QuillAudits

AUDIT REPORT

November 2025

For

Continuum

Table of Content

Executive Summary 07

Number of Security Issues per Severity 11

Summary of Issues 12

Checked Vulnerabilities 21

Techniques and Methods 23

Types of Severity 25

Types of Issues 26

Severity Matrix 27

Critical Severity Issues 28

1. Privileged ERC20s treated as global owners / authorization bypass 28

2. _c3gov inverts proposal failure logic 30

3. Missing onlyGov modifier allows attackers to call doGov 32

4. Split pulls fresh CTM (double-charge) instead of repartitioning 33

5. Structural week-ratcheting suppresses intended decay 35

High Severity Issues 36

6. mintValueX / burnValueX bypass checkpoint updates — accounting drift 36

7. Missing proposalId Assignment in C3GovernorUpgradeable 38

8. _toUint Cannot Decode ABI-Encoded Dynamic Return Data 40

9. Split child lacks immediate flash protection 41

10. Permanent Denial Of Ancillary Functionality Via Zero Address Attachments 42

11. Fee Bypass Via Zero Config And Unchecked ERC20 Returns 43

12. Delegation vote read DoS when >255 token IDs are delegated 44

13. Withdrawing rewards will be completely compromised if the reward token is changed 46

14. Quorum semantics mismatch for Bravo voting vs how totalVotes is tracked 47

15. transferWholeTokenX removes owner enumeration but does not update balance/supply 48

16. transferWholeTokenX: Incorrect Owner in enumeration removal 49

Medium Severity Issues 51

17. Unbounded slippage + permissionless swap drains fee value 51

18. Fee multiplier / decimals normalization risk — rounding & upgrade risk 52

19. Pausable Bypass in C3DAppManager 53

20. Nonce Reuse in Governance Submission (sendParams, sendMultiParams) 54

21. Missing DApp Blacklist / Allowlist 55

22. Version Constants Hard Code Single RWA Type Breaking Future Upgrades 56

23. ID Mismatch Enables Cross Component Data Corruption 57

24. Invalid URI Enum Defaults Cause Silent Misclassification 58

25. Misuse of GovernorSettings._proposalThreshold as a percentage (BPS) while OZ expects an absolute vote count 59

26. Precision and allocation bug as totalVotes uses totalWeight while applied weights may be zero / smaller 60

27. Dynamic Proposal Threshold Manipulation via Expiring Locks (Privilege Escalation) 61

28. Permanent Locking of Delegation via Unbounded Token Loop (Denial of Service) 63

29. Misuse of GovernorSettings._proposalThreshold as a percentage (BPS) while OZ expects an absolute vote count 65

Low Severity Issues 66

30. MAX_TOKENS iteration limits → fragmentation DoS (ERC20 hot-path revert) 66

31. Missing Storage Gaps 67

32. MPC Address Uniqueness & Partial Deletion in C3DAppManager 67

33. _checkOnCTMRWA1Received stubbed to always true — receiver hook ignored (composability / safety) 68

34. Temporary approvals + clearApprovedValuesErc20 role mismatch — approval/clear race & privileged clearing 69

35. Sentry / external manager dependency — operational / availability risk 70

36. withdraw / withdrawInvested rely on IERC20.balanceOf(address(this)) → custodial / sweep risk 71

37. Node data can be cleared by governance even if the node owner is not marked for removal 72

38. Interface incompleteness 72

39. Commission Rate Lacks Bounds Validation 73

40. Unchecked ERC20 Transfers In Withdrawals, Reward And Dividend Claims 74

41. Expired Offering Can Still Receive Reward Funding 75

42. Token Fragmentation From Partial ERC3525 Transfers 76

43. Always True Return Values Mislead Integrators 77

44. Missing 0x Prefix Validation In _stringToAddress 78

45. Unused External createOriginalTokenId Allows Silent ID Skips 79

46. Missing Metadata Validation Allows Creation of Unexecutable Delta Proposals (Denial of Service) 80

47. setRewardToken does not increase the new reward token's VotingEscrow allowance 81

48. spendAllowance is incorrectly set as a public function allowing any user to reduce token allowances 82

49. Persistent Loop Flag Bug in attachSentryManager 82

50. Persistent Loop Flag Bug in attachStorageManager 83

51. Persistent Loop Flag Bug in attachRWAX 83

52. Persistent Loop Flag Bug in addChainContract 84

53. Missing _disableInitializers() in FeeManager.sol 85

Informational Issues 86

54. PUSH0 Opcode Environment Compatibility 86

55. Misleading Modifier Names (onlyOperator, onlyGov) 86

56. Missing Non-Zero DApp ID Validation 87

57. Redundant Zero / Pubkey Checks in addMpcAddr 87

58. Code Duplication: C3Caller vs C3CallerUtils 87

59. Inaccurate Comment For Local Deployment Helper 88

60. Missing __UUPSUpgradeable_init() in C3UUIDKeeperUpgradeable::initialize 88

61. Absent Events for UUID Lifecycle 88

62. Outdated & Incomplete External Documentation 89

63. Node validation status not set on attachment 89

64. Inaccurate genesis assignment in Rewards.setRewardToken 90

65. Missing emit events: setNodeRemovalStatus, initContracts, setNodeQualityOf 90

66. Misleading Function Comments On Transfer And Getter Returns 91

67. Code-comment mismatch (child lock end time differs from comment) 91

68. Redundant SafeCast / narrow range casts (e.g., quality 0–10 to uint208) 92

69. Deployment Scripts Fail (Import Paths & Initialization Calls) 93

70. No Mechanism To Remove Obsolete Chain Attachments 94

71. Linear URI Metadata Lookups Cause Excess Gas 95

72. Duplicate CreateNewCTMRWA1 Event Emission 95

73. Attachment Interface Reference Drift 96

74. Hex Address Casing Not EIP-55 Checksummed 96

75. initContracts is publicly callable and can be set by anyone once 97

76. Duplicate fee-token additions allow array/mapping divergence and bricked deletion logic 98

77. Missing Event Emission 98

78. Improper Admin Address Validation and Missing Event in setTokenAdmin Function 99

Functional Tests 100

Automated Tests 100

Centralization Risk 101

Threat Model 102

Closing Summary & Disclaimer 108

Executive Summary

Project Name	Continuum DAO
Protocol Type	Crosschain, MPC, Governance (Voting Escrow), Real World Asset (RWA)
Project URL	https://continuumdao.org/
Overview	The system composes three strategic strata: (1) the RWA layer offers structured multi-chain semi-fungible (ERC3525) asset management with slot-based classification and controlled cross-chain state transitions; (2) the C3Caller messaging & governance substrate supplies operator-mediated cross-chain invocation plus fallback recovery and UUID lineage; and (3) the veCTM governance layer anchors long-term incentive alignment and upgrade legitimacy. Together they form a vertically integrated stack where governance defines policy, C3Caller executes cross-chain intents securely, and the RWA contracts enforce asset semantics and compliance surfaces to yield a modular yet interdependent framework for regulated, extensible real-world asset tokenization across chains.
Audit Scope	The scope of this Audit was to analyze the ContinuumDAO Smart Contracts for quality, security, and correctness.
Source Code link	veCTM - https://github.com/ContinuumDAO/vectm C3Caller - https://github.com/ContinuumDAO/c3caller AssetX (RWA) - https://github.com/ContinuumDAO/RWA
Branch	main
Contracts in Scope	**AssetX (RWA)** src/sentry/CTMRWA1Sentry.sol src/sentry/ICTMRWA1SentryUtils.sol src/sentry/ICTMRWA1Sentry.sol src/sentry/ICTMRWA1SentryManager.sol src/sentry/CTMRWA1SentryManager.sol src/sentry/CTMRWA1SentryUtils.sol src/dividend/ICTMRWA1Dividend.sol src/dividend/CTMRWA1Dividend.sol

src/dividend/CTMRWA1DividendFactory.sol
src/dividend/ICTMRWA1DividendFactory.sol
src/deployment/ICTMRWA1TokenFactory.sol
src/deployment/ICTMRWAERC20.sol
src/deployment/CTMRWAERC20.sol
src/deployment/CTMRWADeployInvest.sol
src/deployment/ICTMRWAERC20Deployer.sol
src/deployment/ICTMRWADeployInvest.sol
src/deployment/CTMRWA1InvestWithTimeLock.sol
src/deployment/ICTMRWADeployer.sol
src/deployment/CTMRWADeployer.sol
src/deployment/CTMRWA1TokenFactory.sol
src/deployment/ICTMRWA1InvestWithTimeLock.sol
src/deployment/CTMRWAERC20Deployer.sol
src/storage/ICTMRWA1StorageUtils.sol
src/storage/ICTMRWA1StorageManager.sol
src/storage/CTMRWA1Storage.sol
src/storage/ICTMRWA1Storage.sol
src/storage/CTMRWA1StorageUtils.sol
src/storage/CTMRWA1StorageManager.sol
src/shared/CTMRWAMap.sol
src/shared/ICTMRWAMap.sol
src/utils/CTMRWAUtils.sol
src/utils/CTMRWAProxy.sol
src/managers/IFeeManager.sol
src/managers/FeeManager.sol
src/crosschain/CTMRWAGateway.sol
src/crosschain/CTMRWA1X.sol
src/crosschain/ICTMRWA1XFallback.sol
src/crosschain/ICTMRWAGateway.sol
src/crosschain/ICTMRWA1X.sol
src/crosschain/CTMRWA1XFallback.sol
src/identity/CTMRWA1Identity.sol
src/identity/ICTMRWA1Identity.sol
src/identity/IZkMeVerify.sol
src/core/CTMRWA1.sol
src/core/ICTMRWA1.sol
src/core/ICTMRWA.sol
src/core/ICTMRWA1Receiver.sol

C3Caller
src/dapp/C3CallerDapp.sol
src/gov/C3Governor.sol
src/gov/IC3GovernDapp.sol
src/dapp/IC3CallerDapp.sol
src/gov/C3GovClient.sol

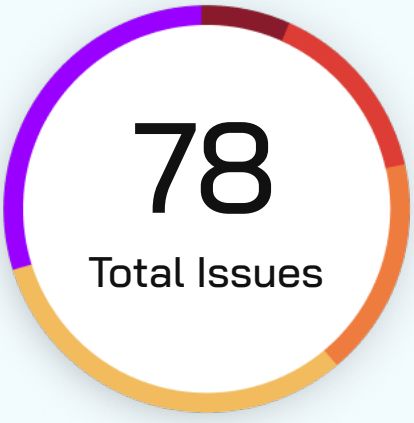
	src/gov/IC3Governor.sol src/gov/IC3GovClient.sol src/gov/C3GovernDapp.sol src/dapp/IC3DappManager.sol src/dapp/C3DappManager.sol src/uuid/C3UUIDKeeper.sol src/uuid/IC3UUIDKeeper.sol src/IC3Caller.sol src/utils/IC3CallerProxy.sol src/utils/C3CallerUtils.sol src/utils/C3CallerProxy.sol src/upgradeable/gov/C3GovernDappUpgradeable.sol src/upgradeable/gov/C3GovClientUpgradeable.sol src/upgradeable/gov/C3GovernorUpgradeable.sol src/upgradeable/C3CallerUpgradeable.sol src/upgradeable/dapp/C3CallerDappUpgradeable.sol src/upgradeable/dapp/C3DappManagerUpgradeable.sol src/C3Caller.sol src/upgradeable/uuid/C3UUIDKeeperUpgradeable.sol **veCTM** VotingEscrow.sol VotingEscrowProxy.sol Rewards.sol NodeProperties.sol GovernorCountingAdvanced.sol
Commit Hash	veCTM: https://github.com/ContinuumDAO/vectm/commit/5ff33b197d49ad61cc7f766351a1f9c40d53e93d C3Caller: https://github.com/ContinuumDAO/c3caller/commit/290716716e7d8399b7063a37595217c85702f4cd RWA: https://github.com/ContinuumDAO/RWA/commit/2c60458708e3030a6c51323f3cbe1b3d2f291e58
Language	Solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	11th August, 2025 - 3rd October, 2025
Updated Code Received	8th November, 2025
Review 2	November 8 - November 21, 2025 and 9th December 2025

Fixed In	veCTM: 5e5269fad69f1810e5737813be1995bbccf8f0bc
	C3Caller: 5e4bceeb1e6d8e0713d403489f7e61414c503d9f
	RWA: 8fcbbab826ec2d445c93bcf74ac07f964cd5678f

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity



Critical	5 (6.4%)
High	11 (14.1%)
Medium	13 (16.6%)
Low	24 (30.9%)
Informational	25 (32.0%)

Severity

Issues	Critical	High	Medium	Low	Informational
Open	0	0	0	0	0
Acknowledged	0	0	0	**3**	**4**
Partially Resolved	0	0	0	0	0
Resolved	**5**	**11**	**13**	**21**	**21**

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Privileged ERC20s treated as global owners / authorization bypass	Critical	Resolved
2	_c3gov inverts proposal failure logic	Critical	Resolved
3	Missing onlyGov modifier allows attackers to call doGov	Critical	Resolved
4	Split pulls fresh CTM (double-charge) instead of repartitioning	Critical	Resolved
5	Structural week-ratcheting suppresses intended decay	Critical	Resolved
6	mintValueX / burnValueX bypass checkpoint updates — accounting drift	High	Resolved
7	Missing proposalId Assignment in C3GovernorUpgradeable	High	Resolved
8	_toUint Cannot Decode ABI-Encoded Dynamic Return Data	High	Resolved
9	Split child lacks immediate flash protection	High	Resolved

Issue No.	Issue Title	Severity	Status
10	Permanent Denial Of Ancillary Functionality Via Zero Address Attachments	High	Resolved
11	Fee Bypass Via Zero Config And Unchecked ERC20 Returns	High	Resolved
12	Delegation vote read DoS when >255 token IDs are delegated	High	Resolved
13	Withdrawing rewards will be completely compromised if the reward token is changed	High	Resolved
14	Quorum semantics mismatch for Bravo voting vs how totalVotes is tracked	High	Resolved
15	transferWholeTokenX removes owner enumeration but does not update balance/supply	High	Resolved
16	transferWholeTokenX: Incorrect Owner in enumeration removal	High	Resolved
17	Unbounded slippage + permissionless swap drains fee value	Medium	Resolved
18	Fee multiplier / decimals normalization risk — rounding & upgrade risk	Medium	Resolved

Issue No.	Issue Title	Severity	Status
19	Pausable Bypass in C3DAppManager	Medium	Resolved
20	Nonce Reuse in Governance Submission (sendParams, sendMultiParams)	Medium	Resolved
21	Missing DApp Blacklist / Allowlist	Medium	Resolved
22	Version Constants Hard Code Single RWA Type Breaking Future Upgrades	Medium	Resolved
23	ID Mismatch Enables Cross Component Data Corruption	Medium	Resolved
24	Invalid URI Enum Defaults Cause Silent Misclassification	Medium	Resolved
25	Misuse of GovernorSettings._proposalThreshold as a percentage (BPS) while OZ expects an absolute vote count	Medium	Resolved
26	Precision and allocation bug as totalVotes uses totalWeight while applied weights may be zero / smaller	Medium	Resolved
27	Dynamic Proposal Threshold Manipulation via Expiring Locks (Privilege Escalation)	Medium	Resolved

Issue No.	Issue Title	Severity	Status
28	Permanent Locking of Delegation via Unbounded Token Loop (Denial of Service)	Medium	Resolved
29	Misuse of GovernorSettings._proposalThreshold as a percentage (BPS) while OZ expects an absolute vote count	Medium	Resolved
30	MAX_TOKENS iteration limits → fragmentation DoS (ERC20 hot-path revert)	Low	Resolved
31	Missing Storage Gaps	Low	Resolved
32	MPC Address Uniqueness & Partial Deletion in C3DAppManager	Low	Resolved
33	_checkOnCTMRWA1Received stubbed to always true — receiver hook ignored (composability / safety)	Low	Resolved
34	Temporary approvals + clearApprovedValuesErc20 role mismatch — approval/clear race & privileged clearing	Low	Resolved
35	Sentry / external manager dependency — operational / availability risk	Low	Acknowledged

Issue No.	Issue Title	Severity	Status
36	withdraw / withdrawInvested rely on IERC20.balanceOf(address(this)) → custodial / sweep risk	Low	Resolved
37	Node data can be cleared by governance even if the node owner is not marked for removal	Low	Resolved
38	Interface incompleteness	Low	Resolved
39	Commission Rate Lacks Bounds Validation	Low	Resolved
40	Unchecked ERC20 Transfers In Withdrawals, Reward And Dividend Claims	Low	Resolved
41	Expired Offering Can Still Receive Reward Funding	Low	Resolved
42	Token Fragmentation From Partial ERC3525 Transfers	Low	Acknowledged
43	Always True Return Values Mislead Integrators	Low	Acknowledged
44	Missing 0x Prefix Validation In _stringToAddress	Low	Resolved
45	Unused External createOriginalTokenId Allows Silent ID Skips	Low	Resolved

Issue No.	Issue Title	Severity	Status
46	Missing Metadata Validation Allows Creation of Unexecutable Delta Proposals (Denial of Service)	Low	Resolved
47	setRewardToken does not increase the new reward token's VotingEscrow allowance	Low	Resolved
48	spendAllowance is incorrectly set as a public function allowing any user to reduce token allowances	Low	Resolved
49	Persistent Loop Flag Bug in attachSentryManager	Low	Resolved
50	Persistent Loop Flag Bug in attachStorageManager	Low	Resolved
51	Persistent Loop Flag Bug in attachRWAX	Low	Resolved
52	Persistent Loop Flag Bug in addChainContract	Low	Resolved
53	Missing _disableInitializers() in FeeManager.sol	Low	Resolved
54	PUSH0 Opcode Environment Compatibility	Informational	Acknowledged
55	Misleading Modifier Names (onlyOperator, onlyGov)	Informational	Resolved

Issue No.	Issue Title	Severity	Status
56	Missing Non-Zero DApp ID Validation	Informational	Resolved
57	Redundant Zero / Pubkey Checks in addMpcAddr	Informational	Resolved
58	Code Duplication: C3Caller vs C3CallerUtils	Informational	Resolved
59	Inaccurate Comment For Local Deployment Helper	Informational	Resolved
60	Missing __UUPSUpgradeable_init() in C3UUIDKeeperUpgradeable::initialize	Informational	Resolved
61	Absent Events for UUID Lifecycle	Informational	Resolved
62	Outdated & Incomplete External Documentation	Informational	Acknowledged
63	Node validation status not set on attachment	Informational	Resolved
64	Inaccurate genesis assignment in Rewards.setRewardToken	Informational	Resolved
65	Missing emit events: setNodeRemovalStatus, initContracts, setNodeQualityOf	Informational	Resolved
66	Misleading Function Comments On Transfer And Getter Returns	Informational	Resolved

Issue No.	Issue Title	Severity	Status
67	Code-comment mismatch (child lock end time differs from comment)	Informational	Resolved
68	Redundant SafeCast / narrow range casts (e.g., quality 0–10 to uint208)	Informational	Resolved
69	Deployment Scripts Fail (Import Paths & Initialization Calls)	Informational	Resolved
70	No Mechanism To Remove Obsolete Chain Attachments	Informational	Acknowledged
71	Linear URI Metadata Lookups Cause Excess Gas	Informational	Resolved
72	Duplicate CreateNewCTMRWA1 Event Emission	Informational	Resolved
73	Attachment Interface Reference Drift	Informational	Resolved
74	Hex Address Casing Not EIP-55 Checksummed	Informational	Acknowledged
75	initContracts is publicly callable and can be set by anyone once	Informational	Resolved
76	Duplicate fee-token additions allow array/mapping divergence and bricked deletion logic	Informational	Resolved
77	Missing Event Emission	Informational	Resolved

Issue No.	Issue Title	Severity	Status
78	Improper Admin Address Validation and Missing Event in setTokenAdmin Function	Informational	Resolved

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Computations Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls

- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw
- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ Unsafe type inference
- ✓ Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC standards
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods, and tools were used to review all the smart contracts:

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Static Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Severity Matrix

Impact

Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Critical Severity Issues

Privileged ERC20s treated as global owners / authorization bypass

Resolved

Path

CTMRWA1.sol

Function Name

`isApprovedOrOwner()`

Description

Any address in _erc20s is treated as an approver/owner for all tokenIds. If an attacker or compromised deployer gets an ERC20 address registered in _erc20s, they can bypass owner approvals and move/spend tokenIds across the system.

Worst case scenario: Attacker registers or otherwise causes an attacker-owned ERC20 to be marked privileged and then drains value from many tokenIds — systemic theft.

```
function isApprovedOrOwner(address _operator, uint256 _tokenId) public view virtual returns (bool) {
    if (!_exists(_tokenId)) {
        revert CTMRWA1_IDNonExistent(_tokenId);
    }
    address owner = CTMRWA1.ownerOf(_tokenId);
    return (_operator == owner || getApproved(_tokenId) == _operator || _erc20s[_operator]);
}

```

```
address newErc20 = ICTMRWAERC20Deployer(erc20Deployer).deployERC20(
    ID, RWA_TYPE, VERSION, _slot, _erc20Name, _symbol, _feeToken
);

_erc20s[newErc20] = true;
_erc20Slots[_slot] = newErc20;
```

Impact

Theft of funds

Likelihood

High, can be executed by anyone

POC

Prepare: Ensure you have an attacker-controlled ERC20 contract (a simple ERC20 you deploy).
Get the deployer to deploy it: Call the repository's deployErc20(...) flow (or otherwise cause the deployer component to create the token). The code above sets _erc20s[newErc20] = true immediately upon deployment. (Alternatively, if there's an admin function that registers ERC20s, cause it to register your token).

Confirm privilege: Call a view or check storage that shows _erc20s[attackerAddress] == true (or rely on the contract behavior).

Exploit transfer/allowance flow: As the attacker ERC20 address, call the protocol function that triggers value movement relying on isApprovedOrOwner (spendAllowance or a transfer flow). Because isApprovedOrOwner returns true for _erc20s, allowance checks will be bypassed and transfers permitted without owner consent.

Example

Invoke CTMRWA1 spend/transfer functions (or CTMRWAERC20 transfer functions) from the attacker ERC20 address. The code will treat the ERC20 as an approved operator and allow movement.

Results

Funds are moved from tokenIds owned by others to attacker-controlled addresses.

Recommendation

- Remove _erc20s from isApprovedOrOwner or scope _erc20s per-slot/per-token with explicit owner opt-in.
- Gate any privileged ERC20 registration behind governance timelock + multisig and require on-chain owner confirmation before privileged status becomes active.

_c3gov inverts proposal failure logic

Resolved

Path

C3GovernorUpgradeable.sol

Function Name

`_c3gov()`

Description

The upgradeable governor writes a failure flag when the low level call actually succeeds. The original non upgradeable version only flags failure when the call reverts. This inversion silently corrupts governance state. A legitimate successful proposal is recorded as failed. A genuinely failed proposal is recorded as if nothing went wrong. A casual reviewer (or any dashboard that trusts the hasFailed bitmap) is now encouraged to retry a proposal that already executed successfully. Each manual retry can repeat external side effects such as parameter changes or downstream contract calls. Over time this produces divergence between intended single execution semantics and on chain history.

```
function _c3gov(bytes32 _nonce, uint256 _offset) internal {
    uint256 _chainId;
    string memory _target;
    bytes memory _remoteData;

    bytes memory _rawData = _proposal[_nonce].data[_offset];
    (_chainId, _target, _remoteData) = abi.decode(_rawData, (uint256, string, bytes));

    if (_chainId == chainID()) {
        address _to = _target.toAddress();
        (bool _success,) = _to.call(_remoteData);
        if (!_success) { // FIX: Audit Bug #3 Fixed
            _proposal[_nonce].hasFailed[_offset] = true;
        }
    } else {
        _proposal[_nonce].hasFailed[_offset] = true;
        emit C3GovernorLog(_nonce, _chainId, _target, _remoteData);
    }
}
```

Impact

- Repeated execution of state changing calls.
- Loss of reliable audit trail for governance actions.
- Elevated operational risk since stakeholders cannot trust failure accounting.
- Potential expansion of attack surface if second execution interacts with external bridges or token flows producing imbalance or stuck funds.
- Increased cost of incident response and slower mitigation because real failing proposals are invisible.

Likelihood

HIGH

POC

- Attacker drafts a proposal that invokes a sensitive but idempotent looking function like setting an address or toggling a feature flag.
- Proposal executes successfully on chain yet hasFailed is set for that segment
- Community reviewers or a governance operator sees a failed mark and chooses to resubmit or manually replay the call believing the chain never applied it
- The side effect happens twice creating inconsistent configuration or double emitting events that off chain infrastructure interprets as separate authoritative updates
- If the function is not strictly idempotent the second execution can magnify limits counters or exhaust allowances leading to real economic side effects (eg duplicate enabling of upgrade windows or double increment of a supply like value in a downstream module)

Recommendation

- Restore the original conditional so only a reverted call sets failure.
- Add tests that assert a successful call leaves the failure flag unset and a deliberately reverting mock sets it.
- Add a one time data migration plan if any proposals have already been mislabeled to reconcile historical records.

Missing onlyGov modifier allows attackers to call doGov

Resolved

Function Name

`doGov`

Description

The core execution function is externally callable by anyone. There is no modifier that restricts entry to an authorized governance identity. An attacker does not need voting power. They only need to craft the calldata that a passed proposal would have produced and call doGov directly. The system assumes higher level flow controls already occurred so it performs the action. This collapses the entire governance process because deliberation and quorum checks are skipped.

```
function doGov(bytes32 _nonce, uint256 _offset) external {
        if (_offset >= _proposal[_nonce].data.length) {
            revert C3Governor_OutOfBounds();
        }
        if (!_proposal[_nonce].hasFailed[_offset]) {
            revert C3Governor_HasNotFailed();
        }
        _c3gov(_nonce, _offset);
    }

```

Impact

1. Complete bypass of the intended voting and delay model.
2. Unauthorized configuration changes.
3. Potential insertion of malicious operators or redirection of privileged flows.
4. Economic risk emerges if downstream actions can trigger asset transfers or permission escalations.

Likelihood

HIGH

POC

- Attacker enumerates sensitive downstream functions reachable through governance (eg adding an operator address or changing configuration in a managed contract)
- Attacker encodes the payload for that function and supplies arbitrary placeholder chain identifiers if required
- Attacker calls doGov directly from their externally owned account
- The action executes immediately without a recorded proposal lifecycle
- Attacker repeats for a batch of changes to entrench control or sabotage settings before detection

Recommendation

- Add an onlyGov or onlyGovOrTimelock modifier and enforce a minimal state machine (Proposed → Queued → Executable).

Split pulls fresh CTM (double-charge) instead of repartitioning

Resolved

Path

VotingEscrow.sol

Function Name

`split()`

Description

split(_tokenId, _extraction) routes through the create-lock path (CREATE_LOCK_TYPE), which triggers a fresh transferFrom of _extraction CTM from the caller. A split should only move already-locked value; here it pulls new tokens. The later _supply -= _extraction “correction” fixes accounting but leaves the extra CTM stuck in the contract.

```
function split(uint256 _tokenId, uint256 _extraction)
    external
    nonflash(_tokenId)
    checkNotAttached(_tokenId)
    checkNoRewards(_tokenId)
    returns (uint256)
{
    _checkApprovedOrOwner(msg.sender, _tokenId);

    address owner = idToOwner[_tokenId];
    LockedBalance memory _locked = locked[_tokenId];
    if (block.timestamp >= _locked.end) {
        revert VotingEscrow_LockExpired(_locked.end);
    }
    int128 value = _locked.amount;
    int128 extraction = SafeCast.toInt128(SafeCast.toInt256(_extraction));
    int128 remainder = value - extraction;
    assert(remainder > 0);
    assert(extraction + remainder <= value);

    uint256 supply_before = _supply;

    locked[_tokenId] = LockedBalance(remainder, _locked.end);
    _checkpoint(_tokenId, _locked, LockedBalance(remainder, _locked.end));

    uint256 extractionId;
    if (nonVoting[_tokenId]) {
        // create another non-voting lock
        // adding a week to lock duration to prevent rounding down exploit
        extractionId = create_nonvoting_lock_for(_extraction, (_locked.end - block.timestamp) + WEEK, owner);
    } else {
        // create another voting lock
        // adding a week to lock duration to prevent rounding down exploit
        extractionId = _create_lock(_extraction, (_locked.end - block.timestamp) + WEEK, owner, DepositType.SPLIT_TYPE);
    }

    // we need to decrease the supply by _extraction because _deposit_for adds it again, when in reality
    // _supply doesn't change in this operation
    _supply -= _extraction;
    assert(_supply == supply_before);

    emit Split(_tokenId, extractionId, _extraction);

    return extractionId;
}
```

Impact

If the caller previously approved CTM, they lose _extraction CTM on split; those tokens remain in the contract with no paired withdrawal path.
If not approved, split reverts unexpectedly.

Likelihood

HIGH

POC

1. User has a valid veCTM lock and has CTM allowance set for VotingEscrow (e.g., from prior increase_amount).
2. User calls split(tokenId, extraction).
3. At L515 → L1047 → L1109, the contract executes transferFrom(user → VotingEscrow, extraction).
4. Split then deducts _supply locally (L520–L521), but the extra CTM stays in the contract, effectively burning the user's tokens.

Structural week-ratcheting suppresses intended decay

Resolved

Path

VotingEscrow

Function Name

`merge`

Description

Unconditional +1 WEEK additions in both merge and split allow a user to cycle weekly (split dust → merge) and keep remaining lock duration roughly constant instead of linearly decaying.

```
27      uint256 supply_before = _supply;
28      LockedBalance memory _locked0 = locked[_from];
29      LockedBalance memory _locked1 = locked[_to];
30      uint256 value0 = uint256(int256(_locked0.amount));
31      uint256 value1 = uint256(int256(_locked1.amount));
32      // value-weighted end timestamp
33      uint256 weightedEnd = (value0 * _locked0.end + value1 * _locked1.end) / (value0 + value1);
34      // round down to week and then add one week to prevent rounding down exploit
35      // uint256 unlock_time = (((block.timestamp + weightedEnd) / WEEK) * WEEK) + WEEK; // Incorrect
36      uint256 unlock_time = ((weightedEnd / WEEK) * WEEK) + WEEK;
```

Impact

- Sustains near-initial voting power over long periods (≈2x advantage after 30 weeks vs passive baseline in PoC).
- Distorts governance weight without adding value or extending lock time.

Likelihood

HIGH

POC

- Create long/max-duration lock.
- Each week: split(tokenId, dust). Child end = parentRemaining + WEEK.
- (Optional) Wait for +1 second tick.
- merge(parent, child). New end floored then +WEEK → restores lost week.
- Repeat 30 cycles: exploit lock power ≈ constant; baseline decays proportionally (observed ~2.876e23 [exploit] vs ~1.438e23 [baseline]).

Recommendation

- Remove +WEEK in merge (use floored weighted end only).
- In split, set child end = parent end (no added week).
- Add invariant test: repeated split/merge without explicit extension must not increase locked.end.

High Severity Issues

mintValueX / burnValueX bypass checkpoint updates — accounting drift

Resolved

Path

CTMRWA1.sol

Function Name

`mintValueX / burnValueX`

Description

Privileged mintValueX and burnValueX update canonical TokenData.balance but do not update checkpoint arrays or emit TransferValue. Snapshots, dividends, and governance votes that depend on checkpoint history will be inconsistent with canonical balances.

```
function mintValueX(uint256 _toTokenId, uint256 _slot, uint256 _value)
        external
        onlyMinter
        whenNotPaused
        returns (bool)
    {
        if (!_exists(_toTokenId)) {
            revert CTMRWA1_IDNonExistent(_toTokenId);
        }
        address owner = ownerOf(_toTokenId);
        string memory toAddressStr = owner.toHexString();


        if (sentryAddr != address(0)) {
            if (!ICTMRWA1Sentry(sentryAddr).isAllowableTransfer(toAddressStr)) {
                revert CTMRWA1_WhiteListRejected(owner);
            }
        }


        TokenData storage toTokenData = _allTokens[_allTokensIndex[_toTokenId]];
        if (toTokenData.slot != _slot) {
            revert CTMRWA1_InvalidSlot(_slot);
        }


        toTokenData.balance += _value;

        return (true);
    }
```

Impact

Worst case scenario: A minter inflates or reduces balances via these shortcuts; snapshot-based governance or dividend calculations are manipulated to benefit attacker or hide supply differences.

Likelihood

HIGH

POC

- **Prerequisites:** Access to an account with onlyMinter role (a test harness or privileged minter in testnet).
- **Make a snapshot baseline:** Query the checkpoint value used by governance/dividends for a given slot (call the checkpoint accessor or snapshot reading function) and record it.
- **Exploit mint path:** Call mintValueX(toTokenId, slot, largeValue) as onlyMinter. The canonical TokenData.balance for toTokenId increases but checkpoint arrays are not updated.
- **Observe divergence:** Read the checkpoint/supply-in-slot (_supplyInSlot[slot].latest() or public snapshot function) and compare to canonical balanceOf(toTokenId). The snapshot remains stale (not showing the new balance), while the canonical balance increased. This allows you to manipulate calculations that use snapshots vs those that use canonical balances.
- **Result:** Using the mismatch, construct scenarios where snapshot-based rights (dividends, governance weight) are misattributed or double-counted.

Recommendation

Make mintValueX/burnValueX call the canonical internal helpers that push checkpoints and emit the appropriate events (or remove shortcuts entirely).

Missing proposalId Assignment in C3GovernorUpgradeable

Resolved

Path

C3Governor.sol

Function Name

`_c3Fallback,`

Description

The upgradeable implementation fails to set _proposal[_nonce].proposalId = _nonce when a proposal is created. The original non upgradeable version performs this assignment so later logic can reliably reference the proposal by its id. In the upgradeable path the proposalId field remains the default bytes32(0). When a cross chain execution attempt fails _c3Fallback() loads proposalId from the struct tied to the failing nonce but receives bytes32(0). It then proceeds to mark the failure status arrays under the zero hash key rather than under the real nonce keyed proposal. The true failing proposal stays marked healthy while the zero keyed proposal (which may correspond to a different real proposal or an empty placeholder) accrues failure flags. This silently corrupts governance bookkeeping and makes retry logic or manual reconciliation unreliable.

```
    function sendParams(bytes memory _data, bytes32 _nonce) external onlyGov {
    _proposal[_nonce].data.push(_data);
    _proposal[_nonce].hasFailed.push(false);
    emit NewProposal(_nonce);
    _c3gov(_nonce, 0);
}

// In fallback after remote failure
bytes32 proposalId = _proposal[_nonce].proposalId; // returns bytes32(0)
uint256 len = _proposal[proposalId].data.length;   // looks at wrong proposal
_proposal[proposalId].hasFailed[len - 1] = true;    // flags wrong slot

```

Impact

- Misattributed failure tracking.
- Genuine failed proposals look healthy and can proceed creating inconsistent governance outcomes.
- Spurious failure flags on the zero hash slot degrade monitoring signal quality and force manual forensic effort.

Likelihood

HIGH

POC

- Governance creates Proposal A with nonce N (example 0x123...) using sendParams
- Missing assignment leaves _proposal[N].proposalId as bytes32(0)
- The cross chain call for Proposal A fails (network issue or target revert)
- _c3Fallback executes and loads proposalId which resolves to zero
- Failure flag is written to _proposal[bytes32(0)] instead of _proposal[N]

Recommendation

- On proposal creation assign _proposal[_nonce].proposalId = _nonce before emitting events or attempting remote execution.
- Add a guard that reverts if proposalId is already set for that nonce to prevent accidental overwrite.

_toUint Cannot Decode ABI-Encoded Dynamic Return Data

Resolved

Path
C3caller

Function Name
`_toUint`

Description
The decoder assumes the raw bytes returned from a low level call map directly to a fixed width unsigned integer. Dynamic return types in Solidity are ABI encoded with an initial offset then a length then the actual data. When such a dynamic payload is encountered the helper cannot parse it and quietly returns a false flag with a zero value. Downstream logic that expects a meaningful non zero numeric result instead proceeds with a default. The absence of an explicit revert or event means operators cannot detect that a critical upstream contract changed its return shape.

Impact
- Silent logical drift between intended and actual data handling.
- Increased difficulty in auditing historical calls since the system did not surface decoding failures.
- Elevated risk of latent downstream bugs that assume a strong invariant on returned numeric ranges.

Likelihood
HIGH

POC
- Integrator or attacker deploys a target contract that intentionally returns a dynamic type (eg bytes or a struct)
- Cross chain call pathway invokes this contract expecting a numeric identifier or count
- Decoder produces zero without signalling an error
- Calling logic treats zero as a legitimate value and may skip validations branches or register incomplete metadata

Recommendation
- Detect dynamic ABI layout by reading the first word as an offset and verifying bounds then decode the length and slice.
- Emit a structured event or revert on unsupported patterns so integrators correct the upstream contract rather than propagating silent zeros.

Split child lacks immediate flash protection

Resolved

Path

VotingEscrow

Function Name

`split`

Description

Child NFT minted during split is not flash-stamped, so it retains full voting power in the same timestamp while parent is temporarily flash-muted.

```
function split(uint256 _tokenId, uint256 _extraction)
    external
    nonflash(_tokenId)
    checkNotAttached(_tokenId)
    checkNoRewards(_tokenId)
    returns (uint256)
{
```

Impact

Future feature additions (delegation/rewards hooks) could allow same-block governance or reward manipulation.
Inconsistent anti-flash model increases auditing complexity.

Likelihood

HIGH

POC

- Call split(parentId, dust).
- Immediately read balanceOfNFT(childId) → non-zero.
- (If implemented) chain further votes/splits/merges within same timestamp then .

Recommendation

After minting child, set ownership_change[childId] = clock() (zero votes until next tick).

Permanent Denial Of Ancillary Functionality Via Zero Address Attachments

Resolved

Path

CTMRWADeployer

Function Name

`deploy, attachContracts , attachDividend, attachStorage, attachSentry, _deployCTMRWA1Local`

Description

A design gap allows initial attachment of address(0) for Dividend, Storage, and Sentry components when the corresponding factory/deployer addresses are unset. The map layer records these zero addresses without validation. Component attach functions only prevent re–attachment (they revert if a non–zero address is already stored) but do not forbid a first write of zero. Once the zero value is persisted, all subsequent attempts to attach a real component revert, permanently disabling that functional surface for the affected RWA ID (dividends, metadata storage operations, or compliance/sentry logic). There is no governance escape hatch or replacement primitive, so remediation requires deploying an entirely new RWA ID and migrating all off–chain references. This constitutes a permanent denial of ancillary functionality (logical brick) rather than a transient misconfiguration.

Impact

Sustains near-initial voting power over long periods (≈2x advantage after 30 weeks vs passive baseline in PoC).
Distorts governance weight without adding value or extending lock time.

Likelihood

HIGH

POC

- Operator (or script) invokes deploy() with one or more ancillary deployer addresses unset, causing the deployer branch to return address(0) for those components.
- CTMRWAMap.attachContracts stores the zero component addresses; _attachCTMRWAID returns true since from its perspective this is the first attachment.
- Later, need arises to enable dividends or storage. Governance attempts to call the relevant attach* function.
- The function detects an existing (zero) address as “already attached” and reverts (e.g., CTMRWA1_DividendAlreadyAttached).
- No contract path exists to overwrite the zero, forcing a full redeploy and remap of the asset ecosystem.

Recommendation

- Add explicit require(_dividendAddr != address(0) && _storageAddr != address(0) && _sentryAddr != address(0)) (or per–component checks) inside attachContracts (and/or each attach* function) for first attachment.
- Provide deployment preflight script/test that asserts all ancillary deployer addresses are configured (non–zero) before invoking deploy().
- Add invariant tests ensuring no persisted zero ancillary addresses post deployment.

Fee Bypass Via Zero Config And Unchecked ERC20 Returns

Resolved

Path
CTMRWA1X, FeeManager

Function Name
`_payFee , getXChainFee`

Description
Fee calculation allows a zero fee when base fees or multipliers are unset. The caller path silently skips transfers when the computed fee is zero. ERC20 interactions rely on raw transferFrom and approve without SafeERC20 semantics or return–value checks for non standard tokens. An attacker selects a whitelisted fee token lacking configured base fees for chosen destination chains or one with multiplier set to zero then calls privileged cross chain operations that should charge a fee. If a non standard token (e.g. returns false) is used or prior fee balances sit in the contract the logic does not detect failed transfers. Operations proceed without economic cost undermining governance fee policy.

If a token returns false on transferFrom attacker can still bypass charges even after fee setup

Impact
Cross chain deployment and administrative lifecycle functions execute at zero cost reducing deterrence and planned revenue. Potential silent under collection fosters economic policy drift.

Likelihood
HIGH

POC
- Governance or misconfiguration leaves base fee = 0 for a fee token on several chains
  Attacker chooses that token and calls deployAllCTMRWA1X with targets requiring fees
- _payFee computes feeWei = 0 and skips transfers and payFee
  Attacker repeats for admin changes and cross chain transfers accruing value without fees

Recommendation
- Enforce feeWei > 0 for operations that are defined as fee bearing or require explicit governance flag to allow zero cost promotional periods.
- Treat missing base fee for any destination chain as a revert unless an allowlist permits zero.
- Wrap all token operations with OpenZeppelin SafeERC20 and assert pre post balance delta equals expected fee.

Delegation vote read DoS when >255 token IDs are delegated

Resolved

Path

CTMRWA1X, CTMRWA1XFallback

Function Name

`mintX , _c3call`

Description

```
VotingEscrow.sol:L1657-L1667
function _calculateCumulativeVotingPower(uint256[] memory _tokenIds, uint256 _t) internal view returns (uint256) {
    uint256 cumulativeVotingPower;
    for (uint8 i = 0; i < _tokenIds.length; i++) {                    // L1659 (uint8 loop index)
        if (nonVoting[_tokenIds[i]]) {
            continue;
        }
        cumulativeVotingPower += _balanceOfNFT(_tokenIds[i], _t);
    }
    return cumulativeVotingPower;
}

VotingEscrow.sol:L837-L840 (uses the function above)
function getVotes(address account) external view returns (uint256) {
    uint256[] memory delegateTokenIdsCurrent = _delegateCheckpoints[account].latest();
    return _calculateCumulativeVotingPower(delegateTokenIdsCurrent, clock());
}
```

The loop index is uint8. If an address has more than 255 veNFT token IDs delegated to it, I++ will overflow in Solidity ⩾0.8 and the call reverts with a panic. Because governance reads voting power through getVotes/getPastVotes, any such read on that address reverts.

Impact

Targeted denial of service on governance actions (propose, vote, quorum accounting for that address). Can be executed without holding significant CTM (many tiny locks) and without interacting with the victim's keys.

Likelihood

HIGH

POC

- Attacker creates many small voting locks (or splits a lock repeatedly) to get >255 distinct veNFT token IDs.
- Attacker calls delegate(victim) (or delegates each token via transfer/delegate flows) so that the victim now has >255delegated token IDs.
- When the Governor (or any caller) tries to read the victim's voting power—getVotes(victim) or getPastVotes(victim, t)—execution reaches _calculateCumulativeVotingPower and reverts at the for
- (uint8 I ...) loop.
  As a result, the victim's propose/castVote attempts revert, effectively DoS'ing their governance participation.

Resolved here

Withdrawing rewards will be completely compromised if the reward token is changed

Resolved

Path

Rewards.sol

Function Name

`setRewardToken, _calculateRewardsOf`

Description

In Rewards.sol when the reward token is changed the previous reward token balance will be fully withdrawn. Furthermore, in _calculateRewardsOf the way rewards are calculated is based on the vePower, baseEmissionRate and nodeEmissionRate historically stored for each day after lastClaimed:

```
    uint256 _baseEmissionRate = baseEmissionRateAt(i);
    uint256 _nodeEmissionRate = nodeEmissionRateAt(i);
    _reward += _calculateRewards(_vePower, _baseEmissionRate, _nodeEmissionRate,
_nodeQuality);
```

The issue will occur when the reward token is changed and users have unclaimed rewards from previous days. When users claim rewards after the update of the reward token, _calculateRewardsOf will still calculate rewards based on the emission rates prior to the update, however these emission rates do not account that the reward token is different and that the Rewards contract no longer holds the previous rewards token. This will be extremely problematic as claimRewards will attempt to claim rewards for the new reward token based on the previous emission rates.

POC

1. The reward token is TokenA and a user has 1e18 of unclaimed rewards. The base emission rate is 2 (simply for the purpose of simplicity). The contract holds 1e18 of TokenA.
2. Before they claim the reward token is changed to TokenB, the emission rates are decreased to 1 as the token has a higher value than TokenA. The contract is transferred 0.5e18 of TokenB (same value as 1e18 of TokenA).
3. Now the user will be unable to withdraw their rewards for the time before the update as the base emission rate of 2 will be used, instead of 1. This will cause their rewards to be calculated as 1e18 of TokenB, which is over the contract balance.

Recommendation

The fix is non-trivial, and a long-term solution would require major changes to the contract. A short-term solution would be to make sure that all users have claimed their rewards before the reward token is changed.

Quorum semantics mismatch for Bravo voting vs how totalVotes is tracked

Resolved

Path

GovernorCountingMultiple.sol

Function Name

`_countVote()`

Description

The contract declares in its COUNTING_MODE function:

```
return "support=bravo&quorum=for,abstain;support=delta&quorum=for";
```

This explicitly states that for Bravo-style voting, quorum should be calculated using only For + Abstain votes. This is the standard OpenZeppelin Governor behavior, where Against votes deliberately do not count toward quorum to prevent a perverse incentive where voting against a proposal helps it reach quorum. However, the implementation contradicts this declaration.

In the _countVote function, the critical line:

```
proposalVote.totalVotes += totalWeight
executes unconditionally for all vote types, including Against votes.
Then _quorumReached uses this total:
return quorum(proposalSnapshot(proposalId)) <= proposalVote.totalVotes;
```

POC

An attacker can cast many against votes which count towards proposalVote.totalVotes (quorum) and cause quorum to be reached while for + abstain is small, affecting final outcome or blocking logic that expects different semantics.

Recommendation

Make proposalVote.totalVotes explicitly reflect the sum required by COUNTING_MODE — i.e., in Bravo branch increment totalVotes by weight only when support is For or Abstain (not Against); and for Delta branch continue with applied weights sum.

transferWholeTokenX removes owner enumeration but does not update balance/supply

Resolved

Path

CTMRWA1X.sol

Function Name

`transferWholeTokenX`

Description

The transferPartialTokenX function correctly uses proper burn accounting:

```
ICTMRWA1(ctmRwa1Addr).spendAllowance(msg.sender, _fromTokenId, _value);

_payFee(FeeType.TX, _feeTokenStr, toChainIdStr._stringToArray(), false);

uint256 slot = ICTMRWA1(ctmRwa1Addr).slotOf(_fromTokenId);

ICTMRWA1(ctmRwa1Addr).burnValueX(_fromTokenId, _value);
```

However, CTMRWA1X.transferWholeTokenX (cross-chain whole-token transfer) calls:

```
ICTMRWA1(ctmRwa1Addr).approveFromX(address(0), _fromTokenId);
ICTMRWA1(ctmRwa1Addr).clearApprovedValues(_fromTokenId);

ICTMRWA1(ctmRwa1Addr).removeTokenFromOwnerEnumeration(msg.sender, _fromTokenId);
```

but does not call the equivalent of _burn or update _balance[owner][slot] and _supplyInSlot[slot]. The public removeTokenFromOwnerEnumeration only swaps/populates the owner list and sets owner=address(0) but does not remove the token from _allTokens or update checkpointed balances & slot supply. This leaves token data inconsistent.

Impact

Routine use is sufficient to corrupt supply: any authorized whole-token cross-chain transfer will mint on the destination without a corresponding burn on the source, inflating supply even without a malicious actor.

Recommendation

Replace removeTokenFromOwnerEnumeration(...) usage with a call that does full burn if moving token off chain — i.e., call ICTMRWA1(ctmRwa1Addr).burnValueX(_fromTokenId, value).

transferWholeTokenX: Incorrect Owner in enumeration removal

Resolved

Path

CTMRWA1X.sol

Function Name

`transferWholeTokenX`

Description

In transferWholeTokenX, the contract uses msg.sender instead of the actual token owner when removing the token from the owner's list. This means if an approved operator (not the owner) executes the transfer, the token remains listed under the original owner. The root cause is that the code calls removeTokenFromOwnerEnumeration(msg.sender, tokenId) rather than using the true owner address.

```
function transferWholeTokenX(
    string memory _fromAddrStr,
    string memory _toAddressStr,
    string memory _toChainIdStr,
    uint256 _fromTokenId,
    uint256 _ID,
    string memory _feeTokenStr
) public nonReentrant {
    string memory toChainIdStr = _toChainIdStr._toLower();

    (address ctmRwa1Addr,) = _getTokenAddr(_ID);
    address fromAddr = _fromAddrStr._stringToAddress();
    if (!ICTMRWA1(ctmRwa1Addr).isApprovedOrOwner(msg.sender, _fromTokenId)) {
        revert CTMRWA1X_OnlyAuthorized(CTMRWAErrorParam.Sender, CTMRWAErrorParam.ApprovedOrOwner);
    }

    if (toChainIdStr.equal(cIdStr)) {
        address toAddr = _toAddressStr._stringToAddress();
        ICTMRWA1(ctmRwa1Addr).approveFromX(address(this), _fromTokenId);
        ICTMRWA1(ctmRwa1Addr).transferFrom(fromAddr, toAddr, _fromTokenId);
        ICTMRWA1(ctmRwa1Addr).approveFromX(toAddr, _fromTokenId);
        _updateOwnedCtmRwa1(toAddr, ctmRwa1Addr);
    } else {
        (, string memory toRwaXStr) = _getRWAX(toChainIdStr);

        _payFee(FeeType.TX, _feeTokenStr, toChainIdStr._stringToArray(), false);

        (, uint256 value,, uint256 slot,,) = ICTMRWA1(ctmRwa1Addr).getTokenInfo(_fromTokenId);

        ICTMRWA1(ctmRwa1Addr).approveFromX(address(0), _fromTokenId);
        ICTMRWA1(ctmRwa1Addr).clearApprovedValues(_fromTokenId);

        ICTMRWA1(ctmRwa1Addr).removeTokenFromOwnerEnumeration(msg.sender, _fromTokenId);

        string memory funcCall = "mintX(uint256,string,string,uint256,uint256)";
        bytes memory callData = abi.encodeWithSignature(funcCall, _ID, _fromAddrStr, _toAddressStr, slot, value);

        _c3call(toRwaXStr, toChainIdStr, callData);

        emit Minting(_ID, _toAddressStr, toChainIdStr);
    }
}
```

Owner Alice approves Bob to transfer a token. Bob calls transferWholeTokenX("Alice", ...). Since Bob is an approved operator, the transfer proceeds but the enumeration removal uses msg.sender (Bob). Alice's list still contains the token. Now Alice and the remote chain both see the token, duplicating it.

Recommendation

Use the actual owner address (e.g. pass fromAddr instead of msg.sender), so the token is removed from the correct owner's record. For example:

```
address trueOwner = ICTMRWA1(ctmRwa1Addr).ownerOf(_fromTokenId);
ICTMRWA1(ctmRwa1Addr).removeTokenFromOwnerEnumeration(trueOwner, _fromTokenId);
```

Medium Severity Issues

Unbounded slippage + permissionless swap drains fee value

Resolved

Description

Function is permissionless once _swapEnabled is true.

Swaps use amountOutMinimum: 0, i.e., no slippage protection.
There is no validation of _deadline (passed straight through at L380). A caller can set a far-future deadline, allowing inclusion in later blocks at stale/unfavorable prices; or set block.timestamp offering no buffer window.

Impact

- An external caller can trigger a swap of the entire feeToken balance (bounded only by contract balance) at arbitrarily bad prices by manipulating pool liquidity/price right before inclusion → the contract receives minuscule rewardToken, destroying fee value meant for ve holders.
- Missing deadline validation amplifies the exposure by allowing long-lived mempool validity (far-future deadlines), increasing the window for MEV/sandwich manipulation and stale execution.

Likelihood

High

Recommendation

Implement slippage checks by comparing expected yield and using that as the amountOutMinimum

Continuum DAO Team's Comment

The function swapFeeToReward has been removed and as such related issues are moot.

Fee multiplier / decimals normalization risk — rounding & upgrade risk

Resolved

Description

Fee calculations use raw baseFee values and assume decimals; if a whitelisted token has nonstandard decimals or an upgradeable token changes decimals/behavior, fees become economically incorrect (undercharged/overcharged).

```
uint256 fee = baseFee * getFeeMultiplier(_feeType);
```

Impact

A whitelisted upgradable token changes decimals (or has fee-on-transfer/rebase), causing severe under/over-charging — financial losses or denial of service.

Create test which include MaliciousERC20 with arbitrary decimals:

```
contract MaliciousERC20 is ITestERC20, ERC20 {
    uint8 _decimals;
    constructor(string memory _name, string memory _symbol, uint8 decimals_) { ... }
}
```

High-level PoC — step by step (followable):

1. Whitelist token T with decimals = 18 and set baseFee accordingly (e.g., expecting 1 token => 1e18).
2. Upgrade T to decimals = 0 (if T is upgradeable) or register an intentionally malicious token with decimals = 0. In tests this is modelled by MaliciousERC20.
3. Invoke fee flow: call the flow that calculates and charges fee (the math uses baseFee directly). The baseFee value no longer corresponds to 1 token — fee magnitudes are now wrong (1e18 of a 0-decimals token is huge or nonsensical).
4. Result: fees are economically incorrect (undercharge or huge overcharge), potentially breaking the protocol or allowing free operations.

Recommendation

Normalize amounts to canonical decimals (e.g., 18) in the contract; validate decimals() of fee tokens when whitelisting and disallow upgradeable tokens or require timelocked governance for their addition.

Pausable Bypass in C3DAppManager

Resolved

Path

C3DAppManager

Function Name

`deposit, withdraw, charging`

Description

Core asset and accounting functions do not check the paused state. During an incident, responders may believe that activating a pause across the system halts risky flows. These functions remain callable thereby allowing an attacker or compromised admin to continue moving value or reshaping internal balances while defenders assume containment. This gap weakens the reliability of exploit-day runbooks and extends the time window for draining or obfuscating movements.

Impact

- Extended exploit window and higher potential loss.
- Increased recovery cost and incident complexity because responders may need to redeploy or revoke
- broader permissions to achieve stoppage.

Likelihood

HIGH

Recommendation

Add whenNotPaused to every externally callable state changing function that is not required for controlled unpause or emergency withdrawal.

Nonce Reuse in Governance Submission (sendParams, sendMultiParams)

Resolved

Path

C3Governor

Description

Proposal submission trusts the caller to supply a unique nonce. There is no storage level assertion that a nonce was never used. A malicious actor or an inattentive operator can intentionally or accidentally reuse an existing nonce with different calldata. Observers that index proposals by nonce will conflate two distinct intents. If off chain execution or cross chain relayers rely on a once only invariant they may replay or discard the wrong payload. This erodes the chronological integrity of the governance ledger and creates plausible deniability about which action was authorized first.

Impact

- Potential inconsistent cross environment state.
- Slower recovery and increased coordination cost.
- Amplified risk when actions sequence dependent because subsequent proposals may assume the first variant executed when in fact the second took precedence for some observers.

Likelihood

HIGH

POC

- Legitimate governance submits a benign configuration update with nonce N
- Attacker observes the event and quickly submits a second proposal with the same nonce N but a more aggressive payload before downstream consumers finalize indexing
- Some watchers record the second payload as the authoritative action while others have already acted on the first
- Divergent external effects appear and reconciliation requires manual intervention opening opportunities for narrative manipulation or social engineering

Recommendation

Maintain a boolean mapping usedNonce and enforce unused on submission then set it atomically.

Missing DApp Blacklist / Allowlist

Resolved

Path

C3DAppManager

Description

The registration flow does not maintain any deny list or explicit status field for deprecating or quarantining a DApp identifier. Once an id has been used and later deemed unsafe there is no on chain enforcement preventing a new party from re registering it. Participants or tooling that treat presence alone as a sign of ongoing legitimacy may unknowingly trust a reintroduced identifier that points to a different operational entity. The system therefore cannot express lifecycle states like suspended or revoked and cannot block reinstatement attempts.

Likelihood

HIGH

POC

- A DApp id is retired after a security concern and community front ends cache that it previously existed
- Attacker registers the same id supplying infrastructure they control
- Unsuspecting integrators resume interaction under the assumption it is the original service
- Attacker leverages this trust channel to solicit cross chain actions or capture fee flows associated with that id

Recommendation

- Introduce a status enum (Active; Suspended; Deprecated) or at minimum a blacklisted mapping.
- Enforce that registration of a blacklisted id always reverts.
- Provide governance controlled functions to transition states with events.

Version Constants Hard Code Single RWA Type Breaking Future Upgrades

Resolved

Path

CTMRWA1X

Description

Upgradeable orchestrator CTMRWA1X inlines compile time constants for type and version while exposed external functions accept _rwaType and _version parameters during deployments or extensions. Mixed dynamic vs static usage cements all internal paths to (1,1). Upgrading implementation to a new literal breaks discovery of previously deployed asset components causing empty map lookups and revert cascades. No dual mode compatibility logic or migration path exists.

Impact

Governance or dev upgrade triggers systemic availability outage and potential perception of asset loss though contracts still deployed under prior key.

Likelihood

HIGH

POC

- New implementation sets VERSION = 2 and upgrades proxy
- Calls to manage existing ID assets internally query map with (1,2) buckets
- Lookups return empty causing operational reverts across admin change mint slot creation
- Teams rush emergency downgrade; window of halted functionality persists

Recommendation

- Replace constants with storage variables initialized once and upgradable only via governance with migration routine that seeds mapping pointers.
- Accept dynamic params but internally reference stored canonical values. Provide explicit multi version router if supporting concurrent versions.

ID Mismatch Enables Cross Component Data Corruption

Resolved

Path

CTMRWA1StorageManager

Function Name

`deployStorage`

Description

deployStorage does not verify that supplied _ID matches the ID encoded within the token contract at _tokenAddr. A compromised deployer or misrouted call can bind a storage contract under a mismatching ID enabling partial cross chain sync to incorrect data sets. Subsequent URI synchronization operations may record metadata under a wrong asset identity.

Impact

Metadata integrity loss and potential regulatory or disclosure misrepresentation across chains.

POC

- Compromised deployer calls deployStorage with _ID = targetID and _tokenAddr referencing different token carrying attacker controlled data
- Storage address returned and registered for target ID
- Future addURI operations propagate manipulated metadata trusting storage contract
- Off chain consumers ingest corrupted data set

Recommendation

Read actualID = ICTMRWA1(_tokenAddr).ID() and revert on mismatch. Emit event logging validated pairing.

Invalid URI Enum Defaults Cause Silent Misclassification

Resolved

Path

CTMRWA1StorageManager

Function Name

`_uToCat, _uToType`

Description

Enum translation functions map any unsupported uint8 to zero value which corresponds to ISSUER or CONTRACT rather than reverting. Malformed cross chain packets or crafted inputs could inflate perceived issuer level disclosures while actual intended category lost. No input range validation protects classification integrity.

Impact

Data layer trust degradation and potential regulatory reporting confusion.

Likelihood

HIGH

POC

- Malicious off chain packager crafts packet with _uriCategory = 99
- _uToCat maps to default ISSUER
- Analytics treat record as authoritative issuer disclosure
- Consumer decisions influenced by mislabeled data

Recommendation

- Require value < enum length else revert.
- Introduce explicit UNKNOWN sentinel separate from material categories.

Misuse of GovernorSettings._proposalThreshold as a percentage (BPS) while OZ expects an absolute vote count

Resolved

Path

CTMDAOGovernor.sol

Function Name

`proposalThreshold()`

Description

CTMDAOGovernor stores a numeric value in GovernorSettings (via the constructor) but then interprets that stored value as a percentage / basis points when computing the actual proposal threshold at runtime. OpenZeppelin's GovernorSettings stores _proposalThreshold as an absolute number of votes; mixing these two semantics is a semantic/storage mismatch that can cause governance to misbehave, be misconfigured, or even become unusable/DoSed.

super.proposalThreshold() returns the raw stored _proposalThreshold (intended by OZ as absolute votes). CTMDAOGovernor incorrectly assumes it's a BPS percent and multiplies by total supply then divides by 100_000.
The same storage slot is therefore being treated as two different units.

Recommendation

Restore OZ semantics (recommended for compatibility). Make proposalThreshold() return the stored absolute value and stop treating it as percentage.

```
function proposalThreshold() public view override(Governor, GovernorSettings)
returns (uint256) {
        // follow OpenZeppelin semantics: stored value is absolute votes
required to propose
    return super.proposalThreshold();
}
```

Precision and allocation bug as totalVotes uses totalWeight while applied weights may be zero / smaller

Resolved

Path

GovernorCountingMultiple.sol

Function Name

`_countVote()`

Description

proposalVote.totalVotes is incremented by totalWeight while the per-option appliedWeight values are computed with integer division and can be zero, so quorum (and Delta-proposal success checks) may be satisfied even though no option received any votes.

In _countVote the contract computes per-option appliedWeight and accumulates totalAppliedWeight:

```
uint256 appliedWeight = (totalWeight * weights[I]) / weightDenominator;
proposalVote.votes[I] += appliedWeight;
totalAppliedWeight += appliedWeight;
```

But after that, the code always does:

```
proposalVote.totalVotes += totalWeight;
```

So proposalVote.totalVotes reflects the voter's entire voting power, while option tallies reflect the rounded down appliedWeight values. The code only asserts that totalAppliedWeight <= totalWeight — it does not use totalAppliedWeight to increment totalVotes.

Quorum and success logic use proposalVote.totalVotes:

```
_quorumReached checks quorum(...) <= proposalVote.totalVotes.
For Delta proposals _voteSucceeded tests  proposalVote.totalVotes > 0.
```

Thus rounding loss can inflate quorum/counts without increasing any option tallies.

POC

1. Voter has totalWeight = 1. They submit weights [1, 99] → denominator = 100.
2. Compute applied weights:
   option0: applied = (1 * 1) / 100 = 0
   option1: applied = (1 * 99) / 100 = 0
3. So totalAppliedWeight = 0. But code does proposalVote.totalVotes += 1. Net result:
   proposalVote.totalVotes increased by 1 (toward quorum),
   proposalVote.votes[0] == 0
   proposalVote.votes[1] == 0
4. Repeat enough such voters to meet quorum(...). The proposal is considered to have sufficient votes although no option received any vote.
   _getWinningIndices will then pick the maximum (all zeros) — this likely resolves to option index 0 (or deterministic tie-breaking behavior), allowing execution of an option with effectively no support.

Recommendation

Count totalAppliedWeight (sum of allocated/applied weights) toward proposalVote.totalVotes instead of totalWeight. That ensures quorum reflects how many votes were actually allocated to options

Dynamic Proposal Threshold Manipulation via Expiring Locks (Privilege Escalation)

Resolved

Path

CTMDAOGovernor.sol

Function Name

`proposalThreshold()`

Description

The CTMDAOGovernor contract overrides proposalThreshold() to compute the proposal eligibility threshold as a percentage of the total voting power in the system. This threshold is dynamically calculated using:

```
  uint256 totalVotingPower = token().getPastTotalSupply(clock() - 1) *
proposalThresholdTsPercentage / 100_000;
```

The total voting power is retrieved from the previous timestamp (clock() - 1), which in this implementation maps directly to block.timestamp - 1, as the underlying VotingEscrow token defines:

```
function clock() public view returns (uint48) {
    return uint48(block.timestamp);
}
```

The vulnerability arises because voting power in VotingEscrow decays over time and drops sharply when users' locks expire. If a large share of the system's total voting power expires at the same timestamp, the getPastTotalSupply() function will report a drastically reduced total supply at that moment.

An attacker can exploit this timing window to create a governance proposal immediately after the major locks expire. Since the threshold calculation uses clock() - 1 (the previous timestamp), and the total voting power at that moment is significantly reduced, the effective proposal threshold becomes extremely small, allowing the attacker to bypass the intended access control mechanism.

POC

1. **Initial State**
   Total voting power: 1,000,000
   Proposal threshold: 1% → requires 10,000 votes.
   Attacker "Alice" holds only 1,000 votes → cannot propose.
2. **Setup**
   Alice observes that several large ve-locks (totaling 950,000 votes) will expire at timestamp T.
3. **Attack Execution**
   At timestamp T, large holders' locks expire, dropping the total voting power to 50,000.
   In the next block (block.timestamp = T + 1), Alice calls propose().
   The governor computes the threshold using: token().getPastTotalSupply(clock() - 1) // clock() - 1 = T
   At timestamp T, total voting power is only 50,000.
   The threshold = 1% × 50,000 = 500.
   Alice's 1,000 votes now exceed the required 500 threshold.
4. **Result**
   Alice successfully creates a proposal, bypassing the intended 10,000-vote requirement.
   The DAO's governance process is compromised, allowing low-stake participants to introduce arbitrary or malicious proposals.

Recommendation

1. Introduce a Minimum Fixed Threshold: Prevent the threshold from ever dropping below a safe baseline:

```
uint256 dynamic = token().getPastTotalSupply(clock() - 1) *
proposalThresholdTsPercentage / 100_000;
uint256 minThreshold = 10_000 * 1e18;
return dynamic > minThreshold ? dynamic : minThreshold;
```

2. Use a More Stable Snapshot Reference: Instead of clock() - 1, use a lagged or averaged snapshot (e.g., 1 week prior) to smooth out sudden total supply drops:

```
token().getPastTotalSupply(clock() - 1 weeks);
```

Continuum DAO Team's Comment

I will fix this by passing a minimal constant threshold to GovernorSettings on deployment — the value passed in on deployment will only be used for this minimal threshold.

Permanent Locking of Delegation via Unbounded Token Loop (Denial of Service)

Resolved

Path

VotingEscrow.sol

Function Name

`_delegate(), _getVotingUnits(), _moveDelegateVotes()`

Description

The delegate() function in VotingEscrow.sol contains a high-severity Denial of Service (DoS) vulnerability. The function iterates over an unbounded list of a user's NFTs in a single transaction. An attacker can exploit this by sending a large number of low-value "dust" NFTs to a victim, causing the gas cost of the delegate() function to exceed the block limit. This attack can permanently lock a user's ability to change their delegate. In the critical edge case where a user has never delegated before, it renders them permanently unable to vote.

The delegate() function is vulnerable because it must enumerate all NFTs owned by a user to update their delegation status. This is performed by the internal _getVotingUnits() function, which contains a for loop that iterates through every token owned by the account.
If a user holds too many NFTs, the gas cost of this enumeration loop will exceed the block gas limit, causing any call to delegate() to fail with an "out of gas" error. The contract has no mechanism to limit the number of NFTs a user can hold.

- Vulnerable Function: _delegate(address account, address delegatee)
- Root Cause: The unbounded loop within _getVotingUnits(address account)

POC

1. Preparation: An attacker creates a veCTM NFT with a minimal amount of CTM (a "dust" NFT).
2. Multiplication: Using the split() function repeatedly, the attacker turns this single, low-value NFT into hundreds or thousands of separate dust NFTs.
3. Attack: The attacker transfers these dust NFTs to a target victim's address. As these are standard ERC721 tokens, the victim cannot reject the transfers.
4. Trigger: When the victim later attempts to call delegate()—either to change their delegate or to enable voting for the first time—the transaction fails. The underlying loop in _getVotingUnits() cannot process the large number of NFTs within the block gas limit.

Recommendation

The fundamental solution is to remove the unbounded loop from the delegation write path.

Immediate Fix (Circuit Breaker): Add a precondition check at the beginning of the delegate() function that reverts if balanceOf(account) exceeds a safe, predetermined limit (e.g., 200). This prevents the transaction from running out of gas and provides a clear error message.

```
function delegate(address delegatee) external {
    if (_balance(msg.sender) > 200) {
        revert TooManyTokensForDelegation();
    }
    // … existing logic
}
```

Continuum DAO Team's Comment

The function _getVotingUnits now returns a state array of the user's owned token IDs. This array is updated whenever a change in their balance occurs. This makes _getVotingUnits O(1).

Fixing _calculateCumulativeVotingPower was more challenging because the voting power is counted retrospectively and the fact that it is a continuous value and not subject to discrete checkpoints.

The suggested fix of implementing a maximum number of iterations eligible for voting power summation would open the door to griefing up to the number of iterations, thus not an option.
The resolution was to implement a global minimumLock value, representing the minimum amount of CTM lockable in an NFT. This governance settable value ensures a significant cost to any would-be DoS attacks and doing so would only stand to benefit the victim monetarily.

Misuse of GovernorSettings._proposalThreshold as a percentage (BPS) while OZ expects an absolute vote count

Resolved

Path

CTMRWA1Dividend.sol

Function Name

`setDividendToken`

Description

In CTMRWA1Dividend.setDividendToken the dividend token can only be changed if the balance of the previous dividend is 0. The issue is that there are several cases in which dividend tokens may become stuck in the contract leading to the contract being unable to change the dividend token. Firstly, if any user has not claimed their dividend, perhaps because it is too low and not enough to cover gas fees, there will be no way for those tokens to be withdrawn, thus setDividendToken will always revert. Secondly, a malicous user can simply transfer 1 wei of the dividend, blocking the update from occuring.

```
1  if (IERC20(dividendToken).balanceOf(address(this)) != 0) {
```

Recommendation

Include a function to withdraw any leftover funds in the contract.

Low Severity Issues

MAX_TOKENS iteration limits → fragmentation DoS (ERC20 hot-path revert)

Resolved

Description

Owner tokenIds are iterated up to MAX_TOKENS. If a holder's balance is fragmented into more than MAX_TOKENS tokenIds, transfers revert. Attackers can force fragmentation and grief users; ordinary users with many small tokenIds may find funds frozen.

Impact

Widespread inability to transfer ERC20s; operational DoS and user funds effectively frozen.

Likelihood

HIGH

Recommendation

Avoid O(N) loops in hot paths;
maintain aggregated balances per owner or provide consolidation functions;
enforce maximum tokenIds per owner on minting or give users a consolidation mechanism.

Continuum DAO Team's Comment

Only pre-approved tokenIds can be used in balance transfers now, so the owner has full control of the number of tokenIds available and the grief/DDoS mechanism alluded to in this issue cannot occur

Missing Storage Gaps

Resolved

Path

C3DAppManagerUpgradeable

Description

The upgradeable contract does not reserve a gap at the end of storage. Future versions that insert new state variables in the inheritance chain risk overwriting existing slots. This turns a safe patch into a latent state corruption event that can only be fixed by migration.

Recommendation

Append a fixed size __gap array sized to anticipated future fields and maintain a documented storage layout file.

MPC Address Uniqueness & Partial Deletion in C3DAppManager

Resolved

Path

C3DAppManager

Function Name

`addMpcAddr,delMpcAddr`

Description

The contract allows the same MPC address to be added multiple times and the delete function stops after removing the first occurrence. Any remaining duplicate remains authorized which breaks operator intent.

Recommendation

- Reject adds when already present.
- Track membership in a mapping and use a swap and pop technique to ensure a single canonical entry.

_checkOnCTMRWA1Received stubbed to always true — receiver hook ignored (composability / safety)

Resolved

Description

Receiver hook always returns true, so safe-transfer guarantees are not enforced: tokens sent to contracts that do not implement the expected receiver may be stuck. Also, malicious recipients can implement hooks to attempt reentrancy.

Impact

- Receiver hook always returns true, so safe-transfer guarantees are not enforced: tokens sent to contracts that do not implement the expected receiver may be stuck.
- Also, malicious recipients can implement hooks to attempt reentrancy.

Likelihood

HIGH

Recommendation

- Implement proper receiver interface checks (call onCTMRWA1Received and revert if not implemented), consistent with ERC721/ERC1155 safe transfer patterns.
- Add nonReentrant guards around external receiver callbacks.

Temporary approvals + clearApprovedValuesErc20 role mismatch — approval/clear race & privileged clearing

Resolved

Description

Approval/transfer/clear sequences combined with clearApprovedValuesErc20 being callable by the ERC20 deployer create a race and role-mismatch risk: approvals can be left in place on failure, and deployers can clear approvals to sabotage.

Impact

Lingering approvals after a failed multi-step transfer allow attackers to spend values, or an attacker controlling the deployer calls clearApprovedValuesErc20 to deny owner recovery.

Recommendation

- Prefer internal atomic transfer helpers (avoid temporary external approvals).
- If approvals are needed, protect with nonReentrant and ensure approve → transfer → clear are atomic or protected by checks/effects/interactions.
- Restrict clearApprovedValuesErc20 role to governance/time-locked operations or require owner consent.

Continuum DAO Team's Comment

The bug fix #1 changed the way that _update in CTMRWAERC20 works. All tokenIds that could be used for transfers now need pre-approval by the owner. The only calls to clearApprovedValuesErc20 happen when the entire tokenId has been consumed in a transfer and then have zero balance.

Sentry / external manager dependency — operational / availability risk

Acknowledged

Description

Critical flows use external managers (Sentry, FeeManager) for whitelist decisions and fees. A malicious or failing manager can block actions or allow bypass..

Impact

Manager down or malicious leads to DoS or bypass of KYC/whitelist.

Recommendation

Treat managers as untrusted: add circuit breakers, multiple attestations, default fail-safe behavior, and timelocked governance for address changes.

Continuum DAO Team's Comment

All critical functions in manager contracts are controlled by Governance - specifically an OpenZeppelin Governor contract on one chain, with cross-chain extensions through C3Governor. This means that state changes/proxy upgrades are inherently timelocked, since the Governor will be set to have a 10 day voting period with a 7 day cool-off period before it can be executed (by anyone). During this 7 days period, anyone can see what the consequences of the proposalId are and take action if needed. Any circuit breakers, or multiple attestations as suggested in the Recommendation would break decentralization and could cause as big a risk to users as having no circuit breaker (e.g. a malicious privileged DAO member could pause execution of a manager contract denying everyone the ability to transfer value etc.). All we can do is to try to identify every possible exploitable risk and then rely on DAO Governance.

withdraw / withdrawInvested rely on IERC20.balanceOf(address(this)) → custodial / sweep risk

Resolved

Description

Admin withdraw flows rely on contract balanceOf; tokens accidentally or maliciously sent directly to the contract can be withdrawn by admin (sweep).

Impact

Admin withdraws tokens that users accidentally sent to the contract; perceived theft or custodial risk.

Recommendation

Track escrowedBalances[token] and only allow withdrawals of tracked amounts; treat any sweep as governance-only with timelock and multisig.

Continuum DAO Team's Comment

The withdraw function that was in CTMRWA1InvestWithTimeLockhas been removed. Only the function withdrawInvested remains, which can only withdraw the value invested in a particular Offering at an index. It cannot withdraw any other ERC20 contract balance and it cannot withdraw more than the investment amount. I have added a delta balance check around the safeTransfer functions. If someone accidentally sends tokens to the contract, they cannot then be removed, so no one can accuse the DAO of taking their tokens. They are lost.

Node data can be cleared by governance even if the node owner is not marked for removal

Resolved

Path
NodeProperties.sol

Function Name
`detachNode()()`

Description
Governance can detach a node and wipe metadata without checking _toBeRemoved[_tokenId] or confirming operator intent. If veCTM ownership is transferred post-attachment (nodes apparently transferable), metadata for a token can be cleared unilaterally.

Impact
Not a funds or direct reward theft vector, but can remove node data unexpectedly. Escalation depends on governance trust assumptions.

Recommendation
Require _toBeRemoved[_tokenId] == true OR explicitly document governance's unconditional authority. Optionally block veCTM transfer while attached or auto-detach on transfer with explicit event.

Interface incompleteness

Resolved

Path
INodeProperties, IRewards

Description
Public/external functions used cross-contract (e.g., quality/history queries or status setters) are missing from the interface(s), reducing ABI completeness and increasing integration drift risk.

Impact
Tooling / integration / upgrade safety degradation.

Recommendation
Enumerate all public/external non-internalized functions and add to interfaces

Commission Rate Lacks Bounds Validation

Resolved

Path

CTMRWADeployer.sol

Function Name

`setInvestCommissionRate`

Description

Setter allows zero or values > 10000 (basis points) enabling disabling commission economics or accidental punitive rates beyond 100 percent. No sanity check or event gating. Documentation intends 1 to 10000 inclusive.

Impact

Revenue distortion or user harm via incorrect fee extraction

Recommendation

- Add require(_rate > 0 && _rate <= 10000) and emit event.
- Consider two step change with timelock for increases above threshold.

Unchecked ERC20 Transfers In Withdrawals, Reward And Dividend Claims

Resolved

Path

CTMRWA1InvestWithTimeLock.sol, CTMRWA1Dividend.sol, Rewards.sol

Function Name

`claimReward, claimDividend, receiveFees, _withdrawToken`

Description

Both functions call IERC20.transfer without using SafeERC20. Non standard tokens that return false or tokens with fee on transfer mechanics could silently fail or under deliver rewards and dividends. Claim logic zeroes internal accounting before transfer so a failed transfer permanently burns user entitlement.

Impact

Permanent loss of user due rewards or dividends and trust erosion. No direct theft path but denial of expected proceeds.

Recommendation

- Use SafeERC20.safeTransfer, safeTransferFrom and revert on failure.
- Reorder to perform transfer before zeroing state or store old amount for assert.

Expired Offering Can Still Receive Reward Funding

Resolved

Path

CTMRWA1InvestWithTimeLock.sol

Function Name

`fundRewardTokenForOffering`

Description

Function lacks an explicit check ensuring offering is within active period before accepting reward funding. Post expiry funding confuses investors regarding stale opportunities and may block accounting clarity if rewards accrue after investment period.

Impact

Misleading incentives and potential misallocation of reward capital after intended closure.

Recommendation

- Require block.timestamp <= endTime before accepting reward funding.
- Emit explicit event on late attempt revert.

Token Fragmentation From Partial ERC3525 Transfers

Acknowledged

Path

CTMRWAERC20.sol

Function Name

`transferFrom`

Description

Partial value transfers mint new tokenIds each time without consolidation logic producing unbounded proliferation until hitting MAX_TOKENS. Gas costs for enumeration and future operations grow. No merge or compaction mechanism exists.

Recommendation

Introduce threshold below which values aggregate or merge small tokenIds periodically. Provide batch consolidation function restricted to owner.

Continuum DAO Team's Comment

(1) After fixing issue #1, the owner can select which tokenIds they wish to make available for transfers and then they approve them specifically. There is no more MAX_TOKENS and the user can decide exactly what they want to approve for transfer.

(2) We already provided a public function in CTMRWA1 called function transferFrom(uint256 _fromTokenId, uint256 _toTokenId, uint256 _value) public override nonReentrant whenNotPaused returns (address) This is effectively a merge function. If a user transfers their whole _fromTokenId balance to _toTokenId, then the first tokenId is removed. This functionality is already in the 'My Tokens' section of factory.assetx.org (see the 'merge' button).

(3) For the receiver of the tokenId, they get a single consolidated tokenId, derived from multiple source tokenIds, so the transfer mechanism effectively reduces token fragmentation.

Always True Return Values Mislead Integrators

Acknowledged

Path

CTMRWA1X, CTMRWAGateway.

Function Name

`deployCTMRWA1, adminX, mintX, createNewSlotX, addChainContract`

Description

Functions declare boolean return yet internal logic cannot return false. Comments reference failure conditions that are unreachable. Off chain indexers or dapp code branching on false never triggers leading to latent dead code.

Recommendation

Remove boolean return or ensure explicit revert semantics only. Update NatSpec to reflect behavior.

Continuum DAO Team's Comment

Functions which have the onlyCaller modifier (e.g. the ones you mentioned adminX, mintX, createNewSlotX) are called by C3Caller and expect a return of true to avoid triggering a c3Fallback, though we shall be removing this requirement in the new deployment of C3Caller and we keep it here for backwards compatibility.

Missing 0x Prefix Validation In _stringToAddress

Resolved

Path
CTMRWAUtils.sol

Function Name
`_stringToAddress()`

Description
Function permits 40 hex char string without 0x and shifts result yielding incorrect address rather than reverting. Could misroute transfers when user omits prefix.

Impact
Potential user fund loss due to address corruption.

Likelihood
HIGH

Recommendation
Enforce exact length 42 with 0x prefix else revert. Provide helper that adds 0x if omitted only in UI layer not contract.

Unused External createOriginalTokenId Allows Silent ID Skips

Resolved

Path

CTMRWA1.sol, onlyERC20 modifier, interface declaration ICTMRWA1.sol

Function Name

`createOriginalTokenId`

Description

The external function createOriginalTokenId() simply increments _tokenIdGenerator via the internal _createOriginalTokenId() and returns the new numeric ID without performing any of the side effects normally associated with token creation (no ownership assignment, no enumeration array updates, no slot association, no events). It is currently unused across the codebase (no on-chain callers); however it remains exposed to any address flagged in _erc20s. This produces a divergence between the public interface expectation (that calling a function named "createOriginalTokenId" creates something) and actual behavior (it only advances a counter). If mistakenly invoked in the future by an integrated ERC20 wrapper or off-chain automation, it can advance the counter causing permanent gaps in the tokenId sequence. While gaps are not inherently unsafe, they may undermine assumptions in indexing, analytics, or external tooling that rely on dense iteration from 1.._tokenIdGenerator-1. Future developers might also incorrectly build logic predicated on its existence (e.g. pre-reserving IDs) introducing subtle bugs.

Impact

Limited. Potential for:

- Non-dense tokenId space (gaps) increasing naive for-loop gas if future code iterates contiguous IDs.
- Developer/operator confusion and accidental reliance on a no-op "creation" path leading to missing enumeration or ownership invariants.
- Interface confusion: external integrators may assume ownership/events are established and mis-handle downstream bookkeeping.
- No direct value minting or balance inflation occurs (function does not assign balances or slots), so financial risk is negligible.

Recommendation

- Remove the external function if not required (most conservative).OR implement full creation semantics (enumeration updates, events, ownership) with clear NatSpec documenting intended use.
- Update ICTMRWA1 interface to match removal or enhanced semantics to prevent integrator drift.

Missing Metadata Validation Allows Creation of Unexecutable Delta Proposals (Denial of Service)

Resolved

Path

src/gov/GovernorCountingMultiple.sol

Function Name

`propose(), _buildOperations()`

Description

The GovernorCountingMultiple contract supports multiple-option (Delta) proposals, where metadata defining the options (nOptions, nWinners, and optionIndices) is encoded in calldatas[0].

However, the propose() function does not validate whether the optionIndices provided in the metadata stay within the valid bounds of the targets, values, and calldatas arrays. Because of this, a malicious proposer can include out-of-range indices (e.g., 100 when the arrays have length 2–3) inside calldatas[0].

The proposal passes all validation and can proceed through the normal voting process since the metadata is not checked during voting. When the proposal later reaches the execution phase, the internal functions _buildOperations() and _countOperations() attempt to access elements at those invalid indices:

```
winningOps.targets[execIndex] = allOps.targets[k];  // k from optionIndices
```

This causes an out-of-bounds access and triggers a revert during queue() or execute().

As a result, the proposal — even if successfully passed by the DAO — can never be executed, leading to a permanent Denial of Service (DoS) for that governance decision.

Impact

- The attacker can create proposals that consume governance resources (votes, gas, attention) but are impossible to execute.
- These proposals clog the governance process and cause confusion or wasted effort.
- Since execution reverts permanently, no DAO action (even after passing) can be carried out.
- The issue directly impacts the integrity and reliability of the DAO's decision-making mechanism.

Recommendation

Add a validation step inside propose() (or _extractMetadata()) to ensure that all optionIndices are strictly less than the array lengths:

```
for (uint256 I = 0; I < metadata.nOptions; I++) {
    require(metadata.optionIndices[I] < targets.length, "Governor: invalid
option index");
}
```

setRewardToken does not increase the new reward token's VotingEscrow allowance

Resolved

Path

Rewards.sol

Function Name

`setRewardToken()`

Description

In the constructor of Rewards.sol the approval of the VotingEscrow for the reward token is increased. This is done in order to allow compoundLockRewards to deposit the reward token for the caller:

```
function compoundLockRewards(uint256 _tokenId) external returns (uint256) {
        uint256 _rewards = claimRewards(_tokenId, address(this));
        if (_rewards == 0) {
            revert Rewards_NoUnclaimedRewards();
        }
        IVotingEscrow(ve).deposit_for(_tokenId, _rewards);
        return _rewards;
    }

```

The issue is that if the reward token is changed the approval of the VotingEscrow for the new tokens remains 0. As a result, compoundLockRewards will always fail.

Recommendation

Increase the approval of the VotingEscrow for the new reward token.

Continuum DAO Team's Comment

I will fix this by removing the ability to update the reward token in the Rewards contract - if the protocol wished to update the reward token in future, a new instance of Rewards could be deployed and its address as seen by VotingEscrow/NodeProperties updated.

spendAllowance is incorrectly set as a public function allowing any user to reduce token allowances

Resolved

Path
CTMRWA1.sol

Function Name
`spendAllowance()`

Description
In CTMRWA1.spendAllowance the token allowances of users is reduced. The function is used in transferFrom in order to allow approved addresses to transfer tokens on behalf of others. The issue is that the function is public. This is extremely dangerous as it would allow malicious users to easily reduce the allowances of operators, even though they are not the owners of the token compromising a majority of the protocol's functionalities.

Recommendation
Make the function internal.

Persistent Loop Flag Bug in attachSentryManager

Resolved

Path
CTMRWAGateway.sol

Function Name
`attachSentryManager()`

Description
attachSentryManager handles cross-chain SentryManager contract registrations in sentryManager[rwaType][version][]. If any entry in the batch already exists, new SentryManager contracts after it are skipped due to the persistent flag bug.

The flag (preExisted) stays true for the rest of the batch upon matching any existing entry, so new contracts in the same batch are dropped without warning, affecting cross-chain security infrastructure.

Recommendation
Declare preExisted inside the loop, or reset it at the start of each iteration

Persistent Loop Flag Bug in attachStorageManager

Resolved

Path

CTMRWAGateway.sol

Function Name

`attachStorageManager()`

Description

attachStorageManager manages registrations in storageManager[rwaType][version][]. The persistent flag bug causes only the first new StorageManager contract to be added/updated in a batch, while subsequent new StorageManager contracts are skipped if one existed in the batch.

With preExisted not resetting each iteration, the function silently skips new StorageManager contracts after encountering an existing entry, leaving the storage manager setup incomplete for multi-chain deployments.

Recommendation

Declare preExisted inside the loop, or reset it at the start of each iteration

Persistent Loop Flag Bug in attachRWAX

Resolved

Path

CTMRWAGateway.sol

Function Name

`attachRWAX()`

Description

attachRWAX registers CTMRWA1X contracts for multiple chains by managing arrays rwaX[rwaType][version][] and rwaXChains[rwaType][version]. If a batch contains an existing chain address, all subsequent new RWA1X contracts in that batch are skipped due to the persistent flag bug.

The persistent preExisted flag, declared outside the iteration loop, starts returning false positives after the first existing entry. It is only reset when a new contract is attached, leading to silent failure for new contracts beyond the first match in a batch.

Recommendation

Declare preExisted inside the loop, or reset it at the start of each iteration

Persistent Loop Flag Bug in addChainContract

Resolved

Path

CTMRWAGateway.sol

Function Name

`addChainContract()`

Description

In CTMRWAGateway.addChainContract, a boolean flag preExisted is declared outside the loop and never reset each iteration. Once it becomes true, subsequent chain IDs in the same call won't be added.

Root Cause is that the code sets preExisted = true if a chain already exists, but never clears it before the next loop iteration.

addChainContract manages the chainContract[] array, which stores gateway contract addresses for multiple chains. When a batch includes both new and existing chain entries, the persistent loop flag (preExisted) causes all subsequent new entries after any existing chain to be silently skipped. This breaks multi-chain registration, resulting in incomplete protocol state and failed gateway setups.

Recommendation

Declare preExisted inside the loop, or reset it at the start of each iteration

Missing _disableInitializers() in FeeManager.sol

Resolved

Path

FeeManager.sol

Description

The FeeManager contract is upgradeable and uses a proxy pattern with an initialize() function. While the initializer modifier ensures initialize() can only be called once per proxy instance, the implementation (logic) contract itself is not protected.

If initialize() is called directly on the logic contract, it will execute and set its own storage (isolated from proxy), which could confuse auditors or developers. While this does not directly affect the deployed proxy, it is a recommended best practice to disable initializers in the implementation contract to prevent accidental or misleading initialization.

Recommendation

Consider adding this in the constructor

Informational Issues

PUSH0 Opcode Environment Compatibility

Acknowledged

Description

Contracts compiled against Cancun introduce the PUSH0 opcode. Deploying the same artifact to a pre Cancun chain will fail at creation or execution because the opcode is unknown there. Will cause failed deployments or unexpected halts on legacy networks.

Continuum DAO Team's Comment

We will only deploy C3Caller and AssetX to Cancun chains or higher. We needed to use this later version of Solidity to reduce our code size, since some of the contracts are as big as is deployable (EIP-170, 24 kb limit). We expect most important blockchains to use Cancun or higher soon and we will consider deploying to these chains as and when they do.

Misleading Modifier Names (onlyOperator, onlyGov)

Resolved

Path

C3GovClient, C3GovernDapp

Description

Current modifier names suggest that only a single role is authorized while logic actually allows broader caller sets. This misalignment can cause integrators to mis judge who can trigger sensitive paths. Rename modifiers to reflect composite authority and update NatSpec and documentation.

Missing Non-Zero DApp ID Validation

Resolved

Path
C3DappManager

Function Name
`deposit, withdraw, setDappId`

Description
Functions accept a zero dApp identifier which is often a sentinel meaning uninitialized. Mixingreal records with a zero id complicates analytics and incident review. Require non zero ids and add a test that zero reverts.

Redundant Zero / Pubkey Checks in addMpcAddr

Resolved

Path
C3DappManager

Function Name
`addMpcAddr`

Description
The function repeats the same zero value validation logic making the flow harder to scan and slightly larger in bytecode size. Repetition raises the chance of uneven edits later. Collapse repeated checks into a single block with distinct error codes if needed.

Code Duplication: C3Caller vs C3CallerUtils

Resolved

Description
Shared helper logic exists in more than one contract. A future change applied to only one copy introduces silent divergence and increases review overhead. Consolidate helpers into a single internal library or abstract base.

Inaccurate Comment For Local Deployment Helper

Resolved

Description

Comment states function only called by deployCTMRWA1 yet also reachable through broader deployment flows causing confusion.

Missing __UUPSUpgradeable_init() in C3UUIDKeeperUpgradeable::initialize

Resolved

Path

C3UUIDKeeperUpgradeable

Function Name

`initialize`

Description

The initializer does not call the expected UUPS parent initializer. This can skip internal setup that enforces upgrade constraints and raises doubt about proxy safety.

Absent Events for UUID Lifecycle

Resolved

Path

C3UUIDKeeper, C3UUIDKeeperUpgradeable

Description

Key lifecycle actions do not emit events. External monitors cannot reconstruct history without full trace replay which is slower and less reliable.

Recommendation

Emit indexed events covering UUID value and dApp id.

Outdated & Incomplete External Documentation

Acknowledged

Description

External docs lag the code and omit architecture context. Readers may implement against stale signatures or miss required sequencing patterns for safe integration. Can also auto generate API docs from NatSpec and add an architecture and roles overview.

Node validation status not set on attachment

Resolved

Path

NodeProperties

Function Name

`attachNode`

Description

_nodeValidated[_tokenId] is only ever written to false during detachment and never set to true on attachment or elsewhere.

Recommendation

Set the node validation status to be true when attaching nodes.

Inaccurate genesis assignment in Rewards.setRewardToken

Resolved

Path

NodeProperties

Function Name

`detachNode`

Description

genesis = _firstMidnight is assigned without flooring to midnight boundary while the accrual logic assumes day-aligned epochs (_getLatestMidnight).

Recommendation

Normalize: `uint48 normalized = _firstMidnight - (_firstMidnight % ONE_DAY); genesis = normalized;`

Missing emit events: setNodeRemovalStatus, initContracts, setNodeQualityOf

Resolved

Path

Rewards

Function Name

`setRewardToken`

Description

These state-mutating functions update internal mappings / configuration but produce no event logs. Off–chain indexers, analytics, and incident response tooling rely on events to reconstruct state transitions. Absence of events increases operational opacity and makes forensic reconstruction slower or incomplete.

Recommendation

Emit specific, structured events `(e.g., NodeRemovalStatusSet(tokenId, oldStatus, newStatus, sender), NodeQualitySet(tokenId, oldQuality, newQuality, sender), ContractsInitialized(rewards, timestamp).`

Misleading Function Comments On Transfer And Getter Returns

Resolved

Path
CTMRWA1X.sol, CTMRWAGateway.sol

Function Name
`transferWholeTokenX, getAttachedRWAX`

Description
Comments omit that transfers can be same chain and that getter returns tuple (bool,addressString). Minor clarity gap.

Recommendation
Update NatSpec to reflect actual behavior.

Code-comment mismatch (child lock end time differs from comment)

Resolved

Path
VotingEscrow

Function Name
`split`

Description
The comments state that child lock end remains the same as parent, but implementation sets child duration to (parentRemaining + WEEK) creating an unintended one-week extension and enabling duration ratcheting.

Recommendation
Align logic with comment (remove + WEEK) OR update comment to note intentional extension.

Redundant SafeCast / narrow range casts (e.g., quality 0–10 to uint208)

Resolved

Path

NodeProperties

Function Name

`detachNode`

Description

Values with very small bounded domains (like a node quality score 0–10) are stored or cast into large integer widths (uint208) via SafeCast. This introduces unnecessary verbosity and slight gas overhead from library calls while providing negligible safety improvement over using a narrower native type (e.g., uint8).

Recommendation

Remove unneeded SafeCast.

Deployment Scripts Fail (Import Paths & Initialization Calls)

Resolved

Path

script/*.s.sol

Function Name

`Upgrade.s.sol, DeployAll.s.sol, DeployAllETH.s.sol, CreateProposal.s.sol, Setup.s.sol, Merge.s.sol, SetGov.s.sol, GovernorCall.sol, Proposal.s.sol, RedeployNodeProperties.s.sol`

Description

Deployment / ops scripts uniformly referenced non-existent build/CTMDAOGovernor.sol and build/VotingEscrow.sol paths (likely a legacy artifact of a previous build output layout) and invoked outdated initializer method names (setUp, setRewards) instead of the currently implemented initContracts. These mismatches produced compile-time failures and, if unnoticed, could result in operators editing scripts ad hoc or copying stale logic—raising drift risk between audited source and executed deployment flows.

Recommendation

- Replace import {IGovernor} from “build/CTMDAOGovernor.sol”; with canonical OpenZeppelin governor interface import.
- Replace import {IVotingEscrow} from “build/VotingEscrow.sol”; with a direct relative path to ../src/token/IVotingEscrow.sol.
- Replaced incorrect initializer invocations: setUp(...) / setRewards(...) -> initContracts(...) across affected scripts.
- Add a lightweight CI step: forge build && forge fmt —check on every PR touching script/ to fail fast on path drift.

No Mechanism To Remove Obsolete Chain Attachments

Acknowledged

Path

CTMRWAGateway.sol

Function Name

`attachRWAX`

Description

Gateway supports add or update but never removal leading to stale chain entries persisting. Over time list size increases raising gas for linear scans and operational ambiguity.

Recommendation

Add governed removal that compacts arrays and updates index mapping. Provide off chain script verifying active chain set.

Continuum DAO Team's Comment

It should be noted that if a chain dies, then this does not affect the ability for an RWA token to transfer value, or otherwise interact on other chains, or between other chains that are working. The system is designed to allow continuing functionality.

Linear URI Metadata Lookups Cause Excess Gas

Resolved

Path

CTMRWA1Storage

Function Name

`existURIHash, getURIHash, getURIHashByIndex, getURIHashCount`

Description

O(n) loops for existence and lookup scale linearly with record count incurring high gas on large metadata sets..

Recommendation

Introduce mappings hashExists, hashToIndex, aggregated counters as outlined in raw notes achieving O(1) lookups.

Duplicate CreateNewCTMRWA1 Event Emission

Resolved

Path

CTMRWA1X.sol

Function Name

`.deployAllCTMRWA1X (emit), _deployCTMRWA1Local (emit)`

Description

Same event emitted twice for local creation producing misleading analytics counts.

Recommendation

Remove one emission or add distinct event names for local vs orchestration.

Attachment Interface Reference Drift

Resolved

Description

Interface naming or pointing inconsistency leads to confusion during integration.

Recommendation

Update interface docstring and references to precise contract.

Hex Address Casing Not EIP-55 Checksummed

Acknowledged

Description

System lower cases address strings forfeiting optional checksum error detection.

Recommendation

Optionally implement checksum verification path for front end tooling while maintaining backward compatibility.

initContracts is publicly callable and can be set by anyone once

Resolved

Path

NodeProperties.sol, VotingEscrow.sol

Description

initContracts(address _rewards) can be called by anyone the first time (it only checks rewards != address(0)), which lets an attacker set rewards to an arbitrary contract address.

Recommendation

Restrict initContracts to governance (onlyGov) or to the deployer/owner, or pass the rewards address in the constructor.

Example:

```
function initContracts(address _rewards) external onlyGov {
    if (rewards != address(0)) revert NodeProperties_InvalidInitialization();
    rewards = _rewards;
}
```

Continuum DAO Team's Comment

This is to prevent the lengthy requirement of a passed governance vote to execute minor initialization operations. Subsequent executions require a governance vote.

Duplicate fee-token additions allow array/mapping divergence and bricked deletion logic

Resolved

Path

FeeManager.sol

Function Name

`addFeeToken`

Description

addFeeToken(string) does not check feeTokenIndexMap before pushing — so the same ERC-20 address can be appended to feeTokenList multiple times. Each add overwrites feeTokenIndexMap[feeToken] with the newest index, but older array entries remain.

And if the token was added twice (or more), delFeeToken uses feeTokenIndexMap[feeToken] to locate and remove one occurrence (the one mapping points to), then sets the mapping to 0. Any earlier duplicate(s) in feeTokenList remain in the array but are no longer reachable by the mapping — leaving a ghost/orphan entry.

So, these duplicate token entries can never be removed and stores forever in the feeTokenList.

getFeeTokenList will always include that duplicate entry.

Recommendation

Enforce uniqueness on addFeeToken

Missing Event Emission

Resolved

Path

CTMRWA1.sol

Description

The functions pause() and unpause() are responsible for changing the contract's operational state. However, they do not emit any events to notify users or off-chain systems when these actions occur.

Recommendation

Consider emitting even

Improper Admin Address Validation and Missing Event in setTokenAdmin Function

Resolved

Path

setTokenAdmin.sol

Description

The setTokenAdmin() function updates the tokenAdmin address without validating input or emitting an event.

This allows setting the admin to the zero address (permanent lockout) or same address (redundant update).

Additionally, since no event is emitted, critical admin changes are not traceable on-chain, impacting auditability and transparency.

Recommendation

Consider the validation

Functional Tests

Some of the tests performed are mentioned below:

- Should perform actions only on existing slots
- Should allow only governance to set fee multiplier
- Failed cross-chain mint/transfer state recovers via fallback
- Reentrancy protection across mint, transfer, dividend, investment, UUID flows
- UUID uniqueness & replay resistance
- Upgrade procedures preserve critical mappings/state
- Access control on operators
- Sentry / KYC state transition & validation logic
- Reentrancy protections across mint, transfer, dividend, investment, KYC identity, and fee flows
- Upgrade procedures preserve mappings, guards, constants, and role-based state
- Deterministic deployment & CREATE2 address invariants
- FeeManager access control & pause semantics
- Storage URI uniqueness, nonce monotonicity, and authorization checks

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Centralization Risk

The above audit works under the assumption that all the admin roles will not act malicious under given circumstances.

Cross Chain mintX And Fallback Replay Lacks Nonce

Cross chain mint and token transfer flows rely on messages that contain ID, from and to addresses slot and value. There is no nonce, replay protection bitmap or consumed message hash. A relayer or observer can resubmit an earlier valid mintX message on the destination chain. Each replay mints additional value or recreates slot allocations inflating supply beyond intended source burns or transfers. Fallback handling also lacks a consumed flag so an originally failed path could be replayed after local recovery. If a token returns false on transferFrom attacker can still bypass charges even after fee setup.

There's no finality attached to the mintX/fallback calls (via a timestamp) or replay protection (via an auditable nonce) for each of the individual calls that happen. If for any reason the system to forward calls gets compromised or creates an end-point for users to directly interact, the protection put in place with auto-incrementing nonces via the uuid would only result in multiple successful calls provided the same data.

Loss of burnt token on account of chain reorg

The current structure of the function follows burning the tokens before initiating cross-chain calls. However, under the situation of chain-reorg, all the tokens from both source chain and destination chain will be lost. The risk still exists and isn't fully mitigated yet because we cannot ascertain 100% the functionality of the off-chain component (the relayer).

Threat Model

C3Caller

Contract	Function	Threats
C3Caller.sol	_c3call / c3call / c3broadcast / execute	Verified isActiveChainID gating on every entrypoint to block unsupported chains, ensure status propagation to composed functions, and prevent downstream payload execution on inactive or spoofed networks.
C3DAppManager .sol	withdraw	Checked minimum deposit enforcement, decommissioned token handling, SafeERC20 pulls, and double-withdraw prevention for both ERC20 and native paths.
C3DAppManager .sol	chargePayload	Confirmed onlyActive modifier blocks paused/deactivated DApps, fee token validation, and single-charge guarantee per request to prevent excess billing.
C3DAppManager .sol	chargeGas	Ensured fee collection is decoupled from DApp status so executors are reimbursed even when a DApp becomes inactive, and validated gas accounting cannot be bypassed.
C3DAppManager .sol	removeFeeConfig	Verified it clears fee tuples but preserves gasPerEtherFee so in-flight cross-chain executions keep their reimbursement rate, avoiding stranded executors.

Contract	Function	Threats
C3DAppManager .sol	_deriveDAppID / registerDApp	Audited keccak256(creator, dappKey) derivation for collision resistance across chains and confirmed uniqueness checks prevent ID reuse or aliasing.
C3DAppManager .sol	getAllDAppAddrs / getAllDAppMPCAddrs / getDAppMPCCount / isValidMPCAddr	Identified missing dappIDExists guard on these view helpers; flagged residual threat of reading/null data for unregistered IDs.
C3GovClient.sol	changeGov / applyGov	Tested two-step governance transfer flow: strict proposer/acceptor roles, single pending slot, revert paths on mismatched caller, and immunity to stale proposals.
C3Governor.sol	applySelfAsGov	Confirmed bootstrap pathway only succeeds once (when gov == address(0)), blocks re-entry after initialization, and prevents arbitrary self-assignment later in lifecycle.

veCTM

Contract	Function	Threats
VotingEscrow.sol	merge / split / _create_lock	Looked for lock-duration ratcheting, dust cycling to freeze decay, and invariant drift between comments and implementation.
VotingEscrow.sol	delegate / _moveDelegateVotes	Assessed DoS from unbounded token enumerations, potential vote locking when attackers airdrop dust NFTs, and flash-epoch bypass on freshly split locks.
VotingEscrow.sol	withdraw / liquidate / _withdraw	Checked that attached nodes or pending rewards can't block withdrawals and that liquidation penalties can't be gamed by tiny balances.
VotingEscrow.sol	initContracts / setBaseURI / setMinimumLock	Ensured one-time initializer cannot rerun, metadata pointers can't be hijacked mid-governance, and min-lock values prevent griefing with dust locks.
VotingEscrowProxy.sol	upgradeTo / admin	Reviewed proxy admin ownership, storage slot consistency, and fallback routing so upgrades can't be seized or brick the implementation.
Rewards.sol	setRewardToken / withdrawToken / claim	Investigated allowance resets, reward-token rug switches, and governance-controlled drains of reward/fee tokens without timelocks.
Rewards.sol	updateEmission / checkpoint	Looked for misaligned genesis timestamps, midnight rounding errors, and unchecked math that could overpay or underpay daily accruals.

Contract	Function	Threats
NodeProperties .sol	attachNode / detachNode / setNodeRemovalStatus	Checked that governance can’t wipe or steal node metadata without owner intent, and that nodes transferring ownership don’t bypass removal workflows.
NodeProperties .sol	setNodeQualityOf / initContracts	Ensured quality bounds are enforced, redundant SafeCast usage doesn’t hide truncation, and missing events don’t reduce observability.
CTMDAOGovern or.sol	proposalThreshold / quorum / propose	Focused on dynamic threshold collapse after mass-expiry, quorum mismatches with Bravo counting, and metadata validation preventing unexecutable proposals.
GovernorCounti ngAdvanced.sol	_quorumReached / _voteSucceeded	Verified totalVotes math can’t double-count zero-weight ballots and that custom counting stays aligned with OZ expectations.
ArrayCheckpoin ts.sol	push / upperLookupRecent	Reviewed bounds, overflow, and DoS vectors when storing >255 checkpoints per delegate.

RWA

Contract	Function	Threats
CTMRWA1.sol	mint / transferFrom / approveErc20	Looked for slot underflows, ERC3525-style ID reuse, ERC20 approval bridging, and hooks that could bypass whitelist or fee enforcement.
CTMRWA1Storage.sol	addSlot / updateSlot / getSlot	Verified slot metadata integrity, prevention of duplicate issuers, and bounds on valuation/redemption fields so oracle data can't be spoofed.
CTMRWA1StorageManager.sol	deploy / updateLatestVersion / getSlotInfo	Focused on gateway/feeManager wiring, version mismatches between tokens and manager, and upgrade paths that could mint stale implementations.
CTMRWA1StorageUtils.sol	addURI / addURIX / validateURI	Checked URI category/type enums for invalid states, hashed content collisions, and cross-chain synchronization of issuer provenance data.
CTMRWAMap.sol	registerToken / getTokenContract	Ensured deterministic ID mapping resists collision, preserves per-version history, and blocks malicious rebinds of token IDs across chains.
CTMRWA1X.sol	deployAllCTMRWA1X / mintX / transferX	Assessed cross-chain bridge security: fallback absence, admin token bookkeeping, MPC rotation, and _getRWAX path validation for spoofed chain IDs.
CTMRWA1XUtils.sol	addAdminToken / updateOwnedCtmRwa1	Looked for inconsistent admin/owner arrays, unchecked push-pop logic, and version-awareness when syncing per-user CTMRWA1 sets.

Contract	Function	Threats
CTMRWAGateway.sol	setRemote / executeMessage	Checked message verification, relayer validation, and fee refunds so remote calls can't be replayed or injected.
FeeManager.sol	setFeeConfig / chargeFee / withdrawFee	Evaluated fee multiplier governance, gas-per-ether normalization, fee token registration, and reductions/expiration logic to prevent executor starvation or fee draining.
CTMRWADeployer.sol	deployAllCTMRWA1X / deployERC20 / deployNewInvestment	Reviewed factory permissions, slot reuse, and correct wiring of downstream managers so unauthorized parties can't mint new AssetX instances.
CTMRWAERC20.sol	transfer / balanceOfApproved / burn	Checked ERC3525-to-ERC20 conversion, approved tokenId tracking, and prevention of double counting between ERC20 balances and underlying NFTs.
CTMRWA1Identity.sol	verify / setVerifier	Ensured ZK verifier updates require governance, proof inputs are chain-bound, and attesters can't bypass allowlists.
CTMRWA1SentryManager.sol	setGuardian / pause / unpause	Looked for guardian rotation safeguards, multi-chain pause propagation, and latch controls that prevent stuck paused states.
CTMRWA1Sentry.sol	report / slash / unlock	Checked that sentry incidents can't grief honest operators, slashing obeys quorum, and unlock windows prevent instant rug pulls.
CTMRWA1Dividend.sol	schedule / claim / cancel	Assessed dividend schedule bounds, per-investor accrual, and fee token usage to avoid double payouts or stuck schedules.

Closing Summary

In this report, we have considered the security of Continuum DAO. We performed our audit according to the procedure described above.

5 Critical, 11 High, 13 Medium, 24 Low, and 25 Informational severity issues were found, few of the issues are resolved and the remaining issues are acknowledged by the Continuum Team

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With seven years of expertise, we've secured over 1400 projects globally, averting over $3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

7+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

November 2025

For

Continuum

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com