Neutrl Contracts

Security Review

Cantina Managed review by:

Anurag Jain, Security Researcher
Ladboy233, Security Researcher

July 17, 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

Neutrl is a market-neutral synthetic dollar designed to unlock untapped yield opportunities in OTC and alt-coin markets. Neutrl leverages OTC arbitrage, funding rate inefficiencies, and DeFi-native market-neutral strategies to provide a single, high-yield access point for capital allocators.

From Jul 8th to Jul 15th the Cantina team conducted a review of neutrl-contracts on commit hash 50be2a80. The team identified a total of **8** issues:

Issues Found

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

3 Findings

3.1 Medium Risk

3.1.1 Fully restricted user can withdraw funds

Severity: Medium Risk

Context: sNUSD.sol#L241-L255

Summary: `FULL_RESTRICTED_STAKER_ROLE` role users are not allowed to withdraw funds normally. However it was observed that if a user was blacklisted post `cooldownAssets/cooldownShares` call then they can withdraw funds even though they are blacklisted.

Finding Description:

1. User `A` stakes amount `100` NUSD and say gets `100` sNUSD.
2. After sometime User wants to withdraw funds so User calls `cooldownAssets` with amount `100` NUSD.
3. Admin is suspicious of User `A` and blacklists by assigning `FULL_RESTRICTED_STAKER_ROLE` role to User `A`.
4. Post cooldown period, User `A` calls `unstake`.
5. Since `unstake` does not check for blacklisting status of caller, hence funds get withdrawn to User `A` chosen recipient, bypassing the blacklisted role.

Impact Explanation: `FULL_RESTRICTED_STAKER_ROLE` can bypass restriction and withdraw funds using `unstake`.

Proof of Concept: Place test at `test\unit\concrete\sNUSD\erc4626\sNUSD_erc4626.t.sol`.

```
function test_POC() external whenCooldownDurationIsNotZero {
    // Setup cooldown
    deal(address(sNusd), normalUser, testAmount);
    vm.startPrank(normalUser);
    nusd.approve(address(sNusd), testAmount);
    sNusd.deposit(testAmount, normalUser);
    sNusd.cooldownAssets(testAmount);
    vm.stopPrank();

    vm.startPrank(users.admin);
    sNusd.grantRole(FULL_RESTRICTED_STAKER_ROLE, normalUser);
    vm.stopPrank();

    // Fast forward time to end of cooldown
    uint24 cooldownDuration = sNusd.cooldownDuration();
    vm.warp(block.timestamp + cooldownDuration + 1);

    // Now unstake
    vm.startPrank(normalUser);
    uint256 initialBalance = nusd.balanceOf(normalUser);
    sNusd.unstake(normalUser);
    uint256 finalBalance = nusd.balanceOf(normalUser);

    // Verify balance increased and cooldown was reset
    assertTrue(finalBalance > initialBalance, "NUSD balance should increase after unstaking");

    // Check cooldown was reset
    (uint104 cooldownEnd, uint152 underlyingAmount) = sNusd.cooldowns(normalUser);
    assertEq(cooldownEnd, 0, "Cooldown end should be reset to zero");
    assertEq(underlyingAmount, 0, "Underlying amount should be reset to zero");
    vm.stopPrank();
}
```

Recommendation: Add a check in `unstake` function to ensure that `msg.sender` is not having role `FULL_RESTRICTED_STAKER_ROLE`.

Neutrl: Fixed in PR 16.

Cantina Managed: Fix verified. Check is added in `unstake` to ensure caller is not full blacklisted.

3.2 Low Risk

3.2.1 Duplicate tokens allowed while adding yield token

Severity: Low Risk

Context: YieldDistributor.sol#L72-L85

Description: Admin can mistakenly add duplicate yield token which is currently accepted. At deletion only the first copy of token in array is deleted, leaving the duplicate one still active. This could allow unintended token to be considered for yield distribution.

Proof of Concept: Place the test at `yield_distribution.t.sol`. Test revert showing token is still active.

```
function test_DuplicateTokenPOC() external whenTheCallerIsDEFAULT_ADMIN_ROLE {

    yieldDistributor.addYieldToken(address(mockUSDC));
    yieldDistributor.addYieldToken(address(mockUSDC));

    // Duplicate token addition (start with index 2,3 as setup already use index 0,1)
    (address token, bool isActive) = yieldDistributor.yieldTokens(2);
    (address token2, bool isActive2) = yieldDistributor.yieldTokens(3);
    assertEq(token, address(mockUSDC));
    assertEq(token2, address(mockUSDC));

    yieldDistributor.removeYieldToken(address(mockUSDC));

    // Verify token is now inactive
    (token2, isActive2) = yieldDistributor.yieldTokens(3);

    assertEq(token2, address(mockUSDC));
    assertFalse(isActive);
}
```

Recommendation: Disallow addition of duplicate yield token.

Neutrl: Fixed in PR 11.

Cantina Managed: Fix verified. Duplicate tokens are rejected on addition now.

3.2.2 Missing status check allow replaying cancel request

Severity: Low Risk

Context: Router.sol#L208-L213, Router.sol#L229-L234

Description: It was observed that the Request status was not verified while cancelling order. This means even a `COMPLETED` or `CANCELLED` order can be re-cancelled. The replay of a cancelled order causes loss of funds, as everytime `order.collateralAmount` gets transferred to `order.benefactor`.

Proof of Concept:

1. Assume below order at Request id `1`.

   ```
   mintRequestStatus[1] = RequestStatus.PENDING
   order.collateralAmount = 1000
   order.benefactor = User A
   order.collateralAsset = CollA
   ```

2. Keeper calls `cancelMintRequest` to cancel this request id `1`.
3. This causes `1000` CollA to transferred to User A and status changed to `RequestStatus.CANCELLED`.
4. Lets say, Keeper accidentally again calls `cancelMintRequest` on request id `1`.
5. Since there is no status check, `cancelMintRequest` again causes `1000` CollA to transferred to User A and status changed to `RequestStatus.CANCELLED`.

Note:Similar issue is on `cancelRedemptionRequest`.

Recommendation: Add below check in `cancelMintRequest`,

```
if (mintRequestStatus[_requestId] != RequestStatus.PENDING) revert InvalidRequestStatus();
```

Add below check in `cancelRedemptionRequest`,

```
if (redemptionRequestStatus[_requestId] != RequestStatus.PENDING) revert InvalidRequestStatus();
```

Neutrl: Fixed in PR 12.

Cantina Managed: Fix verified. Cancelling order now ensures that status is PENDING.

3.2.3 `unlockAsset` contain reentrancy risk if ERC777 token is added as locked asset

Severity: Low Risk

Context: AssetLock.sol#L173-L195

Description: Once the lock expires, the user can withdraw the locked. However, the code updates the state and external transfer call and subject to reentrancy attack.

```
IERC20(_asset).safeTransfer(msg.sender, lock.amount);

delete userLocks[msg.sender][_asset];
totalLocked[_asset] -= lock.amount;
```

If the locked asset is an ERC777 token with a registered hook, a malicious actor can exploit the reentrancy opportunity by triggering the callback and repeatedly invoking the `unlockAsset` function, allowing them to withdraw the asset multiple times.

Recommendation:

```
delete userLocks[msg.sender][_asset];
totalLocked[_asset] -= lock.amount;

ERC20(_asset).safeTransfer(msg.sender, lock.amount);
```

Neutrl: Fixed in PR 13.

Cantina Managed: Fix verified.

3.3 Informational

3.3.1 Typos, Edges cases & Minor Recommendations

Severity: Informational

Context: DeployProtocol.s.sol#L157, AssetReserve.sol#L107, Router.sol#L85, Router.sol#L91, Router.sol#L246, Router.sol#L262, Router.sol#L281, Router.sol#L346-L371, sNUSD.sol#L125-L127, sNUSD.sol#L151, sNUSD.sol#L158

Description / Recommendation:

Typos:

- Router.sol#L85:

  Make the following changes for `isMintWhitelistEnforced`:

  ```
  - /// @notice Whether the whitelist enforcement is enabled
  + /// @notice Whether the mint whitelist enforcement is enabled
  ```

- Router.sol#L91:

  Make the following changes for `isMintWhitelisted`:

  ```
  - /// @notice Mapping of user addresses to their whitelist status
  + /// @notice Mapping of user addresses to their mint whitelist status
  ```

Edges cases & Minor Recommendations:

- AssetReserve.sol#L107:

  Basic sanity check could be added `_custodian != address(0)` in `_setCustodian`.

- sNUSD.sol#L125-L127:

  Only allow changing vesting period if unvested amount is `0` else it would impact share price immediately (since `totalAssets` would change) which would be unexpected by users.

- Router.sol#L262:

  Since `Router` is non upgradable, so `expiry` should be set correctly so that there is no issues later in future integration.

  Something like:

  ```
  // where Waiting_Period is configurable by Admin
  expiry: block.timestamp + Waiting_Period
  ```

- Router.sol#L346-L371:

  Can include arg `startIndex` and `endIndex` in `getPendingMintRequests` and `getPendingRedemptionRequests` which would prevent Gas issue if number of request ids becomes too large.

- sNUSD.sol#L158:

  `_updateVestingAmount` once called is locked for vesting period.

  Thus if `redistributeLockedAmount` calls `_updateVestingAmount` then yield distribution via `transferInRewards` would fail at same time and would only succeed once vesting period is over.

  ```
  function _updateVestingAmount(uint256 _newVestingAmount) internal {
      if (getUnvestedAmount() > 0) revert StillVesting();

      vestingAmount = _newVestingAmount;
      lastDistributionTimestamp = block.timestamp;
  }
  ```

- Router.sol#L246.

- Router.sol#L281

  Currently, Redemption and Mint above max limit will be queued but would fail on execution causing Keeper to finally cancel the transaction from queue. Check for `_belowMaxRedeemPerBlock` and `_belowMaxMintPerBlock` could also be made at Router level for early revert.

- sNUSD.sol#L151:

  Centralization risk is present at multiple instances in code. One such instance is by using `redistributeLockedAmount`. `recoverToken` does not allow `DEFAULT_ADMIN_ROLE` to take `NUSD` but admin still could use `redistributeLockedAmount` with `to` as Admin controlled address to get `sNUSD`.

- DeployProtocol.s.sol#L157:

  Deployment script uses WETH hardcoded address which may change in other chain like Arbitrum. This should be taken care if deployment script need to changed for other chains (along with chain id restriction).

Neutrl: Fixed in PR 17.

Cantina Managed: Fix verified, and acknowledged some issues

Fixed:

- Typos are fixed.
- `_custodian != address(0)` check is added in `_setCustodian`.
- Changing vesting period is only allowed if unvested amount is 0.
- Expiry is default set to 7 days and is configurable and is now set properly.

Acknowledged:

- Can include arg startIndex and endIndex in getPendingMintRequests and getPendingRedemptionRequests.
- _updateVestingAmount once called is locked for vesting period. yield distribution via `transferInRewards` would only succeed once vesting period is over.

- Check for _belowMaxRedeemPerBlock and _belowMaxMintPerBlock could also be made at Router level for early revert.
- Centralization risk.
- Deployment script uses WETH hardcoded address -> This should be taken care if deployment script need to changed for other chains (along with chain id restriction) -> This is a mainnet specific deployment script.

3.3.2 `router.setRedeemWhitelisted` can be removed to save gas

Severity: Informational

Context: DeployProtocol.s.sol#L139-L147

Description: The function aims to mint a tiny share and burn the share to avoid the donation attack. While `router.setMintWhitelisted` is necessary to grant deployer mint capacity, The share is burnt so `deployer` cannot redeem the share.

Recommendation: Remove `router.setRedeemWhitelisted(deployer, true)` to save gas.

Neutrl: Fixed in PR 14.

Cantina Managed: Fix verified.

3.3.3 Consider only emit `_requestId` when the keeper process the mint and redemption request

Severity: Informational

Context: Router.sol#L198-L225

Description: When mint and redemption requests are canceled, only the `_requestId` is emitted.

```
emit MintRequestCancelled(_requestId);
```

However, when `mint` and `redemption` requests are served, the full order detail is still emitted.

```
emit ServeRequestMint(_requestId, order);
```

The order details contains a field `additionalData` and this `additionalData` can be arbitrarily long.

```
struct Order {
    OrderType orderType;
    uint256 expiry;
    address benefactor;
    address beneficiary;
    address collateralAsset;
    /// @dev When minting, this is the amount of collateral to deposit
    /// @dev When redeeming, this is the minimum amount of collateral to receive
    uint256 collateralAmount;
    /// @dev When minting, this is the minimum amount of NUSD to receive
    /// @dev When redeeming, this is the amount of NUSD to burn
    uint256 nusdAmount;
    /// @dev Additional data for the order, in case of future use for minters / redeemers
     bytes additionalData;
```

The order keeper must cover the gas costs even when additionalData is excessively long, causing the keeper to consistently incur losses due to high gas consumption.

Recommendation: Only emit `requestId` when redemption and mint request are served by keeper:

```
emit ServeRequestMint(_requestId);
```

and

```
emit ServeRequestRedemption(_requestId);
```

Neutrl: Fixed in PR 15.

Cantina Managed: Fix verified.

3.3.4 Deployment script fails in case when USDE traded below 1e18

Severity: Informational

Context: DeployProtocol.s.sol#L149-L160

Description: When the protocol attempts to mint the initial share to avoid donation attack, both the `_collateralAmount`, and the `_minNusdAmount` are `10e18`.

```
router.mint(deployer, deploymentArgs.usde, 10e18, 10e18, "");
```

```
function mint(
    address _beneficiary,
    address _collateralAsset,
    uint256 _collateralAmount,
    uint256 _minNusdAmount,
    bytes calldata _additionalData
) external whenNotPaused enforceMintWhitelist(msg.sender) {
```

if the USDE traded below 1e18, then the transaction can revert because of 0 slippage tolerance.

Recommendation:

```
uint256 minSlippage = vm.envUint("NEUTRAL_SNUSD__POST_SLIPPAGE_AMOUNT");

IERC20(deploymentArgs.usde).approve(address(router), minSlippage);

router.mint(deployer, deploymentArgs.usde, 10e18, minSlippage, "");

uint256 balance = nusd.balanceOf(deployer);

nusd.approve(deploymentAddress.sNusd, balance);

sNusd.deposit(balance, deployer);

// kill shares with donation to WETH
// assertEq(sNusd.totalSupply(), 10e18);

uint256 mintedShare = sNusd.balanceOf(deployer);

sNusd.transfer(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2, mintedShare);

assertEq(sNusd.totalSupply(), mintedShare);

vm.stopBroadcast();
```

Neutrl: Acknowledged. We will assess according to deployment simulation.

Cantina Managed: Acknowledged. If the deployment fails, the script can be easily modified to make a redeployment.