

Definitive: EVM Flash Contracts Review

Security Review

Cantina Managed review by:
Valerian Callens, Lead Security Researcher
Chinmay Farkya, Security Researcher

June 15, 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 Codebase context . . . 3
- 2.3 Trust assumptions . . . 3
- 2.4 Cantina Managed Team Statement . . . 4

3 Findings **5**
- 3.1 Medium Risk . . . 5
  - 3.1.1 Client Admin can pull tokens through vault reentrancy . . . 5
- 3.2 Low Risk . . . 6
  - 3.2.1 Temporary DOS of `pullFlashERC2612()` function is possible . . . 6
  - 3.2.2 Inconsistent usage of `_fetchFeeAccount()` for the management of input fees . . . 7
  - 3.2.3 Function `isValidSignature()` not gated by modifier `stopGuarded` . . . 7
  - 3.2.4 Full consumption of `inputAmount` not checked in delegate swap mode . . . 8
  - 3.2.5 Full consumption of `inputAmount` not checked in non-delegate swap mode . . . 8
  - 3.2.6 Client Admin can pull tokens through UserOp . . . 9
- 3.3 Informational . . . 10
  - 3.3.1 Function `_executeFlashSwap()` returns an output amount before fees . . . 10
  - 3.3.2 Consolidated suggestions to improve the documentation . . . 10
  - 3.3.3 No event emitted when the allowance contract is updated in `GlobalGuardian` . . . 11
  - 3.3.4 Function `validateSwapPayloadSigner()` declared but not used . . . 11
  - 3.3.5 Ownership of `DefinitiveFlashAllowance` contract can be renounced . . . 12
  - 3.3.6 Test suite does not cover flash swaps where the output token is the native token . . . 12

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

Definitive Finance delivers a CeFi-like experience on DeFi rails via a fully non-custodial platform & API that is live across Solana, Base, HyperEVM and all other major EVM chains.

From May 6th to May 13th the Cantina team conducted a review of contracts on commit hash c515674d. The team identified a total of **13** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	6	5	1
Gas Optimizations	0	0	0
Informational	6	4	2
Total	**13**	**10**	**3**

2.1 Scope

The security review had the following components in scope for contracts on commit hash c515674d:

```
contracts/
├── base/BaseFlashSwap/BaseFlashSwap.sol
├── base/BaseFlashSwap/IBaseFlashSwap.sol
├── tools/FlashAllowance/DefinitiveFlashAllowance.sol
└── tools/FlashAllowance/IDefinitiveFlashAllowance.sol
```

In addition, the integration of the Flash feature with the existing system was in scope, which involved in particular the following files:

```
contracts/
├── base/BaseAccessControlInitiable.sol
├── base/BaseAccountAbstraction.sol
├── strategies/Trading/v2/TradingVaultImplementation.sol
└── tools/GlobalGuardian/GlobalGuardian.sol
```

2.2 Codebase context

The Definitive Flash feature lets integrators place orders on Definitive Finance on behalf of users: funds stay in the user's wallet until the swap occurs, and output is sent back to the user's wallet or another user-defined recipient. Orders are submitted by trusted actors (the performers), and can take multiple forms like multi-fill (e.g., TWAPs), and can also be filled at a later date (e.g., limit orders). The review mainly focused on the contracts `DefinitiveFlashAllowance` (a singleton contract acting as an allowance gateway and supporting multiple allowance methods) and `BaseFlashSwap` (a capability added to the existing integrator trading vaults). The vaults also have other capabilities defined in other contracts, such as swap execution, fee management, account abstraction, access control and guardian protection. Four roles are involved in that system: the users, the performers, the integrator vault admins, and the Definitive admins.

2.3 Trust assumptions

- Performers are assumed to be trusted. They can theoretically extract value by forcing bad trades or taking 100% native-asset fees. The user accepts and trusts the performers to make honest trades.
- It is assumed that no address will have at the same time the roles of performer and vault admins (also called client admins).

- Contracts are upgradeable, and the Definitive team uses a multisig for upgrades with extensive validations to prevent malicious changes.

2.4 Cantina Managed Team Statement

Definitive Finance engaged Cantina to conduct a security review of the Definitive Flash feature and its integration with existing vault permissions, swap handling, fees, and guardian controls. We would like to thank the Definitive Finance 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 Medium Risk

3.1.1 Client Admin can pull tokens through vault reentrancy

Severity: Medium Risk

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

Description: The contract `DefinitiveFlashAllowance` is expected to prevent any unexpected token pulls that would not be used to perform a flash swap, and does so with:

1. A modifier `onlyPerformer` allowing calls only when `tx.origin == performer`;
2. A verification that the value of `msg.sender` matches the value of the field `order.vault` in the order object (`FlashOrder` or `QuickOrder`) signed by the owner of the funds;

Still, it is possible for a Client Admin (owner of the integrator vault) to request the contract `DefinitiveFlashAllowance` to pull tokens directly, without using it for a swap, through reentrancy triggered when receiving tokens, because the function `BaseAccountAbstraction::executeBatch()` does not reject reentrant calls. Here are the steps:

1. As first precondition, a flash order signed by Alice (for example: 10 WETH) should be partially filled (for example: 1 WETH), so the order and the associated signatures are visible on-chain;
2. As second precondition, an address called Bob with the Client Admin role for a given integrator vault needs to set a function `attack()` with a custom logic that will execute the function `BaseAccountAbstraction::executeBatch()`. The Client Admin can do so in at least three ways:
   - Grant the `DEFAULT_ADMIN_ROLE` to a contract with that function `attack()`;
   - As upgradeable contract, he can update the contract to another implementation including the function `attack()`;
   - As EOA, he can use an EIP-7702 delegation to a delegated contract with that function `attack()`;
3. Bob signs a `FlashOrder` with an output token being either native asset, or an ERC-777 token (supporting hooks on transfers) to be sent to Bob as swap recipient (or any intermediary address further forwarding the call to Bob) that will react when receiving tokens with a call to `Bob.attack()`;
4. Bob's order is processed by a performer who submits it via the function `BaseFlashSwap::flashSwap()` of Bob's integrator vault. Since the function is protected by the modifier `nonReentrant`, the reentrancy flag is now enabled on the vault.
5. During the swap execution, the recipient set by Bob with a custom logic on token reception triggers the logic, which executes the function `Bob.attack()`;
6. The function `Bob.attack()`, executed with Bob as `msg.sender`, meaning with the Client Admin role on Bob's integrator vault will call the function `BaseAccountAbstraction::executeBatch()` which is not using the modifier `nonReentrant`, with the batch of calls: `[Call_1 - DefinitiveAllowance::pullFlashAllowance(order: orderOfAlice, signature: signatureOfAlice, pullAmount: 9 WETH), Call_2 - WETH:transfer(Bob, 9 WETH)]`.

That scenario is possible, for the following reasons:

- `Call_1` is allowed because the function `BaseAccountAbstraction::executeBatch()` allows any method to be executed by Client Admins, and does not use the modifier `nonReentrant`;
- `Call_2` transfers the pulled amount outside of the integrator's vault.
- If executed as part of a legit performer call, the modifier `onlyPerformer` will pass in the function `pullFlashAllowance()`. In that case, `msg.sender` will be Bob's vault, which matches what was allowed in Alice's order.
- And what was not yet swapped for Alice's order is transferred to Bob's vault.

As impact, it would mean that any partially filled order would be at risk of being drained by the allowed but malicious integrator.

Note that the likelihood here depends on an integrator vault admin getting compromised, and its capability to introduce a custom logic able to leverage the reentrancy attack vector. Still, such attacks would be hard for the performer to detect before the fact, which could justify a Medium severity.

Recommendation: Consider adding the modifier `nonReentrant` to the functions:

- `BaseAccountAbstraction::execute()` and...
- `BaseAccountAbstraction::executeBatch()`.

Definitive Finance: Fixed in PR 1413 (commit `3fbd9929dce33a823a47776c2fa966f77d42dbea`).

Cantina Managed: Fixes verified. The functions

- `BaseAccountAbstraction::execute()` and...
- `BaseAccountAbstraction::executeBatch()`

are now using the modifier `nonReentrant`.

3.2 Low Risk

3.2.1 Temporary DOS of `pullFlashERC2612()` function is possible

Severity: Low Risk

Context: DefinitiveFlashAllowance.sol#L162-L170

Description: In `DefinitiveFlashAllowance.sol`, there are multiple ways to pull funds from the `order.swapper` (user who placed the order). One of these functions uses ERC-2612 permit logic to directly execute a permit signature fetched from user on the `inputToken` contract:

```
IERC20Permit(order.fromToken).permit(
    order.swapper,
    address(this),
    auth.value,
    auth.deadline,
    auth.v,
    auth.r,
    auth.s
);
```

The issue here is that anyone can just use the signature parameters, and front-run a legitimate transaction executing the function `flashSwap()` with the option `pullFlashERC2612()`, by directly executing the function `fromToken.permit()` on the token contract.

This will consume the permit signature and lead to a failure of the actual call to `flashSwap()` pushed on-chain by Definitive performer address.

Ethereum Mainnet is in scope, so such an attack is possible by an attacker, to temporarily delay flash swap execution.

Recommendation: Consider using a try-catch approach for this logic, such that even if the permit call fails due to a permit signature already being used, the logic in the function `pullFlashERC2612()` can continue the execution.

Definitive Finance: Fixed in PR 1399 (commit `8d1c1fd6c71e5b1503207475b4f7e7b4be74ce0d`).

Cantina Managed: Fixes verified. The function `pullFlashERC2612()` uses now a try-catch block.

3.2.2 Inconsistent usage of `_fetchFeeAccount()` for the management of input fees

Severity: Low Risk

Context: BaseFeesInitiable.sol#L57-L63, BaseFlashSwap.sol#L82-L86, BaseSimpleSwapInitiable.sol#L64-L68, BaseTransfersNativeInitiable.sol#L33-L38

Description: In the functions `BaseTransfersNativeInitiable::withdrawToV2()` and `BaseFlashSwap::_executeFlashSwap()`, the code uses the function `_fetchFeeAccount()` for the `inputFees`.

However, it is not the case in the function `BaseSimpleSwapInitiable::swapV2()` that directly uses `FEE_ACCOUNT()` instead of `_fetchFeeAccount()`. As a result, that flow will not treat differently the case when the input token is the native asset and the value of `msg.sender` is not the `Entrypoint`;

Recommendation: Consider aligning the behavior of these three functions.

Definitive Finance: Addressed in PR 1402 (commit `0c7d7c22d297486b08cd632b5bc06344227a6439`).

Cantina Managed: Fixes verified. The behavior of the function `swapV2()` regarding the fee account is now aligned with what is done in the functions `withdrawToV2()` and `_executeFlashSwap()`.

3.2.3 Function `isValidSignature()` not gated by modifier `stopGuarded`

Severity: Low Risk

Context: BaseRecoverSignerInitiable.sol#L28-L52

Description: Deployed trading vaults externally expose the functions defined in `BaseRecoverSignerInitiable`. The file `docs/eip-1271.md` explicitly describes the function `isValidSignature()` as the on-chain authorization gate for CoW/Permit2/EIP-3009-style integrations.

Also, a hard-stop for fund-moving actions exists, and is enforced through the modifier `stopGuarded`, which reverts when either the global `STOP_GUARDIAN` key is disabled, or the local stop boolean is set. Every direct value-moving vault entry-point (`flashSwap`, `withdraw*`, `validateUserOp`, `execute`, `executeBatch`) uses that modifier. However, the vault's ERC-1271 authorization path does not, which makes the function `BaseRecoverSignerInitiable::isValidSignature()` externally callable without a check by the modifier `stopGuarded`.

That leaves an economically equivalent fund-moving path triggered by an external actor with active valid allowance live under halt. A halted vault therefore continues returning the ERC-1271 magic value (`0x1626ba7e`) for those external settlement requests, even while every other vault method is temporarily stopped.

Recommendation: Consider adding the modifier `stopGuarded` to the function `isValidSignature()`.

Definitive Finance: Fixed in PR 1406 (commit `40fb1c88152377107f37d63ffa4e70bafb187af4`).

Cantina Managed: Fixes verified. The function `isValidSignature()` is now using the modifier `stopGuarded`.

3.2.4 Full consumption of `inputAmount` not checked in delegate swap mode

Severity: Low Risk

Context: CoreSimpleSwapInitiable.sol#L93-L125, CoreSwapHandlerV1.sol#L130-L157

Description: The function `flashSwap()` pulls `pullAmount` into the vault, but the swap path never verifies that `payload.amount` was actually consumed.

In particular, part of the delegate swap mode, the vault allows an external spender from the vault context, but the vault does not verify if `inputAmount` has been fully consumed. As a result, any route under-consuming the input token leaves:

1. A persistent allowance usable later by the external spender via ERC-20 `transferFrom` without calling the vault, hence bypassing the checks `stopGuarded`, `tradingEnabled`, and `onlyDefinitive`;
2. These user-approved funds, becoming indistinguishable from integrator fees, accumulated in the vault, and that can be withdrawn by a Client Admin;

The root cause is that the vault only verifies output-side deltas and does not require full input consumption, so it does not reset any potential unspent allowances.

Note that the likelihood of that scenario is limited for two reasons:

1. It requires swaps to under-consume the input amount;
2. The Performer is trusted, and expected to only call trusted swap routers;

Recommendation: Consider a mechanism in the vault that will verify the full consumption of the input amount, and react with the following options: revert, reset the open allowance or isolate in the vault what should be considered as unspent user funds or integrator fees.

Definitive Finance: Fixed in PRs 1407 (commit `1f54c7cf69130360f2dd3bd9f9b5d802099a85eb`), 1415 (`13691897896691ab4bd3e1fe0873edd839e44e19`) and 1417 (`97c6e15b0e2ff8b0145610e6f0fc85acdadc1796`).

Cantina Managed: Fixes verified. The amount of input token consumed by the swap is now verified in the function `_executeFlashSwap()`. Any unspent amount is sent back to the owner of the funds. Any potential open allowance remaining after the swap when not all input amount has been consumed is manually reset to `0` using the function `_tryResetAllowanceToZero()` with best effort, silently ignoring if resetting the approval reverts.

3.2.5 Full consumption of `inputAmount` not checked in non-delegate swap mode

Severity: Low Risk

Context: CoreSwapHandlerV1.sol#L22-L24

Description: After performing a swap in non-delegate mode, the function `swapDelegate()` lets anyone sweep any unspent input owned by `MetaSwapHandlerV2`.

In the non-delegate swap mode, the function `CoreSwapHandlerV1.swapCall()` transfers the full declared input amount to the handler before the downstream route runs, but no check verifies that the route actually consumes that input or refunds any unused remainder. It means that in that mode, any unused portion of the user's pulled tokens will remain owned by the contract `MetaSwapHandlerV2`. After that, any external account can directly call the handler's unrestricted function `swapDelegate()` to trigger a transfer of the handler's balance to that external caller.

This breaks the flash-swap safety envelope, since user-approved funds that were supposed to be atomically swapped can be left in a public contract, and taken by arbitrary third parties.

Here is an example illustrating that scenario:

1. An authorized performer executes the function `flashSwap()` with `payload.isDelegate == false`, so the vault goes through the handler's non-delegate swap mode path.

2. The function `_prepareAssetsForNonDelegateHandlerCall()` approves the handler for the full `payload.amount`, and `CoreSwapHandlerV1._swapCall()` immediately pulls that amount from the vault.
3. After the handler execution, the function only checks the output-token delta, and only reports the observed input-token delta back to the vault. There is no verification that `params.inputAmount` was fully consumed, and no refund of any remainder to the vault, or user.
4. If the route spends less than the declared maximum input, which could happen for multiple reasons, the unused input remains owned by the contract `MetaSwapHandlerV2`.
5. Any external address can then directly call the function `MetaSwapHandlerV2.swapDelegate()`. That path has no access control and only requires `inputAmount != 0`. When called directly, the function `MetaSwapHandlerV2._performSwap()` executes the arbitrary `calldata` using the handler's own balances.
6. The external address calling can set the value of `underlyingSwapRouterAddress` to the stranded token contract and `swapData` to `transfer(caller, leftover)`. The handler will transfer these unused input tokens to the caller, and the call will not revert because the value of `params.minOutputAmount` can be set to `0`.

Recommendation: Consider introducing a mechanism at handler's level that automatically reacts if any input amount is still owned by the handler after a non-delegate swap call.

Definitive Finance: Fixed in PR 1408 (commit `9b804d82c9bf7fe35b25a8edbaba3df470f087c6`). We've removed the non-delegated swapCall functionality as it was deprecated functionality.

Cantina Managed: Fixes verified. The contract `CoreSwapHandlerV1` now only supports the delegate swap mode. The functions part of the non-delegate swap mode were removed, and `SwapPayload` objects with `isDelegate == false` are now rejected by the function `_processSwap()`.

3.2.6 Client Admin can pull tokens through UserOp

Severity: Low Risk

Context: BaseAccountAbstraction.sol#L67-L80

Description: A valid performer is expected and allowed to submit bundles of UserOps. In that case, any EVM call from that bundle will have `tx.origin == performer`.

The contract `DefinitiveFlashAllowance` is expected to prevent any unexpected token pulls that would not be used to perform a flash swap, and does so with:

1. A modifier `onlyPerformer` allowing calls only when `tx.origin == performer`;
2. A verification that the value of `msg.sender` matches the value of the field `order.vault` in the order object (`FlashOrder` or `QuickOrder`) signed by the owner of the funds;

Still, it is possible for a Client Admin (owner of the integrator vault) to request the contract `DefinitiveFlashAllowance` to pull tokens directly, without using it for a swap, by submitting a malicious `UserOp`. Here are the steps:

1. Victim address Alice signs a valid `flashOrder` of 10 WETH through the malicious integrator vault of Bob;
2. Valid performer submits a flash swap for a partial fill of 1 WETH using the function `pullFlashAllowance()`. An open allowance of 9 WETH remains open.
3. Bob sees the valid order above with its associated signature, and submits a `UserOp` that calls `BaseAccountAbstraction::executeBatch()` with the batch of calls: `[Call_1 - DefinitiveAllowance::pullFlashAllowance(order: orderOfAlice, signature: signatureOfAlice, pullAmount: 9 WETH), Call_2 - WETH:transfer(Bob, 9 WETH)]`.

That scenario is possible, for the following reasons:

- `Call_1` is allowed because the function `BaseAccountAbstraction::validateUserOp()` allows any method to be executed by Client Admins.
- `Call_2` transfers the pulled amount outside of the integrator's vault.
- If executed within a bundler submitted by a legit performer as `tx.origin`, the modifier `onlyPerformer` will pass in the function `pullFlashAllowance()`. In that case, `msg.sender` will be Bob's vault, which matches what was allowed in Alice's order.
- And what was not yet swapped for Alice's order is transferred to Bob's vault.

As impact, it would mean that any partially filled order would be at risk of being drained by the allowed but malicious integrator.

Note that the likelihood of that scenario depends on the off-chain controls performed by the performer when filtering UserOps that are added to the bundles.

Recommendation: Consider making sure that a performer rejects these particular UserOps when building bundlers.

Definitive Finance: Acknowledged with comment: We will never be processing arbitrary client actions via performers. We have dedicated non-performer relayers to do these.

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 Function `_executeFlashSwap()` returns an output amount before fees

Severity: Informational

Context: BaseFlashSwap.sol#L90-L127

Description: The function `_executeFlashSwap()` currently returns the output amount post swap (`outputAmount`). However, that value does not exactly reflect what is sent to the recipient, as output fees can be deducted from `outputAmount` before what remains (`recipientAmount`) is transferred to the recipient.

Recommendation: Consider assessing whether the function `_executeFlashSwap()` should return the value of `outputAmount` or `recipientAmount`, and update the code accordingly.

Definitive Finance: Acknowledged with comment: We intentionally view all values from the perspective of the trading vault, not what the end actor actually receives.

Cantina Managed: Acknowledged.

3.3.2 Consolidated suggestions to improve the documentation

Severity: Informational

Context: BaseAccessControlInitiable.sol#L13-L24, DefinitiveFlashAllowance.sol#L28, DefinitiveFlashAllowance.sol#L56

Description: Here are multiple items where what the documentation says could be clarified:

1. In `DefinitiveFlashAllowance.sol`, an inline comment could explain that the variable `FlashAllowanceStorage.signatureBypassAllowed.sol` is only used for off-chain simulations, so that successful simulations for swaps can happen without needing to collect signatures beforehand.
2. In `BaseAccessControlInitiable.sol`, a comment enumerates *"Modifiers inherited from CoreAccessControl"*. The modifier `onlyAdmins` is not in the list.
3. In `DefinitiveFlashAllowance`, the NatSpec says "*This contract is used to allow a user to pull funds from a user account for a flash swap*", which is misleading because only a performer address is allowed to originate flash swaps and only the vault address can pull funds.

Recommendation: Consider clarifying all the items above.

Definitive Finance: Fixed in PRs 1395 (commit `5aa3069233a3426e1fe52aef7832ae09fb1bc3b4`), 1397 (commit `f2b5b37e5135a38612cb90ecf4ee7af20c949cf6`) and 1401 (commit `23b2bfd512b0525b77c649cbc0c607a65cbe3151`).

Cantina Managed: Fixes verified. Documentation updated as suggested.

3.3.3 No event emitted when the allowance contract is updated in `GlobalGuardian`

Severity: Informational

Context: GlobalGuardian.sol#L233-L237

Description: The function `setAllowanceContract()` defined in the contract `GlobalGuardian` lets the owner of the contract update the active allowance contract. However, there is no event emission when it happens, so off-chain observers cannot easily monitor these updates to react accordingly.

Recommendation: Consider emitting an event when the function `setAllowanceContract()` is successfully executed.

Definitive Finance: Function removed in PR 1403 (commit `2fda95575ff4b6a12f7ed1db897255c182e7cecc`).

Cantina Managed: Fixes verified.

3.3.4 Function `validateSwapPayloadSigner()` declared but not used

Severity: Informational

Context: ICoreSimpleSwapV1.sol#L4-L12, GlobalGuardian.sol#L239-L260

Description: The function `validateSwapPayloadSigner()` is declared but not used in the codebase. It is also the case for the field `SwapPayload.signature`. As a result, the signature of the `SwapPayload` objects submitted to the system to perform swaps is currently not verified on-chain.

Recommendation: Consider assessing if the function `validateSwapPayloadSigner()` should be used on-chain when swaps are processed, or documenting the fact that this is currently verified off-chain.

Definitive Finance: Acknowledged with comment: This is something not yet supported in trading vaults and we will add in a future contract update.

Cantina Managed: Acknowledged.

3.3.5 Ownership of `DefinitiveFlashAllowance` contract can be renounced

Severity: Informational

Context: DefinitiveFlashAllowance.sol#L18

Description: The `DefinitiveFlashAllowance` contract inherits from OpenZeppelin's `OwnableUpgradeable` library.

This library has a function called `renounceOwnership()` that allows the current contract owner to give up contract ownership permanently.

If this function is called accidentally, it could lead to permanent loss of admin privileges over this contract, and the contract will never be upgradeable as `_authorizeUpgrade()` is protected by the `onlyOwner` modifier.

Also, the current setup lacks a two-step ownership transfer, which is recommended over a single-step process.

Recommendation: Consider using OpenZeppelin's `Ownable2StepUpgradeable` library and overriding the function `renounceOwnership()` to always revert.

Definitive Finance: Fixed in PR 1398 (commit `c2a4b978482ca4880251f6345b248291ae6ce0de`).

Cantina Managed: Fixes verified. The contract imports now the contract `Ownable2StepUpgradeable`, and the function `renounceOwnership()` always reverts.

3.3.6 Test suite does not cover flash swaps where the output token is the native token

Severity: Informational

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

Description: The system support flash swaps with the `outputToken` being the native token. However, the test suite does not cover that case, even if it involves a specific logic. As a result, the test suite may not catch potential breaking changes in the future impacting flash swaps with the `outputToken` being the native token.

Recommendation: Consider adding that scenario to the test suite.

Definitive Finance: Fixed in PR 1404 (commit `9c90071224b62a293344c0e2e18682797862226b`).

Cantina Managed: Fixes verified. Four test cases with native assets as output token were added to the test suite.