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/strategy/OffChainStrategy.sol

1.2 Vulnerability Summary

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

2 Findings

ID	Title	Type	Severity
ID-01	Missing Safe Transfer in `invest` Function	Logic Error	**Critical**
ID-02	Missing Safe Transfer in `takeProfit` Function	Logic Error	**Critical**

3 Detailed Results

3.1 ID-01: Missing Safe Transfer in `invest` Function

Type	Severity	Location
Security Issue	Critical	OffChainStrategy.sol line 49

3.1.1 Description

The `invest` function uses `IERC20(asset()).transfer(agent, amount)` for token transfers. Using `transfer` directly can cause unexpected reverts or silent failures with non-standard ERC20 tokens, potentially leading to loss of funds or locked tokens.

3.1.2 Recommendation

Replace `transfer` with OpenZeppelin's `safeTransfer` to ensure proper handling of return values and compatibility with non-standard ERC20 tokens:

```
IERC20(asset()).safeTransfer(agent, amount);
```

3.2 ID-02: Missing Safe Transfer in `takeProfit` Function

Type	Severity	Location
Security Issue	Critical	OffChainStrategy.sol line 56

3.2.1 Description

The `takeProfit` function uses `IERC20(asset()).transfer(agent, amount)` for token transfers. Using `transfer` directly can cause unexpected reverts or silent failures with non-standard ERC20 tokens, potentially leading to loss of funds or locked tokens.

3.2.2 Recommendation

Replace `transfer` with OpenZeppelin's `safeTransfer` to ensure proper handling of return values and compatibility with non-standard ERC20 tokens:

```
IERC20(asset()).safeTransfer(agent, amount);
```

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`.