// Security Assessment

07.01.2025 - 07.09.2025

CMTAT

Taurus

HALBORN

CMTAT - Taurus

Prepared by: HALBORN

Last Updated 08/12/2025

Date of Engagement: July 1st, 2025 - July 9th, 2025

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
11	0	0	0	2	9

TABLE OF CONTENTS

1. Introduction

`Taurus` engaged `Halborn` to perform a security assessment of their smart contracts from July 1st, 2025, to July 9th, 2025. The assessment scope was limited to the smart contracts provided to Halborn. Commit hashes and additional details are available in the Scope section of this report.

The `Taurus` codebase in scope consists of a Solidity implementation of the CMTAT security token framework, featuring modular compliance and technical capabilities for regulated financial assets on EVM-compatible blockchains, featuring compliance controls and upgradeability.

2. Assessment Summary

`Halborn` was allocated 7 days for this engagement and assigned 1 full-time security engineer to conduct a comprehensive review of the smart contracts within scope. The engineer is an expert in blockchain and smart contract security, with advanced skills in penetration testing and smart contract exploitation, as well as extensive knowledge of multiple blockchain protocols.

The objectives of this assessment were to:

- Identify potential security vulnerabilities within the smart contracts.
- Verify that the smart contract functionality operates as intended.

In summary, `Halborn` identified several areas for improvement to reduce the likelihood and impact of potential risks, which were partially addressed by the `Taurus team`. The primary recommendations were as follows:

- `Modify the _canMintBurnByModule() function to respect the paused state.`
- `Modify the _forcedTransfer() function to handle allowances in a safe and predictable manner.`
- `Lock the pragma version to the same version used during development and testing.`

3. Test Approach And Methodology

`Halborn` conducted a combination of manual code review and automated security testing to balance efficiency, timeliness, practicality, and accuracy within the scope of this assessment. While manual testing is crucial for identifying flaws in logic, processes, and implementation, automated testing enhances coverage of smart contracts and quickly detects deviations from established security best practices.

The following phases and associated tools were employed throughout the term of the assessment:

- Research into the platform's architecture, purpose and use.
- Manual code review and walkthrough of smart contracts to identify any logical issues.
- Comprehensive assessment of the safety and usage of critical Solidity variables and functions within scope that could lead to arithmetic-related vulnerabilities.
- Local testing using custom scripts (`Foundry`).
- Fork testing against main networks (`Foundry`).
- Static security analysis of scoped contracts, and imported functions (`Slither`).

4. Caveats

After the initial assessment, Taurus reported an additional compliance issue affecting batch operations. Halborn performed a post-assessment review at commit `198d019` to verify the fixes, and confirmed that the issue has been addressed at that commit.

5. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

5.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

5.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

5.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

6. SCOPE

REPOSITORY

(a) Repository: CMTAT

(b) Assessed Commit ID: 04dad82

(c) Items in scope:

- contracts/deployment/CMTATStandalone.sol
- contracts/deployment/CMTATUpgradeable.sol
- contracts/deployment/CMTATUpgradeableUUPS.sol
- contracts/deployment/ERC1363/CMTATStandaloneERC1363.sol
- contracts/deployment/ERC1363/CMTATUpgradeableERC1363.sol
- contracts/deployment/ERC7551/CMTATStandaloneERC7551.sol
- contracts/deployment/ERC7551/CMTATUpgradeableERC7551.sol
- contracts/deployment/allowlist/CMTATStandaloneAllowlist.sol
- contracts/deployment/allowlist/CMTATUpgradeableAllowlist.sol
- contracts/deployment/debt/CMTATStandaloneDebt.sol
- contracts/deployment/debt/CMTATUpgradeableDebt.sol
- contracts/deployment/light/CMTATStandaloneLight.sol
- contracts/deployment/light/CMTATUpgradeableLight.sol
- contracts/interfaces/engine/IDebtEngine.sol
- contracts/interfaces/engine/IDocumentEngine.sol
- contracts/interfaces/engine/IRuleEngine.sol
- contracts/interfaces/engine/ISnapshotEngine.sol
- contracts/interfaces/modules/IAllowlistModule.sol
- contracts/interfaces/modules/IDebtModule.sol
- contracts/interfaces/modules/IDocumentEngineModule.sol
- contracts/interfaces/modules/ISnapshotEngineModule.sol
- contracts/interfaces/technical/ICMTATConstructor.sol
- contracts/interfaces/technical/IERC20Allowance.sol
- contracts/interfaces/technical/IERC7802.sol
- contracts/interfaces/technical/IMintBurnToken.sol
- contracts/interfaces/tokenization/ICMTAT.sol
- contracts/interfaces/tokenization/IERC3643Partial.sol
- contracts/interfaces/tokenization/draft-IERC1404.sol
- contracts/interfaces/tokenization/draft-IERC1643.sol
- contracts/interfaces/tokenization/draft-IERC1643CMTAT.sol
- contracts/interfaces/tokenization/draft-IERC7551.sol
- contracts/libraries/Errors.sol
- contracts/modules/0_CMTATBaseCommon.sol
- contracts/modules/0_CMTATBaseCore.sol
- contracts/modules/0_CMTATBaseGeneric.sol
- contracts/modules/1_CMTATBaseAllowlist.sol
- contracts/modules/1_CMTATBaseRuleEngine.sol
- contracts/modules/2_CMTATBaseDebt.sol
- contracts/modules/2_CMTATBaseERC1404.sol

- contracts/modules/3_CMTATBaseERC20CrossChain.sol
- contracts/modules/4_CMTATBaseERC2771.sol
- contracts/modules/5_CMTATBaseERC1363.sol
- contracts/modules/5_CMTATBaseERC7551.sol
- contracts/modules/internal/AllowlistModuleInternal.sol
- contracts/modules/internal/ERC20BurnModuleInternal.sol
- contracts/modules/internal/ERC20EnforcementModuleInternal.sol
- contracts/modules/internal/ERC20MintModuleInternal.sol
- contracts/modules/internal/EnforcementModuleInternal.sol
- contracts/modules/internal/ValidationModuleRuleEngineInternal.sol
- contracts/modules/internal/common/EnforcementModuleLibrary.sol
- contracts/modules/wrapper/controllers/ValidationModule.sol
- contracts/modules/wrapper/controllers/ValidationModuleAllowlist.sol
- contracts/modules/wrapper/core/BaseModule.sol
- contracts/modules/wrapper/core/ERC20BaseModule.sol
- contracts/modules/wrapper/core/ERC20BurnModule.sol
- contracts/modules/wrapper/core/ERC20MintModule.sol
- contracts/modules/wrapper/core/EnforcementModule.sol
- contracts/modules/wrapper/core/PauseModule.sol
- contracts/modules/wrapper/core/ValidationModuleCore.sol
- contracts/modules/wrapper/extensions/DocumentEngineModule.sol
- contracts/modules/wrapper/extensions/ERC20EnforcementModule.sol
- contracts/modules/wrapper/extensions/ExtraInformationModule.sol
- contracts/modules/wrapper/extensions/SnapshotEngineModule.sol
- contracts/modules/wrapper/extensions/ValidationModule/ValidationModuleERC1404.sol
- contracts/modules/wrapper/extensions/ValidationModule/ValidationModuleRuleEngine.sol
- contracts/modules/wrapper/options/AllowlistModule.sol
- contracts/modules/wrapper/options/DebtEngineModule.sol
- contracts/modules/wrapper/options/DebtModule.sol
- contracts/modules/wrapper/options/ERC2771Module.sol
- contracts/modules/wrapper/options/ERC7551Module.sol
- contracts/modules/wrapper/security/AccessControlModule.sol

Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- 067244a
- 86dbd2d
- 4d0a72f
- 6b1c32e
- 52e1106
- f6021de

Out-of-Scope: New features/implementations after the remediation commit IDs.

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	2

INFORMATIONAL
9

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MINTING AND BURNING OPERATIONS BYPASS THE PAUSE MECHANISM	LOW	FUTURE RELEASE - 07/16/2025
INSUFFICIENT ALLOWANCE VALIDATION DURING FORCED TRANSFERS	LOW	RISK ACCEPTED - 07/16/2025
FLOATING PRAGMA	INFORMATIONAL	ACKNOWLEDGED - 07/16/2025
MISLEADING RESTRICTION CODE RETURNED FOR DEACTIVATED CONTRACT	INFORMATIONAL	SOLVED - 07/16/2025
COMMENTED FUNCTIONALITY	INFORMATIONAL	SOLVED - 07/16/2025
TYPOS	INFORMATIONAL	SOLVED - 07/16/2025
PUBLIC FUNCTIONS NOT CALLED WITHIN CONTRACTS	INFORMATIONAL	ACKNOWLEDGED - 07/16/2025
MISLEADING COMMENT REGARDING FROZEN BALANCE CALCULATION	INFORMATIONAL	SOLVED - 07/16/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INCONSISTENT METHOD OF CALLING INHERITED FUNCTIONS	INFORMATIONAL	SOLVED - 07/16/2025
LACK OF NAMED MAPPINGS	INFORMATIONAL	SOLVED - 07/16/2025
UNUSED FILE	INFORMATIONAL	ACKNOWLEDGED - 07/16/2025

8. FINDINGS & TECH DETAILS

8.1 MINTING AND BURNING OPERATIONS BYPASS THE PAUSE MECHANISM

// LOW

Description

The `PauseModule` is designed to "prevent execution of transactions on the distributed ledger" and act as an "emergency switch for freezing all token transfers." However, the implementation does not stop administrative minting and burning operations. The `ValidationModule._canMintBurnByModule()` function, which validates mints and burns, checks if the contract is deactivated but omits a check for the paused state.

```
22  /**
23  * @dev check if the contract is deativated or the address is frozen
24  * check revlevant for mint and burn operations
25  */
26  function _canMintBurnByModule(
27      address target
28  ) internal view virtual returns (bool) {
29      if(PauseModule.deactivated() || EnforcementModule.isFrozen(target)){
30          // can not mint or burn if the contract is deactivated
31          // cannot burn if target is frozen (used forcedTransfer instead if available)
32          // cannot mint if target is frozen
33          return false;
34      }
35      return true;
36  }
```

While tests confirm this is intended behavior, it contradicts the documented purpose of the pause feature and creates a false sense of security. An administrator pausing the contract during a critical incident would reasonably expect all token transfers, including mints and burns to be halted.

Allowing these operations to continue during a pause could lead to severe consequences. For example, if a contract is paused due to a compromised administrator key, that key could still be used to mint or burn tokens, exacerbating the situation. The distinction between `pause()` (stops user transfers) and `deactivate` (stops all transfers) is not clearly enforced, making the `pause()` function an incomplete safety measure.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:C/D:C/Y:N (2.5)

Recommendation

To align with security best practices, modify `_canMintBurnByModule()` to respect the paused state. Alternatively, if the two-tiered safety model (`pause` vs. `deactivate`) is a core requirement, this behavior must be explicitly and prominently documented to warn administrators and users that `pause` does not prevent privileged minting and burning.

Remediation Comment

FUTURE RELEASE: The **Taurus team** made a business decision to accept the risk of this finding and not alter the contracts, stating:

> If the administrator key is compromised, the attacker can unpause the contract since he has all the rights. Therefore, putting the contract in pause state does not protect against this type of attack. If the administrator key is compromised, there are no measures in the CMTAT to remedy this. CMTAT users are encouraged to take the necessary steps to protect access to this key.
>
> An alternative solution would be to provide an additional function pauseAllTransfers which would pause standard transfers, as well as all burn and mint operations. However, due to the architecture of current contracts, it is not possible to add this functionality without exceeding the maximum contract size on Ethereum. Consideration will be given to how this can be achieved in a future release.

8.2 INSUFFICIENT ALLOWANCE VALIDATION DURING FORCED TRANSFERS

// LOW

Description

The `_forcedTransfer()` function in the `ERC20EnforcementModuleInternal.sol` contract is a privileged administrative tool for executing critical transfers, such as moving funds from a frozen account. However, the function lacks a crucial validation check to ensure the transfer amount does not exceed the allowance granted by the `from` address to the `to` address. Instead of reverting on insufficient allowance as per the ERC20 standard, the function proceeds with the transfer. This breaks a fundamental security assumption of the ERC20 standard.

While external developer discussions provided acknowledge this behavior, they suggest avoiding the function for such scenarios, which relies on operational policy rather than secure code to prevent misuse. This design allows a mistaken or malicious administrator (`DEFAULT_ADMIN_ROLE`) to exploit the flawed logic.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:C/Y:N (2.0)

Recommendation

The `_forcedTransfer()` function must be modified to handle allowances in a safe and predictable manner. The logic should be updated to strictly enforce that the transfer amount cannot exceed the existing allowance, causing the transaction to revert if it does.

Remediation Comment

RISK ACCEPTED: The **Taurus team** made a business decision to accept the risk of this finding and not alter the contracts, stating:

> The goal of the forcedTransfer function is exactly to allow the issuer to transfer tokens without the approval of the token holder. Thus, there is no concept of allowance. The function is distinct from a burn to clearly show the difference between a token supply management operation and an operation that may result from a legal request from the judicial authorities.
>
> It should be noted that in terms of result, this function is no different from the burn function present in the CMTAT as well as the corresponding functions in known tokens such as USDC or USDT.
>
> Since CMTAT is not intended to represent tokens in a defi-friendly way, the administrator is considered trusted. Access to private keys must therefore also be protected accordingly.

8.3 FLOATING PRAGMA

// INFORMATIONAL

Description

The contracts in scope currently use different floating pragma versions `^0.8.0` , `^0.8.22` and `^0.8.28` which means that the code can be compiled by any compiler version that is greater than these versions, and less than `0.9.0` .

However, it is recommended that contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively.

Additionally, from Solidity versions `0.8.20` through `0.8.24` , the default target EVM version is set to `Shanghai` , which results in the generation of bytecode that includes `PUSH0` opcodes. Starting with version `0.8.25` , the default EVM version shifts to `Cancun` , introducing new opcodes for transient storage, `TSTORE` and `TLOAD` .

In this aspect, it is crucial to select the appropriate EVM version when it's intended to deploy the contracts on networks other than the Ethereum mainnet, which may not support these opcodes. Failure to do so could lead to unsuccessful contract deployments or transaction execution issues.

BVSS

AO:A/AC:L/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.8)

Recommendation

Lock the pragma version to the same version used during development and testing (for example: `pragma solidity 0.8.30;`), and make sure to specify the target EVM version when using newly released Solidity versions if deploying to chains that may not support newly introduced opcodes.

Additionally, it is crucial to stay informed about the opcode support of different chains to ensure smooth deployment and compatibility.

Remediation Comment

ACKNOWLEDGED: The **Taurus team** made a business decision to acknowledge this finding and not alter the contracts, stating:

> One potential use of CMTAT is to be used as a library, similar to OpenZeppelin library.
>
> In this sense, we use the same convention of OpenZeppelin which for the moment only imposes that the version is higher than 0.8.20: pragma solidity ^0.8.20;
>
> A fixed version is set in the config file (0.8.30). Users are free to use these or conduct their own research before switching to another.

8.4 MISLEADING RESTRICTION CODE RETURNED FOR DEACTIVATED CONTRACT

// INFORMATIONAL

Description

The `ValidationModuleERC1404` contract is designed to provide human-readable reasons for transfer restrictions, conforming to the ERC-1404 standard. The `_detectTransferRestriction()` function checks for various conditions like the contract being paused or an address being frozen.

```
130 function _detectTransferRestriction(
131     address from,
132     address to,
133     uint256 /* value */
134 ) internal virtual view  returns (uint8 code) {
135     if (paused()) {
136         return uint8(IERC1404Extend.REJECTED_CODE_BASE.TRANSFER_REJECTED_PAUSED);
137     } else if (isFrozen(from)) {
138         return uint8(IERC1404Extend.REJECTED_CODE_BASE.TRANSFER_REJECTED_FROM_FROZEN);
139     } else if (isFrozen(to)) {
140         return uint8(IERC1404Extend.REJECTED_CODE_BASE.TRANSFER_REJECTED_TO_FROZEN);
141     }
142     else {
143         return uint8(IERC1404Extend.REJECTED_CODE_BASE.TRANSFER_OK);
144     }
145 }
```

However, the function does not check if the contract has been `deactivated`. The `PauseModule` allows for two distinct states: a temporary `pause` and a permanent `deactivation`. Since `deactivateContract()` can only be called when the contract is already paused, the existing `paused()` check will prevent transfers. The issue is that it will return a `TRANSFER_REJECTED_PAUSED` code, which is inaccurate and misleading for a contract that has been permanently disabled.

Users and systems interacting with the token would be informed that the contract is temporarily paused, when in reality it has been irreversibly deactivated, creating a discrepancy between the contract's actual state and the reason provided for the transfer failure.

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.6)

Recommendation

Update the `_detectTransferRestriction()` function in `ValidationModuleERC1404` to check for the `deactivated` state before checking for the `paused` state.
This will ensure that the most accurate restriction code is returned.

Remediation Comment

SOLVED: The **Taurus team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/CMTA/CMTAT/commit/067244a0801554e2c2a7512573d005e450ffa765

8.5 COMMENTED FUNCTIONALITY

// INFORMATIONAL

Description

In the `__CMTAT_openzeppelin_init_unchained()` function of the `CMTATBaseCore` contract, there is commented out code that is not used. This code may introduce unnecessary confusion to the contract.

```
97 | // We don'use name and symbol set by the OpenZeppelin module
98 | //__ERC20_init_unchained(ERC20Attributes_.name, ERC20Attributes_.symbol);
```

While commenting out code can be useful for debugging or testing purposes, it can also lead to confusion and make the codebase harder to maintain.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Remove the commented-out lines of code to clean up the contract and improve readability.

Remediation Comment

SOLVED: The **Taurus team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/CMTA/CMTAT/commit/86dbd2d0151cfd9825f477bbcbc9848fd69c24c1

8.6 TYPOS

// INFORMATIONAL

Description

Throughout the codebase, there are several instances of typos in comments. While these typos do not affect the functionality of the code, they can make the codebase harder to read and understand. It is recommended to fix these typos to improve the readability of the codebase.

Instances of this issue include:

- The word `contract` is misspelled as `conract` in the `ERC20EnforcementModuleInternal` contract description.
- The word `explanation` is misspelled as `explaination` in the NatSpec for `messageForTransferRestriction` in `ValidationModuleERC1404`.
- The word `relevant` is misspelled as `revlevant` in a comment in `_canMintBurnByModule` in `ValidationModule`.
- The word `deactivated` is misspelled as `deativated` in a comment in `_canMintBurnByModule` in `ValidationModule`.
- The word `spent` is misspelled as `spended` in the NatSpec for `transferFrom` in `ERC20BaseModule`.
- The word `supplementary` is misspelled as `supplémentary` in a comment in `_burnOverride` in `ERC20BurnModuleInternal`.
- The word `functions` is misspelled as `funtions` in the library description for `EnforcementModuleLibrary`.
- The word `Solidity` is misspelled as `Soliditiy` in a comment in `_batchTransfer` in `ERC20MintModuleInternal`.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to fix all typos to improve the readability of the codebase.

Remediation Comment

SOLVED: The **Taurus team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/CMTA/CMTAT/commit/4d0a72f4646a10293cc11f2f1dbc666798acc054

8.7 PUBLIC FUNCTIONS NOT CALLED WITHIN CONTRACTS

// INFORMATIONAL

Description

Several state-changing functions throughout the codebase in scope are currently defined with the `public` visibility modifier, even though the functions are not called from within the contracts. For functions that are only ever called externally (i.e., not by other functions within the same contract), it is a gas-optimization best practice to use the `external` visibility modifier.

Instances of this issue include:

- In 0_CMTATBaseCore.sol:
  - `function burnAndMint(address from, address to, uint256 amountToBurn, uint256 amountToMint, bytes calldata data) public virtual`
  - `function forcedBurn(address account, uint256 value, bytes memory data) public virtual`
- In 0_CMTATBaseCommon.sol:
  - `function burnAndMint(address from, address to, uint256 amountToBurn, uint256 amountToMint, bytes calldata data) public virtual`
- In 3_CMTATBaseERC20CrossChain.sol:
  - `function crosschainMint(address to, uint256 value) public virtual`
  - `function crosschainBurn(address from, uint256 value) public virtual`
  - `function burnFrom(address account, uint256 value) public virtual`
  - `function burn(uint256 value) public virtual`
- In ERC20BaseModule.sol:
  - `function setName(string calldata name_) public virtual`
  - `function setSymbol(string calldata symbol_) public virtual`
- In PauseModule.sol:
  - `function pause() public virtual`
  - `function unpause() public virtual`
  - `function deactivateContract() public virtual`
- In EnforcementModule.sol:
  - `function setAddressFrozen(address account, bool freeze) public virtual`
  - `function setAddressFrozen(address account, bool freeze, bytes calldata data) public virtual`
  - `function batchSetAddressFrozen(address[] calldata accounts, bool[] calldata freezes) public virtual`
- In ERC20MintModule.sol:
  - `function mint(address account, uint256 value) public virtual`
  - `function batchMint(address[] calldata accounts, uint256[] calldata values) public virtual`
  - `function batchTransfer(address[] calldata tos, uint256[] calldata values) public`
- In ERC20BurnModule.sol:

- function burn(address account, uint256 value) public virtual
- function batchBurn(address[] calldata accounts, uint256[] calldata values, bytes memory data) public virtual
- function batchBurn(address[] calldata accounts, uint256[] calldata values) public virtual

- In ERC20EnforcementModule.sol:
  - function forcedTransfer(address from, address to, uint256 value, bytes calldata data) public virtual
  - function forcedTransfer(address from, address to, uint256 value) public virtual
  - function freezePartialTokens(address account, uint256 value) public virtual
  - function unfreezePartialTokens(address account, uint256 value) public virtual
  - function freezePartialTokens(address account, uint256 value, bytes calldata data) public virtual
  - function unfreezePartialTokens(address account, uint256 value, bytes calldata data) public virtual
- In ExtraInformationModule.sol:
  - function setTokenId(string calldata tokenId_) public virtual
  - function setTerms(IERC1643CMTAT.DocumentInfo calldata terms_) public virtual
  - function setInformation(string calldata information_) public virtual
- In SnapshotEngineModule.sol:
  - function setSnapshotEngine(ISnapshotEngine snapshotEngine_) public virtual
- In ValidationModuleRuleEngine.sol:
  - function setRuleEngine(IRuleEngine ruleEngine_) public virtual
- In AllowlistModule.sol:
  - function setAddressAllowlist(address account, bool status) public virtual
  - function setAddressAllowlist(address account, bool status, bytes calldata data) public virtual
  - function batchSetAddressAllowlist(address[] calldata accounts, bool[] calldata status) public virtual
  - function enableAllowlist(bool status) public virtual
- In DebtEngineModule.sol:
  - function setDebtEngine(IDebtEngine debtEngine_) public virtual
- In DebtModule.sol:
  - function setCreditEvents(CreditEvents calldata creditEvents_) public
  - function setDebt(ICMTATDebt.DebtInformation calldata debt_) public virtual
  - function setDebtInstrument(ICMTATDebt.DebtInstrument calldata debtInstrument_) public virtual
- In ERC7551Module.sol:
  - function setMetaData(string calldata metadata_) public virtual
  - function setTerms(bytes32 hash, string calldata uri) public virtual

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Modify the `public` functions not used within the contracts with the `external` visibility modifier.

Remediation Comment

ACKNOWLEDGED: The **Taurus team** made a business decision to acknowledge this finding and not alter the contracts, stating:

> According to RareSkills optimization book, section Outdated tricks, suing the keyword external instead of public is no longer an optimization in terms of gas.
>
> Via the `public` keyword, this allows users of the library to override the function in their contra to change its behavior when needed.

8.8 MISLEADING COMMENT REGARDING FROZEN BALANCE CALCULATION

// INFORMATIONAL

Description

In the `_unfreezeTokens` function within the `ERC20EnforcementModuleInternal.sol` contract, a comment incorrectly describes the relationship between frozen tokens and the total balance. The comment states: `// Frozen token can not be < balance`.

This comment is misleading because the number of frozen tokens for an account can, and typically will, be less than the account's total balance. The actual invariant that is maintained is that the frozen token amount cannot be ***greater*** than the total balance, which is enforced in the `_freezePartialTokens` function.

While this does not introduce a direct vulnerability, it can cause confusion for developers and auditors, potentially leading to incorrect assumptions about the contract's logic.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

It is recommended to correct the comment to accurately reflect the code's logic. This will improve the clarity and maintainability of the code.

Remediation Comment

SOLVED: The **Taurus team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/CMTA/CMTAT/commit/6b1c32e7dd1b7435ea7b829d1a684f22b80ad113

8.9 INCONSISTENT METHOD OF CALLING INHERITED FUNCTIONS

// INFORMATIONAL

Description

The `ERC20EnforcementModuleInternal` contract calls the `balanceOf` function from its parent `ERC20Upgradeable` contract using two different syntaxes. The `_freezePartialTokens` function uses a direct call, `balanceOf(account)`, while the `_getActiveBalanceOf` function uses an explicit call to the parent contract, `ERC20Upgradeable.balanceOf(account)`.

While both calls are functionally equivalent in the current implementation, this inconsistency can reduce code clarity and introduce potential maintenance challenges. Future developers might be confused about whether there is a deliberate reason for the different call styles, which could lead to errors if the contract is extended or modified.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

For improved code consistency and readability, adopt a single, uniform method for calling functions from parent contracts throughout the codebase.

Remediation Comment

SOLVED: The **Taurus team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/CMTA/CMTAT/commit/52e1106bc807057b5b367476661c8090e2b0aae2

8.10 LACK OF NAMED MAPPINGS

// INFORMATIONAL

Description

The project contains several unnamed mappings despite using a Solidity version that supports named mappings.

Named mappings improve code readability and self-documentation by explicitly stating their purpose.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Consider refactoring the mappings to use named arguments, which will enhance code readability and make the purpose of each mapping more explicit. For example:

```
mapping(address myAddress => bool myBool) public myMapping;
```

Remediation Comment

SOLVED: The **Taurus team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/CMTA/CMTAT/commit/f6021de5102172584816d8015e9d4c32f3e4ef68

8.11 UNUSED FILE

// INFORMATIONAL

Description

The file `0_CMTATBaseGeneric.sol` exists within the project's codebase. However, it is not imported, inherited, or otherwise utilized by any other contract in the system.

The presence of such "dead code" can increase the complexity of the project, leading to potential confusion for future developers and auditors who may spend time analyzing code that has no impact on the protocol's logic. Maintaining a clean and concise codebase is essential for security and long-term maintainability.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Remove the `0_CMTATBaseGeneric.sol` file from the project to improve code hygiene.

Remediation Comment

ACKNOWLEDGED: The **Taurus team** made a business decision to acknowledge this finding and not alter the contracts, stating:

> The file 0_CMTATBaseGeneric.sol exists to allow CMTAT users to use CMTAT code with nonstandard ERC-20 token, for example ERC-721 token or Zama FHE ERC-20 encrypted tokens.
>
> While CMTAT does not provide a deployment version using this, functionnalities are tested through an ERC-721 mock contract ERC721Upgradeable.

9. AUTOMATED TESTING

Description

`Halborn` used automated testing techniques to increase coverage of specific areas within the smart contracts under review. Among the tools used was `Slither`, a Solidity static analysis framework. After `Halborn` successfully verified the smart contracts in the repository and was able to compile them correctly into their ABI and binary formats, `Slither` was executed against the contracts. This tool performs static verification of mathematical relationships between Solidity variables to identify invalid or inconsistent usage of the contracts' APIs throughout the entire codebase.

The security team reviewed all findings reported by the `Slither` software; however, findings related to external dependencies have been excluded from the results below to maintain report clarity.

Output

The findings generated by `Slither` were evaluated. Most of them were excluded from this report as they were determined to be false positives.

```
INFO:Detectors:
Reentrancy in CMTATBaseCommon._update(address,address,uint256) (contracts/modules/0_CMTATBaseCommon.sol#137-145):
        External calls:
        - snapshotEngineLocal.operateOnTransfer(from,to,balanceOf(from),balanceOf(to),totalSupply()) (contracts/modules/0_CMTATBaseCommon.sol#142)
        Event emitted after the call(s):
        - Transfer(from,to,value) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#233)
                - ERC20Upgradeable._update(from,to,amount) (contracts/modules/0_CMTATBaseCommon.sol#144)
Reentrancy in CMTATBaseCommon.burnAndMint(address,address,uint256,uint256,bytes) (contracts/modules/0_CMTATBaseCommon.sol#117-120):
        External calls:
        - ERC20BurnModule.burn(from,amountToBurn,data) (contracts/modules/0_CMTATBaseCommon.sol#118)
                - snapshotEngineLocal.operateOnTransfer(from,to,balanceOf(from),balanceOf(to),totalSupply()) (contracts/modules/0_CMTATBaseCommon.sol#142)
        - ERC20MintModule.mint(to,amountToMint,data) (contracts/modules/0_CMTATBaseCommon.sol#119)
                - snapshotEngineLocal.operateOnTransfer(from,to,balanceOf(from),balanceOf(to),totalSupply()) (contracts/modules/0_CMTATBaseCommon.sol#142)
        Event emitted after the call(s):
        - Mint(_msgSender(),account,value,data) (contracts/modules/wrapper/core/ERC20MintModule.sol#91)
                - ERC20MintModule.mint(to,amountToMint,data) (contracts/modules/0_CMTATBaseCommon.sol#119)
        - Transfer(from,to,value) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#233)
                - ERC20MintModule.mint(to,amountToMint,data) (contracts/modules/0_CMTATBaseCommon.sol#119)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
DebtModule._getDebtModuleStorage() (contracts/modules/wrapper/options/DebtModule.sol#92-96) uses assembly
        - INLINE ASM (contracts/modules/wrapper/options/DebtModule.sol#93-95)
ERC7551Module._getERC7551ModuleStorage() (contracts/modules/wrapper/options/ERC7551Module.sol#77-81) uses assembly
        - INLINE ASM (contracts/modules/wrapper/options/ERC7551Module.sol#78-80)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
CMTATBaseAllowlist._msgData() (contracts/modules/1_CMTATBaseAllowlist.sol#196-198) is never used and should be removed
CMTATBaseERC1363._msgData() (contracts/modules/5_CMTATBaseERC1363.sol#136-143) is never used and should be removed
CMTATBaseERC2771._msgData() (contracts/modules/4_CMTATBaseERC2771.sol#34-36) is never used and should be removed
CMTATBaseERC7551._msgData() (contracts/modules/5_CMTATBaseERC7551.sol#60-67) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.