PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Supernova

14 Feb 2026

paladinsec.co   info@paladinsec.co

Table of Contents

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 advocation 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 Supernova 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

Project Name	Supernova
URL	https://supernova.xyz
Platform	Ethereum
Language	Solidity
Preliminary	BHSmartContracts: https://github.com/BlackHoleDEX/SNContracts/pull/1 (https://github.com/BlackHoleDEX/SNContracts/compare/d5dabbc4974505e4f8462cae6a88d36820a9de6b...f3b5db7a1f08a303776b769e1d5539207e6c4511) Algebras Changes: https://github.com/BlackHoleDEX/Algebra/pull/10 (the pull request portion until bc84985f65b5f74335987029e41779a0b97bd497)
Resolution 1	https://github.com/BlackHoleDEX/SNContracts/compare/f3b5db7a1f08a303776b769e1d5539207e6c4511...7b8d79f34259275f98ff4f8ba00d971090d76e96 Algebras Changes: https://github.com/BlackHoleDEX/Algebra/pull/10 (the pull request portion from bc84985f65b5f74335987029e41779a0b97bd497 until 9050b5fa3f62e4a6d67c73c6247b578d5a230499)
Resolution 2	https://github.com/BlackHoleDEX/SNContracts/compare/7b8d79f34259275f98ff4f8ba00d971090d76e96...e06bef8b96e4b888c4e0b6cbecdef7a9c28da64b
Resolution 3	https://github.com/BlackHoleDEX/SNContracts/compare/e06bef8b96e4b888c4e0b6cbecdef7a9c28da64b...16eb66816844cee0d395514427f5d8b93a2e8027
Resolution 4	

	https://github.com/BlackHoleDEX/SNContracts/compare/16eb6681-6844cee0d395514427f5d8b93a2e8027...e7b493a7b2c630feba667-e010add2dce89660650
Resolution 5	https://github.com/BlackHoleDEX/SNContracts/compare/e7b493a7-b2c630feba667e010add2dce89660650...ec90e21201ed4c5c40929-41f1d5c69b6bf065626
Resolution 6	https://github.com/BlackHoleDEX/SNContracts/compare/ec90e212-01ed4c5c4092941f1d5c69b6bf065626...a44e5f2a0278dd15f2b23e-6328a4fc32a0f7d221
Resolution 7	https://github.com/BlackHoleDEX/SNContracts/compare/a44e5f2a0-278dd15f2b23e6328a4fc32a0f7d221...f8ff2b6a3e65caf30441dbe9-9c616e38d007b444
Live Match Notes	Some minor differences were spotted in the following deployed contracts. Paladin has checked them and found no issues with the changes. `NonFungiblePositionManager.sol`: Name changed from Algebra to Supernova `PairFactory.sol`: Default fee of Basic Volatility Pools changed from 0.6% to 0.5% `VoterV3.sol`: Max voting limit has changed from 20 to 30.

1.2 Contracts Assessed

Name	Contract	Live Code Match
SuperNova	0x00da8466b296e382e5da2bf20962d0cb87200c78	✓ MATCH
MinterUpgradeable	0xfe29ea1348f0990273db5e19ad521e45acda84a2	✓ MATCH
RewardsDistributor	0xb3410a30af5033af822b8ea5ad3bd0a19490ea97	✓ MATCH
PairFactory	0x5aef44edfc5a7edd30826c724ea12d7be15bdc30	✓ MATCH
Pair	0xe3B07bc14A3c96E55f474492F1c1C3324cB9CcFe	✓ MATCH
PairFees	0x35b842d371fb9fAAEe00AD751016181Ae7eC59A1	✓ MATCH
PairGenerator	0x42a7a5baafb1818da3a39ce1b97a58799d69bbb8	✓ MATCH
PairBootstrapper	0x7f8f2b6d0b0aae8e95221ce90b5c26b128c1cb66	✓ MATCH
RouterV2	0xf0756789a6fb10ce566a24cbf1b6570753d97ec9	✓ MATCH
RouterHelper	0xd8377aea61c4c4d43bf0588956f4e861720803c6	✓ MATCH
VotingEscrow	0x4c3e7640b3e3a39a2e5d030a0c1412d80fee1d44	✓ MATCH
VotingBalanceLogic	0xed686a5b0bf0df5c97f8eabd1b776ae399319847	✓ MATCH
VotingDelegationLi-b		PENDING
VoterV3	0x1c7bf2532dfa34eeea02c3759e0ca8d87b1d8171	✓ MATCH
GaugeManager	0x19a410046afc4203aece5fbfc7a6ac1a4f517ae2	✓ MATCH
GaugeFactory	0x66647a19452e98e98a9f479883f241e33016adb0	✓ MATCH
GaugeV2	0x094BEf1766Eec5Db769be1B31246b60787359052#code	✓ MATCH
GaugeFactoryCL	0x8d38206e38ec86b14530186aa36cc3b1ed8cd674	✓ MATCH
GaugeCL	0x40f348C884a872efc6144Db381A83BE8cF250935	✓ MATCH
BribeFactoryV3	0xeb37f11c573ab01358d5fefb10f5de2b4237344c	✓ MATCH
Bribe	0x016AC7265C967581227aa6FAc5cF6489D05FC144	✓ MATCH
CustomPoolDeployer	0x2493b36759fb77e40ef863ca59807a9d7689af4a	✓ MATCH
PermissionsRegistr-y	0x344eec31c725187cd026db73ed8805e72967c28d	✓ MATCH
TokenHandler	0xa1154fe44a3d5c740644b9028e4d68fd876de201	✓ MATCH
BlackTimeLibrary		✓ DEPENDENCY
BlackholePairAPIV2	0x1fd265236e240f4f4487ae91de589ec88f7535aa	✓ MATCH
veNFTAPI	0x85dc70913e49e5ebd888ada03034e3be109e5881	✓ MATCH
Math		✓ DEPENDENCY
AlgebraVaultFactor-y	0xafc0497f052A3b5274659308D0b875271C03038d	✓ MATCH
		PENDING

CustomPluginV1Factory and CustomPluginV2Factory		
AlgebraBasePluginV3	0xa665fc4b0C307652Cc7a546FfDC77Ef06fb30660	✓ MATCH
BasePluginV3Factory	0xdbfd67d12cadb8925c1417ff3638693f2bf99b97	✓ MATCH
SecurityPlugin	0xa665fc4b0C307652Cc7a546FfDC77Ef06fb30660	✓ MATCH
SecurityRegistry	0x454e62e725ad5a47931043f7e6369cfbb879bdfd	✓ MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● Governance	3	-	-	3
● High	3	3	-	-
● Medium	3	3	-	-
● Low	24	16	1	7
● Informational	35	13	7	15
Total	**68**	**35**	**8**	**25**

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 Global

ID	Severity	Summary	Status
01	GOV	The voting and native token mechanisms are configurable, allowing the SuperNova team to control the emissions and minting system	ACKNOWLEDGED
02	INFO	Lack of error messages for require statements	ACKNOWLEDGED

1.3.2 SuperNova

ID	Severity	Summary	Status
03	INFO	Typographical issues	✓ RESOLVED

1.3.3 MinterUpgradeable

ID	Severity	Summary	Status
04	GOV	Minting logic is upgradeable, allowing the proxy admin to redefine the minting logic freely and potentially mint a large unexpected amount of supply	ACKNOWLEDGED
05	LOW	The function which calculates how much of the mint should go to the veNOVA rebase is an approximation and might slightly misbehave in edge cases	ACKNOWLEDGED
06	INFO	The _initialize function lacks validation on the distribution amounts	ACKNOWLEDGED
07	INFO	circulating_supply can be manipulated	ACKNOWLEDGED

1.3.4 RewardsDistributor

No issues found.

1.3.5 PairFactory

ID	Severity	Summary	Status
08	LOW	A referral cap was added but setCustomReferralFee remains uncapped, potentially bricking swaps when set to an excessive value	✓ RESOLVED
09	LOW	Swap fees cannot be set to zero	ACKNOWLEDGED
10	INFO	Typographical issues and gas optimizations	PARTIAL

1.3.6 Pair

ID	Severity	Summary	Status
11	GOV	swaps can be blocked by the SuperNova governance	ACKNOWLEDGED
12	LOW	Stable pair MINIMUM_LIQUIDITY requirement is lower than Uniswap V2's under very edge-case circumstances	ACKNOWLEDGED
13	LOW	Fee precision mechanism may cause the pair to block swaps for tokens with an extremely large supply and large amount of generated fees	✓ RESOLVED
14	LOW	_getAmountOut and all functions relying on it are sometimes slightly inaccurate for stable pairs, causing routers to receive slightly fewer tokens	✓ RESOLVED
15	LOW	Several functions do not have a reentrancy lock	✓ RESOLVED
16	LOW	current() should not be used as an actual TWAP function	ACKNOWLEDGED
17	INFO	Typographical issues	✓ RESOLVED

1.3.7 PairFees

ID	Severity	Summary	Status
18	MEDIUM	Fee mechanism will malfunction for fee-on-transfer tokens, causing there to be insufficient fees for everyone	✓ RESOLVED

1.3.8 PairGenerator

ID	Severity	Summary	Status
19	INFO	Typographical issues	PARTIAL

1.3.9 PairBootstrapper

ID	Severity	Summary	Status
20	INFO	Lack of deadline for V2 liquidity addition	✓ RESOLVED
21	INFO	Typographical issues	✓ RESOLVED

1.3.10 RouterV2

ID	Severity	Summary	Status
22	LOW	addLiquidity frontrunning protection can be bypassed	✓ RESOLVED
23	LOW	swapPossible check for fee-on-transfer swaps will succeed prematurely	✓ RESOLVED
24	LOW	The add liquidity functions are inefficient for fee on transfer tokens and do not properly enforce the minimum amount out for them	✓ RESOLVED
25	LOW	The swap routes provided are not properly enforced to actually contain sensible data	PARTIAL
26	LOW	The router is not very robust when malicious reentrancy hooks are present on tokens within the swap route	✓ RESOLVED
27	INFO	The requested minimum amount out might not be correctly enforced for special tokens	ACKNOWLEDGED

1.3.11 RouterHelper

ID	Severity	Summary	Status
28	MEDIUM	Swap prices for stable pairs will mostly return either "0" or "1" due to a normalization error	✓ RESOLVED
29	LOW	getAmountsOut does not handle failures gracefully	✓ RESOLVED
30	INFO	_calculateStableSwapPrice is slightly vulnerable to overflow reverts	✓ RESOLVED
31	INFO	_swapRatio fixes have small edge cases which can cause reverts and small side-effects in very specific situations	✓ RESOLVED
32	INFO	getAmountOut will still return a value even if the factory paused V2 swaps	✓ RESOLVED
33	INFO	Typographical issues and gas optimizations	PARTIAL

1.3.12 VotingEscrow

ID	Severity	Summary	Status
34	HIGH	NFTs are incorrectly self-delegated during NFT transfers	✓ RESOLVED
35	LOW	Split can now leave one of the two NFTs with a zero value, an outcome which was previously impossible	✓ RESOLVED
36	LOW	ownerOf does not revert for invalid tokenId inputs	ACKNOWLEDGED
37	LOW	Unsafe casts occur throughout the contract which reduces code-safety, especially if SUPERNOVAs supply ever increases significantly	ACKNOWLEDGED
38	INFO	Several functions lack reentrancy guards	ACKNOWLEDGED
39	INFO	getsmNFTPastVotes seems to calculate the NFT balance twice	✓ RESOLVED
40	INFO	Typographical issues	✓ RESOLVED

1.3.13 VotingBalanceLogic

ID	Severity	Summary	Status
41	LOW	Block-based historical balance and total supply functions may not be consistent until a snapshot occurs after the provided block	✓ RESOLVED
42	INFO	Typographical issues	PARTIAL

1.3.14 VotingDelegationLib

ID	Severity	Summary	Status
43	HIGH	Anyone can call the internal moveTokenDelegates and moveAllDelegates functions, allowing for exploiters to delegate arbitrarily to their own wallets and fully breaking the delegation logic	✓ RESOLVED
44	HIGH	The delegation logic is fundamentally broken in multiple ways, which can be abused to DoS VotingEscrow mints and transfers, prevent delegations to any wallet and clear a wallet's delegates at will	✓ RESOLVED
45	INFO	Typographical issues	✓ RESOLVED

1.3.15 VoterV3

ID	Severity	Summary	Status
46	INFO	Typographical issues and gas optimizations	PARTIAL

1.3.16 GaugeManager

ID	Severity	Summary	Status
47	LOW	Gauge distribution amounts may still be inaccurate if a distribution is only done after a full epoch has elapsed	✓ RESOLVED
48	LOW	Manual distributeRewards requires tokens to be manually sent to the GaugeManager or else it will use the minter rewards which are supposed to be distributed to gauges	✓ RESOLVED
49	LOW	Factory update functions emit incorrect events	✓ RESOLVED
50	LOW	Rewards directly sent to Algebra’s reward system will never be distributed	ACKNOWLEDGED
51	INFO	Once-per-epoch and authorization checks for distributeFees can be bypassed to some extent	ACKNOWLEDGED
52	INFO	Typographical issues and gas optimizations	ACKNOWLEDGED

1.3.17 GaugeFactory

No issues found.

1.3.18 GaugeV2

ID	Severity	Summary	Status
53	INFO	A small amount of rewardToken dust will accumulate in the gauge	✓ RESOLVED
54	INFO	The GaugeV2 does not support various special ERC-20 tokens such as fee-on-transfer tokens	✓ RESOLVED
55	INFO	Typographical issues and gas optimizations	PARTIAL

1.3.19 GaugeFactoryCL

ID	Severity	Summary	Status
56	INFO	Typographical issues	ACKNOWLEDGED

1.3.20 GaugeCL

ID	Severity	Summary	Status
57	LOW	Unstaked LP positions do not earn trading fees	✓ RESOLVED

1.3.21 BribeFactoryV3

ID	Severity	Summary	Status
58	INFO	Typographical issues	PARTIAL

1.3.22 Bribe

ID	Severity	Summary	Status
59	MEDIUM	Bribe reward claiming will erroneously send the reward to the AVM instead of the actual NFT owner if the NFT is owned by the AVM	✓ RESOLVED
60	LOW	Tokens with a fee on transfer are not supported as bribe rewards	✓ RESOLVED
61	INFO	The contract does not support a ve token with a supply larger than 2**128	✓ RESOLVED
62	INFO	Contract does not support reward tokens with a very high supply	ACKNOWLEDGED
63	INFO	Typographical issues	ACKNOWLEDGED

1.3.23 CustomPoolDeployer

No issues found.

1.3.24 PermissionsRegistry

No issues found.

1.3.25 TokenHandler

No issues found.

1.3.26 BlackTimeLibrary

No issues found.

1.3.27 BlackholePairAPIV2

No issues found.

1.3.28 veNFTAPI

ID	Severity	Summary	Status
64	INFO	Typographical issues	ACKNOWLEDGED

1.3.29 Math

No issues found.

1.3.30 AlgebraVaultFactory

No issues found.

1.3.31 CustomPluginV1Factory and CustomPluginV2Factory

ID	Severity	Summary	Status
65	INFO	Typographical issues	ACKNOWLEDGED

1.3.32 AlgebraBasePluginV3

ID	Severity	Summary	Status
66	INFO	Fee collection cannot be paused	ACKNOWLEDGED

1.3.33 BasePluginV3Factory

No issues found.

1.3.34 SecurityPlugin

No issues found.

1.3.35 SecurityRegistry

ID	Severity	Summary	Status
67	INFO	setPoolsStatus can be called by anyone if an empty pools array is provided, increasing the contract's attack surface	ACKNOWLEDGED
68	INFO	Typographical issues	ACKNOWLEDGED

2 Findings

2.1 Global

The issues in this section apply to the protocol as a whole, and may pertain to more than one contract. Please go through the issues carefully and check them in the relevant contracts.

2.1.2 Issues & Recommendations

Issue #01	**The voting and native token mechanisms are configurable, allowing the SuperNova team to control the emissions and minting system**
Severity	● GOVERNANCE
Description	The team has taken significant steps to ensure the LP stakes into the pairs and gauges are fully decentralized. However, they retain control over various other properties of the system such as the minting of the native token and the voting mechanism. Many of the peripheral contracts are upgradeable or have extensive configurability. It should also be noted that swaps can be paused by governance.
Recommendation	Consider carefully placing all privileged roles behind a carefully-chosen and secure multi-signature set-up of independent parties. Consider documenting the details of the multi-signature wallet.
Resolution	● ACKNOWLEDGED

Issue #02	Lack of error messages for require statements
Severity	● INFORMATIONAL

Description	MinterUpgradeable::148 (example)
	
	`require (epochCount >= TAIL_START);`
	
	Throughout the codebase, many `require` statements lack an appropriate error code, making it difficult for off-chain services to determine why a transaction reverted.
	
	MinterUpgradeable:23-24
	
	`uint public constant MAX_TEAM_RATE = 500;`
	`uint256 public constant TAIL_START = 67;`
	
	Throughout the contract, `uint` and `uint256` are used interchangeably. It is cleaner to stick to one of the two consistently.

Recommendation	Consider including distinct error messages for all the require statements in the codebase.
Resolution	● ACKNOWLEDGED

2.2 SuperNova

The `SuperNova` token is the native token of the SuperNova system and is minted every epoch.

The contract is configured to mint an initial amount of `500,000,000` tokens. Additional tokens can be minted by the deployer, and since the subsequent minter contract that receives the minter role is upgradeable, the minting schedule will remain adjustable by the team through an upgrade.

2.2.1 Privileged Functions

- `setMinter [ minter ]`
- `initialMint [ minter, callable once ]`
- `mint [ minter ]`

2.2.2 Issues & Recommendations

Issue #03	Typographical issues
Severity	● INFORMATIONAL
Description	Line 6 `contract Black is IBlack {` The contract and interface name could be considered misleading as this is now the `SuperNova` token.
Recommendation	Consider fixing the typographical issues.
Resolution	✓ RESOLVED This is resolved throughout the codebase.

2.3 MinterUpgradeable

`MinterUpgradeable` is the only contract with the `minter` role on the `SuperNova` token, meaning it is supposed to be the only place where SuperNova tokens get minted.

Every epoch, it expects that someone calls `update_period` to calculate the emission rate for that epoch and distribute it. This function splits the emissions over three destinations:

- Rebase to `veNOVA` stakers
- Gauge emissions
- Team share (5%)

The proportion of the emissions going to `veNOVA` holders is inversely proportional to the number of `veNOVA` stakers:

```
veNOVAAllocation = (100% - stakedInVeNOVA%)^2/2
```

This means that the rebase quickly diminishes as more people stake in `veNOVA`, but becomes a very strong incentive for staking if not that many people are staking into `veNOVA`. This effect is compounded by the fact that the actual allocation to `veNOVA` will then be split over all `veNOVA` stakers, meaning that when there are few stakers, the allocation will not only be very high, it will also be shared over less stakers. And when there are many stakers, the allocation will not only be very small, it will also be shared over all of these stakers. At the time of this audit, this allocation is quite small due to the very high number of permanently staked `veNOVA` (the "supermassive permalock"), meaning nearly all emissions are forwarded to gauge emissions.

The gauge emissions are then transferred to the gauge manager which is responsible for distributing them over the various gauges according to the vote distribution.

As the contract is upgradeable, all of its behavior is fully adjustable by the team which means that any minting restrictions such as the maximum team share are only partially binding as the team can upgrade the contract with new business logic. Variables such as the `MAX_TEAM_RATE` are thus not very meaningful.

2.3.1 Privileged Functions

- `setTeam [ team ]`
- `acceptTeam [ pending team ]`
- `setGaugeManager [ team ]`
- `setTeamRate [ team ]`
- `setRewardDistributor [ team ]`
- `transferOwnership [ owner, unused ]`
- `renounceOwnership [ owner, unused ]`

2.3.2 Issues & Recommendations

Issue #04	**Minting logic is upgradeable, allowing the proxy admin to redefine the minting logic freely and potentially mint a large unexpected amount of supply**
Severity	● GOVERNANCE
Description	Even though the minter describes a clear supply schedule, the minter is upgradeable. If a malicious actor ever upgrades it with a malicious implementation, they can potentially mint a very large amount of `NOVA` and subsequently sell it.
Recommendation	Consider adding safeguards to `NOVA` to prevent it from suddenly minting very large amounts of supply. Consider safeguarding the proxy behind a secure multisig composed of trusted, independent parties.
Resolution	● ACKNOWLEDGED

Issue #05	**The function which calculates how much of the mint should go to the veNOVA rebase is an approximation and might slightly misbehave in edge cases**
Severity	● LOW SEVERITY
Description	The portion of the weekly mint going to the `veNOVA` holders is based on the following equation: $\texttt{unstakedNOVA\%}^2/2$ This means that if no `NOVA` is staked within the voting escrow contract, 50% of all emissions go to `veNOVA` rebases. This is not very sensible as no one would be able to claim it. If all `NOVA` are staked, 0% would go to rebases. If half are staked, 12.5% goes to rebases. There are a few problems with the implementation of this math — there is no exception for when the `_veTotal` is zero, as in this case the `rebaseAmount` should be zero. Next, `_veTotal` is based on a different epoch as its proportion to the `blackSupply`, which is the total `NOVA` tokens. This means that due to timing mismatches, `unstakedNOVA` in the above equation could theoretically be negative. This is **extremely** unlikely in practice but if it happened, `calculate_rebase` and all minting would be paused until the underflow stops occuring.
Recommendation	Consider documenting this, and making sure that there is always enough liquidity staked and unstaked in the voting escrow contract to ensure these edge cases cannot occur.
Resolution	● ACKNOWLEDGED

Issue #06	The _initialize function lacks validation on the distribution amounts
Severity	● INFORMATIONAL
Description	`_initialize` is the secondary initializer called at a later time from the initial one. It mints and distributes veNOVA tokens to a set of `claimants`. This distribution logic allows the caller to specify the number of `NOVA` to mint to the minter, but there is no validation that this number is equal to the sum of claims minted. Underscoring function names is typically only done for internal functions. On a side-note, the `NOVA` approval to `_ve` can also be reset after creating all the locks, to give further confidence to users seeing an open approval from the minter.
Recommendation	Consider whether over-minting is desired, such as if the team wishes to leave a remainder within the minter. If not, consider validating that the minted amount is equal to the sum of claims.
Resolution	● ACKNOWLEDGED

Issue #07	circulating_supply can be manipulated
Severity	● INFORMATIONAL
Description	`MinterUpgradeable` exposes a `circulating_supply` view function. This function is not used significantly. If a future contract starts relying on it, we want to caution such usage as the value it supplies can be manipulated. For example, it subtracts any `NOVA` staked into the voting escrow contract from the supply. But such stakes can be made with as short as a single second duration. This means that someone might borrow `NOVA`, stake it for 1 second, and significantly decrease `circulating_supply`.
Recommendation	Consider either removing this function or documenting this.
Resolution	● ACKNOWLEDGED

2.4 RewardsDistributor

The RewardsDistributor is a small contract responsible for distributing the weekly rebase to voting escrow stakers. Every week, a small portion of the weekly emissions is distributed to voting escrow NFT holders directly, as described in the MinterUpgradeable contract description. This distribution can then be claimed by voting escrow stakers and it will be compounded into their stake, essentially rebasing the stake amount to a higher value. If the stake has expired, the rebase can be claimed as native tokens.

Rebases are assigned to epochs whenever checkpoint_token gets called with new rewards. This is done by MinterUpgradeable, but can include any tokens donated to the contract, including ones sent there directly. It should be noted that the frequency of checkpoint_token being called will affect which epoch rewards are assigned to. Assuming a schedule where the contract receives all rewards during epoch 1 and checkpoint_token is only called in epoch 3, all these rewards will be equally divided over the three epochs, even though they are for epoch 1. This is by design.

As always, the staking contract is not very robust against the case where there are no stakers. In that case, rewards may end up being unclaimed and stuck until withdrawn. This is by design.

It should be noted that the RewardsDistributor uses the historical balances of tokenIds to distribute, but distributes to the current owner. This is particularly important if a token is removed through a merge. In that case, the owner of the old tokenId is wiped and the rewards for that id can never be claimed. A similar issue occurs when claiming after the tokenId was withdrawn. Users should keep this in mind and ensure they claim their rebases before performing such actions.

It should be noted that anyone can claim the rebase for an NFT, not just the token owner or an approved address. This cannot be disabled.

This contract was audited under the assumption that the token implementation is the SuperNova token we audited, and that the VotingEscrow implementation is the one we audited as well. This is because we assume no reentrancy in claim, but with non-standard tokens and implementations that could be possible. Exercise caution if you are using a forked protocol.

2.4.1 Privileged Functions

- `setDepositor`
- `setOwner`
- `withdrawERC20`

2.4.2 Issues & Recommendations

No issues found.

2.5 PairFactory

`PairFactory` is the entry point for both basic and stable V2 pair deployment. Pairs can only be created by authorized accounts. The factory is also responsible for configuring various fee-related values for the pairs, such as their fee rate and the referral fee recipient.

Upgrading the `PairFactory` with an improperly configured implementation can block swaps on the pair.

2.5.1 Privileged Functions

- `setPause [ owner ]`
- `setFeeManager [ feeManager ]`
- `acceptFeeManager [ pendingFeeManager ]`
- `setDibs [ feeManager ]`
- `setReferralFee [ feeManager ]`
- `setFee [ feeManager ]`
- `setCustomFees [ feeManager ]`
- `setCustomReferralFee [ feeManager ]`
- `createPair [ authorizedAccounts or feeManager ]`
- `addAuthorizedAccount [ feeManager ]`
- `removeAuthorizedAccount [ feeManager ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`

2.5.2 Issues & Recommendations

Issue #08	**A referral cap was added but setCustomReferralFee remains uncapped, potentially bricking swaps when set to an excessive value**
Severity	● LOW SEVERITY
Description	The referral fee was capped to a fixed value, reducing the risk of accidentally setting it to an excessive high value and bricking swaps until it is lowered again. However, this safeguard was not implemented within the `setCustomReferralFee` function.
Recommendation	Consider adding it there as well.
Resolution	✓ RESOLVED

Issue #09	**Swap fees cannot be set to zero**
Severity	● LOW SEVERITY
Description	The swap fees can never be configured to zero within the system, as this value is treated as the "unset" state for fee overrides and is prohibited to be the default fee.
Recommendation	Consider whether this is an issue. If not, this issue will be resolved on that note. If so, consider adjusting both the `Pair` and `PairFactory` code to allow for zero fees, and use, for example, a boolean flag for the fee override to indicate whether it is set or not.
Resolution	● ACKNOWLEDGED

Issue #10	Typographical issues and gas optimizations
Severity	● INFORMATIONAL

Description

Line 5

```
import '../interfaces/IPair.sol';
```

This import is unused.

Line 17

```
uint256 public MAX_REFERRAL_FEE;
```

This variable seems to be a misnomer, as it represents the default, and within the SuperNova system, the recipient of this fee is set to a multi-signature wallet and not a referral address. However, the fee seems to be set to zero within the SuperNova system.Line 78-79

```
require(msg.sender == pendingFeeManager, "NA");
feeManager = pendingFeeManager;
```

It is slightly more idiomatic to also check that `msg.sender` is not the zero address here, since `pendingFeeManager` is zero by default. Although it is practically impossible for that to occur, it is cleaner to reset the `pendingFeeManager` to the default zero address after the `feeManager` gets set.

Lines 123-124 and 130-131

```
if (customFees[_pairAddress] > 0) {
  return customFees[_pairAddress];
...
if (customReferralFees[_pairAddress] > 0) {
  return customReferralFees[_pairAddress];
```

These sections of code are inefficient with regards to gas usage as the functions access the same storage slot twice.

Most of the governance functions lack events.

Recommendation Consider fixing the typographical issues and gas optimizations.

Resolution ● PARTIALLY RESOLVED

2.6 Pair

`Pair` is one of the main liquidity pool contracts for the system. It defines the swap and LP logic for both the basic and the stable pools and is based on the following chain of forks: It is a direct fork from Thena, which forked indirectly from Solidly, which is based on the Uniswap V2 Pair. Unlike the Uniswap V2 pair, Solidly and thus this contract also support a custom curve—the "stable" curve which uses `x³y + y³x >= k` instead of the traditional `x*y >= k` swap invariant.

Another difference compared to Uniswap V2 is that fees are sent to a separate `PairFees` contract instead of being compounded, which reduces the contract's attack surface.

The fee logic has also been made more elaborate, allowing fees to be split among several recipients: the referral fee (currently just a multisig owned by SuperNova, but set to zero) and the fee to liquidity providers. This means that unstaked LP positions earn 100% of the swap fees, unlike most other protocols where a portion of this swap fee returns to the protocol. This is because most commonly the Pair LP will be staked inside a `GaugeV2` contract, which has its own custom fee processing logic where the fee goes to the voters of that gauge.

A side effect of the fee logic is that small rounding errors will cause fee rewards to be permanently stuck in the fee contract. Additionally, there is a risk that the pair contract itself accumulates fees that will forever remain unclaimable, as the burn function expects LP value to be stored in the pair itself. If this temporary LP value accumulates any swap fees, those fees will forever be unclaimable. This is not raised as an issue as it is inherent to the design, and the expected use case is for no swaps to occur between sending liquidity to an LP and burning said liquidity.

`Pair` is meant to be solely interacted with through a peripheral contract like a router, which defines all safeguards to avoid lost tokens. Several components of the `Pair`, such as its fee mechanism, result in small rounding errors that will disfavor protocol users. Some fee tokens may become stuck in the fee contract, and users might overpay for liquidity addition/removal/swaps depending on the router implementation. This is mostly inherent to Uniswap V2. It should also be noted that `Pair` is to be deployed by the `PairFactory` through the `PairGenerator`, and that the factory should perform important checks on the tokens such as order verification.

`Pair` is not compatible with special tokens, most notably tokens with a fee on transfer, as its fee mechanism does not account for the fees. However, the client has indicated that they will coordinate with teams of such tokens to disable the fee for the SuperNova contracts, allowing them to be used. Additionally, tokens such as rebasing tokens will always cause issues. Finally, it is not compatible with tokens with an extremely large number of decimals or extremely large supplies (risk of `_f` overflowing).

Finally, the Pair defines a very rudimentary oracle, though we recommend not overly relying on it as it has the same inherent shortcomings as most on-chain oracles where issues like availability problems due to chain outages can cause it to be very brittle. It should also be noted that many of the oracle functions do not have parameter validation such as checks that window sizes are non-zero, etc. We will not raise this as an explicit issue in this audit as it is meant to save gas, and the minimum parameterizations (e.g., window size 1) are often still insufficient. However, we still recommend the client document this in their documentation.

2.6.1 Privileged Functions

None.

2.6.2 Issues & Recommendations

Issue #11	**swaps can be blocked by the SuperNova governance**
Severity	● GOVERNANCE
Description	The `swap` functions can be blocked or paused by the SuperNova team. This should be taken into consideration by teams building contracts on top of SuperNova. The `swap` function can be blocked via multiple methods: pausing the factory, upgrading the factory to revert on `isPaused()`, `dibs()`, `getReferralFee()` and `stakingNFTFee()`.
Recommendation	Consider documenting this carefully.
Resolution	● ACKNOWLEDGED

Issue #12	**Stable pair MINIMUM_LIQUIDITY requirement is lower than Uniswap V2's under very edge-case circumstances**
Severity	● LOW SEVERITY
Description	The team has significantly strengthened the minimum liquidity that gets locked for stable pairs. This is a positive change as it avoids a common issue present with many Solidly forks. However, under very specific token parameters, the requirement now appears to be lower than the original Uniswap one -- most notably when the decimals of the individual tokens are very low, e.g. less than or equal to four. The `_k` check on the minimum liquidity still appears to be sufficient requirement to avoid any issues, but given that a requirement from the original Uniswap protocol is loosened, we still want to recommend that it is at least always as strong as Uniswap's.

Recommendation	Consider updating the code-section to something like:
	`uint minimumLiquidity;`
	`if(stable) {`
	`  minimumLiquidity = Math.max(MINIMUM_LIQUIDITY, _getMinimumLiquidity(_amount0, _amount1));`
	`} else {`
	`  minimumLiquidity = MINIMUM_LIQUIDITY;`
	`}`

Resolution	● ACKNOWLEDGED

Issue #13	Fee precision mechanism may cause the pair to block swaps for tokens with an extremely large supply and large amount of generated fees
Severity	● LOW SEVERITY

Description	Line 180-182 (and 205-208)
	
	`uint256 _ratio = amount * 1e18 / totalSupply;`
	`if (_ratio > 0) {`
	`    index0 += _ratio;`
	
	During Pair `swap`s, the fees are added to an index for eventual distribution. The math for this calculation may be prone to overflow in very niche cases as `amount` is a quantity of individual tokens while `totalSupply` is a quantity of the LP supply. In theory, `index0` could thus become excessively large to eventually overflow and deny swaps on pairs.
	
	This issue is rated as low as Uniswap pairs are not supposed to work well with tokens with a large supply anyways. However, this issue is included as it can be avoided with mitigation code.

Recommendation	Consider whether such tokens or pairs will ever be added. If so, consider mitigating this by performing the addition in an overflow-safe manner, where the fee is set to zero or (for example) sent to an admin instead if it is about to overflow. This issue will be resolved on the note that such tokens are not in the scope of the protocol.
Resolution	✓ RESOLVED The client will not support such tokens.

Issue #14	**_getAmountOut and all functions relying on it are sometimes slightly inaccurate for stable pairs, causing routers to receive slightly fewer tokens**
Severity	● LOW SEVERITY

Description	Lines 441-442
	
	`function _get_y(uint x0, uint xy, uint y) internal pure returns (uint) {`
	`    for (uint i = 0; i < 255; i++) {`
	
	The `_getAmountOut` function relies on a binary search algorithm to invert the `_k` function for stable pairs. This binary search is capped at 255 iterations to prioritize liveness over accuracy. Under specific inputs, as can be achieved through fuzzing, the 255 iterations can be exhausted and `_getAmountOut` will be slightly inaccurate.
	
	Routers and other periphery contracts relying on this function may receive slightly fewer tokens in this case compared to what they theoretically could have withdrawn from the `Pair`.
	
	It should also be noted that this issue might always be present to a lesser extent due to rounding in functions such as `_f`.

Recommendation	Consider whether this is a problem. It seems that using a capped loop instead of a while loop was an explicit decision to prioritise liveness over absolute correctness. If that was indeed the goal, and it is acceptable that the router may very infrequently give back slightly fewer tokens, then this issue can be resolved on that note.
Resolution	✓ RESOLVED

Issue #15	Several functions do not have a reentrancy lock
Severity	● LOW SEVERITY
Description	The `skim`, `sync`, `mint`, `burn` and `swap` functions all have the `lock` modifier to protect them from reentrancy. However, `claimFees`, `claim-StakingFees`, `transfer` and `transferFrom` function do not have this lock, allowing for reentrancy not only between these functions, but also from functions such as `burn` into for example `transferFrom`. This was initially a concern for us as these unmarked functions all call `_updateFor`, which is an important fee-accrual function, and is thus tied to swaps, mints and burns. However, after running tests, we could not find a way to abuse this through reentrancy. We still raise this as a concern as we do not recommend violating the c-hecks-effects-interactions pattern in a codebase. That being said, we do understand that certain contracts may want to re-enter into functions such as `claimFees` or `transfer` during a `swap` and adding guards would prevent this.
Recommendation	Consider this trade-off between availability of these functions within the reentrancy hook and more formal security. If more formal security is desired, consider adding reentrancy guards to these functions.
Resolution	✓ RESOLVED The client has checked that this is fine for them.

Issue #16	current() should not be used as an actual TWAP function
Severity	● LOW SEVERITY

Description	Line 284-285
	
	`function current(address tokenIn, uint amountIn) external view returns (uint amountOut) {`
	`  Observation memory _observation = lastObservation();`
	
	The `Pair` exposes a `current` function which might be confused by other contract developers as exposing a `TWAP`. However, `current` can be easily manipulated as it simplly uses the current accumulated price since the last snapshot, which can be extremely recent.

Recommendation	Consider documenting this with a comment to avoid confusion amongst developers using this function.
Resolution	● ACKNOWLEDGED

Issue #17	Typographical issues
Severity	● INFORMATIONAL

Description	Line 183
	
	`_safeTransfer(token1, _dibs, _referralFee); // transfer the fees out to PairFees`
	
	We did not notice this typographical issue in the original audit. The fees are not transferred to the `PairFees` contract here.
	
	Lines 544-545
	
	`address recoveredAddress = ECDSA.recover(digest, v, r, s);`
	`require(recoveredAddress != address(0) && recoveredAddress == owner, 'ISIG');`
	
	The OpenZeppelin `ECDSA.recover` function already checks that the resulting address is non-zero; the first portion of the subsequent requirement is thus redundant.

Recommendation	Consider fixing the typographical issues.
Resolution	✓ RESOLVED

2.7 PairFees

`PairFees` is a sub-contract of the `Pair` contract. It is responsible for the trading fees that are meant to be distributed to liquidity providers.

During any swap, the fee amount eligible for distribution is sent into `PairFees` by the `Pair`.

When liquidity providers call `claimFees` on the `Pair`, this function will calculate the exact claimable amount for the user, mark that amount as claimed, and subsequently call `claimFeesFor` on its `PairFees` instance, thus providing the user ("liquidity provider") who called the function, the amounts to claim. These amounts are then transferred from the `PairFees` contract to the user.

`PairFees` gets deployed during the creation of `Pair` and is fully managed by said `Pair`. Each `Pair` thus has its own `PairFees` child.

As with most contracts within this codebase, `PairFees` strictly does not support negative rebasing tokens (where the balances can decrease naturally through rebases) and positive rebases will be permanently stuck within the contract.

2.7.1 Privileged Functions

None.

2.7.2 Issues & Recommendations

Issue #18	**Fee mechanism will malfunction for fee-on-transfer tokens, causing there to be insufficient fees for everyone**
Severity	● MEDIUM SEVERITY
Description	When fees are claimed, the pair will transfer them to this `PairFees` contract. This transfer will however cause a smaller amount of tokens to arrive with a popular type of token called a "fee-on-transfer" token. These tokens are known to incur a transfer fee whenever they are transferred, which is typically either burned or sent to a fee address. However, the `Pair` indiscriminately withdraws the fees as if they were all fully received by `PairFees`. If the fee is 10% for example, a user who generated $10 in fees will be able to fully claim them, even though the contract only has $9 in fees. If no other funds are present in `PairFees`, the user will not be able to withdraw their fees due to insufficient funds in the contract. Even if there are still funds, there will always be users who cannot withdraw their fees due to the 10% shortfall that will inevitably be taken from other users' share of the fees. This issue is rated as *medium* instead of *high* as fee-on-transfer tokens are only a subset of all tokens. However, Ethereum still has many of them.
Recommendation	Consider using a before-after pattern, or strictly documenting the fact that fee-on-transfer tokens are not supported for tokens where the `PairFees` contract is not whitelisted.
Resolution	✓ RESOLVED The client has indicated they will coordinate with the teams of these tokens to ensure that the SuperNova tokens are whitelisted from the fee, before adding these tokens.

2.8 PairGenerator

`PairGenerator` deploys new `Pair` instances for both basic and stable pools on the platform. The contract contains the code for a `Pair` and uses a factory pattern to deploy new instances whenever the `createPair` function is called, which is callable by anyone.

2.8.1 Privileged Functions

- `setFactory [ factory ]`

2.8.2 Issues & Recommendations

Issue #19	Typographical issues
Severity	● INFORMATIONAL

Description	Line 26
	
	`function setFactory(address _factory) external onlyFactory {`
	
	`PairFactory` does not provide functionality to call this function, though it appears upgradeable so this could be added down the line.
	
	Lines 11-13
	
	`address internal _temp0;`
	`address internal _temp1;`
	`bool internal _temp;`
	
	The naming of these variables is rather ambiguous. Consider renaming them to `tempToken0`, `tempToken1` and `tempStable`.
	
	—
	
	The new `setFactory` function lacks an event.

Recommendation	Consider fixing the typographical issues.
Resolution	● PARTIALLY RESOLVED

2.9 PairBootstrapper

The `PairBootstrapper` is a simple utility contract added to safeguard the pair creation and initial mint for both the V2 and CL pairs further. It is meant to be used for simple tokens (eg. no reentrancy tokens) as reentrancy tokens could circumvent or affect the safeguards. As the whole system is designed for simple tokens we have not re-iterated this as an issue.

For V2 pairs, it explicitly confirms that the minimum amount was burned and if not- it burns it. It furthermore does a more robust mint as the router-based mints are more brittle. That being said- the V2 mint lacks checks on the liquidity minted, but given that the pair must be created in the same transaction, this should not be a problem as long as no reentrancy occurs. Worst case, the `mint` occurs at a different price due to someone donating tokens to the address beforehand, in which case the creator gets these tokens and makes a profit. If reentrancy is possible, the mint can be affected and someone may frontrun it with a malicious price, to extract most of the value. Reentrancy guards would not protect against this so this function should simply not be used for such tokens.

For V3 pairs, it sets up an initial full range position and sends it to the provided recipient. It should be noted that the caller needs to carefully consider `initialSqrtPriceX96`, as it's independent of the provided token order.

2.9.1 Privileged Functions

- `createBasicPairAndAddLiquidity [ owner or authorized account ]`
- `createCLPoolAndAddFullRange [ owner or authorized account ]`
- `addAuthorizedAccount [ owner ]`
- `removeAuthorizedAccount [ owner ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`

2.9.2 Issues & Recommendations

Issue #20	Lack of deadline for V2 liquidity addition
Severity	● INFORMATIONAL
Description	Even though the `createCLPoolAndAddFullRange` function has a deadline parameter, the `createBasicPairAndAddLiquidity` function lacks this parameter. This means that this functionality can't be used for the V2 liquidity addition.
Recommendation	Consider adding the deadline parameter to the V2 function as well, if it's considered a useful feature.
Resolution	✓ RESOLVED

Issue #21 **Typographical issues**

Severity ● INFORMATIONAL

Description

Line 75

```
event BasicPairCreatedAndSeeded(address indexed pair,
address indexed tokenA, address indexed tokenB, bool
stable, uint liquidity, uint amountAUsed, uint amount-
BUsed, address to);
```

The token related parameters should be called 0 and 1 instead.

Line 128-129

```
amountA = tokensSwapped ? p.amountBDesired : p.amountADe-
sired;
amountB = tokensSwapped ? p.amountADesired : p.amountB-
Desired;
```

These variable names (`amountA` and `amountB`) are misnomers and should refer 0 and 1 instead.

Line 218

```
CreateCLParams memory p = p_;
```

This copy instruction appears unnecessary.

Lines 238 and 241

```
IERC20(token0).transfer(msg.sender, IERC20(token0).bala-
nceOf(address(this)));
```

	`IERC20(token1).transfer(msg.sender, IERC20(token1).balanceOf(address(this)));` These lines wrongly don’t use safetransfer, which can cause issues for specific non-compliant tokens.
Recommendation	Consider fixing the typographical issues.
Resolution	✓ RESOLVED

2.10 RouterV2

The `RouterV2` is the entry point for users to swap, add liquidity and remove liquidity from the DEX pairs. It integrates the contrentrated liquidity, basic, and stable pools all into a single interface, though certain functions only support basic or stable pools. For the concentrated liquidity pools, users will sometimes have to directly interact with the Algebra-based router.

`RouterV2` is re-configurable: the router for the concentrated liquidity and the address of WETH, which calculates all swap amounts and is indirectly used to validate the minimum amount that users receive, can all be changed. This can create risk for users, but fortunately there is no risk for open approvals to be drained to our knowledge, as transfers from the user appear to always require a message to be sent by said user.

Tokens sent by accident to the `RouterV2` can and will be taken out by searchers almost immediately, as the router provides mechanisms that allow anyone to extract approved token balances.

2.10.1 Privileged Functions

- `setSwapRouter`
- `setAlgebraFactory`
- `setAlgebraPoolAPI`
- `setWeTH`
- `transferOwnership`
- `renounceOwnership`

2.10.2 Issues & Recommendations

Issue #22	addLiquidity frontrunning protection can be bypassed
Severity	● LOW SEVERITY
Description	Basic reentrancy protection has been added to `addLiquidity`, which is intended to prevent reentrancy during token transfers that call `mint` or `skim` on the pair to extract the just-added tokens. The `minAmountOut` safeguards of the router were insufficient as they are performed on a simulation of the liquidity addition rather than on the actual result. This is a common issue with all Uniswap V2 routers that is unlikely to materialize as very few legitimate tokens allow for such reentrancy, but it is theoretically possible. The resolution that was introduced was to check that no liquidity was added or removed after the token transfers, as an attempt to check that `mint` was not called during that period. However, this is insufficient as `skim` could still be called alongside with a combination of `mint` and `burn`, which would leave the `totalSupply` unchanged. The check is therefore insufficient.
Recommendation	Consider either acknowledging the issue as most Uniswap V2 routers do, or fetching the `totalSupply` and reserves after the `mint` is done, and doing the `minAmountOut` checks on the actual reserve values of the `liquidity` returned by the `mint` call. This should guarantee that the liquidity minted to the user is valued at at least what the user inputs, but adds a few extra calls and lines of code, increasing the gas cost and complexity of the function.
Resolution	✓ RESOLVED The protection has now been removed as it was not adding security. Note that no new protection was implemented. Keep this in mind and be extremely careful with any special tokens.

Issue #23	swapPossible check for fee-on-transfer swaps will succeed prematurely
Severity	● LOW SEVERITY
Description	Within the `SwapRouter`, adjustments have been made to include any tokens accidentally sent to the pair without syncing in swap calculations. This is beneficial and improves the accuracy of the math. However, within the fee-on-transfer swaps within the `RouterV2` contract, tokens are first sent to the pair before this math is called. This causes the math to double count as the balance differential will be included as part of the input amount. This token differential is also explicitly provided to the `_swapRatio` within `getAmountOut`, guaranteeing that it will be counted exactly twice. This will cause the function to prematurely indicate that `swapPossible` is `true` even when it is not.
Recommendation	Consider refactoring the math within `_swapRatio` to support a zero `amountIn`. Consider providing an `amountIn` of zero into `getAmountOut`. Note that this will require a refactor in the `SwapHelper` as it will still need to provide the correct amount in into `pair.getAmountOut`.
Resolution	✓ RESOLVED

Issue #24	The add liquidity functions are inefficient for fee on transfer tokens and do not properly enforce the minimum amount out for them
Severity	● LOW SEVERITY
Description	Although the `addLiquidity` functions are designed to support fee-on-transfer tokens, they do not interact ideally with them. First, the minimum token amounts for fee-on-transfer tokens are not properly accounted for during the liquidity addition process. Second, the pair contract receives an imbalanced amount of tokens because fewer tokens arrive than were initially sent. This discrepancy causes the user to overpay when adding liquidity, and the excess tokens are effectively donated to the pair. As a result, the pair's liquidity increases slightly, while the token's price within the pool correspondingly decreases. Additionally, the `removeLiquidityETHSupportingFeeOnTransferT-okens` function is inefficient. It can trigger an unnecessary additional transfer fee because the token is first routed back into the router before being sent to the user.
Recommendation	Consider displaying a warning on the frontend regarding this issue. If an efficient method is desired, a fallback function can be implemented that first transfers the fee-on-transfer token and subsequently calculates the appropriate amount of the secondary token to send. However, this approach is cumbersome to implement and still does not support pairs containing two fee-on-transfer tokens. Additionally, consider raising a warning when the `ETH` liquidity removal function is invoked, recommending that users utilize the `WETH` function instead.
Resolution	✓ RESOLVED The client has indicated they will coordinate with the teams of these tokens to ensure that the SuperNova tokens are whitelisted from the fee, before adding these tokens.

Issue #25	The swap routes provided are not properly enforced to actually contain sensible data
Severity	● LOW SEVERITY
Description	The swap functions require the user to provide a swap route, which represents a chain of pairs through which the swap will be executed. This swap data is quite verbose, and much of it is not validated to ensure sensibility. For example, there is no validation that the destination token of one hop in the route matches the source token for the next hop. There is also no validation that the provided pair contract actually corresponds to the two tokens being swapped. In many cases, incorrect data can be provided, yet the function will still execute successfully because it only utilizes specific elements from the route. Similar to the issue described above, this behavior can mislead users who are inspecting their transaction data.
Recommendation	Consider validating all values of the route.
Resolution	● PARTIALLY RESOLVED

Issue #26	The router is not very robust when malicious reentrancy hooks are present on tokens within the swap route
Severity	● LOW SEVERITY
Description	If any token within the swap route permits arbitrary code execution, it can potentially weaken the security of certain router functions. For example, the `removeLiquidityETHSupportingFeeOnTransferTokens` functions could be exploited to drain their token balance by performing a reentrancy attack that re-invokes the function before the token transfer is completed. Although this scenario is theoretically possible, the likelihood is very low because such reentrancy would require the involvement of a malicious token, which legitimate users would typically avoid pairing with in a liquidity pool. Similarly, the `addLiquidity` function could, in theory, be frontrun by an attacker who triggers the mint operation in advance. However, this is also considered highly unlikely. In addition, the `amountOutMin` parameter can be effectively bypassed in swap functions that support fee-on-transfer tokens under specific conditions. For example, if a user has an active CowSwap order involving the same receipt token, a reentrancy on any token in the route could trigger the CowSwap order. Since such an order might execute at minimal cost, the tokens could be sent to the user nearly free of charge if the order price is close to market value. This would mislead the before-and-after validation pattern, making it appear as though the swap generated tokens for the user and causing the `amountOutMin` check to pass, regardless of the real swap amount. Overall, the router lacks robustness against these edge-case scenarios. Fortunately, these cases are expected to be extremely rare since all pairs are whitelisted within the SuperNova system, making their occurrence improbable.
Recommendation	It is recommended to combine the before-and-after balance validation pattern in the swap functions with an additional check on the advertised output amount of the final pair. While the actual received amount may be lower, this provides a useful secondary layer of verification. Furthermore, a highly effective additional security measure would be to simulate the swap on the frontend and validate the results of this simulation (a so-called "dry-run"). This allows potential issues to be identified before the transaction is executed on-chain. It should be noted that implementing reentrancy guards alone is insufficient, as they do not mitigate the CowSwap-based reentrancy vector.

Resolution	✓ RESOLVED The client has indicated that they will not add reentrancy tokens to their routes. The client has indicated they will coordinate with the teams of fee-on-transfer tokens to ensure that the SuperNova tokens are whitelisted from the fee, before adding these tokens.

Issue #27	**The requested minimum amount out might not be correctly enforced for special tokens**
Severity	● INFORMATIONAL
Description	The requested minimum output amount is validated against the amount that the pair attempts to send to the user, rather than the actual amount the user ultimately receives. If the final token in the swap has special behavior, this can result in the minimum output requirement not being properly enforced.
Recommendation	It is recommended to document this behavior to ensure clarity and prevent misuse. Implementing a before-and-after balance validation pattern alongside the existing strict output check could provide an ideal balance between safety and functionality.
Resolution	● ACKNOWLEDGED

2.11 RouterHelper

`RouterHelper` is an upgradeable sub-contract of `RouterV2`. It performs the pre-calculations for the all the individual swaps of a swap route, and facilitates the integration of concentrated liquidity, stable and basic pairs all into a single swap.

2.11.1 Privileged Functions

- `transferOwnership`
- `renounceOwnership`

2.11.2 Issues & Recommendations

Issue #28	Swap prices for stable pairs will mostly return either "0" or "1" due to a normalization error
Severity	● MEDIUM SEVERITY
Description	The stable swap prices have been updated by the client to be correct, as previously the returned prices were quite arbitrary. The prices are now calculated within `_calculateStableSwapPrice-`, which calculates the derivative of the stable curve invariant. Though this derivative appears correct, the normalization of the result is incorrectly set to `1` instead of `1e18`. This causes the prices to be rounded to full numbers. For most stable pairs, since their prices mostly hover around the 1:1 exchange rate, their returned prices will mostly be either `0` or `1` due to the normalization and rounding down behavior of the function. The prices returned from the `RouterHelper` functions do not appear to be used anywhere, so this issue is only relevant for integrating contracts.
Recommendation	Normalize the result correctly. Take into account that division before multiplication causes precision to be lost, while doing multiplication before division can cause overflow with high supply tokens. A trade-off should thus be made.
Resolution	✓ RESOLVED We remind integrators that the new prices are accurate but have complex rounding behavior, as both the math numerator and denominator round down.

Issue #29	getAmountsOut does not handle failures gracefully
Severity	● LOW SEVERITY
Description	`getAmountsOut` has multiple branches in its body that can fail. `If` statements are created for all of them, and risky calls are wrapped in try-catch blocks. The concern is that the exception cases are only properly handled in a single branch, while all other exception branches are not properly handled. Only in a single branch is the function returned early via a `break` statement. In all other branches, such as when the pair address is not a V2 pair and when the try statements fail, iteration will be attempted to continue instead of failing early. In the commits for this review, an attempt was made to resolve this, and nearly all of the locations of the above issue now have break statements. However, handling is not included in the exception scenario where the following if-statement is incorrect.
Recommendation	Consider handling the missed scenario as well.
Resolution	✓ RESOLVED

Issue #30	_calculateStableSwapPrice is slightly vulnerable to overflow reverts
Severity	● INFORMATIONAL

Description	`_calculateStableSwapPrice` accurately calculates the marginal swap price of stable pairs given their reserves. However, the math within this function can become a bit large for tokens with a very high supply and low number of decimals:
	
	Lines 188-193
	
	`uint normR0 = decimals0 <= 18 ? reserve0 * 10**(18 - decimals0) : reserve0 / 10**(decimals0 - 18);`
	`uint normR1 = decimals1 <= 18 ? reserve1 * 10**(18 - decimals1) : reserve1 / 10**(decimals1 - 18);`
	
	`uint r0Sq = normR0 * normR0 / 1e18;`
	`uint r1Sq = normR1 * normR1 / 1e18;`
	`uint den = normR0 * (r0Sq + 3 * r1Sq);`
	
	Given pairs with large reserves, this math will revert seemingly more quickly than the stable pairs will start reverting (all swap pairs have an upper limit on reserve sizes as well).

Recommendation	This does not need to be resolved if such tokens are not planned to be added.
Resolution	✓ RESOLVED We remind integrators that the new prices have complex rounding behavior.

Issue #31	_swapRatio fixes have small edge cases which can cause reverts and small side-effects in very specific situations
Severity	● INFORMATIONAL

Description	Line 156
	
	`uint actualAmountIn = amountIn + (pairSwapMetaData.balanceA - pairSwapMetaData.reserveA);`
	
	The brackets here seem unnecessary and slightly increase underflow revert risk if the token is rebasing with a contracting supply. Obviously such tokens are not well supported in Uniswap in the first place.
	
	Line 158
	
	`pairSwapMetaData.balanceA += amountIn - feeAmount;`
	
	A similar concern to the above issue is present here, where if `feeAmount` exceeds `amountIn` due to there being a large balance increase, the function reverts. Arguably reverting in this scenario is probably desirable as this is an odd and unnatural scenario indicating that the user is likely being abused somehow. No changes are thus needed from our side with regards to this.
	
	`afterReserveB` is more accurately set to `balanceB - amountOut` as well.

Recommendation	Consider fixing the above concerns with the new `_swapRatio` implementation as you see fit. These are very niche edge cases so this issue can also simply be acknowledged.
Resolution	✓ RESOLVED

Issue #32	getAmountOut will still return a value even if the factory paused V2 swaps
Severity	● INFORMATIONAL
Description	The getAmountOut function does not account for the fact that the factory might pause v2 swaps.
Recommendation	It is recommended to evaluate whether this constitutes a security or operational issue. If determined to be problematic, consider adding checks for the `isPaused` state to all relevant functions
Resolution	✓ RESOLVED The client has confirmed this is not a concern for them.

Issue #33	Typographical issues and gas optimizations
Severity	● INFORMATIONAL
Description	Line 8

```
import '@cryptoalgebra/integral-periphery/contracts/int-
erfaces/ISwapRouter.sol';
```

This import appears unused.

Lines 12 and 14

```
import "@openzeppelin/contracts-upgradeable/access/Owna-
bleUpgradeable.sol";
contract RouterHelper is OwnableUpgradeable {
```

`OwnableUpgradeable` appears to currently be unused, though we understand it being included for future upgrades.

Line 32

```
address public factory;
```

All other address variables are cast to their type. Consider casting this to `IPairFactory` as well to avoid having to cast it in all use cases.

Lines 108-109

```
(uint decimals0, uint decimals1, , , , ) = IPair(rout-
es[i].pair).metadata();
(uint beforeReserve0, uint beforeReserve1,) = IPair(rou-
tes[i].pair).getReserves();
```

All of these values are already fetched within the earlier `_swapRatio` call. Fetching them again wastes gas.

Line 141

```
uint afterReserveA = pairSwapMetaData.reserveA +
(amountIn - (amountIn * IPairFactory(factory).getFee(p-
air, pairSwapMetaData.stable) / 10000));
```

This fee was already fetched on line 138, consider simply re-using it.

Line 165

```
// performs chained getAmountOut calculations on any
number of pairs
```

The function below this comment does not do any chained operations.

Line 210

```
// calculates the CREATE2 address for a pair without
making any external calls
```

Actually, the function below this does not use CREATE2 and does use an external call.

Line 212

```
(address token0, address token1) = sortTokens(tokenA,
tokenB);
```

This is currently unnecessary as the `getPair` function below it is robust against the ordering of the tokens.

Recommendation	Consider fixing the typographical issues and gas optimizations.
Resolution	● PARTIALLY RESOLVED

2.12 VotingEscrow

`VotingEscrow` is the primary governance contract for the system. It defines the voting escrow NFT token, which is a locked/staked variant of the native token and is based on the Curve `veCRV` token.

Similar to the original Curve voting-escrow, it facilitates locks of up to 4 years to be created. These locks then decay linearly in voting power over those four years. If locks are created with a smaller lock duration such as two years, their voting power starts off proportionally lower, i.e., 50% of the amount of voting escrow locked for 2 years. The voting power will then decay at the same rate for the remaining two years. The system also provides the option of "permanently" locking a position, which disables decay. This position can then be converted back at any time to re-enable the decay, but doing so will lock it for four years.

A unique feature is that native tokens can be permanently staked without the option to unlock as well, which permanently burns the underlying native tokens. This type of NFT is called an `SMNFT`. A bonus of 10% in voting power is granted to such locks — this bonus is configurable.

The `VotingEscrow` locks are represented as transferable NFTs. However, transfers are disabled while the NFT has votes cast in `VoterV3`.

Users can mint new locks for themselves and others, and they can increase the duration or value of their own locks as well as increase the value of others' NFTs. If enabled by governance, they can split their NFTs into two. They can merge their NFTs at any time — merging NFTs will combine them into a single NFT with the largest lock duration and upgrade the other NFT into permanent/sMNFT status if it is not already that. It allows an NFT to be moved into lock and unlock status, or upgraded to sMNFT (which is non-reversible).

When NFTs are eventually withdrawn back into the native token after they expire, the native tokens are sent to the caller of that transaction which can be the owner of the NFT but can also be an approved caller.

`VotingEscrow` positions can vote on gauges such that they receive weekly emissions, and receive the bribes and trading fees of those pools as compensation. They also receive a small weekly rebase staking reward.

2.12.1 Privileged Functions

- `setTeam`
- `setArtProxy`
- `setVoter`
- `toggleSplit`
- `setSmNFTBonus`

2.12.2 Issues & Recommendations

Issue #34	NFTs are incorrectly self-delegated during NFT transfers
Severity	● HIGH SEVERITY
Description	The vote delegation logic was redesigned after several concerns were raised in the original audit. However, the refactored code still contains several issues. Token mints, burns, and transfers all move the delegation from the previous owner of the NFT to the new owner. This is flawed as the delegation logic assumes delegation is always moved from the delegatee of the previous owner to the delegatee of the new owner. This becomes an issue because the `delegate` functions still properly move all tokens from the previous delegatee to the new delegatee. The combination of these two different behaviors allows a token to be delegated to multiple accounts. For example, assume Alice executes: 1. `delegate(bob)` 2. `create_lock(…)` - Adds the minted `tokenId` to the delegated tokens of `alice` 3. `delegate(alice)` - Adds the minted `tokenId` **AGAIN** to alice This sequence will end up with the `tokenId` being included twice in Alice's delegations list, allowing the various `getPastVotes` functions to double count her voting power.
Recommendation	Consider removing all delegation logic completely if there is no planned use for it. Alternatively, consider consistently using the delegates for the `moveTokenDelegates` call. The `ownerOf` checks within `moveTokenDelegates` still appear fully incorrect and the `MAX_DELEGATES` DDOS risk still appears to be present. We do not recommend using this delegation logic in its current state.
Resolution	✓ RESOLVED Delegation was fully removed.

Issue #35	Split can now leave one of the two NFTs with a zero value, an outcome which was previously impossible
Severity	● LOW SEVERITY
Description	Line 1181 `require(_splitAmount > 0 && _splitAmount <= newLocked.amount, "ISA");` The second portion of the above requirement was `newLocked.amount > _splitAmount`, meaning that the full amount could not be split off previously, but now it can be. This means that the `split` function can be used to mint NFTs with a zero value when it was initially impossible and thus increases the attack surface of this contract. As fixes were being iterated quickly with the client, this was immediately fixed when indicated to them; hence the impact was not investigated further.
Recommendation	Consider reverting to a strict lesser than check.
Resolution	✓ RESOLVED The requirement is made strict again.

Issue #36	ownerOf does not revert for invalid tokenId inputs
Severity	● LOW SEVERITY
Description	`ownerOf` does not explicitly revert for invalid inputs, which is not an expected property for NFTs. This is worsened by the fact that a zero owner is sometimes used to denote that the NFT does not exist, but NFTs can be transferred to the zero address while they still exist.
Recommendation	Consider explicitly reverting this function. However, other locations use this function to check whether a token exists, and those functions may break if it starts reverting. All relevant locations would need to be refactored.
Resolution	● ACKNOWLEDGED

Issue #37	Unsafe casts occur throughout the contract which reduces code-safety, especially if SUPERNOVAs supply ever increases significantly
Severity	● LOW SEVERITY
Description	Throughout the contract, inputs and balances get downcast into smaller types without explicitly checking against overflows. This is typically fine as the `SUPERNOVA` token supply will not be large enough for such overflows to happen, but as demonstrated in a high severity overflow issue, it can easily cause for issues to slip through. If this contract were to be ever re-used for a token with a larger supply, it may also cause issues.
Recommendation	Consider carefully going through the contract and refactoring all locations where unsafe casts occur.
Resolution	● ACKNOWLEDGED

Issue #38	Several functions lack reentrancy guards
Severity	● INFORMATIONAL
Description	`VotingEscrow` has reentrancy guards. It is not immediately clear why this is, but the argument to add them out of precaution is understood. However, currently some key functions lack such guards and could still be reentered into if reentrancy was possible. The user facing functions are: `checkpoint`, `transferFrom`, `safeTransferFrom`, `lockPermanent` and `unlockPermanent`. This issue is raised informationally as reentrancy hooks are not observed, except with `safeTransferFrom`, but there it happens at the end and is still permitted to reenter (though from this perspective, reentrancy at the end there is desired).
Recommendation	Consider adding reentrancy guards to the above functions if there is an actual reason for the guards on the other functions. Pay explicit attention to the reentrancy guard for `safeTransferFrom`: we recommend adding it to `_transferFrom` instead to allow reentrancy from the reentrancy hook into the various other functions, as often the receipt contract will want to immediately do something with the token, and would otherwise be prevented from doing so. Once a guard is added to `_transferFrom`, do not add an additional guard to `transferFrom` as they will be nested and revert.
Resolution	● ACKNOWLEDGED

Issue #39	getsmNFTPastVotes seems to calculate the NFT balance twice
Severity	● INFORMATIONAL

Description	Lines 1306-1308
	
	`if ((up.smNFT + up.smNFTBonus) == 0) continue;`
	`// Use the provided input timestamp here to get the right decay`
	`votes = votes + VotingBalanceLogic.balanceOfNFT(tId, timestamp, votingBalanceLogicData);`
	
	It is unclear why this function does not simply add `up.smNFT + up.smNFTBonus` to `votes`. If everything is implemented correctly, this appears to be what `balanceOfNFT` should return.

Recommendation	Consider investigating the above optimization to see if there is any reason to not directly implement `balanceOfNFT`. A don't-repeat-yourself argument could be made for the more expensive code, acknowledging this issue is therefore understandable as it would reduce the risk for future refactors.
Resolution	✓ RESOLVED

Issue #40	Typographical issues
Severity	● INFORMATIONAL

Description	Line 1324
	
	`function _delegate(address delegator, address delegatee) internal {`
	
	This function appears to be unused and can be removed.
	
	Line 1379
	
	`if (delegatee == address(0)) delegatee = signatory;`
	
	Though harmless, a non-zero check already occurs at the start of this function.

Recommendation	Consider fixing the typographical issues.
Resolution	✓ RESOLVED

2.13 VotingBalanceLogic

`VotingBalanceLogic` is a library used within the `VotingEscrow` contract. It is responsible for calculating historical balances and total supplies, based on the checkpoint data recorded in the `VotingEscrow` contract.

Vulnerabilities which have been described in the `VotingEscrow` contract might have their root cause within this library. The reader is recommended to read these two sections of the report together. The casting issues from `VotingEscrow` in the initial audit are present here as well and should be resolved here as well.

The block-based `totalSupply` function will start malfunctioning if no snapshot occurs for 255 weeks. This seems unlikely enough to not explicitly describe as an issue.

The epochs are now overwritten if they occur in the same timestamp. This means if multiple blocks occur in a single timestamp, only the last one will be used. Data fetched for previous blocks will be approximated, even though this data was present at some point. The client is advised to be extremely careful with block-based functions, as even if they are fetched for historical blocks, their data may not be final and can change until the block is fully locked in. Using the block-based balance functions without consulting an auditor for every use case is strongly discouraged.

2.13.1 Privileged Functions

None.

2.13.2 Issues & Recommendations

Issue #41	Block-based historical balance and total supply functions may not be consistent until a snapshot occurs after the provided block
Severity	● LOW SEVERITY
Description	The block-based `totalSupplyAt` and `balanceOfAtNFT` approximate the balance and supply at the provided historical block by interpolating the two snapshots around the provided block number. The issue occurs when there are no snapshots after the provided block number. In this case, the interpolation occurs with the snapshot before it and the current block's number and timestamp. This interpolation will change as soon as an actual snapshot occurs with the real `block.number`, alongside when any new block is produced. This means that `totalSupplyAt` and `balanceOfAtNFT` will return changing values for a fixed historical block until a snapshot locks that historical block in.
Recommendation	Avoid using the block based accounting functions, since the interpolation logic is an approximation to some extent anyway.
Resolution	✓ RESOLVED The client confirms they will avoid using this. No code changes were made.

Issue #42	Typographical issues
Severity	● INFORMATIONAL

Description	Line 15
	
	`/// @notice Get the current voting power for `_tokenId``
	
	This comment is outdated as the function below it gets a historical value.
	
	Lines 16, 68 and 234
	
	`/// @dev Adheres to the ERC20 `balanceOf` interface for Aragon compatibility`
	`/// @dev Adheres to MiniMe `balanceOfAt` interface: https://github.com/Giveth/minime`
	`/// @dev Adheres to the ERC20 `totalSupply` interface for Aragon compatibility`
	
	This token is an NFT and not an ERC20 token. These comments are likely outdated and incorrect.
	
	Lines 77-78
	
	`// Copying and pasting totalSupply code because Vyper cannot pass by`
	`// reference yet`
	
	These comments appear outdated as the contract is written in solidity.
	
	Lines 156 and 159
	
	`/// @notice Binary search to estimate timestamp for block number`
	`/// @return Approximate timestamp for block`
	
	This function does something else.
	
	Finally, this contract has identical casting issues to `VotingEscrow`.

Recommendation	Consider fixing the typographical issues.
Resolution	● PARTIALLY RESOLVED

2.14 VotingDelegationLib

`VotingDelegationLib` is a utility contract used by the `VotingEscrow` contract to handle most of its "delegate" voting logic, where `veNOVA` NFT holders can delegate their voting rights (though not the `VoterV3` voting rights) to a different wallet.

It is used on any token transfer to update the index of which `tokenIds` have been delegated to specific wallets.

This contract has been fully removed from the codebase.

2.14.1 Privileged Functions

- `setTeam [ team ]`
- `setVotingEscrow [ team ]`

2.14.2 Issues & Recommendations

Issue #43	**Anyone can call the internal moveTokenDelegates and moveAllDelegates functions, allowing for exploiters to delegate arbitrarily to their own wallets and fully breaking the delegation logic**
Severity	● HIGH SEVERITY
Description	During the changes made in this audit scope, `VotingDelegationLib` was moved from an external library to a separate contract. External libraries are contracts deployed at a separate address but called from within the context of the `VotingEscrow` contract using `DELEGATECALL`. This means that even though the functions can be called by anyone, the context of the `VotingEscrow` address will not be used and there is no problem if they are called directly. During the changes that triggered this diff-based audit, the client moved this contract to a real separate contract. This means that the new `VotingDelegationLib` is no longer delegated to but instead gets called directly. It now has its own internal storage that is used by all callers. The client did not fully realize this and forgot to add authorization to the critical functions of this contract, meaning anyone can call them. This allows anyone to arbitrarily move delegates for `tokenIds` and even allow a single `tokenId` to be added arbitrarily many times as a delegation. This fully breaks the system, and any other systems relying on delegation vote counts will be exploitable as the vote counts can be arbitrarily inflated.
Recommendation	Consider only allowing the external functions to be called by the `votingEscrow` address.
Resolution	✓ RESOLVED The delegation logic has been fully removed.

Issue #44	The delegation logic is fundamentally broken in multiple ways, which can be abused to DoS VotingEscrow mints and transfers, prevent delegations to any wallet and clear a wallet's delegates at will
Severity	● HIGH SEVERITY

Description

`VotingEscrow` supports functions to delegate voting rights to other wallets. Whenever an NFT is transferred to a different wallet, the delegation gets updated to the configured delegate of the recipient.

This logic is fundamentally broken for multiple reasons.

Reason 1: Moving delegates to a recipient is not permitted if it causes them to have more than `MAX_DELEGATES` delegated `tokenIds`.

Lines 96-99

```
require(
    dstRepOld.length + 1 <= MAX_DELEGATES,
    "tokens>1"
);
```

Whenever a token is transferred to a wallet (mints, transfers and splits), its added to the list of delegated tokens of the delegatee of the recipient. This addition reverts as soon as the delegatee has reached `MAX_DELEGATES` to avoid the gas consumption of the various for-loops becoming too excessive (though it will already be very high at `MAX_DELEGATES`).

The critical issue is that minting and sending tokens to a wallet is nearly free, as there is no minimum value attached to tokens nor a real minimum stake duration. A malicious actor can permanently DoS **all** `veNFT` mints and transfers in the public mempools permanently by consistently frontrunning them with a transaction that mints 1024 tokens to the destinations delegatee.

Reason 2: Fundamental logic flaw in `moveTokenDelegates` allows an exploiter to clear delegates for any wallet without consent of the delegators.

Lines 68-73

```
if(_isCheckpointInNewBlock) {
  if(ownerOfFn(tId) == srcRep) {
    srcRepNew.push(tId);
  }
  i++;
} else {
  if(ownerOfFn(tId) != srcRep) {
    srcRepNew[i] = srcRepNew[length -1];
    srcRepNew.pop();
    length--;
  } else {
    i++;
}
```

The above extract of code is taken from the `moveTokenDelegates` function. This function is called whenever an NFT is moved from wallet A to B, and moves the `tokenId` from the delegate index of delegate(A) to delegate(B). In the above snippet, `srcRep` is delegate(A), while `ownerOf(tId)` is always `address(0)` due to the ordering within `transferFrom`.

Delegates can never be `address(0)` due to the default case being self-delegation. This means that the code-snippet above will consistently not push any `tokenId` to new checkpoints and clear the whole array of an existing checkpoint in the other branch. In other words, the whole delegation logic is fundamentally broken. Even though it is already quite broken in normal usage, an exploiter can also target specific wallets to clear their delegates by first delegating to them and subsequently removing this delegate.

This issue has been copied from the initial audit as neither were resolved at the commit of the preliminary scope.

Recommendation

Resolving these issues while retaining a correct index will require a fundamental rewritting of the whole delegation algorithm. This may even require advanced binary search/tree algorithms to make it work in an acceptable time-complexity. It is clear that achieving the desired use-case without `MAX_DELEGATES` will be an engineering endevour outside of the scope of this audit. A short-term fix could be to only add tokens to delegates if their value exceeds a certain value, and requiring wallets to opt into receiving delegations.

Since the `VotingEscrow` is not upgradeable, there is not much which can be done here in practice, apart from acknowledging this issue and not using the delegation logic.

Resolution

✓ RESOLVED

The delegation logic has been fully removed.

Issue #45	Typographical issues
Severity	● INFORMATIONAL
Description	`setTeam` and `setVotingEscrow` lack events.
Recommendation	Consider fixing the typographical issues.
Resolution	✓ RESOLVED The delegation logic has been fully removed.

2.15 VoterV3

VoterV3 is the entry point for voting escrow holders to vote on which gauges should receive emissions. It keeps track of all issued votes and is called by the GaugeManager when it needs to determine how to distribute the weekly emissions across all gauges.

VoterV3 interacts with the VotingEscrow contract directly to exchange information on voting escrow position changes, so that the voter is updated appropriately and that certain actions are not possible while a voting escrow NFT has outstanding votes on it. However, VoterV3 does not account for the decay effect of the voting escrow NFTs. It has a feature where users can trigger the decay for other NFTs by calling poke, but if this is not done, the voting weight of an NFT will remain identical to when the initial vote was cast, even after the NFT is fully decayed to a zero balance.

The audit has been conducted under the assumption that the bribes, which also get a notification whenever a vote gets cast, do not perform any external calls. This audit should not be used if the implementation of the bribes changes to one where they interact back with the voter, as the bribes would potentially interact with it during an inconsistent state, which can lead to issues. This audit should only be used with the current version of all contracts in the system, and not when there are changes to how certain contracts interact with each.

The voter has been slightly redesigned to keep an index of the voting totals for each epoch that passes. This way, the GaugeManager can more correctly use the votes. This system will only work correctly as long as at least a single vote occurs each epoch. Ideally, all pools should be polled via checkpointPoolWeightsForNextEpoch. The checkpoint system could've been avoided by keeping track of the last epoch where a change occurred and simply using the votes from that point. If certain pools are not checkpointed, the system should still function, but the total sum of votes for that epoch will be less than the epochTotalWeight for that epoch. This means that some rewards will become permanently unclaimable.

2.15.1 Privileged Functions

- `setMaxVotingNum [ VOTER_ADMIN ]`
- `setGaugeManager [ VOTER_ADMIN ]`
- `setEpochOwner [ owner ]`
- `setPermissionsRegistry [ owner ]`

- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`

2.15.2 Issues & Recommendations

Issue #46	Typographical issues and gas optimizations
Severity	● INFORMATIONAL

Description

Lines 4, 6-8, 17, 18, 21, 27, 28, 83-86, 88-90

```
import './libraries/Math.sol';
import './interfaces/IERC20.sol';
import './interfaces/IPairInfo.sol';
import './interfaces/IPairFactory.sol';
import "@openzeppelin/contracts-upgradeable/token/ERC20-
/utils/SafeERC20Upgradeable.sol";
import "@openzeppelin/contracts-upgradeable/token/ERC20-
/IERC20Upgradeable.sol";
using SafeERC20Upgradeable for IERC20Upgradeable;
address[] public pools;
address public epochOwner;
modifier Governance() {
  require(IPermissionsRegistry(permissionRegistry).hasR-
ole("GOVERNANCE",msg.sender), 'GOVERNANCE');
  _;
}
modifier GenesisManager() {
  require(IPermissionsRegistry(permissionRegistry).hasR-
ole("GENESIS_MANAGER", msg.sender), 'GENESIS_MANAGER');
  _;
}
```

All of these lines appear unused, alongside with the functions related to the `pools` and `epochOwner`. All of this can be deleted from our perspective, though we recommend making sure that no third-party contracts are using any of these functions. Given that the contracts are already in production this can probably not be guaranteed.

Line 25

```
address internal base; // $the token
```

This should say the `$NOVA` token instead.

Line 31

```
uint public EPOCH_DURATION;
```

Consider marking this as `constant` instead.

Line 42

```
// nft => timestamp of last vote (this is shifted to
thursday of that epoc)
```

This should say "epoch" instead.

Lines 45-46

```
event Voted(address indexed voter, uint256 tokenId,
uint256 weight);
event Abstained(uint256 tokenId, uint256 weight);
```

These events should include the `pool` address. It's also not clear to use why the indexation is inconsistent.

Line 88

```
modifier EpochManagerOrVoterAdmin() {
```

This new modifier appears unused, alongside other modifiers. Perhaps it was added for future-proofing.

Line 156

```
votes[_tokenId][_pool] -= _votes;
```

Consider simply setting this to zero to save gas.

Lines 159, 160, 238 and 239

```
IBribe(internal_bribe).withdraw(uint256(_votes),
_tokenId);
IBribe(external_bribe).withdraw(uint256(_votes),
_tokenId);
IBribe(internal_bribe).deposit(uint256(_poolWeight),
_tokenId);
IBribe(external_bribe).deposit(uint256(_poolWeight),
_tokenId);
```

Nearly all setters lack events and gas optimizations.

Line 255-256 (example)

```
totalWeight += _usedWeight;
epochTotalWeight[epochNext] = totalWeight;
```

	The second line re-fetches values from storage even though they could've been stored in memory to save gas. This pattern occurs with all the check-pointing logic added in this PR.
Recommendation	Consider fixing the typographical issues.
Resolution	● PARTIALLY RESOLVED

2.16 GaugeManager

GaugeManager is responsible for keeping track of all gauges which can be voted on to receive emissions. Anyone can create new gauges for pools which have been deployed by the official pair factories.

After gauge creation, the gauge and its related liquidity pool are stored and bribe contracts are created for them. Every epoch, the GaugeManager will distribute the weekly emissions to all registered gauges based on the votes each received relative to the others. This voting happens in VoterV3.

GaugeManager also defines some utility functions for users to claim rewards from multiple gauges and bribes all at once, instead of having to create a transaction for each of them.

GaugeManager has several highly intrusive setter functions which can replace core dependencies such as the farmingParam, minter, voter and bribeFactory. Replacing these can and likely will lead to severe side-effects such as complete mis-accounting of emissions. Auditor confirmation is always recommended before adjusting any intrusive setters.

Within this code update, votes must be present for emissions to be distributed to gauges. This may create a deadlock in the minter if there are no votes yet and no one has tokens to vote. As long as users have tokens to vote, which is more realistic, it appears that in this scenario at least one week must pass before resolution. This scenario should be avoided by ensuring votes are always present.

2.16.1 Privileged Functions

- setBribeFactory [GAUGE_ADMIN]
- setVoter [GAUGE_ADMIN]
- setBlackGovernor [GAUGE_ADMIN]
- distributeFees [GAUGE_ADMIN]
- distributeAll [GAUGE_ADMIN]
- distributeRewards [GAUGE_ADMIN]
- distribute [GAUGE_ADMIN]
- setMinter [GAUGE_ADMIN]

- updateGaugeFactory [GAUGE_ADMIN]
- updateGaugeFactoryCL [GAUGE_ADMIN]
- updatePairFactory [GAUGE_ADMIN]
- updatePairFactoryCL [GAUGE_ADMIN]
- acceptAlgebraFeeChangeProposal [GAUGE_ADMIN]
- distributeFees [EPOCH_MANAGER or GAUGE_ADMIN]
- distribute [EPOCH_MANAGER or GAUGE_ADMIN]
- carryForwardTotalVotesForNextEpoch [EPOCH_MANAGER or GAUGE_ADMIN]
- carryForwardVotesForNextEpoch [EPOCH_MANAGER or GAUGE_ADMIN]
- killGauge [GOVERNANCE]
- reviveGauge [GOVERNANCE]
- setPermissionsRegistry [owner]
- setAlgebraPoolApiStorage [owner]
- transferOwnership [owner]
- renounceOwnership [owner]

2.16.2 Issues & Recommendations

Issue #47	**Gauge distribution amounts may still be inaccurate if a distribution is only done after a full epoch has elapsed**
Severity	● LOW SEVERITY
Description	The behavior of gauge distributions has been significantly improved. Previously, there was a race condition where the total eligible amount of votes was considered to be the amount of votes when `notifyRewardAmount` was called, but then the vote amounts for distributing these rewards to the gauges were used as the vote amounts at the time this distribution was called. Now, the mechanism has been improved to always use the last total vote count of the epoch before `notifyRewardAmount` was called, and the last vote counts of the epoch before the distribute functions are called. This means that as long as both of these functions are called within the same epoch (7 days), the count is accurate. This leaves the risk that if the gauge distribution does not occur for a full epoch, it will misassign rewards in the subsequent epoch.
Recommendation	Consider always distributing to all gauges every epoch. Also consider adding default behavior which prevents distribution if a full epoch has elapsed. A new function with an explicit boolean to perform an incorrect manual distribution could be added to catch up.
Resolution	✓ RESOLVED The client has not made changes to this behavior but is aware of this and will ensure that they are always timely called. The possibility therefore theoretically persists that they don't, eg. if a week-long chain outage occurs.

Issue #48	Manual distributeRewards requires tokens to be manually sent to the GaugeManager or else it will use the minter rewards which are supposed to be distributed to gauges
Severity	● LOW SEVERITY
Description	During the changes, a new manually callable `distributeRewards` functions was added. This function can only be called by the `GAUGE_ADMIN`. Its role is to let the team grant additional rewards to particular gauges in the ongoing epoch. This function simply uses any tokens already present in the `GaugeManager`. By default, these are only the tokens minted to it for automatic distribution, and would thus be incorrectly assigned to these manual rewards. The way to prevent this, which is likely what the team would do, is to manually send additional rewards into the `GaugeManager` before calling the `distributeRewards` function. This seems prone to error.
Recommendation	Consider transferring the rewards via `safeTransferFrom` into the `GaugeManager` at the top of the function, after the `require` statement.
Resolution	✓ RESOLVED

Issue #49	Factory update functions emit incorrect events
Severity	● LOW SEVERITY

Description

Lines 567-597

```
function updateGaugeFactory(address _gaugeFactory)
external GaugeAdmin {
  require(_gaugeFactory != address(0), "ZA");
  require(_gaugeFactory.code.length > 0, "CODELEN");
  require(_gaugeFactory != gaugeFactory, "NA");
  gaugeFactory = _gaugeFactory;
  emit SetGaugeFactory(gaugeFactory, _gaugeFactory);
}

function updateGaugeFactoryCL(address _gaugeFactoryCL)
external GaugeAdmin {
  require(_gaugeFactoryCL != address(0), "ZA");
  require(_gaugeFactoryCL.code.length > 0, "CODELEN");
  require(_gaugeFactoryCL != gaugeFactoryCL, "NA");
  gaugeFactoryCL = _gaugeFactoryCL;
  emit SetGaugeFactoryCL(gaugeFactoryCL, _gaugeFacto-
ryCL);
}

function updatePairFactory(address _pairFactory)
external GaugeAdmin {
  require(_pairFactory != address(0), "ZA");
  require(_pairFactory.code.length > 0, "CODELEN");
  require(_pairFactory != pairFactory, "NA");
  pairFactory = _pairFactory;
  emit SetPairFactory(pairFactory, _pairFactory);
}

function updatePairFactoryCL(address _pairFactoryCL)
external GaugeAdmin {
  require(_pairFactoryCL != address(0), "ZA");
  require(_pairFactoryCL.code.length > 0, "CODELEN");
  require(_pairFactoryCL != pairFactoryCL, "NA");
  pairFactoryCL = _pairFactoryCL;
  emit SetPairFactoryCL(pairFactoryCL, _pairFactoryCL);
}
```

The events emitted in the new factory update functions emit the same variable twice due to an ordering issue in the event emission.

	The goal of the event is however to first emit the old value and subsequently emit the updated value.
Recommendation	Consider moving the event to above the line which updates storage.
Resolution	✓ RESOLVED

Issue #50	**Rewards directly sent to Algebra's reward system will never be distributed**
Severity	● LOW SEVERITY
Description	The Algebra reward system allows any address to invoke `AlgebraEternalFarming::addRewards`. Due to the logic in `notifyRewardAmount` within the gauge manager, the reward rate will only ever be based on rewards distributed via the `GaugeCL` instance. Any funds sent via `addRewards` appear to be locked until the gauge ceases updating the reward rate.
Recommendation	`addRewards` should be restricted such that it can only be called by the gauges.
Resolution	● ACKNOWLEDGED

Issue #51	Once-per-epoch and authorization checks for distributeFees can be bypassed to some extent
Severity	● INFORMATIONAL
Description	The `distributeFees` functions include several checks: only authorized callers may invoke them, and they can execute `claimFees` only once per epoch. These checks can be almost entirely bypassed (except for the portion that withdraws from the community vaults for CL gauges) by directly invoking `claimFees` on the gauges. This function lacks any authorization checks.
Recommendation	Authorization checks should be added to `claimFees` if per-epoch claiming is required.
Resolution	● ACKNOWLEDGED

Issue #52	Typographical issues and gas optimizations
Severity	● INFORMATIONAL

Description	Line 13
	
	`import './interfaces/IGaugeManager.sol';`
	
	Consider actually inheriting this such that the interface is guaranteed to be correct.
	
	Line 20
	
	`import './interfaces/IBribe.sol';`
	
	This is already imported.
	
	Line 26
	
	`import './libraries/Math.sol';`
	
	This library is unused and can be removed.
	
	Line 27
	
	`import "@openzeppelin/contracts-upgradeable/token/ERC20-/IERC20Upgradeable.sol";`
	
	`IERC20` was already imported, we see no reason to import this as well as the interface is identical.
	
	Lines 38 and 42

```
uint256 internal index;
mapping(address => uint256) internal supplyIndex;
```

Consider exposing a `pendingEmissions` function for these variables.

Line 39

```
address internal base;
```

This variable can be marked as `public` to allow inspection both off-chain and by other contracts.

Lines 43

```
mapping(address => uint256) public claimable; // gauge =>
claimable $the
```

This should say `$NOVA` instead.

Line 50

```
VoterFactoryLib.Data private _factoriesData;
```

This should have view functions as its currently an advanced structure without any introspection.

Lines 64-65

```
bytes32 public constant COMMUNITY_FEE_WITHDRAWER_ROLE =
keccak256('COMMUNITY_FEE_WITHDRAWER');
bytes32 public constant COMMUNITY_FEE_VAULT_ADMINISTRAT-
OR = keccak256('COMMUNITY_FEE_VAULT_ADMINISTRATOR');
```

It’s unclear to us why these permissions from the community fee contract are also exposed here.

Line 88

```
permissionRegistry = _permissionRegistory;
```

This should say “registry”.

Lines 95, 100 and 105

```
modifier GaugeAdmin() {
modifier Governance() {
modifier EpochManagerOrGaugeAdmin() {
```

Consider using the naming convention which is *snakeCase* and starts like *onlyGaugeAdmin*.

Lines 132-133

```
/// @notice Set a new Minter
function setGenesisManager(address _genesisManager)
external GaugeAdmin {
```

The comment above this function is wrong.

Line 165

```
uint poolLen = _pool.length;
```

This length is already used above this declaration. Consider declaring this at the top of the function and just using the local variable everywhere.

Line 177

```
function createGaugeWithBonusReward(address _pool,
uint256 _gaugeType, address bonusRewardToken)
```

Consider removing the `_gaugeType` parameter as this is only supposed to be for the CL gauge. Alternative consider validating that the `_gaugeType` is not zero.

Line 185

```
/// @dev To create stable/Volatile pair gaugeType = 0,
Concentrated liqudity = 1, ...
```

This should say "liquidity" instead.

Line 193

```
address bonusRewardToken = bonusRewardToken;
```

This seems rather redundant.

Line 225 and 261

```
if(_gaugeType == 1) {
```

Consider using `else if` instead.

Line 230

```
// approve spending for $the
```

This should say $NOVA instead.

Line 247

```
// todo: below line will go to ve33 rewarder.
```

Its unclear what this comment is about.

Line 273

```
bytes memory alphabet = "0123456789abcdef";
```

If desired, this can be marked as a constant `bytes16` to save gas.

Line 295

```
/// @dev the function is called by the minter each epoch.
Anyway anyone can top up some extra rewards.
```

We couldn’t find a way to top up extra rewards for the V2 gauges.

Lines 340-344

```
address communityVault = algebraPool.communityVault();
uint _balanceToken0 = IERC20(_token0).balanceOf(algebra-
Pool.communityVault());
...
uint _balanceToken1 = IERC20(_token1).balanceOf(algebra-
Pool.communityVault());
```

The `communityVault` variable should be re-used to save gas.

Line 371

```
/// @notice distribute reward onyl for given gauges
```

This should say “only” instead.

Line 417

```
/// @dev this function track the gauge index to emit the
correct $the amount after the distribution
```

This should say “tracks” and $NOVA instead.

Line 426

```
// SupplyIndex will be updated for Killed Gauges as well
so we don't need to udpate index while reviving gauge.
```

This should say “update” instead.

Line 462

```
claimable[_gauge] = 0;
```

It would be slightly more idiomatic to move this setter up a bit in line with checks-effects-interactions. However, since `NOVA` has no external interactions, there is no impact.

Line 475

```
require(isGauge[_gauge], 'DEAD');
```

This error message is wrong.

Line 488

```
require(_gauge.code.length > 0, "CODELEN");
```

This check is unnecessary, as it’s preceded by an `isGauge` check.

It should also be noted that most setters do a codelength check, but some setters don’t have this making it rather inconsistent.

Finally, `setVoter`, `setBlackGovernor` and `setAVM` lack events. Some other functions lack events as well but they seem to still have downstream events.

Recommendation	Consider fixing the typographical issues.
Resolution	● ACKNOWLEDGED

2.17 GaugeFactory

`GaugeFactory` is the responsible for deploying new `GaugeV2` contracts via its `createGauge` function, which can only be called by the configurable `GaugeManager` contract.

2.17.1 Privileged Functions

- `createGauge [ gauge manager ]`
- `setRegistry [ owner ]`
- `activateEmergencyMode [ emergency council ]`
- `stopEmergencyMode [ emergency council ]`
- `setDistribution [ owner or GAUGE_ADMINs ]`
- `setGaugeManager [ owner or GAUGE_ADMINs ]`

2.17.2 Issues & Recommendations

No issues found.

2.18 GaugeV2

`GaugeV2` is the main rewarder contract for all V2 (basic and stable) pairs ("LPs") within the system. Every epoch, the `GaugeManager` forwards a number of emissions to each LP's gauge that is proportional to the amount of voting escrow votes for the gauge.

These emissions are then distributed over the subsequent epoch to all LP stakers of the gauge, proportional to their portion of the total LP staked within the gauge.

Even though `GaugeV2` is a fork of the established Thena gauge, some changes were made.

An emergency mode can be enabled by the owner that prevents further deposits into the gauge and only allows the calling of `emergencyWithdraw`. This should be taken into careful consideration by vaults and other contracts building on top of the contracts.

Finally, the claimFees function can be called by anyone, which claims the underlying swap fees on the staked LP tokens. These fees are subsequently fully sent to the `internal_bribe` contract as compensation to the voters who voted that this gauge should receive emissions.

All rewards will be distributed by the time the epoch is finished. This means that between the time the new epoch starts and its rewards are distributed, a brief period without any rewards will exist. This has been communicated and it was confirmed that new epoch emissions will be distributed quickly.

2.18.1 Privileged Functions

- `setDistribution [ owner ]`
- `activateEmergencyMode [ owner ]`
- `stopEmergencyMode [ owner ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`

2.18.2 Issues & Recommendations

Issue #53	A small amount of rewardToken dust will accumulate in the gauge
Severity	● INFORMATIONAL
Description	The `GaugeV2` contract does various operations within its reward distribution and `notifyRewardAmount` which can contain a small rounding error. This rounding error accumulates as a reward token balance within the gauge which cannot be withdrawn.
Recommendation	Consider whether this is an issue. As long as the reward token has a small nominal value (a high number of decimals), the amount stuck within the gauge will be small enough to ignore.
Resolution	✓ RESOLVED The client has confirmed that this dust will never amount to any real value for the tokens they add, no changes were made.

Issue #54	The GaugeV2 does not support various special ERC-20 tokens such as fee-on-transfer tokens
Severity	● INFORMATIONAL
Description	The deposit function in GaugeV2 does not support fee-on-transfer tokens as it assumes the full amount requested is received by the gauge. This limitation could result in some users being unable to withdraw their full balances if such tokens were used for staking. Additionally, other special ERC-20 tokens, such as rebasing tokens, are not directly supported by the gauge. Regarding reward accumulation precision, the current 1e18 precision might cause issues for sub-tokens with extremely large or small supplies. However, this precision level is generally a reasonable trade-off and is commonly used. This issue is rated informational rather than low severity, since the gauges are primarily intended for LP tokens, which typically do not exhibit these special behaviors.
Recommendation	Consider whether there's any plan to ever add such tokens to a gauge. If so- consider using a before-after pattern within the deposits to support fee-on-transfer tokens. If other types of fringe tokens need to be supported, consider discussing this with us as we can recommend solutions tailored to the specific token type.
Resolution	✓ RESOLVED The client has indicated they will coordinate with the teams of the fee-on-transfer tokens to ensure that the SuperNova tokens are

whitelisted from the fee, before adding these tokens. No other special tokens will be added.

Issue #55	**Typographical issues and gas optimizations**
Severity	● INFORMATIONAL

Description

Line 15

```
interface IRewarder {
```

It's considered best practice to keep interfaces in separate solidity files and actually inherit from them. Currently this interface is not directly inherited within the `GaugeExtraRewarder`, which could lead for the interface to no longer match if changes are made.

Line 27

```
address public internal_bribe;
```

The `internal_bribe` can now be marked as `immutable`.

Line 35 and 39

```
address public VE;
```

This variable appear unused internally, and can likely be removed unless they are used by a dependent contract.

Line 41 and 217

```
mapping(address => uint256) public maturityTime;
require(block.timestamp >= maturityTime[msg.sender],
"!MATURE");
```

These sections are unused and can be removed.

Line 41 (old)

```
uint256 public DURATION;
```

This variable can be marked as `immutable` to save gas.

Lines 93, 111, 145 and 152

```
require(emergency == false, "EMER");
emergency = false;
require(emergency == false, "EMER");
equire(emergency == true,"EMER");
```

These statements are unnecessarily verbose. For example `require(emergency == true)` can be reduced to `require(emergency)`. The setter to false appears unnecessary as the variable is already `false` at the time of deployment. Finally, `require(emergency == false)` can be simplified to `require(!emergency)`.

Line 272

```
function _withdraw(uint256 amount) internal nonReentrant
isNotEmergency updateReward(msg.sender) {
```

This function is slightly inconsistent with `_deposit`, which defines an `account` address. Consider being consistent and either using `msg.sender` in both or in neither.

Line 274

```
require(_balanceOf(msg.sender) > 0, "ZV");
```

This check can be strengthened to `_balanceOf(msg.sender) >= amount` if desired. In its current state its a rather wasteful operation from a gas perspective without serving much purpose.

Line 334

```
function getReward(address _user) public nonReentrant
onlyDistribution updateReward(_user) {
```

`getReward` can be marked as `external` as its not used internally.

Line 384

```
rewardToken.safeTransferFrom(DISTRIBUTION, address(this-
), reward);
```

The origin of this transfer can be `msg.sender` to save a small amount of gas. This has the additional benefit that code reviewers will be more confident that this function cannot drain approvals without consent.

Line 399

```
require(rewardRate <= balance / DURATION, "REWARD_HIGH");
```

The comments above this line explaining it appear outdated, as the situation they explain is protected against with this requirement still appears possible. We assume that it's from a time where there wasn't a `transferFrom` within the function. Furthermore, this check is insufficient as an actual sanity check given that `balance` also incorporates user balances for an insufficient token balance.

Line 415

```
address _token = address(TOKEN);
```

It's unclear why this is cast back to `address`, given that within all uses this `_token` gets cast back to `IPair`.

Line 419-420

```
uint256 _fees0 = claimed0;
uint256 _fees1 = claimed1;
```

These variables appear unnecessary as `claimed0` and `claimed1` can just be used directly.

Lines 147 and 155

```
emit EmergencyActivated(address(this), block.timestamp-
);
emit EmergencyDeactivated(address(this), block.-
timestamp);
```

The arguments of these events are redundant with off-chain metadata always attached to events.

Finally, `setDistribution`, `setGaugeRewarder`, `setInternalBribe`, `setGenesisPool` and `setGenesisPoolManager` lack events.

Recommendation	Consider fixing the typographical issues and gas optimizations.
Resolution	● PARTIALLY RESOLVED

2.19 GaugeFactoryCL

`GaugeFactoryCL` is the contract responsible for deploying `GaugeCL` instances and is used by the `GaugeManager` to do so. When creating a gauge, the eternal farming virtual pool is also automatically created, which is the actual staking contract for the pool. Due to the farming plugin that is supposed to be connected on the underlying CL pools, the eternal farming system will be notified of any position changes.

2.19.1 Privileged Functions

- `activateEmergencyMode [ emergency council ]`
- `stopEmeregencyMode [ emergency council ]`
- `setDibs [ owner or GAUGE_ADMIN ]`
- `setReferralFee [ owner or GAUGE_ADMIN ]`
- `setGaugeManager [ owner or GAUGE_ADMIN ]`
- `setRegistry [ owner ]`
- `setAlgebraPoolApi [ owner ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`

2.19.2 Issues & Recommendations

Issue #56	Typographical issues
Severity	INFORMATIONAL

Description	Lines 19 and 25
	`interface IGaugeCL {`
	`interface ICustomPoolDeployer {`
	It is recommended to move these interfaces to a separate file, as this follows best practice for modularity and maintainability. Additionally, consider having the actual contract implementations explicitly inherit their corresponding interfaces. This ensures that the contracts fully comply with the declared interface specifications and helps catch discrepancies during compilation.
	Line 47
	`dibsPercentage = 0;`
	This appears unnecessary. Though it is no problem either.
	Lines 56 and 61
	`require(owner() == msg.sender, 'not owner');`
	Consider simply marking the function as `onlyOwner` instead, as is common practice.
	Line 72
	`return last_gauge;`
	This value is unnecessarily fetched from storage.
	Finally, all of the setters lack events.

Recommendation	Consider fixing the typographical issues.
Resolution	ACKNOWLEDGED

2.20 GaugeCL

GaugeCL is a contract similar to GaugeV2. However, unlike GaugeV2, it simply acts as an interface into Algebra's farming mechanism. Users are still free to fully bypass this gauge and directly interact with Algebra's FarmingCenter.

According to the team, GaugeCL mainly exists as a compatible interface with the other gauges, allowing dependency systems to easily interact with multiple types of gauges without having to learn about how Algebra's special farming system works. It is also useful for the emissions voting system, which can provide emissions to gauges in a unified manner.

GaugeCL is set as the communityFeeReceiver with the community fee set to 100%, allowing the fees to be routed to the bribes.

The fee percentages in this contract are set to a denominator of 1_000, which is different from the denominator of 10_000 used within the normal V2 pools. Care should be taken with this difference.

2.20.1 Privileged Functions

- `activateEmergencyMode`
- `stopEmergencyMode`
- `transferOwnership`
- `renounceOwnership`

2.20.2 Issues & Recommendations

Issue #57	Unstaked LP positions do not earn trading fees
Severity	● LOW SEVERITY
Description	Within the V2 pools, the trading fees accrued by the tokens staked in the gauge are distributed to the `veNOVA` voters of those pools. However, the fees for swaps of LP tokens which are not staked into the gauge, still go the liquidity provider who owns the LP tokens. This is very different with the `GaugeCL` and the SuperNova concentrated liquidity pools in general. For these pools, the fees *always* go to the voters, regardless of whether the LP that generated them is staked into the gauge or not. This is because within the SuperNova design, the fees are distributed through the `communityFee` which is set to `100%`. This means that all fees are essentially taken as a protocol fee, and then sent to the `GaugeCL` via the `CommunityVault`.
Recommendation	As this seems desired behavior, consider clearly documenting this as the frontend still currently has a section `Trading Fees` for the unstaked concentrated liquidity positions which simply appears to remain zero.
Resolution	✓ RESOLVED This is desired behavior.

2.21 BribeFactoryV3

`BribeFactoryV3` is responsible for deploying the `Bribe` contract instances for each gauge. During gauge creation within the `GaugeManager`, the bribe factory will be called to deploy the internal and external bribes for the gauge.

`BribeFactoryV3` also has some governance functionality to perform administrative tasks on the bribes, such as draining their reward tokens.

2.21.1 Privileged Functions

- `createBribe [ gaugeManager or owner ]`
- `setVoter [ owner ]`
- `setPermissionsRegistry [ owner ]`
- `setTokenHandler [ owner ]`
- `setBribeVoter [ owner ]`
- `setBribeMinter [ owner ]`
- `setBribeOwner [ owner ]`
- `recoverERC20From [ owner ]`
- `recoverERC20AndUpdateData [ owner ]`
- `transferOwnership`
- `renounceOwnership`

2.21.2 Issues & Recommendations

Issue #58	Typographical issues
Severity	● INFORMATIONAL

Description	Line 21
	`address[] internal _bribes;`
	This array should likely be marked as public for inspection purposes, alongside a length function.
	Line 25
	`address[] public defaultRewardToken;`
	This array appears fully unused.
	Lines 65-66
	`_bribes.push(last_bribe);`
	`return last_bribe;`
	The local `lastBribe` variable can be used instead to save gas.
	Lines 100 and 108
	`/// @notice set the bribe factory permission registry`
	This comment is incorrectly copy-pasted and does not actually describe the function in question.
	—
	Several functions lack events, though many of the underlying functions still emit events.

Recommendation	Consider fixing the typographical issues.
Resolution	● PARTIALLY RESOLVED

2.22 Bribe

The `Bribe` contract is linked twice to each gauge within the SuperNova system: an internal bribe and an external bribe. When voters vote for the gauge through `VoterV3`, the two bribes of that gauge are notified of the vote.

At the end of each epoch, voters can then claim any rewards which have accumulated within the `Bribe` contract. For the internal bribe, this is always the swap fees that were generated (all swap fees for CL pools and specifically the swap fees of the tokens staked in the gauge for the V2 pools), while for the external bribe this is any bribes which protocol bribes send into that contract. This allows token owners to persuade voters to vote on their protocol's token's pools by adding additional incentive alongside the swap fees for voting on that pool.

Similar to most of the other reward distributors, rewards can become stuck if no one actually stakes into the distributor (`Bribe`, `Gauge`, etc.). If the total number of stakers is zero, rewards remain permanently unclaimed as the only fallback mechanism is the owner of the Bribe taking them out.

Even though bribe tokens may be added over time, they can never be removed again. This appears to be by design.

2.22.1 Privileged Functions

- `recoverERC20AndUpdateData [ owner or bribeFactory ]`
- `emergencyRecoverERC20 [ owner or bribeFactory ]`
- `setVoter [ owner or bribeFactory ]`
- `setGaugeManager [ owner or bribeFactory ]`
- `setMinter [ owner or bribeFactory ]`
- `setOwner [ owner or bribeFactory ]`

2.22.2 Issues & Recommendations

Issue #59	Bribe reward claiming will erroneously send the reward to the AVM instead of the actual NFT owner if the NFT is owned by the AVM
Severity	● MEDIUM SEVERITY

Description	Lines 266-271
	
	`function getReward(uint256 tokenId, address[] memory tokens) external nonReentrant {`
	`  address _owner = IVotingEscrow(ve).ownerOf(tokenId);`
	`  if(IAutoVotingEscrowManager(avm).tokenIdToAVMId(tokenId) > 0) {`
	`    uint idx = IAutoVotingEscrowManager(avm).tokenIdToAVMId(tokenId)-1;`
	`    IAutoVotingEscrow[] memory avmList = IAutoVotingEscrowManager(avm).getAVMs();`
	`    _owner = address(avmList[idx]);`
	
	The `getReward` function has a special exception that allows for claiming rewards for NFTs which are owned by the auto-voting mechanism. In this special exception, the rewards should be sent to the original NFT owner, and not the AVM which currently owns the NFT.
	
	However, due to an error in the refactoring of this exception logic, the reward is still sent to the auto voting contract instead of to the owner.

Recommendation	We do not believe there is any need to figure out the original AVM contract. Instead, it is sufficient to call the following on the global AVM after confirming that the token is owned via the `tokenIdToAvmId(tokenId) > 0` check. `_owner = IAutoVotingEscrowManager(avm).getOriginalOwner(tokenId);`
Resolution	✓ RESOLVED AVMs have been fully removed.

Issue #60	Tokens with a fee on transfer are not supported as bribe rewards
Severity	● LOW SEVERITY
Description	The `notifyRewardAmount` function will operate incorrectly and add too many rewards compared to what the contract receives if the token has a fee on transfer.
Recommendation	Consider whether this is an issue; if not, consider documenting that tokens with a fee on transfer should whitelist the bribe and all other contracts (as most of the codebase does not support these tokens). If it is an issue, consider using a before-after pattern.
Resolution	✓ RESOLVED The client has indicated they will coordinate with the teams of these tokens to ensure that the SuperNova tokens are whitelisted from the fee, before adding these tokens.

Issue #61	The contract does not support a ve token with a supply larger than 2**128
Severity	● INFORMATIONAL
Description	Line 238 (example) `votingSupplyPlots[nPlots - 1] = VotingSupplyPlot(uint128(totalSupply), ts);` Throughout the contract, the total amount of votes is cast down to `uint128`. This will cause the contract to break if this variable is ever exceeded. This issue is raised informationally as many other contracts break once a token has a supply higher than 128 bits. Furthermore, the SuperNova token currently has a supply significantly lower than this number.
Recommendation	Consider using a larger type for the token values. Given that this contract is deployed on a cheap network we recommend using a full uint256 for both the timestamp and the token balances.
Resolution	✓ RESOLVED The client has indicated they will not have such a token.

Issue #62	Contract does not support reward tokens with a very high supply
Severity	● INFORMATIONAL
Description	Line 122 `reward += (cp0.balanceOf * tokenRewardsPerEpoch[_reward-Token][_currTs]) / _supply;` This line of code may fail due to overflow if the reward is very high. This appears rather unlikely to us as not many such tokens exist.
Recommendation	Consider failing early on `notifyRewardAmount` instead, this avoids the unhappy realisation that a bribe is not claimable.
Resolution	● ACKNOWLEDGED

Issue #63	Typographical issues
Severity	● INFORMATIONAL
Description	Line 112 `uint256 _currTs = BlackTimeLibrary.epochStart(lastEarn[-_rewardToken][tokenId]);` Since the `lastEarn` now always appears to be aligned to the `epochStart`, this extra operation seems unnecessary.
Recommendation	Consider fixing the typographical issues.
Resolution	● ACKNOWLEDGED

2.23 CustomPoolDeployer

`CustomPoolDeployer` is responsible for deploying new Algebra pools within the SuperNova system. Only pools deployed by it will be supported within the SuperNova system. If pools are created directly without using this deployer, they should not be listed on the SuperNova website, nor be used for gauge rewards.

`CustomPoolDeployer` is upgradeable, meaning that the proxy owner can fully change its functionality. the proxy admin should be safeguarded extremely carefully, ideally behind a secure multi-signature wallet consisting of trusted, independent parties.

2.23.1 Privileged Functions

- `createCustomPool [ owner or authorizedAccounts ]`
- `setPluginForPool [ owner or authorizedAccounts ]`
- `setPlugin [ owner or authorizedAccounts ]`
- `setPluginConfig [ owner or authorizedAccounts ]`
- `setFee [ owner or authorizedAccounts ]`
- `setCommunityFee [ owner or authorizedAccounts ]`
- `setAlgebraFeeRecipient [ owner ]`
- `setAlgebraFeeManager [ owner ]`
- `setAlgebraFeeShare [ owner ]`
- `setAlgebraFarmingProxyPluginFactory [ owner ]`
- `setAlgebraFactory [ owner ]`
- `setAlgebraPluginFactory [ owner ]`
- `addAuthorizedAccount [ owner ]`
- `removeAuthorizedAccount [ owner ]`
- `transferOwnership [ owner ]`
- `renounceOwnership [ owner ]`

2.23.2 Issues & Recommendations

No issues found.

2.24 PermissionsRegistry

`PermissionsRegistry` is the main role-based access-control contract for the system. It defines many of the governance roles and allows for the team to assign these roles to accounts. These accounts can then call various governance functions throughout the system.

2.24.1 Privileged Functions

- `addRole [ blackMultisig ]`
- `removeRole [ blackMultisig ]`
- `setRoleFor [ blackMultisig ]`
- `removeRoleFrom [ blackMultisig ]`
- `setEmergencyCouncil [ blackMultisig or emergencyCouncil ]`
- `setBlackTeamMultisig [ blackTeamMultisig ]`
- `setBlackMultisig [ blackMultisig ]`

2.24.2 Issues & Recommendations

No issues found.

2.25 TokenHandler

`TokenHandler` is a central registry which keeps track of whitelisted tokens, `veNFT` ids and whitelisted connector tokens. It also keep track of various other things. Small changes to the governance roles were made.

2.25.1 Privileged Functions

- `whitelistTokens [ GOVERNANCE ]`
- `whitelistToken [ GOVERNANCE ]`
- `blacklistTokens [ GOVERNANCE ]`
- `blackListToken [ GOVERNANCE ]`
- `whitelistNFT [ GOVERNANCE ]`
- `blacklistNFT [ GOVERNANCE ]`
- `whitelistConnectors [ GOVERNANCE ]`
- `whitelistConnector [ GOVERNANCE ]`
- `blacklistConnector [ GOVERNANCE ]`
- `setBucketType [ GOVERNANCE ]`
- `updateTokenVolatilityBucket [ GOVERNANCE ]`
- `setPermissionsRegistry [ OWNER ]`
- `transferOwnership [ OWNER ]`
- `renounceOwnership [ OWNER ]`

2.25.2 Issues & Recommendations

No issues found.

2.26 BlackTimeLibrary

`BlackTimeLibrary` is a simple shared utility used by several contracts.

The current epoch length is set to 7 days and is aligned to UNIX timestamps.

The only changes were the removal of various functions. No issues were found with this removal.

2.26.1 Privileged Functions

None.

2.26.2 Issues & Recommendations

No issues found.

2.27 BlackholePairAPIV2

`BlackholePairAPIV2` is a utility contract used for the frontend. It provides several utility functions to display information to the frontend.

The only changes to this contract were removals of functionality that are no longer present. No issues were found with these removals.

2.27.1 Privileged Functions

- `setOwner`
- `setVoter`
- `setGaugeManager`
- `setAlgebraFactory`
- `setQuoterV2`
- `setAlgebraPoolAPI`
- `setPairFactory`

2.27.2 Issues & Recommendations

No issues found.

2.28 veNFTAPI

`veNFTAPI` is a utility contract used for the frontend. It provides several utility functions to display information to the frontend.

The final version of this contract where the AVM was already removed was audited, which is a later version compared to the initial commit from the preliminary commit.

2.28.1 Privileged Functions

- `setOwner`
- `setVoter`
- `setGaugeManager`
- `setGaugeFactory`
- `setGaugeFactoryCL`
- `setRewardDistro`
- `setPairAPI`
- `setPairFactory`

2.28.2 Issues & Recommendations

Issue #64	Typographical issues
Severity	● INFORMATIONAL
Description	`hasVotedForEpoch` may not always be set to true — e.g., for `poke` it will not be. Also, it will be set to true if `vote` is called with an empty array or with amounts rounding to zero. In this case no real voting occurs but it still gets set to zero. It is not necessary to fix this as this is a frontend contract. — `setGaugeFactoryCL` lacks an event. All unchanged setters lack one as well.
Recommendation	Consider fixing the typographical issues.
Resolution	● ACKNOWLEDGED

2.29 Math

The `Math` library provides the `min`, `max` and `sqrt` functions which are used by various contracts. The `min` and `max` functions are trivially correct, while the `sqrt` function is a direct port of the `Uniswap V2` `sqrt` math function, with the most notable change being that it is now defined in `0.8` meaning that any overflow or divisions are now checked for overflow and division by zero, consuming slightly more gas.

The library also defines `cbrt` but this function is fully unused.

We recommend users of this library to keep the rounding behavior in mind for these two root functions.

2.29.1 Privileged Functions

None.

2.29.2 Issues & Recommendations

No issues found.

2.30 AlgebraVaultFactory

The standard `AlgebraVaultFactoryStub` is replaced with a custom factory. The original factory simply returns the same community vault for all newly deployed pools. Due to the way SuperNova works, the team wants to have individual vaults for each pair. This allows them to accurately track fees and send them to the gauges.

2.30.1 Privileged Functions

- `setOwner`

2.30.2 Issues & Recommendations

No issues found.

2.31 CustomPluginV1Factory and CustomPluginV2Factory

The client has extended the V1 and V2 plugin factories of Algebra with a simple extension contract that allows for deploying a plugin to existing pools. This was needed to easily retrofit the existing Algebra plugins and their factories in the custom deployer of the client.

2.31.1 Privileged Functions

- `createPluginForExistingCustomPool [ customPoolDeployer or POOLS_ADMINISTRATION_ROLE ]`
- `(base privileged functions)`

2.31.2 Issues & Recommendations

Issue #65	Typographical issues
Severity	● INFORMATIONAL
Description	Line 15 `require(msg.sender == customPoolDeployer || factory.hasRoleOrOwner(factory.POOLS_ADMINISTRATOR_ROLE(), msg.sender), 'Only deployer or admin');` Consider storing the role as a constant instead.
Recommendation	Consider fixing the typographical issues.
Resolution	● ACKNOWLEDGED

2.32 AlgebraBasePluginV3

The `AlgebraBasePluginV3` is a slight rewrite of the `AlgebraBasePluginV1-`
, integrating the `SecurityPlugin`. It hooks into the security plugin for swaps, flashloans, liquidity additions and liquidity removals. This allows for the team to disable all of these four actions, or only allow for liquidity removals.

2.32.1 Privileged Functions

- `setSecurityRegistry [ plugin factory or ALGEBRA_BASE_PLUGIN_MANAGER ]`
- `changeFeeConfiguration [ ALGEBRA_BASE_PLUGIN_MANAGER ]`

2.32.2 Issues & Recommendations

Issue #66	Fee collection cannot be paused
Severity	● INFORMATIONAL
Description	Many of the user-callable pool functions can now be paused in emergencies by the team through the integration of the `SecurityPlugin`. However, no such pausing is possible for fee claiming by users. If the team wishes to fully disable pools including fee claiming, this is not possible.
Recommendation	Consider also preventing fee claiming when the pool status is set to `DISABLED`.
Resolution	● ACKNOWLEDGED The team has indicated that fee claiming is a low risk path. Furthermore, they have indicated that they'd rather not make intrusive changes to Algebra, which would be required here. We agree with them that this would indeed require an intrusive change and understand that that is not desired.

2.33 BasePluginV3Factory

The `BasePluginV3Factory` is a modified version of the `BasePluginV1Factory` allowing for the configuration of a `SecurityRegistry` alongside with a new function to deploy plugins for existing pools.

The plugins can be deployed by and for any pool deployer, but only for their own pools.

2.33.1 Privileged Functions

- `setDefaultFeeConfiguration [ ALGEBRA_BASE_PLUGIN_FACTORY_ADMINISTRATOR ]`
- `setFarmingAddress [ ALGEBRA_BASE_PLUGIN_FACTORY_ADMINISTRATOR ]`
- `setSecurityRegistry [ ALGEBRA_BASE_PLUGIN_FACTORY_ADMINISTRATOR ]`

2.33.2 Issues & Recommendations

No issues found.

2.34 SecurityPlugin

The `SecurityPlugin` is an Algebra plugin written by the Algebra team in their safety-switch extension. It was added to this scope during the resolution rounds of this audit to increase the security of the Algebra stack, allowing the team to pause swaps, liquidity addition and liquidity removal.

The plugin exposes two important internal functions to the main plugin, namely `_checkStatus()` and `_checkStatusOnBurn()`. The first check will revert if the configured `SecurityRegistry` has the pool status set to anything other than `ENABLED`. The secondary check will still succeed even if the status is set to `BURN_ONLY`.

The `SecurityPlugin` can therefore not be used by itself and should be integrated in a larger parent plugin, in this case the `AlgebraBasePluginV3`.

2.34.1 Privileged Functions

- `setSecurityRegistry [ plugin factory or ALGEBRA_BASE_PLUGIN_MANAGER ]`

2.34.2 Issues & Recommendations

No issues found.

2.35 SecurityRegistry

The `SecurityRegistry` is the central registry for consulting the pool status used by all the `SecurityPlugin`s. The algebra factory owner can disable all pools at once by setting the global status to `DISABLED` or set all pools to `BURN_ONLY` at once, which only allows for liquidity removal.

Individual pools can still be overwritten to `DISABLED` or `BURN_ONLY` via the `setPoolsStatuses` function. These overwritten statuses will only be used if the global status is set to `ENABLED`. This means that as soon as the global status changes from `ENABLED`, all individual overwritten statuses are ignored.

2.35.1 Privileged Functions

- `setPoolsStatuses [ owner for ENABLED and BURN_ONLY, GUARD for DISABLED ]`
- `setGlobalStatus [ owner for ENABLED and BURN_ONLY, GUARD for DISABLED ]`

2.35.2 Issues & Recommendations

Issue #67	**setPoolsStatus can be called by anyone if an empty pools array is provided, increasing the contract's attack surface**
Severity	● INFORMATIONAL
Description	The access control within the `setPoolsStatus` triggers on each iteration over the `pools` array provided into the function. This means that if an empty `pools` array is provided, no access control occurs and anyone can call the `setPoolsStatus` function with such a parameter. The `newStatuses` parameter can still be non-zero length. This issue is raised as informational since no state changes are expected to occur when a malicious actor calls this function with an empty `pools` array. However, it needlessly increases the attack surface of the contract. If a compiler issue ever gets discovered in solidity, leaving functions like this open increases the risk that this contract can be affected by it, especially since attackers can still choose `newStatuses` freely.
Recommendation	Consider revamping `_hasAccess` to exactly check the authorization once, based on the maximum privilege required by the `newStatuses` array. Make sure that both arrays are enforced to be equal length as well. It could finally be considered to allow the `owner` to also be able to set the status to `DISABLED`, as right now this is only permitted if they have the `GUARD` role.
Resolution	● ACKNOWLEDGED Given that this code is forked the client understandably wishes to keep it unchanged.

Issue #68	Typographical issues
Severity	● INFORMATIONAL

Description	Line 15
	
	`EnumerableSet.AddressSet private overriddenPools;`
	
	This enumerable set lacks view functions. This prevents other contracts from using this data. As this set may only serve an internal purpose, keeping it private may also be desired.
	
	Lines 51-52
	
	`bool _isPoolStatusOverrided = isPoolStatusOverrided;`
	`if (_isPoolStatusOverrided) {`
	
	It's unclear to us why this exception case is added as it does not seem to save significant gas. The shortcut it makes still requires a read from storage and the outcome it leads to seems identical to the outcome without such an exception case. We are also unsure why the storage variable gets cached into `_isPoolStatusOverrided` even though it's only used once on the next line.

Recommendation	Consider fixing the typographical issues.
Resolution	● ACKNOWLEDGED Given that this code is forked the client understandably wishes to keep it unchanged.