OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 34	2021-09-24	Zapmore, Hebilicious	Audit Final

Audit Notes

Audit Date	2021-08-10 - 2021-09-20
Auditor/Auditors	Hebilicious, DoD4uFN
Auditor/Auditors Contact Information	tibereum-obelisk@protonmail.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB589965127

Disclaimer

This audit is not financial, investment, or any other kind of advice and is for informational purposes only. This report is not a substitute for doing your own research and due diligence. Obelisk is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Obelisk has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Obelisk is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. The audit is paid by the project but neither the auditors nor Obelisk has any other connection to the project and has no obligations other than to publish an objective report. Obelisk will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Obelisk assumes that the provided information and material were not altered, suppressed, or misleading. This report is published by Obelisk, and Obelisk has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Obelisk.

Obelisk Auditing

Defi is a relatively new concept but has seen exponential growth to a point where there is a multitude of new projects created every day. In a fast-paced world like this, there will also be an enormous amount of scams. The scams have become so elaborate that it’s hard for the common investor to trust a project, even though it could be legit. We saw a need for creating high-quality audits at a fast phase to keep up with the constantly expanding market. With the Obelisk stamp of approval, a legitimate project can easily grow its user base exponentially in a world where trust means everything. Obelisk Auditing consists of a group of security experts that specialize in security and structural operations, with previous work experience from among other things, PricewaterhouseCoopers. All our audits will always be conducted by at least two independent auditors for maximum security and professionalism.

As a comprehensive security firm, Obelisk provides all kinds of audits and project assistance.

Audit Information

The auditors always conducted a manual visual inspection of the code to find security flaws that automatic tests would not find. Comprehensive tests are also conducted in a specific test environment that utilizes exact copies of the published contract.

While conducting the audit, the Obelisk security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Obelisk assesses the risks and assigns a risk level to each section together with an explanatory comment. Take note that the comments from the project team are their opinion and not the opinion of Obelisk.

Table of Content

Project Information

Project Name	Gravity Finance
Description	A first-in-class governance token with true intrinsic value. GFI is backed by an ever-growing amount of bitcoin. Gravity removes the manual & repetitive tasks from DeFi. Every process has been designed from the ground up with automation in mind.
Website	https://www.gravityfinance.io/
Contact	@GravityFi_D
Contact information	@GravityFi_D on TG
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	Polygon

Executive Summary

The audit of Gravity Finance was conducted by two of Obelisks' security experts between the 10th of August 2021 and the 20th of September 2021.

The security issues found while conducting the contract audit were swiftly solved by the project team. There are only a few informational findings left that don't impact the project on a larger scale on the audited implementation. The on-chain analysis has a Low-Risk issue commented on by the project team and a lack of a timelock which also is accompanied by a comment.

The team has not reviewed the UI/UX, logic, team, or tokenomics of the Gravity.

Please read the full document for a complete understanding of the audit.

Summary Table

Audited Part	ID	Severity	Resolved
Redundant Parameter	#0001	Low Risk	YES
Unbound Array Iteration	#0002	Low Risk	YES
Implicit Getters	#0003	Informational	See Comment
No Events Emitted For Changes To Protocol Values	#0004	Informational	Partial
Missing Zero Checks	#0005	Informational	See Comment
Redundant Parameter	#0006	Informational	YES
Unrestricted Protected Function	#0007	Informational	YES
TODO Comment	#0008	Informational	YES
Unused Interfaces	#0009	Informational	YES
Implicit Statements And Declaration	#0010	Informational	YES
Avoidable Repetition	#0011	Informational	YES
Division Before Multiplication	#0012	Medium Risk	YES
Boolean Statement Tautology	#0013	Informational	YES
Missing Inheritance	#0014	Informational	See Comment
Function That Can Withdraw Balance	#0015	Low Risk	See Comment
No Timelock	#0016	Low Risk	In Progress

Introduction

Obelisk was commissioned by Gravity Finance on the 8th of August 2021 to conduct a comprehensive audit of Gravity Finances' contracts. The following audit was conducted between the 10th of August 2021 and the 20th of September 2021. Two of Obelisk's security experts went through the related contracts using industry standards to find if any vulnerabilities could be exploited.

As Obelisk didn't audit the full project, there are some possible vectors of risk that could hide in un-audited contracts. Please see "Appendix A2" for contracts that are part of the project but that are not audited by Obelisk. The project also uses untested custom oracles and modified Uniswap contracts. Farm contracts could pose a risk of Rewards/Maths/Reentrancy. The decision to not audit the full project at the start was a decision made by the project team in order to expedite that audit of the contracts that, according to the project team, handles users' funds.

During the audit of the contracts stated in "Appendix A1", we found 2 Low-Risk issues and 1 Medium Risk issue. These were forwarded to the project team together with multiple informational findings. The project team worked swiftly to resolve these Low and Medium risk issues. Obelisk then re-audited those parts of the project and confirmed that indeed they were solved.

We always do an on-chain analysis of every audited project in order to check that the deployed contracts are exact copies of the audited contracts. During the on-chain analysis, we found a Low-Risk security issue that the project team added a comment to. There also needs to be a timelock added for extra security. The timelock is a work in progress by the team so please see the comment attached to issue #16.

The informational findings are good to know while interacting with the project but don't directly damage the project in its current state, hence it's up to the project team if it's worth solving these issues.

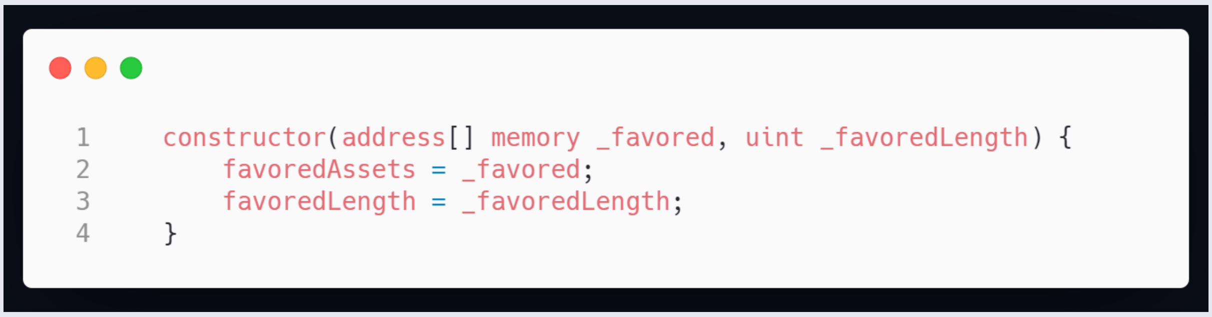
Please see each section of the audit to get a full understanding of the audit.

Findings

Manual Analysis

Redundant Parameter

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0001
LOCATION	PathOracle.sol -> 37-40

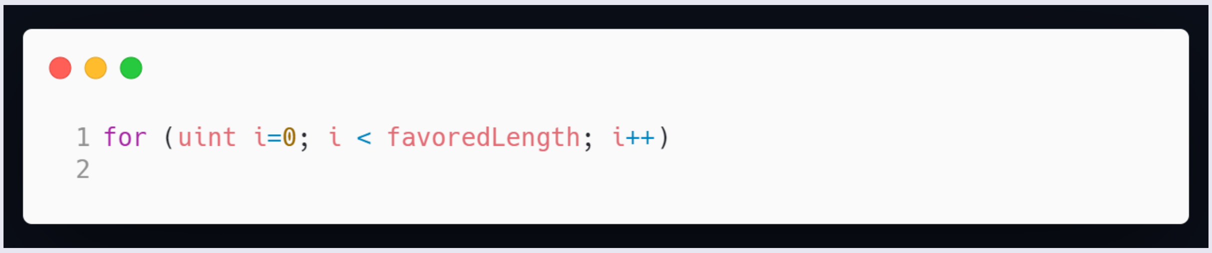


```
constructor(address[] memory _favored, uint _favoredLength) {
    favoredAssets = _favored;
    favoredLength = _favoredLength;
}
```

DESCRIPTION	Parameter *_favoredLength* indicates the length of *_favored* array. This is potentially dangerous since the length is used for loops to iterate over the *_favored* array.
RECOMMENDATION	The length of *_favored* is easily accessible data, remove the parameter and use *_favored.length*.
MITIGATED/COMMENT	The team removed the parameter and replaced it with `.length`.

Unbound Array Iteration

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0002
LOCATION	PathOracle.sol l83, l100

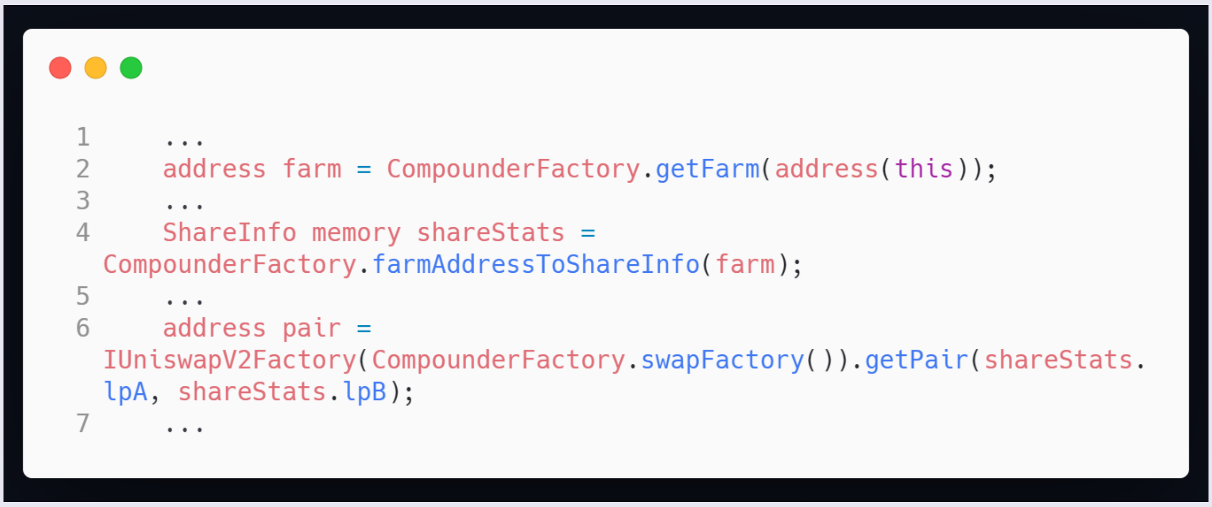


```
for (uint i=0; i < favoredLength; i++)

```

DESCRIPTION	Unbound array iteration can lead to contract execution failures due to the gas limit.
RECOMMENDATION	Add an upper bound limit to the array in the constructor of the function to guarantee contract execution. Alternatively used a fixed size array type for *favoredAssets* if its length will be constant.
MITIGATED/COMMENT	The team added the necessary checks to guarantee contract execution.

Implicit Getters

SEVERITY	Informational
RESOLVED	**NO**
FINDING ID	#0003
LOCATION	Share.sol -> 47-69

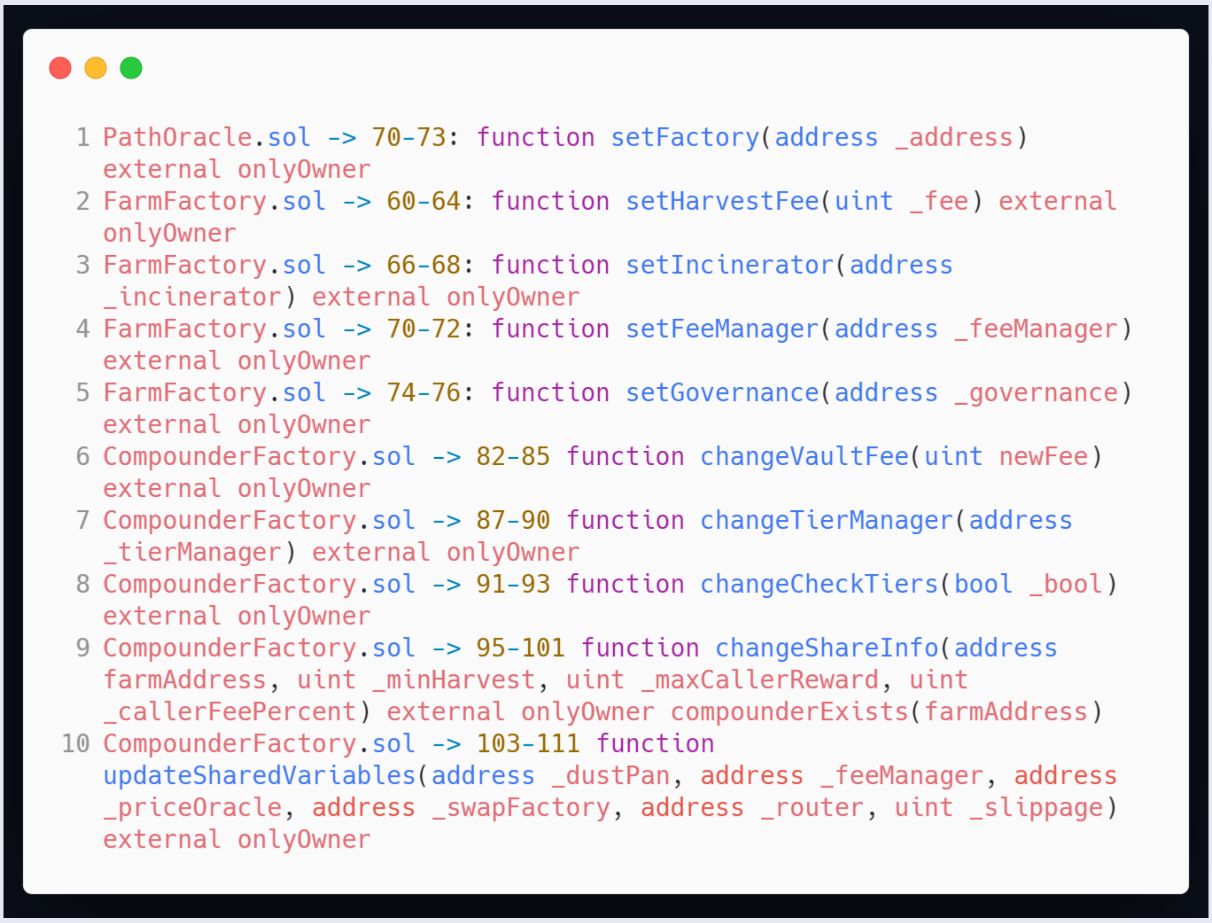


```
    ...
    address farm = CompounderFactory.getFarm(address(this));
    ...
    ShareInfo memory shareStats =
CompounderFactory.farmAddressToShareInfo(farm);
    ...
    address pair =
IUniswapV2Factory(CompounderFactory.swapFactory()).getPair(shareStats.
lpA, shareStats.lpB);
    ...
```

DESCRIPTION	The functions *getFarm*, *farmAddressToShareInfo,* and *swapFactory* are implemented implicitly.
RECOMMENDATION	Avoid using implicit getters for better readability. Be explicit
MITIGATED/COMMENT	Project Team Comment: Accepted no changes made.

No Events Emitted For Changes To Protocol Values

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0004
LOCATION	See code block below

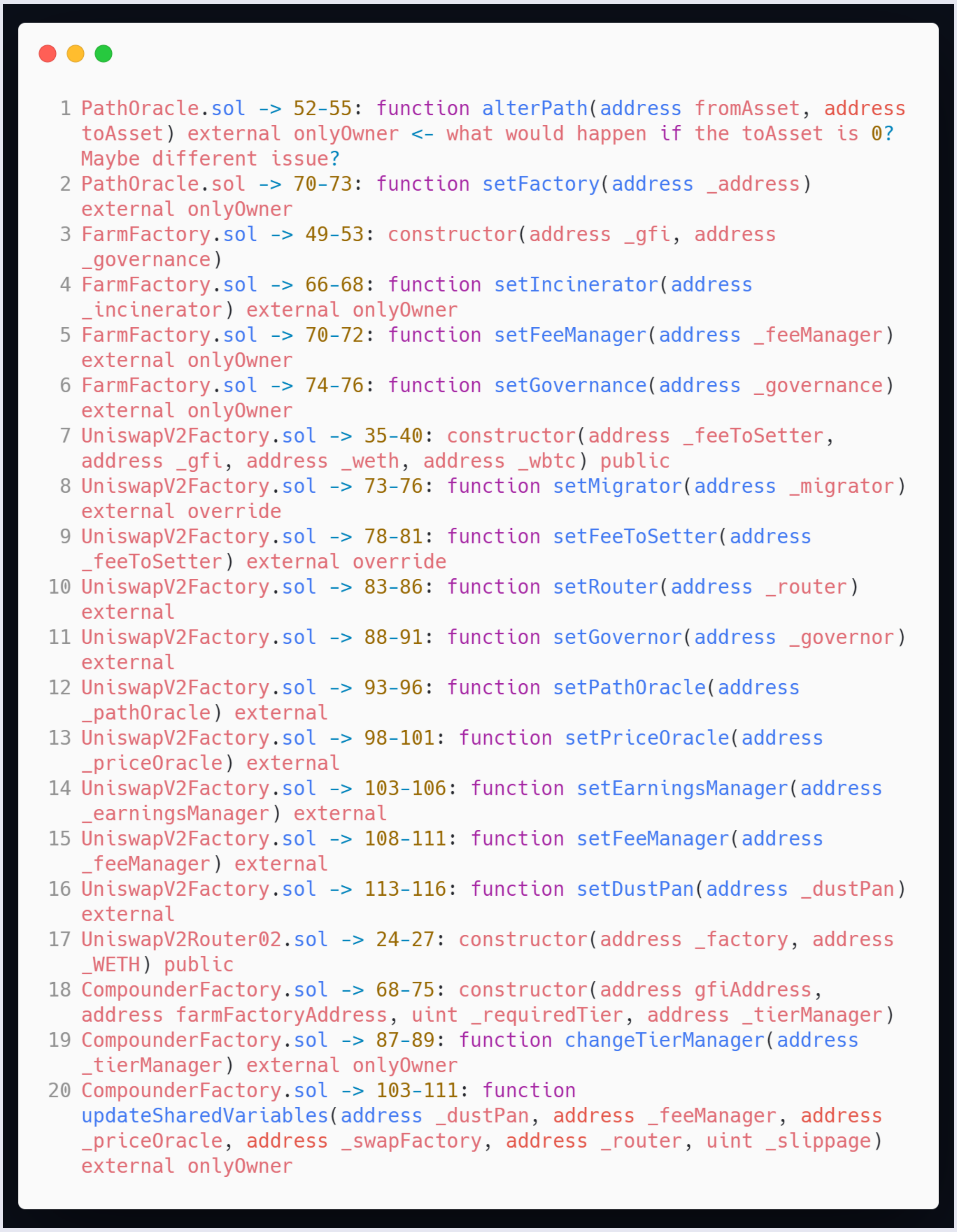


```
PathOracle.sol -> 70-73: function setFactory(address _address) external onlyOwner
FarmFactory.sol -> 60-64: function setHarvestFee(uint _fee) external onlyOwner
FarmFactory.sol -> 66-68: function setIncinerator(address _incinerator) external onlyOwner
FarmFactory.sol -> 70-72: function setFeeManager(address _feeManager) external onlyOwner
FarmFactory.sol -> 74-76: function setGovernance(address _governance) external onlyOwner
CompounderFactory.sol -> 82-85 function changeVaultFee(uint newFee) external onlyOwner
CompounderFactory.sol -> 87-90 function changeTierManager(address _tierManager) external onlyOwner
CompounderFactory.sol -> 91-93 function changeCheckTiers(bool _bool) external onlyOwner
CompounderFactory.sol -> 95-101 function changeShareInfo(address farmAddress, uint _minHarvest, uint _maxCallerReward, uint _callerFeePercent) external onlyOwner compounderExists(farmAddress)
CompounderFactory.sol -> 103-111 function updateSharedVariables(address _dustPan, address _feeManager, address _priceOracle, address _swapFactory, address _router, uint _slippage) external onlyOwner
```

DESCRIPTION	Functions that change important variables should include emit logs such that users can more easily monitor the change.
RECOMMENDATION	Add emit logs to these functions.
MITIGATED/COMMENT	The team added new events.

Missing Zero Checks

SEVERITY	Informational
RESOLVED	**NO**
FINDING ID	#0005
LOCATION	See code block below

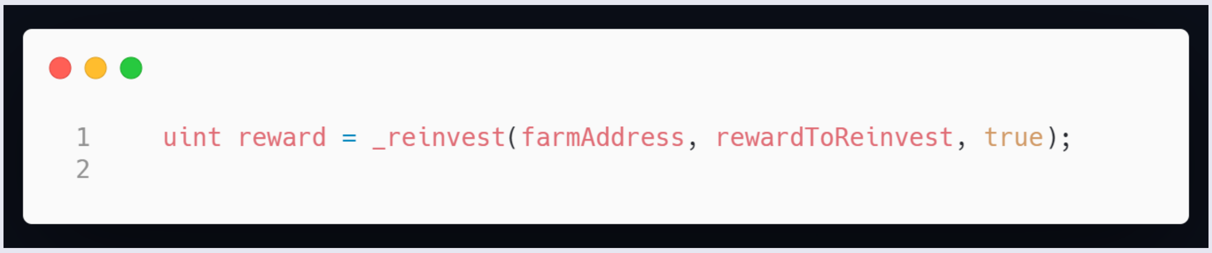


```
PathOracle.sol -> 52-55: function alterPath(address fromAsset, address toAsset) external onlyOwner <- what would happen if the toAsset is 0? Maybe different issue?
PathOracle.sol -> 70-73: function setFactory(address _address) external onlyOwner
FarmFactory.sol -> 49-53: constructor(address _gfi, address _governance)
FarmFactory.sol -> 66-68: function setIncinerator(address _incinerator) external onlyOwner
FarmFactory.sol -> 70-72: function setFeeManager(address _feeManager) external onlyOwner
FarmFactory.sol -> 74-76: function setGovernance(address _governance) external onlyOwner
UniswapV2Factory.sol -> 35-40: constructor(address _feeToSetter, address _gfi, address _weth, address _wbtc) public
UniswapV2Factory.sol -> 73-76: function setMigrator(address _migrator) external override
UniswapV2Factory.sol -> 78-81: function setFeeToSetter(address _feeToSetter) external override
UniswapV2Factory.sol -> 83-86: function setRouter(address _router) external
UniswapV2Factory.sol -> 88-91: function setGovernor(address _governor) external
UniswapV2Factory.sol -> 93-96: function setPathOracle(address _pathOracle) external
UniswapV2Factory.sol -> 98-101: function setPriceOracle(address _priceOracle) external
UniswapV2Factory.sol -> 103-106: function setEarningsManager(address _earningsManager) external
UniswapV2Factory.sol -> 108-111: function setFeeManager(address _feeManager) external
UniswapV2Factory.sol -> 113-116: function setDustPan(address _dustPan) external
UniswapV2Router02.sol -> 24-27: constructor(address _factory, address _WETH) public
CompounderFactory.sol -> 68-75: constructor(address gfiAddress, address farmFactoryAddress, uint _requiredTier, address _tierManager)
CompounderFactory.sol -> 87-89: function changeTierManager(address _tierManager) external onlyOwner
CompounderFactory.sol -> 103-111: function updateSharedVariables(address _dustPan, address _feeManager, address _priceOracle, address _swapFactory, address _router, uint _slippage) external onlyOwner
```

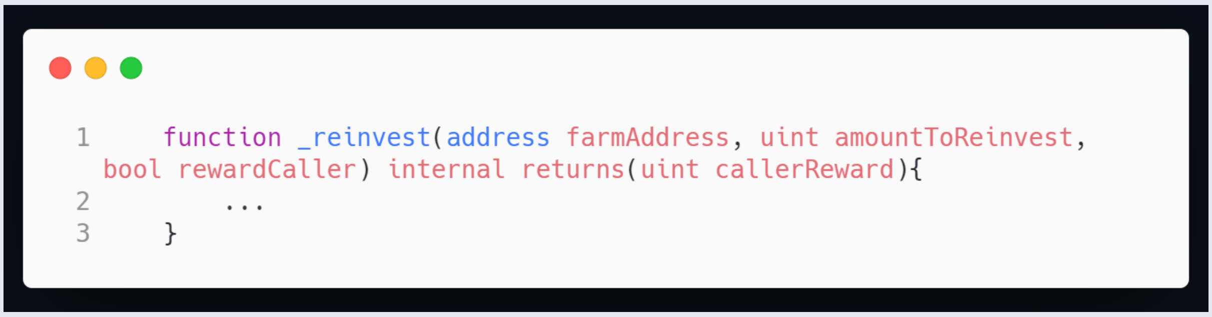
DESCRIPTION	Functions don't check for a zero address before assigning variables.
RECOMMENDATION	Add a check for zero address if deemed necessary.
MITIGATED/COMMENT	Project Team Comment: Accepted no changes made.

Redundant Parameter

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0006
LOCATION	CompounderFactory.sol -> 248 CompounderFactory.sol -> 263-358



```
uint reward = _reinvest(farmAddress, rewardToReinvest, true);

```

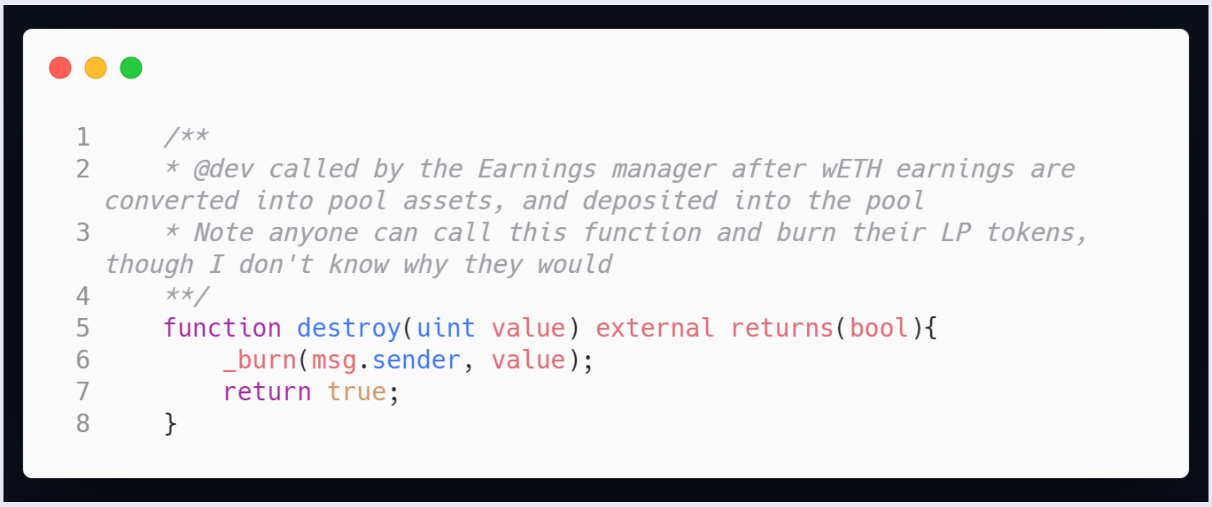


```
function _reinvest(address farmAddress, uint amountToReinvest, bool rewardCaller) internal returns(uint callerReward){
    ...
}
```

DESCRIPTION	The function *_reinvest* is called once at the of *harvestCompounding*, with *rewardCaller* as a constant *true*.
RECOMMENDATION	Remove the redundant parameter *rewardCaller* or add logic to make use of it.
MITIGATED/COMMENT	The team removed the parameter.

Unrestricted Protected Function

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0007
LOCATION	UniswapV2ERC20.sol -> 100-103

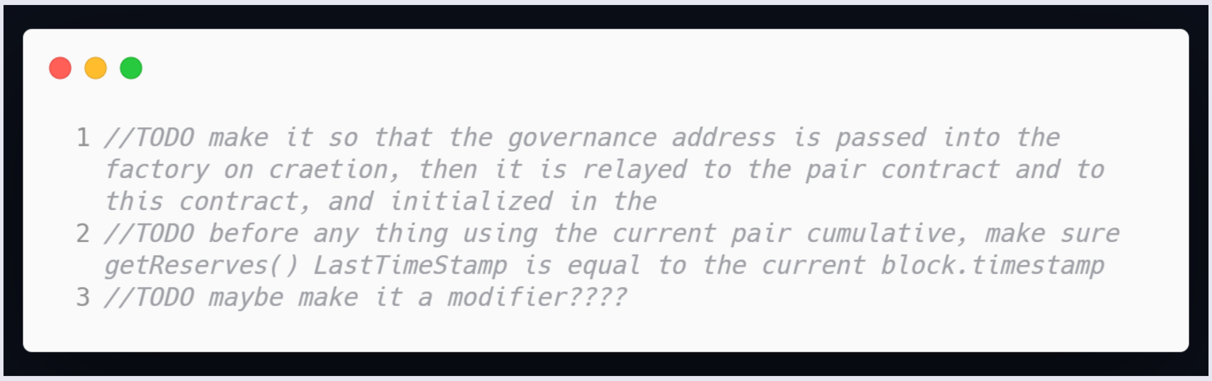


```
    /**
    * @dev called by the Earnings manager after wETH earnings are converted into pool assets, and deposited into the pool
    * Note anyone can call this function and burn their LP tokens, though I don't know why they would
    **/
    function destroy(uint value) external returns(bool){
        _burn(msg.sender, value);
        return true;
    }
```

DESCRIPTION	The dev comments indicate that this function is only meant to be called by a privileged address.
RECOMMENDATION	Consider using a required statement or a modifier to enforce this at the smart contract level.
MITIGATED/COMMENT	The function was moved to UniswapV2Pair.sol and the team added a modifier to only be callable by `EarningsManager`.

TODO Comment

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0008
LOCATION	Holding.sol => l16-18



```
//TODO make it so that the governance address is passed into the factory on craetion, then it is relayed to the pair contract and to this contract, and initialized in the
//TODO before any thing using the current pair cumulative, make sure getReserves() LastTimeStamp is equal to the current block.timestamp
//TODO maybe make it a modifier????
```

DESCRIPTION	A TODO comment is present on the contract.
RECOMMENDATION	Remove the comment, or turn it into documentation.
MITIGATED/COMMENT	The team removed all TODO comments.

Unused Interfaces

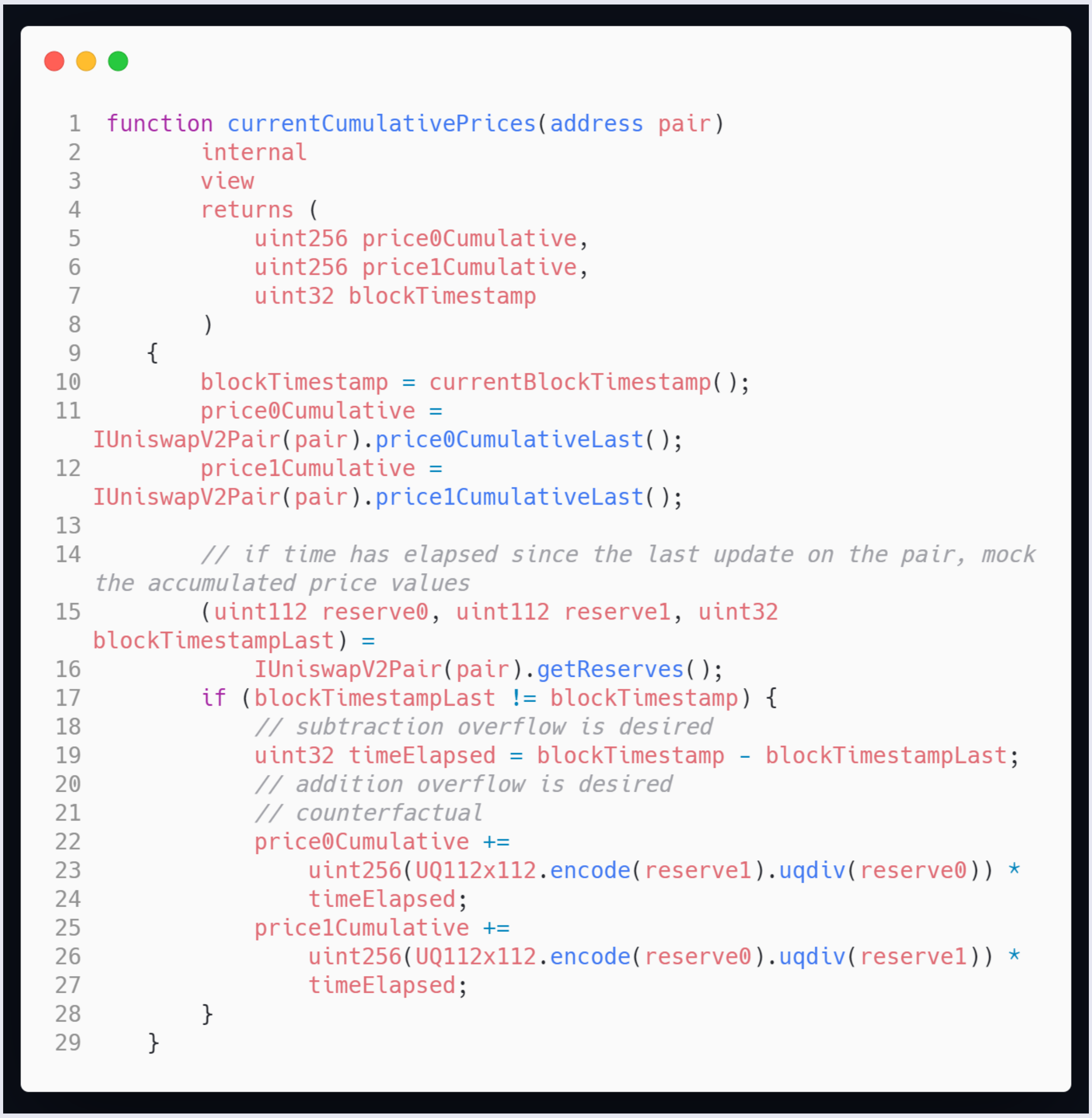
SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0009
LOCATION	Holding.sol => l5-14

```
import "./interfaces/OZ_IERC20.sol"
import "./libraries/SafeMath.sol"
import "./interfaces/iGovernance.sol"
import "./interfaces/IERC20.sol"
import "./interfaces/IUniswapV2Pair.sol"
import "./interfaces/IUniswapV2Factory.sol"
import "./interfaces/IUniswapV2Router02.sol"
import "./interfaces/IUniswapV2ERC20.sol"
import "./interfaces/IPathOracle.sol"
import "./interfaces/IPriceOracle.sol"
```

DESCRIPTION	Unused interfaces are imported into the contract.
RECOMMENDATION	Remove the unused interfaces.
MITIGATED/COMMENT	The team removed the unused import.

Implicit Statements And Declaration

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0010
LOCATION	PathOracle.sol => l81 PriceOracle.sol => l188-215



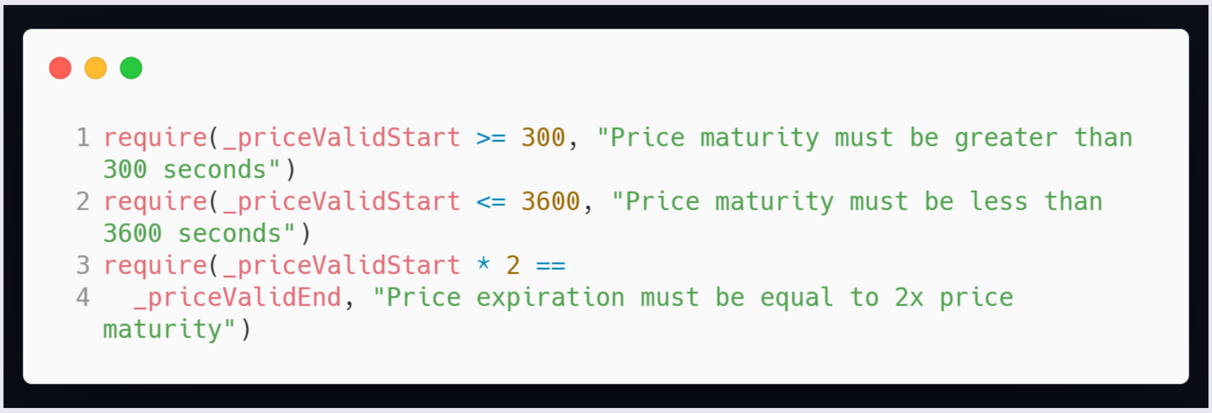
```
function currentCumulativePrices(address pair)
        internal
        view
        returns (
            uint256 price0Cumulative,
            uint256 price1Cumulative,
            uint32 blockTimestamp
        )
    {
        blockTimestamp = currentBlockTimestamp();
        price0Cumulative = IUniswapV2Pair(pair).price0CumulativeLast();
        price1Cumulative = IUniswapV2Pair(pair).price1CumulativeLast();

        // if time has elapsed since the last update on the pair, mock the accumulated price values
        (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast) =
            IUniswapV2Pair(pair).getReserves();
        if (blockTimestampLast != blockTimestamp) {
            // subtraction overflow is desired
            uint32 timeElapsed = blockTimestamp - blockTimestampLast;
            // addition overflow is desired
            // counterfactual
            price0Cumulative +=
                uint256(UQ112x112.encode(reserve1).uqdiv(reserve0)) *
                timeElapsed;
            price1Cumulative +=
                uint256(UQ112x112.encode(reserve0).uqdiv(reserve1)) *
                timeElapsed;
        }
    }
```

DESCRIPTION	The boolean variable *inFavored* is implicitly initialized and the *currentCumulativePrices* function has implicit return statements.
RECOMMENDATION	As a rule of thumb, prefer being explicit over being implicit to improve readability.

MITIGATED/COMMENT	Project Team Comment: - PathOracle.sol: Explicitly set `inFavored` to false at initialization(line 88) - PriceOracle.sol: Accepted the `currentCumulativePrices` function was straight up copied from Uniswaps Oracle Library found here https://github.com/Uniswap/uniswap-v2-periphery/blob/master/contracts/libraries/UniswapV2OracleLibrary.sol

Avoidable Repetition

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0011
LOCATION	PriceOracle.sol => l40-42, l53-55



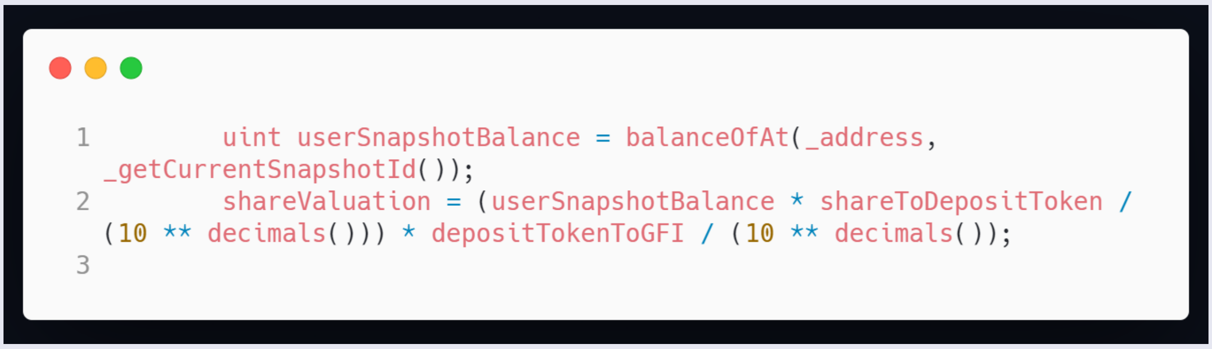
```
require(_priceValidStart >= 300, "Price maturity must be greater than 300 seconds")
require(_priceValidStart <= 3600, "Price maturity must be less than 3600 seconds")
require(_priceValidStart * 2 ==
  _priceValidEnd, "Price expiration must be equal to 2x price maturity")
```

DESCRIPTION	The variable validation logic is being repeated.
RECOMMENDATION	Extract that logic to a pure function to ensure that the constructor logic and the setTimingReq logic remain the same.
MITIGATED/COMMENT	The team extracted the `require` statements to a pure function and made use of it.

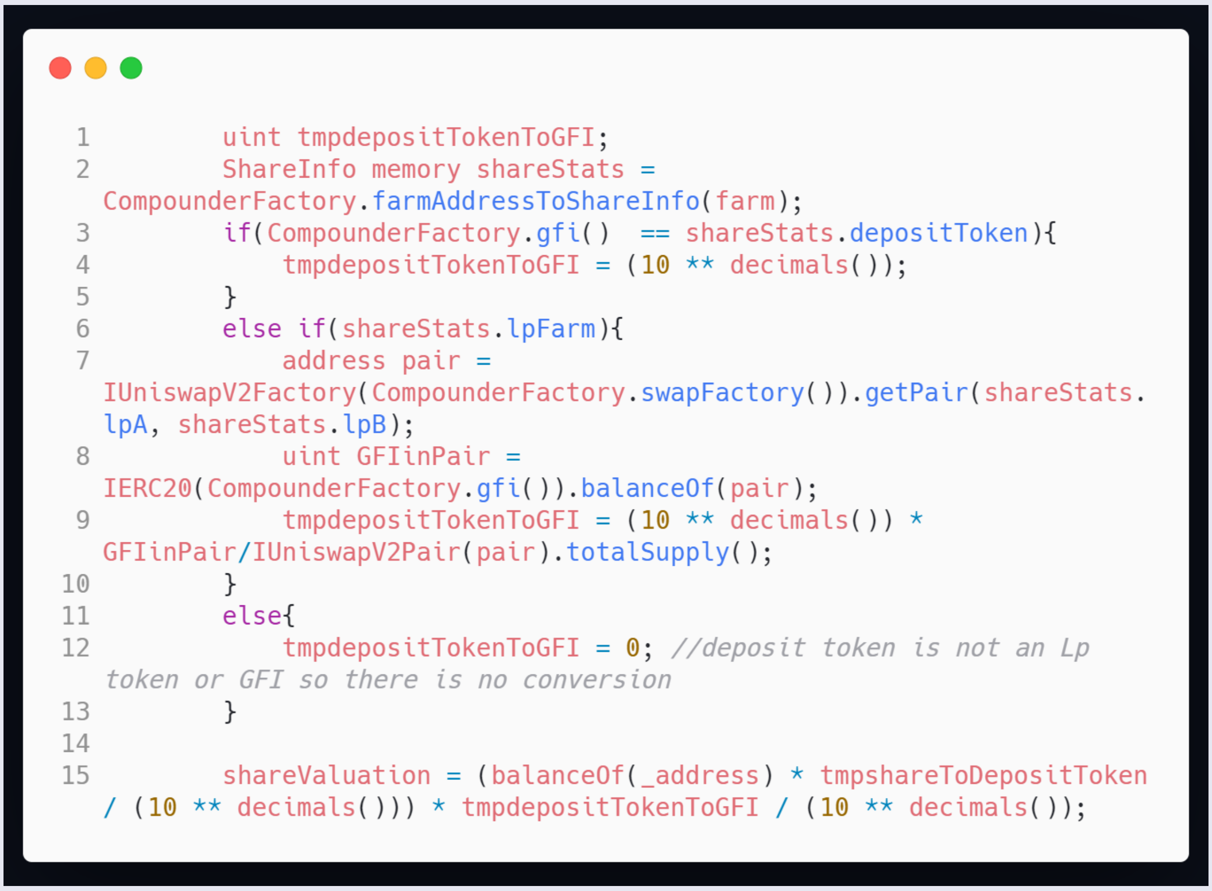
Static Analysis

Division Before Multiplication

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0012
LOCATION	Share.sol -> 74-75 Share.sol -> 85-99



```
        uint userSnapshotBalance = balanceOfAt(_address, _getCurrentSnapshotId());
        shareValuation = (userSnapshotBalance * shareToDepositToken / (10 ** decimals())) * depositTokenToGFI / (10 ** decimals());

```



```
        uint tmpdepositTokenToGFI;
        ShareInfo memory shareStats = CompounderFactory.farmAddressToShareInfo(farm);
        if(CompounderFactory.gfi()  == shareStats.depositToken){
            tmpdepositTokenToGFI = (10 ** decimals());
        }
        else if(shareStats.lpFarm){
            address pair = IUniswapV2Factory(CompounderFactory.swapFactory()).getPair(shareStats.lpA, shareStats.lpB);
            uint GFIinPair = IERC20(CompounderFactory.gfi()).balanceOf(pair);
            tmpdepositTokenToGFI = (10 ** decimals()) * GFIinPair/IUniswapV2Pair(pair).totalSupply();
        }
        else{
            tmpdepositTokenToGFI = 0; //deposit token is not an Lp token or GFI so there is no conversion
        }

        shareValuation = (balanceOf(_address) * tmpshareToDepositToken / (10 ** decimals())) * tmpdepositTokenToGFI / (10 ** decimals());
```

DESCRIPTION	The calculation for the amounts noted above uses mixed orders of multiplication and division. This may result in rounding errors.

	The rounding errors can potentially cause transactions to revert if attempting to transfer the remaining tokens in the contract and the amounts rounded up. The rounding errors can also create a discrepancy between amounts minted and burned.
RECOMMENDATION	Change the calculations to first multiply, then divide. Ensure that the USDT balance always exceeds the amount to be transferred. Make sure to use parentheses and beware of the order operations.
MITIGATED/COMMENT	Project Team Comment: Share.sol: Add parenthesis to clearly indicate what order of operations needs to be used. The values calculated here will not be used to transfer tokens.

Boolean Statement Tautology

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0013
LOCATION	FarmFactory.sol -> 62

```
require(_fee >= 0, "New fee must be greater than or equal to 0")

```

DESCRIPTION	*_fee* is a uint, it can't be negative.
RECOMMENDATION	Remove the redundant requirement.
MITIGATED/COMMENT	The team removed the \`require\` statement.

Missing Inheritance

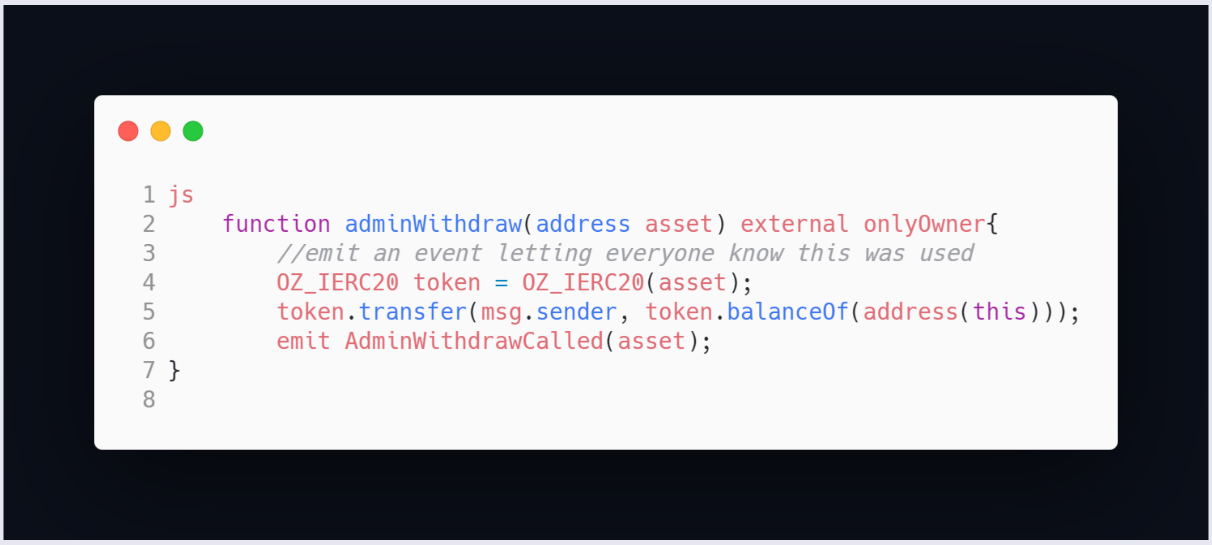
SEVERITY	Informational
RESOLVED	**NO**
FINDING ID	#0014
LOCATION	● FarmV2.sol & IFarmV2.sol ● Holding.sol & IHolding.sol ● PathOracle.sol & IPathOracle.sol ● CompounderFactory.sol & ICompounderFactory.sol

DESCRIPTION	The aforementioned contracts do not inherit from their interfaces.
RECOMMENDATION	Implement the interfaces to the contracts.
MITIGATED/COMMENT	Project Team Comment: Accepted no changes made.

On-Chain Analysis

Function That Can Withdraw Balance

SEVERITY	Low Risk
RESOLVED	**NO - See Comment**
FINDING ID	#0015
LOCATION	EarningsManager.sol -> 752-757



```
js
    function adminWithdraw(address asset) external onlyOwner{
        //emit an event letting everyone know this was used
        OZ_IERC20 token = OZ_IERC20(asset);
        token.transfer(msg.sender, token.balanceOf(address(this)));
        emit AdminWithdrawCalled(asset);
}

```

DESCRIPTION	The function can withdraw assets from the contract. It emits an event of which asset it withdraws.
RECOMMENDATION	Remove the `adminWithdraw` function and extract the leftover dust in another way.
MITIGATED/COMMENT	Although this function gives elevated privileges to the `Owner`, the balance of the contract will be any leftover dust from adding liquidity. Project Team Comment: “The earnings manager will only ever have a very small amount of funds in it because the same function that transfers money in will supply liquidity to one of our swap pairs, then burn the minted LP token. So the only thing left over will be a small amount of dust from not being able to perfectly add liquidity.”

No Timelock

SEVERITY	Low Risk
RESOLVED	**IN PROGRESS**
FINDING ID	#0016
LOCATION	CompounderFactory https://polygonscan.com/address/0xDc15F68E5F80ACD5966c84f518B1504A7E1772CA FarmFactory https://polygonscan.com/address/0x41d8920282eEDCcfC2f857e5e40Aa560a65d762B EarningsManager https://polygonscan.com/address/0x867A1CCE2Df35b26B9a50Ce5cbD5c1B603938E6F Incinerator https://polygonscan.com/address/0x4F6cCd1242323c23fEcf95b9C073C839db18649F FeeManager https://polygonscan.com/address/0x12e26Ad5Ce1Ed4b51F6D2d12Ac92765659D4e756

DESCRIPTION	The aforementioned contracts are owned by an externally owned account (EOA). The owner has the ability to change a number of protocol values: * CompounderFactory `setOptimizedReinvest`: Line 2250-2252 `setPerformBuyBacks`: Line 2254-2257 `changeBuyBacks`: Line 2259-2262 `setTxOriginWhitelist`: Line 2264-2267 `adjustWhitelist`: Line 2275-2278 `changeTierManager`: Line 2280-2283 `changeGovernor`: Line 2285-2288 `changeIncinerator`: Line 2290-2293 `changeCheckTiers`: Line 2295-2298 `changeShareInfo`: Line 2300-2309 `updateSharedVariables`: Line 2311-2317

	* FarmFactory `adjustWhitelist`: Line 722-725 `setHarvestFee`: Line 727-731 `setIncinerator`: Line 733-736 `setFeeManager`: Line 738-741 `setGovernance`: Line 743-746 `approveOrRevokeFarm`: 781-784 * EarningsManager `adjustWhitelist`: Line 505-508 `adminWithdraw`: Line 752-757 * Incinerator `setSlippage`: Line 464-467 `adminConvertEarningsToGFIandBurn`: Line 500-518 * FeeManager `adjustWhitelist`: Line 479-482 `adminWithdraw`: Line 596-600 The whitelisted addresses are excluded from harvest fees and can call functions that non-whitelisted cannot: * CompounderFactory `createCompounder`: Line 2319-2347 * EarningsManager `oracleProcessEarnings`: Line 576-665 `manualProcessEarnings`: Line 673-746 * FeeManager `deposit`: Line 505 `oracleStepSwap`: Line 540-562 `manualStepSwap`: Line 570-590
RECOMMENDATION	Transfer ownership to a timelock.
MITIGATED/COMMENT	We discussed extensively with the Gravity team about the scope of each of these parameters values : Gravity : " [...] I want to be clear that none of the functions give us the ability to steal users deposited funds, the worst we can do is steal the interest earned by farms, or steal the weth farm earnings(both of which are significantly smaller than the funds deposited by users). Functions that can steal the interest earned by users' deposits have been identified, and I agree they should be behind timelock. I

	also agree on adding timelocks to functions that will significantly change how the platform behaves [...]” The Gravity Finance team plans to document all information concerning owner privileges and timelock here: https://inthenextversion.gitbook.io/gravity-finance/owner-priv-and-time-locks

Appendix A1 - Reviewed Documents

Document	**Address**
DeFi/uniswapv2/interfaces/IUniswapV2Router01.sol	N/A
DeFi/uniswapv2/interfaces/IUniswapV2Router02.sol	N/A
DeFi/uniswapv2/interfaces/IUniswapV2Factory.sol	N/A
DeFi/uniswapv2/interfaces/IUniswapV2Callee.sol	N/A
DeFi/uniswapv2/interfaces/IEarningsManager.sol	N/A
DeFi/uniswapv2/interfaces/IUniswapV2ERC20.sol	N/A
DeFi/uniswapv2/interfaces/IUniswapV2Pair.sol	N/A
DeFi/uniswapv2/interfaces/IPriceOracle.sol	N/A
DeFi/uniswapv2/interfaces/iGovernance.sol	N/A
DeFi/uniswapv2/interfaces/IFeeManager.sol	N/A
DeFi/uniswapv2/interfaces/IPathOracle.sol	N/A
DeFi/uniswapv2/interfaces/OZ_IERC20.sol	N/A
DeFi/uniswapv2/interfaces/IHolding.sol	N/A
DeFi/uniswapv2/interfaces/IERC20.sol	N/A
DeFi/uniswapv2/interfaces/IWETH.sol	N/A

DeFi/uniswapv2/libraries/UniswapV2Library.sol	N/A
DeFi/uniswapv2/libraries/TransferHelper.sol	N/A
DeFi/uniswapv2/libraries/UQ112x112.sol	N/A
DeFi/uniswapv2/libraries/SafeMath.sol	N/A
DeFi/uniswapv2/libraries/Math.sol	N/A
DeFi/uniswapv2/UniswapV2Router02.sol	N/A
DeFi/uniswapv2/UniswapV2Factory.sol	N/A
DeFi/uniswapv2/UniswapV2ERC20.sol	N/A
DeFi/uniswapv2/UniswapV2Pair.sol	N/A
DeFi/uniswapv2/Holding.sol	N/A
DeFi/ERC20SnapshotInitializable.sol	N/A
DeFi/CompounderFactory.sol	N/A
DeFi/FarmFactory.sol	N/A
DeFi/FarmV2.sol	N/A
DeFi/Share.sol	N/A
interfaces/ICompounderFactory.sol	N/A
interfaces/iGravityToken.sol	N/A
interfaces/ITierManager.sol	N/A
interfaces/IIncinerator.sol	N/A

interfaces/IPriceOracle.sol	N/A
interfaces/IFarmFactory.sol	N/A
interfaces/iGovernance.sol	N/A
interfaces/OZ_IERC20.sol	N/A
interfaces/IFarmV2.sol	N/A
interfaces/IShare.sol	N/A
helper/ERC20Initializable.sol	N/A
core/PriceOracle.sol	0x2e0DfCD5D693DdcE4f0E0c7472561f62B912b19a
core/PathOracle.sol	0x3E22044f743c35C9689F9ab76063942beBdF559D

Appendix A2 - NOT Reviewed Contracts

Document	**Address**
Governance.sol	0xEe5578a3Bab33F7A56575785bb4846B90Be37d50
EarningsManager.sol	0x867A1CCE2Df35b26B9a50Ce5cbD5c1B603938E6F
FeeManager.sol	0x12e26Ad5Ce1Ed4b51F6D2d12Ac92765659D4e756
Incinerator.sol	0x4F6cCd1242323c23fEcf95b9C073C839db18649F

Appendix B - Risk Ratings

Risk	Description
High Risk	A fatal vulnerability that can cause immediate loss of Tokens / Funds
Medium Risk	A vulnerability that can cause some loss of Tokens / Funds
Low Risk	A vulnerability that can be mitigated
Informational	No vulnerability

Appendix C - Testing Standard

An ordinary audit is conducted using these steps.

1. Gather all information
2. Conduct a first visual inspection of documents and contracts
3. Go through all functions of the contract manually (2 independent auditors)
    a. Discuss findings
4. Use specialized tools to find security flaws
    a. Discuss findings
5. Follow up with project lead of findings
6. If there are flaws, and they are corrected, restart from step 2
7. Write and publish a report

During our audit, a thorough investigation has been conducted employing both automated analysis and manual inspection techniques. Our auditing method lays a particular focus on the following important concepts:

- Ensuring that the code and codebase use best practices, industry standards, and available libraries.
- Testing the contract from different angles ensuring that it works under a multitude of circumstances.
- Analyzing the contracts through databases of common security flaws.

Follow Obelisk Auditing for the Latest Information

ObeliskOrg ObeliskOrg

OBELISK

Part of Tibereum Group