ad637c13c7a2bf/contracts/OptionToken/OptionTokenV3.sol#L481</a>  Additionally, it must be ensured that discount != veMaxDiscount and minLpDiscount != maxLpDiscount, otherwise these calculations would revert due to a division by zero error.
Resolution	 RESOLVED

Issue #45 Lack of safeguard for setTwapSeconds	
Severity	 LOW SEVERITY
Description	The function lacks a minimum safeguard. A very low value could make the TWAP oracle more prone to manipulation.
Recommendation	Consider setting a minimum limit for twapSeconds.
Resolution	 RESOLVED

Issue #46 Lack of slippage parameters for exerciseLp	
Severity	 LOW SEVERITY
Description	The exerciseLp function allows users to create a LP token for Lynex/USDC. Due to the lack of amountAMin and amountBMin parameters, users can end up with an undesired ratio of tokens.
Recommendation	Consider incorporating a minAmountB parameter.
Resolution	 RESOLVED

Issue #47 exerciseExternal seems unfinalized	
Severity	 INFORMATIONAL
Description	The exerciseExternal function allows a user to invoke the exercise function on another options token. The issue is that the msg.sender of the exercise call will be the 0Lynx token and it is expected that tokens are burned from msg.sender and the paymentAmount is transferred in by the msg.sender. This will revert on multiple occasions.
Recommendation	Consider simply removing this function.
Resolution	 RESOLVED


**Issue #48****TWAP oracle weakness can result in paymentAmount being exploited****Severity** INFORMATIONAL**Description**

TWAP oracles can be vulnerable to price manipulation especially in illiquid markets, as malicious actors can influence the price by executing strategic trades over the averaging period. They also introduce a lag between the current market price and the reported price, which can be problematic in volatile market conditions. The accuracy of TWAP oracles relies on sufficient liquidity, and they may not provide real-time price data, which can be an issue for applications requiring up-to-date information.

In this specific scenario, it can be possible to pay less USDC as expected. The protocol however would not experience a loss, just less fees.

**Recommendation**

This seems to be an acceptable risk considering that there is no reliable Lynex oracle out yet.

**Resolution** ACKNOWLEDGED

The side effect was added to the documentation of the function.





---



## 2.9 Gauge/RewardsDistributorV2



RewardsDistributorV2 is a rewarder contract which allows VE token owners to receive Lynex rewards for their corresponding position. There are two reward distribution scenarios:

- 1) tokenId is unlocked: Rewards are transferred directly to the tokenId owner
- 2) tokenId is not unlocked: Rewards are used to increase amount size of tokenId



The reward calculation algorithm ensures that each epoch gets its correct portion of the total distributed rewards — it can handle single epoch updates, multi-epoch updates and partial epoch updates. It is also ensured that users can only claim rewards if the desired epoch has been fully rewarded. Users will receive a pro-rata share of the total distributed rewards on the corresponding epoch based on their tokenId VP and the total VP at the start of the epoch.



## 2.8.1 Issues & Recommendations



<b>Issue #49</b>	<b>Loop break within <code>_claim</code> can result in side-effects</b>
<b>Severity</b>	 HIGH SEVERITY
<b>Description</b>	<p>The loop within <code>_claim</code> determines how much a user receives as rewards based on the <code>tokenId</code>'s VP and the total VP at the time of the epoch start.</p> <p>The loop breaks early if the balance of the <code>tokenId</code> is zero. This is incorrect as it should continue instead of break. If for example a <code>tokenId</code>'s VP becomes zero at one point and later becomes non-zero again, it will not be possible to reach the later point because the loop will always break without increasing the <code>week_cursor</code>.</p> <p>Since the voting power algorithm is out of scope, we are not able to determine with 100% certainty such a scenario. However, if for example a <code>tokenId</code> is claimed after it has been unlocked, <code>balanceOfNFTAt</code> will return zero. If this <code>tokenId</code> is reactivated, it will have a valid VP again. This means that there will be a time period where this <code>tokenId</code> will not have a balance and therefore the loop will break. Therefore, that <code>tokenId</code>'s <code>week_cursor</code> will always be stuck at this point without any ability to claim future rewards.</p>
<b>Recommendation</b>	Consider using <code>continue</code> instead of <code>break</code> .
<b>Resolution</b>	 RESOLVED



Issue #50	Burning of tokenId will result in permanently stuck rewards
Severity	 HIGH SEVERITY
Description	<p>The <code>claim</code> and <code>claimMany</code> functions allow rewards to be claimed for a specific <code>tokenId</code>. These rewards are then transferred to the owner if the <code>tokenId</code> is unlocked. A problem will arise if users burn their expired <code>tokenId</code> before they have claimed their rewards, as this will reset <code>_owners[tokenId]</code>: <a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/e3b20c89aacff1cf150a596a9dbb0ff6478f2820/contracts/token/ERC721/ERC721Upgradeable.sol#L333">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/e3b20c89aacff1cf150a596a9dbb0ff6478f2820/contracts/token/ERC721/ERC721Upgradeable.sol#L333</a></p> <p>This means the <code>claim</code> function will revert due to the <code>ownerOf</code> call reverting: <a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/e3b20c89aacff1cf150a596a9dbb0ff6478f2820/contracts/token/ERC721/ERC721Upgradeable.sol#L79">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/e3b20c89aacff1cf150a596a9dbb0ff6478f2820/contracts/token/ERC721/ERC721Upgradeable.sol#L79</a></p>
Recommendation	Consider interconnecting the <code>claim</code> within the <code>VotingEscrowV2</code> and the <code>RewardsDistributorV2</code> contracts.
Resolution	<div data-bbox="451 1050 644 1084">  ACKNOWLEDGED         </div> <p>The side effect was added to the documentation of the function.</p>



Issue #51      Accrued rewards can serve as bait if tokenId is unlocked	
Severity	 MEDIUM SEVERITY
Description	<p>Users are incentivized for holding VE tokens. They can claim rewards via the <code>claim</code> and <code>claim_many</code> functions, using the corresponding <code>tokenIds</code>.</p> <p>These accrued rewards can serve as bait where users can list the <code>tokenId</code> with accrued rewards but frontrun the purchase transaction with claiming these rewards. The buyer would effectively lose out on these desired rewards.</p>
Recommendation	A fix is non-trivial and would include modifying transfer function to automatically claim rewards upon transfer.
Resolution	 ACKNOWLEDGED The side effect was added to the documentation of the function.

Issue #52      Off-by-one error will temporarily prevent claiming	
Severity	 LOW SEVERITY
Description	<p>There are two claim scenarios. As soon as <code>block.timestamp</code> is larger than the <code>endTime</code> of a position, tokens are transferred to the owner. Therefore in reverse, if <code>block.timestamp = endTime</code>, <code>increaseAmount</code> is invoked (L193).</p> <p>This will not work because <code>increaseAmount</code> will revert if <code>block.timestamp = endTime</code> (<code>VotingEscrowV2Upgradeable.sol::L352</code>).</p> <p>This is a simple off-by-one issue.</p>
Recommendation	Consider using the same time scenarios for both contracts.
Resolution	 RESOLVED

Issue #53 Unreachable code-section within _checkpoint_token	
Severity	 INFORMATIONAL
Description	<p>The _checkpoint_token function is invoked whenever a period is updated via update_period within the Minter contract. It contains the following if-clause:</p> <pre>if (last_token_time == 0) {     last_token_time = timestamp();     start_time = last_token_time; }</pre> <p>This condition is however completely redundant as last_token_time will inherently be set within the constructor and is therefore never zero:</p> <pre>constructor(address _voting_escrow) {     uint _t = (block.timestamp / WEEK) * WEEK;     start_time = _t;     last_token_time = _t;     time_cursor = _t;     [...] }</pre>
Recommendation	Consider removing the unreachable code.
Resolution	 RESOLVED

Issue #54 Gas optimizations	
Severity	 INFORMATIONAL
Description	WEEK can be made constant, and voting_escrow and token can be made immutable to save some gas
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	 RESOLVED

---

## 2.10 Gauge/MinterUpgradeableV2

MinterUpgradeableV2 is the core contract for the LYNEX minting logic. It handles the correct emission calculation and distributes LYNEX to the following recipients:

- Team
- RewardsDistributorV2
- Gauges

The calculation of the Lynex emission is split in two scenarios:

- Very first emission calculation: This is the first period update after contract deployment, it will mint a total of 5 000 000 Lynex tokens which is distributed as follows:
  - 52% towards the RewardsDistributorV2
  - 4% to the team
  - 44% to gauges
- Subsequent emissions (early state) : These are all emissions after the very first emission, the circulating supply is still considerably low at this point. This is handled via the following algorithm:
  - Weekly emission is calculated using 99% of the previous weekly emission
  - Rebase rate is steadily decreased by 1% each epoch and is calculated as the percentage of the locked supply. It can never exceed 52%, respectively 51%, 50% (depending on the epoch)
  - Distribution is then calculated based on the overall weekly emission and the rebase rate, with 4% to the team and the leftover to all gauges
- Subsequent emissions (progressed): These are all emissions after the very first emission and after some progress. The main difference to b) is that the circulating supply becomes very large which then triggers the weekly emission

to be based on the circulating supply. This is handled via the following algorithm:



- Weekly emission is calculated as 0.2% the circulating supply
- Rebase rate is steadily decreased by 1% each epoch and is calculated as the percentage of the locked supply. It can never exceed 52%, respectively 51%, 50% (depending on the epoch)
- Distribution is then calculated based on the overall weekly emission and the rebase rate, with 4% to the team and the leftover to all gauges

## 2.9.1 Privileged Functions


- `setTeam`
- `acceptTeam`
- `setGovernor`
- `acceptGovernor`
- `setVoter`
- `setTeamRate`
- `setEmission`
- `setRebase`
- `setRebaseChange`
- `setRewardDistributor`



## 2.9.2 Issues & Recommendations

<b>Issue #55</b>	<b>Lack of epoch update may result in skipped epoch</b>
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	<p>The <code>update_period</code> function is the most fundamental function within the whole architecture. It handles the whole epoch update including:</p> <ul style="list-style-type: none"><li>- update of period</li><li>- emission calculation</li><li>- emission minting</li><li>- emission distribution to team/gauges/rewarder</li></ul> <p>The problem hereby is the fact that the period is updated as follows: <code>_period = (block.timestamp / WEEK) * WEEK;</code></p> <p>Which means it is based on the current <code>block.timestamp</code>. If 7 days have passed without an update, the next <code>update_period</code> call will inherently skip the un-updated epoch due to the usage of <code>block.timestamp</code>.</p>
<b>Recommendation</b>	Consider if this is an acceptable downside from the design choice or consider refactoring this logic.
<b>Resolution</b>	 ACKNOWLEDGED
	The side effect was added to the documentation of the function.



**Issue #56****\_initialize can result in undistributed epoch rewards****Severity** LOW SEVERITY**Description**


When the initialize function is invoked, active\_period is set to two weeks in the future. For example, if the contract is initialized at TS 604 800, active\_period will become TS 1 814 400. Therefore, the first valid distribution would happen at TS 2 419 200.

If \_initialize is then invoked at TS 2 419 200, active\_period will be set to 2 419 200 and therefore distributions can only happen after 3 024 000.

This means that essentially 1 period of rewards will not be distributed amongst users.

**Recommendation**

Consider invoking \_initialize at a reasonable timestamp after the original initialize function.


**Resolution** ACKNOWLEDGED**Issue #57****Emission increase can happen retroactively****Severity** LOW SEVERITY**Description**

setEmission allows privileged addresses to change the EMISSION parameter up to 1050, while it is 990 by default.


This change can also have retroactive effects on epochs if the current epoch is not updated yet.

**Recommendation**

Consider updating the epoch before the EMISSION parameter is changed.

**Resolution** ACKNOWLEDGED

## Severity

 LOW SEVERITY

## Description

The Lynex token is distributed to three different recipients:

- a) RewardsDistributorV2
- b) Gauges
- c) Team

The shares for each recipient are based on different parameters, which can, if incorrectly set, be above 100%.

For example, REBASEMAX can be set up to 1000, and if all tokens are locked, this will result in a rebase amount of 100% from the total minted amount plus the team share on top, which would then underflow.

## Recommendation

Consider ensuring that the sum of all shares is not above 100%.

## Resolution

 RESOLVED

---

## 2.11 Gauges/CLFeesVault

CLFeesVault is a simple distributor contract that handles the calculation and distribution of the corresponding pool tokens to:



- a) Gamma
- b) Referral
- c) Gauge / BribeV2



Disclaimer: The whole Hypervisor and Algebra logic is out of scope. It is assumed that the fee distribution in form of token0/token1 to the CLFeesVault contract is handled correctly. This is likely handled within the HyperVisor contract.

### 2.11.1 Privileged Functions


- setReferralShare
- setGammaShare
- setGammaRecipient
- setDibs
- setPairFactory
- setVoter
- setPermissionRegistry
- setPool
- emergencyRecoverERC20


## 2.11.2 Issues & Recommendations

<b>Issue #59</b>	<b>Potential underflow in share calculation if MAX_REFERRAL_FEE is in a different denomination</b>
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	<p>_getFee calculates the gamma, referral and gauge share on the amount. The referral fee is optionally fetched from the pairFactoryClassic contract. In a scenario where the return value of MAX_REFERRAL_FEE() is &gt; 10_000, the whole function would revert due to an underflow.</p>
<b>Recommendation</b>	Consider implementing a safeguard that downgrades this value in such a scenario.
<b>Resolution</b>	 RESOLVED

<b>Issue #60</b>	<b>Incorrect configuration can left partial fees unclaimed</b>
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	<p>Within the <code>claimFee</code> function, shares are calculated for</p> <ul style="list-style-type: none"><li>a) gauge</li><li>b) referrer</li><li>c) gamma</li></ul> <p>Afterwards, these fees are sent to the corresponding addresses. A problem arises if the contract exposes a valid referral share but no dibs address (within the constructor, this address is not necessarily set).</p> <p>In that scenario, the referral fee is simply not distributed until the referral share is set to zero or the dibs address is set to a valid address.</p>
<b>Recommendation</b>	Consider setting the configuration with utmost care and implement a constant surveillance mechanism.
<b>Resolution</b>	 RESOLVED



Issue #61      Several events are not emitted	
Severity	 INFORMATIONAL
Description	<p>Several functions that update the state variables do not emit an event:</p> <ul style="list-style-type: none"> <li>- setPool</li> <li>- setPermissionRegistry</li> <li>- setVoter</li> <li>- setDibs</li> <li>- setGammaRecipient</li> <li>- setGammaShare</li> <li>- setReferralShare</li> <li>- emergencyRecoverERC20</li> </ul>
Recommendation	Consider adding events to the aforementioned functions.
Resolution	

Issue #62      Unused imports	
Severity	 INFORMATIONAL
Description	<p>There are unused imports in the contract. Consider removing them.</p> <ul style="list-style-type: none"> <li>- SafeMath</li> <li>- ReentrancyGuard</li> <li>- IBribe</li> </ul>
Recommendation	Consider removing the unused imports.
Resolution	

---

## 2.12 TWAP/AlgebraV1Twap

AlgebraV1Twap is a simple TWAP contract which leverages Algebra's `DataStorageLibrary` to fetch an output amount for an input amount and input token for a specific pool.

The corresponding pool is set upon deployment and cannot be changed.

### 2.12.1 Issues & Recommendations

No issues found.





## 2.13 TWAP/DataStorageLibrary

DataStorageLibrary is a simple helper contract for fetching the time-weighted-average tick of a specific AlgebraPool and the quote (amount of quoteToken received for amount of baseToken) at a specific tick.

It is important to note that all other contracts involved in the consult function, namely AlgebraPool, DataStorageOperator and DataStorage are excluded from the scope.

It is therefore assumed that the return value of tickCumulatives is adjusted accordingly if the period does not exactly match a written slot.

### 2.13.1 Issues & Recommendations

<b>Issue #63</b>	<b>TWAP only works for 136 years</b>
<b>Severity</b>	 INFORMATIONAL
<b>Description</b>	<p>The period parameter determines the time in the past from which the TWAP should be determined. For instance, if period is 86400, the TWAP price for the last 24 hours is determined.</p> <p>Due to the fact that period is a uint32, this will only work for 4294967295 seconds.</p>
<b>Recommendation</b>	In our opinion this is an acceptable time-period, however, if deemed insufficient, it can be raised to a larger uint type.
<b>Resolution</b>	 ACKNOWLEDGED





**PALADIN**  
BLOCKCHAIN SECURITY