CANTINA

Kintsu

Security Review

Cantina Managed review by:

Desmond Ho, Lead Security Researcher
Cccz, Security Researcher

September 12, 2025

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	**Description**
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

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 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

Kintsu is a Composable Liquid Staking Protocol aiming to boost the GDP of DeFi by allowing users to participate in on-chain activities while also benefitting from the yield-bearing staking that secures the blockchain.

From Aug 6th to Aug 10th the Cantina team conducted a review of hyperliquid-contracts on commit hash 94c711fd. The team identified a total of **11** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	3	3	0
Medium Risk	3	3	0
Low Risk	2	2	0
Gas Optimizations	0	0	0
Informational	3	2	1
Total	**11**	**10**	**1**

3 Findings

3.1 High Risk

3.1.1 The amounts for bonding and unbonding in `submitBatch()` are incorrect

Severity: High Risk

Context: Vault.sol#L346-L360

Description: `submitBatch()` will bond or unbond Hype from nodes based on the requested deposits and withdrawals for each batch:

```
if (batchDepositRequest.assets > batchWithdrawRequest.assets) {
    // Net ingress
    uint96 bondedAmountEvm = _doBonding(batchDepositRequest.assets - batchWithdrawRequest.assets, _totalPooled);
    uint96 dust = batchDepositRequest.assets - bondedAmountEvm;
    if (dust > 0) {
        batchDepositRequests[_currentBatchId + 1].assets = dust;
    }
    batchSubmission.unbondingThaw = _safeIncrease(batchSubmission.unbondingThaw,
    ↪   _cooldowns.unbondingFreezeFromBonding);
} else if (batchWithdrawRequest.assets > batchDepositRequest.assets) {
    // Net egress
    if (block.timestamp <= batchSubmission.unbondingThaw) revert UnbondingFreeze();
    uint96 unbondedAmountEvm = _doUnbonding(batchWithdrawRequest.assets - batchDepositRequest.assets,
    ↪   _totalPooled);
```

The problem here is that it should bond or unbond the difference between deposits and withdrawals, rather than the whole deposit or the whole withdrawal. The impact is as follows:

1. Bonding too much amount, resulting in users being unable to withdraw funds.
2. Unbonding too much amount, resulting in funds being stuck in the Vault and not getting staked by nodes.

Recommendation: Consider using differences to bond or unbond:

```
  if (batchDepositRequest.assets > batchWithdrawRequest.assets) {
      // Net ingress
-     uint96 bondedAmountEvm = _doBonding(batchDepositRequest.assets, _totalPooled);
+     uint96 bondedAmountEvm = _doBonding(batchDepositRequest.assets - batchWithdrawRequest.assets,
↪   _totalPooled);
      uint96 dust = batchDepositRequest.assets - bondedAmountEvm;
      if (dust > 0) {
          batchDepositRequests[_currentBatchId + 1].assets = dust;
      }
      batchSubmission.unbondingThaw = _safeIncrease(batchSubmission.unbondingThaw,
      ↪   _cooldowns.unbondingFreezeFromBonding);
  } else if (batchWithdrawRequest.assets > batchDepositRequest.assets) {
      // Net egress
      if (block.timestamp <= batchSubmission.unbondingThaw) revert UnbondingFreeze();
-     uint96 unbondedAmountEvm = _doUnbonding(batchWithdrawRequest.assets, _totalPooled);
+     uint96 unbondedAmountEvm = _doUnbonding(batchWithdrawRequest.assets - batchDepositRequest.assets,
↪   _totalPooled);
```

Kintsu: Fixed in commit f22a9710.

Cantina Managed: Fix verified.

3.1.2 `requestUnlock()` does not charge `exitFee` to the caller

Severity: High Risk

Context: Vault.sol#L233-L267

Description: When a user calls `requestUnlock()` to request a withdrawal, no `exitFee` is charged to that user:

```
function requestUnlock(uint96 shares) external whenNotPaused {
    ExitFee memory _exitFee = exitFee; // shadow (SLOAD 1 slot)

    uint96 sharesToToll = uint96(uint256(shares) * _exitFee.bips / BIPS);
    uint96 sharesToUser = uint96(uint256(shares) - sharesToToll);
    if (sharesToUser == 0) revert NoChange();

    _updateFees();

    uint96 spotValue = convertToAssets(sharesToUser);
    if (spotValue == 0) revert MinimumUnlock();

    uint40 _currentBatchId = currentBatchId; // shadow

    // Update user's unlock requests
    userUnlockRequests[msg.sender].push(UnlockRequest({
        shares: sharesToUser,
        spotValue: spotValue,
        batchId: _currentBatchId
    }));

    // Update current batch
    Batch memory _batchWithdrawRequest = batchWithdrawRequests[_currentBatchId]; // shadow (SLOAD 1 slot)
    _batchWithdrawRequest.assets += spotValue;
    _batchWithdrawRequest.shares += sharesToUser;
    batchWithdrawRequests[_currentBatchId] = _batchWithdrawRequest;

    // Escrow toll
    if (sharesToToll > 0) {
        _exitFee.escrowShares += sharesToToll;
        exitFee = _exitFee;
    }

    // Escrow shares
    token.transferFrom(msg.sender, address(this), sharesToUser);
```

Instead, the exitFee is cached to `escrowShares` and then charged together with the management fee:

```
function mintProtocolShares(address to) external whenNotPaused onlyRole(ROLE_FEE_CLAIMER) {
    _updateFees();

    uint96 shares = managementFee.virtualSharesSnapshot + exitFee.escrowShares;
    if (shares == 0) revert NoChange();

    managementFee.virtualSharesSnapshot = 0;
    exitFee.escrowShares = 0;
    token.mint(to, shares);
```

This result in the `exitFee` not being charged to the user who requested the unstake, but the remaining users. Malicious users can repeatedly request unstake and then cancel unstake, thereby making other users suffer losses.

Recommendation: It is recommended to charge users who request to unstake, and to no longer mint `escrowShares`, but to send `escrowShares` to the fee recipient.

Kintsu: Fixed in commit eccc3a60.

Cantina Managed: Fix verified.

3.1.3 Dust is incorrectly calculated

Severity: High Risk

Context: Vault.sol#L350

Description: The dust calculation uses the full batch deposit / withdrawal asset amounts, but it should using the net amounts instead.

Proof of Concept:

```
function test_depositDustAmount() public setRoles setNodesAndWeights {
    vm.startPrank(user);
    vault.deposit{value: 1 ether}(user);
    vault.requestUnlock(1 ether - 1e10 + 1); // such that amountToBond is 0
    vault.submitBatch();

    // check dust for current batch, it's 1 ether when it should be 1e10 - 1
    (uint96 assets,) = vault.batchDepositRequests(1);
    assertEq(assets, 1 ether);
}
```

Recommendation:

```diff
- uint96 dust = batchDepositRequest.assets - bondedAmountEvm;
+ uint96 dust = batchDepositRequest.assets - batchWithdrawalRequest.assets - bondedAmountEvm;
```

likewise for the other side:

```diff
- uint96 dust = batchWithdrawRequest.assets - bondedAmountEvm;
+ uint96 dust = batchWithdrawRequest.assets - batchDepositRequest.assets - bondedAmountEvm;
```

Kintsu: Fixed in commit 95a6b111.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 Bonding and unbonding using incorrect argument to call `_getImbalances()`

Severity: Medium Risk

Context: Vault.sol#L541-L597

Description: `_getImbalances()` will calculate the imbalance based on the new total stake:

```solidity
function _getImbalances(Node[] memory nodes, uint96 newTotalPooled)
    private
    view
    returns (
        uint96 overAllocation,
        uint96 underAllocation,
        uint96[] memory overAllocations,
        uint96[] memory underAllocations
    )
{
    uint256 _totalWeight = Registry.totalWeight; // shadow

    uint256 len = nodes.length;
    overAllocations = new uint96[](len);
    underAllocations = new uint96[](len);

    for (uint256 i; i < len; ++i) {
        uint256 stakedAmountCurrent = nodes[i].staked;
        uint256 stakedAmountOptimal = _totalWeight > 0 ? uint256(nodes[i].weight) * uint256(newTotalPooled) /
        ↪  _totalWeight : 0;
```

But `_doBonding()` and `_doUnbonding()` used incorrect arguments to call it. Below `totalPooled` represents the total assets deposited by the user, and `totalStaked` represents the assets already staked to nodes, and `totalPooled == totalStaked + stakingReserveCore + batchDepositRequest.asset`.

1. For `_doBonding()`, consider `totalStaked == 10000`, `stakingReserveCore == 0`, `batchDepositRequest.assets == 2000`, `batchWithdrawRequest.assets == 0`, and `totalPooled == 12000`.

When `_doBonding()` calls `_getImbalances()`, it should pass `10000 + 2000` instead of `12000 + 2000`.

~~~
```solidity
(uint96 underAllocation,, uint96[] memory underAllocations,) = _getImbalances(_nodes, _totalPooled +
↪  amountToBond);
```
~~~

2. For `_doUnbonding()`, consider `totalStaked == 10000`, `stakingReserveCore == 0`, `batchDepositRequest.assets == 0`, `batchWithdrawRequest.assets == 2000`, and `totalPooled == 12000`.

   After fixing "The amounts for bonding and unbonding in `submitBatch()` are incorrect", when `_doUnbonding()` calls `_getImbalances()`, it should pass `10000 - 1000` instead of `12000 - 1000`.

   ```solidity
   (uint96 overAllocation,, uint96[] memory overAllocations,) = _getImbalances(_nodes, _totalPooled -
   ↪  amount);
   ```

Recommendation: When calling `_getImbalances()` on bonding and unbonding, replace `_totalPooled` in the arguments with `totalStaked`.

Kintsu: Fixed in commit 5ca04099.

Cantina Managed: Fix verified.

3.2.2 `syncStaking()` can be exploited for various purposes

Severity: Medium Risk

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

Description: Because this function is permissionless and can be called by anyone,

1. If there are pending rewards that haven't been synced, it would be advantageous to call this after `deposit()` to dilute the rewards, or before `requestUnlock()` to ensure their shares value increased.
2. Crucially, write operations via the `CoreWriter` are not atomic, so there will be a delay between bonding / unbonding delegations in `_doBonding()` / `_doUnbonding()` and its updated state being reflected on the node. As such, it is rather simple to write a contract that calls both `syncStaking()` and `submitBatch()` in a single transaction, causing the delegation deltas to be recognised as rewards / slashes instead.

Proof of Concept:

```
function test_syncAfterDepositBeforeWithdraw() public setRoles setNodesAndWeights {
    vm.startPrank(user);
    vault.deposit{value: 1 ether}(user);
    vault.submitBatch();

    // appreciate share price
    writer.addStakingReward{value: 0.01 ether}(address(vault));

    // may be advantageous to sync after calling deposit
    uint96 sharesReceived = vault.deposit{value: 1 ether}(user);
    uint96 sharesValue = vault.convertToAssets(sharesReceived);
    vault.syncStaking();
    uint96 newSharesValue = vault.convertToAssets(sharesReceived);
    assertGt(newSharesValue, sharesValue);

    // likewise, for withdraw, may be advantageous to sync before calling withdraw
    writer.addStakingReward{value: 0.01 ether}(address(vault));
    uint96 sharesToWithdraw = 0.1 ether;
    sharesValue = vault.convertToAssets(sharesToWithdraw);
    vault.syncStaking();
    newSharesValue = vault.convertToAssets(sharesToWithdraw);
    assertGt(newSharesValue, sharesValue);
}
```

Recommendation: Consider making `syncStaking()` permissioned.

Kintsu: Fixed in commit 2ff97b27.

Cantina Managed: Fix verified.

3.2.3 Vault is susceptible to share inflation attack

Severity: Medium Risk

Context: Vault.sol#L214

Description: The vault contains a common vulnerability affecting vault deposits whereby the first depositor can massively inflate the share price to cause subsequent depositors to receive zero/fewer shares than expected, thereby draining their deposits.

Proof of Concept:

```
function test_firstDepositorInflationAttack() public setRoles setNodesAndWeights {
    address attacker = makeAddr("attacker");
    vm.deal(attacker, 100 ether);

    vm.startPrank(attacker);
    vault.deposit{value: 0.01 ether}(attacker);
    sHype.approve(address(vault), type(uint256).max);
    vault.requestUnlock(0.01 ether - 1);
    vault.submitBatch();

    // attacker deposits a large amount to inflate sHype share price
    vault.contributeToPool{value: 90 ether}(attacker);

    // user deposits 1 ether, gets 0 shares
    vm.startPrank(user);
    uint96 actualShares = vault.deposit{value: 1 ether}(user);
    assertEq(actualShares, 0);
}
```

Recommendation: A common mitigation is to burn a small amount of the shares minted by the first depositor.

Kintsu: Fixed in commit f8d13750.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 Unbonding should withdraw from `stakingReserveCore`

Severity: Low Risk

Context: Vault.sol#L594-L600

Description: The assets deposited by users consist of three parts: `batchDepositRequests.assets`, `stakingReserveCore`, and `totalStaked`. When processing user withdrawal requests, `batchDepositRequests.assets` is first used to cover the request, followed by unbonding.

```
function _doUnbonding(uint96 amount, uint96 _totalPooled) private returns (uint96 unbondSummationEvm) {
    Node[] memory _nodes = Registry.nodes; // shadow (SLOAD 2n-nodes slots)
    uint256 n = _nodes.length;
    (uint96 overAllocation,, uint96[] memory overAllocations,) = _getImbalances(_nodes, _totalPooled - amount);

    // Amount to withdraw from over-allocated agents
    uint256 phase1 = amount < overAllocation ? amount : overAllocation;

    // Remaining amount to withdraw equitably from all agents
    uint256 phase2 = amount - phase1;
```

The problem here is that during unbonding, only `totalStaked` is considered, not `stakingReserveCore`. In the bad case, `totalStaked` may not be sufficient to cover the remaining withdrawal requests, leading to unbonding failure.

Recommendation: Consider withdrawing from `stakingReserveCore` during unbonding.

Kintsu: Fixed in commit 7e320932.

Cantina Managed: Fix verified.

3.3.2 Users are not refunded exit fees when cancelling withdrawing requests

Severity: Low Risk

Context: Vault.sol#L260-L264

Description: The refund amount upon cancelling a withdraw request is `sharesToUser`, which is after tolls, and is not refunded.

Proof of Concept:

```
function test_exitFee() public setRoles setNodesAndWeights setExitFee {
    vm.startPrank(user);
    vault.deposit{value: 1 ether}(user);
    uint256 userBalanceBefore = sHype.balanceOf(user);
    vault.requestUnlock(0.1 ether);
    // should have pulled the full share amount from the user
    uint256 userBalanceAfterRequest = sHype.balanceOf(user);
    assertEq(userBalanceAfterRequest, userBalanceBefore - 0.1 ether);
    // escrowed shares should have increased
    (uint40 bips, uint96 escrowedShares) = vault.exitFee();
    assertEq(escrowedShares, 0.1 ether * bips / 10_000);
    // as escrowed shares have already been minted, shouldn't affect mintableProtocolShares()
    assertEq(vault.mintableProtocolShares(), 0);

    // if user cancels, will not be refunded exit fee
    vault.cancelUnlockRequest(0);
    uint256 userBalanceAfterCancel = sHype.balanceOf(user);
    uint256 expectedRefundAmount = 0.1 ether - escrowedShares;
    assertEq(userBalanceAfterCancel, userBalanceAfterRequest + expectedRefundAmount);

    vm.startPrank(privileged);
    vault.mintProtocolShares(receiver);
    // should have transferred escrowed shares to receiver
    assertEq(sHype.balanceOf(receiver), escrowedShares);
    assertEq(vault.mintableProtocolShares(), 0);
    // escrowedShares should have been zeroed
    (, escrowedShares) = vault.exitFee();
    assertEq(escrowedShares, 0);
}
```

Recommendation: Consider charging exit fees on redemptions.

Kintsu: Fixed in commit 3f5c80d3.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 `contributeToPool()` does not appreciating shares

Severity: Informational

Context: Vault.sol#L480-L484

Description: `contributeToPool()` only increases `batchDepositRequests.assets` but does not increase `totalPooled`, which results in contributions not appreciating shares.

```
function contributeToPool(address benefactor) external payable {
    if (msg.value > type(uint96).max) revert DepositOverflow();
    batchDepositRequests[currentBatchId].assets += uint96(msg.value);
    emit Contribution(msg.value, benefactor);
}
```

Recommendation: Consider increasing `totalPooled` in `contributeToPool()`.

```
  function contributeToPool(address benefactor) external payable {
      if (msg.value > type(uint96).max) revert DepositOverflow();
+     totalPooled += uint96(msg.value);
      batchDepositRequests[currentBatchId].assets += uint96(msg.value);
      emit Contribution(msg.value, benefactor);
  }
```

Kintsu: Fixed in commit 3703748e.

Cantina Managed: Fix verified.

3.4.2 Share dilution causes collected fees to be less than expected

Severity: Informational

Context: Vault.sol#L707-L708

Description: Note that the fees collected in `HYPE` will be less than expected because of share dilution. E.g: `totalPooled == totalSupply = 10_000e18`, `managementFee.bips = 1%`. Assume no activity for 1 year, `totalPooled` has increased to `11_000e18` from staking rewards, an increase of 10%. Hence, we expect the protocol to be entitled to 1% of `11_000e18` = `110e18` HYPE.

The minted shares for the protocol is `10_000e18` * 1% = `100e18`, bringing the total share supply to `10_100e18`. When redeeming these shares, will be receiving `100e18 / 10_100e18 * 11_000e18` = `108.91e18` < `110e18` HYPE.

The formula to calculate the correct amount however requires tracking the profit, which seems difficult given the various moving parts + time weighted minting.

Recommendation: Since the affected party is the protocol, it is for informational purposes and is something to consider tweaking.

Kintsu: Acknowledged.

Cantina Managed: Acknowledged.

3.4.3 Minor Recommendations

Severity: Informational

Context: Registry.sol#L53, Registry.sol#L91, Vault.sol#L605

Description/Recommendation:

- Registry.sol#L53: It would be convenient to have an external function with the same functionality, as there is a use-case where one wants to fetch information about a single node.
- Registry.sol#L91: Increasing a deleted / non-existent node's weight will revert with `InvalidNode()`, not a no-op.
- Vault.sol#L605: `total_staked_after_phase1` should be in mixedCase:

```
- total_staked_after_phase1
+ totalStakedAfterPhase1
```

- The team made the following changes w.r.t cooldown variables for simplicity and greater clarity in commit 87fe056d.

Kintsu: Registry comment fixed in commit ccefd768; `total_staked_after_phase1` fixed in commit fe68b49c.

Cantina Managed: Fix verified.