AstraSec

StellaSwap veSTELLA Security Audit Report

November 17, 2025

Contents

1 Introduction

1.1 About StellaSwap

StellaSwap is a pioneering automated market maker (AMM) and decentralized exchange (DEX) built on the Moonbeam parachain network. It offers a unified platform where users can swap tokens, earn rewards, yield farm, bridge assets, explore new projects, and trade NFTs. The audited veSTELLA system enables users to lock their STELLA tokens to generate voting power and earn a significant share of the protocol's revenue.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/stellaswap/ve-contracts.git
- CommitID: fd1cee4

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/stellaswap/ve-contracts.git
- CommitID: 22d1d59

Note that this audit only covers the following contracts: contracts/ve/veNFT.sol, contracts/ve/libraries/BalanceLibrary.sol, contracts/voting.sol, contracts/StellaMinter.sol, contracts/libraries/**/, contracts/StellaVetoGovernor.sol, and contracts/ProtocolGovernor.sol.

1.3 Revision History

Version	Date	Description
v1.0	December 2, 2024	Initial Audit

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the StellaSwap veSTELLA protocol. Throughout this audit, we identified a total of 8 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	–	–	–	–
High	4	–	–	4
Medium	3	2	–	1
Low	1	–	–	1
Informational	–	–	–	–
Undetermined	–	–	–	–

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Acknowledged
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and re-mediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

3.3.1 [H-1] Improper Burn Logic in VeNFT::_afterTokenTransfer()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
VeNFT.sol	Business Logic	High	High	Addressed

The _afterTokenTransfer() function in the VeNFT contract fails to properly handle token burns due to the condition if (to != address(0)) (line 885). When to is equal to address(0) (indicating a burn scenario), the function skips calling _removeTokenFromOwnerList() and _moveTokenDelegates() (lines 887/888), leaving stale ownership records and lingering delegation states. This results in an inconsistent contract state where burned tokens remain in ownership lists and vote calculations, potentially inflating vote counts and misrepresenting token ownership.

ve-contracts - VeNFT.sol

```
874 function _afterTokenTransfer(address from, address to, uint256 firstTokenId, uint256 batchSize)
875     internal
876     virtual
877     override
878 {
879     if (from == address(0)) {
880         // mint
881         _addTokenToOwnerList(to, firstTokenId);
882         _moveTokenDelegates(address(0), delegates(to), firstTokenId);
883     }
884
885     if (to != address(0)) {
886         // burn
887         _removeTokenFromOwnerList(from, firstTokenId);
888         _moveTokenDelegates(delegates(from), address(0), firstTokenId);
889     }
890
891     ...
892 }
```

Remediation Improve the implementation of the _afterTokenTransfer() function to ensure proper handling of token burns.

3.3.2 [H-2] Improper Logic of VeNFT::_increaseAmountFor()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
VeNFT.sol	Business Logic	High	High	Addressed

The _increaseAmountFor() function in the VeNFT contract is intended to increase the locked amount for a given _tokenId. According to the design, if the _tokenId is of type LockType.MNFT, the increased Stella tokens should be treated as locked rewards and sent to the corresponding reward manager via notifyRewardAmount(). However, due to an incorrect condition (if (_lockType == LockType.MANAGED) on line 429), the function mistakenly checks for LockType.MANAGED instead of the expected LockType.MNFT. This logic error prevents the reward notification process from being triggered, leading to a deviation from the intended functionality.

ve-contracts - VeNFT.sol

```
417 function _increaseAmountFor(uint256 _tokenId, uint256 _value, DepositType _depositType) internal {
418     LockType _lockType = lockType[_tokenId];
419     if (_lockType == LockType.MANAGED) revert veNFTErrors.NotNormalNFT();
420     LockedBalance memory oldLocked = _locked[_tokenId];
421
422     if (_value == 0) revert veNFTErrors.ValueCannotBeZero();
423     if (oldLocked.amount <= 0) revert veNFTErrors.LockExpired();
424     if (oldLocked.end <= block.timestamp && !oldLocked.isPermanent) revert veNFTErrors.LockExpired();
425     if (oldLocked.isPermanent) permanentLockedBalance += _value;
426     // _checkpointDelegatee(_delegates[_tokenId], _value, true);
427     _depositFor(_tokenId, _value, 0, oldLocked, _depositType);
428
429     if (_lockType == LockType.MANAGED) {
430         // increaseAmount called on managed tokens are treated as locked rewards
431         address lockedReward = managedNFTRewards[_tokenId].lockedReward;
432         IERC20(stella).safeApprove(lockedReward, _value);
433         IIncentiveManager(lockedReward).notifyRewardAmount(stella, _value);
434         IERC20(stella).safeApprove(lockedReward, 0);
435     }
436
437     // emit MetadataUpdate(_tokenId);
438 }
```

Remediation Improve the implementation of the _increaseAmountFor() function as above-mentioned.

3.3.3 [H-3] Possible Duplicate Vote via Voting::depositIntoManagedNft()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Voting.sol	Business Logic	High	High	Addressed

The depositIntoManagedNft() function in the Voting contract introduces a duplicate voting vulnerability due to insufficient validation of already-voted veSTELLAs. A user can vote with a normal veSTELLA via the vote() function and then deposit the same NFT into a LockType.MNFT veSTELLA using depositIntoManagedNft(). This action transfers the voting power of the already-voted veSTELLA to the LockType.MNFT veSTELLA, allowing the same voting power to be reused. Consequently, this flaw enables malicious actors to inflate their voting power by counting the same votes multiple times, undermining the fairness and integrity of the voting process.

ve-contracts - voting.sol

```
237 function vote(uint256 nftId, address[] memory _poolAddresses, uint256[] memory weights)
238     public
239     onlyDuringEpoch(nftId)
240 {
241     ...
242     _vote(nftId, _poolAddresses, weights, voterWeight, totalWeights);
243 }
244
245 function depositIntoManagedNft(uint256 _tokenId, uint256 _managedTokenId)
246     external
247     nonReentrant
248     onlyDuringEpoch(_tokenId)
249 {
250     if (!veStellaToken.isApprovedOrOwner(msg.sender, _tokenId)) revert VotingErrors.NotApprovedOrOwner();
251     if (veStellaToken.lockType(_tokenId) != IVEStella.LockType.NORMAL) revert VotingErrors.NotNormalNFT();
252
253     veStellaToken.depositIntoManagedNFT(_tokenId, _managedTokenId);
254
255     uint256 voterWeight = veStellaToken.balanceOfNFT(_managedTokenId);
256
257     _poke(_managedTokenId, voterWeight);
258 }
259
```

Remediation Add necessary sanity checks to ensure that veSTELLAs with active votes cannot be deposited into a LockType.MNFT veSTELLA.

3.3.4 [H-4] Improper Voting Weight Update in withdrawFromManagedNft()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Voting.sol	Business Logic	High	High	Addressed

The withdrawFromManagedNft() function enables a user to withdraw a normal veSTELLA (represented by _tokenId) from its associated LockType.MNFT veSTELLA, detaching its voting power and restoring it to an independent state.

However, upon thorough examination, we identify that while the voting power of the LockType.MNFT veSTELLA changes after the withdrawal, the function fails to update the reward-related state to reflect this change. As a result, the outdated voting power is used for reward calculations, leading to an inaccurate and unfair distribution of rewards.

ve-contracts - voting.sol

```
388 function withdrawFromManagedNft(uint256 _tokenId) external nonReentrant onlyDuringEpoch(_tokenId) {
389     if (!veStellaToken.isApprovedOrOwner(msg.sender, _tokenId)) revert VotingErrors.NotApprovedOrOwner();
390
391     veStellaToken.withdrawFromManagedNFT(_tokenId);
392
393     _reset(_tokenId);
394 }
```

Remediation Update the reward-related states based on the latest voting power of the LockType.MNFT veSTELLA to ensure fair reward distribution.

3.3.5 [M-1] Possible DoS Attack for MAX_DELEGATES Limit

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
VeNFT.sol	Business Logic	Medium	Medium	Acknowledged

The _moveTokenDelegates() function, which handles transferring delegated votes between delegatees, is vulnerable to a Denial of Service (DoS) attack due to the absence of restrictions on veSTELLA creation. A malicious actor can repeatedly create veSTELLAs with negligible deposits (e.g., 1 wei Stella) and delegate them to a target address. This artificially inflates the delegate count, eventually exceeding the MAX_DELEGATES limit and causing the function to revert. As a result, the target address is blocked from further delegations or related operations. To mitigate this, enforcing a minimum deposit amount and lock duration during veSTELLA creation is recommended to prevent exploitation.

ve-contracts - VeNFT.sol

```
840 mapping(address => address) private _delegates;
841 uint256 public constant MAX_DELEGATES = 256; // avoid too much gas
842 ...
843
844 function _moveTokenDelegates(address srcRep, address dstRep, uint256 _tokenId) internal {
845     if (srcRep != dstRep && _tokenId > 0) {
846         if (srcRep != address(0)) {
847             ...
848         }
849
850         if (dstRep != address(0)) {
851             uint32 dstRepNum = numCheckpoints[dstRep];
852             uint256[] storage dstRepOld =
853                 dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].tokenIds : checkpoints[dstRep][0].tokenIds;
854             uint32 nextDstRepNum = _findWhatCheckpointToWrite(dstRep);
855             uint256[] storage dstRepNew = checkpoints[dstRep][nextDstRepNum].tokenIds;
856             // All the same plus _tokenId
857             if (dstRepOld.length + 1 > MAX_DELEGATES) revert veNFTErrors.TooManyTokenIds();
858             for (uint256 i = 0; i < dstRepOld.length; i++) {
859                 uint256 tId = dstRepOld[i];
860                 dstRepNew.push(tId);
861             }
862             dstRepNew.push(_tokenId);
863
864             numCheckpoints[dstRep] = dstRepNum + 1;
865         }
866     }
867 }
```

Remediation Set a minimum deposit amount and lock duration for normal veSTELLA creation to mitigate the DoS risk.

3.3.6 [M-2] Improper Logic of VeNFT::_moveTokenDelegates()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
VeNFT.sol	Business Logic	High	High	Addressed

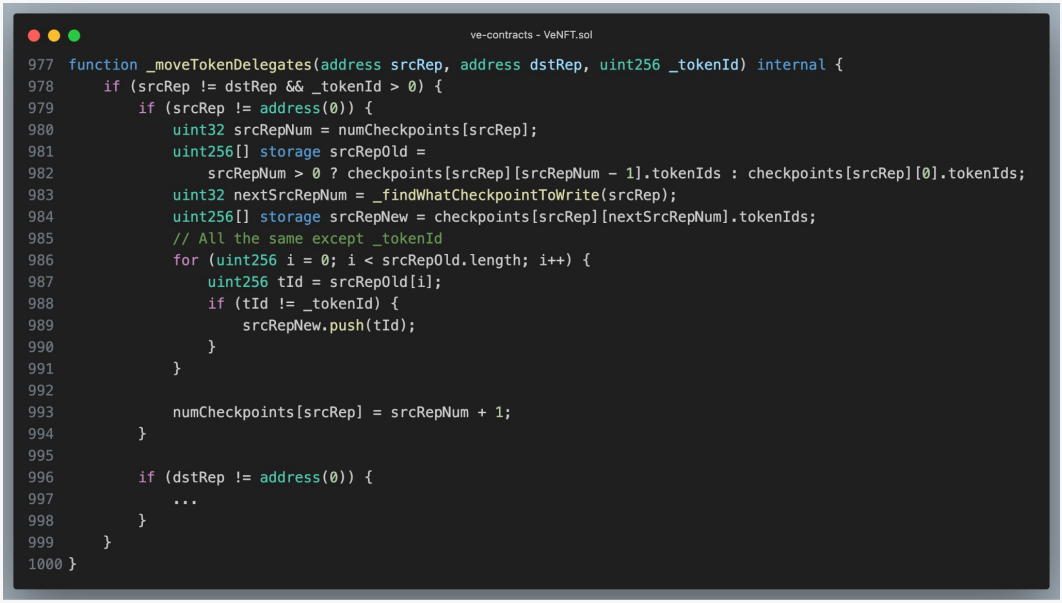
The _moveTokenDelegates() function, responsible for updating the delegation state when veSTELLA is transferred, contains several logical flaws:

- ➔ **Timestamp Not Updated:** The function fails to update the timestamp when creating or modifying checkpoints. This results in stale timestamps, undermining accurate tracking of delegation history.
- ➔ **Duplicate Token IDs:** If the current timestamp matches the last checkpoint's timestamp, _findWhatCheckpointToWrite() returns the last checkpoint index. The function then retrieves the tokenIds from this checkpoint and redundantly appends them (excluding the currently transferred _tokenId) back into srcRepNew. This leads to duplicate tokenIds being stored within the same checkpoint, inflating vote counts during calculations.
- ➔ **Improper Increment of numCheckpoints:** Despite reusing the last checkpoint when the timestamp remains unchanged, numCheckpoints[srcRep] is still incremented. This artificially inflates the total number of checkpoints, causing further inconsistency in the delegation state.

Moreover, the _moveAllDelegates() function shares the similar issue.

ve-contracts - VeNFT.sol

```
1015 function _findWhatCheckpointToWrite(address account) internal view returns (uint32) {
1016     uint256 _timestamp = block.timestamp;
1017     uint32 _nCheckPoints = numCheckpoints[account];
1018
1019     if (_nCheckPoints > 0 && checkpoints[account][_nCheckPoints - 1].timestamp == _timestamp) {
1020         return _nCheckPoints - 1;
1021     } else {
1022         return _nCheckPoints;
1023     }
1024 }
```



Remediation Ensure accurate timestamp updates, prevent duplicate token IDs in checkpoints, and avoid unnecessary increments of numCheckpoints.

3.3.7 [M-3] Potential Risks Associated with Centralization

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	High	Low	Acknowledged

In the StellaSwap veSTELLA protocol, the existence of a series of privileged accounts introduces centralization risks, as they hold significant control and authority over critical operations governing the protocol. In the following, we show the representative function potentially affected by the privileges associated with the privileged accounts.

ve-contracts - StellaMinter.sol

```
104 function setWeekly(uint256 _weekly) public onlyOwner {
105     require(_weekly > 0, "Minter: weekly must be greater than zero");
106     weekly = _weekly;
107     emit WeeklyUpdated(_weekly);
108 }
109
110 function updateRates(uint256 _teamRate, uint256 _investorRate, uint256 _treasuryRate, uint256 _ecosystemRate) public onlyOwner {
111     require(_teamRate <= 1500, "Minter: team rate cannot exceed 1500");
112     require(_investorRate <= 1500, "Minter: investor rate cannot exceed 1500");
113     require(_treasuryRate <= 1500, "Minter: treasury rate cannot exceed 1500");
114     require(_ecosystemRate <= 1500, "Minter: ecosystem rate cannot exceed 1500");
115
116     teamRate = _teamRate;
117     investorRate = _investorRate;
118     treasuryRate = _treasuryRate;
119     ecosystemRate = _ecosystemRate;
120
121     emit RatesUpdated(_teamRate, _investorRate, _treasuryRate, _ecosystemRate);
122 }
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged accounts. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team This issue has been confirmed by the team.

3.3.8 [L-1] Incompatibility with Non-Standard ERC20 Tokens

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Business Logic	Low	Low	Addressed

Inside the Voting::notifyRewardAmount() function, the statement of IERC20(_tokens[i]).transferFrom(msg.sender, address(this), _amounts[i]) is employed to transfer the user's asset into the Voting contract. However, in the case of USDT-like token whose transferFrom() lacks a return value, it would lead to a revert. Given this, we recommend employing the widely-used SafeERC20 library (which serves as a wrapper for ERC20 operations while accommodating a diverse range of non-standard ERC20 tokens) to address this case.

ve-contracts - voting.sol

```
180 function notifyRewardAmount(address[] calldata _tokens, uint256[] calldata _amounts)
181     external
182     nonReentrant
183     onlyRole(ADMIN_ROLE)
184 {
185     if (_tokens.length != _amounts.length) {
186         revert VotingErrors.ArrayLengthMismatch();
187     }
188     for (uint256 i = 0; i < _tokens.length; i++) {
189         ...
190         IERC20(_tokens[i]).transferFrom(msg.sender, address(this), _amounts[i]);
191     }
192
193     emit RewardNotified(_tokens, _amounts);
194 }
```

Remediation Suggest replacing transfer(), transferFrom(), and approve() with safeTransfer(), safeTransferFrom(), and safeApprove().

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://x.com/AstraSecAI