YieldFarm Project Security Audit

Audit Resources:

Github Repository of the project was provided. (Link)

Project Author:

- Marvellous (Github)

Project Auditor:

- Umair Mirza (Github)

Table of Contents

Audit Summary

The YieldFarm project has been compiled, deployed and tested using the **Foundry** smart contract development tool chain. The project is comprised of two smart contracts, namely:

- AIMVault.sol
- AIMVaultFactory.sol

Following libraries and interfaces have been integrated with the smart contracts:

- Solmate
- OpenZeppelin
- CErc20 Interface

The contracts have been audited by 1 resident from September 28th to October 1st. The repository was under active development during the audit.

Scope

The scope of this audit is limited to the smart contracts mentioned above. Frontend modules of the project have not been audited.
The commit that has been audited is: **2e8419e2d34684a4da0ff20ca9f13310c7fab093**
This audit is about identifying potential vulnerabilities in the smart contracts. The audit may not identify all potential attack vectors or areas of vulnerability.

Findings Description

Findings have been broken down into sections by their respective impact:

- Critical, High, Medium, Low Impact
    - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas Savings
    - Findings that can improve the gas efficiency of the contracts.
- Informational
    - Findings including recommendations and best practices.

Medium Findings

1. Medium - The return value of an external call is not stored in a local or state variable

Proof of Concept

AIMVault.afterDeposit(uint256,uint256) (src/AIMVault.sol#107-110) ignores return value by UNDERLYING.approve(address(cToken),_assets) (src/AIMVault.sol#108):

```
function afterDeposit(uint256 _assets, uint256) internal override {
        UNDERLYING.approve(address(cToken), _assets);
        require(cToken.mint(_assets) == 0, "COMP: Deposit Failed");
```

```
    }
```

Impact

If the approve function returns false then the afterDeposit() function will revert without any reason. This can have undesired consequences.

Recommendation

Use the return value of the approve function to check if the address has been correctly approved for allowance.

Informational Findings

2. Informational - Pragma version^0.8.15 necessitates a version too recent to be trusted.

Proof of Concept

Pragma version^0.8.15 (src/interface/CErcInterface.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.15;
```

Impact

Using an old or unstable version prevents access to new Solidity security checks.

Recommendation

Consider using the latest version of Solidity for testing or deploy with any of the following Solidity versions:

- 0.5.16 - 0.5.17
- 0.6.11 - 0.6.12
- 0.7.5 - 0.7.6
- 0.8.16

3. Informational - Unused parameter without name in AIMVault.sol

Proof of Concept

afterDeposit() function written in AIMVault.sol accepts two parameters and the second parameter has no name and no use.

```
function afterDeposit(uint256 _assets, uint256) internal override {
        UNDERLYING.approve(address(cToken), _assets);
        require(cToken.mint(_assets) == 0, "COMP: Deposit Failed");
    }
```

Impact

Adding unused on nameless parameters can have undesired effects and will confuse the reviewer of the code

Recommendation

If there is no use for the extra parameter then it should be removed from the function.