CERTORA

Smart Contract Security Analysis Report and Formal Verification Properties of MUTE v2

Last updated: September 20, 2023

Table of Contents

Summary

This document describes the specification and verification of the **MUTEv2 protocol** using the Certora Prover and manual code review findings. The work was undertaken from **21 August 2023** to **20 September 2023**. The commits reviewed and run through the Certora Prover were ce259eb, 4b06ee0, ee4b710, b43309c, and the final commit, fffb677.

The following contract list is included in our scope:

`mute-switch-core/contracts/daov2/veMuteV2.sol`
`mute-switch-core/contracts/governance/MuteGovernor.sol`
The contracts are written in Solidity 0.8.0 and 0.8.2.

The Certora Prover demonstrated the implementation of the Solidity contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solidity contracts. During the verification process and the manual audit, the Certora Prover discovered bugs in the Solidity contracts code, as listed below.

Summary of findings

The table below summarizes the issues discovered during the audit, categorized by severity.

	Total discovered	Total fixed	Total acknowledged
Critical	2	2	0
High	1	1	0
Medium	6	4	2
Low	6	4	2
Informational	40	20	20
Total	**55**	**31**	**24**

Main Issues Discovered

Crit-01: Any user can multiply his votes transferred from the legacy contract

Severity: Critical
Category: Vote manipulation
File(s): veMuteV2.sol
Bug description: This bug was introduced later in the code after attempting to fix M-01. It is possible to exploit the LockLegacyIndex() function and mint duplicate NFTs representing the same locked Mute tokens. This allows any user to multiply his votes transferred from the legacy contract by any number, limited only by the maximal possible length of the _indexes array.
Exploit scenario: By calling LockLegacyIndex() with a list of duplicate indices, it is possible to create many NFTs representing the same locked Mute tokens in the legacy contract. The LockLegacyIndex() function will call mintLegacyToken() for each element in _userLocks as many times as the corresponding index appears in the _indexes array.
Mute's response: ***Fixed*** in the final commit fffb677.

Crit-02: Any user can multiply his votes transferred from the legacy contract (another attack vector)

Severity: Critical
Category: Vote manipulation
File(s): veMuteV2.sol
Bug description: This bug was introduced later in the code and was not prevented by the first fix to Crit-01. It is possible to exploit the LockLegacyIndex() function and mint duplicate NFTs representing the same locked Mute tokens. This allows any user to multiply his transferred votes from the legacy contract by any number, limited only by the maximal possible length of the locks[][] array and the length of _userLocks in the legacy contract.
Exploit scenario: The safeMint() function invokes _checkOnERC721Received() on the target address (if that address is a contract). This allows any contract that calls the LockLegacyIndex() function to gain execution right during each call to mintLegacyToken().

The proposed method of attack is to set up a contract that owns many redeemable "dust" locks on the legacy contract and one true lock which the attacker means to duplicate (_userLocks[CONTRACT_ADDRESS][i].tokens_minted == 1, _userLocks[CONTRACT_ADDRESS][i].time <= block.timestamp for any i <= _userLocks[CONTRACT_ADDRESS].length - 2, _userLocks[CONTRACT_ADDRESS][_userLocks[CONTRACT_ADDRESS].length - 1].tokens_minted == MEANINGFUL_AMOUNT). The attacking contract would then call the LockLegacyIndex() function with a list of indices in descending order (locks[LEGACY_CONTRACT][j] = _userLocks[CONTRACT_ADDRESS].length - 1 - j), using his execution rights during each iteration to call the Redeem() function on the second to last element in _userLocks in the legacy contract. On each iteration, the last element in _userLocks[CONTRACT_ADDRESS] is being read from the legacy contract, while the call to

Redeem() then pops out the second to last element and moves the significant element one place backwards (to the place which corresponds to the new last index in _userLocks[CONTRACT_ADDRESS]).
Mute's response: ***Fixed*** in the final commit fffb677.

H-01: Impossible to transfer votes stored at the first element in _userLocks

Severity: High
Category: Broken functionality
File(s): veMuteV2.sol
Bug description: This bug was introduced in a later version of the code after attempting to fix M-01. It is impossible to transfer votes stored at the first element in _userLocks using the LockLegacyIndex() function.
Exploit scenario: As the initializedIds[] array is initialized to be all zeros, calling LockLegacyIndex() with locks[i][j] == 0 for any i, j will cause the function to revert. As the first index in _userLocks is 0, transferring votes stored at that element would be impossible.
Mute's response: ***Fixed*** in the final commit fffb677.

M-01: Denial to transfer votes from the legacy contracts

Severity: Medium
Category: Denial of functionality
File(s): veMuteV2.sol
Bug description: It is possible to target a specific victim and deny him the possibility to transfer votes from the legacy contracts to the new one.
Exploit scenario: In the original dMute.sol contract, an attacker can call the lockTo function on behalf of the victim many times, each time minting a minimal amount of voting power, and locking it for the maximal amount of time (i.e. calling LockTo() with _amount = 1, _lock_time = 52 weeks, to = VICTIM as parameters). This process allows the attacker to boundlessly increase _userLocks[VICTIM].length in the legacy contract. As the LockLegacy() function iterates over all the elements in the _userLocks[VICTIM] array, there exists a limit on the length of the array beyond which all calls to the function would revert due to an out-of-gas exception. Note that while the attacker can use many transactions to inflate _userLocks[VICTIM] in the legacy contract, the process of iterating over all the elements in _userLocks[VICTIM] has to occur in one single transaction and thus is bounded by Ethereum's block gas limit. Also note, that by setting _lock_time to the maximum possible amount it would not be possible for the victim to call RedeemTo for the locks that were set by the attacker before any previous locks set by the victim had already expired.
Mute's response: ***Fixed*** in the final commit fffb677.

M-02: Expired NFTs are still good for voting

Severity: Medium
Category: Economic
File(s): veMuteV2.sol

Bug description: Expired NFTs are still good for voting, and can be sold in a secondary market without the need to re-lock Mute tokens.
Exploit scenario: It is possible for someone to lock a substantial amount of Mute tokens for the maximal allowed time (52 weeks); After that period of time has passed, there is the least economic incentive to redeem that NFT as it has the best possible voting power to locked Mute tokens ratio (vote_weight/amount == 1) and is fully liquidable. This in turn might lead to a situation in which the owner of the NFT would be able to split that NFT and sell parts of it to new users who wish to own voting NFTs, bypassing the intended requirement of committing to Mute by locking their mute tokens.
Mute's response: ***Acknowledged.*** *We came to the conclusion that having a premium market for vote weight mute that are close to unlock time (or are expired) adds an interesting dynamic for both markets, so we are fine with the current design*

M-03: The domainSeparator mismatches with DOMAIN_TYPEHASH

Severity: Medium
Category: Non-compliance with standard
File(s): veMuteV2.sol
Bug description: The domainSeparator in delegateBySig() uses the string version member of the domain separator equal to "1":

```
bytes32 domainSeparator = keccak256(abi.encode(
                                        DOMAIN_TYPEHASH,
                                        keccak256(bytes(name())),
                                        keccak256(bytes("1")),
                                        Block.chainid,
                                        address(this)));
```

However, the declared and used DOMAIN_TYPEHASH doesn't have it:

```
/// @notice The EIP-712 typehash for the contract's domain
bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(
                                        string name,
                                        uint256 chainId,
                                        address verifyingContract)");
```

Therefore the signature won't respect the standard, and the one used by the protocol user is very likely to mismatch with the one calculated by the protocol.
Mute's response: ***Fixed*** in the final commit fffb677.

M-04: _safeMint() should be used rather than _mint() wherever possible

Severity: Medium
Category: Non-compliance with standard
File(s): veMuteV2.sol lines 153, 169

Bug description: _mint() is discouraged in favor of _safeMint() which ensures that the recipient is either an EOA or implements IERC721Receiver. Both OpenZeppelin and solmate have versions of this function so that NFTs aren't lost if they're minted to contracts that cannot transfer them back out.

Be careful, however, to respect the CEI pattern or add a re-entrancy guard as _safeMint adds a callback-check (_checkOnERC721Received) and a malicious onERC721Received could be exploited if not careful.

Reading material:

- https://blocksecteam.medium.com/when-safemint-becomes-unsafe-lessons-from-the-hypebears-security-incident-2965209bda2a
- https://samczsun.com/the-dangers-of-surprising-code/
- https://github.com/KadenZipfel/smart-contract-attack-vectors/blob/master/vulnerabilities/unprotected-callback.md

Mute's response: ***Fixed*** in the final commit fffb677.

M-05: Return values of transfer()/transferFrom() not checked

Severity: Medium
Category: Deviation from best practices
File(s): veMuteV2.sol lines 211, 270
Bug description: Not all IERC20 implementations revert() when there's a failure in transfer()/transferFrom(). The function signature has a boolean return value, and they indicate errors that way instead. By not checking the return value, operations that should have been marked as failed may potentially go through without actually making a payment
Mute's response: ***Fixed*** in the final commit fffb677.

M-06: A user could momentarily double his voting power

Severity: Low
Category: Vote manipulation
File(s): veMuteV2.sol
Bug description: As the burnLegacy() function must actively be called by the system's owner after the expiration of Legacy NFTs, it is possible for a user to momentarily double his voting power.
Exploit scenario: As the system relies on the system's owner to call the burnLegacy() function in order to burn expired legacy NFTs, a user might take advantage of the time window between the expiration of the legacy NFT and the execution of burnLegacy() and redeem his mute tokens from the legacy contract. This in turn will allow him to re-lock these tokens in the new contract, momentarily holding both the legacy NFT and the new NFT. Note that by calling LockLegacy() only after the expiration time has passed, the momentary doubling of the votes could be achieved without the need to "race" the system's owner, as the transferring of the votes from the legacy contract, the redemption of the Mute tokens from the legacy contract and the process of re-locking the redeemed Mute tokens could all be done in the same transaction.
Mute's response: ***Acknowledged***.

L-01: It is impossible to split legacy NFTs

Severity: Low
Category: Broken functionality
File(s): veMuteV2.sol, line 276
Bug description: It is impossible to split legacy NFTs.
Exploit scenario: Legacy NFTs can only be minted with _nftLocks[_tokenId].amount == 0 (line 233). As the _amount parameter in the Split() function must be strictly smaller than _nftLocks[_tokenId].amount, any call attempting to split a legacy NFT is guaranteed to revert.
Mute's response: ***Fixed*** in the final commit fffb677.

L-02: Front-running the system owner can cause a revert of burnLegacy()

Severity: Low
Category: Denial of functionality
File(s): veMuteV2.sol
Bug description: It is possible to front-run the system's owner and cause the execution of burnLegacy to revert.
Exploit scenario: As the system's owner has to call burnLegacy() and provide an array of all the IDs of legacy NFTs needed to be burned, front running this call by burning one of the NFTs in the array (e.g., by calling the Redeem() function) would cause the owner's transaction to revert.
Mute's response: ***Acknowledged.***

L-03: Denial of the possibility to call both the GetUnderlyingTokens() and the GetVotingTokens() functions

Severity: Low
Category: Denial of functionality
File(s): veMuteV2.sol
Bug description: It is possible to target a specific victim and deny him the possibility to call both the GetUnderlyingTokens() and the GetVotingTokens() functions.
Exploit scenario: As both the GetUnderlyingTokens() and the GetVotingTokens() functions iterate over all the NFTs owned by the victim, it is possible for an attacker to mint/transfer many insignificant NFTs to the victim, causing the functions to revert with an out-of-gas exception. Note that even though the external view function does not cost gas, the queried node will still limit the amount of operations in that call.
Mute's response: ***Acknowledged***. Since those functions cannot risk any loss of funds and are mainly used as peripheral view functions, it might not be worth addressing since there are plenty of ways to get around it.

L-04: Use a 2-step ownership transfer pattern

Severity: Low
Category: Deviation from best practices
File(s): veMuteV2.sol
Bug description: Consider implementing a two-step process where the owner or admin nominates an account and the nominated account needs to call an acceptOwnership() function for the transfer of ownership to succeed fully. This ensures the nominated EOA account is a valid and active account. The lack of a two-step procedure for critical operations leaves them error-prone. Consider adding a two-step procedure on the critical functions.
Mute's response: ***Fixed*** in the final commit fffb677.

L-05: A known vulnerability exists in the currently used @openzeppelin/contracts version

Severity: Low
Category: Deviation from best practices
File(s): package.json, veMuteV2.sol lines 277, 321, 483
Bug description: As a known vulnerability impacting the current codebase exists in the current @openzeppelin/contracts@4.9.2 version, consider updating to at least @openzeppelin/contracts@4.9.3
Mute's response: ***Fixed*** in the final commit fffb677.

L-06: Inconsistency of token expiration between burnLegacy() and GetVotingTokens()

Severity: Low
Category: Inconsistent behavior
File(s): veMuteV2.sol lines 197, 393

```
194:    function burnLegacy(uint256[] memory _ids) external onlyOwner nonReentrant {
...
197:            if(_nftLocks[_ids[i]].legacy == true && block.timestamp >=
_nftLocks[_ids[i]].time){
198:                burnToken(_ids[i]);

385:    function GetVotingTokens(address account) public view returns(uint256 amount) {
...
393:            if(_nftLocks[_token].legacy == true && _nftLocks[_token].time <
block.timestamp)
394:                vote_bal = 0;
```

Bug description: At a certain block.timestamp == _nftLocks[_token].time, burnLegacy() will consider the token expired and burn it while GetVotingTokens() will return a non-zero vote balance
Mute's response: ***Fixed*** in the final commit fffb677.

Info-01: Unreachable code

Severity: Info
Category: Gas
File(s): veMuteV2.sol, line 346
Bug description: Unreachable code. The check if(new_lock_legacy == true) and its subsequent code are redundant as new_lock_legacy == lock_0.legacy, which is required to be false.
Mute's response: ***Fixed*** in the final commit fffb677.

Info-02: Unnecessary checks

Severity: Info
Category: Gas
File(s): veMuteV2.sol, line 254
Bug description: These check are redundant:
require(lock_info.amount >= 0 , "veMUTE::Redeem: INSUFFICIENT_AMOUNT");
require(lock_info.vote_weight >= 0 , "veMUTE::Redeem: INSUFFICIENT_MINT_AMOUNT");
Both lock_info.amount and lock_info.vote_weight are uint256 and thus the condition is always met.
Mute's response: ***Fixed*** in the final commit fffb677.

Info-03: Unnecessary multiplication and division

Severity: Info
Category: Gas
File(s): veMuteV2.sol, line 187
Bug description: The following expression is unnecessarily complicated:
_amount.mul(_lock_time.mul(10**18).div(max_lock)).div(10**18);
It can be replaced with this alternate expression:
_amount.mul(_lock_time).div(max_lock);
As all the values are too small for an overflow-caused revert to occur, the alternate expression would not only be simpler but also more accurate.
Mute's response: ***Fixed*** in the final commit fffb677.

Info-04: Redundant function call

Severity: Info
Category: Gas
File(s): veMuteV2.sol, line 546
Bug description: The delegates() function is called to return the value of _delegation[tokenId], which was set to be the "to" variable in the preceding line.
Mute's response: ***Fixed*** in the final commit fffb677.

Info-05: No reentrancy guard on the delegate() and delegateBySig() functions

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol, line 482
Bug description: As both delegate() and delegateBySig() are state-changing functions, it is advisable to add the nonReentrant modifier.
Mute's response: ***Fixed*** in the final commit fffb677.

Info-06: It is possible to split an NFT into an equivalent NFT and a NFT which does not hold any value

Severity: Info
Category: Inconsistent behavior
File(s): veMuteV2.sol, line 285
Bug description: Even though calling the Split() function with the parameter _amount = _nftLocks[_tokenId].amount is disallowed, it is still possible to call it with _amount = 0. This is an inconsistent behavior as these options are essentially equivalent.
Mute's response: ***Fixed*** in the final commit fffb677.

Info-07: It is possible to accidentally burn tokens using RedeemTo()

Severity: Info
Category: Potential mistake prevention
File(s): veMuteV2.sol, line 244
Bug description: When calling the RedeemTo() function, one can accidentally set the 'to' address as the zero address, thus resulting in the loss of all the tokens requested to be redeemed.
Mute's response: ***Fixed*** in the final commit fffb677.

Info-08: Constants should be in CONSTANT_CASE

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol, lines 31, 32
Bug description: For constant variable names, each word should use all capital letters, with underscores separating each word (CONSTANT_CASE)
Mute's response: ***Fixed*** in the final commit fffb677.

Info-09: Constants should be defined rather than using magic numbers

Severity: Info
Category: Deviation from best practices

File(s): MuteGovernor.sol, lines 32, 36
Bug description: Even assembly can benefit from using readable constants instead of hex/numeric literals
Mute's response: ***Fixed*** in the final commit fffb677.

Info-10: Control structures do not follow the Solidity Style Guide

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol, lines 50, 197, 299, 346, 393
Bug description: See the control structures section of the Solidity Style Guide
Mute's response: ***Acknowledged***.

Info-11: Default Visibility

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol, lines 31, 32
Bug description: Some constants are using the default visibility. For readability, consider explicitly declaring them as internal.
Mute's response: ***Fixed*** in the final commit fffb677.

Info-12: Consider disabling renounceOwnership()

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol, line 17
Bug description: If the plan for your project does not include eventually giving up all ownership control, consider overwriting OpenZeppelin's Ownable's renounceOwnership() function in order to disable it.
Mute's response: ***Fixed*** in the final commit fffb677.

Info-13: Event is never emitted

Severity: Info
Category: Gas
File(s): veMuteV2.sol, line 70
Bug description: The following are defined but never emitted. They can be removed to make the code cleaner and lighter.
event TransferLockToEvent(address from, address to, uint256 unlockedAmount, uint256 burnAmount);
Mute's response: ***Fixed*** in the final commit fffb677.

Info-14: Event missing indexed field

Severity: Info
Category: Deviation from best practices

File(s): veMuteV2.sol, lines 68-72
Bug description: Index event fields make the field more quickly accessible to off-chain tools that parse events. This is especially useful when it comes to filtering based on an address. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Where applicable, each event should use three indexed fields if there are three or more fields and gas usage is not particularly of concern for the events in question. If there are fewer than three applicable fields, all of the applicable fields should be indexed.
Mute's response: ***Fixed*** in the final commit fffb677.

Info-15: Function ordering does not follow the Solidity style guide

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol, MuteGovernor.sol
Bug description: According to the Solidity style guide, functions should be laid out in the following order: constructor(), receive(), fallback(), external, public, internal, and private, but the contract files do not follow this pattern.
Mute's response: ***Acknowledged***.

Info-16: Change uint to uint256

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol lines 66, 630
Bug description: Throughout the code base, some variables are declared as uint. To favor explicitness, consider changing all instances of uint to uint256
Mute's response: ***Fixed*** in the final commit fffb677.

Info-17: Interfaces should be defined in separate files from their usage

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol lines 629, 633
Bug description: The interfaces below should be defined in separate files, so that it's easier for future projects to import them, and to avoid duplication later on if they need to be used elsewhere in the project
Mute's response: ***Acknowledged***

Info-18: NatSpec is completely non-existent on functions that should have them

Severity: Info
Category: Deviation from best practices

File(s):veMuteV2.sol lines 140, 194, 203, 207, 224, 240, 244, 276, 320
MuteGovernor.sol lines 46, 55
Bug description: Public and external functions that aren't view or pure should have NatSpec comments
Mute's response: ***Acknowledged***

Info-19: Use a modifier instead of a require/if statement for a special msg.sender actor

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol lines 141, 225, 252
Bug description: If a function is supposed to be access-controlled, a modifier should be used instead of a require/if statement for more readability.
Mute's response: ***Acknowledged***

Info-20: Consider using named mappings

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol lines 42, 43, 56, 59, 64
Bug description: Consider moving to solidity version 0.8.18 or later, and using named mappings to make it easier to understand the purpose of each mapping
Mute's response: ***Acknowledged***

Info-21: Adding a return statement when the function defines a named return variable, is redundant

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol lines 363, 367, 382
Bug description: Consider moving to solidity version 0.8.18 or later, and using named mappings to make it easier to understand the purpose of each mapping
Mute's response: ***Acknowledged***

Info-22: Deprecated library used for Solidity >= 0.8: SafeMath

Severity: Info
Category: Gas
File(s): veMuteV2.sol lines 13, 18
Bug description: SafeMath is generally not needed starting with Solidity 0.8, since the compiler now has built in overflow and underflow checking.
Mute's response: ***Acknowledged***

Info-23: Use scientific notation (e.g. 1e18) rather than exponentiation (e.g. 10**18)

Severity: Info
Category: Deviation from best practices
File(s):veMuteV2.sol line 187
MuteGovernor.sol line 36
Bug description: While this won't save gas in the recent solidity versions, this is shorter and more readable (this is especially true in calculations).
Mute's response: ***Fixed*** in the final commit fffb677.

Info-24: Strings should use double quotes rather than single quotes

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol lines 77, 216
Bug description: See the Solidity Style Guide:
https://docs.soliditylang.org/en/v0.8.20/style-guide.html#other-recommendations
Mute's response: ***Fixed*** in the final commit fffb677.

Info-25: Contract does not follow the Solidity style guide's suggested layout ordering

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol
Bug description: The style guide says that, within a contract, the ordering should be:

1. Type declarations
2. State variables
3. Events
4. Modifiers
5. Functions

However, the contract does not follow this ordering.
Mute's response: ***Acknowledged***.

Info-26: Internal and private variables and functions names should begin with an underscore

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol lines 28, 46, 66, 147, 163, 176, 182
Bug description: According to the Solidity Style Guide, Non-external variable and function names should begin with an underscore. Also please note that the _nftLocks variable (line 42) is declared as public while its name begins with an underscore.
Mute's response: ***Acknowledged***.

Info-27: Usage of floating pragma is not recommended

Severity: Info
Category: Deviation from best practices
File(s):veMuteV2.sol line 2
MuteGovernor.sol line 2
Bug description: If you leave a floating pragma in your code, you won't know which version was deployed to compile your code, leading to unexpected behavior.
Mute's response: ***Acknowledged***.

Info-28: public functions not called by the contract should be declared external instead

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol lines 140, 363, 370, 385
Bug description: public functions not called by the contract should be declared external instead
Mute's response: ***Acknowledged***.

Info-29: Variables need not be initialized to zero

Severity: Info
Category: Deviation from best practices
File(s): veMuteV2.sol lines 195, 227, 230, 245, 246, 374, 389
Bug description: The default value for variables is zero, so initializing them to zero is superfluous.
Mute's response: ***Fixed*** in the final commit fffb677.

Info-30: Splitting require() statements that use && saves gas

Severity: Info
Category: Gas
File(s): veMuteV2.sol lines 326, 327, 332
Bug Description: See this issue which describes the fact that there is a larger deployment gas cost, but with enough runtime calls, the change ends up being cheaper.
Saves around 3 gas per instance
Mute's response: ***Fixed*** in the final commit fffb677.

Info-31: Unchecking arithmetics operations that can't underflow/overflow

Severity: Info
Category: Gas
File(s): veMuteV2.sol lines 249, 291, 294

Bug Description: Solidity version 0.8+ comes with implicit overflow and underflow checks on unsigned integers. When an overflow or an underflow isn't possible (as an example, when a comparison is made before the arithmetic operation), some gas can be saved by using an unchecked block:
https://docs.soliditylang.org/en/v0.8.10/control-structures.html#checked-or-unchecked-arithmetic
Consider wrapping with an unchecked block where it's certain that there cannot be an underflow.
25 gas saved per instance
Mute's response: ***Acknowledged***.

Info-32: Don't use _msgSender() if not supporting EIP-2771

Severity: Info
Category: Gas
File(s): veMuteV2.sol lines 277, 321, 483
Bug Description: Use msg.sender if the code does not implement EIP-2771 trusted forwarder support
Mute's response: ***Fixed*** in the final commit fffb677.

Info-33: Comparing to a Boolean constant

Severity: Info
Category: Gas
File(s): veMuteV2.sol lines 197, 225, 299, 332, 346, 393
Bug Description: Comparing to a constant (true or false) is a bit more expensive than directly checking the returned boolean value.
Consider using if(directValue) instead of if(directValue == true) and if(!directValue) instead of if(directValue == false)
Mute's response: ***Fixed*** in the final commit fffb677.

Info-34: Cache array length outside of loop

Severity: Info
Category: Gas
File(s): veMuteV2.sol lines 195, 227, 248
Bug Description: If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first), and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).
Mute's response: ***Acknowledged***.

Info-35: Use calldata instead of memory for function arguments that do not get mutated

Severity: Info
Category: Gas

File(s): veMuteV2.sol lines 194, 240, 244
Bug Description: When a function with a memory array is called externally, the abi.decode() step has to use a for-loop to copy each index of the calldata to the memory index. Each iteration of this for-loop costs at least 60 gas (i.e. 60 * <mem_array>.length). Using calldata directly bypasses this loop.
If the array is passed to an internal function which passes the array to another internal function where the array is modified and therefore memory is used in the external call, it's still more gas-efficient to use calldata when the external function uses modifiers, since the modifiers may prevent the internal functions from being called. Structs have the same overhead as an array of length one.
Saves 60 gas per instance
Mute's response: ***Acknowledged***.

Info-36: Use Custom Errors instead of Revert Strings to save Gas

Severity: Info
Category: Gas
File(s): veMuteV2.sol lines 141, 183-184, 208, 225, 252-255, 280-281, 285, 325-327, 332, 417, 438, 455, 484, 509, 516-519
Bug Description: Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also saves deployment gas. Additionally, custom errors can be used inside and outside of contracts (including interfaces and libraries).
Source: https://blog.soliditylang.org/2021/04/21/custom-errors/:

> Starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. Until now, you could already use strings to give more information about failures (e.g., revert("Insufficient funds.");), but they are rather expensive, especially when it comes to deploy cost, and it is difficult to use dynamic information in them.

Consider replacing all revert strings with custom errors in the solution, particularly those that have multiple occurrences.
Mute's response: ***Acknowledged***.

Info-37: State variables only set in the constructor should be declared immutable

Severity: Info
Category: Gas
File(s): veMuteV2.sol lines 133, 134
Bug Description: Variables only set in the constructor and never edited afterward should be marked as immutable, as it would avoid the expensive storage-writing operation in the constructor (around 20 000 gas per variable) and replace the expensive storage-reading operations (around 2100 gas per reading) to a less expensive value reading (3 gas)
Mute's response: ***Acknowledged***.

Info-38: ++i costs less gas compared to i++ or i += 1 (same for --i vs i-- or i -= 1)

Severity: Info
Category: Gas
File(s): veMuteV2.sol lines 195, 227, 230, 248, 374, 389
Bug Description: Pre-increments and pre-decrements are cheaper.

For a uint256 i variable, the following is true with the Optimizer enabled at 10k:

Increment:
i += 1 is the most expensive form
i++ costs 6 gas less than i += 1
++i costs 5 gas less than i++ (11 gas less than i += 1)

Decrement:
i -= 1 is the most expensive form
i-- costs 11 gas less than i -= 1
--i costs 5 gas less than i-- (16 gas less than i -= 1)

Note that post-increments (or post-decrements) return the old value before incrementing or decrementing, hence the name post-increment:
uint i = 1;
uint j = 2;
require(j == i++, "This will be false as i is incremented after the comparison");

However, pre-increments (or pre-decrements) return the new value:
uint i = 1;
uint j = 2;
require(j == ++i, "This will be true as i is incremented before the comparison");

In the pre-increment case, the compiler has to create a temporary variable (when used) for returning 1 instead of 2.

Consider using pre-increments and pre-decrements where they are relevant (meaning: not where post-increments/decrements logic are relevant).
Saves 5 gas per instance
Mute's response: ***Acknowledged***.

Info-39: Use a more recent version of Solidity

Severity: Info
Category: Gas
File(s):veMuteV2.sol line 2
MuteGovernor.sol line 2
Bug Description: We recommend using a more recent version of solidity. Explicitly, when running the Certora prover, we used 0.8.19.

0.8.19: SMTChecker: New trusted mode that assumes that any compile-time available code is the actual used code, even in external calls. Bug Fixes:

- Assembler: Avoid duplicating subassembly bytecode where possible.
- Code Generator: Avoid including references to the deployed label of referenced functions if they are called right away.
- ContractLevelChecker: Properly distinguish the case of missing base constructor arguments from having an unimplemented base function.
- SMTChecker: Fix internal error caused by unhandled z3 expressions that come from the solver when bitwise operators are used.
- SMTChecker: Fix internal error when using the custom NatSpec annotation to abstract free functions.
- TypeChecker: Also allows external library functions in using for.

Mute's response: ***Acknowledged***.

Info-40: Increments/decrements can be unchecked in for-loops

Severity: Info
Category: Gas
File(s): veMuteV2.sol lines 195, 227, 230, 248, 374, 389
Bug Description: In Solidity 0.8+, there's a default overflow check on unsigned integers. It's possible to uncheck this in for-loops and save some gas at each iteration, but at the cost of some code readability, as this uncheck cannot be made inline.
ethereum/solidity#10695

The change would be:

```
- for (uint256 i; i < numIterations; i++) {
+ for (uint256 i; i < numIterations;) {
 // ...
+   unchecked { ++i; }
}
```

These save around 25 gas saved per instance.

The same can be applied with decrements (which should use break when i == 0).
The risk of overflow is non-existent for uint256.
Mute's response: ***Acknowledged***.

CERTORA

Assumptions and Simplifications Made During Verification

We made the following assumptions during our verification:

- Loop unrolling: We assume any loop can have at most 3 iterations.
- The return value of the function GetUserLockLength() called in LockLegacy() is always 1, for simplicity
- Some of the rules assume the system starts from a valid state, which we described as: for non-legacy tokens, the locked amount is greater or equal to the vote weight received
- There are less than MAX_UINT256 - 10 NFTs already minted (to avoid overflow)
- The maximum lockable amount is less than 40,000,000 $MUTE (as this is the MUTE's max supply)
- In MuteGovernor.sol we simplified the hashProposal() function by limiting its input to arrays of the length of 1
- In MuteGovernor.sol we simplified the propose() function by limiting its input to arrays of the length of 1

Notations

✅Indicates the rule is formally verified.
❌Indicates the rule is violated.
⏳Indicates the rule is timing out.
The rule's name will be shown as: (`ruleName`)

Formal Verification Properties

1. ✅ One cannot lock a MUTE token for more than 52 weeks or less than 1 week (`canLockOnlyWithinLegalTimeRange`)
2. ✅ One cannot lock zero MUTE tokens (`cannotLockZeroMUTE`)
3. ✅ One cannot lock more MUTE tokens than what he has (`cannotLockMoreMUTEThanAvailable`)
4. ✅ Once a user locks X MUTE tokens, his available MUTEToken balance is reduced by X and the balance of the veMuteV2 contract is increased correctly by X (`lockingMUTECorrectlyUpdatesBalances`)
5. ✅ The _tokenIds can only increase (`tokenIdsCanOnlyIncrease`)
6. ✅ When one locks MUTEtokens he must get NFT with correct parameters up to vote_weight deviation (`lockIntegrity`)

7. ✅ When one locks MUTEtokens he must get NFT with a correct vote_weight up to 99.99% (`lockIntegrityVoteWeight`)
8. ✅ As long as there is no redemption, the sum of balances (of the sender and the veMuteV2Contract) of MUTEToken cannot change (`withoutRedemptionTheSumOfBalancesIsTheSame`)
9. ✅ One cannot LockTo() the zero address (`cannotLockToTheZeroAddress`)
10. ✅ If a user locked X MUTE tokens for Y time, he can redeem ALL of them ONLY after Y time passed (`canRedeemTokensFullyOnlyAfterLockTimePassed`)
11. ✅ If user A locked MUTE tokens and did nothing else, then only he can redeem them (`onlySameUserCanRedeemHisTokens`)
12. ✅ One cannot consecutively redeem to self twice the same NFT (`cannotConsecutivelyRedeemTwiceTheSameNFT`)
13. ✅ One cannot redeem to self twice the same NFT, any method can be called after the initial redemption (`cannotRedeemTwiceTheSameNFTAdvanced`)
14. ✅ One cannot redeem to any two addresses the same NFT twice (`cannotRedeemToTwoDifferentAddressesTheSameNFT`)
15. ✅ One cannot redeem twice the same NFT by sending an array of NFTs that contains the same tokenId (`cannotRedeemTwiceTheSameNFTInTheSameCall`)
16. ❌ One cannot redeem to the zero address (`cannotRedeemToTheZeroAddress`) see Info-07: It is possible to accidentally burn tokens using RedeemTo()
    Note: this rule passes in the final commit fffb677, therefore we verified the issue is fixed.
17. ✅ One cannot create/modify an NFT so that its UserLockInfo.vote_weight > UserLockInfo.amount (excluding LockLegacy()) (`cannotGetMoreVoteWeight2NonLegacy`)
18. ✅ Manipulations of an NFT preserve its legacy and time (`cannotChangeLegacyOrReduceLockTime`)
19. ✅ The same user cannot call LockLegacy() twice (`sameUserCannotCallLockLegacyTwice`)
20. ✅ Calling LockLegacy() only affects the tokens of the msg.sender (`callingLockLegacyDoesNotAffectOtherUsers`)
21. ✅ Split creates two consecutive tokenIds that are larger than the split tokenId (`splitIncreasesTokenIdsConsecutively`)
22. ✅ The sum of the resulting split amounts is the same as the amount before split (`splitPreservesAmountVotesTimeAndLegacy`)

23. ✅ The sum of the resulting split vote_weight is the same as before split (`splitPreservesAmountVotesTimeAndLegacy`) - same as rule 22
24. ✅ The resulting split time and legacy is the same (`splitPreservesAmountVotesTimeAndLegacy`) - same as rule 22
25. ✅ Splitting an NFT doesn't affect any other already existing NFTs (`splittingNFTDoesNotAffectOtherExistingNFTs`)
26. ✅ Once an NFT is split, it cannot be split again (`cannotSplitTwiceTheSameNFT`)
27. ✅ Once an NFT is split, it does not exist anymore, i.e., is owned by address(0) (`onceSplitNFTIsNotOwnedByAnyUser`)
28. ✅ Splitting an NFT does not change the total voteSupply (`splittingDoesNotChangeTotalVoteSupply`)
29. ✅ Only owner or approved user can split an NFT (`onlyOwnerOrApprovedCanSplit`)
30. ✅ msg.sender becomes the owner of both the splits (`msgSenderIsOwnerOfSplits`)
31. ❌ Cannot split an NFT with _amount = 0 (`cannotSplitWithZeroAmount`) see Info-06: It is possible to split an NFT into an equivalent NFT and a NFT which does not hold any value
    Note: this rule passes in the final commit fffb677, therefore we verified the issue is fixed.
32. ✅ One cannot split a legacy NFT using the Split() function (`cannotSplitALegacyNFT`)
    Note: the passing of this rule confirms a bug since one should be able to split a legacy NFT. In the final commit fffb677, a new function SplitLegacy() was introduced that allows for the splitting of legacy NFTs.
33. ✅ If there are no legacy tokens, the only way to get a legacy token is by calling the LockLegacy() function (`onlyLockLegacyCanCreateNewLegacyTokens`)
    Note: this rule fails in the final commit fffb677, because two new functions were introduced: LockLegacyIndex() and SplitLegacy() that can generate a legacy token. The new failure is expected for those new functions.
34. ✅ One cannot merge the same token with itself (`cannotMergeNFTWithItself`)
35. ✅ One cannot merge a legacy token (`mergeRevertsCorrectly`)
36. ✅ Cannot merge two tokens that are not either owned by or approved to the same user (`mergeRevertsCorrectly`) - same as rule 35
37. ✅ Cannot merge two tokens if one of them is owned by the zero address (`mergeRevertsCorrectly`) - same as rule 35

38. ✅ After merging, the resulting token has a tokenId greater than each of the merged (`mergeIntegrity`)
39. ✅ After merging, the resulting token has the longest expiry date of both the merged token (`mergeIntegrity`) - same as rule 38
40. ✅ After merging, the resulting token is owned by the msg.sender (who is either approved by or owner of the merged tokens (`mergeIntegrity`) - same as rule 38
41. ✅ After merging, the resulting token has an amount equal to the sum of the merged amounts (`mergeIntegrity`) - same as rule 38
42. ✅ After merging, the resulting token has vote_weight equal to the sum of the merged vote_weights (`mergeIntegrity`) - same as rule 38
43. ✅ After merging, the resulting token must not be legacy since it is not allowed to merge legacy tokens (`mergeIntegrity`) - same as rule 38
44. ✅ If one splits an NFT and then merges the two resulting NFTs he must get the same parameters (up to tokenId) (`splitAndMergeResultsTheSame`)
45. ✅ The total vote supply does not change after merging two tokens (`voteSupplyDoesNotChangeAfterMerge`)
46. ✅ Only the token owner (or approved by the owner) can delegate his votes (`delegateIntegrity`)
47. ✅ After delegation, the new delegatee is updated correctly (got by delegates(tokenId)) (`delegateIntegrity`) - same as rule 46
48. ✅ After delegation of a specific NFT, only the delegatee of that NFT can change (`delegateIntegrity`) - same as rule 46
49. ✅ After delegation, the new delegatee has exactly the NFT vote_weight more votes (got by getVotes(delegatee)) (`delegateIntegrity`) - same as rule 46
50. ✅ After delegation, the old delegatee loses exactly the NFT vote_weight (`delegateIntegrity`) - same as rule 46
51. ✅ Delegation cannot change the total vote supply (`voteSupplyDoesNotChangeAfterDelegation`)
52. ✅ When one locks MUTE tokens, the voteSupply must increase by the minted vote_weight (`lockIncreasesVoteSupply`)
53. ✅ One can lock two consecutive times (`sameUserCanCallLockTwice`)
54. ✅ The length of the legacyNFTs array is increased when splitting a legacy NFT (`splitIncreasesLengthOfLegacyNFTsArray`)

55. ✅ When one redeems two NFTs, he is redeemed the correct amount, i.e., the sum of the amounts stored in the redeemed NFTs (`RedeemToIntegrity`)
56. ✅ Once a propose is accepted, the only allowed voting period is from propose_time + votingDelay() until propose_time + votingDelay() + votingPeriod(), any votingCasts outside this time frame should revert (`cannotVoteOutsideLegalTimeFrame`)
57. ✅ To offer a proposal, the proposer must have at least proposalThreshold() votes (`proposerMustHaveAtLeastMoreVotesThanThreshold`)
58. ✅ If a proposal was not made, no one can castVote() for it (`cannotCastVoteOnEmptyPropose`)
59. ✅ One cannot make the exact same proposal twice (`oneCannotMakeTheSameProposeTwice`)
60. ✅ After casting a vote, the proposal can pass only if the number of cast votes > quorum (`castingCanPassPropose`)
61. ✅ Once an account has cast a vote, he cannot cast an additional vote for the same proposal (`cannotCastSameVoteTwice`)

Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification and the Certora Prover does not check any cases not covered by the specification.

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.