Velvet Staking Contracts

Security Review

Cantina Managed review by:
Joran Honig, Lead Security Researcher

July 9, 2025

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

Velvet Capital is a DeFAI Trading & Portfolio Management Ecosystem powered by intents, offering a vertically integrated stack with a native user app, an agentic TG bot, APIs for easy integration and a DeFAI OS for agents.

On Jul 8th the Cantina team conducted a review of velvet-staking-contracts on commit hash a61f6af3; the scope if the review was limited to `veVelvet.sol`. The team identified a total of **7** issues in the following risk categories:

- **Critical**: 0
- **High**: 2
- **Medium**: 2
- **Low**: 1
- **Gas Optimizations**: 0
- **Informational**: 2

3 Findings

3.1 High Risk

3.1.1 Autorenewed locks have value in the past

Severity: High Risk

Context: veVelvet.sol#L115-L137, veVelvet.sol#L228-L229

Description: The logic which determines the balance of a lock at a given timestamp incorrectly handles locks which are auto renewed. Regardless of the timestamp passed to `_balanceOfLockAt()` the function will return `lock.amount` as the value of the lock at that timestamp. An attacker can manipulate downstream integrations which rely on the output of `balanceOfAt`, by retroactively increasing their balance for all timestamps. In addition, when autorenewal is toggled the system resets the `start` and `end` timestamps of a lock. This similarly inhibits the functioning of `balanceOfAt`, misrepresenting the balance of a lock at a given timestamp. This flaw might lead to a loss of funds as such integrations often include reward mechanisms.

Recommendation: It's worthwhile to adjust tracking of auto renewed locks such that this function accurately checks when the lock was created and when it ended. Note however, that this requires changes in multiple places including `toggleAutoRenew`.

Velvet Capital:

1. Auto-renew snapshot bug — fixed. We added a `timestamp < lock.start ⇒ 0` guard, so auto-renewed locks no longer report voting power for periods before they were created.
2. Start/end reset on toggle — expected behaviour. Turning auto-renew on or off intentionally opens a fresh lock from block.timestamp; this mirrors Curve/Velodrome practice and keeps historical snapshots intact, so no further change is planned.

3.1.2 Re-entrancy allows a malicious user to arbitrarily transfer governance voting units

Severity: High Risk

Context: veVelvet.sol#L216-L218, veVelvet.sol#L288-L300

Description: At the end of a withdrawal the contract calls `_transferVotingUnits()`. This will look up the delegatee of the account and reduce their voting units. However, the `delegate()` function (available through `VotesUpgradable`) is not protected with a `nonReentrant` modifier. Similarly, the implementation of `_getVotingUnits` in `veVevet` does not prevent reentrant calls.

As a result an attacker can potentially leverage the external call (`safeTransfer`) to change delegatee before the `_transferVotingUnits` call. A change of delegatees executed during the `safeTransfer` would use a state where the lock and thus `_getVotingUnits()` result is already updated. So it would not attempt to move the to be burned voting units. The consecutive `_transferVotingUnits()` to address zero would then remove voting units from the new delegatee even though they never received them. In effect the attacker will have transferred amount tokens from the new delegatee to the old delegatee.

This attack is possible when baseToken performs external calls, note however that the scope of this audit is limited to `veVelvet`.

Recommendation: Consistent application of reentrancy modifiers should prevent this from being exploited.

Velvet Capital: Acknowledged — the reported vector relies on baseToken.transfer executing an external callback; our VELVET ERC-20 is audited, non-upgradeable, and follows the plain ERC-20 spec, which by design does not re-enter callers. `withdraw()` is already guarded by OpenZeppelin's `ReentrancyGuardUpgradeable`, preventing nested calls even if such a callback existed

Because those conditions cannot occur in our deployment, the delegate-swap scenario is not exploitable, so no code change is planned.

Cantina Managed: If you're sure that the base token doesn't do external calls then you can consider accepting this risk. I'd still recommend resolving this vector as it reduces the chance of accidents in future deployments.

A quick sidenote, using the reentrancy guard on the withdraw function does not prevent reentrant calls to functions which do not have the reentrancy guard modifier applied to them. Calls to `delegate()` would remain possible.

3.2 Medium Risk

3.2.1 Lacking a limit on the amount of locks can cause issues with gas limits

Severity: Medium Risk

Context: veVelvet.sol#L29, veVelvet.sol#L97-L101, veVelvet.sol#L190-L195, veVelvet.sol#L200, veVelvet.sol#L223, veVelvet.sol#L236, veVelvet.sol#L260, veVelvet.sol#L294-L300, README.md#L16

Description/Recommendation: Various parts of the code base loop over a list of locks for a user. Such loops will start failing as the user gets too many positions and can for example (temporarily) prevent them from withdrawing funds. The codebase does have constant indicating the maximum number of positions `MAX_POSITIONS`. Similarly, the documentation refers to this invariant being enforced ensuring a maximum of 200 positions. However, this constant is not used to ensure that a user doesn't create too many positions. As a result they're able to create an arbitrary amount of positions and potentially run into gas issues locking them out of their positions.

3.2.2 Admin functionality reduces efficacy of governance

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The `veVelvet` token allows admins to set the `adminUnlocked` state and `maxWeeks`.

Both of these variables can be used to effect a situation where a user can lock their tokens and get voting power while also immediately being able to withdraw the tokens. As a result an admin can perform flashloan attacks on governance where they perform a sequence like the following:

- Get flashloan.
- Purchase veToken.
- Lock veToken.
- Vote on proposal.
- Call `setAdminUnlocked(true)`.
- Withdraw veToken.
- Call `setAdminUnlocked(false)`.
- Repay flashloan.

Recommendation: Introduce time locks for admin functionality such as unlocks. Furthermore, it could be worthwhile requiring maxWeeks to be greater than or equal to 1.

Velvet Capital: Acknowledged — the admin role is a trusted, multisig, so immediate access to adminUnlocked and setMaxWeeks is an intentional design decision and will remain unchanged.

3.3 Low Risk

3.3.1 `setMaxWeeks` should have delay to prevent locking in autorenew positions

Severity: Low Risk

Context: veVelvet.sol#L154-L156, veVelvet.sol#L252-L254

Description/Recommendation: Updates to `maxWeeks` are instant and existing autorenewed positions will automatically use the latest maxWeeks duration when they turns of autorenewal for one of their locks. It's important to give users a time window in which they can turn off auto renewal as `setMaxWeeks` can inadvertently lock them in for an undesired period.

Velvet Capital: Acknowledged — this behaviour is intentional: auto-renew locks are designed to follow whatever maxWeeks is currently set to (as in Curve's veCRV), and users can disable auto-renew at any time if they don't want the new duration.

Cantina Managed: The problem illustrated here isn't that positions are auto renewed. It's that users don't get a chance to turn off autorenewal before `setMaxWeeks` is called. `maxWeeks` might be set at 2 weeks at first, and later changed to 2 years. A user that expected their position to renew every two weeks would suddenly be locked in for 2 years.

3.4 Informational

3.4.1 Consider introducing code documentation

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The veToken contract is mostly undocumented. As a result divergences between intended functioning and implementation can go undetected.

Recommendation: Introduce NatSpec documentation describing the functionality of the public interfaces. This has the added benefit of improving the development workflow of components that interact with `veToken` and improves maintainability.

3.4.2 Use battle tested math libraries over manually implementing complex arithmetic

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: `veToken` includes various manual *mulDiv* computations.

Recommendation: Consider leveraging the existing math functions available in the OpenZeppelin contracts. These are well tested, provide high precision and allow for simplification of the `veToken` codebase.