FYEO

Security Assessment of the CivFund Contracts

CIV Fund

April 2023
Version 1.2

Presented by:

FYEO Inc.

PO Box 147044
Lakewood CO 80214
United States

Security Level
Strictly Confidential

TABLE OF CONTENTS

List of Figures

LIST OF TABLES

EXECUTIVE SUMMARY

OVERVIEW

CIV Fund engaged FYEO Inc. to perform a Security Assessment of the CivFund Contracts.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on March 15 - March 28, 2023, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

The following timeline outlines the process of this audit:

- Code Audit: March 15th - March 28th (10 working days)
- V1 Draft Report: No later than April 4th
- Remediation Period: April 4th - April 11th
- Re-Review: Finalised 19th April 2023
- Final Report Date: Dated at the time of issue of this document

KEY FINDINGS

The following issues were identified during the testing period. They have since been remediated:

- FYEO-CV-01 – Incorrect startingEpoch calculation
- FYEO-CV-02 – withdrawPending doesn't burn fundRepresentToken
- FYEO-CV-03 – Incorrect currentWithdraw calculation
- FYEO-CV-04 – Incorrect starting value in loop in claimGuaranteeToken
- FYEO-CV-05 – Incorrect withdraw amount calculation for the second withdraw
- FYEO-CV-06 – Possible DOS in withdrawFromVault

- FYEO-CV-07 – Possible DOS when burning represent tokens
- FYEO-CV-08 – Logic that depends on user balance can be manipulated
- FYEO-CV-09 – NAV in deposit may include fees
- FYEO-CV-10 – NAV in withdrawAll may include fees
- FYEO-CV-11 – Possible DOS in depositToFund
- FYEO-CV-12 – Race condition on updating NAV
- FYEO-CV-13 – watermark is not updated on deposit
- FYEO-CV-14 – withdrawPending doesn't updates some storage variables
- FYEO-CV-15 – Fee calculation is incorrect if NAV increasing slowly
- FYEO-CV-16 – Incomplete events
- FYEO-CV-17 – Incorrect check when deleting userInfo
- FYEO-CV-18 – Incorrect type for locksCount
- FYEO-CV-19 – Incorrect withdraw balance check
- FYEO-CV-20 – Missing zero-address check
- FYEO-CV-21 – Reentrancy in collectFee
- FYEO-CV-22 – The size of withdrawAddress array is not validated in addPool
- FYEO-CV-23 – The withdrawERC20 is able to transfer guaranteeToken and lpToken
- FYEO-CV-24 – Usage of non-existing array in getUnclaimedTokenEpochs
- FYEO-CV-25 – Vulnerable OpenZeppelin version
- FYEO-CV-26 – lpToken can be withdrawn before withdrawFromVault
- FYEO-CV-27 – CivVault has no unit tests
- FYEO-CV-28 – Deposit info storage can be optimized
- FYEO-CV-29 – Distinguishing deposits in the same epoch is unnecessary
- FYEO-CV-30 – GAS usage optimization
- FYEO-CV-31 – Incorrect amount check in withdrawPending
- FYEO-CV-32 – Insufficient functions documentation
- FYEO-CV-33 – MAX_UINT check does nothing
- FYEO-CV-34 – Not used events
- FYEO-CV-35 – Possible DOS in claimGuaranteeToken
- FYEO-CV-36 – Public visibility is set for the function that is not called internally
- FYEO-CV-37 – Some function change state without emitting events
- FYEO-CV-38 – Two pools can be created with the same tokens

- FYEO-CV-39 – Variable feeBase can not be changed

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Assessment of the CivFund Contracts. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/CivilizationCIV/farm-contracts/blob/auto_lock_farm/contracts/ with the commit hash 914b5720fafb0c518928716bc0f3e5c24c60b8a7.

A re-review was carried out on various commits. Those are as follows:

- 9fde59af1646c735a457568795f5ee3b07072ec1
- a9aa442e842fa448ce09b0a93a831a2a45b47687
- 702769af9e0f77cb0f452764bce189c585207c44
- a376dbcc64b36af500176547e36d7e8b4038dfc8
- ef7dcb763fcd875d4550d68905e1210cbf1c5cd6
- fb051f77e038a470d099eb8aa548b9c7ea8a947e
- ce77a551b143edd762256873aa9e8ba2061ea3c3

Files included in the code review
farm-contracts/ ├── contracts/ │   └── CivVault.sol

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Assessment of the CivFund Contracts, we discovered:

- 2 findings with CRITICAL severity rating.
- 5 findings with HIGH severity rating.
- 7 findings with MEDIUM severity rating.
- 12 findings with LOW severity rating.
- 13 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

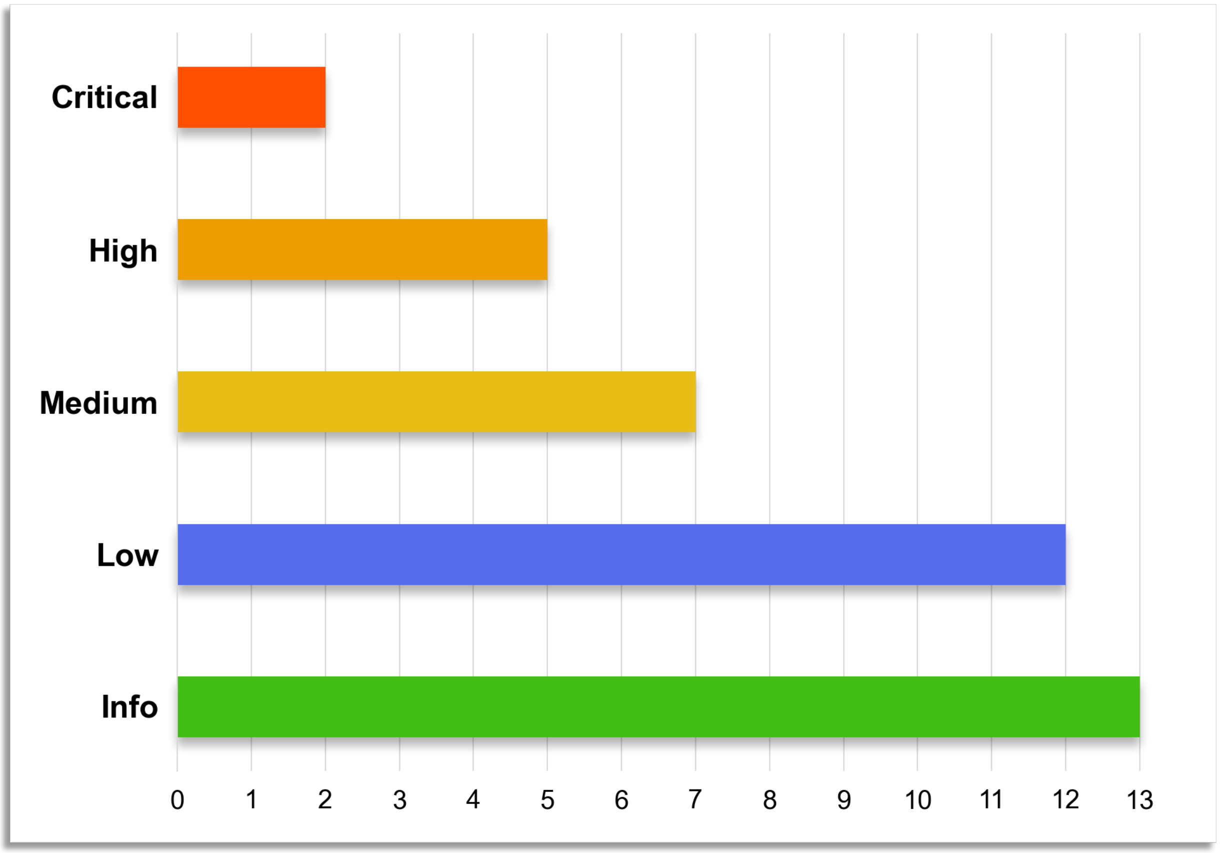

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Status	Severity	Description
FYEO-CV-01	Remediated	Critical	Incorrect startingEpoch calculation
FYEO-CV-02	Remediated	Critical	withdrawPending doesn't burn fundRepresentToken
FYEO-CV-03	Remediated	High	Incorrect currentWithdraw calculation
FYEO-CV-04	Remediated	High	Incorrect starting value in loop in claimGuaranteeToken
FYEO-CV-05	Remediated	High	Incorrect withdraw amount calculation for the second withdraw
FYEO-CV-06	Remediated	High	Possible DOS in withdrawFromVault
FYEO-CV-07	Remediated	High	Possible DOS when burning represent tokens
FYEO-CV-08	Remediated	Medium	Logic that depends on user balance can be manipulated
FYEO-CV-09	Remediated	Medium	NAV in deposit may include fees
FYEO-CV-10	Remediated	Medium	NAV in withdrawAll may include fees
FYEO-CV-11	Remediated	Medium	Possible DOS in depositToFund
FYEO-CV-12	Remediated	Medium	Race condition on updating NAV
FYEO-CV-13	Remediated	Medium	watermark is not updated on deposit
FYEO-CV-14	Remediated	Medium	withdrawPending doesn't updates some storage variables
FYEO-CV-15	Remediated	Low	Fee calculation is incorrect if NAV increasing slowly

FYEO-CV-16	Remediated	Low	Incomplete events
FYEO-CV-17	Remediated	Low	Incorrect check when deleting userInfo
FYEO-CV-18	Remediated	Low	Incorrect type for locksCount
FYEO-CV-19	Remediated	Low	Incorrect withdraw balance check
FYEO-CV-20	Remediated	Low	Missing zero-address check
FYEO-CV-21	Remediated	Low	Reentrancy in collectFee
FYEO-CV-22	Remediated	Low	The size of withdrawAddress array is not validated in addPool
FYEO-CV-23	Remediated	Low	The withdrawERC20 is able to transfer guaranteeToken and lpToken
FYEO-CV-24	Remediated	Low	Usage of non-existing array in getUnclaimedTokenEpochs
FYEO-CV-25	Remediated	Low	Vulnerable OpenZeppelin version
FYEO-CV-26	Remediated	Low	lpToken can be withdrawn before withdrawFromVault
FYEO-CV-27	Remediated	Informational	CivVault has no unit tests
FYEO-CV-28	Remediated	Informational	Deposit info storage can be optimized
FYEO-CV-29	Remediated	Informational	Distinguishing deposits in the same epoch is unnecessary
FYEO-CV-30	Remediated	Informational	GAS usage optimization
FYEO-CV-31	Remediated	Informational	Incorrect amount check in withdrawPending
FYEO-CV-32	Remediated	Informational	Insufficient functions documentation
FYEO-CV-33	Remediated	Informational	MAX_UINT check does nothing
FYEO-CV-34	Remediated	Informational	Not used events
FYEO-CV-35	Remediated	Informational	Possible DOS in claimGuaranteeToken

FYEO-CV-36	Remediated	Informational	Public visibility is set for the function that is not called internally
FYEO-CV-37	Remediated	Informational	Some function change state without emitting events
FYEO-CV-38	Remediated	Informational	Two pools can be created with the same tokens
FYEO-CV-39	Remediated	Informational	Variable feeBase can not be changed

Table 2: Findings Overview

TECHNICAL ANALYSIS

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Technical Findings

General Observations

The CivVault Smart Contract is designed to securely deposit user tokens by locking them into any available pool. The owner of the contract can add new pools and finalize users' deposit and withdrawal requests. Finalizing of deposit/withdrawal can be done only if a specific interval has passed. The user is able to cancel a pending deposit.

To deposit tokens into a pool, users must hold the required amount of guarantee tokens, which are determined by the prices in Uniswap. Users receive representation tokens in exchange for depositing their tokens. When the owner processes a user's deposit, the staked tokens are sent to the Invest Fund, and the user's representation tokens are replaced with fund representation tokens. This process involves burning the old tokens and minting new ones.

When the owner processes a user's withdrawal, the fund representation tokens are burned, and guarantee tokens have a fixed lock time.

The following actors can interact with the contract:

- Owner - the owner of the Vault contract; is set at the contract creation time
    - Can add new pools
    - Can set pool fees
    - Can set the vault's fee collection frequency
    - Can set the vault's deposit duration
    - Can set the vault's withdrawal duration
    - Can set new vault’s withdrawal addresses. These addresses are used for withdrawing fees
    - Can update pool NAV
    - Can pause/unpause any vault
    - Can set max deposit for a vault
    - Can process users' deposits
    - Can process users' withdrawals from a vault
    - Can collect fees
    - Can pause/unpause the contract
    - Can withdraw all ETH from the contract
    - Can withdraw any ERC-20 token that was sent to the contract (except the one used in pools)
- Users without deposited tokens - users that didn't deposit any tokens to any pool yet
    - Can deposit tokens
- Users that deposited tokens that weren't sent to the Fund yet
    - Can withdraw their pending tokens (withdrawPending)
- Users that deposited tokens (whose tokens were already sent to the Invest Fund)
    - Can withdraw their tokens from a vault (withdraw or withdrawAll)
- Users that requested a withdrawal from the Vault

    - No additional actions available
- Any actor can update the prices of any pair in any pool using the update function

During the audit, the development team was very responsive in explaining the logic and fast in fixing the found issues.

The readability of the code is good, and it was improved in the process by adding in-code documentation.

The administration of the contract is done by the Owner. The governance contract can be set as Owner to achieve additional security and decentralization.

Among the found vulnerabilities the most impactful were issues related to DoS attacks and small mistakes in the code.

Overall, most of the found issues were effectively addressed with minor modifications, while a few required some architecture adjustment. The modified version was checked in the next phase of the audit.

With implemented changes in architecture, the system relies on the represent token to process users' balances and guarantee token logic is separated. The updated code had several new vulnerabilities, including incorrect processing of the represent and the guarantee tokens.

During the audit all found vulnerabilities were remediated.

INCORRECT STARTINGEPOCH CALCULATION

Finding ID: FYEO-CV-01
Severity: Critical
Status: Remediated

Description

In the `claimGuaranteeToken` function, the `_startingEpochFinal` value is set to the last processed epoch. It allows claiming tokens from the last epoch indefinitely.

Proof of Issue

File name: contracts/CivVault.sol

Line number:

```
    function claimGuaranteeToken(uint256 _pid) external {
...
        for( uint256 i = user.startingEpoch; i < curEpoch; i++ ) {
            for (uint256 j = _startingPoint; j < user.locksCount[i]; j++) {
                if (block.timestamp >= depositTime[_pid][_msgSender()][i][j] +
vault.lockPeriod) {
...
                    _startingEpochFinal = i;
                }
            }
        }
...
        user.startingEpoch = _startingEpochFinal;
...
    }
```

Severity and Impact Summary

The `guaranteeToken` of other users can be stolen.

Recommendation

It is recommended to set the epoch to the next one after the final processed: `_startingEpochFinal = i + 1;`.

WITHDRAWPENDING DOESN'T BURN FUNDREPRESENTTOKEN

Finding ID: FYEO-CV-02
Severity: Critical
Status: Remediated

Description

The `fundRepresentToken` is minted on `deposit` but it is not burned in reverse operation `withdrawPending`.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 663

```
function deposit(uint256 _pid, uint256 _amount)
...
    pool.fundRepresentToken.mint(_msgSender(), addShare);
```

Severity and Impact Summary

The malicious actor can withdraw tokens deposited in the current epoch, and then use `fundRepresentToken` to claim tokens deposited in previous epochs by other users.

Recommendation

It is recommended to burn `fundRepresentToken` on `withdrawPending`. Also, consider minting `fundRepresentToken` after `depositToFund`, for example: 1. deposit - user stores unclaimed `fundRepresentToken` for the current epoch 2. depositToFund - owner increments epoch 3. claimRepresentToken - user mints unclaimed `fundRepresentToken` for epochs that are less than a current

INCORRECT CURRENTWITHDRAW CALCULATION

Finding ID: FYEO-CV-03
Severity: High
Status: Remediated

Description

The `withdrawAll` function doesn't check if the user already called it (or called `withdraw` previously) and adds the user's balance to `currentWithdraw`.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 484

```
    function withdrawAll(uint256 _pid) public nonReentrant whenNotPaused {
        ...
        uint256 currentBalance = getBalanceOfUser(_pid,_msgSender());
        ...
        withdrawInfo[_pid][_msgSender()] = currentBalance;
        vault.currentWithdraw += currentBalance;
        ...
    }
```

Severity and Impact Summary

The issue creates two problems: - possible DOS if the currentWithdraw became larger than the balance - `NAV` and `totalShares` may be calculated incorrectly

Recommendation

It is recommended to validate the value that is added to the `currentWithdraw`.

Incorrect starting value in loop in claimGuaranteeToken

Finding ID: FYEO-CV-04
Severity: **High**
Status: **Remediated**

Description

The `claimGuaranteeToken` function iterates through guarantee token deposits but skips some items. In each epoch the loop begins from the same `_startingPoint`, but when deposits are added - the starting index is 0.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 572

```
        for( uint256 i = user.startingEpoch; i < curEpoch; i++ ) {
            for (uint256 j = _startingPoint; j < user.locksCount[i]; j++) {
                if (block.timestamp >= depositTime[_pid][_msgSender()][i][j] +
vault.lockPeriod) {
                    actualReward += depositQuantity[_pid][_msgSender()][i][j];
                    _startingPointFinal = j;
                    _startingEpochFinal = i;
                }
            }
        }
```

Severity and Impact Summary

Some guarantee tokens may be lost.

Recommendation

It is recommended to start the inner loop from `0`.

Incorrect Withdraw Amount Calculation for the Second Withdraw

Finding ID: FYEO-CV-05
Severity: High
Status: Remediated

Description

The `withdrawAll` function uses `fundRepresentToken` balance as a withdraw amount and transfers those tokens from user's balance. When the user calls the function the second time for the same epoch, it will override the withdraw amount instead of increasing it.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 722

```
    function withdrawAll(uint256 _pid) external nonReentrant whenNotPaused {
...
        uint256 currentBalance = getBalanceOfUser(_pid,_msgSender());
...
        uint256 addWithdraw = currentBalance -
withdrawInfo[_pid][_msgSender()][curWithdrawEpoch];
        vault.currentWithdraw += addWithdraw;
        withdrawInfo[_pid][_msgSender()][curWithdrawEpoch] = currentBalance;
...
        pool.fundRepresentToken.safeTransfer(address(this), shareAmount);
...
    }
```

Severity and Impact Summary

The issue may cause asset loss.

Recommendation

It is recommended to add the withdraw amount:

```
        vault.currentWithdraw += currentBalance;
        withdrawInfo[_pid][_msgSender()][curWithdrawEpoch] += currentBalance;
```

POSSIBLE DOS IN WITHDRAWFROMVAULT

Finding ID: FYEO-CV-06
Severity: High
Status: Remediated

Description

The `withdrawUsers` array can grow indefinitely and at some point GAS, required for the execution of `withdrawFromVault`, will reach the block limit, preventing its execution.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 481

```
withdrawUsers[_pid].push(_msgSender()
```

Line number: 522

```
for( uint256 i = 0; i < withdrawUsers[_pid].length; i++ ) {
```

Severity and Impact Summary

The `withdrawFromVault` function will be blocked. The current implementation can process approximately 400 users in one transaction.

’Users’funds will be locked in the contract

Recommendation

It is recommended to implement the ability to process `withdrawUsers` array in several transactions.

Possible DOS when burning represent tokens

Finding ID: FYEO-CV-07
Severity: High
Status: Remediated

Description

Represent tokens can be manually transferred by users, which can disable `depositToFund` and `withdrawFromVault` functions.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 406

A user receives `representToken` on `deposit`:

```
    function deposit(uint256 _pid, uint256 _amount) {
...
        pool.representToken.mint(_msgSender(), _amount);
```

Line number: 442

And can transfer them before `depositToFund` is executed, resulting in failure during burning:

```
function depositToFund(uint256 _pid, address invest) {
...
   pool.representToken.burn(user, amount);
```

Line number: 443

A user receives `fundRepresentToken` on `depositToFund`:

```
    function deposit(uint256 _pid, uint256 _amount) {
...
        pool.fundRepresentToken.mint(user, xTokenAmount);
```

Line number: 540

And can transfer them before `withdrawFromVault` is executed, resulting in failure during burning:

```
function withdrawFromVault(uint256 _pid) {
...
   pool.fundRepresentToken.burn(user, xTokenAmount);
```

Severity and Impact Summary

All participants of `depositToFund` and `withdrawFromVault` will be blocked if one of them manually transfers represent tokens.

Recommendation

It is recommended to either block the ability to transfer represent tokens or change the logic of processing deposits and withdrawals.

LOGIC THAT DEPENDS ON USER BALANCE CAN BE MANIPULATED

Finding ID: FYEO-CV-08
Severity: Medium
Status: Remediated

Description

User balance is determined by their balance of `fundRepresentToken`. Tokens can be transferred between different accounts and some operations change contract state depending on the user's balance.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 383

```
    function getBalanceOfUser(uint256 _pid, address _user)
        public
        view
        returns (uint256)
    {
        PoolInfo memory pool = poolInfo[_pid];
        if( pool.totalShares == 0) return 0;
        uint256 currentNAV = pool.NAV - pool.unpaidFee;
        uint256 balance = pool.fundRepresentToken.balanceOf(_user);
        return currentNAV * balance / pool.totalShares;
    }
```

Severity and Impact Summary

The problem may cause DoS on `withdrawFromVault` and incorrect `guaranteeAmount` calculation on `withdrawPending`.

Recommendation

It is recommended to handle changes in the Represent token balance. For example, the balance can be frozen when a user initiates withdrawal.

NAV IN DEPOSIT MAY INCLUDE FEES

Finding ID: FYEO-CV-09
Severity: Medium
Status: Remediated

Description

The `deposit()` function uses the `NAV` variable that may also include unprocessed fees.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 655

```
function deposit(uint256 _pid, uint256 _amount)
...
  pool.valuePerShare = pool.NAV * 10 **18 / pool.totalShares;
```

Severity and Impact Summary

Share value may be calculated incorrectly.

Recommendation

It is recommended to subtract fees when using NAV.

NAV IN WITHDRAWALL MAY INCLUDE FEES

Finding ID: FYEO-CV-10
Severity: Medium
Status: Remediated

Description

The `withdrawAll()` function uses the `NAV` variable that may also include unprocessed fees.

Proof of Issue

File name: contracts/CivVault.sol

Line number:

```
    function withdrawAll(uint256 _pid) external nonReentrant whenNotPaused {
...
        pool.valuePerShare = pool.NAV * 10 **18 / pool.totalShares;
...
    }
```

Severity and Impact Summary

Share value may be calculated incorrectly.

Recommendation

It is recommended to return `shareAmount` from `getBalanceOfUser` instead of calculating it in `withdrawAll`:

```
    function getBalanceOfUser(uint256 _pid, address _user)
        public
        view
        returns (uint256, uint256)
    {
        PoolInfo memory pool = poolInfo[_pid];
        if( pool.totalShares == 0) return 0;
        uint256 currentNAV = pool.NAV - pool.unpaidFee;
        uint256 shares = pool.fundRepresentToken.balanceOf(_user);
        uint256 balance = currentNAV * shares / pool.totalShares;
        return (balance, shares);
    }

   function withdrawAll(uint256 _pid) external nonReentrant whenNotPaused {
...
        uint256 (currentBalance, shareAmount) = getBalanceOfUser(_pid,_msgSender());
...
   }
```

Possible DOS in depositToFund

Finding ID: FYEO-CV-11
Severity: Medium
Status: Remediated

Description

The `depositUsers` array can grow indefinitely and at some point GAS, required for the execution of `depositToFund`, may reach the block limit, preventing its execution.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 401

```
depositUsers[_pid].push(_msgSender());
```

Line number: 434

```
uint256 length = depositUsers[_pid].length;
...
for(uint256 i = 0; i < length; i++) {
```

Severity and Impact Summary

The `depositToFund` function will be blocked. The current implementation can process approximately 400 users in one transaction.

Users will be able to withdraw pending deposits using `withdrawPending` function.

Recommendation

It is recommended to implement the ability to process `depositUsers` array in several transactions.

RACE CONDITION ON UPDATING NAV

Finding ID: FYEO-CV-12
Severity: Medium
Status: Remediated

Description

The `updateNAV()` function sets the new value for `NAV`. This function is called by an owner. The `deposit()` function increments the current `NAV` value. This function is called by a user. The `updateNAV()` function may be executed without handling recent deposits.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 399

```
function updateNAV(uint256 _pid, uint256 _newNAV) external onlyOwner {
...
  pool.NAV = _newNAV;
```

Line number: 657

```
function deposit(uint256 _pid, uint256 _amount)
...
  pool.NAV += _amount;
```

Severity and Impact Summary

The `NAV` may be updated incorrectly. It can be mitigated by pausing the contract before executing `updateNAV()`.

Recommendation

Possible solutions: - Change parameters of `updateNAV()` to provide a difference with the previous value to add to the `NAV` variable instead of setting it. - Add new storage variables to be updated in the `deposit()` function, and update actual pool parameters on `depositToFund`.

WATERMARK IS NOT UPDATED ON DEPOSIT

Finding ID: FYEO-CV-13
Severity: Medium
Status: Remediated

Description

The `NAV` variable is updated on `deposit`, but the `watermark` is updated on `depositToFund`. The `updateNAV()` function expects those variables to be updated simultaneously.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 657

```
function deposit(uint256 _pid, uint256 _amount)
...
pool.NAV += _amount;
```

Line number: 700

```
function depositToFund(uint256 _pid, address invest)
...
pool.watermark += currentDeposit;;
```

Severity and Impact Summary

The calculation of the fee may be incorrect.

Recommendation

It is recommended to update the `watermark` on `deposit` and also in reverse operation - `withdrawPending`.

WITHDRAWPENDING DOESN'T UPDATES SOME STORAGE VARIABLES

Finding ID: FYEO-CV-14
Severity: Medium
Status: Remediated

Description

`withdrawPending` is a reverse operation for the `deposit`, but some storage variables that are updated on `deposit`, are not changed after `withdrawPending`. It includes: - pool.valuePerShare - pool.NAV - pool.totalShares - vault.curDepositUser

Proof of Issue

File name: contracts/CivVault.sol

Line number: 652

```
    function deposit(uint256 _pid, uint256 _amount)
        external
        nonReentrant
        whenNotPaused
    {
...
            vault.curDepositUser++;
...
            if( pool.totalShares == 0 ) {
                pool.valuePerShare = _amount;
            } else {
                pool.valuePerShare = pool.NAV * 10 **18 / pool.totalShares;
            }
            pool.NAV += _amount;
             uint256 addShare = _amount * 10 ** 18 / pool.valuePerShare;
            pool.totalShares += addShare;
...
    }
```

Severity and Impact Summary

The issue may cause incorrect balances calculation.

Recommendation

It is recommended to update the pool variables in the `withdrawPending` function.

FEE CALCULATION IS INCORRECT IF NAV INCREASING SLOWLY

Finding ID: FYEO-CV-15

Severity: Low

Status: Remediated

Description

The `unpaidFee` calculation produces a different result the same total increase of `NAV` depending on the size of the increase in one operation.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 287

```
    function updateNAV(uint256 _pid, uint256 _newNAV) public onlyOwner {
        require(poolInfo.length > _pid, "Pool does not exist");
        PoolInfo storage pool = poolInfo[_pid];
        pool.NAV = _newNAV;
        if( pool.watermark < _newNAV) {
            uint256 actualFee = (_newNAV - pool.watermark) * pool.fee / feeBase;
            pool.watermark = _newNAV - actualFee;
            pool.unpaidFee += actualFee;
        }
    }
```

Example 1: - starting values: fee=100, NAV=0, watermark=0, unpaidFee=0 - call updateNAV once with a value 100 - resulting values: NAV=100, watermark=99, unpaidFee=1

Example 2: - starting values: fee=100, NAV=0, watermark=0, unpaidFee=0 - call updateNAV with a value 50 - call updateNAV with a value 100 - resulting values: NAV=100, watermark=100, unpaidFee=0

Severity and Impact Summary

The fee calculation may be incorrect.

The function is intended to be used with 10^18 precision, so the issue is unlikely to occur.

Recommendation

It is recommended to update the `watermark` only if `actualFee > 0`

INCOMPLETE EVENTS

Finding ID: FYEO-CV-16
Severity: Low
Status: Remediated

Description

Some events don't specify the pool on which the changes were applied.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 115

```
event Deposit(address indexed user, uint256 indexed depositTime, uint256 amount);
event Withdraw(address indexed user, uint256 indexed withdrawTime, uint256 amount);
```

Line number: 130

```
event SetFee(uint256 oldFee, uint256 newFee);
event SendFeeWithOwner(address treasuryAddress,uint256 feeAmount);
```

Severity and Impact Summary

Event handlers will receive incomplete information.

Recommendation

It is recommended to add `poolId` parameter in the events mentioned above.

INCORRECT CHECK WHEN DELETING USERINFO

Finding ID: FYEO-CV-17
Severity: Low
Status: Remediated

Description

The code has an incorrect check that will never be true.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 536

```
function withdrawFromVault(uint256 _pid) {
...
  if( amount > 0) {
...
    UserInfo storage curUserInfo = userInfo[_pid][user];
    curUserInfo.userShare -= addShare;
    if (amount == 0) {
        delete userInfo[_pid][user];
```

Severity and Impact Summary

With the incorrect check, the storage will not be cleansed, resulting in higher execution costs.

Recommendation

It is recommended to replace the check with `curUserInfo.userShare == 0`.

Also, to save up the GAS, consider refactoring

```
if (curUserInfo.userShare == addShare) {
    delete userInfo[_pid][user];
} else {
    curUserInfo.userShare -= addShare;
}
```

INCORRECT TYPE FOR LOCKSCOUNT

Finding ID: FYEO-CV-18
Severity: Low
Status: Remediated

Description

The `locksCount` variable is declared as an array but used as a map.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 97

```
uint256[] locksCount;
```

Line number: 650

```
user.locksCount[curEpoch]++;
```

Severity and Impact Summary

Accessing an unexistent item of an array will cause an exception that will revert the transaction.

Recommendation

It is recommended to replace the type with `mapping(uint256 => uint256)`.

INCORRECT WITHDRAW BALANCE CHECK

Finding ID: FYEO-CV-19
Severity: Low
Status: Remediated

Description

The balance check in `withdrawFromVault` doesn't include `unpaidFee` and is not allowed to be equal to the `currentWithdraw`.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 517

```
require(pool.lpToken.balanceOf(address(this)) > vault.currentWithdraw, "not enough
amount to withdraw");
```

Severity and Impact Summary

In the case when `unpaidFee = 0`, users will not be able to withdraw everything from the vault.

Recommendation

It is recommended to change the check to `pool.lpToken.balanceOf(address(this)) >= vault.currentWithdraw + pool.unpaidFee`.

Missing zero-address check

Finding ID: FYEO-CV-20
Severity: Low
Status: Remediated

Description

Some setter functions do not check for zero addresses.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 197, 198, 203

The array of `addPoolParam._withdrawAddresses`, `addPoolParam._representToken` address and `addPoolParam._fundRepresentToken` address:

```
    function addPool(
        AddPoolParam memory addPoolParam
    ) {
...
        representToken: addPoolParam._representToken,
        fundRepresentToken: addPoolParam._fundRepresentToken,
...
        withdrawAddress: addPoolParam._withdrawAddresses,
```

Line number: 261

The array of `_newAddress`:

```
    function setWithdrawAddress(uint256 _pid, address[] memory _newAddress) external
onlyOwner {
        require(_newAddress.length == 2, "Withdraw Addresses length must be 2");
        require(poolInfo.length > _pid, "Pool does not exist");
        poolInfo[_pid].withdrawAddress = _newAddress;
    }
```

Line number: 433

The `invest` address:

```
    function depositToFund(uint256 _pid, address invest)
...
        pool.lpToken.safeTransfer(invest, vault.currentDeposit);
```

Severity and Impact Summary

Zero address may lead to financial losses in the case when assets are transferred to it.

Recommendation

It is recommended to check that the address is not zero.

Reentrancy in collectFee

Finding ID: FYEO-CV-21
Severity: Low
Status: Remediated

Description

The `sendFee` changes state after an external call.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 551

```
    function sendFee(uint256 _pid, uint256 feeAmount) internal {
        PoolInfo storage pool = poolInfo[_pid];
        pool.NAV -= feeAmount;
        pool.lpToken.safeTransfer(pool.withdrawAddress[0], feeAmount / 2);
        pool.lpToken.safeTransfer(pool.withdrawAddress[1], feeAmount / 2);
        emit SendFeeWithOwner(pool.withdrawAddress[0], feeAmount / 2);
        emit SendFeeWithOwner(pool.withdrawAddress[1], feeAmount / 2);
        pool.unpaidFee = 0;
    }
```

Line number: 568

```
        sendFee(_pid, poolInfo[_pid].unpaidFee);
        vault.lastCollectFee = block.timestamp;
```

Severity and Impact Summary

The external call is done to a verified token, so the problem is unlikely to occur.

Recommendation

It is recommended to change the contract state before the external call: - perform `vault.lastCollectFee = block.timestamp;` before executing `sendFee()` - emit events and set `unpaidFee` before `safeTransfer`

THE SIZE OF WITHDRAWADDRESS ARRAY IS NOT VALIDATED IN ADDPOOL

Finding ID: FYEO-CV-22
Severity: Low
Status: Remediated

Description

The contract logic requires the array to be the size of 2, but it is not validated in the `addPool` function.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 203

```
withdrawAddress: addPoolParam._withdrawAddresses,
```

Line number: 258

```
    function setWithdrawAddress(uint256 _pid, address[] memory _newAddress) external
onlyOwner {
        require(_newAddress.length == 2, "Withdraw Addresses length must be 2");
        require(poolInfo.length > _pid, "Pool does not exist");
        poolInfo[_pid].withdrawAddress = _newAddress;
    }
```

Severity and Impact Summary

Some operations will fail until the correct array is set using `setWithdrawAddress`

Recommendation

It is recommended to check the size of `withdrawAddress` array.

THE WITHDRAWERC20 IS ABLE TO TRANSFER GUARANTEETOKEN AND LPTOKEN

Finding ID: FYEO-CV-23
Severity: Low
Status: Remediated

Description

Tokens that are used in pools (guaranteeToken and lpToken) are available for withdrawal in `withdrawERC20` function.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 594

```
    function withdrawERC20(IERC20 _tokenContract) external onlyOwner {
        _tokenContract.safeTransfer(
            _msgSender(),
            _tokenContract.balanceOf(address(this))
        );
    }
```

Severity and Impact Summary

The manual withdrawal of pools token may affect the logic of rewards distribution. This operation is only available for the owner.

Recommendation

It is recommended to check that tokens available to withdraw are not used in pools.

USAGE OF NON-EXISTING ARRAY IN GETUNCLAIMEDTOKENEPOCHS

Finding ID: FYEO-CV-24
Severity: Low
Status: Remediated

Description

The `getUnclaimedTokenEpochs` function accesses non-existing items of the `_epochs` array.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 821

```
    function getUnclaimedTokenEpochs(uint256 _pid) external view returns(uint256[]
memory _epochs) {
        require(poolInfo.length > _pid, "Pool does not exist");
        PoolInfo memory pool = poolInfo[_pid];
        uint256 epochLen = 0;
        uint256 curWithdrawEpoch = pool.currentWithdrawEpoch;
        for(uint i = 0; i < curWithdrawEpoch; i++) {
            if( withdrawInfo[_pid][_msgSender()][i] > 0 ) {
                _epochs[epochLen++] = i;
            }
        }
        return _epochs;
    }
```

Severity and Impact Summary

The execution will be reverted.

Recommendation

It is recommended to create the array before usage: `_epochs = new uint256[](curWithdrawEpoch);`.

VULNERABLE OPENZEPPELIN VERSION

Finding ID: FYEO-CV-25
Severity: **Low**
Status: **Remediated**

Description

The vulnerable version of `openzeppelin` library is used. The audited contract doesn't seem to be affected by those issues.

Proof of Issue

File name: package.json

Line number: 7

```
    "@openzeppelin/contracts": "^4.6.0",
```

Severity and Impact Summary

The following vulnerabilities are known to be present in the used version: - SignatureChecker may revert on invalid EIP-1271 signers - ERC165Checker may revert instead of returning false - GovernorVotesQuorumFraction updates to quorum may affect past defeated proposals - OpenZeppelin Contracts vulnerable to ECDSA signature malleability - Cross chain utilities for Arbitrum L2 see EOA calls as cross chain calls - ERC165Checker unbounded gas consumption

Recommendation

It is recommended to upgrade the OpenZeppelin library to version `4.7.3`.

LPTOKEN CAN BE WITHDRAWN BEFORE WITHDRAWFROMVAULT

Finding ID: FYEO-CV-26
Severity: Low
Status: Remediated

Description

The `lpToken` is withdrawn in the `getWithdrawedToken` function, which allows to withdraw from the `currentWithdrawEpoch`.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 800

```
    function getWithdrawedToken(uint256 _pid, uint256[] memory epochs) external
nonReentrant whenNotPaused {
        require(poolInfo.length > _pid, "Pool does not exist");
        PoolInfo memory pool = poolInfo[_pid];
        uint256 unclaimedAmount = 0;
        for(uint i = 0; i < epochs.length;i++) {
            unclaimedAmount += withdrawInfo[_pid][_msgSender()][epochs[i]];
            withdrawInfo[_pid][_msgSender()][epochs[i]] = 0;
        }
        pool.lpToken.safeTransfer(_msgSender(), unclaimedAmount);
    }
```

Severity and Impact Summary

The withdrawal algorithm works incorrectly.

Recommendation

It is recommended to verify that `epochs[i] < currentWithdrawEpoch`.

CivVault has no unit tests

Finding ID: FYEO-CV-27
Severity: Informational
Status: Remediated

Description

The CivVault contract has no unit tests and existing tests don't work.

Proof of Issue

```
$ npx hardhat test

...

  4 passing (10s)
  4 failing
```

Severity and Impact Summary

Unit tests help to verify the correctness of the code and protect it from regression.

Recommendation

It is recommended to cover the CivVault contract with unit tests and fix existing tests for other contracts.

DEPOSIT INFO STORAGE CAN BE OPTIMIZED

Finding ID: FYEO-CV-28
Severity: Informational
Status: Remediated

Description

Deposit information is stored in several similar mappings.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 125

```
    /// @notice Each Pools Info of each user that request for deposit
    mapping(uint256 => mapping(address => mapping(uint256 => uint256))) public
depositInfo;
    /// @notice Each Pools Info of each user that request for withdraw
    mapping(uint256 => mapping(address => mapping(uint256 => uint256))) public
withdrawInfo;
    /// @notice Each Pools deposit times of each user
    mapping(uint256 => mapping(address => mapping(uint256 => mapping(uint256 =>
uint256)))) public depositTime;
    /// @notice Each Pools deposit guarantee token amount of each user
    mapping(uint256 => mapping(address => mapping(uint256 => mapping(uint256 =>
uint256)))) public depositQuantity;
```

Severity and Impact Summary

Accessing several mappings consumes more GAS.

Recommendation

It is recommended to combine deposit information in the struct:

```
   struct DepositParams {
        uint256 depositInfo;
        uint256 withdrawInfo;
        mapping(uint256 => uint256) depositTime;
        mapping(uint256 => uint256) depositQuantity;
    }

    mapping(uint256 => mapping(address => mapping(uint256 => DepositParams ))) public
depositParams;

...
    DepositParams storage dp = depositParams[_pid][_msgSender()][curEpoch];
```

Distinguishing deposits in the same epoch is unnecessary

Finding ID: FYEO-CV-29
Severity: **Informational**
Status: **Remediated**

Description

The `depositQuantity` and the `depositTime` variables allow distinguishing between deposits in the same epoch, but when they are processed - the loop is used to through all items.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 573

```
            for (uint256 j = _startingPoint; j < user.locksCount[i]; j++) {
                if (block.timestamp >= depositTime[_pid][_msgSender()][i][j] +
vault.lockPeriod) {
                    actualReward += depositQuantity[_pid][_msgSender()][i][j];
                    _startingPointFinal = j;
                    _startingEpochFinal = i;
                }
            }
```

Severity and Impact Summary

The logic is unnecessarily complicated.

Recommendation

It is recommended to sum deposit values in one `depositQuantity` variable per epoch and store the last time of deposit in the epoch.

GAS USAGE OPTIMIZATION

Finding ID: FYEO-CV-30
Severity: Informational
Status: Remediated

Description

Some code can be optimized to decrease GAS usage.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 313

Conditional return can be performed earlier in `getBalanceOfUser`:

```
        PoolInfo memory pool = poolInfo[_pid];
        UserInfo memory user = userInfo[_pid][_user];
        uint256 currentNAV = pool.NAV - pool.unpaidFee;
        if( pool.totalShares == 0) return 0;
```

Optimization:

```
        PoolInfo memory pool = poolInfo[_pid];
        if( pool.totalShares == 0) return 0;
        UserInfo memory user = userInfo[_pid][_user];
        uint256 currentNAV = pool.NAV - pool.unpaidFee;
```

Line number: 412

In the `depositToFund` function `vault.currentDeposit` and `pool.valuePerShare` are read from storage several times:

```
require(vault.currentDeposit > 0, "no token for deposit to fund");
...
require(pool.lpToken.balanceOf(address(this)) >= vault.currentDeposit);
...
pool.valuePerShare = vault.currentDeposit;
...
```

To optimize, store values in local variables:

```
uint256 currentDeposit = vault.currentDeposit;
uint256 valuePerShare = pool.valuePerShare;
```

Line number: 505

In the `withdrawFromVault` function `vault.currentWithdraw` and `pool.valuePerShare` are read from storage several times:

```
require(pool.lpToken.balanceOf(address(this)) > vault.currentWithdraw, "not enough
amount to withdraw");
...
```

```
uint256 addShare = amount * 10 ** 18 / pool.valuePerShare;
uint256 xTokenAmount = amount * 10 ** 24 / pool.valuePerShare;
...
```

To optimize, store values in local variables:

```
uint256 currentWithdraw = vault.currentWithdraw;
uint256 valuePerShare = pool.valuePerShare;
```

Line number: 551

The `sendFee` function reads the content of `pool.withdrawAddress` twice:

```
pool.lpToken.safeTransfer(pool.withdrawAddress[0], feeAmount / 2);
pool.lpToken.safeTransfer(pool.withdrawAddress[1], feeAmount / 2);
emit SendFeeWithOwner(pool.withdrawAddress[0], feeAmount / 2);
emit SendFeeWithOwner(pool.withdrawAddress[1], feeAmount / 2);
```

To optimize, store values in local variables:

```
address addr0 = pool.withdrawAddress[0];
address addr1 = pool.withdrawAddress[1];
pool.lpToken.safeTransfer(addr0, feeAmount / 2);
pool.lpToken.safeTransfer(addr1, feeAmount / 2);
emit SendFeeWithOwner(addr0, feeAmount / 2);
emit SendFeeWithOwner(addr1, feeAmount / 2);
```

Severity and Impact Summary

Unoptimized functions will be more expensive.

Recommendation

It is recommended to implement the improvements mentioned above.

INCORRECT AMOUNT CHECK IN WITHDRAWPENDING

Finding ID: FYEO-CV-31
Severity: Informational
Status: Remediated

Description

The condition checks that the unsigned variable is `>= 0` which is always true. The error message is not correct. The storage variable is read twice.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 741

```
        require( depositInfo[_pid][_msgSender()][curEpoch] >= 0, "Amount exceeds
pending balance");
        uint256 amount = depositInfo[_pid][_msgSender()][curEpoch];
```

Severity and Impact Summary

The problem doesn't cause security issues but impacts performance.

Recommendation

It is recommended to optimize the code and fix the condition:

```
        uint256 amount = depositInfo[_pid][_msgSender()][curEpoch];
        require(amount > 0, "No pending balance");
```

INSUFFICIENT FUNCTIONS DOCUMENTATION

Finding ID: FYEO-CV-32
Severity: **Informational**
Status: **Remediated**

Description

It is a good practice to document all external functions in the same way as it is done for `safeTransferETH()`. Third-party tools can be used to generate web documentation from solidity files.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 190

```
    function addPool(
        AddPoolParam memory addPoolParam
    ) public onlyOwner {
```

Severity and Impact Summary

No security risks

Recommendation

It is recommended to document all external functions.

References

- https://docs.soliditylang.org/en/develop/natspec-format.html
- https://github.com/OpenZeppelin/solidity-docgen

MAX_UINT CHECK DOES NOTHING

Finding ID: FYEO-CV-33
Severity: **Informational**
Status: **Remediated**

Description

The code validates that uint256 variable is <= than its maximum value, but this condition will always be true.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 112

```
uint256 public immutable MAX_UINT = 2**256 - 1; // Prevents overflows
```

Line number: 385

```
require(_amount <= uint256(MAX_UINT), "Wrong amount");
```

Line number: 462

```
require(_amount <= uint256(MAX_UINT), "Wrong amount");
```

Severity and Impact Summary

Excessive checks spend GAS without adding any value to the code.

Recommendation

It is recommended to remove the comparison of uint256 with MAX_UINT.

NOT USED EVENTS

Finding ID: FYEO-CV-34
Severity: Informational
Status: Remediated

Description

Several events are declared but not used in the code.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 122

```
    event SetTotalAllocPoint(
        uint256 indexed prevTotalAllocPoint,
        uint256 indexed newTotalAllocPoint
    );
    event SetTokensPerBlock(
        uint256 indexed prevTokensPerBlock,
        uint256 indexed newTokensPerBlock
    );
    event SendFee(address treasuryAddress,uint256 feeAmount);
```

Severity and Impact Summary

No impact on security.

Recommendation

It is recommended to remove unused events.

Possible DOS in claimGuaranteeToken

Finding ID: FYEO-CV-35
Severity: Informational
Status: Remediated

Description

If a user makes a lot of deposits in different epochs, the calculation of the total balance may result in denial of service.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 571

```
        for( uint256 i = user.startingEpoch; i < curEpoch; i++ ) {
            for (uint256 j = _startingPoint; j < user.locksCount[i]; j++) {
                if (block.timestamp >= depositTime[_pid][_msgSender()][i][j] +
vault.lockPeriod) {
                    actualReward += depositQuantity[_pid][_msgSender()][i][j];
                    _startingPointFinal = j;
                    _startingEpochFinal = i;
                }
            }
        }
```

Severity and Impact Summary

The probability of the issue is very low, but it may lock users guarantee tokens.

Recommendation

It is recommended to add to the `claimGuaranteeToken()` function the `maxEpoch` parameter, which will limit the number of epochs processed:

```
function claimGuaranteeToken(uint256 _pid, uint256 maxEpoch) external {
...
  uint256 epoch = Math.min(maxEpoch, pool.currentDepositEpoch);
...
```

Public visibility is set for the function that is not called internally

Finding ID: FYEO-CV-36
Severity: **Informational**
Status: **Remediated**

Description

Public visibility is used for functions that should be accessible from other contracts, via transactions, and from the current contract. Functions that are not meant to be called internally should have external visibility.

Proof of Issue

The issue is actual for the following functions: - withdrawFromVault - withdrawAll - deposit - poolLength - depositToFund - updateNAV - withdraw - withdrawPending

Severity and Impact Summary

External functions may be more GAS efficient, especially with functions that receive a lot of data in parameters.

Recommendation

It is recommended to set external visibility for functions that are not called internally.

References

https://docs.soliditylang.org/en/v0.7.4/contracts.html?highlight=external#visibility-and-getters

SOME FUNCTION CHANGE STATE WITHOUT EMITTING EVENTS

Finding ID: FYEO-CV-37
Severity: **Informational**
Status: **Remediated**

Description

Events emitting is a mechanism that provides a convenient way of handling changes in a smart contract state. Some functions of the CivVault contract change state without emitting events.

Proof of Issue

There are no events for the following functions: - setFeeDuration - setDepositDuration - setWithdrawDuration - setWithdrawAddress - update - updateNAV - setPaused - setMaxDeposit - withdrawFromVault

Severity and Impact Summary

Operations without events will not be processed by event handlers, which can make overall application logic more complex.

Recommendation

It is recommended to add missing events if necessary.

TWO POOLS CAN BE CREATED WITH THE SAME TOKENS

Finding ID: FYEO-CV-38
Severity: Informational
Status: Remediated

Description

The uniqueness of tokens in a pool is not validated.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 191

```
    function addPool(
        AddPoolParam memory addPoolParam
    ) public onlyOwner {
         poolInfo.push(
            PoolInfo({
                lpToken: addPoolParam._lpToken, // Rewardable contract: token for
staking, LP for Funding, or NFT for NFT staking
                representToken: addPoolParam._representToken,
                fundRepresentToken: addPoolParam._fundRepresentToken,
                guaranteeToken: addPoolParam._guaranteeToken,
                NAV: 0, //init = always 0
                totalShares: 0, //init = always 0
                fee: addPoolParam._fee,
                withdrawAddress: addPoolParam._withdrawAddresses,
                unpaidFee: 0,
                watermark: 0,
                valuePerShare: 0 //init = always 0
            })
        );
        vaultInfo.push(
            VaultInfo({
                maxDeposit: addPoolParam._maxDeposit,
                paused: addPoolParam._paused,
                currentDeposit: 0,
                currentWithdraw: 0,
                collectFeeDuration: addPoolParam._feeDuration,
                lastCollectFee: block.timestamp,
                depositDuration: addPoolParam._depositDuration,
                withdrawDuration: addPoolParam._withdrawDuration,
                lastDeposit: block.timestamp,
                lastWithdraw: block.timestamp
            })
        );
```

Severity and Impact Summary

The existence of pools with the same tokens can lead to confusion in their usage and operation. Some functions that rely on token balance may work incorrectly.

Recommendation

It is recommended to validate the uniqueness of tokens when a new pool is created.

VARIABLE FEEBASE CAN NOT BE CHANGED

Finding ID: FYEO-CV-39
Severity: Informational
Status: Remediated

Description

The `feeBase` variable is used only for reading and therefore can be declared as a constant.

Proof of Issue

File name: contracts/CivVault.sol

Line number: 102

```
    uint256 feeBase = 10000;
```

Severity and Impact Summary

The usage of constants saves GAS.

Recommendation

It is recommended to declare `feeBase` variable as a constant or add a setter function.

OUR PROCESS

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Kickoff
Ramp-up
Review
Report
Verify

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages

- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases

- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

The Classification of vulnerabilities

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions

- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations