Security Review

Cantina Managed

FOREST ROAD: VAULT

Cantina Managed review by:
Valerian Callens, Lead Security Researcher
Jiri123, Security Researcher

September 9, 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
- 2.2 Trust assumptions . . . 7
- 2.3 Cantina Managed Team Statement . . . 8

3 Findings **9**
- 3.1 Informational . . . 9
  - 3.1.1 Oversized redemption input panics before balance validation . . . 9
  - 3.1.2 Invalid immutable module wiring requires an upgrade to recover . . . 9
  - 3.1.3 `previewRedeem()` can publish a redeem price that would not settle . . . 9
  - 3.1.4 Direct redemption does not consume the `ADR-0033` exit interlock . . . 10
  - 3.1.5 `redeem(uint256 usdfrAmount, uint256 minUsdcOut)` has no deadline, contradicting `ADR-0034 W` . . . 10

1 Introduction

1.1 About Cantina

Cantina is a security platform that pairs a world-class network of security researchers with agentic AI to help teams building onchain find and fix real risk. Through managed reviews, competitions, and bounties, Cantina secures protocols before vulnerabilities can be exploited.
Learn more at cantina.security

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

From Aug 17th to Aug 20th the Cantina team conducted a review of forest-road-vault on commit hash 71c285ec. The team identified a total of **5** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	5	1	4
Total	**5**	**1**	**4**

2.1 Scope

The security review had the following components in scope for forest-road-vault on commit hash 71c285ec:

```
contracts/src
├── MintRedeemController.sol
└── libraries/Roles.sol
```

`MintRedeemController.sol` : `MintRedeemController` is the protocol's USDfr supply gateway. It coordinates `USDfr`, `ComplianceRegistry`, and `ReserveManager` to provide:

- KYC-gated USDC-to-USDfr minting.
- Direct USDfr redemption.
- Yield-backed protocol minting.
- Loss-absorption burns.
- Backing, deficit, redemption-price, and mint-capacity views.

The contract is a UUPS-upgradeable implementation using OpenZeppelin access control, pausing, and reentrancy protection. Its constructor disables initialization of the implementation contract.

Storage and initialization

ERC-7201 namespaced storage contains:

- `usdfr` : the USDfr token.
- `compliance` : the user eligibility registry.
- `reserves` : the backing and USDC custody manager.
- `yieldSink` : governance-approved recipients of yield mints.
- `lossSource` : governance-approved contracts from which loss burns may occur.
- `subParShortfall` : cumulative value crystallized by below-par redemptions.

`SCALE = 1e12` converts between 6-decimal USDC and 18-decimal USDfr.

`initialize` assigns `DEFAULT_ADMIN_ROLE`, `GUARDIAN_ROLE`, and `UPGRADER_ROLE` and records the three module addresses. The endpoint allowlists start empty, so credit-layer supply changes remain disabled until governance completes the wiring. The controller must separately receive `MINTER_ROLE` on `USDfr` and `CONTROLLER_ROLE` on `ReserveManager`.

Minting

`mint(usdcAmount)` is KYC-gated, pause-protected, and non-reentrant. It:

1. Rejects zero amounts, observable USDC custody shortfalls, and minting while supply already exceeds recorded backing.
2. Pulls USDC from the user and temporarily approves `ReserveManager`.
3. Calls `depositUSDC`.
4. Independently verifies the reserve's exact USDC balance increase, the reported `usdcAmount * 1e12` credit, the controller's unchanged USDC balance, and the corresponding backing increase.
5. Mints USDfr to the user.
6. Verifies that the backing deficit did not increase.

Minting is always at par and remains closed while the protocol is under-backed.

Redemption

The contract exposes three overloads:

- `redeem(amount)` applies a par minimum and therefore never silently haircuts the caller.
- `redeem(amount, minUsdcOut)` permits an explicitly accepted sub-par settlement.
- `redeem(amount, minUsdcOut, deadline)` additionally protects against delayed execution and is the canonical integration path.

The offered USDfr is rounded down to the whole-USDC grid:

```
usdfrIn = floor(usdfrAmount / 1e12) * 1e12
```

Sub-unit dust remains in the user's wallet.

When backing covers supply, redemption settles at par. When the protocol is short, the controller first attempts to draw junior capital through `ReserveManager.lossAbsorber()`. The draw target is:

```
min(ceil(usdfrIn * (supply - backing) / backing), supply - backing)
```

The draw source must be an authorized loss source. The controller measures the USDfr delivered by `drawForSeniorExit`, rejects over-reporting or under-delivery, and burns the reported amount at the source. Production routes this draw through `DefaultManager`, which uses curator first-loss capital before the sGROVE backstop.

The short-state payout is derived from:

```
min(usdfrIn, (usdfrIn + juniorDrawn) * backing / supply)
```

and rounded down to native USDC units. A partial junior draw improves the payout; a sufficient draw can restore it to par.

After checking `minUsdcOut`, the controller burns the user's USDfr, releases USDC through `ReserveManager`, and verifies the user's exact USDC balance increase. Any unpaid value is added to `subParShortfall`, preventing a later reversal of a conservative impairment from turning the exiter's crystallized loss into distributable yield.

Redemption remains limited by idle USDC liquidity even when total backing includes deployed principal.

Protocol supply paths

`mintYield(to, amount)`:

- Requires `CREDIT_ROLE`, an authorized yield sink, a nonzero amount, no controller pause, and the reentrancy lock.
- Enforces non-worsening deficits on both recorded and recognition-aware backing.
- Requires the remaining recognized surplus to cover both `subParShortfall` and `ReserveManager.exitPrepaidAbsorption()`.

In production, `WaterfallEngine` holds this role and directs mints only to the sUSDfr vault or the configured fee recipient.

`burnLoss(from, amount)`:

- Requires the separate `LOSS_BURNER_ROLE`.
- Restricts `from` to a governance-approved loss-source contract.
- Is deliberately not pause-protected, allowing loss absorption to continue during an emergency.
- Needs no backing assertion because burning supply cannot increase a deficit.

Production grants this role to `DefaultManager` and `ReserveManager`, not to `WaterfallEngine`.

Backing model and views

The controller distinguishes two backing bases:

- `backingValue()` uses `ReserveManager.totalBackingValue()`, the recorded accounting ledger.
- `recognizedBackingValue()` additionally reflects any observable idle-USDC custody shortfall.

Its principal solvency rule is:

`max(0, supplyAfter - backingAfter) <= max(0, supplyBefore - backingBefore)`

When the protocol begins whole, this reduces to `supply <= backing`. When already short, it allows operations that preserve or repair the deficit while refusing further dilution.

`mintableHeadroom()` returns zero while either the controller or USDfr is paused. Otherwise it returns:

`max(0, recognizedBacking - supply - subParShortfall - exitPrepaidAbsorption)`

`previewRedeem()` returns `(0, 0)` while redemption is paused, the reserve has a custody shortfall, the protocol is empty, or the amount is below the settleable floor. It does not simulate the junior draw, so an executable redemption may settle above the previewed floor, but the draw itself cannot make it settle below that quote.

Composite views revert while the reentrancy guard indicates an unfinished supply transition. Raw single-module readings remain available because each remains individually accurate during the transition.

Administrative and security boundaries

- `DEFAULT_ADMIN_ROLE` manages roles, yield sinks, and loss sources.
- `GUARDIAN_ROLE` pauses minting, redemption, and yield minting.
- `UPGRADER_ROLE` authorizes UUPS upgrades.
- New loss sources must have deployed contract code and cannot be EIP-7702 delegated EOAs. Revocation remains unconditional.
- Governance can still authorize an unsuitable contract as a loss source; pro-rata burn safety therefore also depends on correct deployment wiring.
- Exact balance-delta checks make under-delivery, over-delivery, dishonest reserve accounting, and controller-side cash stranding fail closed.
- Instant redemption is first-come-first-served with respect to available idle liquidity.

`Roles.sol`: `Roles.sol` is the generated canonical namespace for protocol role identifiers. It contains no storage, holder registry, or authorization logic. Each role is an `internal constant` equal to `keccak256("<ROLE_NAME>")`.

Its source of truth is `config/privilege-topology.json`; the generated Solidity file should not be edited directly.

Role identifiers are shared across the system, but grants are local to each `AccessControl` contract. Holding `CREDIT_ROLE` on one module grants no authority on another module. No protocol contract overrides `_setRoleAdmin`, so every role is administered by that contract's `DEFAULT_ADMIN_ROLE`.

Role	Purpose and intended holder
`DEFAULT_ADMIN_ROLE`	OpenZeppelin's `bytes32(0)` root role. Grants and revokes other roles and controls governance-only setters. Held by the governance timelock in production.
`UPGRADER_ROLE`	Authorizes UUPS upgrades. Held by the governance timelock.
`GUARDIAN_ROLE`	Pauses and unpauses value-moving paths and activates bounded emergency controls. Held by the guardian.
`MINTER_ROLE`	Authorizes both `USDfr.mint` and `USDfr.burn`. Held only by `MintRedeemController`.
`CONTROLLER_ROLE`	Authorizes reserve deposit and release operations. Held by `MintRedeemController`.
`CREDIT_ROLE`	Grants trusted module-to-module credit primitives, including accounting updates, attestation consumption, loss allocation, and yield minting. Never intended for an EOA.
`LOSS_BURNER_ROLE`	Authorizes `MintRedeemController.burnLoss`. Separated from `CREDIT_ROLE` so the repayment engine cannot combine mint and burn powers. Held by `DefaultManager` and `ReserveManager`.
`FEE_ACCOUNTING_ROLE`	Brackets fee-neutral junior-capacity changes in sUSDfr. Held by `CuratorModule`, `SGrove`, and `DefaultManager`.
`COMPLIANCE_ADMIN_ROLE`	Manages user KYC eligibility and jurisdiction blocking, but not protocol exemptions. Held by compliance operations.
`RESERVE_ADMIN_ROLE`	Ratifies bounded custody losses and records physically recovered reserve capital. Held by the governance timelock in production.
`ORIGINATOR_ROLE`	Creates, amends, and cancels facilities through `ClaimBridge`, subject to attestation and concentration checks. Held by origination operations.
`ATTESTER_ROLE`	Identifies approved signers in the m-of-n attestation system. It is checked against recovered signatures rather than used as a caller modifier.
`SERVICER_ROLE`	Executes facility funding, repayment distribution, default, acceleration, loss realization, and cure operations, generally against authenticated facts.
`SETTLEMENT_KEEPER_ROLE`	Calls `RedemptionQueue.closeEpoch` and controls when its bounded liquidity snapshot is taken. Held by a dedicated keeper plus an operational backstop.

The former unused `QUEUE_ROLE` was removed. Queue-only vault operations instead authenticate the configured queue address directly, while `SETTLEMENT_KEEPER_ROLE` governs the economically meaningful epoch-closing operation.

2.2 Trust assumptions

1. Actor and external-system trust assumptions:

- A valid governance outcome executed through the Governor and governance timelock is treated as the protocol's trusted administrative root. Compromise or malicious use of sufficient governance authority is outside the smart-contract threat model.
- Privileged actors are trusted to perform the off-chain diligence required by their roles and to use their authority only under the applicable procedures.
- The review assumes adherence to the principle of Segregation of Duties. No single actor is expected to simultaneously hold operational roles intended to remain independent, and collusion between incompatible roles is considered out of scope unless explicitly stated otherwise.
- Attesters are the primary trust boundary for off-chain facts. Once `AttestationOracle` accepts the required signatures, the protocol treats the asserted fact as authoritative and cannot independently verify the underlying servicing, valuation, payment, default, amendment, assignment, or perfection event.
- Security therefore assumes that no dishonest or compromised set of current attester keys can satisfy the applicable threshold. Governance is trusted to maintain enough available and independently controlled keys, rotate compromised keys, and configure usable thresholds.
- The operational Guardian is trusted to pause and unpause guarded paths only under the incident procedures. These controls can affect issuance, redemption, transfers, staking, claims, reserve operations, funding, distributions, attestations, and queue settlement. Required compliance exemptions must remain correctly wired so internal loss-burn and cascade legs can execute.
- `ReserveManager.armReserveLossFreeze()` creates a persistent reserve-loss arm with no expiry and closes new senior queue settlement and curator first-loss withdrawals. `GUARDIAN_ROLE` alone cannot cancel, adjudicate, finalize, or execute the arm; governance or the reserve-loss administrator must perform the applicable terminal or accounting action. Protected exits may remain closed indefinitely if governance does not respond, while queue claims filled before the lock remain claimable.
- `CuratorModule.preArmCustodyFreeze` is a separate, time-bounded and contract-budgeted control that governance may cancel or replenish.
- Curators are trusted only as to facts the contracts cannot establish. Governance is assumed to approve each curator for the appropriate collateral class after due diligence, and capital presented as first-loss capital is assumed to be legitimately controlled, unencumbered, and intended to bear the disclosed junior risk.
- Curators and users are otherwise untrusted. Slippage, deadline, authorization, sanctions, pause, and accounting checks remain in scope.
- Production is assumed to use the canonical Ethereum-mainnet USDC proxy as the sole reserve token. The protocol treats one USDC as one US dollar and has no independent depeg oracle.
- USDC is assumed to retain six-decimal, non-rebasing, no-fee-on-transfer, and no-transfer-callback behavior. Circle and the USDC administration are assumed not to pause USDC, blacklist protocol-owned addresses required for core operation, or deploy an incompatible upgrade. Depeg, issuer insolvency, blacklist, pause, and incompatible upgrade risks are external asset and availability risks; exact receipt and delivery checks remain in scope.
- The truth and performance of off-chain servicing, physical custody restoration, banking and fiat movements, and legal enforcement are outside the smart-contract review.

2. Intended and accepted protocol behavior: The following behavior is disclosed so it is not misreported as a safety guarantee or trust assumption:

- Senior capital is impairable, and direct redemption may settle below par after available junior protection is applied. Prevention of every senior loss is not a protocol invariant. This statement does not approve any deviation between the implemented direct-exit waterfall and the protocol specification; any such deviation is assessed separately. The `redeem(uint256)` overload settles at par or reverts; a holder opts into a sub-par result only through an overload that supplies `minUsdcOut`.

- Realized facility and adjudicated custody losses apply the configured junior and coverage layers before eligible senior-vault assets. Facility loss realization reverts if the residual exceeds those assets; custody loss recognition may instead record a remaining reserve deficit.
- For direct USDfr redemption, any junior draw is atomic with the exit. When `0 < backing < supply`, the target is based on the aggregate book deficit, and pricing uses the junior capital actually delivered. If the deficit reflects a reversible mark, this may crystallize junior capital even if the loss does not later occur.
- A shortfall between the idle-USDC ledger and actual custody closes direct mint and redeem until custody is restored or an arm-bound accounting action reduces the ledger. `reconcileIdleUSDC()` observes the difference but does not itself write down the ledger.

2.3 Cantina Managed Team Statement

Forest Road engaged Cantina to conduct a security review of the Forest Road Vault. We would like to thank the Forest Road team for their responsiveness and constructive engagement throughout the review process. No significant issues were identified during the assessment, and the protocol is expected to operate as intended.

3 Findings

3.1 Informational

3.1.1 Oversized redemption input panics before balance validation

Severity: Informational.

Context: MintRedeemController.sol#L1627.

Description: In `_quoteRedeem()`, the redemption quote calculates `usdfrIn + drawn` before checking the caller's `USDfr` balance. A max-sized input with a nonzero draw therefore overflows and reverts, instead of returning the intended token or controller error. No funds are at risk because the caller cannot hold or burn such an amount.

Recommendation: Check the value of `usdfrAmount` against the caller's balance before calculating the quote.

Forest Road: Acknowledged the overflow panic at `usdfrIn + drawn`, only reachable on the under-backed path, no funds at risk.

Cantina Managed: Acknowledged.

3.1.2 Invalid immutable module wiring requires an upgrade to recover

Severity: Informational.

Context: MintRedeemController.sol#L279.

Description: In `initialize()`, the controller only checks that the `usdfr`, `compliance`, and `reserves` addresses are nonzero during initialization. A wrong or codeless address would leave the controller unusable, and the wiring cannot be corrected without an upgrade.

This is limited to deployment misconfiguration and is mitigated by the hash-bound deployment manifest, post-deployment validation, and verification of the live addresses.

Recommendation: Retain immutable wiring, but ensure deployment validation checks that each address contains code and matches the expected module configuration before the deployment is accepted.

Forest Road: Acknowledged as accurate with the wiring kept immutable, and the recommended validation is already in place; post-deploy checks assert `controller.modules()` against the manifest and confirm code through the ERC-1967 implementation slot, so the residual is limited to deployment misconfiguration and already mitigated.

Cantina Managed: Acknowledged.

3.1.3 `previewRedeem()` can publish a redeem price that would not settle

Severity: Informational.

Context: MintRedeemController.sol#L1364.

Description: The `previewRedeem()` view function is meant to publish the executable redemption price. However, it uses two checks that are not aligned with the logic of the `redeem()` function. It checks that none of the `USDfr` and the `MintRedeemController` contracts are paused, but it does not check if the `ReserveManager` contract is paused, so it can return a full quote while the `releaseUSDC()` function is blocked and every redemption reverts. It also passes `drawn` as zero into `_quoteRedeem()`, so in a below-par context, it publishes the gross marked price while `redeem()` settles at the junior drawn price.

Recommendation: Verify if the `ReserveManager` contract is paused alongside the other two contracts, and return `(0, 0)`. Also, document the returned quote as a lower bound until the draw can be simulated.

Forest Road: Acknowledged the `previewRedeem` mismatch, which skips the `ReserveManager` pause check so it can quote a full price while redemptions revert, no funds at risk.

Cantina Managed: Acknowledged.

3.1.4 Direct redemption does not consume the `ADR-0033` exit interlock

Severity: Informational.

Context: MintRedeemController.sol#L594.

Description: `ADR-0033 §5` states that junior and senior exits share one interlock, so neither cohort can escape while the other stays exposed. `CuratorModule.withdrawFirstLoss()` and `RedemptionQueue.closeEpoch()` read `reserveLossExitsLocked()`, but `MintRedeemController._redeem()` does not, so direct redemption stays open while both other exits are shut. The comment at line `RedemptionQueue.sol:337` calls the queue the sole senior exit, which is what §5 was written against.

Most locked states are already covered elsewhere, since `supply > backing` and a latched deficit both price sub par and a live shortfall reverts in `_requireCustodiedReserve()`. The state that leaves par reachable is an active arm before any physical shortfall.

Recommendation: Correct `ADR-0033 §5` and the `RedemptionQueue` comment either way.

Forest Road: Accepted as a known limitation of mainnet v1, not remediated in this release.

Cantina Managed: Acknowledged.

3.1.5 `redeem(uint256 usdfrAmount, uint256 minUsdcOut)` has no deadline, contradicting `ADR-0034 W`

Severity: Informational.

Context: MintRedeemController.sol#L453.

Description: `ADR-0034 W` requires the `redeem()` functions to take both a minimum out and a deadline, However, only the three-argument form of the `redeem()` function matches that requirement. The one-argument form supplies the par floor internally, so it settles at par or reverts and a delay cannot make it settle worse.

The two-argument form bounds price, but not time, which is the free option W describes: a caller who names a floor and is not included for an hour lets a searcher hold the transaction until the ratio decays to that floor.

Recommendation: Amend W to record which form is canonical, and point integrators at the three-argument form.

Forest Road: Fixed in commit ab844dc.

Cantina Managed: Fixed in the commit above, by updating `ADR-0034 W` accordingly.