Coinbase: Custom Stable-coin

Security Review

Cantina Managed review by:
Sujith Somraaj, Lead Security Researcher
Om Parikh, Security Researcher

April 7, 2026

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**
- 2.1 Scope . . . 3

3 Findings **4**
- 3.1 Low Risk . . . 4
  - 3.1.1 ERC-3009 typehash constants are private, diverging from specification's public visibility requirement . . . 4
  - 3.1.2 Function `currentLimit()` reverts with unhandled panic for unconfigured accounts 4
  - 3.1.3 Function `_configureRateLimit()` resets remaining capacity to full limit on reconfiguration . . . 5
  - 3.1.4 `PAUSE_ROLE` controls both pause and unpause, preventing privilege separation . . . 6
  - 3.1.5 Functions `approve()` and `permit()` bypass blocklist enforcement . . . 6
  - 3.1.6 Unoverridden `renounceOwnership()` bypasses two-step ownership protection in `TwoStepUpgradeableBeacon` . . . 6
  - 3.1.7 Separate `BLOCKLIST_ROLE` into `BLOCKLIST_ROLE` and `UNBLOCKLIST_ROLE` . . . 7
- 3.2 Informational . . . 8
  - 3.2.1 Unused import of `IBeacon` in `MutableBeaconProxy.sol` . . . 8
  - 3.2.2 `LimitReplenished` event is declared but never emitted . . . 8
  - 3.2.3 Stale comment references `uint192` instead of `uint216` . . . 8
  - 3.2.4 Custom functions use `msg.sender` instead of `_msgSender()` . . . 9
  - 3.2.5 Add more parameters to `StablecoinDeployed` event . . . 9
  - 3.2.6 Documented immutability guarantee of `BEACON` does not hold . . . 10
  - 3.2.7 Function `supportsInterface()` does not declare all supported interfaces . . . 10

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Base is a secure, low-cost, builder-friendly Ethereum L2 built to bring the next billion users onchain.

From Mar 30th to Apr 2nd the Cantina team conducted a review of custom-stablecoin on commit hash 8d1bee02. The team identified a total of **14** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	7	6	1
Gas Optimizations	0	0	0
Informational	7	7	0
Total	**14**	**13**	**1**

2.1 Scope

The security review had the following components in scope for custom-stablecoin on commit hash 8d1bee02:

```
evm/src
├── MutableBeaconProxy.sol
├── Stablecoin.sol
├── StablecoinFactory.sol
└── lib
    ├── Blocklist.sol
    ├── ERC3009Upgradeable.sol
    ├── RateLimit.sol
    └── TokenMetadata.sol
```

3 Findings

3.1 Low Risk

3.1.1 ERC-3009 typehash constants are private, diverging from specification's public visibility requirement

Severity: Low Risk

Context: ERC3009Upgradeable.sol#L35-L45

Description: EIP-3009 defines three typehash constants as part of the contract's public interface:

```
bytes32 public constant TRANSFER_WITH_AUTHORIZATION_TYPEHASH =
  0x7c7c6cdb67a18743f49ec6fa9b35f50d52ed05cbed4cc592e13b44501c1a2267;
bytes32 public constant RECEIVE_WITH_AUTHORIZATION_TYPEHASH =
  0xd099cc98ef71107a616c4f0f941f04c322d8e254fe26b3c6668db87aae413de8;
bytes32 public constant CANCEL_AUTHORIZATION_TYPEHASH =
↪  0x158b0a9edf7a828aad02f63cd515c68ef2f50ba807396f6d12842833a1597429;
```

However, the implementation in `ERC3009Upgradeable.sol` declares all three as private. This results in a slight non-compliance with the standard. Hence any tooling or integration that expects to query the typehashes on-chain (as the EIP mandates) will fail.

Recommendation: Change the visibility from private to public on all three constants:

```
- bytes32 private constant TRANSFER_WITH_AUTHORIZATION_TYPEHASH =
+ bytes32 public constant TRANSFER_WITH_AUTHORIZATION_TYPEHASH =
        0x7c7c6cdb67a18743f49ec6fa9b35f50d52ed05cbed4cc592e13b44501c1a2267;

- bytes32 private constant RECEIVE_WITH_AUTHORIZATION_TYPEHASH =
+ bytes32 public constant RECEIVE_WITH_AUTHORIZATION_TYPEHASH =
        0xd099cc98ef71107a616c4f0f941f04c322d8e254fe26b3c6668db87aae413de8;

- bytes32 private constant CANCEL_AUTHORIZATION_TYPEHASH =
+ bytes32 public constant CANCEL_AUTHORIZATION_TYPEHASH =
        0x158b0a9edf7a828aad02f63cd515c68ef2f50ba807396f6d12842833a1597429;
```

Coinbase: Fixed in commit 40b88d31.

Cantina Managed: Fix verified.

3.1.2 Function `currentLimit()` reverts with unhandled panic for unconfigured accounts

Severity: Low Risk

Context: RateLimit.sol#L108

Description: Function `currentLimit()` is a public view function intended to return the available capacity for a given (key, account) pair. When called for an account that has no rate-limit configuration (i.e., all struct fields are zero-initialized), it reverts with a raw `Panic(0x12)` (division by zero) instead of returning 0 or reverting with a descriptive error.

Proof of Concept:

```
function test_audit() external {
      // Query an account with no rate-limit configuration
      stablecoin.currentLimit(stablecoin.MINT_RATE_LIMIT_KEY(), address(101));
      // Result: panic: division or modulo by zero (0x12)
  }
```

Test Output:

```
[FAIL: panic: division or modulo by zero (0x12)] test_audit() (gas: 22799).
```

Recommendation:

1. Add an early return for unconfigured accounts, consistent with how `_consumeLimit` guards the same path:

```
  function currentLimit(bytes32 key, address account) public view virtual returns
  ↪  (uint256) {
     RateLimitConfig storage config = _getRateLimitLayout().limits[key][account];
+    if (config.interval == 0) return 0;

     uint256 elapsed = block.timestamp - config.lastConsumed;
     uint256 replenishmentAmount = Math.mulDiv(elapsed, config.limit, config.interval);
     return Math.min(config.remaining + replenishmentAmount, config.limit);
  }
```

This returns 0 for unconfigured accounts (which is semantically correct as they have no capacity) without breaking existing callers.

2. Or consider returning early with a "`RateLimitNotConfigured`" error for unconfigured accounts.

Coinbase: Fixed in commit 40b88d31.

Cantina Managed: Fix verified.

3.1.3 Function `_configureRateLimit()` resets remaining capacity to full limit on reconfiguration

Severity: Low Risk

Context: RateLimit.sol#L153-L159

Description: Function `_configureRateLimit()` overwrites the entire `RateLimitConfig` struct, resetting `remaining` to the full limit regardless of how much capacity the account has already consumed.

This function is called by both `grantMinterRoleWithLimit()` (admin-only) and `configureMinter()` (rate-limit-role). Any reconfiguration like even a minor parameter adjustment like changing `interval` fully replenishes the account's capacity.

Proof of Concept:

```
function test_audit_reconfigure_resets_remaining() external {
    // Minter starts with 1_000_000e6 capacity
    uint256 limitBefore = stablecoin.currentMintLimit(minter);
    assertEq(limitBefore, 1_000_000e6);

    // Minter consumes 900_000e6 (90% of capacity)
    vm.prank(minter);
    stablecoin.mint(user1, 900_000e6);

    uint256 limitAfterMint = stablecoin.currentMintLimit(minter);
    assertEq(limitAfterMint, 100_000e6);

    // Admin reconfigures the minter (e.g. just to change interval)
    // but remaining resets to the full limit
    vm.prank(admin);
    stablecoin.grantMinterRoleWithLimit(minter, 1_000_000e6, 2 days);

    uint256 limitAfterReconfig = stablecoin.currentMintLimit(minter);
    // Remaining is fully refilled despite 90% being consumed moments ago
    assertEq(limitAfterReconfig, 1_000_000e6);

    // Minter can now mint the full limit again and rate limit was bypassed
    vm.prank(minter);
    stablecoin.mint(user1, 1_000_000e6);
    assertEq(stablecoin.balanceOf(user1), 1_900_000e6);
}
```

Test Output:

```
[PASS] test_audit_reconfigure_resets_remaining() (gas: 132304)
```

Recommendation: Preserve the current consumed state when reconfiguring. Compute the current available capacity and carry it forward proportionally, or at minimum cap `remaining` to the lesser of the new limit and the current available capacity.

Coinbase: Added documentation explaining this behaviour.

Cantina Managed: Acknowledged.

3.1.4 `PAUSE_ROLE` controls both pause and unpause, preventing privilege separation

Severity: Low Risk

Context: Stablecoin.sol#L237-L245

Description: Both `pause()` and `unpause()` are gated by the same `PAUSE_ROLE`. In practice, pause and unpause have very different risk profiles:

- Pause is an emergency action, typically triggered by automated off-chain watchers or hot wallets that monitor for exploits, anomalous transfers, or oracle failures. Speed is critical and the key must be readily available.
- Unpause is a recovery action that should require stronger assurance that the threat has been resolved. It benefits from a higher-privilege signer (multisig, timelock, or cold wallet).

Recommendation: Introduce a separate `UNPAUSE_ROLE`:

```
+ bytes32 public constant UNPAUSE_ROLE = keccak256("UNPAUSE_ROLE");

- function unpause() external onlyRole(PAUSE_ROLE) {
+ function unpause() external onlyRole(UNPAUSE_ROLE) {
    _unpause();
```

This allows operators to grant `PAUSE_ROLE` to a hot wallet or monitoring bot for rapid response, while restricting `UNPAUSE_ROLE` to a multisig or governance contract that confirms the threat is resolved before resuming operations.

Both roles can still be assigned to the same address if separation isn't desired.

Coinbase: Fixed in commit 40b88d31.

Cantina Managed: Fix verified.

3.1.5 Functions `approve()` and `permit()` bypass blocklist enforcement

Severity: Low Risk

Context: Stablecoin.sol#L39

Description: The `Stablecoin` contract enforces its blocklist exclusively in the `_update()` hook, which gates all token movement (transfers, mints, burns). However, approve and permit modify allowances through `_approve()`, which does not pass through `_update()`.

Neither path checks whether `msg.sender`, `owner`, or `spender` is blocklisted. A blocklisted address can freely call `approve()`, and anyone can submit a pre-signed `permit()` for a blocklisted signer and the ERC-2612 nonce is consumed and the allowance is set.

Recommendation: Consider overriding the `_approve()` internal function to include blocklist checks.

Coinbase: Fixed in commit 40b88d31.

Cantina Managed: Fix verified. Commit 40b88d31 addresses this by overriding `_approve()` to enforce the blocklist check for both `approve()` and `permit()`, mirroring the check already present in `_update()`. This is consistent with the approach used by Circle's `FiatTokenV2` implementation.

3.1.6 Unoverridden `renounceOwnership()` bypasses two-step ownership protection in `TwoStepUpgradeableBeacon`

Severity: Low Risk

Context: TwoStepUpgradeableBeacon.sol#L17

Description: `TwoStepUpgradeableBeacon` inherits from both `UpgradeableBeacon` and `Ownable2Step` to enforce a two-step (propose + accept) ownership transfer pattern. The contract explicitly overrides `transferOwnership()` and `_transferOwnership()` to route through Ownable2Step's two-step logic, and the inline documentation states:

> "Ownership does not change until the pending owner explicitly calls acceptOwnership, preventing an erroneous address from permanently locking beacon upgrades".

However, `Ownable2Step` does not override `renounceOwnership()`. The single-step `Ownable.renounceOwnership()` remains publicly callable by the current owner and immediately sets the owner to `address(0)` with no pending step, no confirmation, and no recovery path.

This directly contradicts the contract's stated security invariant.

Recommendation: Override `renounceOwnership()` in `TwoStepUpgradeableBeacon` to revert unconditionally, making it impossible to accidentally or intentionally orphan the beacon:

```
/// @notice Renouncing ownership is disabled to prevent permanently locking beacon
↪  upgrades.
function renounceOwnership() public pure override {
  revert("TwoStepUpgradeableBeacon: renounce disabled");
}
```

If there is a legitimate need to allow renunciation in the future, it should be implemented with the same two-step confirmation pattern used for transfers (e.g., `proposeRenounce()` + `confirmRenounce()` with a timelock), ensuring consistency with the contract's design intent.

Coinbase: Fixed in commit 40b88d31.

Cantina Managed: Fix verified.

3.1.7 Separate `BLOCKLIST_ROLE` into `BLOCKLIST_ROLE` and `UNBLOCKLIST_ROLE`

Severity: Low Risk

Context: Stablecoin.sol#L226-L228

Description: Function `updateBlocklistStatus()` handles both blocklisting and un-blocklisting addresses and is gated by a single `BLOCKLIST_ROLE`. In practice, these two operations have very different risk profiles:

- Blocklisting is a protective action, typically triggered by compliance systems, automated monitoring bots, or hot wallets. that detect sanctioned addresses, stolen funds, or illicit activity. Speed is critical and the key must be readily available to freeze assets before they can be moved.
- Un-blocklisting is a restorative action that re-enables an address to transfer tokens. It should require stronger assurance that the compliance concern has been resolved (e.g., false positive confirmed, legal review completed, law enforcement clearance). Granting the same hot wallet or bot that blocklists the ability to also un-blocklist introduces risk that a compromised bot key could silently un-blocklist a sanctioned address.

Recommendation: Introduce a separate `UNBLOCKLIST_ROLE` and split the single updateBlocklistStatus function into two distinct functions:

```
+ bytes32 public constant UNBLOCKLIST_ROLE = keccak256("UNBLOCKLIST_ROLE");

- function updateBlocklistStatus(address account, bool blocklisted) external
↪  onlyRole(BLOCKLIST_ROLE) {
-     _updateBlocklistStatus({account: account, blocklisted: blocklisted});
- }

+ function blocklist(address account) external onlyRole(BLOCKLIST_ROLE) {
+     _updateBlocklistStatus({account: account, blocklisted: true});
+ }
+
```

```
+ function unblocklist(address account) external onlyRole(UNBLOCKLIST_ROLE) {
+     _updateBlocklistStatus({account: account, blocklisted: false});
+ }
```

This allows operators to:

- Grant `BLOCKLIST_ROLE` to a hot wallet or compliance bot for rapid response to illicit activity.
- Restrict `UNBLOCKLIST_ROLE` to a multisig, compliance committee, or governance contract that confirms the issue is resolved before re-enabling the address.

Both roles can still be assigned to the same address if separation isn't desired and the change is purely additive.

Coinbase: Fixed in commit 40b88d31.

Cantina Managed: Fix verified.

3.2 Informational

3.2.1 Unused import of `IBeacon` in `MutableBeaconProxy.sol`

Severity: Informational

Context: MutableBeaconProxy.sol#L6

Description: The `MutableBeaconProxy` contract imports `IBeacon` from `@openzeppelin/contracts/proxy/beacon/IBeacon.sol`, but the identifier is never used within the contract. The contract only uses `BeaconProxy` and `ERC1967Utils`.

Recommendation: Remove the unused import statement to improve code clarity and eliminate dead dependencies:

```
- import {IBeacon} from "@openzeppelin/contracts/proxy/beacon/IBeacon.sol";
```

Coinbase: Fixed in commit 40b88d31.

Cantina Managed: Fix verified.

3.2.2 `LimitReplenished` event is declared but never emitted

Severity: Informational

Context: RateLimit.sol#L65-L71

Description: `RateLimit.sol` declares the `LimitReplenished` event but never emits it anywhere in the contract. Replenishment occurs implicitly within `_consumeLimit()`, where `currentLimit()` computes the replenished capacity and the result is written directly to `config.remaining`. However, only `LimitConsumed` is emitted and the replenishment amount is silently folded into the updated remaining value with no corresponding event.

Recommendation: Either emit the event in `_consumeLimit()` when replenishment occurs, or remove the dead declaration.

Coinbase: Fixed in commit 40b88d31.

Cantina Managed: Fix verified. The unused `LimitReplenished` event declaration was removed from `RateLimit.sol`.

3.2.3 Stale comment references `uint192` instead of `uint216`

Severity: Informational

Context: RateLimit.sol#L140

Description: The safety comment at `RateLimit.sol` references the wrong bit-width:

```
// Safe: amount <= currentLimit_ is enforced above, and currentLimit_ fits in uint192.
config.remaining = uint216(currentLimit_ - amount);
```

The actual type of remaining and the cast is `uint216`, not `uint192`. The safety reasoning is correct and `currentLimit_` is capped at `config.limit` which is `uint216` but the comment is stale.

Recommendation:

```
- // Safe: amount <= currentLimit_ is enforced above, and currentLimit_ fits in uint192.
+ // Safe: amount <= currentLimit_ is enforced above, and currentLimit_ fits in uint216.
```

Coinbase: Fixed in commit 40b88d31.

Cantina Managed: Fix verified.

3.2.4 Custom functions use `msg.sender` instead of `_msgSender()`

Severity: Informational

Context: Stablecoin.sol#L131, Stablecoin.sol#L144, Stablecoin.sol#L154, Stablecoin.sol#L167, Stablecoin.sol#L177, Stablecoin.sol#L188, Stablecoin.sol#L306

Description: OpenZeppelin's ERC-20 functions resolve the caller via `_msgSender()`, which supports overriding for meta-transaction patterns like ERC-2771. But every custom function in `Stablecoin.sol` uses raw `msg.sender` instead.

Recommendation: Replace `msg.sender` with `_msgSender()` in all custom functions for consistency with the inherited OZ pattern.

Coinbase: Fixed in a766dc15.

Cantina Managed: Verified fix. The team removed ERC 2771 support entirely by overriding `_msgSender()` function.

3.2.5 Add more parameters to `StablecoinDeployed` event

Severity: Informational

Context: StablecoinFactory.sol#L37

Description: The `StablecoinDeployed` event in `StablecoinFactory.sol` currently only emits the deployed proxy address. However, the deploy function receives five additional parameters including `name`, `symbol`, `decimals`, `stablecoinAdmin`, and `salt` none of which are included in the emitted event.

This means off-chain consumers (indexers, subgraphs, dashboards, monitoring tools) cannot determine the token's configuration from the event alone and must make additional RPC calls to the deployed contract to retrieve this information.

Since all of these values are already available in scope at the point of emission, there is no additional storage cost, only a marginal increase in gas for the extra log topics and data.

Recommendation: Update the event signature and emission to include all deploy parameters:

```
event StablecoinDeployed(
  address indexed stablecoin,
  string name,
  string symbol,
  uint8 decimals,
  address indexed stablecoinAdmin,
  bytes32 indexed salt
);

emit StablecoinDeployed(stablecoin, name, symbol, decimals, stablecoinAdmin, salt);
```

Coinbase: Fixed in commit 40b88d31.

Cantina Managed: Fix verified.

3.2.6 Documented immutability guarantee of `BEACON` does not hold

Severity: Informational

Context: StablecoinFactory.sol#L17

Description: The NatSpec on `StablecoinFactory` states:

```
/// @dev The beacon is fixed at construction time as an immutable and cannot be changed.
```

However, `StablecoinFactory` inherits `UUPSUpgradeable`, and Solidity immutable variables are embedded in the implementation contract's bytecode, not in proxy storage.

When `DEFAULT_ADMIN_ROLE` performs a UUPS upgrade, the proxy delegates to a new implementation whose constructor may have been called with a different `BEACON` address. After the upgrade, `factory.BEACON()` returns the new implementation's value, effectively changing what the documentation claims "cannot be changed". This creates a discrepancy between the documented security property and the actual behavior of the system.

Recommendation: Update the NatSpec to accurately reflect that BEACON is immutable per-implementation but can change across UUPS upgrades:

```
- /// @dev The beacon is fixed at construction time as an immutable and cannot be
 ↪  changed.
+ /// @dev The beacon is set at construction time as an immutable. Note that upgrading
 ↪  the
+ /// factory implementation via UUPS will adopt the new implementation's BEACON value.
```

Coinbase: Fixed in commit 40b88d31.

Cantina Managed: Fix verified.

3.2.7 Function `supportsInterface()` does not declare all supported interfaces

Severity: Informational

Context: Stablecoin.sol#L45

Description: `Stablecoin` inherits `AccessControlDefaultAdminRulesUpgradeable`, which implements ERC-165's `supportsInterface()` function. However, the contract does not override `supportsInterface` to declare support for the token-related interfaces it actually implements: ERC-20, ERC-2612, and ERC-3009.

Any on-chain or off-chain system that calls `supportsInterface` to detect token capabilities such as wallets, aggregators, bridge contracts, or protocol routers will receive false for all three interfaces, despite the contract fully implementing them. Only AccessControl and ERC-165 itself will return true.

Recommendation: Override supportsInterface in Stablecoin to advertise the implemented token interfaces:

```
function supportsInterface(bytes4 interfaceId)
      public
      view
      override
      returns (bool)
  {
      return interfaceId == type(IERC20).interfaceId
          || interfaceId == type(IERC20Permit).interfaceId
          || interfaceId == type(IERC3009).interfaceId
          || super.supportsInterface(interfaceId);
  }
```

Coinbase: Added the interfaces support checks for OZ contracts in 40b88d3. We don't have interface for ERC3009 so skipped that.

Cantina Managed: Verified fix.