Coinbase: SecurityToken

Security Review

Cantina Managed review by:
Sujith Somraaj, Lead Security Researcher
0xHuy0512, Security Researcher

June 23, 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 `SecurityToken` omits the standard `IB20` `pausedFeatures()` selector . . . 4
3.1.2 Allowlist transfer policies can disable minting and burning . . . 4
3.2 Informational . . . 5
3.2.1 `burnBlocked` natspec claims `to` and `origin` checks that do not exist . . . 5
3.2.2 Init-time `ALWAYS_ALLOW_ID` guard does not extend to per-deploy `extraData` overrides . . . 5
3.2.3 `setTransferPolicyId` accepts `ALWAYS_ALLOW_ID` (0) that initialize rejects (undocumented asymmetry) . . . 6
3.2.4 `redeemWithMemo` natspec misattributes the Memo event to the redeemer . . . 6
3.2.5 Single `UPGRADER_ROLE` controls both the shared token beacon and the factory's UUPS upgrade . . . 6
3.2.6 `PolicyRegistry._create` silently wraps the policy counter on overflow, diverging from the `IPolicyRegistry` specification . . . 7
3.2.7 Redundant burn pause checks . . . 8
3.2.8 Missing memo variants for batch minting and blocked burning . . . 8
3.2.9 Privileged roles can configure state for unregistered tokens . . . 8
3.2.10 `SecurityToken` omits the IB20-mandated supply cap . . . 9
3.2.11 `SecurityToken.announce` bubbles inner revert reasons, diverging from the `IB20Asset` `InternalCallFailed` contract . . . 9
3.2.12 Paused token irrecoverably burns users' funds on HyperCore to HyperEVM bridge-in 9

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 Jun 8th to Jun 12th the Cantina team conducted a review of tokenize on commit hash 9b6d2410. 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	2	1	1
Gas Optimizations	0	0	0
Informational	12	10	2
Total	**14**	**11**	**3**

2.1 Scope

The security review had the following components in scope for tokenize on commit hash 9b6d2410:

```
src
├── PolicyRegistry.sol
├── SecurityToken.sol
├── SecurityTokenFactory.sol
├── TokenSupplyManager.sol
├── base
│   └── B20AssetFactory.sol
└── hyperevm
    └── SecurityToken.sol
```

3 Findings

3.1 Low Risk

3.1.1 `SecurityToken` omits the standard `IB20` `pausedFeatures()` selector

Severity: Low Risk

Context: SecurityToken.sol#L80

Description: `IB20` declares `pausedFeatures() external view returns (PausableFeature[] memory)`, but `SecurityToken` implements only `pause`, `unpause`, and `isPaused` and there is no `pausedFeatures()`. A caller casting the token to `IB20Asset` and invoking `pausedFeatures()` hits no matching function and reverts.

Unlike the policy-surface omissions (`policyId` / `updatePolicy`), which the contract explicitly documents as intentional divergences, this omission is undocumented and reads as an oversight.

Recommendation: Implement `pausedFeatures()` by expanding the stored `pausedMask` bits into a `PausableFeature[]`. If the omission is instead deliberate, document it alongside the existing policy-surface divergence note rather than leaving it silent.

Coinbase: Fixed in commit 69ed8f18.

Cantina Managed: Fix verified.

3.1.2 Allowlist transfer policies can disable minting and burning

Severity: Low Risk

Context: PolicyRegistry.sol#L274, SecurityToken.sol#L661-L667, SecurityToken.sol#L683-L694, SecurityToken.sol#L748-L761

Description: `PolicyRegistry` contract rejects `address(0)` in `_batchSetMembers()`, so zero can never be added as a policy member. `SecurityToken` contract runs the configured `transferPolicyId` policy inside `SecurityToken._update()` against `from`, `to`, and `msg.sender`. This includes minting, where `from == address(0)`, and burning, where `to == address(0)`.

`transferPolicyId` is only documented as conventionally a `BLOCKLIST`; it is not required to be one. `setTransferPolicyId()` only checks that the new policy exists and does not enforce that it is a `BLOCKLIST`. As a result, an admin can configure an `ALLOWLIST` as `transferPolicyId` after deployment.

With an `ALLOWLIST` transfer policy, `address(0)` is not authorized because it is not a member and cannot be added. This makes minting and burning revert whenever `SecurityToken._update()` checks the zero address. The highest impact is that a valid policy update can unintentionally disable token issuance and redemption flows. Note that the policy can be changed post-deploy via `setTransferPolicyId()`.

Recommendation: Choose one invariant and enforce it consistently:

1. Require `transferPolicyId` to be a `BLOCKLIST` in both documentation and on-chain validation in `initialize()` and `setTransferPolicyId()`.
2. Permit `address(0)` in policy membership, so allowlisting zero unblocks `mint` and `burn`, while blocklisting zero can intentionally pause `mint` and `burn`.
3. Skip policy checks for `address(0)` in `SecurityToken._update()`, so `mint` and `burn` do not depend on zero-address policy membership.

Coinbase: Acknowledged. The token deployment and configuration process enforces off-chain that only BLOCKLIST-type policies are assigned as the transfer policy. This prevents an allowlist policy from ever occupying that slot and inadvertently blocking the zero address, which would cause mint and burn to revert.

Cantina Managed: Acknowledged.

3.2 Informational

3.2.1 `burnBlocked` natspec claims `to` and `origin` checks that do not exist

Severity: Informational

Context: SecurityToken.sol#L487

Description: The `@dev` natspec on `burnBlocked` says the path preserves the "to/sender/origin checks." But `_burnSkippingFromSanctions` hardcodes `to = address(0)`, and the contract never checks `tx.origin`. Only the sender check is enforced, alongside the `BURN` pause vector.

Recommendation: Reword to name only the checks that run — the `msg.sender` sanctions check and the `BURN` pause vector:

```
- /// design) while preserving the to/sender/origin checks and the
+ /// design) while preserving the `msg.sender` sanctions check and the
```

Coinbase: Fixed in commit b3b6af94.

Cantina Managed: Fix verified.

3.2.2 Init-time `ALWAYS_ALLOW_ID` guard does not extend to per-deploy `extraData` overrides

Severity: Informational

Context: B20AssetFactory.sol#L270

Description: Function `initialize()` rejects `ALWAYS_ALLOW_ID` (0) as the factory default to force sanctions gating onto every token's transfer slots.

But `_buildInitCalls()` seeds those four slots first and then appends the caller's `extraData` init-calls, so a `TOKEN_DEPLOYER_ROLE` caller can pass `updatePolicy(slot, 0)` to overwrite all four with the open-allow sentinel and the token still auto-registers with the `TokenSupplyManager`. The guard therefore constrains only the stored default, not the deployed outcome it implies.

Recommendation: Document on `initialize` / `deployToken` that the `InvalidPolicyId` check is init-only and does not prevent a trusted `TOKEN_DEPLOYER_ROLE` deployer from re-pointing the slots to 0 via `extraData` — i.e. the open-allow protection rests on the trusted-role assumption plus off-chain review, not on an on-chain end-to-end guarantee.

Optionally, for defense-in-depth, scan the decoded `extraData` for `updatePolicy(<transfer slot>, ALWAYS_ALLOW_ID)` and reject it, mirroring the init-time guard.

Coinbase: Understood, but we won't be allowing this via offchain checks, the documentation is updated in commit b3b6af9.

Cantina Managed Managed: Fix verified.

3.2.3 `setTransferPolicyId` accepts `ALWAYS_ALLOW_ID` (0) that initialize rejects (undocumented asymmetry)

Severity: Informational

Context: SecurityToken.sol#L661

Description: Function `initialize()` rejects `transferPolicyId == ALWAYS_ALLOW_ID` (0), but `setTransferPolicyId()` checks only if the policy exists. So a `DEFAULT_ADMIN_ROLE` admin can set it post-deploy, making `_isSanctioned()` return `false` for everyone and silently disabling `_checkSanctions()` on every `transfer` / `mint` / `burn`. By design: a trusted admin can reach the same fail-open state (or brick the token via an all-blocking policy / `ALWAYS_BLOCK_ID`) through any user-created policy, so a literal-0 check at the update path adds no real protection. The asymmetry is simply undocumented.

Recommendation: Add an inline comment on `setTransferPolicyId()` so readers understand why 0 is rejected at init but not here, and extend the existing admin-trust natspec to name the fail-open / brick (centralization) risk.

Coinbase: Fixed in commit b3b6af94.

Cantina Managed: Fix verified.

3.2.4 `redeemWithMemo` natspec misattributes the Memo event to the redeemer

Severity: Informational

Context: TokenSupplyManager.sol#L547-L549

Description: The `redeemWithMemo()` natspec claims the memo is emitted "associating it with the redeemer not the manager." But `transferFromWithMemo()` emits `Memo(msg.sender, memo)` (`SecurityToken.sol:637`), whose first field is the caller, and when the TSM calls it, that caller is the manager (TSM), not the redeemer.

The redeemer is actually captured by `Transfer.from` and `Redeemed.from`, which do the real tracking; the memo path exists only so a redeemer can tag their redemption.

Recommendation: Simplify the comment to drop the redeemer-attribution claim and state that `Memo.caller` is the manager, that redeemer attribution comes from `Redeemed` (and `Transfer.from`), and that the memo just lets a redeemer tag their redemption.

Coinbase: Fixed in commit b3b6af94.

Cantina Managed: Fix verified.

3.2.5 Single `UPGRADER_ROLE` controls both the shared token beacon and the factory's UUPS upgrade

Severity: Informational

Context: SecurityTokenFactory.sol#L317, SecurityTokenFactory.sol#L368

Description: `SecurityTokenFactory` defines one upgrade credential, `UPGRADER_ROLE` (`keccak256("UPGRADER_ROLE")`), and uses it to gate two distinct and independently-scoped upgrade surfaces:

- `upgradeBeacon(address newImplementation)` (`SecurityTokenFactory.sol:317`) moves the shared `UpgradeableBeacon` pointer, atomically replacing the logic of every deployed `SecurityToken` in a single call.

- `_authorizeUpgrade(address newImplementation)` (`SecurityTokenFactory.sol:368`) authorizes the factory's own UUPS implementation upgrade.

Concentrating both upgrade surfaces into a single `UPGRADER_ROLE` reduces granularity of control and increases the trust placed in that one credential, so that a single compromised key is sufficient to subvert both the factory and every deployed token at once.

Recommendation: Separate the two upgrade surfaces into distinct roles: for example `BEACON_UPGRADER_ROLE` (token implementation) and `FACTORY_UPGRADER_ROLE` (factory UUPS), so they can be assigned to different holders and governed independently (if required).

Coinbase: Fixed in commit f5d44ba2.

Cantina Managed: The fix splits `UPGRADER_ROLE` into separate `BEACON_UPGRADER_ROLE` and `FACTORY_UPGRADER_ROLE` roles. Note that `FACTORY_UPGRADER_ROLE` is a superset of `BEACON_UPGRADER_ROLE` in practice (the factory owns the beacon and UUPS allows it to re-grant anything), but the escalation is one-directional: a `BEACON_UPGRADER_ROLE` holder still cannot touch factory roles or config, so the split buys real least-privilege for the token-upgrade surface.

3.2.6 `PolicyRegistry._create` silently wraps the policy counter on overflow, diverging from the `IPolicyRegistry` specification

Severity: Informational

Context: PolicyRegistry.sol#L257-L261

Description: The `IPolicyRegistry` interface documents an explicit overflow contract for both creation entry points:

```
@dev Panics with arithmetic overflow (Panic 0x11) when the policy counter has reached
↪  its maximum value.
```

However, the implementation does the opposite. `_create` reads the uint56 `nextCounter` and increments it inside an unchecked block:

```
uint56 counter = $.nextCounter;
// No overflow guard: at one policy per 2-second block, exhausting
// the 56-bit counter space takes ~4.6 billion years.
unchecked {
  $.nextCounter = counter + 1;
}
```

`unchecked` suppresses the over/underflow check for all integer widths, `uint56` included, so at the ceiling `type(uint56).max + 1` wraps to 0 with no `Panic(0x11)`, directly contradicting the interface.

Beyond the documentation mismatch, the wrap is actively corrupting rather than merely non-conformant. The condition is unreachable in practice as exhausting the 56-bit space at one policy per ~2s block takes ~4.6 billion years. So there is no exploitable risk.

Recommendation: Make the implementation comply to the standard interface definition by removing the unchecked wrapper so the increment reverts with `Panic(0x11)` at the ceiling.

Or consider documenting this inconsistency.

Coinbase: Fixed in commit aa959f3b.

Cantina Managed: Fix verified.

3.2.7 Redundant burn pause checks

Severity: Informational

Context: SecurityToken.sol#L471-L481

Description: In the `SecurityToken` contract, `burn()` and `burnWithMemo()` call `_requireUnpaused()` for `IB20.PausableFeature.BURN` before calling `_burn()`. However, `_burn()` routes through `_update()`, which already checks the same burn pause state when `to == address(0)`. This leads to successful burns paying unnecessary gas, and the duplicated check adds maintenance noise without changing behavior.

Recommendation: Remove the redundant `_requireUnpaused()` calls and rely on `_update()`.

```
- _requireUnpaused(IB20.PausableFeature.BURN);
  _burn(msg.sender, amount);
```

Coinbase: Fixed in commit d0bec598.

Cantina Managed: Fix verified.

3.2.8 Missing memo variants for batch minting and blocked burning

Severity: Informational

Context: SecurityToken.sol#L454

Description: In `SecurityToken` and `TokenSupplyManager` contracts, several state-changing operations lack variants that accept a memo. While `mintWithMemo()` and `burnWithMemo()` exist, there are no equivalent functions for `batchMint()`, `burnBlocked()`, or `create()` in `TokenSupplyManager`.

Recommendation: Consider adding `batchMintWithMemo()`, `burnBlockedWithMemo()`, and `createWithMemo()` to provide a consistent API surface for memoized operations across all token flows.

Coinbase: Acknowledged. Memo variants are not defined in the standard token interface for batch minting or blocked burning, so omitting them keeps the implementation aligned with the interface specification we are using on other chains. The create function in TokenSupplyManager is an admin-only operation with no end-user-facing flow, so a memo variant is not needed there either.

Cantina Managed: Acknowledged.

3.2.9 Privileged roles can configure state for unregistered tokens

Severity: Informational

Context: TokenSupplyManager.sol#L223-L224

Description: In the `TokenSupplyManager` contract, several administrative functions omit a check for whether the target token is registered (`_storage().state[token].registered`). These functions include `setMinimumRedeemableShares()`, `grantRedeemPauser()`, `revokeRedeemPauser()`, `pauseRedeem()`, `unpauseRedeem()`, `removeCaller()`, and `replenishAllowance()`.

This can lead to an admin accidentally configuring state, such as pausers or redeem settings, for a token that has been deregistered or does not exist. While this does not pose a direct security risk, it can leave confusing stale configuration for an inactive token. Moreover, if the same token address is later re-registered, those previously applied settings will become active and affect the new registration unexpectedly.

Recommendation: Consider adding the `TokenNotRegistered` check to these administrative functions to ensure state is only modified for active tokens.

Coinbase: Fixed in commit 2abee10d.

Cantina Managed: Fix verified.

3.2.10 `SecurityToken` omits the IB20-mandated supply cap

Severity: Informational

Context: SecurityToken.sol#L80

Description: `SecurityToken` claims `IB20Asset` conformance (which extends `IB20`), but implements none of the IB20 supply-cap surface: there is no `supplyCap` storage field, no `supplyCap()` view, no `updateSupplyCap()` setter, and no `SupplyCapExceeded` check on any mint path.

`IB20` mandates that all mint functions revert with `SupplyCapExceeded` when `totalSupply + amount > supplyCap`.

Recommendation: Implement the full IB20 supply-cap surface. Alternatively, if the omission is intentional, document it explicitly as a deliberate divergence and remove `updateSupplyCap` references from the `B20AssetFactory` contract.

Coinbase: Fixed in commit f2b69a73.

Cantina Managed: Fix verified.

3.2.11 `SecurityToken.announce` bubbles inner revert reasons, diverging from the `IB20Asset` `InternalCallFailed` contract

Severity: Informational

Context: SecurityToken.sol#L360-L369

Description: `IB20Asset` specifies that when an inner call dispatched by `announce` reverts, the contract reverts with `InternalCallFailed(call)` and "the inner revert reason is not bubbled."

`SecurityToken.announce` inverts this: on a failed inner `delegatecall` it bubbles the inner revert data verbatim when returndata is present, and only falls back to `InternalCallFailed(internalCalls[i])` when the inner call reverts with empty returndata.

Recommendation: Decide which behavior is canonical and align the two.

Coinbase: Acknowledged. The behaviour of surfacing inner revert reasons is intentional - it gives off-chain callers and tooling actionable failure. We accept the divergence.

Cantina Managed: Acknowledged.

3.2.12 Paused token irrecoverably burns users' funds on HyperCore to HyperEVM bridge-in

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: On HyperLiquid, bridging a linked token from HyperCore into HyperEVM is not atomic and has no retry or idempotency.

The sequence is:

1. HyperCore debits (burns) the Core side first, then...
2. HyperCore calls `transfer(recipient, amount)` on the HyperEVM ERC-20 as the token's system address to credit the recipient.

If step 2 reverts, HyperCore does not retry or roll back step 1. Since the Core-side debit has already happened, the bridged tokens are irrecoverably destroyed.

The HyperEVM `SecurityToken` is pausable per-operation (`TRANSFER` / `MINT` / `BURN`) via the base `_update()` gate, and it starts fully paused at initialization (operators must explicitly unpause each feature).

The inbound bridge credit is an ordinary transfer, so it routes through `_update` to `_requireUnpaused(TRANSFER)`. Whenever `TRANSFER` is paused — at launch before unpausing, or any time an operator pauses transfers (e.g. during an incident) — every Core to EVM bridge-in reverts on the credit leg and the user's bridged tokens are permanently lost, even though the user did nothing wrong.

There is no on-chain recourse: the revert does not block the move, it burns the funds. This makes a routine, intended administrative action (pausing transfers) silently destructive to honest users who bridge in during the pause window.

The issue was identified by the Coinbase team during the course of the review.

Recommendation: Exempt the inbound system-address credit leg from the pause gate so the credit settles instead of reverting, while keeping all compliance (sanctions/policy) checks intact.

Coinbase: Fixed in commit 85df459c.

Cantina Managed: Fix verified.