Velodrome govNFT

Security Review

Cantina Managed review by:

D-Nice, Lead Security Researcher

Kaden, Security Researcher

Brian McMichael, Security Researcher

June 14, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Velodrome Finance is a next-generation AMM that combines the best of Curve, Convex and Uniswap, designed to serve as Optimism's central liquidity hub. Velodrome NFTs vote on token emissions and receive incentives and fees generated by the protocol.

From Apr 3rd to Apr 8th the Cantina team conducted a review of contracts on commit hash b6d27740. The team identified a total of **22** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 3
- Low Risk: 7
- Gas Optimizations: 4
- Informational: 8

3 Findings

3.1 Medium Risk

3.1.1 `GovNFTFactory`/`GovNFTTImelockFactory.govNFTs()` will fail if registry values grow.

Severity: Medium Risk

Context: GovNFTFactory.sol#L74, GovNFTTimelockFactory.sol#L79

Description: The factory getter for deployed NFT's returns the array of deployed addresses. These contracts are expected to deploy many GovNFT's and this getter will begin to fail on gas limitations when returning an array of 100-200 values or more. The registry values are `internal` so there is no other available accessor to iterate these values after that point. Without this available integrators will need to catalogue creation events to access this information.

Recommendation: Add accessor functions to get individual addresses or a partial array of the registry values. Consider functions like `function govNFTsByIndex(uint256 _index)` and/or `function govNFTs(uint256 _startIndex, uint256 _endIndex)` which will allow an integrator to call `govNFTsLength()` and make a partial iteration over the set.

Velodrome: Applied the recommended fix:

- Add paginated gettter `govNFTs(start, end)`.
- Add getter by index `govNFTs(index)`.

See commit 1d142bff.

Cantina Managed: Fix confirmed as of commit 75d06900, which includes slightly more optimized code.

3.1.2 Split finalization succeeds with non-existent split proposals

Severity: Medium Risk

Context: GovNFTTimelock.sol#L70

Description: `GovNFTTimelock.finalizeSplit` includes a check to enforce that a `timelock` has passed since a split has been committed before creating the split.

```
if (block.timestamp < splitProposal.timestamp + timelock) revert SplitTooSoon();
```

However, it's possible that there exists no `splitProposal` for `_proposedSplits[_from]`, in which case `splitProposal.timestamp == 0`. In this case, the check will succeed since `block.timestamp` will always be greater than `0 + timelock`. As a result, the full function execution will succeed unexpectedly. We can validate this by adding the following test to `FinalizeSplit.t.sol`:

```
function test_finalizeNonExistentSplit() public {
    vm.startPrank(address(recipient));
    // We skip the timelock delay because the block.timestamp is 0 in this environment
    // In a real environment we would already have a block.timestamp > timelockDelay
    skip(timelockDelay);
    vm.expectEmit(false, false, false, true, address(govNFTLock));
    emit IERC4906.MetadataUpdate(from);
    govNFTLock.finalizeSplit(from);
    vm.stopPrank();
}
```

As we can see from running the test, execution succeeds, running all the relevant logic for splitting, without any splits actually being created. This costs the user significant amounts of gas to update `_parentLock` storage and may lead to unexpected side effects.

Recommendation: Revert if `splitProposal.timestamp` is 0 in addition to reverting if the timelock has yet to pass, e.g.:

```
- if (block.timestamp < splitProposal.timestamp + timelock) revert SplitTooSoon();
+ if (block.timestamp < splitProposal.timestamp + timelock || splitProposal.timestamp == 0) revert
↪  SplitTooSoon();
```

Additionally, consider reverting in `GovNFT._split` if `_paramsList.length` is 0.

Velodrome: The issue has been acknowledged and fixed. This was fixed by the timelock protection refactoring (see commit 9fc40e72). There is no commit/finalize mechanism anymore and the `_split` function enforces the necessary checks.

Cantina Managed: The issue has been fixed by removing the commit/finalize mechanism.

3.1.3 Current frontrun protection of `GovNFTTimelock` works only under limited conditions

Severity: Medium Risk

Context: GovNFTTimelock.sol#L35-L103

Description: The `GovNFTTimelock` contract variant is intended to provide an extended contract which has a degree of frontrun protection of its value, to provide the ability to sell it safely on a marketplace and increase utility of these contracts.

The frontrun protection currently implemented, only provides a guarantee of frontrun protection to buyers that inspect the Lock NFT put up for sale on a marketplace, and ensure that its corresponding ID, has had no `CommitSplit` events emitted. If a split has been committed to by the owner of the Lock NFT prior to listing it for sale, and the `timelock` for it exceeded, the owner can still frontrun up to the entire locked value.

Therefore, buyers are required to be vigilant with the current implementation and check for the state of the Lock, and discount any from the marketplace not meeting that criteria.

Proof of concept *(pluggable into `FinalizeSplit` test)*:

```
function test_FrontrunViaPreCommit() public {
    skip(WEEK + 2 days); // skip somewhere after cliff ends

    IGovNFT.Lock memory lock = govNFT.locks(from);

    IGovNFT.SplitParams] memory paramsList = new IGovNFT.SplitParams[;
    paramsList[0] = IGovNFT.SplitParams({
        beneficiary: address(recipient2),
        amount: amount,
        start: uint40(block.timestamp),
        end: lock.end,
        cliff: 0,
        description: ""
    });
    vm.startPrank(address(recipient));
    govNFTLock.commitSplit(from, paramsList);

    skip(timelockDelay);

    // recipient mock lists NFT on marketplace
    TestOwner NFTMarket = new TestOwner();
    TestOwner buyer = new TestOwner();
    // NFT marketplaces receives sell order, and gets approval to give it to buyer
    govNFTLock.approve(address(NFTMarket), from);

    // buyer sees vault value, and adds txn to mempool to buy it
    uint256 buyerExpectedVault = IERC20(testToken).balanceOf(lock.vault);
    // recipient frontruns it via the pre-commited split
    uint256 tokenId = govNFTLock.finalizeSplit(from)[0];
    vm.stopPrank();
    // buyers transaction confirms... they receive vault, but it's less due to split
    vm.prank(address(NFTMarket));
    govNFTLock.safeTransferFrom(address(recipient), address(buyer), from);


    assertEq(buyerExpectedVault, IERC20(testToken).balanceOf(lock.vault), 'buyer did not receive expected
↪  vault value at NFT market listing');
    assertLt(buyerExpectedVault - amount, IERC20(testToken).balanceOf(lock.vault), 'Vault should not be
↪  splittable/drainable right when listed');//'NFT sale was still frontrun by committing to split before
↪  listing');
}
```

Recommendation: Strengthen the frontrun guarantees by requiring a Lock NFT owner to `commit` to a "*freezing*" of functionality for a transfer prior to a transfer being able to complete. In this case, the func-

tionality frozen would be `split` or any other functionality available to owners/approved parties of the NFT. Once in this committed/frozen state, after a timelock period ends, for safety, transfers will be enabled for that NFT, this would guarantee that no frontruns can occur, and not require vigilance on a buyer's part, as any NFT not in this state, would revert during the buying process.

There are a few options for implementing this. The likeliest best candidate would be overriding `_update` and adding a check that the NFT is in a committed state for transfers, and the `timelock` of the GovNFT for it has passed, or revert. And any functionality that should be frozen during the `commit` state, should revert if attempted to be accessed. Care will need to be taken to account for non-transfer cases, such as burn and mint. One potential downside of this is that owners may list NFTs, not committed, that will keep failing, but its committed state could be communicated via the `ArtProxy` as well. Another con is that all transfers for these types of contracts would be delayed by the timelock, but the benefit is it guarantees a specific value of the Lock on commit.

Velodrome: Applied the recommended fix. The GovNFTTimelock has been refactored to include freeze/unfreeze functions and corresponding locking mechanisms. The functionality is better described in the project's `SPECIFICATION.md`. See commit 9fc40e72. The freeze/unfreeze was added to the "full split" in commit db9e856a.

Cantina Managed: Recommendation has been followed and fix confirmed.

3.2 Low Risk

3.2.1 `parentLock` that does a split before its cliff adversely alters its vesting schedule

Severity: Low Risk

Context: GovNFT.sol#L255-L259

Description: When a `parentLock` splits prior to passing its cliff, the splitting logic fails to account any virtually vested amounts that may not be claimable yet, but should still be accruing and claimable once the cliff is passed. This effectively alters the vesting schedule of the updated `parentLock` to one that will always be worse in payout until convergence at the end date.

The following graph showcases this, with `f1` denoting an initial vesting schedule, and `f2` denoting how the vesting schedule looks after a pre-cliff split. `f2` has an increased slope, but discards any accounting of the initial cliff vestment, resulting in worse payouts throughout the lifetime.

Recommendation: Under this design, `Lock` owners should avoid splits until the cliff is met.

This design can be fixed by not altering the `parentLock` times within `_split` in the instance its cliff has not been met yet. This should be safe as in such instances`totalClaimed` and `unclaimedBeforeSplit` should be 0. It may be worthwhile to assert these invariants.

Velodrome: Applied the recommended fix. Only update timestamps when splitting after the cliff period end, i.e. the if condition update from above. See commit 98bd5469.

Cantina Managed: Fix confirmed.

3.2.2 `parentLock` with commitless `finalizeSplit` bug can be bricked under exceptional circumstance

Severity: Low Risk

Context: GovNFT.sol#L257, GovNFT.sol#L82, GovNFT.sol#L313-L332

Description: By abusing the `finalizeSplit` bug from "Split finalization succeeds with non-existent split proposals" which skips validity checks, if a `parentLock` split happens to occur on the exact second of its end timestamp, its `Lock` struct will have the same `start` and `end` value. This will result in core functionality associated with that NFT bricking, such as `claim` and `split` due to `_totalVested` throwing from a divide by zero error. Effectively causing any unclaimed amounts in the NFT's vault to be lost.

This should be rare in practice, as it requires a split to be done essentially at a Lock's end of life. However, this may occur in the wild with a combination of bad luck and network congestion, where a split transaction is not confirmed until this prerequisite happens to be met, which may be further exacerbated by MEV bots.

This can also result in instances of a `parentLock` whose `start` exceeds its `end`, which doesn't lead to division by zero exceptions, but is still invalid. It is saved by the following early return condition GovNFT.sol#L79-L82 to return 0, instead of underflow which would result in a brick for these cases as well leading to a greater likelihood.

Recommendation: Upon update of the `parentLock` parameters in `_split`, the invariant checks from `_createLockChecks` should also be applied on the `parentLock`. In the above case, either the `ZeroAmount` or `InvalidParameters` error should trigger, depending on their respective order. This would act as a defense-in-depth mechanism in case any further such bugs could exist that allow for circumvention of validity checks. Fixing "Split finalization succeeds with non-existent split proposals" should also resolve this.

Velodrome: The issue has been acknowledged and fixed. This was fixed by the timelock protection refactoring (see commit 9fc40e72). There is no commit/finalize mechanism anymore and the _split function enforces the necessary checks

Cantina Managed: Fix confirmed, all codepaths with respect to split functionality will run through `_createLockChecks` as required.

3.2.3 `ArtProxy` depends on unmaintained `Base64` library

Severity: Low Risk

Context: ArtProxy.sol#L10, .gitmodules#L7-L9

Description: As a preface, the `ArtProxy` contract was not in the scope of the audit, hence no exhaustive examination of it has been completed, but this finding affecting it that was noticed is still being documented.

There is a dependency for a `Base64` library that is not longer maintained, and has outstanding issues filed on its repository that have not been addressed, such as lacking special characters support and incompatibility with certain L2 EVMs. There is also a security advisory likely affecting the library.

Recommendation: The imported OpenZeppelin contracts provide their own `Base64` library based off the one by Brechtpd, but it is maintained and more featureful. Consider dropping this additional dependency and utilize the one already available, but ensure the OZ dependencies are updated to at least v5.0.2, where the fix for GHSA-9vx6-7xxf-x967 is implemented.

Velodrome: Applied the recommended fix:

- Update oz to 5.0.2.
- Use OZ's base64 lib.
- Remove the unmaintained base64 lib.

See commit e607f22f.

Cantina Managed: Fix confirmed.

3.2.4 Ambiguous Lock NFT valuation due to frontrun & arbitrage opportunities on `claim` and `sweep`

Severity: Low Risk

Context: GovNFTTimelock.sol#L5-L14, GovNFT.sol#L137-L191

Description: This issue is specific to `GovNFTTimelock` contracts which are intended to provide mechanisms to make the NFTs safely placeable on marketsplaces for buyers. The current design only intends to try and guarantee the `totalLocked` amount for a specific NFT at its exact time of sale.

In reality, buyers are not just buying the `totalLocked`, but the entire associated value of the `Vault` connected to the Lock NFT. This associated valuation is ambiguous, as any accrued claim value could be frontrun via `claim` by either the seller/approved or even arbitraged by the approved NFT marketplace. The same could happen for any airdropped tokens via `sweep`. Also, a certain value could be placed on the delegation of tokens, although most delegations should be reversible, while the prior value transfers are not.

An additional problem with relying only on `totalLocked` as the valuation basis is that once the `cliff` for it passes, its value decreases linearly over time. This would mean the longer an NFT is listed, the less its

nominal value is, possibly requiring multiple instances of relisting from the seller. The balance of a `Vault`, on the other hand, can stay constant for the duration of a marketplace listing.

In summary, the chosen basis for value makes it exceptionally hard for the market to place an appropriate and safe pricing model on the NFTs as there are still multiple existential frontrun & arbitrage risks that are not safeguarded against. In the case of no such events occurring, buyers can come out on top for getting much greater value of the NFT than just their locked tokens, but in other cases buyers may pay a heavy premium in hopes of getting this associated `Vault` value, but it gets frontrun/arbitraged out, and they lose (although more sophisticated buyers could implement their own checks against such frontruns via their own smart contracts that check the value before finalizing their bid).

Recommendation: Instead of relying on only `totalLocked` as the valuation mechanism, consider the `commit` recommendation in the issue "Current frontrun protection of `GovNFTTimelock` works only under limited conditions", and additionally apply that recommendation upon a `commit` to freeze `claim` and `sweep` and possibly cancel delegations, in preparation for a transfer/market listing. This should have the effect of keeping the value of the associated `Vault` at least what it was at market listing, maybe higher in case of airdrops or forced send mechanisms. And in turn, would transfer the valuation to be much more unambiguous and constant, and easier to price for buyers and sellers.

Velodrome: Applied the recommended fix. Applied timelock protection by using freeze/unfreeze on gov-nft functionality (claim, sweep, split) in commit 9fc40e72. The freeze/unfreeze was added to the "full split" in commit db9e856a.

Cantina Managed: Fix confirmed, for cases where the contracts are deployed with sufficiently high time-lock freeze.

3.2.5 Vault logic may not be sufficient to claim airdrops.

Severity: Low Risk

Context: Vault.sol#L35

Description: One reason noted for creating an external contract address as a vault was to be able to accept airdrops that could be `sweep()`ed by the owner to another address. The vault may offer limited functionality for this behavior since most airdrops require the owning address to perform some affirmative action as the owner address in order to claim the airdrop. The current vault structure assumes that tokens will be externally transferred to the vault address whereby they can be swept elsewhere. Since there is no additional functionality to make an arbitrary call to another contract to claim tokens with this address, some airdrops may be unclaimable by the owner.

Recommendation: This is a tricky one. Adding arbitrary claim functionality would have significant security implications and severely undermine the ability of the contract to enforce the lock. It may just need to be accepted that claimable airdrop tokens cannot be retrieved by this contract and documented to that effect.

Velodrome: Applied the recommended fix:

- New split function that "splits the whole amount".
- Effectively creates a new vault.
- Transfers lock's funds to new vault.
- Transfers ownership of old vault to lock recipient.
- Add arbitrary execution function restricted by `onlyOwner`.

See commit c7bd8981.

Cantina Managed: Fix confirmed as of commit db9e856a, which includes freeze protection on the new functionality. Additionally there are caveats where existing Locks should never be topped up with a respective Lock token, allowing for these to be potentially withdraw. The client considers topping up of existing Locks to be an anti-pattern and not something users should do.

3.2.6 Delegation is not reset when transferring lock tokens

Severity: Low Risk

Context: GovNFT.sol#L158-163

Description: GovNFT lock tokens include functionality to delegate voting power for ERC5805 compliant tokens which are held in the associated vault.

```
function delegate(uint256 _tokenId, address _delegatee) external nonReentrant {
    _checkAuthorized({owner: _ownerOf(_tokenId), spender: msg.sender, tokenId: _tokenId});

    IVault(_locks[_tokenId].vault).delegate(_delegatee);
    emit Delegate({tokenId: _tokenId, delegate: _delegatee});
}
```

ERC5805 connects delegatees to delegators via the **account** delegating, see `delegate` function spec:

> This function changes the caller's delegate, updating the vote delegation in the meantime.

This means that when ERC5805 tokens are transferred, the delegated voting power is also reset.

In the case of the lock tokens being transferred, while the ownership of the held ERC20 tokens is effectively transferred, the actual holder of the tokens, the vault, remains consistent. As a result, delegated voting power is not reset as expected. This results in the recipient of the lock token unintentionally delegating to the previous delegatee.

Recommendation: According to ERC5805, it's possible to delegate to no one: "_Each user account (address) can delegate to an account of its choice. This can be itself, someone else, or no one (represented by address(0))_". This is also the default case when transferring to an account which has not yet delegated. As such, a reasonable design choice would be to delegate the vault token to `address(0)` on transfer of the lock by overriding the ERC721 `_update` function to call `vault.delegate` with `address(0)` as the `_delegatee`.

Velodrome: The issue has been acknowledged and will not be fixed. The delegate function can be called by the new owner to choose a delegatee. Keeping this behaviour is also advantageous for a DAO (lock's recipient) that want to keep the same delegatee in a case where split or transfers are being done through several DAO members.

Cantina Managed: Acknowledged.

3.2.7 Split lock vesting schedules may change as a result of the delay between committing and finalizing

Severity: Low Risk

Context: GovNFTTimelock.sol#L80-L85

Description: In `GovNFTTimelock`, we impose a delay between committing and finalizing splits to be created for locks. As a result of this delay, it's possible that the initially committed and validated `start` timestamp was valid at the time of commitment, but is no longer valid at the time of finalization, i.e. it's in the past. In this circumstance, we update the split `start` to be the current `block.timestamp`, and adjust the `cliff` such that the cliff ends at the same time.

```
// Update Split Proposal's timestamps if proposed `start` is in past
if (block.timestamp > params.start) {
    splitCliffEnd = params.start + params.cliff;
    params.start = uint40(block.timestamp);
    params.cliff = uint40(block.timestamp < splitCliffEnd ? splitCliffEnd - block.timestamp : 0);
}
```

The amount that has vested for a given lock, `totalVested`, is calculated based on the following logic, effectively `totalLocked` applied to the current relative amount of time that has passed between `start` and `end`:

```
(_lock.totalLocked * (time - _lock.start)) / (_lock.end - _lock.start)
```

This means that by delaying the start timestamp, we're modifying the slope at which the lock vests at. As a result, users will not be able to claim as much of the lock as previously expected at any given time before the `end`. We can visualize this in Desmos.

Recommendation: Consider reverting in the case that `block.timestamp > params.start`, or otherwise adding documentation which indicates that the slope of split locks may change in this circumstance.

Velodrome: The issue has been acknowledged and fixed. This was fixed by the timelock protection refactoring (see commit 9fc40e72). There is no commit/finalize mechanism anymore. There is no delay between commit and finalizing splits since the splits are performed normally when it's done in an unfrozen state.

Cantina Managed: The issue has been fixed by removing the commit/finalize mechanism.

3.3 Gas Optimization

3.3.1 Additional if condition in `claim` for generally lower gas cost

Severity: Gas Optimization

Context: GovNFT.sol#L146-L151

Description: The current initial conditional

```
if (claimable > lock.unclaimedBeforeSplit) {
    lock.totalClaimed += claimable - lock.unclaimedBeforeSplit;
    delete lock.unclaimedBeforeSplit;
```

includes a number of operations associated with `lock.unclaimedBeforeSplit`, even though under most runs, it is likeliest to be 0, rendering them unneeded for those cases, and therefore introducing an avoidable gas overhead.

Recommendation: Consider refactoring the above portion to:

```
if (lock.unclaimedBeforeSplit == 0) {
    lock.totalClaimed += claimable;
} else if (claimable > lock.unclaimedBeforeSplit) {
    lock.totalClaimed += claimable - lock.unclaimedBeforeSplit;
    delete lock.unclaimedBeforeSplit;
} else {
    lock.unclaimedBeforeSplit -= claimable;
}
```

Which would make the likeliest branch first in the codepath, without unneeded overhead.

Velodrome: Applied the recommended fix. Refactor the if to the above code block. See commit af892746.

Cantina Managed: Fix confirmed.

3.3.2 Utilization of `Clones` for `Vault` and `GovNFT` deployments

Severity: Gas Optimization

Context: GovNFT.sol#L107, GovNFT.sol#L290, GovNFTFactory.sol#L24, GovNFTFactory.sol#L54, GovNFTTimelockFactory.sol#L24, GovNFTTimelockFactory.sol#L57

Description: Currently every `Vault` & `GovNFTSplit/GovNFTTimelock` contract is deployed standalone. Their respective gas costs in their current state are 330,000 and ~3,500,000. On ETH mainnet, assuming a range of 9 to 72 gwei in gas costs, which has been extrapolated from the past 7 days, this translates into a cost ranging from about $10 to $80 for any Locks created, just to deploy the Vault. The cost for each `GovNFT` would be ten times that.

The primarily intended network is Optimism, where these costs would translate into the cents currently.

Recommendation: Even though the intended network may be Optimism or other L2s, it would be good practice to consider making these contracts viable for more costly networks such as Ethereum mainnet. `Clones` are a great usecase here, as all the deployments are essentially copies of logic with one another, while the OpenZeppelin `Clones` library would help ensure they stay immutable as intended, as long as no metamorphic characteristics exist on their targeted implementations, while saving at least 330,000 gas per such run.

It makes sense for L2s as well, which may be cheap now, but may increase in costs in the future with greater use, and in general would help respective blockchain sizes stay smaller.

Velodrome: The issue has been acknowledged and fixed on the vaults. Applied the recommended fix:

- deploy vaults with `Clones`.

See commit 3ed8873d.

Regarding the deployment of GovNFTs using Clones, we decided to not implement it since we would have to refactor out the original openzeppelin repository (which would've introduced a lot of changes and would reduce readability / maintainability).

Cantina Managed: Confirming appropriate commit for optimizing vault deploy using clones.

3.3.3 `Lock` struct has an unused member

Severity: Gas Optimization

Context: IGovNFT.sol#L20-L32

Description: The `Lock` struct contains a member which is unused throughout the codebase: `minter`. As a result, the `Lock` struct requires an additional storage slot, resulting in reading and writing to the struct being more gas intensive than necessary.

Recommendation: Consider removing `minter` from the struct. If `minter` is included to be retrieved externally, e.g. by a frontend, consider instead including it as a parameter on the `Create` event such that it can still be accessed externally.

Velodrome: The issue has been acknowledged and will not be fixed. Although this parameter is not used on chain, it is widely used UX/UI wise. We could include an extra parameter on the Create event effectively allowing to fetch the minter on chain but it would make its fetching a lot more difficult than checking the lock struct. We understand that this is not ideal but is the best compromise.

Cantina Managed: Acknowledged.

3.3.4 Incrementors can be made `unchecked`

Severity: Gas Optimization

Context: GovNFT.sol#L217, GovNFT.sol#L296

Description: There exist two instances of incrementors being used which cannot possibly overflow in the codebase.

- GovNFT.sol#L217:

```
_tokenId = ++tokenId;
```

- GovNFT.sol#L296:

```
splitTokensByIndex[_from][_parentLock.splitCount++] = _tokenId;
```

We can be certain that these incrementors will never overflow because they are incrementing a `uint256` variable by one with each execution. In practice this would require the logic to be executed 2^{256} times, which is practically impossible.

Note: *Starting in Solidity 0.8.22, the compiler automatically makes for loop counters unchecked if they can't overflow. This issue applies separately only to the incrementors which are not for loop counters.*

Recommendation: Place the incrementors in `unchecked` blocks as follows:

- GovNFT.sol#L217:

```
unchecked {
    _tokenId = ++tokenId;
}
```

- GovNFT.sol#L296:

```
unchecked {
splitTokensByIndex[_from][_parentLock.splitCount++] = _tokenId;
}
```

Velodrome: Applied the recommended fix (add the unchecked blocks in the incrementors). See commit c6c57644.

Cantina Managed: The issue has been fixed as recommended.

3.4 Informational

3.4.1 Sequence off-by-one between `tokenId` and `ERC721Enumerable.tokenByIndex`

Severity: Informational

Context: GovNFT.sol#L217, ERC721Enumerable.sol#L60-L68, IERC721Enumerable.sol#L24-L28

Description: `tokenId` is assigned in a sequential manner, with 1 as its starting index. The contract also inherits `ERC721Enumerable` which exposes tokenByIndex. A `tokenId` of 1, will have an enumerable index of 0. If there is only 1 NFT, and a user attempts to insert 1 into `tokenByIndex` it'll error out.

This is a fine detail that could cause frontend or external services implementation errors resulting in a classic off-by-one error. Additionally may be abused in social engineering cases or scams to sell a less worth NFT, that may be neighbour to a high-value NFT, and tell prospective buyers to check their token using `tokenByIndex`. This latter issue is somewhat alleviated in case of marketplace listings, with the associated `ArtProxy` contract that should relay some pertinent information back.

Recommendation: For simplification purposes and to minimize chances of an error or confusion, consider starting the `tokenId` with an zero-numbering.

```
  function _createNFT(address _recipient, Lock memory _newLock) internal returns (uint256 _tokenId) {
-     _tokenId = ++tokenId;
+     _tokenId = tokenId++;

      _safeMint({to: _recipient, tokenId: _tokenId});

      _locks[_tokenId] = _newLock;
  }
```

Or consider at least noting this relationship within the documentation of `GovNFT` and to avoid relying on `tokenByIndex` as an indicator for `tokenId`, as there in general is no expectation for `tokenId` and `tokenIndex` to match, as per the ERC721 specification 'While some ERC-721 smart contracts may find it convenient to start with ID 0 and simply increment by one for each new NFT, callers SHALL NOT assume that ID numbers have any specific pattern to them, and MUST treat the ID as a "*black box*".

Velodrome: The issue has been acknowledged and will not be fixed. Given that we currently have a mapping that maps to a `tokenId`, we wouldn't be able to differentiate between a null value 0 or the `tokenId` 0.

Cantina Managed: Acknowledged.

3.4.2 Inconsistent OpenZeppelin contract version used

Severity: Informational

Context: package.json#L11C1-L11C40, yarn.lock#L742-L745

Description: The currently linked git submodule pulls OZ v5.0.1, however, the node package manifest and yarn lockfile point to 4.8.0. This discrepancy could culminate in a bunch of issues ranging from build problems to developing for the wrong version of contracts depending on what applies to which part of the toolchain.

Recommendation: Consider if pulling the dependency with yarn is even necessary. If not, remove it, if it is, synchronize it with git submodules.

Velodrome: Applied the recommended fix. Remove the dependency from yarn completely. See commit 669a8fb3.

Cantina Managed: Fix confirmed.

3.4.3 Consider expanding invariant testing to handle expected errors.

Severity: Informational

Context: test/invariants/

Description: The current bounding in the invariant handlers force call parameters into expected successful paths. While this can make shallower invariant runs find numerical regressions more successfully, consider an invariant process that allows random bounds outside of expected ranges and which handle the errors expected on given paths. It can be valuable to conditionally handle expected errors and alert if those errors do not trigger.

Recommendation: Consider allowing parameter bounds outside of expected ranges and handling expected errors and reversions in handler catch blocks. This can provide additional insight into the performance of the contracts when operating outside of expected bounds, but the cost is that more and deeper runs will be necessary to achieve successful call chains.

Velodrome: The issue has been acknowledged and will be fixed. Will will apply the recommended fix:

- Allowing parameters bounds outside expected ranges and handle expected errors.

Cantina Managed: Acknowledged. The fix will be applied in a near future.

3.4.4 Cannot create retroactive vesting

Severity: Informational

Context: GovNFT.sol#L326

Description: Certain large protocols have promised vesting schedules to contributors that begin at a certain timestamp, but due to the heaviness of governance processes the vest is not programmed onchain until after the vest is intended to begin.

The line `if (_startTime < block.timestamp) revert InvalidStart();` would prevent creation of a vesting accrual period in the past, or, if a new lock is created using the a start at the current timestamp, but not mined until the next block, the creation of the vest would fail.

Recommendation: Consider whether this check is necessary to create a valid lock.

Velodrome: Applied the recommended fix. Removed the

```
if (_startTime < block.timestamp) revert InvalidStart();
```

from `_createLockChecks` and added it in the `_validateSplitParams()`

```
if (params.start < _parentLock.start || params.start < block.timestamp) revert InvalidStart();
```

See commit c7d0c9bc.

Cantina Managed: Fixed.

3.4.5 `startTime != endTime` requirement may be too restrictive

Severity: Informational

Context: GovNFT.sol#L330

Description: Lock `startTime` must not equal lock `endTime`. Consider a scenario where a funder wishes to unlock all of the funds after a certain timestamp but without a gradual vesting period. This can be approximated if `endTime == startTime + 1`, but it may be less conceptually sound to require this.

Recommendation: Consider whether there's a need to revert if `startTime == endTime`

Velodrome: The issue has been acknowledged and will not be fixed. As said, this can be achieved by `endTime == startTime + 1`. If we remove the if condition we open the door to other issues (for example, division by zero on `duration/timeElapsed` on the `totalVested`).

Cantina Managed: Acknowledged.

3.4.6 Shadowed function name

Severity: Informational

Context: GovNFTTimelock.sol#L41, GovNFTTimelock.sol#L91, GovNFTSplit.sol#L26

Description: `totalVested` is used as a local variable a few times throughout the codebase (see context above). This variable shadows the name of an existing function.

```
function totalVested(uint256 _tokenId) external view returns (uint256) {
    return _totalVested(_locks[_tokenId]);
}
```

This doesn't cause any logical errors but may affect readability and maintainability.

Recommendation: Consider changing the variable name to avoid shadowing the function name, e.g.:

```
- uint256 totalVested = _totalVested(parentLock);
+ uint256 totalVested_ = _totalVested(parentLock);
```

Velodrome: Applied the recommended fix (change the variable name to `totalVested_`). See commit 552e0fea.

Cantina Managed: The issue has been fixed as recommended.

3.4.7 Large contract deployments require optimizations.

Severity: Informational

Context: GovNFTFactory.sol, GovNFTTimelockFactory.sol

Description: Factory contracts exceed maximum deployment sizes post-Shanghai. Optimizations are required to deploy on mainnet.

Recommendation: Add optimization flags and runs to foundry.toml to document settings.

Velodrome: The issue has been acknowledged and will not be fixed. The recent timelock protection refactor (and other changes) decreased the code size. We have no more contracts that exceed 49152 bytes. For reference, current contract sizes are:

Contract	Size (B)	Margin (B)
Address	86	24,490
ArtProxy	12,780	11,796
Base64	86	24,490
Checkpoints	86	24,490
Clones	86	24,490
ECDSA	86	24,490
ERC721ReceiverMock	1,098	23,478
EnumerableSet	86	24,490
GovNFT	13,935	10,641
GovNFTFactory	17,365	7,211
GovNFTTimelock	14,955	9,621
GovNFTTimelockFactory	18,447	6,129
Math	86	24,490
MessageHashUtils	86	24,490
MockAirdropper	896	23,680
MockERC20	1,831	22,745
MockFeeERC20	1,934	22,642
MockGovernanceToken	7,213	17,363
SafeCast	86	24,490
SafeERC20	86	24,490
ShortStrings	86	24,490
SignedMath	86	24,490

Contract	Size (B)	Margin (B)
StorageSlot	86	24,490
Strings	86	24,490
TestOwner	1,436	23,140
Time	86	24,490
TimeStore	474	24,102
Vault	2,201	22,375

Cantina Managed: Would still recommend having the optimizer on, as it will generally decrease both deploy costs and run costs with the default parameter (200), and for more expensive networks can be tuned to higher optimization runs for cheaper run costs. Although this isn't security related, but more so optimization.

Velodrome: We also decided to leave this stale because the default for forge is to have the optimizer on with 200 runs. So, setting the optimizer to true and 200 in the `foundry.toml` config file won't change anything. If we get to deploy on other networks we can add that config to the `foundry.toml` with the necessary values.

Cantina Managed: Acknowledged.

3.4.8 Missing `LICENSE`

Severity: Informational

Context: GovNFT/contracts/

Description: License recently modified to `GPL-3.0`. `LICENSE` or `COPYING` file is required to be distributed with the code per GNU specification.

Recommendation: Add a `LICENCE` or `COPYING` file to the repo for license adherence.

Velodrome: Applied the recommended fix (add the license.md file to the project's root). See commit 16ecc140.

Cantina Managed: Fix confirmed.