SOLIDIFIED

Audit Report for Aperture Finance - September 24, 2022

Summary

Audit Report prepared by Solidified covering the Aperture Finance smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on September 1, 2022, and the results are presented here.

Audited Files

The source code has been supplied in a private source code repository:

https://github.com/Aperture-Finance/Aperture-Contracts

Commit number: `3c5c7758e7a58b3b8bdcad89ef1a27be8534ed4f`

Update: Fixes were received on September 12, 2022.
Updated commit number: `9f703214e4e5a522e25d1667fcbaf1368e2bea47`

Intended Behavior

Aperture Finance is a cross-chain investment platform. The repository implements a delta-neutral strategy on top of Homora V2.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: code complexity, code readability, level of documentation, and test coverage.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	Medium	-

Issues Found

Solidified found that the Aperture Finance contracts contain no critical issues, 6 major issues, 10 minor issues, and 10 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	HomoraPDNVault.sol: Function initializeConfig() can be called multiple times	Major	Acknowledged
2	HomoraPDNVault.sol: withdrawFee should not be changeable after users have deposited funds	Major	Acknowledged
3	ApertureManager.sol: Increasing a position will fail due to missing the appropriate token spending allowance	Major	Resolved
4	ApertureManager.sol: updateApertureManager() can leave an active chainId with no manager	Major	Acknowledged
5	ApertureManager.sol: a strategy with active positions can be removed	Major	Resolved
6	ApertureManager.sol: Failed cross-chain transactions leave bridged assets unrecoverable	Major	Acknowledged
7	ApertureManager.sol: Function disburseAssets() can potentially fail when transferring ETH to a smart contract	Minor	Resolved
8	ApertureManager.sol: crossChainFeeContext.feeSink could be set to the zero-address	Minor	Resolved
9	HomoraPDNVault.sol: feeCollector could be set to a zero-address	Minor	Resolved
10	ApertureManager.sol: crossChainFeeContext.feeBps could be	Minor	Resolved

	initialized to a value more than MAX_FEE_BPS		
11	VaultLib.sol: getOffset() calculation is asymmetric	Minor	Resolved
12	libraries/CurveRouterLib.sol: Swap routes utilizing older Curve pools will break swaps	Minor	Acknowledged
13	HomoraPDNVault.sol: The owner can pause withdrawals by limiting the maximum withdrawal amount per transaction	Minor	Resolved
14	ApertureManager.sol: The ApertureManager.swapTokenAndExecuteStrategy() function does not validate the swapped asset	Minor	Resolved
15	Silently failed ERC-20 token transfers can lead to incorrect accounting balances	Minor	Resolved
16	ApertureManager.sol: Adding strategies with the same strategy manager address will prevent other strategies from disbursing assets when removing said strategy	Minor	Resolved
17	ApertureManager.sol: Expensive storage access in the createPositionInternal() function	Note	Resolved
18	ApertureManager.sol: Function validateAndTransferAssetFromSender() declares assetInfos as memory instead of calldata	Note	Acknowledged
19	ApertureManager.sol: addStrategy() lacks a zero-address check on _strategyManager	Note	Resolved
20	HomoraPDNVault.sol: apertureManager can be set to zero-address	Note	Resolved
21	HomoraPDNVault.sol: depositInternal() does not carry msg.value	Note	Acknowledged
22	HomoraPDNVault.sol: Restrict native token funds receivable by contract	Note	Acknowledged
23	VaultLib.sol: managementFee collected during a	Note	Resolved

	leap year might be higher than anticipated		
24	Receive functions do not emit events	Note	Acknowledged
25	Use of floating pragma	Note	Acknowledged
26	Miscellaneous	Note	Resolved

Critical Issues

No critical issues have been found.

Major Issues

1. HomoraPDNVault.sol: Function initializeConfig() can be called multiple times

Calling function `initializeConfig()` by the owner after users have deposited funds to the protocol can potentially lead to user loss of funds. For instance, if the owner changes the value of `deltaThreshold` after users have deposited funds, `apertureManager` can potentially be prevented from calling `withdrawInternal()` due to `isDeltaNeutral()` potentially no longer returning `true`.

Recommendation

Enforce that `initializeConfig()` can only be called once, or preferably have it as an `internal` function that is called from within `initialize()`. Alternatively, create a time interval that allows protocol participants ample time to withdraw their deposits before the newly set config values are enforced.

Note

The same discussion also applies to the `setConfig()` function.

Status

Acknowledged. Team's response: "*initializeConfig() sets up important operational parameters while initialize() defines critical vault-related addresses. If isDeltaNeutral() in withdrawInternal() returns false, rebalance() will be triggered, which will bring delta back to neutral*".

2. HomoraPDNVault.sol: withdrawFee should not be changeable after users have deposited funds

---

The protocol's `withdrawFee` is in a sense a contract between the users and the protocol that they accept before depositing their funds. Hence, its value should not change after users already have their tokens locked in the contract. Additionally, an incorrect assignment (or a malicious owner) can potentially set this value so high that users are entirely prevented from withdrawing their funds.

Recommendation

Either entirely prevent changing `withdrawFee` in the function `setFees()`, or make sure there are no depositors left before allowing the change.

Status

Acknowledged. Team's response: "*withdrawFee is 0 initially and may increase in the future, which will be controlled by a time-lock mechanism*".

3. ApertureManager.sol: Increasing a position will fail due to missing the appropriate token spending allowance

---

In `ApertureManager`, the function `executeStrategy()` calls `validateAndTransferAssetFromSender()`, to check the tokens are whitelisted and transfer them to the contract, and subsequently calls `executeStrategyInternal()`. If the decoded action is `Action.Increase`, `increasePosition()` is called, attempting to transfer

`stableTokenDepositAmount` and `assetTokenDepositAmount` from `ApertureManager` to the `HomoraPDNVault` contract. This will fail, as the tokens have not been approved from `ApertureManager` to be transferred out.

Recommendation

Ensure approval is given to the strategy manager to transfer assets out of the contract, similarly to the approval in `createPositionInternal()`.

Status

Resolved

4. ApertureManager.sol: updateApertureManager() can leave an active chainId with no manager

---

In `ApertureManager`, the function `updateApertureManager` can be called with a zero-address passed as the `managerAddress`. This will cause `CrossChainLib.sol.publishExecuteStrategyInsturction()` to revert in `sendTokensCrossChainAndConstructCommonPayload()` making it impossible to decrease / close existing positions via the `WormholeCoreBridge`.

Recommendation

Consider keeping track of chainIds with active strategies and only allow updating of `managerAddress` to a non-zero value for those chains.

Status

Acknowledged. Team's response: "*The ability to map an active chainId to the zero address is by design. We intend for the owner to be able to "disable" a certain chainId when the need arises, e.g. in response to an ongoing network-wide security incidence*".

5. ApertureManager.sol: a strategy with active positions can be removed

In `ApertureManager`, the function `removeStrategy()` allows the `ApertureManager` to remove a strategy that has active positions. This will make subsequent calls from `ApertureManage` to execute on that strategy (i.e. to `closePosition()`) revert, as the `strategyManager` will no longer be whitelisted in `allowedToDisburseAssets`, and thus impossible to `disburseAssets()`.

Recommendation

Ensure there are no active positions before removing a strategy.

Status

Resolved

6. ApertureManager.sol: Failed cross-chain transactions leave bridged assets unrecoverable

Aperture operates across multiple blockchains and allows users to facilitate cross-chain transactions using the Wormhole protocol. Assets are bridged from the source chain to the destination chain, followed by a generic message with an appropriate instruction to the `ApertureManager` contract on the target chain. However, failed transactions on the destination chain will render the bridged tokens unrecoverable.

Recommendation

We recommend adding functionality for manual, owner-callable token reimbursements via the `ApertureManager` contract.

Status

Acknowledged. Team's response: "*The team are aware of this and plan to introduce a refund mechanism that would send back the tokens to the initiator on the original chain before launching the cross-chain module for public use*".

Minor Issues

7. ApertureManager.sol: Function disburseAssets() can potentially fail when transferring ETH to a smart contract

The function `disburseAssets()` calls `transfer()` when sending ETH, which only forwards 2300 gas. In cases where the recipient address is a smart contract whose fallback function consumes more than 2300 gas, the call will always fail. This will have the side effect of potentially preventing smart contracts (e.g. DAOs) from receiving transfers.

For a more in-depth discussion of issues with `transfer()` and smart contracts, please refer to `https://diligence.consensys.net/blog/2019/09/stop-using-soliditys-transfer-now/`

Recommendation

Replace instances of `transfer()` with `call()`.

Note

The same issue also exists in `CrossChainLib.processMultiTokenDisbursementInstruction()`.

Status

Resolved

8. ApertureManager.sol: crossChainFeeContext.feeSink could be set to the zero-address

---

The `feeSink` address supplied to the initializer of `ApertureManager` is not validated, allowing for a zero-address `feeSink`. Whilst this might be fixed by calling `updateCrossChainFeeContext` at a later stage, if unnoticed, it might result in permanent loss of funds.

Recommendation

Consider adding a zero-address check for `feeSink` in the `initialize` function.

Status

Resolved

9. HomoraPDNVault.sol: feeCollector could be set to a zero-address

---

The `_feeCollector` address supplied to the initializer of `HomoraPDNVault` is not validated, allowing for a zero-address `feeCollector`. Also, the function `setFeeCollector()` responsible for updating the `feeCollector` does not check against a zero-address assignment. If unnoticed, this might result in permanent loss of funds.

Recommendation

Consider adding a zero-address check for `feeCollector` in the `initializer()` and `setFeeCollector()`.

Status

Resolved

10. ApertureManager.sol: crossChainFeeContext.feeBps could be initialized to a value more than MAX_FEE_BPS

The `feeBps` value supplied to the `initialize()` function of `ApertureManager` is not validated, allowing for a value more than the `MAX_FEE_BPS`. Whilst this might be fixed by calling `updateCrossChainFeeContext()` at a later stage, if unnoticed, it might result in higher than intended fees.

Recommendation

Consider ensuring that `feeBps` is less than or equal to `MAX_FEE_BPS` in the initializer.

Status

Resolved

11. VaultLib.sol: getOffset() calculation is asymmetric

When `getOffset()` is used in the context of `isDeltaNeutral()`, `pos.debtAmtB` is set as the target. This will return a false value for `isDeltaNeutral()` in the case where `pos.debtAmtB` is `deltaThreshold` % less than `pos.amtB` but will return true in the case where `pos.amtB` is `deltaThreshold` % less than `pos.debtAmtB`, which might not be intended.

Recommendation

In the case of slippage calculation in `rebalanceShort()` and `rebalanceLong()` this is intended, therefore a different pure function can be implemented that takes the average instead of the target as the denominator.

Status

Resolved

12. libraries/CurveRouterLib.sol: Swap routes utilizing older Curve pools will break swaps

The `CurveRouterLib.swapToken` function uses pre-configured Curve swap routes to swap between various ERC-20 tokens. A swap route can be composed of one or more token swaps utilizing multiple Curve pools. The swap function iterates over all route paths and calls either `ICurve(pool).exchange_underlying(..)` or `ICurve(pool).exchange(..)`. The returned value `amount` is then passed on to the subsequent swap. However, older Curve pools may not return the number of tokens received after swapping. For those pools the return value will be `0` which will fail the `amount >= minAmountOut` invariant for `minAmountOut > 0`. Using such a swap path will revert and prevent swapping tokens.

For more details, please see https://curve.readthedocs.io/exchange-pools.html#id8

Recommendation

We recommend using the token balance before and after a swap to determine the received token amount.

Status

Acknowledged. Team's response: "*We appreciate the finding and the suggestion, and have documented this in https://github.com/Aperture-Finance/Aperture-Contracts/commit/620e76528a61d25a449940fa9619282eb751cebc. We decided not to make changes at this time as we don't foresee adding routes involving older Curve pools*".

13. HomoraPDNVault.sol: The owner can pause withdrawals by limiting the maximum withdrawal amount per transaction

---

The `HomoraPDNVault` contract enforces certain limits for a vault. One of these vault limits is `maxWithdrawPerTx` - the maximum amount allowed to withdraw in a single transaction. The owner of the `HomoraPDNVault` contract can configure those vault limits by calling the `HomoraPDNVault.setVaultLimits` function. Setting `maxWithdrawPerTx` to a value of `0` will disable withdrawals from an active position.

Recommendation

We recommend using a reasonable lower bound for `vaultLimits.maxWithdrawPerTx`.

Status

Resolved

14. ApertureManager.sol: The ApertureManager.swapTokenAndExecuteStrategy() function does not validate the swapped asset

---

For convenience, a user can use the `ApertureManager.swapTokenAndExecuteStrategy()` function to swap tokens and add liquidity to an existing position. However, the received token address `toToken` is currently not validated and checked for being whitelisted for usage with a specific strategy. Using any other tokens than the ones allowed by a strategy will revert the transaction within the `HomoraPDNVault.increasePositionInternal()` function.

Recommendation

Use the `isTokenWhitelistedForStrategy` mapping to check if the specified `toToken` is currently allowed.

Status

Resolved

15. Silently failed ERC-20 token transfers can lead to incorrect accounting balances

---

ERC20 tokens do not necessarily fully adhere to the ERC20 standard. Some tokens return `false` instead of reverting on failed token transfers. It is considered best practice to either add a `require()` statement that checks the return value of ERC20 token transfers or to use OpenZeppelin's `safeTransfer()`. Failure to do so can cause silent failures of transfers and affect token accounting.

Findings:

- `HomoraPDNVault.sol#L538`
- `HomoraPDNVault.sol#L539`
- `HomoraPDNVault.sol#L546`
- `HomoraPDNVault.sol#L550`

Recommendation

We recommend using `ERC20.safeTransfer()` consistently instead of `ERC20.transfer()` to prevent silent token transfer failures.

Status

Resolved

16. ApertureManager.sol: Adding strategies with the same strategy manager address will prevent other strategies from disbursing assets when removing said strategy

---

The owner of the `ApertureManager` contract can add and remove strategies. Strategies with the same `_strategyManager` address can be added. Once the owner removes one of the strategies with the reused `_strategyManager` address by calling the `removeStrategy` function, the `_strategyManager` address is removed from the `allowedToDisburseAssets` allowlist, leaving other strategies with the same manager address unable to disburse assets.

Recommendation

We recommend adding a check to the `ApertureManager.addStrategy` function to prevent reusing a `_strategyManager` address.

Status

Resolved

Informational Notes

17. ApertureManager.sol: Expensive storage access in the createPositionInternal() function

The function `createPositionInternal()` declares the variable `strategy` as `storage`. Since reading from `storage` is significantly more expensive than `memory`, the function ends up consuming a lot more gas than it should.

Recommendation
Store `strategy.strategyManager` in a `memory` variable and use it instead of `strategy`.

Status
Resolved

18. ApertureManager.sol: The function validateAndTransferAssetFromSender() declares assetInfos as memory instead of calldata

Since parameters declared as `calldata` are directly accessed without being copied from `memory` first, the function `validateAndTransferAssetFromSender()` can potentially consume a lot more gas than it should by declaring `assetInfos` as `memory`.

Recommendation
Declare assetInfos as `calldata` instead of `memory`.

Note

This issue applies to all functions in the codebase that declare `assetInfos` as `memory`.

Status

Acknowledged. Team's response: "*Some callers of validateAndTransferAssetFromSender() constructs assetInfos in memory before passing it into the function, for example, swapTokenAndCreatePosition() , so the storage location needs to be memory instead of calldata*".

19. ApertureManager.sol: addStrategy() lacks a zero-address check on _strategyManager

---

In `ApertureManager`, the function `addStrategy()` does not check against the possibility of the `_strategyManager` argument being the zero-address. This will create an unusable strategy with a zero-address `strategyManager`.

Recommendation

Consider adding a zero-address check for `_strategyManager` in `addStrategy()`.

Status

Resolved

20. HomoraPDNVault.sol: apertureManager can be set to zero-address

---

The `HomoraPDNVault` instance will be deemed unusable if `apertureManager` is initialized to the zero-address in `HomoraPDNVault.sol: initializer()`.

Recommendation

Consider adding a zero-address check for the assignment of `apertureManager`.

Status
Resolved

21. HomoraPDNVault.sol: depositInternal() does not carry msg.value

---

`msg.value` from ApertureManager functions `createPosition()` and `executeStrategy()` is consumed in `validateAndTransferAssetFromSender()`, by calling `crossChainContext.wrapEtherAndExpandAssetInfos.` `wrapEtherAndExpandAssetInfos()` is responsible for wrapping `msg.value` to `WETH` and adding it to `assetInfo` to be processed if `WETH` is whitelisted for the strategy. Even though the `HomoraPDNVault` function `depositInternal()` passes `msg.value` to `VaultLib.sol`, `msg.value` is always `0`.

Recommendation

Consider calling `VaultLib.sol`: `deposit()` with `0` for value, or removing the argument value from `deposit()` completely. Subsequently, neither `addLiquidity()`'s `value` argument is needed and `homoraExecute` can be called with a 0 value.

Status
Acknowledged. Team's response: "*Although msg.value is always 0 beyond ApertureManager contract, it is retained for future upgradability*".

22. HomoraPDNVault.sol: Restrict native token funds receivable by contract

---

The `HomoraPDNVault` contract implements the `receive()` function to receive native token transfers. However, native token transfers to contracts should be, if possible, limited to a limited set of addresses to prevent accidental fund transfers from other sources.

Recommendation

We recommend modifying the `receive()` function to only accept transfers from expected addresses.

Status

Acknowledged. Team's response: "*We decided not to introduce a whitelist mechanism for receive() to reduce gas cost*".

23. `VaultLib.sol`: `managementFee` collected during a leap year might be higher than anticipated

---

In `VaultLib.sol`, the function `collectManagementFee()` divides the number of seconds between the time the function is called and the last time it has been called by `31536000`, which is the number of seconds in a Gregorian common year. However, a true Earth year is closer to `365.25` days of `86400` seconds each.

Recommendation

It is widely accepted in finance and asset management that the real value fee in common and leap years should be the same if the underlying `managementFee` and `vaultState.totalShareAmount` remains the same, thus consider using `31557600` as a denominator instead. `years` suffix has been depreciated since solidity `0.5.0` for the aforementioned reason.

Status

Resolved

24. Receive functions do not emit events

The `receive()` functions in `HomoraPDNVault.sol` and `HomoraAdapter.sol` do not emit events.

Recommendation

Consider logging events with the `msg.sender` for the aforementioned functions.

Status

Acknowledged. Team's response: "*We don't have plans to monitor receive() calls so we don't feel it's necessary to emit events*".

25. Use of floating pragma

The contracts allow solidity compiler versions `0.8.0` or higher. On the one hand, the floating pragma would allow for deployment with recent versions that the code might have not been tested with, and could contain yet to be discovered bugs. On the other hand, it allows for deployment with older compiler versions which are known to have critical bugs.

Recommendation

As a general advice, the compiler version that is used should be as old as possible to ensure it is tested and fit for production deployment, but as recent as necessary to ensure bug fixes have been introduced for any discovered vulnerabilities. Consider fixing the compiler version to `0.8.9`.

Status

Acknowledged. Team's response: "*We decided to keep the current compiler version pragma for flexibility*".

26. Miscellaneous

- Unnecessary imports at:
  - `VaultLib.sol: IERC20.sol`.
  - `HomoraPDNVault.sol: IERC20.sol`
  - `ApertureManager.sol: IERC20.sol, IWormhole.sol`
  - `CrossChainLib.sol: IERC20.sol`
  - `CurveRouterLib.sol: IERC20.sol`
  - `HomoraAdapterLib.sol: IERC20.sol`
- Comments in code:
  - Omission of `@param router` in `VaultLib.sol`: `deltaNeutralMath`
- Functions `HomoraPDNVault.sol`: `setConfig()`, `VaultLib.sol`: `convertCollateralToTokens()`, `getCollateralFactor()`, `getBorrowFactor()`, `CurveRouterLib.sol`: `updateRoute()` might revert without a meaningful error message.
- Spelling issues:

  - `HomoraPDNVault.sol#L165` - `valut` - Suggestion: `value`

  - `HomoraPDNVault.sol#L221` - `valut` - Suggestion: `value`

  - `HomoraPDNVault.sol#L814` - `Evalute` - Suggestion: `Evaluate`

  - `libraries/CrossChainLib.sol#L12` - `instructure` - Suggestion: `instruction`

  - `libraries/CrossChainLib.sol#L66` - `distingish` - Suggestion: `distinguish`

  - `libraries/VaultLib.sol#L172` - `Evalute` - Suggestion: `Evaluate`
- `ApertureManager.sol#L446`: The `INSTRUCTION_TYPE_SINGLE_TOKEN_DISBURSEMENT` instruction is currently unused. We recommend removing the code related to this instruction.

Status

Resolved

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Aperture Finance or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Oak Security GmbH