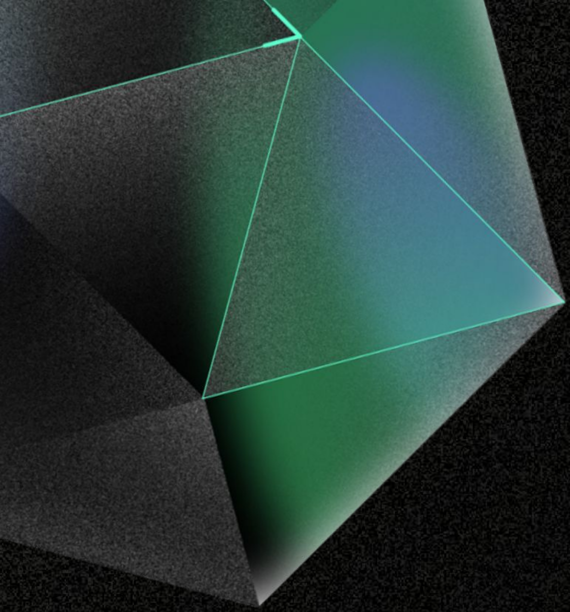
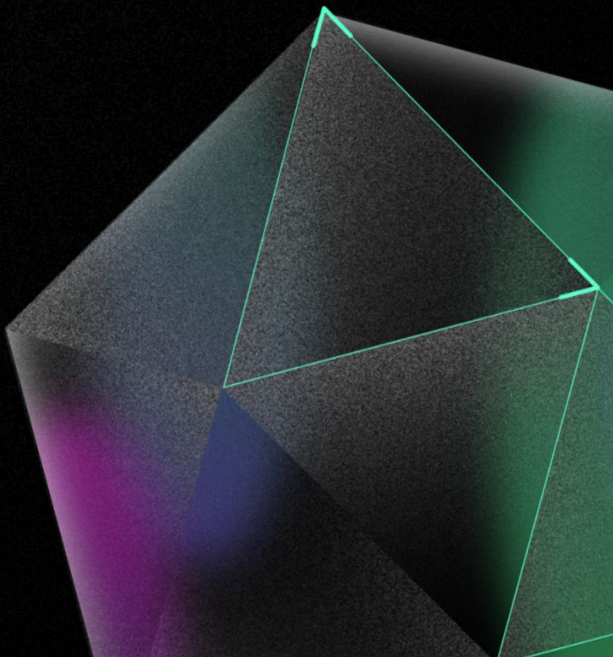


hexens

SMART CONTRACT AUDIT REPORT FOR SYNCDAO

25.05.2022



Contents

Summary

Severity	Number of Findings
Critical	0
High	1
Medium	1
Low	5
Informational	2

Total: 9

Scope

The analyzed smart contracts are located in the following repository folder: https://github.com/syncdao/contracts/commit/2744b00e9de9aa7a00be5c79791a2e95553e8d9a

Weaknesses

This section contains the list of discovered weaknesses.

1. Incorrect reward calculation

Severity: High

Status: fixed

Resolution: make sure the withdrawal logic is correct

Description:

Calling minting PVT tokens calls the **farm** function of **IStrategy.sol**. Looking at the implementation of the function in the **YearnStrategy.sol** contract, we can see that it makes a call to a **deposit** function of the given yearn vault.

This function returns the exact amount of shares given to the message sender (if **to** isn't specified) but this amount is not saved anywhere. Then, when the user comes for rewards, **estimateReward** is being called, calculates the token amount based on the total vault shares balance, and returns only the integer part of it. After that, the contract makes a call to the **takeReward** function **IStrategy.sol**, which calls the **withdraw** function of the yearn vault, sending the token amount as the first argument. But the withdraw function accepts the amount of the share as the first argument, not the amount of the tokens.

2. Centralization risk

Severity: Medium

Status: acknowledged

Resolution: remove the function or implement a time lock mechanism

Description:

The contract admin role on line 254 of **Manager.sol** is allowed to take all users' provided tokens. Moreover, it also can reset the contract while users' tokens are staked.

```
function takeAllStableTokens(address newPVTTokenAddress_) public
onlyRole(DEFAULT_ADMIN_ROLE) {

    (bool success,) =

address(strategy).delegatecall(abi.encodeWithSignature('takeReward(add
ress)', _msgSender()));
    require(success, 'Delegate call failed');
    if (address(0x0) != newPVTTokenAddress_) {
        pvtToken = PVTToken(newPVTTokenAddress_);
        _reset();
    }
 }
```

3. Incorrect lastTotalWork calculation

Severity: Low

Status: no impact by design

Resolution: revise the **lastTotalWork** logic

Description:

If multiple transactions to mint PVT tokens will appear in the same block, the **lastTotalWork** on lines 270 and 280 in the file **Manager.sol** would include only the first transaction's **pvtAmount**.

```
function _mintPVTToken(uint256 stableTokenAmount_, bool autoStake_) private {

    uint256 pvtAmount = stableTokenAmount_ * tokenRateT / tokenRateB;
    lastTotalWork += totalPVTAmount * (block.number - lastBlockNumber);
    if (autoStake_) {
      pvtToken.mint(address(this), pvtAmount);
      _stake(_msgSender(), pvtAmount);
    } else {
      pvtToken.mint(_msgSender(), pvtAmount);
      ownerStake.lastTotalWork += ownerStake.pvtAmount * (block.number -
ownerStake.lastBlockNumber);
      ownerStake.lastBlockNumber = block.number;
      ownerStake.pvtAmount += pvtAmount;
    }
    totalPVTAmount += pvtAmount;
    lastBlockNumber = block.number;
    totalStableTokenAmount += stableTokenAmount_;

    emit Minted(_msgSender(), pvtAmount);
  }
```

4. Redundant check

Severity: Low

Status: fixed

Resolution: make sure there should not be any other checks/restrictions

Description:

The function argument **percentage_** in the file **Manager.sol** on lines 118, 124 and 130 respectively is declared as uint256 which means the "unsigned integer of 256 bits". The check of the unsigned integer to be greater or equal to zero (0 <=) is redundant, because it can never be less than zero by its definition.

```
function changeAffiliatePercentage(uint256 percentage_) public
...
    require(0 <= percentage_ && percentage_ <= 100, 'Percentage must be from 0
to 100');
...
  }
...
    require(0 <= percentage_ && percentage_ <= 100, 'Percentage must be from 0
to 100');
...
  function changeOwnerPercentage(uint256 percentage_) public
onlyRole(DEFAULT_ADMIN_ROLE) {
    require(0 <= percentage_ && percentage_ <= 100, 'Percentage must be from 0
to 100');
```

5. Gas optimisation

Severity: Low

Status: fixed

Resolution: call the function once and store the returned value in the local memory variable

Description:

The function **_distributeReward** in the file **Manager.sol** makes frequent external calls to the **strategy.vaultTokenAddress()**, which causes excessive gas costs.

```
 _delegateTakeRewardIfNeeded(_msgSender(), strategy.vaultTokenAddress(),
                          totalAmount_ - ownerAmount_ - affiliateAmount_ -
charityAmount_);
    if (address(0x0) == affiliateAddress_) {
      _delegateTakeRewardIfNeeded(defaultAffiliate,
                            strategy.vaultTokenAddress(),
                            affiliateAmount_ + ownerAmount_);
    } else {
      _delegateTakeRewardIfNeeded(defaultAffiliate,
strategy.vaultTokenAddress(), ownerAmount_);
      _delegateTakeRewardIfNeeded(affiliateAddress_,
strategy.vaultTokenAddress(), affiliateAmount_);
    }
    _delegateTakeRewardIfNeeded(charity, strategy.vaultTokenAddress(),
charityAmount_);
  }
```

6. Gas optimisation

Severity: Low

Status: fixed

Resolution: call the function once and store the returned value in the local memory variable

Description:

The function **_distributeReward** in the file **YearnStrategy.sol** makes frequent internal calls to the **vaultTokenAddress()**, which causes excessive gas costs.

```
if (erc20Token_ != vaultTokenAddress()) {
            uint256 amountBefore =
IERC20(vaultTokenAddress()).balanceOf(address(this));
            _swap(erc20Token_, vaultTokenAddress(), address(this), amount_);
            vaultTokenAmount =
IERC20(vaultTokenAddress()).balanceOf(address(this)) - amountBefore;
        }
        IERC20(vaultTokenAddress()).safeApprove(vaultAddress(),
vaultTokenAmount);
```

7. Function missing authorization

Severity: Low

Status: acknowledged

Resolution: add authorization logic to the function to be called only by the contract itself (delegate call)

Description:

The function **takeReward** on lines 75 and 86 in the file **YearnStrategy.sol** is not restricted to be called only as a delegate. This function will allow anyone to withdraw the misused balance, in case it is called directly.

```
function takeReward(address to_, address expectedToken_, uint256 amount_)
public override {

    address tokenAddress = vaultTokenAddress();
    if (tokenAddress == expectedToken_) {
      VaultInterface(vaultAddress()).withdraw(amount_, to_);
    } else {
      VaultInterface(vaultAddress()).withdraw(amount_, address(this));
      _swap(tokenAddress, expectedToken_, to_, amount_);
    }
  }

  function takeReward(address to_) public override {

    VaultInterface(vaultAddress()).withdraw(type(uint256).max, to_);
  }
```

8. Public function can be declared external

Severity: Informational

Status: fixed

Resolution: use the external attribute for functions that are never called from the contract

Description:

Public functions in the files **PVTToken.sol**, **YearnStrategy.sol** and **Manager.sol** that are never called by the contract should be declared external.

9. Implicit visibility

Severity: Informational

Status: fixed

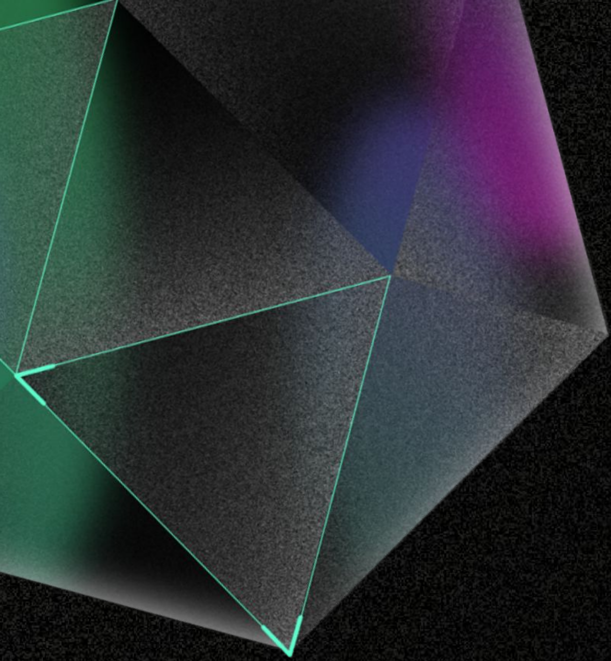
Resolution: always declare the visibility implicitly

Description:

The visibility is not explicitly defined for the state variable **stakersLookup** on line 57 in the file **Manager.sol**, which makes it implicitly defined as internal, making it unclear for some reviewers or developers.

```
PVTToken public pvtToken;
  IStrategy public strategy;


  mapping(address => Stake) public stakesMapping;
  address[] stakersLookup;
```



hexens