Level Money Contracts

Security Review

Cantina Managed review by:

Mario Poneder, Security Researcher

Om Parikh, Security Researcher

Delvir0, Junior Security Researcher

November 12, 2024

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**

3 Findings **4**
- 3.1 High Risk . . . 4
  - 3.1.1 Insufficient slippage protection in `mintlvlUSD` . . . 4
  - 3.1.2 `USDT` cannot be withdrawn from `AaveV3YieldManager` . . . 4
  - 3.1.3 DoS due to not accepting native ETH transfer . . . 5
  - 3.1.4 Inability to claim various protocol rewards due to missing implementation . . . 5
- 3.2 Medium Risk . . . 6
  - 3.2.1 No slippage protection in `mintlvlUSD` in case of matching decimals . . . 6
- 3.3 Low Risk . . . 7
  - 3.3.1 Conflicting method permissions . . . 7
  - 3.3.2 Ineffective role seggregation in `LevelBaseReserveManager` . . . 7
  - 3.3.3 The `maxSlippageThresholdBasisPoints` could exceed 100% . . . 8
  - 3.3.4 Treasury rake might not be taken due to rounding direction . . . 8
  - 3.3.5 Conflicting decimals handling . . . 8
  - 3.3.6 Excessive use of `whenNotPaused` might block withdrawals . . . 9
- 3.4 Informational . . . 10
  - 3.4.1 Superfluous granting of `DEFAULT_ADMIN_ROLE` in `AaveV3YieldManager` . . . 10
  - 3.4.2 Insufficient wrapper type checking . . . 10
  - 3.4.3 Unused storage variable in `LevelEigenlayerReserveManager.sol` . . . 11
  - 3.4.4 Outdated version of openzeppelin is used with potential security advisories . . . 11

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

Level is the first delta-neutral synthetic dollar with first-loss protection.

From Oct 23rd to Oct 30th the Cantina team conducted a review of level-money-contracts-1022 on commit hash fbd857ff. The team identified a total of **15** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 4
- Medium Risk: 1
- Low Risk: 6
- Gas Optimizations: 0
- Informational: 4

3 Findings

3.1 High Risk

3.1.1 Insufficient slippage protection in `mintlvlUSD`

Severity: High Risk

Context: LevelBaseReserveManager.sol#L215-L219

Description: In the `mintlvlUSD` method, the minimum `lvlUSDAmount` (slippage protection) is overwritten by the allowed slippage amount leading to a negligibly low minimum `lvlUSDAmount`, which can cause potential losses for the receiver / collateral provider.

The `mintlvlUSD` method of the `LevelBaseReserveManager` contract is responsible to create and submit a `MINT` order to the `LevelMinting` contract to mint the appropriate amount of `lvlUSD` for the given amount of collateral.

This order includes a minimum `lvlUSDAmount` which serves as a slippage parameter ensuring that the receiver / collateral provider is not at a loss due to an unexpectedly low amount of minted `lvlUSD`.

Of course, one cannot always expect full 1:1 minting therefore an amount proportional to `maxSlippageThresholdBasisPoints` is intended to be the allowed slippage:

```
// Apply max slippage threshold
lvlUSDAmount = lvlUSDAmount.mulDiv(
    maxSlippageThresholdBasisPoints,
    MAX_BASIS_POINTS
);
```

However, instead of subtracting the allowed slippage amount from the minimum `lvlUSDAmount`, it is overwritten by this small amount effectively eliminating the slippage protection.

Impact: Having a negligibly low minimum `lvlUSDAmount` exposes the receiver / collateral provider to a maximum slippage risk in terms of minted `lvlUSD` vs. provided collateral, which can turn out to be a severe loss.

Likelihood: With `maxSlippageThresholdBasisPoints` initially being set to `5` (5 bps = 0.05%), every call to `mintlvlUSD` is subject to a full slippage risk.

Recommendation: It is recommended to subtract the slippage amount from `lvlUSDAmount` to arrive at the desired minimum `lvlUSDAmount`:

```
  // Apply max slippage threshold
- lvlUSDAmount = lvlUSDAmount.mulDiv(
+ lvlUSDAmount -= lvlUSDAmount.mulDiv(
      maxSlippageThresholdBasisPoints,
      MAX_BASIS_POINTS
  );
```

Level: Fixed in PR 25.

Cantina Managed: Fixed.

3.1.2 `USDT` cannot be withdrawn from `AaveV3YieldManager`

Severity: High Risk

Context: LevelEigenlayerReserveManager.sol#L70, AaveV3YieldManager.sol#L92, AaveV3YieldManager.sol#L104

Description: `USDT` can only be deposited into but not withdrawn from the `AaveV3YieldManager` contract due to the implementation of `USDT` which is not fully ERC20-compliant.

The `AaveV3YieldManager` is intended to handle `USDC` & `USDT` (more to be added in the future) for interest accrual by supplying them as collateral to AAVE and wrapping the resulting rebasing & interest-accruing tokens.

However, the `withdraw` method relies on `IERC20.transfer` which expects a `bool` return value that Solidity is trying to decode even though the return value is not checked, while `USDT.transfer` does not implement a return value. Consequently, any attempt to withdraw `USDT` using this method will result in a revert.

```
function withdraw(
    address token, // e.g. USDT
    uint256 amount
) external {
    address aTokenAddress = underlyingToaToken[token];
    address wrapper = tokenToWrapper[aTokenAddress];
    IERC20(wrapper).safeTransferFrom(msg.sender, address(this), amount);
    _unwrapToken(wrapper, amount);
    _withdrawFromAave(token, amount);
    IERC20(token).transfer(msg.sender, amount);    // IERC20.transfer expects bool return value
}
```

Impact: `USDT` cannot be withdrawn from the `AaveV3YieldManager` contract breaking the protocol's intended flow of funds. Unwrapping & withdrawal from AAVE has to be processed manually instead.

Likelihood: `USDT` is fully intended to be used with the `AaveV3YieldManager` contract.

Recommendation: It is recommended to always rely on the `SafeERC20` library instead of calling `transfer`, `transferFrom` & `approve` of ERC20 tokens directly.

This recommendation applies to all three instances attached to this finding.

Level: Fixed in PR 25 and PR 26.

Cantina Managed: Fixed.

3.1.3 DoS due to not accepting native ETH transfer

Severity: High Risk

Context: LevelBaseReserveManager.sol#L26

Description: When depositing into underlying restating protocol, which is either vault or strategy which delegates to operator. If strategy / vault supports native restating or accrues rewards in ETH then contract will revert when withdrawing from such strategy / vault because it doesn't implement `receive` or `fallback` function.

The protocol correctly implements `transferEther` to remove the ETH out of contract once it is redeemed by reserve manager.

Recommendation: - Add `receive` / `fallback` with appropriate access control to accept ETH wherever required. If protocol doesn't want to interact with any currently deployed or future vault / strategies which can potentially transfer eth then they should document this explicitly.

3.1.4 Inability to claim various protocol rewards due to missing implementation

Severity: High Risk

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

Description: The current implementation lacks mechanisms to claim various rewards that accrue to the protocol from different sources. This affects:

- Aave Protocol Rewards:
  - Staked AAVE tokens rewards.
  - Chain-specific incentive campaign rewards (ARB, OP, ZKSYNC tokens).
  - Protocol integration specific incentives (e.g., SNX incentives for providing sUSD).
  - Other protocol-specific rewards.
- Reserve Manager Contract Rewards:
  - EigenLayer rewards.

Rewards accrue to the wrapped rebasing token wrapper contract as it holds the underlying funds. There is no implementation to call claimAllRewards or utilize allowClaimOnBehalf functions on the `RewardsController` contract for aave and `RewardsCoordinator` for eigenlayer

Impact:

- Accrued rewards become effectively locked in the protocol.
- Loss of value for protocol participants who should benefit from these reward mechanisms.
- Reduced protocol efficiency as incentive mechanisms cannot be fully utilized.
- Potential compound effect as unclaimed rewards may also miss out on additional yield opportunities.

Recommendation:

- Implement mechanism to claim rewards from specific integrations.
- Ensure rewards are re-invested, distributed to user or withdrawn.
- Write test cases to verify the same.

Level: Fixed in PR 25.

Cantina Managed: Fixed.

3.2 Medium Risk

3.2.1 No slippage protection in `mintlvlUSD` in case of matching decimals

Severity: Medium Risk

Context: LevelBaseReserveManager.sol#L203-L213

Description: In case `collateralDecimals == lvlUsdDecimals` in the `mintlvlUSD` method, the minimum `lvlUSDAmount` (slippage protection) will remain `0` leading to potential losses for the receiver / collateral provider.

The `mintlvlUSD` method of the `LevelBaseReserveManager` contract is responsible to create and submit a `MINT` order to the `LevelMinting` contract to mint the appropriate amount of `lvlUSD` for the given amount of collateral.

This order includes a minimum `lvlUSDAmount` which serves as a slippage parameter ensuring that the receiver / collateral provider is not at a loss due to an unexpectedly low amount of minted `lvlUSD`.

Assuming the collateral is also a USD stablecoin, the minimum `lvlUSDAmount` only needs to be scaled according to the collateral token's decimals:

```
uint256 lvlUSDAmount;

if (collateralDecimals < lvlUsdDecimals) {
    lvlUSDAmount =
        collateralAmount *
        (10 ** (lvlUsdDecimals - collateralDecimals));
} else if (collateralDecimals > lvlUsdDecimals) {
    lvlUSDAmount =
        collateralAmount /
        (10 ** (collateralDecimals - lvlUsdDecimals));
}
```

However, in case `collateralDecimals == lvlUsdDecimals`, the `lvlUSDAmount` remains `0`.

Impact: Having a minimum `lvlUSDAmount` of `0` exposes the receiver / collateral provider to a maximum slippage risk in terms of minted `lvlUSD` vs. provided collateral, which can turn out to be a severe loss.

Likelihood: The `lvlUSD` token has `18` decimals which is most common for ERC20 tokens. Despite `USDC/USDT` having only `6` decimals on Ethereum, it is not unlikely that another stablecoin with `18` decimals will be used as collateral.

Recommendation: It is recommended to cover the case where `collateralDecimals == lvlUsdDecimals`:

```
  uint256 lvlUSDAmount;

  if (collateralDecimals < lvlUsdDecimals) {
      lvlUSDAmount =
          collateralAmount *
          (10 ** (lvlUsdDecimals - collateralDecimals));
  } else if (collateralDecimals > lvlUsdDecimals) {
      lvlUSDAmount =
          collateralAmount /
          (10 ** (collateralDecimals - lvlUsdDecimals));
- }
+ } else {
+     lvlUSDAmount = collateralAmount;
+ }
```

Level: Fixed in PR 25.

Cantina Managed: Fixed.

3.3 Low Risk

3.3.1 Conflicting method permissions

Severity: Low Risk

Context: LevelBaseReserveManager.sol#L102-L112, LevelKarakReserveManager.sol#L26-L33, LevelSymbioticReserveManager.sol#L29-L43, AaveV3YieldManager.sol#L45-L53, AaveV3YieldManager.sol#L64-L67, AaveV3YieldManager.sol#L87-L93

Description: Throughout the protocol there are multiple methods that involve a transfer of funds, i.e. pulling funds via `ERC20.transferFrom`. These methods have different pemission levels, e.g. permissionless, restricted to the `MANAGER_AGENT_ROLE`, etc...

However, these methods do not increase token allowances to actually facilitate the involved transfers. Per protocol design this is manually handled by `forceApprove` methods which are restricted to the `DEFAULT_ADMIN_ROLE`.

Consequently, this leads to conflicting method permissions since many lower-permissioned methods are still dependent on the admin for token approvals.

Recommendation: We recommend to directly implement the necessary allowance handling wherever funds are transferred.

Level: Fixed in PR 25.

Cantina Managed: Fixed.

3.3.2 Ineffective role seggregation in `LevelBaseReserveManager`

Severity: Low Risk

Context: LevelBaseReserveManager.sol#L88-L90

Description: In the `LevelBaseReserveManager` contract, the same `_admin` account is assigned to the `DEFAULT_ADMIN_ROLE` as well as the `PAUSER_ROLE`:

```
_grantRole(DEFAULT_ADMIN_ROLE, _admin);
_grantRole(ALLOWLIST_ROLE, _allowlister);
_grantRole(PAUSER_ROLE, _admin);
```

Recommendation: We recommend to assign the `PAUSER_ROLE` to a separate account (different from `_admin`) at contract construction.

Level: Acknowledged.

Cantina Managed: Acknowledged.

3.3.3 The `maxSlippageThresholdBasisPoints` could exceed 100%

Severity: Low Risk

Context: LevelBaseReserveManager.sol#L332-L336

Description: The `setMaxSlippageThresholdBasisPoints` method of the `LevelBaseReserveManager` contract does not prevent the `maxSlippageThresholdBasisPoints` parameter from being set to values greater than 100%, i.e. `MAX_BASIS_POINTS`:

```
function setMaxSlippageThresholdBasisPoints(
    uint16 _maxSlippageThresholdBasisPoints
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    maxSlippageThresholdBasisPoints = _maxSlippageThresholdBasisPoints;
}
```

Recommendation: We recommend to enforce an upper limit of `MAX_BASIS_POINTS`:

```
  function setMaxSlippageThresholdBasisPoints(
      uint16 _maxSlippageThresholdBasisPoints
  ) external onlyRole(DEFAULT_ADMIN_ROLE) {
+     require(maxSlippageThresholdBasisPoints <= MAX_BASIS_POINTS);
    maxSlippageThresholdBasisPoints = _maxSlippageThresholdBasisPoints;
  }
```

Level: Fixed in PR 25.

Cantina Managed: Fixed.

3.3.4 Treasury rake might not be taken due to rounding direction

Severity: Low Risk

Context: LevelBaseReserveManager.sol#L167-L169

Description: In the `LevelBaseReserveManager` contract, the treasury rake will not be taken when minting small amounts of `lvlUSD`, i.e. when `amount * rakeBasisPoints < MAX_BASIS_POINTS`.

```
uint256 rake = amount.mulDiv(rakeBasisPoints, MAX_BASIS_POINTS);
uint256 remainder = amount - rake;
IERC20(token).safeTransfer(treasury, rake);
```

Due to `mulDiv` rounding down, the treasury rake can be circumvented by splitting a mint operation into multiple small ones which satisfy the above `amount` condition.

Recommendation: We recommend to implement a `mulDivUp` pattern which always rounds up and therefore the treasury rake will be taken in any case.

Level: Fixed in PR 25.

Cantina Managed: Fixed.

3.3.5 Conflicting decimals handling

Severity: Low Risk

Context: WrappedRebasingERC20.sol#L46-L54, AaveV3YieldManager.sol#L115-L126

Description: The `AaveV3YieldManager` contract expects the `wrapper`'s underlying `token` to have a `decimals` method, while the `WrappedRebasingERC20` contract also supports `_underyling` tokens without a `decimals` method.

```
// AaveV3YieldManager
function setWrapperForToken(
    address token,
    address wrapper
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (address(WrappedRebasingERC20(wrapper).underlying()) != token) {
        revert InvalidWrapper();
    }
    if (ERC20(token).decimals() != ERC20(wrapper).decimals()) {    // decimals essential
        revert TokenAndWrapperDecimalsMismatch();
    }
    tokenToWrapper[token] = wrapper;
}

// WrappedRebasingERC20
function decimals() public view virtual override returns (uint8) {
    try IERC20Metadata(address(_underlying)).decimals() returns (    // decimals optional
        uint8 value
    ) {
        return value;
    } catch {
        return super.decimals();
    }
}
```

Note that having a `decimals` method is optional according to the ERC-20 specification.

Recommendation: We recommend to resolve this conflict in favor of ERC-20 tokens that do not have a `decimals` method, i.e. potentially remove the decimals check from `setWrapperForToken`. Additionally, the (underlying) decimals in the constructor of `WrappedRebasingERC20` could be specified.

Level: Acknowledged.

Cantina Managed: Acknowledged.

3.3.6 Excessive use of `whenNotPaused` might block withdrawals

Severity: Low Risk

Context: LevelEigenlayerReserveManager.sol#L100, LevelKarakReserveManager.sol#L54, LevelSymbioticReserveManager.sol#L67

Description: `whenNotPaused` might prevent redeeming funds or completing withdrawal in case of emergency if pause and unpause are guarded by timelock in reserve managers.

Recommendation: Protocol should considering allowing withdrawals when paused if having pausing / unpausing functionality behind timelock and instead pause redeeming of lvlUSD to prevent systemic issues.

Level: Acknowledged.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Superfluous granting of `DEFAULT_ADMIN_ROLE` in `AaveV3YieldManager`

Severity: Informational

Context: AaveV3YieldManager.sol#L40

Description: In the constructor of `AaveV3YieldManager`, the `DEFAULT_ADMIN_ROLE` is assigned to the `_admin` account. However, this is already handled in the constructor of the base contract `BaseYieldManager`.

```
// AaveV3YieldManager
constructor(IPool _aavePoolProxy, address _admin) BaseYieldManager(_admin) {
    aavePoolProxy = _aavePoolProxy;
    _grantRole(DEFAULT_ADMIN_ROLE, _admin);
}

// BaseYieldManager
constructor(address _admin) {
    _grantRole(DEFAULT_ADMIN_ROLE, _admin);
}
```

Recommendation: We recommend to remove the `_grantRole(DEFAULT_ADMIN_ROLE, _admin)` call from the constructor of `AaveV3YieldManager`.

Level: Fixed in PR 25.

Cantina Managed: Fixed.

3.4.2 Insufficient wrapper type checking

Severity: Informational

Context: AaveV3YieldManager.sol#L115-L126

Description: When adding a rebasing token wrapper to the `AaveV3YieldManager` contract, the `setWrapperForToken` method performs an underlying token and a decimals check.

```
function setWrapperForToken(
    address token,
    address wrapper
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (address(WrappedRebasingERC20(wrapper).underlying()) != token) {
        revert InvalidWrapper();
    }
    if (ERC20(token).decimals() != ERC20(wrapper).decimals()) {
        revert TokenAndWrapperDecimalsMismatch();
    }
    tokenToWrapper[token] = wrapper;
}
```

However, it is still not assured that the `wrapper` is using the protocol's `WrappedRebasingERC20` contract. Any other contract having an `underlying` method and a `decimals` method could pass these checks too.

Recommendation: One way to ensure that a contract is actually a `WrappedRebasingERC20` contract is to deploy it. Therefore, we recommend to implement a factory contract that maintains a mapping of valid/deployed `WrappedRebasingERC20` contracts.

Level: Acknowledged.

Cantina Managed: Acknowledged.

3.4.3 Unused storage variable in `LevelEigenlayerReserveManager.sol`

Severity: Informational

Context: LevelEigenlayerReserveManager.sol#L17

```
string public operatorName;
```

is defined and set but not used for anything

Level: Acknowledged. This is more an internal tool for us to keep track of which operator we've delegated the reserve manager to.

Cantina Managed: Acknowledged.

3.4.4 Outdated version of openzeppelin is used with potential security advisories

Severity: Informational

Context: StakedlvlUSD.sol#L6-L8

Description: There are various security advisories affecting 4.9.0 version of the OpenZeppelin Contracts.

Recommendation:

- Consider upgrading to latest stable release of openzeppelin.
- Change imports in `StakedlvlUSD.sol` and other contracts importing from `4.9.0` to latest stable release.

Level: Acknowledged.

Cantina Managed: Acknowledged.