True Yield Review

Review Resources:
A github repo containing protocol documentation, smart contracts and web app was provided.
Project Author:
- Umir Mirza (dreamygeek)

Project Auditor:
- Ahiara Ikechukwu Marvellous

Table of Contents

Review Summary

True Yield

TrueYield is a DeFi Staking protocol that lets you stake your ETH and Generate Passive income yield

The main branch of the fusion finance repo was reviewed today. All main contracts, FusionCore.sol and FusionToken.sol were covered.

Scope

Code
Commit

The commit reviewed was `b1bfae301e6493e71eae0700ff7b5cd567eb8653`. The review covered the repository at the specific commit and focused on the contracts directory.
All findings were presented to the umir.

The review is a code review of smart contracts to identify potential vulnerabilities in the code.

Code Evaluation Matrix

Category	Mark	Description
Access Control	None	Access control wasn't applied in the contract.
Mathematics	Good	Solidity 0.8.9 is used, which provides overflow and underflow protection. Staking and withdrawal maths checks out. No low-level bitwise operation was used.
Libraries	Good	OpenZeppelin contracts such as ERC20, Ownable, IERC20 are imported. Chainlink's AggregatorV3Interface was used for ETH/USD price feed
Complexity	Low	Asides the mathematical calculations and aave integration no extra complexity was added. No proxy contracts or delegatecalls used.
Documentation	Good	Comments exist for a couple of functions and state variables but more detailed comments needed.
Monitoring	None	No events for core functions that modify state variables.
Testing	Good	All tests were passing and test coverage was expansive.

Findings Explanation

Findings are 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

Low Findings

1. **Ignored return at IERC20(aWethAddress).approve(address(iWethGateway),type()(uint256).max) (contracts/TrueYield.sol#L164)**

2. **Dangerous strict equality at contracts/TrueYield.sol#L155.**

   Impact
   Can result in a denial of service attack

   Proof of concept

```
    function closePosition(uint256 positionId) external {
        require(
            positions[positionId].walletAddress == msg.sender,
            "Only the creator can modify the position"
        );
        require(positions[positionId].open, "Position is closed");

        positions[positionId].open = false;

        uint256 _staked = positions[positionId].weiStaked;

        //Withdraw lended funds via the Weth Gateway
        //It will convert back the WETH to ETH and send it to the contract
        //Ensure you set the relevant ERC20 allowance of aWETH, before calling this
function, so the WETHGateway contract can burn the associated aWETH
        IERC20(aWethAddress).approve(address(iWethGateway), type(uint256).max);
        iWethGateway.withdrawETH(lendingPoolAddress, _staked, address(this));

        // If the user is un-staking before the Unlock period, they won't gain any
interest
        if (block.timestamp > positions[positionId].unlockDate) {
            uint256 amount = _staked + positions[positionId].weiInterest;
            (bool success, ) = payable(msg.sender).call{value: amount}("");
            require(success, "Transaction failed");
        } else {
            (bool success, ) = payable(msg.sender).call{value: _staked}("");
            require(success, "Transaction failed");
        }
    }
```

3. **Ether received through receive() is stuck in contract**

Impact
None, no loss of funds since ether is stuck in contract.

Recommendation
Add a withdraw() function

4. Unused function parameter in TrueYield.sol and MockTrueYield.sol

Recommendation
Remove uint256 numDays from line 115 in TrueYield.sol and 89 in MockTrueYield.sol

Gas Savings

1. withdrawLend function in FusionCore.sol

Proof of concept
Unnecessary initialization of local variable to store _amount argument.

```
function withdrawLend(uint256 _amount) public {
       require(isLending[msg.sender], "Can't withdraw before lending!");
       require(
           lendingBalance[msg.sender] >= _amount,
           "Insufficient lending balance!"
       );

       uint256 yield = calculateYieldTotal(msg.sender);
       fusionBalance[msg.sender] += yield;
       startTime[msg.sender] = block.timestamp;

       lendingBalance[msg.sender] -= _amount;

       if (lendingBalance[msg.sender] == 0) {
           isLending[msg.sender] = false;
       }

       require(baseAsset.transfer(msg.sender, _amount), "Transaction
failed!");

       emit WithdrawLend(msg.sender, _amount);
   }
```

Impact
Gas savings, 100 gas saved

Informational Findings

1. **Solc version 0.8.0 not recommended for deployment.**

   Recommendation
   Use solc version 0.8.13-0.8.16 for bug fixes, compiler optimization and contract bytecode optimizations

2. **Add more comments on functions and its parameters .**

   Clarification needed on what the function does.

3. **State variable, position at TrueYield.sol#L50 is never used in TrueYield.sol**

   Recommendation
   Remove unused state variables

Final Remarks

Having reviewed the contracts, no critical vulnerabilities were found across the main contracts with openzeppelin and aave integrations. The maths behind staking work as expected. Test covers all functions and edge cases that can occur with aave.