METALEX METAVEST SECURITY AUDIT REPORT

Nov 13, 2024

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

MetaLeX protocol allows for the customization of Gnosis Safe multisigs to be used as a committee for a DAO. This customization involves restricting actions for the specific multisig via a special Guard contract, which serves as a filter for all transactions from the multisig. Apart from the Guard contract, MetaLeX offers a variety of different implants that can be installed into the Safe contract to extend its functionality. An example of such an extension is an implant that enables multisig owners to create grants.

1.4 Project Dashboard

Project Summary

Title	Description
Client	MetaLeX
Project name	MetaVesT
Timeline	03.10.2024 - 31.10.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
03.10.2024	b614405e60bce8b852e46d06c03fd47b04d86dde	Commit for the audit
20.10.2024	1bae6630e222e9bf2527cefac75e5e4f802ec78c	Commit for the re-audit
25.10.2024	9799c3cf8b1ddd55350d59e23ee5400dec50f560	Commit for the re-audit 2
25.10.2024	2edc351663cff8ae0b1e31950d6ef9fe93a1af5f	Commit with updates
31.10.2024	ae78cb4eec19480ed9fde23e527e67a98c4a4f9a	Commit with updates 2

Project Scope

The audit covered the following files:

File name	Link
src/BaseAllocation.sol	BaseAllocation.sol

File name	Link
src/MetaVesTController.sol	MetaVesTController.sol
src/MetaVesTFactory.sol	MetaVesTFactory.sol
src/RestrictedTokenAllocation.sol	RestrictedTokenAllocation.sol
src/RestrictedTokenFactory.sol	RestrictedTokenFactory.sol
src/TokenOptionAllocation.sol	TokenOptionAllocation.sol
src/TokenOptionFactory.sol	TokenOptionFactory.sol
src/VestingAllocation.sol	VestingAllocation.sol
src/VestingAllocationFactory.sol	VestingAllocationFactory.sol
src/lib/EnumberableSet.sol	EnumberableSet.sol

Deployments

File name	Contract deployed on mainnet	Comment
src/VestingAllocationFactory.sol	0x297163...Ae5D318E	
src/TokenOptionFactory.sol	0xa6a485...638BA4B6	
src/RestrictedTokenFactory.sol	0x8Ec6b6...e5DC1BE5	
src/MetaVesTController.sol	0x4E12E3...19846677	
src/MetaVesTFactory.sol	0x854E7F...9A7E2417	

1.5 Summary of findings

Severity	# of Findings
Critical	2
High	6
Medium	9
Low	23

ID	Name	Severity	Status
C-1	Tokens Can Get Stuck on the `RestrictedTokenAllocation` Contract	Critical	Fixed
C-2	Potential Blockage of Funds if Terminated Before Vesting Start	Critical	Fixed
H-1	Possible Reverts Due to Underflow in Case if `vestingRate` or `unlockRate` was Reduced	High	Fixed
H-2	Voting Logic Issue: Lack of Handling for Amendments without Reached Quorum	High	Fixed
H-3	Missing Call to `resetAmendmentParams` in `setMetaVesTGovVariables`	High	Fixed
H-4	Incorrect Handling of `tokensRepurchased` in `repurchaseTokens`	High	Fixed
H-5	Issues with MetaVesTs Having Different Decimals in the Same Set	High	Acknowledged
H-6	Exercised but Not Unlocked Tokens Will Be Sent to Authority During `recoverForfeitTokens()`	High	Fixed
M-1	Incomplete Check on Milestone Completion in `addMetavestMilestone`	Medium	Fixed

M-2	Incomplete Validation of `callData` in `proposeMajorityMetaVesTAmendment`	Medium	Fixed
M-3	Lack of Validation for `shortStopTime` and `shortStopDuration` in `updateStopTimes`	Medium	Acknowledged
M-4	Incorrect Handling of `milestoneUnlockedTotal` in `getUnlockedTokenAmount`	Medium	Acknowledged
M-5	Not All the Tests are Passing	Medium	Fixed
M-6	Several Issues with Consents Obtained Through Voting	Medium	Fixed
M-7	Inefficient Storage and Iteration Over MetaVesT Sets	Medium	Fixed
M-8	Inability to Repurchase Tokens Due to Low Repurchase Price in `RestrictedTokenAward`	Medium	Acknowledged
M-9	Possible reverts due to early vesting termination	Medium	Fixed
L-1	Unclear Usage of the `prevOwners` Variable	Low	Fixed
L-2	Milestone Index Validation Before Access	Low	Fixed
L-3	Inefficient Hash Check in `consentCheck` Modifier	Low	Acknowledged
L-4	Lack of Uniqueness Check in `updateFunctionCondition`	Low	Fixed
L-5	Unused Parameter `_longStopDate` in `createMetaVesT` Function	Low	Acknowledged
L-6	Redundant `conditionCheck` Modifier in `createVestingAllocation` Function	Low	Fixed
L-7	Lack of Condition Limit Check in `addMetaVesTMilestone`	Low	Fixed
L-8	Ambiguous Return Value in `getSetOfMetaVesT` Function	Low	Fixed
L-9	Unused Variable `ipaymentToken`	Low	Fixed

L-10	Uninitialized `tokensToRecover` and Unused `milestonesAllocation` in `terminate` Function	Low	Fixed
L-11	Token Decimals Consideration in Rate Check	Low	Acknowledged
L-12	Missing Validation for `vestingStartTime` and `unlockStartTime` in All the Allocation's Contracts' Constructor	Low	Acknowledged
L-13	Unnecessary Token Balance Check	Low	Fixed
L-14	Incorrect Variable Naming	Low	Fixed
L-15	Voter is Added to `proposal.voters` List Even if They do not Have Voting Power	Low	Acknowledged
L-16	Creation of `MajorityAmendmentProposal` is Possible for a Non-existent `setName`	Low	Fixed
L-17	Unused Errors, Events and Variables	Low	Fixed
L-18	Missing Zero Check of Milestone Award	Low	Fixed
L-19	`removeMetaVestFromSet()` Does Nothing if MetaVesT is Not in the Set	Low	Fixed
L-20	Mappings `vestingAllocations`, `restrictedTokenAllocations`, and `tokenOptionAllocations` are not updated if the grantee's address was changed	Low	Fixed
L-21	`transferRights()` requires a two-step process to change the address	Low	Fixed
L-22	Missing Minimum Exercise Price Check for `TokenOptionAllocation`	Low	Acknowledged
L-23	Unused `getGoverningPower` function and `govType` variable	Low	Acknowledged

1.6 Conclusion

The primary objective of the audit was to verify that the vesting system functions correctly, ensuring tokens are distributed to users in accordance with the business logic. Aside from the findings presented in the report, we examined the following attack vectors and potential issues.

Voting for New Amendments and the Consent System in `MetaVesTController`: It is impossible to vote for non-existent amendments. It is also not possible to manipulate transaction calldata to send calldata with a length not equal to 68 bytes to the `consentCheck()` modifier. All updates to the voting power will not affect the voting process because, for voting purposes, the protocol creates a slice of the current voting powers and stores them. Future voting for the same `msg.sig` for the same `set` functions correctly.

Vesting Lifecycle: Tokens are correctly transferred to the allocation contract balance upon the creation of a MetaVesT. The amount of withdrawn tokens cannot exceed the vested amount. Token decimals are accurately accounted for in the exercise or repurchase price calculations in the `TokenOptionAllocation` and `RestrictedTokenAllocation` contracts. However, it is worth noting that the exercise or repurchase price should be set in the `tokenContract` decimals.

Working with Milestones: It is impossible to remove a confirmed milestone. Unconfirmed milestones will be processed correctly during termination.

Management Part: Only the `authority` can create new MetaVesTs using the `MetaVesTController`. There are no possible reentrancy attacks, as all critical functions are protected with a `nonReentrant` modifier.

It is also crucial to highlight how the protocol logic is implemented:

1. If, after redeeming a portion of the tokens in `RestrictedTokenAward`, the remaining balance results in `getPaymentAmount()` equaling 0, that remainder cannot be withdrawn from the contract.
2. If `_inFavor == false` in the `MetaVesTController.voteOnMetaVesTAmendment()` call, a user can vote again on the same amendment. However, if `_inFavor` is true, the function cannot be called again.
3. Once the majority has voted in favor of the amendment, it can be applied to all grants in the set.
4. To claim the remaining tokens that were not vested in the `RestrictedTokenAllocation`, the authority must purchase tokens at the price set before termination.
5. After the termination of the allocation, the unlock rate cannot be changed. If it is sufficiently low, users will have to wait for a long time for their tokens to unlock.

Despite the overall architectural correctness, the findings section outlines several recommendations that should be implemented for protocol to function optimally.

2.FINDINGS REPORT

2.1 Critical

C-1	Tokens Can Get Stuck on the `RestrictedTokenAllocation` Contract
Severity	Critical
Status	Fixed in 2edc3516

Description

The issue has been identified within the `RestrictedTokenAllocation` contract. If the unlock rate is reset to zero (which is a MetaVesTController.sol#L464 to enable temporary freezes) and the vesting contract is subsequently terminated, some tokens may get stuck in the contract. This occurs because the repurchase amount of tokens is calculated using the RestrictedTokenAllocation.sol#L190 function, which can exceed the unlocked amount of tokens. However, `getAmountWithdrawable()` takes the minimum of vested and unlocked token amounts to determine the withdrawable balance. As a result, `tokensWithdrawn` would be larger than `_min(_tokensVested, _tokensUnlocked)` if we assume there were previous withdrawals and the unlock rate was reduced.

This issue is classified as **Critical** severity because it renders the stuck tokens impossible to withdraw after the vesting has been terminated.

Recommendation

We recommend disallowing the unlock rate to being reset to zero or implementing the necessary checks to ensure that all tokens can be withdrawn from the vesting.

Client's commentary

> Client: We have updated to handle the case when the rate has changed, to ensure all tokens can still be withdrawn from vesting.
> MixBytes: The mentioned issue is still present for the all vesting contracts. If `unlockRate` was set to zero and vesting was terminated right afterwards, it will prevent the grantee from withdrawing already vested tokens (as the `withdrawableAmount` at RestrictedTokenAllocation.sol#L224 would be equal to zero).
> We recommend enhancing the vesting/unlock rate update logic by adding special checkpoint variables to store the amounts already vested and unlocked at the time of the rate update. After the rate is

updated, newly vested and unlocked tokens should be calculated using the checkpoint time as the starting point. This approach will prevent the reduction of already vested and unlocked token amounts.
Client: This is how we intended changing the rate to 0 to operate. There needs to be mutual (or majority in the case of a set) consent for this to occur, and would only be applied in some extreme situations. Once both parties have agreed to this change, it effectively retroactively suspends vesting or unlocking until the rate is adjusted again. Not a common case as the grantee would need to agree to it. It is not intended to track or store vesting/unlocking up until this has changed.
MixBytes: We recommend implementing a special `emergencyUpdateMetaVesTUnlockRate` function to the `MetaVesTController` contract, which will allow the update of the `unlockRate` if the vesting has already been terminated. It should call a new function in the `BaseAllocation` contract `emergencyUpdateUnlockRate` , which should include the check `terminated == true && unlockRate == 0`, so that it can be called only in this particular case when the `unlockRate` is zero. It is important to mention that the `emergencyUpdateMetaVesTUnlockRate` function should have the following modifiers: `onlyAuthority conditionCheck`.
Client: We have added the recommended emergencyUpdateMetavestUnlockRate as described above to handle the case of termination with a 0 unlock rate.

C-2	Potential Blockage of Funds if Terminated Before Vesting Start
Severity	Critical
Status	Fixed in 1bae6630

Description

This issue has been identified in the `getVestedTokenAmount` function of the RestrictedTokenAllocation.sol#L201 contract.
If the restricted token allocation is terminated prior to the vesting start time, the current implementation will lock the funds in the contract without providing a way to claim them. This can lead to a situation where vested tokens remain inaccessible to the grantee and authority, causing a potential loss to the protocol.
The issue is classified as **Critical** severity because it directly affects the protocol's ability to access its funds in case of early termination of the vesting schedule.

Recommendation

We recommend adding a check to ensure that if the termination occurs before the vesting start time, all the funds are immediately transferred to the authority. This will prevent the locking of funds in the contract in such scenarios.

Client's commentary

> We have added checks to ensure that terminations before the vesting start time properly recovery funds to the authority.

2.2 High

H-1	Possible Reverts Due to Underflow in Case if `vestingRate` or `unlockRate` was Reduced
Severity	High
Status	Fixed in 9799c3cf

Description

The issue was identified within the TokenOptionAllocation.sol#L180 of the `TokenOptionAllocation` contract. If `vestingRate` is reduced, the calculated `_tokensVested` may lead to a revert at TokenOptionAllocation.sol#L193 as there could be more `tokensExercised` than the recalculated vested amount.

Another issue was identified within the same `TokenOptionAllocation` contract. If `unlockRate` is reduced, then a call to the TokenOptionAllocation.sol#L213 function will return a smaller value, which may be less than `tokensExercised` and `tokensWithdrawn`, leading to an underflow during subtraction.

This issue is classified as **High** severity because it will lead to unexpected reverts in the `getAmountWithdrawable()`, `getUnlockedTokenAmount()`, and `terminate()` functions. A similar issue is also present in the RestrictedTokenAllocation.sol#L230 function of the `RestrictedTokenAllocation` contract. The only way to resolve the issue is by increasingboth the `vestingRate` and the `unlockRate`.

Recommendation

We recommend disallowing reductions in the `vestingRate` and `unlockRate`, or implementing necessary checks to ensure that all tokens can be withdrawn from vesting.

Client's commentary

> Client: We have added checks to ensure all tokens can be withdrawn from vesting after vesting or unlock rates have been reduced.
> MixBytes: There is a redundant `return` operator at TokenOptionAllocation.sol#L214, which leads to the same issue described above. We recommend removing this unnecessary line.
> Client: We have removed the redundant 'return' operator.

H-2	Voting Logic Issue: Lack of Handling for Amendments without Reached Quorum
Severity	High
Status	Fixed in 1bae6630

Description

This issue has been identified in the `voteOnMetaVesTAmendment` function of the MetaVesTController.sol#L598-L618 contract.
Currently, if the voting period for an amendment extends beyond the allowed one-week timeframe without reaching the required quorum, the associated set becomes locked. This situation prevents any further changes, such as adding or removing grants, deleting the set, or proposing a new amendment, effectively blocking future governance actions for that set.
The issue is classified as **High** severity because it can lead to governance gridlock and prevent the proper functioning of the contract's voting mechanism, causing potential disruptions in decision-making.

Recommendation

We recommend updating the voting logic to include a mechanism that allows for the removal or cancellation of amendments for which the quorum was not reached within the specified timeframe. This change would ensure that sets do not become permanently blocked due to unmet quorum requirements.

Client's Commentary

> We have added cancelExpiredMajorityMetavestAmendment(string memory _setName, bytes4 _msgSig) to allow amendments that haven't reached quorum by the expiry date to be properly canceled and a new amendment to be re-proposed.

H-3	Missing Call to `_resetAmendmentParams` in `setMetaVesTGovVariables`
Severity	High
Status	Fixed in 1bae6630

Description

This issue has been identified in the `setMetaVesTGovVariables` function of the MetaVesTController.sol#L507-L509 contract.
The function does not call `_resetAmendmentParams`, which could lead to a situation where amendments cannot be created or processed properly if a previous amendment to update the governing variables was made. This oversight may result in inconsistencies or hinder the inability to proceed with governance updates as intended.
The issue is classified as **High** severity because it affects the ability to manage and update governance parameters, potentially leading to governance gridlock or an inability to adjust critical parameters.

Recommendation

We recommend adding a call to `_resetAmendmentParams` in the `setMetaVesTGovVariables` function to ensure that any previous amendments are cleared, allowing new amendments to be processed smoothly.

Client's Commentary

> We have added the missing _resetAmendmentParams to the setMetaVestGovVariables function.

H-4	Incorrect Handling of `tokensRepurchased` in `repurchaseTokens`
Severity	High
Status	Fixed in 1bae6630

Description

This issue has been identified in the `repurchaseTokens` function of the RestrictedTokenAllocation.sol#L138-L153 contract.
The `tokensRepurchased` variable is not properly accounted for in the `getAmountRepurchasable` calculation. After the first repurchase, the `getAmountRepurchasable` function will display an incorrect amount because it does not consider the tokens that have already been repurchased. This can lead to a misleading view of the number of tokens available for repurchase and potentially allow attempts to repurchase more tokens than are actually available.
The issue is classified as **High** severity because it affects the accuracy of the token repurchase logic, leading to potential miscalculations and incorrect token handling.

Recommendation

We recommend updating the `getAmountRepurchasable` function to include the `tokensRepurchased` variable in its calculations. This will ensure that the displayed amount accurately reflects the tokens that are still available for repurchase.

Client's Commentary

> We have corrected the getAmountRepurchasable function to include tokensRepurchased.

H-5	Issues with MetaVesTs Having Different Decimals in the Same Set
Severity	High
Status	Acknowledged

Description

If MetaVesTs with different decimals are included in the same set, metavests with higher decimals will have an exponential advantage over those with lower decimals MetaVesTController.sol#L583-L589, resulting in an incorrect voting process. Additionally, using `updateExerciseOrRepurchasePrice()`, `updateMetavestUnlockRate()`, and `updateMetavestVestingRate()` in the case of sets with different decimals may lead to errors.

This issue is classified as **High**, as the voting functionality works incorrectly in certain cases, and fixing it requires redeployment.

Recommendation

We recommend prohibiting the inclusion of MetaVesTs with different decimals in the same set.

Client's commentary

> Different decimals for vesting tokens will not occur due to business logic of how Sets will be used. It will be up to the BORG to ensure proper Set creation and management. We did not include additional checks for this.

H-6	Exercised but Not Unlocked Tokens Will Be Sent to Authority During `recoverForfeitTokens()`
Severity	High
Status	Fixed in 9799c3cf

Description

If the authority calls `TokenOptionAllocation.recoverForfeitTokens()` for a given vesting after the termination, all tokens, except for those that are currently withdrawable, TokenOptionAllocation.sol#L174 to the authority's address. Withdrawable includes all unwithdrawn exercised and unlocked tokens. As a result, the authority will also receive tokens that have already been exercised by the grantee (the grantee has paid for them) but have not yet been unlocked.

This issue has a **High** severity level because the grantee will spend money but not receive the exercised tokens. The authority will then have to manually compensate the grantee for the tokens sent to the authority's address.

Recommendation

We recommend calculating `tokensToRecover` as:
`IERC20M(allocation.tokenContract).balanceOf(address(this)) - tokensExercised;`

Client's commentary

> Client: We have corrected the tokensToRecover calculation with the recommended change.
> MixBytes: Since some exercised tokens may already be withdrawn, we recommend using the following more accurate fix:

```
if(
  IERC20M(allocation.tokenContract).balanceOf(address(this))
    > tokensExercised - tokensWithdrawn)
    tokensToRecover =
        IERC20M(allocation.tokenContract).balanceOf(address(this))
        +
        tokensWithdrawn - tokensExercised;
```

> Client: We have updated to the new recommended fix.

2.3 Medium

M-1	Incomplete Check on Milestone Completion in `addMetavestMilestone`
Severity	Medium
Status	Fixed in 1bae6630

Description

This issue has been identified in the `addMetavestMilestone` function of the MetaVesTController.sol#L449-L461 contract.
The logic does not verify whether the `_milestone.complete` flag is set to `false` before adding a new milestone. If this check is not enforced, there could be scenarios where a milestone is mistakenly added as already completed, which would prevent its proper execution and awarding process.
The issue is classified as **Medium** severity because it could lead to mismanagement of milestone states, affecting the logic of the vesting process.

Recommendation

We recommend adding a validation to ensure that the `_milestone.complete` flag is set to `false` before adding the milestone to prevent incorrect initialization of milestones.

Client's Commentary

> We have added a check to addMetavestMilestone to ensure that the condition is not already completed/true when adding a milestone.

M-2	Incomplete Validation of `callData` in `proposeMajorityMetaVesTAmendment`
Severity	Medium
Status	Fixed in 1bae6630

Description

This issue has been identified in the `proposeMajorityMetaVesTAmendment` function of the MetaVesTController.sol#L570 contract.
The function does not check the length of the `_callData` parameter before processing, which could allow arbitrary or malformed data to be sent to the function. This lack of validation increases the risk of passing incorrect data that might lead to unintended behavior or errors. Proper validation would help ensure that only well-formed data is used in the proposal.
The issue is classified as **Medium** severity because improper handling of data could lead to errors in contract execution, impacting the reliability of the voting process.

Recommendation

We recommend adding a validation check to ensure that the length of `_callData` is exactly 68 bytes to match the expected data format, preventing arbitrary or incorrect data from being processed.

Client's Commentary

> We have added the recommended check to ensure _callData is exactly 68 bytes in the amendement proposal

M-3	Lack of Validation for `_shortStopTime` and `shortStopDuration` in `updateStopTimes`
Severity	Medium
Status	Acknowledged

Description

This issue has been identified in the `updateStopTimes` function of the RestrictedTokenAllocation.sol#L98-L102 contract and in the `updateStopTimes` function of the TokenOptionAllocation.sol#L98-L102 contract.
The function does not validate the `_shortStopTime` parameter before setting its value. If `_shortStopTime` is set to an extremely large value, it could make it practically impossible to repurchase tokens, as the stop time would be set far in the future. Proper validation should be implemented to ensure that the value is within a reasonable range.
Also, there are no limits set on the value of `shortStopDuration`, which could potentially be set to an extremely large value, leading to unintended consequences such as infinite delays or a practically unreachable stop time. Without appropriate constraints, this could impact the flexibility of the token repurchase process and cause operational issues.
The issue is classified as **Medium** severity because an unreasonably large stop time could impact the token repurchase process.

Recommendation

We recommend adding validation to ensure that `_shortStopTime` is within a reasonable range to prevent scenarios where the repurchase process becomes infeasible due to an excessively long stop duration. Also, we recommend adding reasonable upper bounds to the `shortStopDuration` parameter to ensure that it cannot be set to an excessively high value, preventing potential disruptions to the vesting and repurchase processes.

Client's Commentary

> It will be up to the BORG members to ensure the correct MetaVesT time values and we don't want to restrict possible options. We did not add additional checks.

M-4	Incorrect Handling of `milestoneUnlockedTotal` in `getUnlockedTokenAmount`
Severity	Medium
Status	Acknowledged

Description

This issue has been identified in the `getUnlockedTokenAmount` function of all the allocation's contracts. The current implementation includes `milestoneUnlockedTotal` in the calculation of unlocked tokens, even when the unlocking process is complete. This can result in an overestimation of the unlocked tokens when all tokens have already been fully unlocked.
The issue is classified as **Medium** severity because it could lead to inaccurate calculations of unlocked tokens.

Recommendation

We recommend updating the logic to exclude `milestoneUnlockedTotal` from the calculation once the unlocking process is finished, ensuring a more accurate representation of the unlocked tokens.

Client's Commentary

> This is how we intended. milestoneUnlockedTotal is incremented when a milestone is completed and has "unlockOnCompletion" set to true. The milestone award value should be added to the users vested and unlocked total when this occurs. We are not concerned about unlocked total as the unlocking rate completes, we did not make any changes to this.

M-5	Not All the Tests are Passing
Severity	Medium
Status	Fixed in 1bae6630

Description

There are some failing test cases, which make it impossible to check test coverage and prepare PoCs.

Recommendation

We recommend preparing and setting up all the tests before the protocol deployment.

Client's commentary

> We have corrected the tests and added additional for the audit responses.

M-6	Several Issues with Consents Obtained Through Voting
Severity	Medium
Status	Fixed in ae78cb4e

Description

A consent obtained through voting will also apply to MetaVesTs MetaVesTController.sol#L691 after the voting process finished and MetaVesTController.sol#L625. New set members haven't participated in the voting process, so applying a consent looks unfair in such cases.
Additionally, the consent can be applied to each MetaVesT an unlimited number of times. For example, once the authority receives consent to remove a milestone with index 0, they can remove all milestones from all MetaVesTs in the set.
This issue is classified as **Medium**, as the consent functionality works incorrectly in certain cases.

Recommendation

We recommend applying consent only to MetaVesTs that participated in the vote. Additionally, we recommend prohibiting repeated function calls protected by `consentCheck` for the same `MajorityAmendmentProposal` and MetaVesT.

Client's commentary

> Client: We have added the ability to restrict the application of an amendment only once to a MetaVesT. We do not prevent the amendment being applied to newly added MetaVesTs as the new legal agreements for that set should then match what the amendment changed in the past.
> MixBytes: The `changeApplied` flag is not reset to `false` when the new amendment proposal is created for the same vesting (for the same action). It will prevent `consentCheck` from passing. We recommend resetting the `proposal.changeApplied` flag to `false` in the `proposeMajorityMetaVesTAmendment` function.
> Client: We have implemented the suggested reset of the changeApplied flag.
> MixBytes: Several scenarios have been identified in which the removal and subsequent re-adding of metavests to the set can cause the `changeApplied` boolean flag to function inaccurately. For example, this issue arises in the following scenario:
>
> 1. The amendment for `msg.sig` successfully passed the vote.
> 2. The corresponding change was applied to the MetaVesT.
> 3. The MetaVesT was removed from the set.
> 4. During this time, a new vote was initiated for the same set and the same `msg.sig`.

5. The MetaVesT was then re-added to the set. In this case, the `changeApplied` flag remains set to `true`, even though the new amendment has not been applied to the MetaVesT.

To address this issue, we recommend a modified fix: instead of storing a boolean value, store the `MajorityAmendmentProposal.time` of the last applied proposal. For instance, this value can be saved in a mapping as `mapping(address => uint256) MajorityAmendmentProposal.appliedProposalCreatedAt`.

So, MetaVesTController.sol#L648 of `MetaVesTController._resetAmendmentParams()` changes to:
`proposal.appliedProposalCreatedAt[_grant] = proposal.time;`

MetaVesTController.sol#L145 of `MetaVesTController.consentCheck()` changes to:
`if(proposal.appliedProposalCreatedAt[_grant] == proposal.time) revert MetaVesTController_AmendmentCanOnlyBeAppliedOnce();`

In such a fix, it's also not required to clear `appliedProposalCreatedAt`, so MetaVesTController.sol#L595 is removed.
Client: We have implemented the modified fix to address the issues of using just a boolean to track.

M-7	Inefficient Storage and Iteration Over MetaVesT Sets
Severity	Medium
Status	Fixed in 1bae6630

Description

In the current implementation of MetaVesT sets, MetaVesTController.sol#L675-L680 during each `consentCheck` call. The total number of MetaVesTs grows over time, so `consentCheck` will consume increasing amounts of gas. Eventually, this may lead to reaching the gas hard limit. If, at that moment, `setMajorityVoteActive[set]` is `true` for all sets, the contract will require redeployment.
This issue is classified as **Medium**, as the inefficient approach to set storage will result in higher gas consumption, potentially leading to a contract DoS.

Recommendation

We recommend using `EnumerableSet` from OpenZeppelin for managing sets.

Client's commentary

> Client: We have updated with the recommended fix and now use OpenZepplin's EnumerableSet for managing sets.
> MixBytes: The `removeSet` function can be optimized further. It is advisable to start removing elements from the last element in the set (`sets[nameHash].remove(sets[nameHash].at(length - 1))`). This approach avoids unnecessary internal element swaps (during the element removal, as the last array element is swapped with the first one during removal).
> Client: We have implemented the suggested removeSet optimization

M-8	Inability to Repurchase Tokens Due to Low Repurchase Price in `RestrictedTokenAward`
Severity	Medium
Status	Acknowledged

Description

In the `RestrictedTokenAward` contract, if the repurchase price RestrictedTokenAllocation.sol#L107-L111, RestrictedTokenAllocation.sol#L146 to repurchase tokens after the `terminate()` function is called. This occurs because the contract calculates the repurchase amount based on the price, and if the price is too low, the resulting `repurchaseAmount` could end up being zero. This would prevent the authority from repurchasing the remaining tokens, effectively locking them in the contract and making them unrecoverable.

This issue has a **Medium** severity level due to the potential risk of token loss arising from an error by the authority.

Recommendation

We recommend setting a minimal repurchase price.

Client's commentary

> It will be up to the BORG members to ensure the correct repurchase price. We did not add any additional checks here.

M-9	Possible reverts due to early vesting termination
Severity	Medium
Status	Fixed in 9799c3cf

Description

This issue has been identified in the BaseAllocation.sol#L197 of the `BaseAllocation` contract. It is possible that there would be an attempt to create a proposal for the set, which has already terminated MetaVesTs (and they were terminated before the `vestingStartTime`). In the case of `VestingAllocation` and `TokenOptionAllocation` contracts, the call to the `getAmountWithdrawable` would revert.

This issue has a severity level of **Medium** because it may lead to the inability to create a proposal for an entire set.

Recommendation

We recommend adding a check to the `getMajorityVotingPower` function, which will ensure that in case if the vesting was terminated, it will have a `majorityVotingPower` equal to zero and wouldn't call `getAmountWithdrawable` function.

Client's commentary

> We have implemented the recommended fix, returning 0 if the vest has been terminated.

2.4 Low

L-1	Unclear Usage of the `prevOwners` Variable
Severity	Low
Status	Fixed in 1bae6630

Description

This issue has been identified in the `BaseAllocation` contract, specifically with the BaseAllocation.sol#L168 variable.
The variable `prevOwners` is not used anywhere within the contract's logic. The purpose of this variable is unclear, and its presence may cause confusion or unnecessary complexity in the code. Unused variables can increase gas costs and make the code harder to maintain.

Recommendation

We recommend removing the `prevOwners` variable if it is not required for any specific functionality.

Client's Commentary

> We have updated this with the recommended fix of removing the prevOwners variable.

L-2	Milestone Index Validation Before Access
Severity	Low
Status	Fixed in 1bae6630

Description

This issue has been identified in the `confirmMilestone` function of the BaseAllocation.sol#L223-L243 contract.
The function does not adequately validate the `_milestoneIndex` parameter before attempting to access the milestone details. If the `_milestoneIndex` is out of bounds, this could lead to an unexpected revert when the code tries to read from a non-existent milestone.

Recommendation

We recommend adding a check to ensure that the `_milestoneIndex` is within the valid range before attempting to read or modify the milestone data.

Client's Commentary

> We have updated this with the recommended fix of ensuring the index is in a valid range.

L-3	Inefficient Hash Check in `consentCheck` Modifier
Severity	Low
Status	Acknowledged

Description

This issue has been identified in the `consentCheck` modifier of the MetaVesTController.sol#L147-L152 contract.
The current implementation uses a method to check the hash of a parameter within the calldata by slicing the last 32 bytes. This approach is inefficient and error-prone, as it requires passing the entire calldata, which can lead to increased gas costs and potential security risks. Moreover, this approach may lead to missing checks for other parameters of the calldata as it verifies only the last 32 bytes. If there were amendment functions with more than one argument, all arguments except the last one would not be accounted for.

A more effective solution would be to pass the precomputed hash of the parameter directly to the modifier, thereby reducing complexity and improving performance.

Recommendation

We recommend modifying the consent check logic to pass the hash of the parameter directly to the modifier instead of using the current approach.

Client's Commentary

> We will keep the consentModifer as is, we need to check the length of the calldata to ensure extra parameters aren't added on at the end to by pass the last bytes check.

L-4	Lack of Uniqueness Check in `updateFunctionCondition`
Severity	Low
Status	Fixed in 9799c3cf

Description

This issue has been identified in the `updateFunctionCondition` function of the MetaVesTController.sol#L202-L207 contract.
The function does not check for the uniqueness of a condition before adding it to the list of conditions for a function signature. As a result, duplicate conditions can be added, which could require them to be removed multiple times, leading to unnecessary gas consumption and potentially creating confusion during condition verification.

Recommendation

We recommend adding a uniqueness check to ensure that a condition is not already present in the list before adding it.

Client's Commentary

> Client: We have updated this with the recommended fix.
> MixBytes: It is still possible to add a duplicate condition using `updateFunctionCondition` function.
> Client: We have now corrected Low #4 by added a check for unique condition contracts.

L-5	Unused Parameter `_longStopDate` in `createMetaVesT` Function
Severity	Low
Status	Acknowledged

Description

This issue has been identified in the `createMetaVest` function of the MetaVesTController.sol#L221 contract.
The `_longStopDate` parameter is declared but not used anywhere within the function.

Recommendation

We recommend removing the `_longStopDate` parameter from the function declaration if it is not required for any specific logic.

Client's Commentary

> We will be keeping this in for backward compatibility for other contracts that interface with metavestController.

L-6	Redundant `conditionCheck` Modifier in `createVestingAllocation` Function
Severity	Low
Status	Fixed in 1bae6630

Description

This issue has been identified in the `createVestingAllocation` function of the MetaVesTController.sol#L322 contract.
The `conditionCheck` modifier is applied in this internal function, but it is redundant because it is already called in the parent function from which this internal function is invoked.

Recommendation

We recommend removing the redundant `conditionCheck` modifier from the `createVestingAllocation` function to optimize gas usage and streamline the function's execution.

Client's Commentary

> We have updated this with the recommended fix.

L-7	Lack of Condition Limit Check in `addMetaVesTMilestone`
Severity	Low
Status	Fixed in 1bae6630

Description

This issue has been identified in the `addMetaVesTMilestone` function of the MetaVesTController.sol#L460 contract.
The function currently lacks validation for the number of conditions associated with a milestone before adding it. If a milestone is allowed to have an excessive number of conditions, it might become too expensive to satisfy all conditions, potentially leading to the milestone being permanently blocked from completion.

Recommendation

We recommend adding a limit on the number of conditions that can be associated with a milestone to ensure that the milestone completion process remains feasible and manageable.

Client's Commentary

> We have updated this with the recommended fix.

L-8	Ambiguous Return Value in `getSetOfMetaVesT` Function
Severity	Low
Status	Fixed in 1bae6630

Description

This issue has been identified in the `getSetOfMetaVesT` function of the MetaVesTController.sol#L688 contract.
Currently, the function returns an empty string if the specified `_MetaVesT` is not found in any set. However, since an empty string could also be a valid set name, this return value might lead to confusion or unintended behavior.

Recommendation

We recommend either reverting the transaction with a clear error message when the MetaVesT is not found in any set, or returning a unique value that distinctly indicates the absence of the MetaVesT, thereby eliminating any ambiguity.

Client's Commentary

> We have updated this with the recommended fix.

L-9	Unused Variable `ipaymentToken`
Severity	Low
Status	Fixed in 1bae6630

Description

This issue has been identified in the RestrictedTokenAllocation.sol#L8 and TokenOptionAllocation.sol#L8 contracts. The variable `ipaymentToken` is declared and assigned a value during the contract's construction but it is not utilized anywhere else in the code.

Recommendation

We recommend removing the `ipaymentToken` variable from the contracts if it is not required for any specific functionality.

Client's Commentary

> We have updated this with the recommended fix.

L-10	Uninitialized `tokensToRecover` and Unused `milestonesAllocation` in `terminate` Function
Severity	Low
Status	Fixed in 1bae6630

Description

This issue has been identified in the `terminate` function of the RestrictedTokenAllocation.sol#L167 contract.
The `tokensToRecover` variable is declared but never initialized. Additionally, the variable `milestonesAllocation` is calculated but not used anywhere in the function.

Recommendation

We recommend initializing `tokensToRecover` with the correct value to ensure proper functionality. Furthermore, either utilize `milestonesAllocation` for its intended purpose or remove it if it is not required to maintain clarity and clean code within the contract.

Client's Commentary

> We have updated this with the recommended fix.

L-11	Token Decimals Consideration in Rate Check
Severity	Low
Status	Acknowledged

Description

This issue has been identified in the constructor of all the allocation's contracts. The current validation logic checks if the `vestingRate` and `unlockRate` exceed a certain threshold (1000 * 1e18) without taking into account the token decimal precision. This oversight can lead to incorrect validations for tokens with fewer than 18 decimals (e.g., tokens with 6 decimals), potentially allowing rates that are excessively high.

Recommendation

We recommend updating the rate validation logic to account for the token's decimal precision, ensuring that the limits are appropriately adjusted based on the token's decimals.

Client's Commentary

> We did not update. It will be up to the BORG members to ensure the correct rate values.

L-12	Missing Validation for `vestingStartTime` and `unlockStartTime` in All the Allocation's Contracts' Constructor
Severity	Low
Status	Acknowledged

Description

This issue has been identified in the constructor of all the allocation's contracts. The `vestingStartTime` and `unlockStartTime` fields are not checked to ensure that they are not set to values in the past or to unreasonably distant future dates.

Recommendation

We recommend implementing validation checks to ensure that both `vestingStartTime` and `unlockStartTime` are set within a reasonable range — not earlier than the current timestamp and not in the distant future — to maintain predictable and secure vesting behavior.

Client's Commentary

> We did not update. It will be up to the BORG members to ensure the correct start time values.

L-13	Unnecessary Token Balance Check
Severity	Low
Status	Fixed in 1bae6630

Description

The issue was identified within the TokenOptionAllocation.sol#L147 function of the `TokenOptionAllocation` contract. There is an unnecessary check for the `msg.sender` token balance, as there are usually checks on the token contract side, and if the user's balance is insufficient, the transaction will revert.

The issue is classified as **Low** severity because it introduces unnecessary gas expenditure.

Recommendation

We recommend removing the mentioned check.

Client's commentary

> We have updated this with the recommended fix.

L-14	Incorrect Variable Naming
Severity	Low
Status	Fixed in 1bae6630

Description

The issue was identified in the RestrictedTokenAllocation.sol#L118 function of the `RestrictedTokenAllocation` contract. `repurchaseTokenDecimals` should be used instead of `exerciseTokenDecimals`.

The issue is classified as **Low** severity because it may lead to misunderstanding of the contract logic.

Recommendation

We recommend changing the mentioned variable name to `repurchaseTokenDecimals`.

Client's commentary

> We have updated this with the recommended fix.

L-15	Voter is Added to `proposal.voters` List Even if They do not Have Voting Power
Severity	Low
Status	Acknowledged

Description

The issue was identified in the MetaVesTController.sol#L615 function of the `MetaVesTController` contract. If `_callerPower` of `_grant` is zero, its address is still added to the `proposal.voters` list.

The issue is classified as **Low** severity because there is a missing check for the `_callerPower` before adding a voter to the voters' list.

Recommendation

We recommend adding a check to ensure that `_callerPower` is not zero.

Client's commentary

> We did not update the voting power check if a voter has 0 power.

L-16	Creation of `MajorityAmendmentProposal` is Possible for a Non-existent `setName`
Severity	Low
Status	Fixed in 1bae6630

Description

The issue is identified within the MetaVesTController.sol#L567 function of the `MetaVesTController` contract. There is a missing check for the `setName` parameter. It allows the initialization of a `MajorityAmendmentProposal` for a non-existent set of MetaVesTs.

The issue is classified as **Low** severity because it doesn't lead to any voting issues and allows to re-initialize the voting after a set with the used name has been added.

Recommendation

We recommend adding the following check: `require(doesSetExist(setName), "Set doesn't exist")`.

Client's commentary

> We have updated this with the recommended fix.

L-17	Unused Errors, Events and Variables
Severity	Low
Status	Fixed in 1bae6630

Description

There were identified unused errors in the following lines:
MetaVesTController.sol#L107,
MetaVesTController.sol#L111,
MetaVesTController.sol#L119,
MetaVesTController.sol#L120,
RestrictedTokenAllocation.sol#L17.
Unused events:
MetaVesTController.sol#L87,
MetaVesTController.sol#L88,
MetaVesTController.sol#L90,
MetaVesTController.sol#L93,
MetaVesTController.sol#L94,
RestrictedTokenAllocation.sol#L18.
Unused variables:
TokenOptionAllocation.sol#L15,
MetaVesTController.sol#L30.

Recommendation

We recommend removing unused errors, events, and variables.

Client's commentary

> We have cleaned any unused errors, events and variables from the suggested fix.

L-18	Missing Zero Check of Milestone Award
Severity	Low
Status	Fixed in 1bae6630

Description

This issue has been identified within the `validateAndCalculateMilestones` function of the `MetaVesTController` contract.
There is MetaVesTController.sol#L261-L272 to ensure that the `milestoneAward` is not zero. As a result, MetaVesT instances may end up with milestones that cannot be removed later, since the removal process MetaVesTController.sol#L442 if the `milestoneAward` is zero.

The issue is classified as **Low** severity because it may have an impact on milestone management process.

Recommendation

We recommend adding a check to ensure that the `milestoneAward` is greater than zero when validating milestones.

Client's Commentary

> We have updated this with the recommended fix.

L-19	`removeMetaVestFromSet()` Does Nothing if MetaVesT is Not in the Set
Severity	Low
Status	Fixed in 1bae6630

Description

This issue has been identified within MetaVesTController.sol#L698-L708.
The function lacks a revert condition when attempting to remove a MetaVesT from a set if the MetaVesT is not found in that set. The function also has no return value. As a result, the authority may face difficulties detecting errors in scenarios where the MetaVesT is not in the set.

The issue is classified as **Low** severity because it makes authority's mistakes more difficult to detect.

Recommendation

We recommend reverting `MetaVesTController.removeMetaVestFromSet()` if the specified MetaVesT is not in the set. Another option here is to add a return value to the function.

Client's Commentary

> We have updated this with the recommended fix by reverting when the metavest is not in the set.

L-20	Mappings `vestingAllocations`, `restrictedTokenAllocations`, and `tokenOptionAllocations` are not updated if the grantee's address was changed
Severity	Low
Status	Fixed in 1bae6630

Description

Mappings `MetaVesTController.vestingAllocations`, `MetaVesTController.restrictedTokenAllocations`, and `MetaVesTController.tokenOptionAllocations` reference the original grantee and BaseAllocation.sol#L273-L279 when the `BaseAllocation.transferRights()` function is called to change the grantee. As a result, allocations remain associated with the previous grantee, which can lead to inaccurate allocation records.

The issue is classified as **Low** severity because it may cause inconsistencies in data tracking without affecting the core functionality of the contract.

Recommendation

We recommend either updating mappings in `BaseAllocation.transferRights()` or adding a relevant comment to `MetaVesTController`.

Client's Commentary

> We have removed these mappings in favor of handling the record keeping via event indexing.

L-21	`transferRights()` requires a two-step process to change the address
Severity	Low
Status	Fixed in 1bae6630

Description

This issue has been identified within the `transferRights` function of the `BaseAllocation` contract. The current implementation allows the grantee to transfer the rights to claim tokens in BaseAllocation.sol#L273-L279, which can lead to accidental errors. If the grantee mistakenly enters an incorrect address, tokens may become locked in the contract, potentially causing permanent loss of tokens.

The issue is classified as **Low** severity because it poses a risk of token loss in case of a low probability user's mistake.

Recommendation

We recommend implementing a two-step process for address changes in the `BaseAllocation.transferRights()` function, requiring confirmation from the new owner before completing the address change.

Client's Commentary

> We have updated with the recommended fix of adding a 2 step transfer process confirmTransfer().

L-22	Missing Minimum Exercise Price Check for `TokenOptionAllocation`
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the `updatePrice` function and within the constructor of the `TokenOptionAllocation` contract.
There is no validation for a minimum exercise price. If the exercise price is set too low, the `getPaymentAmount()` function could return zero, TokenOptionAllocation.sol#L145 to execute the token option.

This issue is classified as **Low** severity because the authority can later set a normal price or recover tokens after termination.

Recommendation

We recommend adding a minimum exercise price check to `TokenOptionAllocation.updatePrice()` and to `TokenOptionAllocation.constructor()`.

Client's Commentary

> We did not update this. It will be up to the BORG members to ensure the correct execution price.

L-23	Unused `getGoverningPower` function and `govType` variable
Severity	Low
Status	Acknowledged

Description

This issue has been identified in all three vesting contracts. An unused `getGoverningPower` function is implemented in each contract. Also, the function BaseAllocation.sol#L229 in the `BaseAllocation` contract is unused, along with the related storage variable `govType`.

This issue is classified as **Low** severity because the mentioned logic is never used in the contract.

Recommendation

We recommend removing the mentioned unused function and storage variable.

Client's Commentary

> The method "setGovVariables" is used by the MetaVestController to set the GovType for a vesting contract. The external view "getGoverningPower" returns the governance power that the vesting contract should give towards external governance systems.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes