Halborn Security Assessment - Review

Code	Name	Severity	Status	Note
HAL-01	MINTING AND BURNING OPERATIONS BYPASS THE PAUSE	Low	Acknowledge	Design choice but possible improvement in a future release
HAL-02	INSUFFICIENT ALLOWANCE VALIDATION DURING FORCED TRANSFERS	Low	Acknowledge	Design choice
HAL-03	Floating Pragma	Informational	Acknowledge	Design choice
HAL-04	MISLEADING RESTRICTION CODE RETURNED FOR DEACTIVATED CONTRACT	Informational	Fixed as recommended	Commit
HAL-05	COMMENTED FUNCTIONALITY	Informational	Fixed as recommended	Commit
HAL-06	TYPOS	Informational	Fixed as recommended	Commit
HAL-07	PUBLIC FUNCTIONS NOT CALLED WITHIN CONTRACTS	Informational	Acknowledge	Design choice
HAL-08	MISLEADING COMMENT REGARDING FROZEN BALANCE CALCULATION	Informational	Fixed as recommended	Commit

Code	Name	Severity	Status	Note
HAL-09	INCONSISTENT METHOD OF CALLING INHERITED FUNCTIONS	Informational	Fixed as recommended (always use explicit call to the parent contract)	Commit
HAL-10	LACK OF NAMED MAPPINGS	Informational	Fixed as recommended (use named argument)	commit
HAL-11	UNUSED FILE	Informational	Acknowledge	Design choice (File is useful)

Acknowledge issues

7.1 (HAL-01) - MINTING AND BURNING OPERATIONS BYPASS THE PAUSE

MECHANISM

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.

Compromised administrator key

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.

> **Administrator key:**
>
> 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.
>
> **Burner/minter key:**
>
> If the burner and minter keys are compromised, they are two steps to perform:
>
> - Revoke their roles through the admin key
> - Pause all transfer
>
> By separating, burn/mint and standard transfer, the admin can re-adjust the supply while the standard transfers are paused.

> The admin can also perform a `forcedTransfer` if the ERC20EnforcementModule is included.
>
> We highlight also that specific function for cross-chain bridge (`3_CMTATBaseERC20CrossChain.sol`) will revert if contract is paused because they are not intended to be used by the issuer to manage the supply.
>
> **ERC-3643**
> The ERC-3643 implementation by Tokeny implements also the same behavior:
> github.com/ERC-3643/ERC-3643/blob/main/contracts/token/Token.sol#L454

> **Future possible improvement:**
>
> 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.

Distinction with deactivate

The distinction between pause() (stops user transfers) and deactivate (stops all transfers) is not clearly enforced, making the pause() function an incomplete safety measure.

> There are clearly enforced in the smart contract. If the contract is deactivated, minting and burning will no longer worked like for regular transfer, which makes sense because the contract has been marked as deactivated.

7.2 (HAL-02) 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.

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.

> 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.

7.3 (HAL-03) FLOATING PRAGMA

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

> 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.

7.7 (HAL-07) 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.

> 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.

7.11 (HAL-11) UNUSED FILE

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 file 0_CMTATBaseGeneric.sol exists to allow CMTAT users to use CMTAT code with non-standard 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.