MANTLE NETWORK CMETH SECURITY AUDIT REPORT

Nov 04, 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

Mantle cMETH is a cross-chain, yield-generating protocol. The underlying tokens are locked in the BoringVault on the mainnet and then deposited into integrated staking protocols by a trusted account. In return, users receive cMETH tokens, which have a steadily increasing exchange rate over time, generating yield. cMETH can also be bridged to other blockchain networks via the LayerZero protocol, where it functions as a standard ERC20 token.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Mantle
Project name	cMETH
Timeline	21.08.2024 - 15.10.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
21.08.2024	d6e2d18f45a3e05d749d34966139fc85fc47f7e6	Commit for the audit
15.10.2024	b7d10e14f766c9dbdb2e8a589c62245bfc844bee	Commit for the re-audit

Project Scope

The audit covered the following files:

File name	Link
src/lib/TransparentUpgradeableProxy.sol	TransparentUpgradeableProxy.sol
src/lists/SanctionsList.sol	SanctionsList.sol
src/lists/Blocklist.sol	Blocklist.sol

File name	Link
src/base/Roles/DelayedWithdraw.sol	DelayedWithdraw.sol
src/base/Roles/Pauser.sol	Pauser.sol
src/base/Roles/AccountantWithRateProviders.sol	AccountantWithRateProviders.sol
src/base/Roles/ManagerWithMerkleVerification.sol	ManagerWithMerkleVerification.sol
src/base/Roles/TellerWithMultiAssetSupport.sol	TellerWithMultiAssetSupport.sol
src/base/BoringVault.sol	BoringVault.sol
src/ClientSanctionsListUpgradeable.sol	ClientSanctionsListUpgradeable.sol
src/L2cmETH.sol	L2cmETH.sol
src/ClientBlockListUpgradable.sol	ClientBlockListUpgradable.sol
src/L2MessagingStatus.sol	L2MessagingStatus.sol
src/L1MessagingStatus.sol	L1MessagingStatus.sol
src/L1cmETH.sol	L1cmETH.sol
src/L1cmETHAdapter.sol	L1cmETHAdapter.sol
script/ArchitectureDeployments/DeployArcticArchitecture.sol	DeployArcticArchitecture.sol
script/ArchitectureDeployments/Mainnet/DeployMantleCmETHBoringVault.s.sol	DeployMantleCmETHBoringVault.s.sol
script/CreateMerkleRoot.s.sol	CreateMerkleRoot.s.sol

Deployments

Ethereum:mainnet

File name	Contract deployed on mainnet	Comment
TransparentUpgradeableProxy.sol	0xE6829d...1859e8fA	L1cmETH.sol proxy
TransparentUpgradeableProxy.sol	0x4aFA96...d475e948	L1cmETHAdapter.sol proxy
TransparentUpgradeableProxy.sol	0x4733E6...5D3d30F5	L1MessagingStatus.sol proxy
L1cmETH.sol	0x9d7aeF...468bf9F6	Old commit. Lacks minor change in `payable` modifier in `setMaxTotalSupply` function
L1cmETHAdapter.sol	0xaE96dF...E39FAc44	
L1MessagingStatus.sol	0xa91377...7C84DE7e	
TransparentUpgradeableProxy.sol	0x33272D...e85D4fE4	BoringVault.sol proxy
ManagerWithMerkleVerification.sol	0xAEC024...64878bCE	
AccountantWithRateProviders.sol	0x6049Bd...D610a2eC	Old commit. Lacks console logging removal and renaming of `_changeDecimals` to `changeDecimals`
TellerWithMultiAssetSupport.sol	0xB6f7D3...0f1912b0	Old commit. Lacks minor changes related to share lock logic, events, errors, and `nonReentrant`
DelayedWithdraw.sol	0x12Be34...2a66B113	Old commit. Lacks formatting changes
Pauser.sol	0x589A72...6640a95e	
BoringVault.sol	0x191126...6E2C4407	

Mantle:mainnet

File name	Contract deployed on mainnet	Comment
TransparentUpgradeableProxy.sol	0xE6829d...1859e8fA	L2cmETH.sol proxy
TransparentUpgradeableProxy.sol	0x4733E6...5D3d30F5	L2MessagingStatus.sol proxy
L2cmETH.sol	0x5A7b3C...C54Ca033	
L2MessagingStatus.sol	0xf9Ca01...42eDcB81	Old commit. Lacks minor changes in function naming and ether handling comment

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	1
Medium	5
Low	18

ID	Name	Severity	Status
H-1	`getRateInQuote` Function Lacks Last Update Time Check	High	Acknowledged
M-1	Cross-Chain Transfers Exceeding the Cap Are Temporarily Locked	Medium	Acknowledged
M-2	Incorrect Loss Calculation in `_completeWithdraw`	Medium	Acknowledged
M-3	Delayed Transaction Handling in `updateExchangeRate`	Medium	Acknowledged
M-4	Fee Abuse in Integrated Contract Actions	Medium	Acknowledged
M-5	Centralization Risks	Medium	Acknowledged
L-1	Arbitrary Parameters in `changeCompletionWindow` and `changeWithdrawDelay`	Low	Acknowledged
L-2	Bypassing the Total Supply Constraint in `manageVaultWithMerkleVerification`	Low	Acknowledged
L-3	Redundant Argument in `flashLoan` Function	Low	Acknowledged

L-4	Race Condition in `removePausable` Function	Low	Acknowledged
L-5	Unused Constants, Errors, and Events in `TellerWithMultiAssetSupport`	Low	Fixed
L-6	Unintended Logging in Production Code (`console.log`)	Low	Fixed
L-7	Missing `nonReentrant` Modifier in Administrative and Bulk Withdrawal Functions	Low	Fixed
L-8	Streamline Error Handling with `require` Statements	Low	Acknowledged
L-9	Gas Optimization and Code Efficiency Improvements	Low	Acknowledged
L-10	Default Public Admin Side Setters	Low	Acknowledged
L-11	Error Handling Improvements in `getRateInQuoteSafe`	Low	Acknowledged
L-12	Uninitialized Mapping in `senderPause` and `senderUnpause`	Low	Acknowledged
L-13	Incorrect Handling of `NATIVE` Parameter in `depositWithPermit`	Low	Acknowledged
L-14	Inconsistent Naming Convention: `changeDecimals`	Low	Fixed
L-15	Missing Event Emission in `withdrawNonBoringToken`	Low	Acknowledged
L-16	Improve Security by Hashing All Data in `keccak256` in Flash Loan Logic	Low	Acknowledged
L-17	Double Pause/Unpause in `pause` and `unpause`	Low	Acknowledged
L-18	Arbitrage Risk in `bulkDeposit` and `bulkWithdraw`	Low	Acknowledged

1.6 Conclusion

In our analysis, we investigated several potential attack vectors, most of which did not reveal any vulnerabilities. Below is a list of the attack vectors we examined. The confirmed issues will be presented in the findings section.

Attack Vectors

1. **Centralization**
   The project has a significant level of centralization. Administrators can update the contract's code, manipulate exchange rates, cancel or ignore withdrawal requests, and block token transfers based on the sender or recipient address.

   - We verified if administrators have these abilities and if the project's governance model and multisig requirements provide adequate checks and balances to prevent abuse.

2. **Cross-Chain Transfers**
   This vector includes risks related to capped cross-chain transfers, repeated transfers in case of temporary failures, etc. The project implements the OFT (Omnichain Fungible Token) standard.

   - We verified that the implementation correctly adheres to the OFT standard and handles transfer limits and potential replay attacks.

3. **Correct Behavior During Rate Update**
   Possible yield stealing and risks associated with the implementation of auto-pause.

   - We ensured that rate updates were implemented correctly, including the auto-pause feature, and that no yield manipulation was possible.

4. **Configuration Changes**
   Incorrect project configuration or parameter setting could introduce risks.

   - We validated that all parameters undergo strict validation and that configuration changes do not introduce undue risks based on the project architecture.

5. **Merkle Tree-Based Permission System**
   The Merkle tree is used to limit the strategist's ability to interact with contracts.

   - We ensured that the restrictions were secure and properly enforced. Additionally, we verified that the strategist cannot manipulate parameters that are not accounted for in sanitizers.

6. **Rate Update Service Failure**
   The project relies on a service to update rates. If this service fails, it could lead to discrepancies.

- We tested the project's handling of service downtime and ensured no critical dependencies could break the system.

7. **Griefing Attacks**
   Attackers may perform actions that harm the project or its users without benefiting themselves.

   - We found no effective methods of griefing during our review.

8. **Rate Manipulation During Withdrawals**
   Changing rates during withdrawals could allow users to gain unfair advantages.

   - We confirmed that rate changes are handled securely, preventing unwarranted gains during withdrawals.

9. **Inflation Attack**
   The project does not implement the standard share valuation methodology, which could leave it vulnerable to inflation attacks.

   - We verified that the project is not vulnerable to such attacks due to the structure of share value calculations.

10. **Strategist Sandwich Attack**
    A strategist could reorder transactions to gain an unfair advantage.

    - We assessed the strategist's role and verified that the order of transactions cannot be manipulated for undue gain.

11. **Reentrancy Attack**
    Reentrancy attacks exploit vulnerabilities in contract logic to drain funds.

    - We reviewed the project to ensure it is protected against reentrancy attacks, including cases involving ERC777 tokens.

12. **Oracle Manipulation**
    The project uses centralized oracles, which may be subject to manipulation.

    - We confirmed that the oracles are sufficiently secure and that malicious users cannot influence their outcomes.

13. **Flashloan Implementation Attack**
    The implementation of flash loans could be vulnerable to attacks.

    - We confirmed that the logic of flashloan requests and reception is robust and cannot be manipulated by attackers.

14. **Slashing**
    The project must handle slashing events properly, ensuring they do not block user withdrawals or result in unfair user advantages.

    - We verified that the project handles slashing scenarios correctly without unfairly affecting users.

15. **Deployment Script Verification**
    The security of the deployment scripts can affect the overall project.

    - We reviewed the deployment scripts and confirmed they do not introduce any security issues.

16. **Withdraw Request Locking**
    Users may be locked out of their funds due to incorrect parameters.

    - We ensured that incorrect parameters cannot result in users losing access to their funds.

17. **Unauthorized Withdraw Request Modifications**
    Users may be able to modify or cancel withdrawal requests maliciously.

    - We verified that only authorized actions can modify withdrawal requests.

18. **Double Spend on Withdrawal**
    An attacker could attempt to double spend by canceling or processing a withdrawal.

    - We ensured that such scenarios are handled securely and cannot lead to profit.

19. **Volatility Prediction Attack**
    Users could predict rate changes to profit unfairly from volatility.

    - We confirmed that the system does not offer any guaranteed arbitrage opportunities.

20. **Rate Conversion and Decimals Handling**
    Mishandling decimals during rate conversions could lead to either blocked transactions or unfair profits.

    - We confirmed that decimals are handled correctly throughout the code.

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

H-1	`getRateInQuote` Function Lacks Last Update Time Check
Severity	High
Status	Acknowledged

Description

This issue has been identified within the AccountantWithRateProviders.sol#401 function of the `AccountantWithRateProviders` contract.

The function does not verify if the retrieved rate is outdated. If the oracle fails to update the exchange rate over an extended period, the contract may continue using stale data. This could lead to incorrect calculations during withdrawals, potentially exposing the system to arbitrage attacks.

The issue is classified as **high severity** due to the risk of significant financial loss if stale data is exploited.

Recommendation

We recommend revising the rate retrieval and update logic by implementing a check to ensure the rate is recently updated before it is used.

Client's Commentary

> Client: cmETH won't be using rate providers as users will only be entering and exiting with mETH. This means that this issue has no affect on cmETH. Outside of cmETH scope, BoringVaults only accept highly correlated assets, so I see 2 of the most probable worst case paths.
> Bad) A rate provider isn't updated for entire month and we dont notice it in time to stop deposits
> Worse) An LST undergoes some exploit so its rate drops significantly, and someone front runs us to deposit before we can stop deposits with the depegged asset.

> The bad case isn't that big of a deal because withdraws are permissioned so even if this rare scenario happens, the attacker can only win if we somehow miss the fact that the rate provider is not updating(Which is unlikely since we perform daily share price calculations, and peg monitoring), and we also need to move the money into the DelayedWithdrawer to fulfill this withdraw.
> In the worse case scenario given that a significant portion of the BoringVault's TVL was lost from the depegged LST, all deposit and withdraws would stopped, and then from there we can decide on a custom best course of action, but there is really no way an attacker would be able to profit from this scenario since withdraws are permissioned.
> This issue should be removed from cmETH as it does not affect it. As for broader BoringVaults, I dont think it is a bad idea to check a timestamp when getting a rate, however I am not aware of any RateProvider interface that exposes a way to get the updated timestamp. The vast majority seem to only have functions to get the rate, and possibly to get the oracle address, so if you have examples that use a standardized interface, I can look into addressing it.
> MixBytes: We agree that in the case of cmETH, where only mETH will be used as collateral, this should not be an issue. Marked as acknowledged.

2.3 Medium

M-1	Cross-Chain Transfers Exceeding the Cap Are Temporarily Locked
Severity	Medium
Status	Acknowledged

Description

This issue has been identified within the L2cmETH.sol#122 function of the `L2cmETH` contract.

The function reverts when the credited amount, combined with the `totalSupply`, exceeds the cap. In cases where multiple cross-chain transactions are processed simultaneously, or if the token cap is lowered during the transfer, tokens may be locked on the destination chain without prior warning that the cap would be exceeded. These tokens remain locked until the admin manually intervenes to increase the cap.

The issue is classified as **medium severity** because it can disrupt cross-network operations, resulting in failed transfers and negatively impacting the usability of the contract.

Recommendation

We recommend implementing a rollback mechanism for transactions that exceed the cap, allowing failed transactions to return to the sender chain without requiring admin intervention, or enabling admins to resolve the issue through actions other than increasing the cap.

Client's Commentary

> Client: The main purpose of Cap is to limit the amount of cross-chains on each network. Changing the parameters provides potential limiting capabilities, but it is not currently enabled
> MixBytes: We understand that the cap is intended to limit cross-chain transfers on each network. However, when transfers exceed this cap, tokens become locked on the destination chain without an automatic resolution. This requires manual intervention to increase the cap, which can delay transactions and negatively affect user experience. The current design lacks real-time transaction management for these scenarios, which could hinder the contract's scalability and usability. We recommend implementing a mechanism to handle such cases without manual intervention.

M-2	Incorrect Loss Calculation in `_completeWithdraw`
Severity	Medium
Status	Acknowledged

Description

This issue has been identified within the DelayedWithdraw.sol#568 function of the `DelayedWithdraw` contract.

The logic designed to check for acceptable loss when completing a withdrawal request is not functioning correctly. Specifically, the formula used to ensure the loss stays within the `maxLoss` limit incorrectly causes the function to revert before reaching the maximum allowed loss. For example, if `maxLoss` is set to 100%, any loss value should be accepted, but the current formula restricts the `maxRate / minRate` ratio not to exceed 2. This leads to reverts even when the loss is within acceptable limits, thereby blocking valid withdrawal requests.

The issue is classified as **medium severity** because, while it doesn't pose a security threat, it impacts the contract's operational flexibility and prevents legitimate withdrawals from being processed.

Recommendation

We recommend revising the loss calculation logic to correctly handle the `maxLoss` condition, preventing reverts and allowing valid withdrawals to proceed.

Client's Commentary

> Client: Max loss can not be set to 100%, the DelayedWithdrawer enforces that the maxLoss is not greater than the constant MAX_LOSS which is set to 50%. So the math seems to work out fine when you account for the constraints the contract enforces on the max loss.
> MixBytes: We used 100% as a trivial example to illustrate the issue. Even with `MAX_LOSS` set to 50%, the current formula in the `_completeWithdraw` function causes a revert when `minRate` is lower than `maxRate / 1.5`, which is approximately `66%` of `maxRate` (or a 33% loss). This means the function reverts before reaching the allowed 50% loss threshold, preventing valid withdrawals that should be permitted under the contract's constraints.

M-3	Delayed Transaction Handling in `updateExchangeRate`
Severity	Medium
Status	Acknowledged

Description

This issue has been identified within the AccountantWithRateProviders.sol#306 function of the `AccountantWithRateProviders` contract.

If there is a delay in processing an exchange rate update transaction, a subsequent transaction may be initiated shortly after the first one completes, within a time period shorter than the set `withdrawDelay`. When both transactions are processed consecutively, the contract may unexpectedly pause, disrupting normal operations and negatively impacting users.

The issue is classified as **medium severity** because it can cause unintended pauses in the contract, leading to operational disruptions during normal usage.

Recommendation

We recommend modifying the function logic to include a `timestamp` argument. This would ensure that if the `block.timestamp` of a transaction exceeds a preset deadline, the transaction reverts without pausing the contract. This modification would allow the most recent transaction to proceed successfully while reverting any outdated or delayed transactions.

Client's Commentary

> cmETH's exchange rate will only be updated in black swan situations where the BoringVault has lost money and needs to socialize losses. In the event this happens we would actually want it to be paused, and would have bots actively trying to pause the accountant. As for broader BoringVault scope I will give this some more thought, but I am liking this idea

M-4	Fee Abuse in Integrated Contract Actions
Severity	Medium
Status	Acknowledged

Description

This issue has been identified within the DelayedWithdraw.sol#525 and DelayedWithdraw.sol#428 functions of the `DelayedWithdraw` contract.

The issue allows for a potential exploit where an attacker can manipulate the deposit and withdrawal processes to abuse fees and negatively affect the exchange rate for other users.

The attack involves the following steps:

1. The attacker deposits a large portion of the total value locked (TVL).
2. The attacker creates a withdrawal request.
3. The attacker waits until the protocol withdraws the underlying assets from the integrated protocol to fulfill the withdrawal request, then cancels the request.
4. After canceling the withdrawal, the attacker waits for the protocol to reallocate assets and repeats the process.

This cycle forces strategists to keep a significant portion of assets locked, which negatively impacts the yield for other users. Moreover, the attacker can exploit withdrawal and deposit fees from integrated contracts while continuing to earn yield, reducing the protocol's efficiency and motivating users to withdraw their funds.

The issue is classified as **medium severity** because it impacts the protocol's overall performance, causing inefficiencies and potential loss for regular users.

Recommendation

We recommend recalculating user shares when a withdrawal request is canceled or updated by using the lower of the saved rate at the time of the request creation or the rate at the time of cancellation or updating. This would prevent the attacker from unfairly earning yield while canceling and recreating requests.

Client's Commentary

> Client: This is an interesting attack vector, however its purely griefing and requires the attacker to utilize a large sum of capital that to perform it, rather than earning actual money on their capital. So to

> me the attacker is not getting anything out of this and is actively losing money(as it could be invested elsewhere), and if this really became an issue we can just chose to not move the into the DelayedWithdrawer, or we wait for their request to become fulfillable, then in a single TX we rebalance the BoringVault to move the funds in the delayed withdrawer, then we call completeUserWithdraw to complete it. Now they could deposit again, but that point we have more than enough proof that they are doing this to themselves, and we can just choose to not fulfill their withdraw.
> MixBytes: The main issue is that the user continues to earn yields while their withdrawal request is in the waiting queue, even if they cancel the request. Therefore, it's not as unprofitable for an attacker with large capital to perform this exploit as it might seem. Our recommendation is to adjust the calculation of user shares when a withdrawal request is canceled or updated, so that the user does not earn yields during the time they were in the waiting queue. This would prevent attackers from unfairly benefiting from the process.

M-5	Centralization Risks
Severity	Medium
Status	Acknowledged

Description

The project presents several centralization risks, including:

- **Full control over exchange rates:** The admin or a specific role has the authority to set and update exchange rates, potentially manipulating asset values.
- **Risk of incorrect configuration (e.g., Merkle Root setup):** If the Merkle Root is misconfigured, it could invalidate key functionality or compromise access control mechanisms.
- **Strategist privileges:** The Strategist has the ability to deposit or withdraw unlimited amounts of tokens in whitelisted protocols, which could be abused to mismanage funds.
- **Ability to lock withdrawals, deposits, and transfers:** Certain roles have the power to pause essential functions like withdrawals, deposits, and transfers, potentially halting user operations.
- **Admin control over BoringVault:** The admin has complete control over vault operations, which can lead to a single point of failure or misuse of assets stored in the vault.

This issue is classified as **medium severity** because it does not directly impact security under normal operation, but centralization poses a risk if malicious or erroneous actions are taken by the privileged entities.

Recommendation

To mitigate these centralization risks, we recommend the following:

1. **Implement a Multi-Signature Wallet for Administrative Actions:**
   Require multiple authorized signatures for any critical actions, such as setting exchange rates, modifying the Merkle Root, or making significant deposits and withdrawals. This reduces the risk of a single point of failure or misuse of authority.

2. **Introduce Timelocks for Critical Functions:**
   Add a time delay for executing sensitive operations like changing exchange rates, adjusting withdrawal/deposit settings, or making strategic changes. This provides the community or stakeholders time to review and react to potential harmful actions.

3. **Decentralize Governance:**
   Introduce a governance model where critical decisions, such as pausing withdrawals or modifying

vault operations, require community or token-holder votes. This would distribute control and ensure decisions reflect the interests of a wider group of stakeholders.

4. **Transparent Monitoring and Alerts:**
   Implement real-time monitoring and alerts for all sensitive actions, especially those carried out by the admin or strategist roles.

By distributing control across multiple parties, implementing time delays, and increasing transparency, these recommendations would significantly reduce the project's exposure to centralization risks.

Client's Commentary

> We already use multisigs for admin actions, and are developing extensive monitoring and alerts. Introducing a timelock, and some way to add in a DAO is a great idea though.

2.4 Low

L-1	Arbitrary Parameters in `changeCompletionWindow` and `changeWithdrawDelay`
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the DelayedWithdraw.sol#286, DelayedWithdraw.sol#269, and DelayedWithdraw.sol#236 functions of the `DelayedWithdraw` contract.

The `completionWindow` parameter can be set to an arbitrarily low value, potentially to 0, which would disable withdrawals for the affected asset. Similarly, the `withdrawDelay` parameter can be set to an arbitrarily high value, causing unnecessary delays in user withdrawals. Both issues could disrupt normal operations and negatively impact the user experience.

The issue is classified as **low severity** because, while it does not present a security risk, it can cause operational inconveniences by disabling or delaying withdrawals.

Recommendation

We recommend setting reasonable lower and upper bounds for both the `completionWindow` and `withdrawDelay` parameters to prevent accidental misconfigurations.

Client's Commentary

> Even if these bounds were added, the strategist could still choose to not move funds into the Delayed Withdrawer contract, or the admin multisig could make the withdraw functions not publicly callable, both of which would prevent users from withdrawing. So adding these checks only adds further complexity and inflexibility while not really providing any more guarantees for users.

L-2	Bypassing the Total Supply Constraint in `manageVaultWithMerkleVerification`
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the ManagerWithMerkleVerification.sol#160 function of the `ManagerWithMerkleVerification` contract.

The code enforces that the total supply must remain constant during the management of the vault. However, there is a possibility of bypassing this constraint through deposit/pump/withdraw activities, which could manipulate the exchange rate and avoid the check.

This issue is classified as **low severity** because the current set of actions defined in the `CreateMerkleRoot` contract does not allow such activities, and the strategist role is a trusted entity.

Recommendation

We recommend implementing additional safeguards to prevent potential manipulation of the total supply, even if such actions are not currently allowed.

Client's Commentary

> To perform this would require adding in malicious leafs to the merkle tree, as well as granting several roles to malicious actors/contracts in the RolesAuthority. This issue seems to fall under the scope of M5, because we already have the trust assumption that the strategist/admins will act in the best intereset of the BoringVault.

L-3	Redundant Argument in `flashLoan` Function
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the ManagerWithMerkleVerification.sol#175 function of the `ManagerWithMerkleVerification` contract.

The `recipient` argument is redundant because the recipient is always implicitly `address(this)`. This conclusion is based on several factors:

1. The `flashLoanIntentHash` is set with a non-zero value before the `flashLoan` call.
2. The `balancerVault.flashLoan` function is non-reentrant, meaning there is no possibility for an external call to `this.receiveFlashLoan` unless the recipient is `address(this)`.
3. The `flashLoanIntentHash` is reset to zero only within the `receiveFlashLoan` function.
4. The code reverts if the `flashLoanIntentHash` remains non-zero, ensuring the operation is tightly controlled.

The issue is classified as **low severity** because, while the argument is redundant, it does not pose a security risk or critical functional issue.

Recommendation

We recommend removing the `recipient` argument to simplify the code and avoid redundancy, given that `address(this)` is always the recipient.

Client's Commentary

> cmETH won't use flash loans so this issue does not affect it. As for broader BoringVaults I will look into fixing this.
> Does not affect cmETH as we are not using flashloans, also at the end of the day its just a bit of redundancy and not a big deal.

L-4	Race Condition in `removePausable` Function
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the Pauser.sol#64 function of the `Pauser` contract.

When multiple admins execute this function simultaneously, there is a risk of inadvertently removing the wrong address from the list of pausables. This could occur due to changes in the list's length during concurrent executions, which would cause an unwanted address to be removed.

The issue is classified as **low severity** because it does not create a security vulnerability but could result in unintended modifications to the list of pausables.

Recommendation

We recommend adding an additional argument, such as the specific address to be removed, or adjusting the function to search by address rather than by index. This would ensure that only the intended address is removed, preventing errors during concurrent executions.

Client's Commentary

> There is only one admin multisig capable of adjusting the pausables array, so this is very unlikely to ever happen.

L-5	Unused Constants, Errors, and Events in `TellerWithMultiAssetSupport`
Severity	Low
Status	Fixed in 5a87a8a4

Description

Several constants, errors, and events defined in the TellerWithMultiAssetSupport.sol contract are currently unused:

- The constant `MAX_SHARE_LOCK_PERIOD` is defined but not referenced in the contract.
- Several errors, such as `TellerWithMultiAssetSupport__ShareLockPeriodTooLong`, `TellerWithMultiAssetSupport__SharesAreLocked`, and others, are declared but not used.
- The events `DenyFrom`, `DenyTo`, `DenyOperator`, `AllowFrom`, `AllowTo`, and `AllowOperator` are also unused.

The issue is classified as **low severity** since these unused items do not impact the contract's functionality but add unnecessary complexity and can confuse developers and auditors.

Recommendation

We recommend removing these unused constants, errors, and events to simplify the contract and avoid confusion.

Client's Commentary

> Fixed here: https://github.com/Se7en-Seas/cMETH-boring-vault/commits/5a87a8a414504145e31b781df7ea040c1db79e8d

L-6	Unintended Logging in Production Code (`console.log`)
Severity	Low
Status	Fixed in ef53b385

Description

Within the AccountantWithRateProviders.sol#461 function of the `AccountantWithRateProviders` contract, there is a `console.log` statement that outputs `minimumAssets` during the fee calculation process. This is a leftover debug log intended for development purposes, and should not be present in the production code. Keeping it in the live contract may lead to unnecessary gas consumption.

This issue is classified as **low severity** because it does not pose a security risk but is an inefficient use of resources in a live environment.

Recommendation

We recommend removing the `console.log` statement to improve performance and avoid unnecessary gas costs.

Client's Commentary

> Fixed here: PR-7

L-7	Missing `nonReentrant` Modifier in Administrative and Bulk Withdrawal Functions
Severity	Low
Status	Fixed in e1ae003d

Description

This issue has been identified in both the DelayedWithdraw.sol#355 function of the `DelayedWithdraw` contract and the TellerWithMultiAssetSupport.sol#253 function of the `TellerWithMultiAssetSupport` contract.

While the `cancelWithdraw` function includes a `nonReentrant` modifier, the `cancelUserWithdraw` function lacks this protection. Though only accessible by administrators, this omission could theoretically expose the contract to reentrancy risks. Similarly, the `bulkWithdraw` function, which handles withdrawals and interacts with external contracts, also lacks reentrancy protection. Despite following the check-effect-interaction pattern, the absence of the `nonReentrant` modifier could possibly leave the contract vulnerable in specific edge cases.

The issue is classified as **low severity** because no immediate exploitation path has been identified, and administrative access limits the risk in `cancelUserWithdraw`. Additionally, `bulkWithdraw` already follows secure patterns but would benefit from extra protection.

Recommendation

We recommend adding the `nonReentrant` modifier to both the `cancelUserWithdraw` and `bulkWithdraw` functions to mitigate any potential reentrancy risks, ensuring robust security even for administrative and batch processes.

Client's Commentary

> I can't really think of when this would become an issue, but the only downside I can think of with adding non reentrant is some extra gas usage, which isn't a big deal, so will fix.
> Fixed here: PR-7

L-8	Streamline Error Handling with `require` Statements
Severity	Low
Status	Acknowledged

Description

As of Solidity 0.8.27, the preferred approach for error handling is to use `require(bool, Error)` statements rather than `if (bool) revert Error`. This modern approach improves code readability, simplifies logic, and increases gas efficiency.

The issue is classified as **low severity** because it does not pose security risks but affects code quality and performance.

Recommendation

We recommend updating the error handling mechanisms to use `require(bool, Error)` to align with current Solidity best practices.

Client's Commentary

> In an effort to make as little changes as possible to contract code that has been in production for almost 6 months with 1B+ in TVL, I think it is better to accept this, and address it in the future when the codebase is revamped.

L-9	Gas Optimization and Code Efficiency Improvements
Severity	Low
Status	Acknowledged

Description

Several gas optimization and code efficiency issues have been identified across multiple functions and contracts. These primarily affect gas usage and code clarity without compromising security or core functionality:

1. **`nativeWrapper` Before Shares Check**
   In the TellerWithMultiAssetSupport.sol#181 contract, the native token is wrapped before the `minimumMint` check, leading to unnecessary gas consumption for transactions that revert later. Moving the `nativeWrapper.deposit{value: msg.value}()` call after the `minimumMint` check would optimize gas usage.

2. **Max Loss Calculation Simplification**
   The max loss calculation in the `DelayedWithdraw` contract can be simplified. The current verbose logic can be streamlined for better readability without altering its functionality.

3. **`_credit` Function Gas Optimization**
   The code at L2cmETH.sol#L122 makes two external calls to the same method. This can be optimized by using a temporary variable, reducing gas consumption.

4. **Redundant Assignment in `removePausable`**
   In the `Pauser` contract, the Pauser.sol#70 function performs unnecessary assignments when removing the last element from the array. Adding a conditional check to skip this step when the element is already the last one will optimize gas usage.

5. **Using Immutable Variables**
   Several variables across the following contracts can be converted to `immutable` to reduce gas consumption:

   - BoringVault.sol#L24
   - L1cmETH.sol#L40
   - L2cmETH.sol#L30

   Converting these variables to `immutable` would reduce storage reads, improving gas efficiency.

These issues are classified as **low severity** because they primarily impact gas efficiency and code readability, without affecting the contracts' security or functionality.

Recommendation

- Move the `nativeWrapper.deposit{value: msg.value}()` call after the `minimumMint` check in the `TellerWithMultiAssetSupport` contract.
- Simplify the max loss calculation in the `DelayedWithdraw` contract as follows:

```
(uint256 minRate, uint256 maxRate) = req.exchangeRateAtTimeOfRequest
  < currentExchangeRate
    ? (req.exchangeRateAtTimeOfRequest, currentExchangeRate)
    : (currentExchangeRate, req.exchangeRateAtTimeOfRequest);
```

- Use a temporary variable and a `require` statement in the `_credit` function to reduce external calls:

```
uint256 cap = IL2StatusRead(status).capacity();
require(cap == 0 ||
        totalSupply() + _amountLD <= cap, MaxSupplyOutOfBound());
```

- Add a conditional check in the `removePausable` function to avoid redundant assignments when removing the last element from the array.
- Convert eligible variables in the specified contracts to `immutable` to reduce gas costs.

Client's Commentary

> 1,2,4 I will review. 5. The BoringVault's cmETH variable needs to be a state variable in order for the BoringVaultUpgradeable to work.
> These are just gas optimizations, that are not needed

L-10	Default Public Admin Side Setters
Severity	Low
Status	Acknowledged

Description

The ClientBlockListUpgradable.sol#L66 and ClientSanctionsListUpgradeable.sol#L66 functions in the `ClientBlockListUpgradable` and `ClientSanctionsListUpgradeable` contracts are public by default.

If these functions are not overridden in inheriting contracts, they could introduce critical vulnerabilities by allowing unauthorized access to sensitive admin functions. While no immediate security risk exists, failing to override these functions in future development could lead to unintended vulnerabilities.

The issue is classified as **low severity** because it impacts future development and the potential for misconfigured access control.

Recommendation

We recommend removing these public functions from the abstract contracts, leaving only internal functions in place. This ensures that inheriting contracts can implement the necessary access controls to prevent unauthorized access.

Client's Commentary

> Is related to the setBlockList setSanctionList functions. I dont think it is neccessary to fix it, but it is very important to keep in mind for future development. In fact that fix I would recommend is to add a test to your testing suite where you try to have a random EOA call those functions and insure that the tx reverts. That way this issue is always verified to not be an issue when developing new code.

L-11	Error Handling Improvements in `getRateInQuoteSafe`
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the AccountantWithRateProviders.sol#401 function of the `AccountantWithRateProviders` contract.

If the `rateProvider` for an asset is not properly initialized, the call to `rateProvider.getRate()` results in a revert that lacks clear error messaging. While the functionality remains intact, the error handling could be improved for better clarity and user experience.

The issue is classified as **low severity** because it impacts error handling and clarity rather than core functionality.

Recommendation

We recommend enhancing error handling by adding a custom revert message when the asset is not initialized, making it easier to diagnose the issue.

Client's Commentary

> I will consider if this is worth adding.
> This issue seems it would only come up if some developer was trying to use the accountant for pricing arbitrary assets, which is something we have never encouraged anyone to do, and something that I could never imagine an external developer wanting to do(especially since the logic needed to integrate with RateProvider contracts is so minimal). So this is perfectly fine to not fix.

L-12	Uninitialized Mapping in `senderPause` and `senderUnpause`
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the Pauser.sol#166 and Pauser.sol#177 functions of the `Pauser` contract.

A revert occurs if the address in the `senderToPausable` mapping has not been initialized for a specific sender, potentially disrupting contract operations. This issue results from administrative misconfiguration rather than a flaw in the contract's logic.

The issue is classified as **low severity** because it arises from an administrative error and is unlikely to occur during normal operations.

Recommendation

We recommend adding a check to ensure that the address in the `senderToPausable` mapping is properly initialized before attempting to call `pause` or `unpause`.

Client's Commentary

> I will consider if this is worth adding.
> Again we are not advertising any external parties to attempt to call the senderPause and senderUnpause functions, so its not needed to make this error path nicer as no external developers will be using it.

L-13	Incorrect Handling of `NATIVE` Parameter in `depositWithPermit`
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the TellerWithMultiAssetSupport.sol#218 function of the `TellerWithMultiAssetSupport` contract.

If the `asset` parameter is set to `NATIVE`, the function does not handle it correctly, resulting in an undesired revert. This only occurs when a user mistakenly provides an invalid parameter, causing an inelegant revert without broader impact.

The issue is classified as **low severity** because it stems from user error and only affects the error handling, not the contract's core functionality.

Recommendation

We recommend adding validation for incorrect input or providing clearer error messages when native assets are used with `depositWithPermit`.

Client's Commentary

> cmETH is only accepting mETH for deposit assets, so NATIVE will not be in the isSupported array.

L-14	Inconsistent Naming Convention: `changeDecimals`
Severity	Low
Status	Fixed in 4cb7bda2

Description

This issue has been identified within the AccountantWithRateProviders.sol#424 function of the `AccountantWithRateProviders` contract.

The function name does not adhere to the naming conventions followed in the rest of the contract, which may cause confusion and reduce overall code clarity.

The issue is classified as **low severity** because it only impacts code readability and consistency.

Recommendation

We recommend renaming the function to align with the project's established naming conventions to improve code clarity.

Client's Commentary

> Fixed here: PR-7

L-15	Missing Event Emission in `withdrawNonBoringToken`
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the DelayedWithdraw.sol#397 function of the `DelayedWithdraw` contract.

The function does not emit any event when a withdrawal is made, reducing the contract's transparency and making it harder to track actions. Event emissions are crucial for maintaining auditability and transparency in blockchain contracts.

The issue is classified as **low severity** because it impacts auditability but does not affect the contract's functionality.

Recommendation

We recommend emitting an event each time the `withdrawNonBoringToken` function is called to improve transparency and ease of tracking.

Client's Commentary

> Withdrawing ERC20s from the delayed withdrawer could be a common strategy practice, like if some users cancel their withdraw, or if their withdraws become stale. That being said the event might be helpful for strategy management, I will check internally.

L-16	Improve Security by Hashing All Data in `keccak256` in Flash Loan Logic
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the flash loan handling logic in the ManagerWithMerkleVerification.sol#205 function of the `ManagerWithMerkleVerification` contract.

The current implementation hashes only the `userData` using `keccak256`, but it would be more secure to include all relevant flash loan parameters, such as token addresses and amounts. This would prevent potential manipulation of flashloan data and ensure the integrity of the transaction.

The issue is classified as **low severity** because the current implementation is secure, but hashing more data would enhance overall security and integrity.

Recommendation

We recommend hashing all relevant flash loan parameters, including token addresses, amounts, and `userData`, to ensure full data integrity and security.

Client's Commentary

> cmETH will not be using flashloans. Also if the userData is guaranteed to match, if the tokens or amounts were to be changed by some malicous balancer pool, it is very likely that the call would revert because the userData contained actions that needed a certain amount of tokens.

L-17	Double Pause/Unpause in `pause` and `unpause`
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the `pause` and `unpause` functions of the `AcountantWithRateProviders`, `DelayedWithdraw`, `ManagerWithMerkleVerification` and `TellerWithMultiAssetSupport` contracts.

The current implementation allows the `pause` and `unpause` functions to be called multiple times without checking the contract's current state. While this does not cause any on-chain errors, it may lead to off-chain confusion and inefficiencies in tracking the contract's state.

The issue is classified as **low severity** because it does not impact the on-chain logic but could create inconsistencies in off-chain monitoring systems.

Recommendation

We recommend adding checks to ensure the contract is not already in the desired state before invoking `pause` or `unpause`, to improve clarity and avoid redundant actions.

Client's Commentary

> My main hesitation with adding this is consider the following scenario. There are two pausing bots, one of them calls pauseAll(), the other calls senderPause(). If the second bot is able to call senderPause() before the first bot calls pauseAll(), if we have logic to revert if the contract is already paused, then the first bots pauseAll() call would revert. I would prefer to keep these functions as barebones as possible.

L-18	Arbitrage Risk in `bulkDeposit` and `bulkWithdraw`
Severity	Low
Status	Acknowledged

Description

This issue has been identified within the TellerWithMultiAssetSupport.sol#234 and TellerWithMultiAssetSupport.sol#253 functions of the `TellerWithMultiAssetSupport` contract.

Due to market volatility, these functions could potentially allow for arbitrage opportunities. However, both functions are administrative, and according to the developers, they are not planned for use in this project.

The issue is classified as **low severity** because the functions are administrative and unlikely to be used, reducing the risk of exploitation.

Recommendation

We recommend removing the `bulkDeposit` and `bulkWithdraw` functions if they are not planned for use. If they may be used in the future, further auditing should focus on mitigating potential risks related to their usage.

Client's Commentary

> Arbitrage is possible which is why giving an account the ability to call bulkWithdraw is a highly privelaged action.

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