infiniFi PR 245

Security Review

Cantina Managed review by:
R0bert, Lead Security Researcher
Slowfi, Security Researcher

January 20, 2026

Contents

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

infiniFi is a self-coordinated depositor-driven system designed to tackle the challenges of duration gaps in traditional banking.

From Jan 6th to Jan 7th the Cantina team conducted a review of infinifi-contracts on commit hash 69e6f2ca. The team identified a total of **20** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	4	3	1
Gas Optimizations	1	1	0
Informational	15	7	8
Total	**20**	**11**	**9**

2.1 Scope

The security review had the following components in scope for infinifi-contracts on commit hash 69e6f2ca:

```
src
├── core
│   └── InfiniFiCore.sol
├── finance
│   └── PerformanceFeeSplitter.sol
├── gateway
│   └── InfiniFiGatewayV3.sol
├── governance
│   └── PeriodicSwapper.sol
├── integrations
│   ├── MultiAssetFarmV2.sol
│   └── outland
│       └── connectors
│           └── ConnectorLZ.sol
├── libraries
│   └── CoreRoles.sol
└── tokens
    └── GovernanceToken.sol
```

3 Findings

3.1 Low Risk

3.1.1 Migrate lacks user min output

Severity: Low Risk

Context: InfiniFiGatewayV3.sol#L16-L28

Description: The `migrate` flow in `InfiniFiGatewayV3` forwards funds into the migration controller and relies on external farm accounting and oracle pricing to determine how much iUSD is minted. There is no user supplied minimum output check, so a price swing, stale feed, or manipulation within the same block can reduce the minted amount without any user level revert condition. This removes user control over slippage and makes the outcome dependent on external pricing at the moment of execution. Impact: users can receive materially less iUSD than expected with no protection.

Recommendation: Consider adding a user provided minimum output parameter to migrate and enforce it against the minted iUSD amount before returning.

infiniFi: Fixed in commit a06866de.

Cantina Managed: Fix verified.

3.1.2 Oracle removal can block asset disable

Severity: Low Risk

Context: MultiAssetFarmV2.sol#L79-L84

Description: `MultiAssetFarmV2` requires an asset to be supported in order to disable it and support is defined as the asset being listed and having a valid oracle. If an oracle is removed or reverts, `isAssetSupported` becomes false and `_disableAsset` reverts. At the same time, `assets()` calls `Accounting.price` directly for every enabled asset, so a missing or reverting oracle causes `assets()` to revert and blocks any operations that depend on it such as deposit and withdraw. This is recoverable by setting a temporary oracle, but during the outage the farm cannot be maintained or have its asset list corrected.

Impact: a reverting or removed oracle can temporarily freeze accounting dependent flows and prevent asset removal until governance intervenes.

Recommendation: Allow disabling assets even when the oracle is invalid, or add an emergency removal path that does not require a valid oracle. Consider skipping assets with invalid oracles in `assets()` to avoid total reverts and allow graceful recovery.

infiniFi: Acknowledged. If we really want to disable an asset that has a reverting oracle, we can also update the oracle in the same payload. And if an oracle starts to revert we do not want an automatic way to disable it / replace the oracle, we want accounting to be paused, to avoid erroneous spikes in `assets()` reportings (better to have a pause than report an erroneous value).

Cantina Managed: Acknowledged.

3.1.3 `GovernanceToken` owner/delegate not moved

Severity: Low Risk

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

Description: The `GovernanceToken` and the locked governance share tokens are deployed with the `deployer` as the `owner` and as the LayerZero delegate. The proposal that introduces these tokens does not transfer ownership or update the delegate afterward. This leaves the `deployer` with the ability to configure the OApp, including setting peers and delegates, which effectively controls cross chain messaging for the token. If the deployer key is compromised or misconfigured, bridging configuration can be changed or halted, which can disrupt token transfers and governance distribution flows. The impact is a persistent centralization and operational risk around bridge configuration.

Recommendation: Add explicit post deploy steps to transfer ownership of each `GovernanceToken` instance to the intended admin (timelock or multisig) and call `setDelegate` to that same address. Set peers only after ownership transfer so the final owner controls OApp configuration.

infiniFi: Fixed in commit 644de6fb.

Cantina Managed: Fix verified.

3.1.4 L2 minting can create unescrowed supply that cannot be bridged to L1

Severity: Low Risk

Context: GovernanceToken.sol#L46-L48

Description: The function `mint` from contract `GovernanceToken` mints tokens on any chain where `_MINTER_ROLE` is granted. However, the OFT bridge logic assumes that L2 → L1 credits are paid out from the L1 `escrow`:

- In `_debit`, L2 → L1 burns the tokens on the source chain (`_burn(_from, amountSentLD)`).
- In `_credit`, when `block.chainid == 1`, the destination credit is executed via `_update(escrow, _to, _amount)`.

As a result, if tokens are minted on an L2, that supply is not backed by tokens held in the L1 `escrow`. When a holder attempts to bridge such tokens from L2 to L1, `_credit` on L1 will revert if `escrow` does not have sufficient balance, causing the cross-chain receive to fail. This makes L2-minted supply effectively non-bridgeable to L1 under insufficient L1 escrow conditions and can also create failed inbound messages that require manual recovery/handling.

Recommendation: Consider to restrict `mint` to the hub chain, L1, so that all circulating supply remains consistent with the L1 escrow-backed credit model, or adjust the L1 `_credit` logic/design to handle unescrowed supply (e.g., minting on L1 for certain flows or introducing explicit accounting/invariants that prevent escrow underfunding).

infiniFi: Fixed in commit d29748cc.

Cantina Managed: Fix verified.

3.2 Gas Optimization

3.2.1 Unused state variable `splitB`

Severity: Gas Optimization

Context: PerformanceFeeSplitter.sol#L27

Description: The function `constructor` from contract `PerformanceFeeSplitter` assigns the value of `_splitB` to the state variable `splitB`. However, `splitB` is never read or used anywhere else in the contract, making it an unused state variable.

Recommendation: Consider to remove `splitB` and its associated constructor parameter if it is not required, or integrate it into the fee-splitting logic if it is intended to be used.

infiniFi: Fixed in commit 7851b1bc.

Cantina Managed: Fix verified.

3.3 Informational

3.3.1 `PerformanceFeeSplitter` could revert on zero transfers

Severity: Informational

Context: PerformanceFeeSplitter.sol#L45-L46

Description: The `PerformanceFeeSplitter` always calls `safeTransfer` for both receivers after computing the shares. If one share is zero, the contract still performs a zero value transfer. Some non standard ERC20 tokens revert on zero amount transfers, which would cause `split()` to revert even though the balance is nonzero. This can happen when a split is configured as 100% to one side or when rounding

makes `aShare` equal to zero for small balances. Impact: a single zero transfer can block fee distribution for non standard tokens.

Recommendation: Skip transfers when the computed amount is zero or enforce both splits to be nonzero in the constructor. If supporting tokens that revert on zero transfers is important, add explicit zero checks before calling `safeTransfer`.

infiniFi: Fixed in commit d3b116c3.

Cantina Managed: Fix verified.

3.3.2 `PeriodicSwapper` assumes `buyToken` matches `receiptToken`

Severity: Informational

Context: PeriodicSwapper.sol#L21-L42

Description: `PeriodicSwapper.distribute` approves and deposits `buyToken` into the locking controller as rewards. This assumes `buyToken` is the same token as the locking controller `receiptToken`. If the swapper is misconfigured with a different `buyToken`, `depositRewards` will revert and halt distributions until the configuration is corrected. This is a configuration risk rather than an on chain exploit, but it can block rewards in production if addresses are set incorrectly.

Recommendation: Consider adding a constructor or runtime check that `buyToken` equals `LockingController(lockingController).receiptToken()`. This makes misconfiguration fail early and protects distributions from silent setup errors.

infiniFi: Fixed in commit 5616c561.

Cantina Managed: Fix verified.

3.3.3 `PerformanceFeeSplitter` parameters are immutable

Severity: Informational

Context: PerformanceFeeSplitter.sol#L15-L28

Description: The `PerformanceFeeSplitter` stores receiver addresses and split ratios as `immutable` variables, which cannot be updated after deployment. If the recipient addresses or desired split change in the future, the only option is to deploy a new splitter and update any upstream contracts that point to it. This is expected for simplicity, but it is a permanent configuration choice that reduces operational flexibility.

Recommendation: If future changes are expected, use an upgradeable splitter or add a controlled setter with appropriate governance checks. Otherwise document that the splitter is intentionally immutable and must be replaced to change recipients or ratios.

infiniFi: Acknowledged. We'll keep it immutable and will deploy a different splitter if the ratio need to change

Cantina Managed: Acknowledged.

3.3.4 Escrowed balances count in total supply

Severity: Informational

Context: GovernanceToken.sol#L95

Description: On L1, bridged out tokens are transferred to an `escrow` address instead of being burned. `ERC20Votes` only changes total supply on `mint` or `burn`, so total supply checkpoints do not decrease when balances move into escrow, even though the sender's voting power is reduced. If future governance relies on `totalSupply` or `getPastTotalSupply` for quorum on L1 and L2 holders cannot vote on L1, large bridge outs can inflate quorum relative to active voting power and make proposals harder to pass.

Impact: Quorum can become misaligned with circulating voting power under L1 only governance assumptions.

Recommendation: If L1 quorum will be based on `ERC20Votes` total supply, exclude the `escrow` balance from quorum calculations or treat escrowed tokens as burned for voting supply. Alternatively, ensure governance is cross chain or uses a quorum definition that reflects circulating voting power.

infiniFi: Acknowledged. We'll keep in mind for when governance is here.

Cantina Managed: Acknowledged.

3.3.5 Migration controller not configured

Severity: Informational

Context: Proposal_11SEP25.sol#L25

Description: The deployment proposals deploy the `MigrationController` and wire it into the gateway, but no proposal sets any migration configs. Without configuration, `migrate` will revert because the per farm and token caps, minimums, fees and optional hooks are unset. This is not a code bug but a missing operational step that will leave migration inactive until a separate governance or ops action configures each farm and token. Impact: migration flows are unusable until configs are set.

Recommendation: Consider adding explicit `setConfig` steps in a follow on proposal or document an operational runbook to configure migration parameters before enabling user access.

infiniFi: Acknowledged. We do not enable any migration config yet on purpose.

Cantina Managed: Acknowledged.

3.3.6 `PeriodicSwapper` approve may fail for tokens

Severity: Informational

Context: PeriodicSwapper.sol#L36-L42

Description: `PeriodicSwapper.distribute` uses a direct `approve` call on `buyToken`. Some ERC20 implementations require setting allowance to zero before updating it, or revert when approve returns no value. This can make `distribute` revert even when `buyToken` is otherwise compatible. The current setup uses a standard token, but the contract itself is not robust to non standard ERC20 behavior.

Recommendation: To follow best practices, consider using `SafeERC20.forceApprove` or a zero reset pattern before approve to support non standard tokens.

infiniFi: Fixed in commits 943e4f4c and de332ee6.

Cantina Managed: Fix verified.

3.3.7 INFI oracle disables slippage guard

Severity: Informational

Context: Proposal_21NOV25.sol#L227-L230

Description: The TGE proposal deploys the INFI oracle as a fixed price of 1000 USD. `SwapFarmV2` relies on the accounting oracle to compute the expected output in `convert()` and then applies `maxSlippage` against that oracle based value for both aggregator swaps and CoW orders. With a fixed high price, the minimum output check becomes extremely low relative to actual market prices, so swaps can execute at very unfavorable rates without reverting. This behavior is intentional to bypass slippage checks during bootstrap, but it shifts protection entirely to offchain quoting and keeper or router trust. If the `FARM_SWAP_CALLER` or swap path is compromised or misconfigured, performance fee funds can be swapped at a large discount while still passing on chain checks.

Impact: Buyback swaps can lose significant value without triggering a revert.

Recommendation: Consider documenting this is an expected and wanted behaviour.

infiniFi: Acknowledged. This is a temporary situation.

Cantina Managed: Acknowledged.

3.3.8 `PeriodicSwapper` assets may skew accounting

Severity: Informational

Context: PeriodicSwapper.sol#L14

Description: `PeriodicSwapper` inherits `MultiAssetFarmV2` and therefore reports `assets()` based on its token balances and oracle prices. If the swapper is mistakenly added to the `FarmRegistry`, its balances could be included in protocol accounting and skew total assets and related metrics. The contract is not intended to be a farm and explicitly states it should not be registered, but there is no on chain enforcement to prevent accidental inclusion.

Impact: Accounting figures can be distorted if the swapper is registered by mistake.

Recommendation: Consider overriding `assets()` to return zero so the contract cannot affect accounting even if listed by mistake.

infiniFi: Fixed in commit 4cfa1921.

Cantina Managed: Fix verified.

3.3.9 Unchecked ERC20 calls in INFI paths

Severity: Informational

Context: InfiniFiGatewayV3.sol#L34-L36

Description: The new INFI gateway functions use direct `transferFrom` and `approve` calls on the governance token and its share tokens without SafeERC20 wrappers. Non standard ERC20 implementations may return false instead of reverting or require zero reset allowances, which can make lock or unwinding flows revert or behave unexpectedly. This is low risk for the current `GovernanceToken` implementation but reduces robustness if the token changes or if share tokens are replaced by a non standard implementation.

Recommendation: Use `SafeERC20.safeTransferFrom`and `SafeERC20.forceApprove`(or a zero reset pattern) for all token interactions in the INFI gateway functions to support non standard ERC20 behavior.

infiniFi: Acknowledged. The forge-link warning should be enough for us not to copy paste mindlessly.

Cantina Managed: Acknowledged.

3.3.10 `PeriodicSwapper` direction not enforced

Severity: Informational

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

Description: `PeriodicSwapper` stores `sellToken` and `buyToken` but does not use them in its swap paths. The inherited swap functions accept any pair of supported tokens and the pair configuration is symmetric, so a keeper can sign or execute swaps in either direction once the pair is configured. That means the contract can swap `buyToken` back into `sellToken` even if the design intent is to only convert `sellToken` into `buyToken` before distributing rewards.

Recommendation: Add an explicit direction check in `PeriodicSwapper` by overriding the swap functions and requiring `tokenIn == sellToken` and `tokenOut == buyToken`. If both directions are intended, replace the `sellToken` and `buyToken` fields with a direction-specific configuration or document the invariant clearly.

infiniFi: Fixed in commit 49d78b38.

Cantina Managed: Fix verified.

3.3.11 Unused custom error `TransferFailed`

Severity: Informational

Context: PerformanceFeeSplitter.sol#L13

Description: The function `constructor` from contract `PerformanceFeeSplitter` declares the custom error `TransferFailed(address token, address receiver, uint256 amount)`. However, this error is never referenced or reverted to anywhere in the contract, making it unused.

Recommendation: Consider to remove the `TransferFailed` custom error if it is not required, or use it in the relevant transfer failure paths if it is intended to be part of the error-handling logic.

infiniFi: Fixed in commit a9b8050c.

Cantina Managed: Fix verified.

3.3.12 Missing event emission on fee split

Severity: Informational

Context: PerformanceFeeSplitter.sol#L47

Description: The function `split` from contract `PerformanceFeeSplitter` performs the fee distribution logic but does not emit any event upon execution. As a result, off-chain systems and users cannot easily observe or index fee split operations without relying on token transfer events alone, which do not fully capture the semantic meaning of the action.

Recommendation: Consider to emit a dedicated event when `split` is executed, including relevant parameters such as the token address and distributed amounts, to improve observability and off-chain tracking.

infiniFi: Acknowledged.

Cantina Managed: Acknowledged.

3.3.13 `split` is not reentrancy safe for callback tokens

Severity: Informational

Context: PerformanceFeeSplitter.sol#L40

Description: The function `split` from contract `PerformanceFeeSplitter` performs external token transfers to `receiverA` and `receiverB`. If `token` is a callback-enabled/non-standard ERC20 (e.g., ERC 777-like behavior) and/or either receiver is a contract that can re-enter during `transfer`, `split` can be re-entered mid-execution. In this scenario, the outer call may later attempt to transfer amounts computed from a stale pre-transfer balance, which can revert due to insufficient remaining balance, effectively making `split` susceptible to a denial-of-service in such token/receiver configurations.

Recommendation: Consider to make `split` reentrancy-safe (e.g., using a reentrancy guard), or restructure the transfers so the second transfer amount is derived from the post-transfer remaining balance (e.g., transferring the remaining balance to `receiverB` after sending `receiverA`'s share).

infiniFi: Fixed in commit c6ba57a8.

Cantina Managed: Fix verified.

3.3.14 Missing zero address validation for fee receivers

Severity: Informational

Context: PerformanceFeeSplitter.sol#L24-L25

Description: The function `constructor` from contract `PerformanceFeeSplitter` assigns `receiverA` and `receiverB` from user-provided inputs without validating they are non-zero addresses. If either receiver is set to `address(0)`, subsequent calls to `split` may transfer funds to the zero address, resulting in an unintended loss of funds.

Recommendation: Consider to validate `receiverA` and `receiverB` are non-zero addresses in the constructor and revert on misconfiguration.

infiniFi: Acknowledged. We don't usually check on address 0 & assume the deployment is checked.

Cantina Managed: Acknowledged.

3.3.15 No pause mechanism for critical token state transitions

Severity: Informational

Context: GovernanceToken.sol#L19

Description: The contract `GovernanceToken` is `CoreControlled` but does not implement any pause/unpause mechanism to halt critical state transitions in contingency situations. As a result, if an incident occurs affecting token transfers, voting power updates, minting/burning, or cross-chain messaging, the system does not provide an on-chain mechanism to temporarily stop these operations.

Recommendation: Consider to add a pause/unpause mechanism and apply it to the relevant state transition functions to improve operational control during emergencies.

infiniFi: Acknowledged.

Cantina Managed: Acknowledged.