TRUFY

Smart Contract Audit Report
for
OmniFarming V2

Preliminary Comments

August, 2025

Contents

1 Introduction

OmniFarming V2 is a yield optimization protocol that extends the Yearn V3 vault architecture. It allows users to deposit assets into a vault, which are then allocated to various strategies to maximize returns. Key features include automated fund allocation, performance-based fees, and mechanisms to handle unrealized losses

1.1 Project Summary

- Project Name: OmniFarming V2
- Language: Solidity
- Codebase: https://github.com/Ghoulouis/yield_contracts.git
- Commit: f55aad5f2a1fe4f2fad394193056f55201e2aec1
- Audit method: Static Analysis, Manual Review
- Scope:
  - ◇ contracts/protocol/Vault.sol
  - ◇ contracts/protocol/libraries/logic/DebtLogic.sol
  - ◇ contracts/protocol/libraries/logic/ERC4626Logic.sol
  - ◇ contracts/strategy/OffChainStrategy.sol
  - ◇ contracts/protocol/libraries/Constants.sol
  - ◇ contracts/protocol/libraries/logic/UnlockSharesLogic.sol

1.2 Vulnerability Summary

Severity	# of Findings
Critical	2
Medium	5
Low	0
Informational	0

2 Findings

ID	Title	Type	Severity
ID-01	Incorrect Function Call in `maxMint` Implementation	Logic Error	**Medium**
ID-02	Conflicting **`require`** and Conditional Branch	Logic Error	**Medium**
ID-03	Incorrect Unlock Calculation (uses `profitMaxUnlockTime`)	Logic / Math Error	**Critical**
ID-04	Incorrect Role Assignment for `ROLE_QUEUE_MANAGER`	Access Control Error	**Medium**
ID-05	Incorrect Return in `withdraw` Function	Logic Error	**Medium**
ID-06	Incorrect Conversion in `maxMint` Function	Logic Error	**Medium**
ID-07	Incorrect Calculation in `maxRedeem` Function	Logic Error	**Critical**

3 Detailed Results

3.1 ID-01: Incorrect Function Call in `maxMint` Implementation

Type	Severity	Location
Logic Error	Medium	Vault.sol line 174

3.1.1 Description

The `maxMint` function in `Vault.sol` is incorrectly delegating its return value to `ERC4626Logic.maxDeposit(vaultData, user)` instead of `ERC4626Logic.maxMint(vaultData, user)`.

According to the ERC-4626 specification, the `maxMint` method should return the maximum number of shares that can be minted for the given `user`, not the maximum assets that can be deposited.
Using `maxDeposit` here leads to inconsistent behavior between `maxMint` and other related ERC-4626 functions, potentially causing UI misrepresentation, incorrect allowance calculations, or unexpected reverts in integrations.

3.1.2 Recommendations

- Replace the function call with the correct logic:

```
return ERC4626Logic.maxMint(vaultData, user);
```

3.2 ID-02: Conflicting `require` and Conditional Branch

Type	Severity	Location
Logic Error	Medium	DebtLogic.sol lines 30, 31, 39, 49

3.2.1 Description

In `ExecuteProcessReport`, the function begins with:

```
require(strategy != address(this), "Invalid strategy");
require(
    vault.strategies[strategy].activation != 0,
    "Inactive strategy"
);
```

Later, it contains:

```
if (strategy != address(this)) {
    require(
        vault.strategies[strategy].activation != 0,
        "Inactive strategy"
    );
    ...
} else {
    ...
}
```

The first `require(strategy != address(this), ...)` makes the `else` branch unreachable, resulting in **dead code** and contradictory logic.
If the intention is to allow `strategy == address(this)` for self-reporting, the initial `require` prevents this scenario.
If the intention is to forbid self-reporting, the `else` branch should be removed to avoid confusion and reduce maintenance risk.

This redundancy may mislead future developers, cause incorrect assumptions in audits, or lead to accidental logic breaks during refactoring.

3.2.2 Recommendations

- **Clarify intended behavior** for `strategy == address(this)`:
    - If **not allowed**, remove the `else` branch entirely and keep the initial `require`.
    - If **allowed**, remove the initial `require(strategy != address(this), ...)` and rely on the `if-else` block to handle both cases.

3.3 ID-03: Incorrect Unlock Calculation (uses `profitMaxUnlockTime`)

Type	Severity	Location
Logic / Math Error	Critical	UnlockSharesLogic.sol line 14

3.3.1 Description

The `unlockShares` function incorrectly uses `vault.profitMaxUnlockTime` instead of `vault.profitUnlockingRate` when calculating unlocked shares. This leads to inaccurate results in share unlocking logic, potentially affecting profit distribution and accounting.

3.3.2 Recommendation

Replace `vault.profitMaxUnlockTime` with `vault.profitUnlockingRate` in the calculation, ensuring the formula correctly reflects the intended unlocking rate:

```
return ((vault.profitUnlockingRate *
    (block.timestamp - vault.lastProfitUpdate)) /
    Constants.MAX_BPS_EXTENDED);
```

3.4 ID-04: Incorrect Role Assignment for ROLE_QUEUE_MANAGER

Type	Severity	Location
Access Control Error	Medium	Contants.sol line 26

3.4.1 Description

The constant ROLE_QUEUE_MANAGER is incorrectly assigned the hash of the string ”ROLE_ACCOUNTANT_MANAGER” instead of ”ROLE_QUEUE_MANAGER”. This results in a mismatch between the role name and its identifier, potentially causing access control checks to fail.

3.4.2 Recommendation

Update the assignment to match the correct role name:

```
bytes32 public constant ROLE_QUEUE_MANAGER =
    keccak256("ROLE_QUEUE_MANAGER");
```

3.5 ID-05: Incorrect Return in `withdraw` Function

Type	Severity	Location
Logic Error	Medium	Vault.sol line 297

3.5.1 Description

The `withdraw` function currently executes `WithdrawLogic.executeRedeem` and returns its result. However, according to the intended behavior, the function should only return the calculated `shares` value after converting from the given `assets`. Executing the redeem process here may cause unintended state changes or double processing.

3.5.2 Recommendation

Modify the function to return only the calculated shares:

```solidity
function withdraw(
    uint256 assets,
    address receiver,
    address owner
) public override(ERC4626Upgradeable, IVault) returns (uint256) {
    ManagementFeeLogic.caculateManagementFee(vaultData);
    uint256 shares = ERC4626Logic.convertToShares(
        vaultData,
        assets,
        Math.Rounding.Ceil
    );
    return shares;
}
```

3.6 ID-06: Incorrect Conversion in `maxMint` Function

Type	Severity	Location
Logic Error	Medium	ERC4626Logic.sol line 42

3.6.1 Description

The `maxMint` function currently returns the result of `vault._convertToAssets(maxDepositAmount, Math.Rounding.Floor)`. This is incorrect, as it should return the equivalent shares value, not the assets value.

3.6.2 Recommendation

Update the function to use `_convertToShares`:

```
function maxMint(
    DataTypes.VaultData storage vault,
    address receiver
) external view returns (uint256) {
    uint256 maxDepositAmount = vault.maxDeposit(receiver);
    return vault._convertToShares(maxDepositAmount, Math.
       Rounding.Floor);
}
```

3.7 ID-07: Incorrect Calculation in maxRedeem Function

Type	Severity	Location
Logic Error	Critical	ERC4626Logic.sol line 70

3.7.1 Description

The maxRedeem function currently returns only the result of vault._convertToShares(maxWithdrawAmount, Math.Rounding.Floor) without ensuring that it does not exceed the owner's share balance. This can result in returning more shares than the owner actually possesses. Additionally, _maxWithdraw is incorrectly called with 0 as the second parameter, which may lead to inaccurate withdrawal limits.

3.7.2 Recommendation

Update the function to take the minimum between the converted shares and vault.balanceOf (owner), and replace the 0 parameter in _maxWithdraw with the correct value based on intended withdrawal logic:

```
function maxRedeem(
    DataTypes.VaultData storage vault,
    address owner
) external view returns (uint256) {
    uint256 maxWithdrawAmount = vault._maxWithdraw(
        owner,
        /* correct param here */,
        new address[](0)
    );
    uint256 shares = vault._convertToShares(maxWithdrawAmount,
       Math.Rounding.Floor);
    return Math.min(shares, vault.balanceOf(owner));
}
```

4 Appendix

4.1 Severity Definitions

Critical

This level vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.

Medium

This level vulnerabilities are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to critical-risk severity.

Low

This level vulnerabilities should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution.

Informational

This level vulnerabilities can be ignored. They are code style violations and informational statements in the code. They may not affect the smart contract execution.

4.2 Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.