OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 51	2021-12-30	Zapmore, Plemonade	Audit Final

Audit Notes

Audit Date	YYYY-MM-DD - YYYY-MM-DD
Auditor/Auditors	Plemonade, Mechwar
Auditor/Auditors Contact Information	contact@obeliskauditing.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB522123259

Disclaimer

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 Contents

Project Information

Name	FarmersOnlyFi
Description	Auto-compounding vaults on Harmony
Website	http://farmersonly.fi
Contact	https://discord.gg/tEDFEgHSJD
Contact information	@Cryptina_defi on TG
Token Name(s)	FOX
Token Short	FOX
Contract(s)	See Appendix A
Code Language	Solidity
Chain	Harmony

Audit of FarmersOnlyFi

The contracts deployed aren’t the updated contracts which include fixes to issues found.

Obelisk was commissioned by FarmersOnlyFi on the 2nd of November 2021 to conduct a comprehensive audit of FarmersOnlyFis’ contracts. The following audit was conducted between the 17th of November 2021 and the 29th of December 2021. Two of Obelisk's security experts went through the related contracts manually using industry standards to find if any vulnerabilities could be exploited either by the project team or users.

During the audit, the auditors found multiple issues of all risk levels. Many of these were fixed by the project team with only Medium issues #8 and #9 still open, and Low-Risk issues #11 and #25 still open.

Issue #8 is a centralization issue, but as the project team fixed it, they added a new bug which is still open. In issue #9, the user can earn rewards accumulated before the user deposited which creates an unfair advantage as this can be taken advantage of multiple times.

Issue #11 refers to the possibility of the reward calculation being slightly different than expected under certain circumstances, please see the project comment on this open issue. Issue #25 refer to an issue where standardization is not used which could make it hard to verify the list of operators.

However as the contracts that contain the fixes are not yet deployed, all the issues found are still present in the deployed contracts. See issue #22, #23, #24.

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 they deem that it’s worth solving these issues.

The team has not reviewed the UI/UX, logic, team, or tokenomics of the FarmersOnlyFi project**.**

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

Summary Table

Finding	ID	Severity	Status
Vault Shares Is Calculated As Deposited Tokens	#0001	High Risk	Closed
Strategy Governor Can Change Router Address	#0002	High Risk	Closed
Strategy Governor Can Change Reward Address	#0003	High Risk	Closed
Strategy Governor Can Change The StakingPoolAddress	#0004	High Risk	Closed
Operator Addresses Cannot Easily Be Identified	#0005	High Risk	Closed
Potential Reentrancy	#0006	Medium Risk	Closed
Swapping Tokens Can Be Frontrun	#0007	Medium Risk	Partially Closed
Centralization Risk	#0008	Medium Risk	Open
Users Can Earn Rewards From Before Deposit	#0009	Medium Risk	Open
Non-Standard ERC20 Tokens Not Supported	#0010	Low Risk	Mitigated
Incorrect Fee/Reward Calculation	#0011	Low Risk	Open
Divide By Zero	#0012	Informational	Closed
Unbound Loop	#0013	Informational	Open
Unused Base Contract Functions	#0014	Informational	Open
Redundant Want Amount Check	#0015	Informational	Open
Timelock Minimum Delay Is Too Short	#0016	Informational	Closed
LP Specific Variables	#0017	Informational	Open
No Events Emitted For Changes To Protocol Values	#0018	Informational	Open
Unused Parameters	#0019	Informational	Open

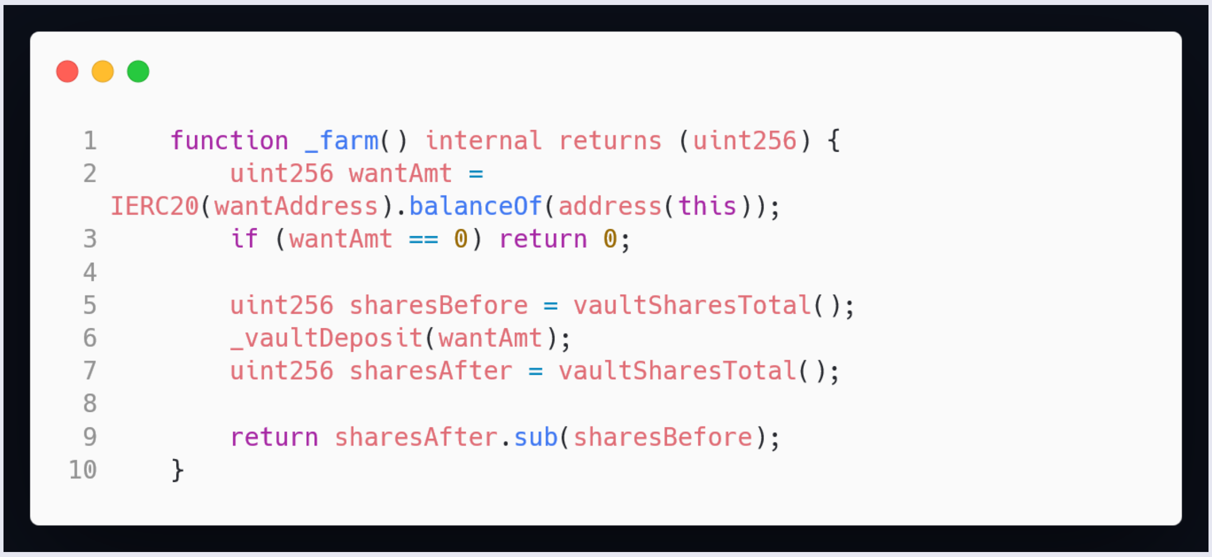
Unused Variable	#0020	Informational	Open
Missing Zero Checks	#0021	Informational	Open
Unverified Unknown Strategy Contracts	#0022	High Risk	Open
Revised Contracts Not Deployed	#0023	High Risk	Open
No Timelock	#0024	High Risk	Open
Hard To Verify List Of Operators	#0025	Low Risk	Open
Changes To Deployed Contract - StrategyElk	#0026	Informational	Closed
Changes To Deployed Contract - StrategyTranquil	#0027	Informational	Closed

Findings

Manual Analysis

Vault Shares Is Calculated As Deposited Tokens

FINDING ID	#0001
SEVERITY	High Risk
STATUS	Closed
LOCATION	Vault2/BaseStrategy.sol -> 99-108



```
function _farm() internal returns (uint256) {
    uint256 wantAmt = IERC20(wantAddress).balanceOf(address(this));
    if (wantAmt == 0) return 0;

    uint256 sharesBefore = vaultSharesTotal();
    _vaultDeposit(wantAmt);
    uint256 sharesAfter = vaultSharesTotal();

    return sharesAfter.sub(sharesBefore);
}
```

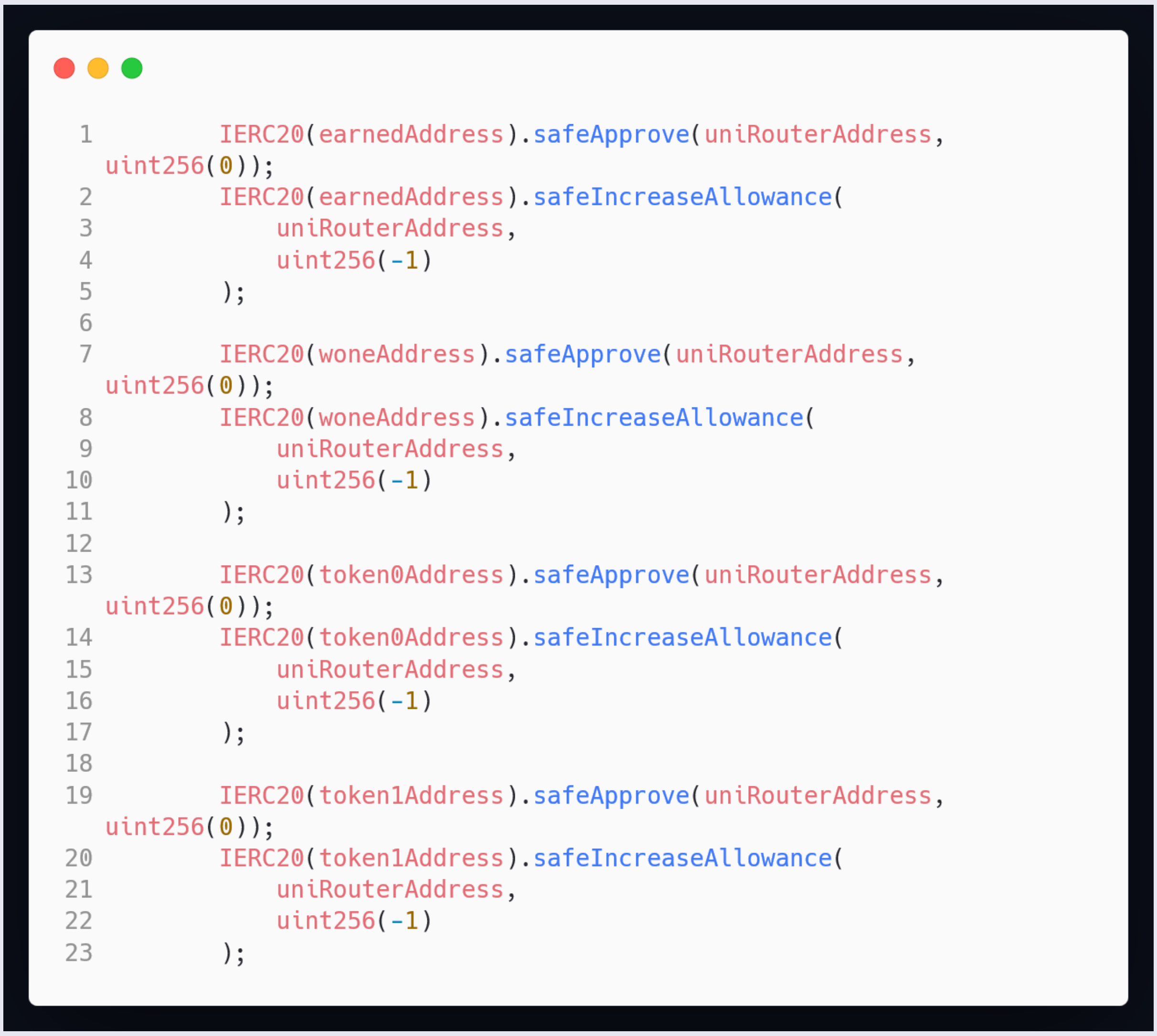
LOCATION	• Vault2/StrategyArtemis.sol -> 177-180: *function vaultSharesTotal() public override view returns (uint256)* • Vault2/StrategyElk.sol -> 173-176: *function vaultSharesTotal() public override view returns (uint256)* • Vault2/StrategyFarmersOnly.sol -> 177-180: *function vaultSharesTotal() public override view returns (uint256)* • Vault2/StrategyFuzz.sol -> 177-180: *function vaultSharesTotal() public override view returns (uint256)* • Vault2/StrategySushiBurn.sol -> 122-125: *function vaultSharesTotal() public override view returns (uint256)* • Vault2/StrategySushiSwap.sol -> 210-213: *function vaultSharesTotal() public override view returns (uint256)*

DESCRIPTION	The number of deposited tokens is used to determine how many shares should be minted.

	However, after a *panic()* or *emergencyPanic()* then an *unpause()* the entire want token balance is held by the strategy while being unpaused. Afterward, the next depositor will be allocated shares as if they deposited the entire value of the strategy again. This will effectively half the value of all other depositors’ shares. A malicious actor as the governor of a strategy can transfer governorship to a malicious contract to repeatedly exploit this in a single transaction. Also the same can be done with the rewards to a lesser extent when *emergencyRewardWithdraw()* is present.
RECOMMENDATION	Ensure that panicked tokens are counted in the share calculations.
RESOLUTION	The project implemented the fix in commit:5920b6024287f3c0ba7b22bc49f1e76ecb420079 and commit:f9fdc6e72ecd49d902f95099b0bf18d20f7e196e

Strategy Governor Can Change Router Address

FINDING ID	#0002
SEVERITY	High Risk
STATUS	Closed
LOCATION	Vault2/StrategyArtemis.sol -> 201-223 Vault2/StrategyElk.sol -> 197-213 Vault2/StrategyFarmersOnly.sol -> 201-223 Vault2/StrategyFuzz.sol -> 201-223 Vault2/StrategySushiBurn.sol -> 214-236 Vault2/StrategySushiSwap.sol -> 234-256



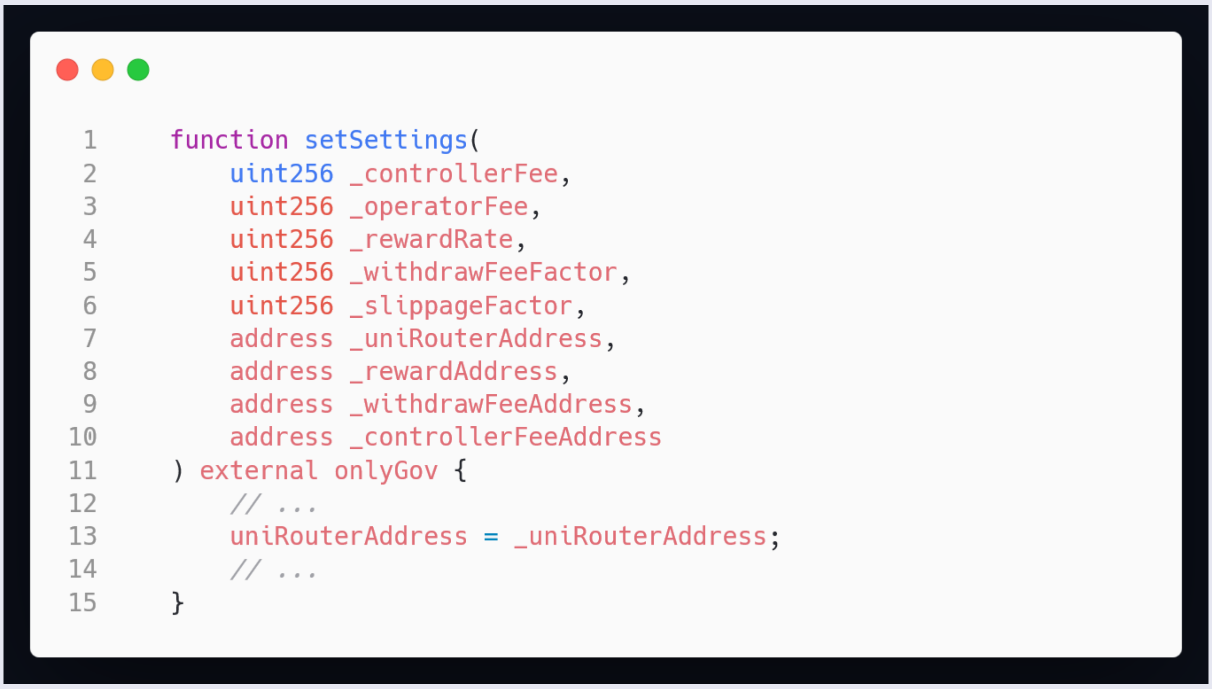
```
        IERC20(earnedAddress).safeApprove(uniRouterAddress, uint256(0));
        IERC20(earnedAddress).safeIncreaseAllowance(
            uniRouterAddress,
            uint256(-1)
        );

        IERC20(woneAddress).safeApprove(uniRouterAddress, uint256(0));
        IERC20(woneAddress).safeIncreaseAllowance(
            uniRouterAddress,
            uint256(-1)
        );

        IERC20(token0Address).safeApprove(uniRouterAddress, uint256(0));
        IERC20(token0Address).safeIncreaseAllowance(
            uniRouterAddress,
            uint256(-1)
        );

        IERC20(token1Address).safeApprove(uniRouterAddress, uint256(0));
        IERC20(token1Address).safeIncreaseAllowance(
            uniRouterAddress,
            uint256(-1)
        );
```

LOCATION	[Vault2/BaseStrategy.sol -> 229-266]()



```
function setSettings(
    uint256 _controllerFee,
    uint256 _operatorFee,
    uint256 _rewardRate,
    uint256 _withdrawFeeFactor,
    uint256 _slippageFactor,
    address _uniRouterAddress,
    address _rewardAddress,
    address _withdrawFeeAddress,
    address _controllerFeeAddress
) external onlyGov {
    // ...
    uniRouterAddress = _uniRouterAddress;
    // ...
}
```

DESCRIPTION	The vault strategies use a Uniswap router to exchange tokens. A malicious actor as the governor can change the router to a malicious contract. As the router has the approval to withdraw deposited and earned tokens from the strategy, this can be used to drain the entire contract balance.
RECOMMENDATION	Set the router path a single time during the constructor. Alternatively add a timelock to allow users to react to changes to the protocol. Obelisk recommends a timelock of at least 72 hours.
RESOLUTION	The project team has implemented the recommended fix. Fixed in commit 5920b6024287f3c0ba7b22bc49f1e76ecb420079.

Strategy Governor Can Change Reward Address

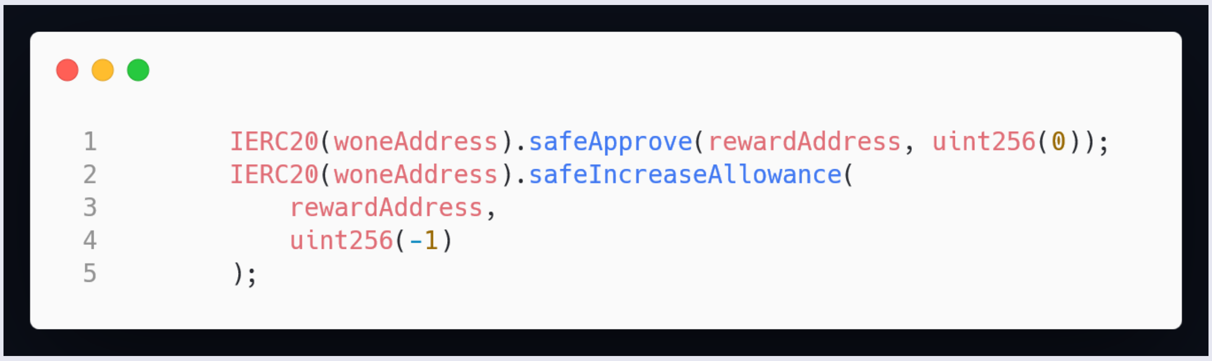
FINDING ID	#0003
SEVERITY	High Risk
STATUS	Closed
LOCATION	Vault2/BaseStrategy.sol -> 229-266

```
function setSettings(
    uint256 _controllerFee,
    uint256 _operatorFee,
    uint256 _rewardRate,
    uint256 _withdrawFeeFactor,
    uint256 _slippageFactor,
    address _uniRouterAddress,
    address _rewardAddress,
    address _withdrawFeeAddress,
    address _controllerFeeAddress
) external onlyGov {
    // ...
    rewardAddress = _rewardAddress;
    // ...
}
```

LOCATION	Vault2/BaseStrategy.sol -> 195 Vault2/StrategyArtemis.sol -> 161 Vault2/StrategyArtemis.sol -> 170 Vault2/StrategyElk.sol -> 157 Vault2/StrategyElk.sol -> 166 Vault2/StrategyFarmersOnly.sol -> 161 Vault2/StrategyFarmersOnly.sol -> 170 Vault2/StrategyFuzz.sol -> 161 Vault2/StrategyFuzz.sol -> 170 Vault2/StrategySushiSwap.sol -> 194 Vault2/StrategySushiSwap.sol -> 203

```
IStrategyBurnVault(rewardAddress).depositReward(woneAfter);
```

LOCATION	Vault2/StrategyArtemis.sol -> 189-193 Vault2/StrategyElk.sol -> 185-189 Vault2/StrategyFarmersOnly.sol -> 189-193 Vault2/StrategyFuzz.sol -> 189-193 Vault2/StrategySushiSwap.sol -> 222-226

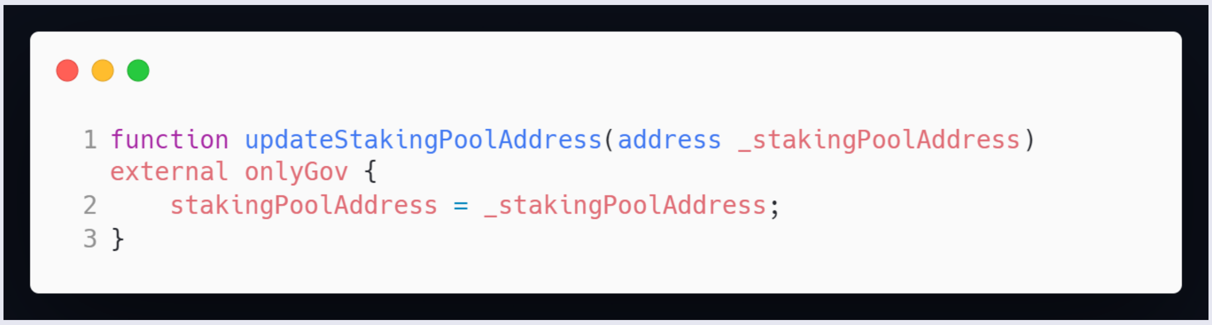


```
        IERC20(woneAddress).safeApprove(rewardAddress, uint256(0));
        IERC20(woneAddress).safeIncreaseAllowance(
            rewardAddress,
            uint256(-1)
        );
```

DESCRIPTION	A malicious actor as the governor can change the reward address to a malicious contract. The reward address receives unlimited approval for the wrapped native token. If the deposited token is the wrapped native token, this can be used to drain the entire contract value.
RECOMMENDATION	Set the reward address a single time during the constructor. Alternatively, add a timelock to allow users to react to changes to the protocol. Obelisk recommends a timelock of at least 72 hours. Avoid adding unnecessary permissions. It should be possible to transfer the reward tokens to the reward contract and handle the added reward in target contract.
RESOLUTION	The project team implemented the recommendation by transferring the amount from the current contract and thereby removing the approval to the *rewardsAddress* Fixed in commit 5920b6024287f3c0ba7b22bc49f1e76ecb420079 and 9608ca01a9b5264878cdfcd38b2369b7d5b878a8

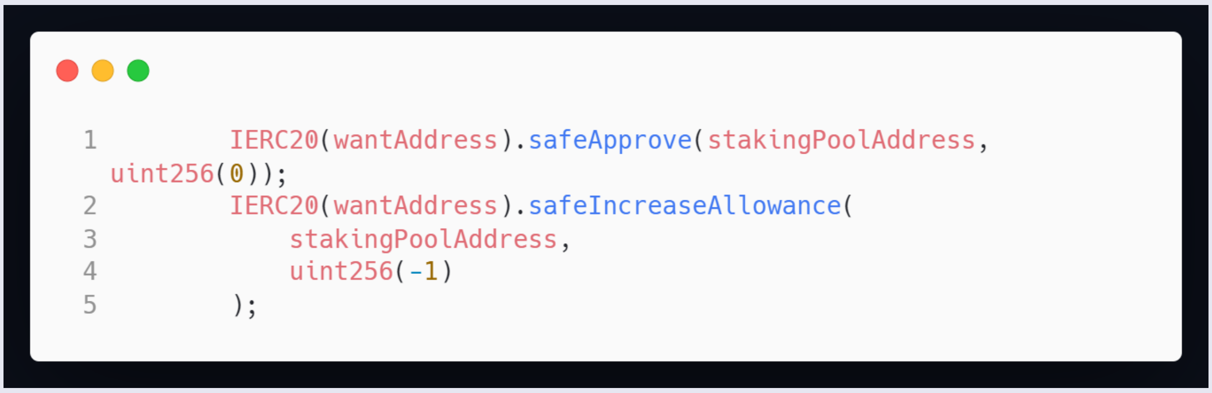
Strategy Governor Can Change The StakingPoolAddress

FINDING ID	#0004
SEVERITY	High Risk
STATUS	Closed
LOCATION	Vault2/StrategyElk.sol -> 232-235



```solidity
function updateStakingPoolAddress(address _stakingPoolAddress)
external onlyGov {
    stakingPoolAddress = _stakingPoolAddress;
}
```

LOCATION	Vault2/StrategyElk.sol -> 191-195



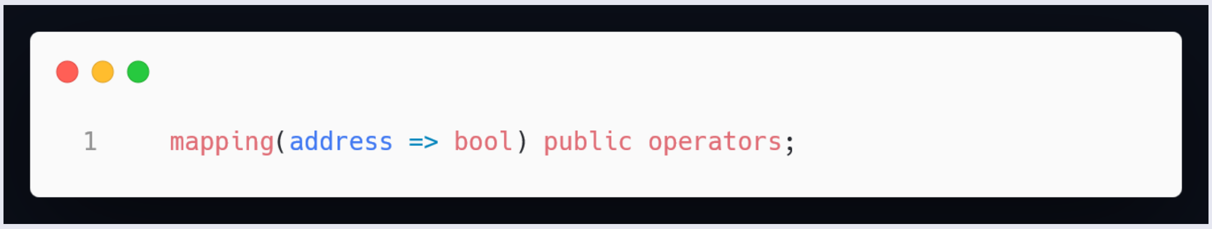
```solidity
        IERC20(wantAddress).safeApprove(stakingPoolAddress,
uint256(0));
        IERC20(wantAddress).safeIncreaseAllowance(
            stakingPoolAddress,
            uint256(-1)
        );
```

DESCRIPTION	A malicious actor as the governor can change the stakingPoolAddress to a malicious contract. The stakingPoolAddress receives unlimited approval for the wantAddress token. This can be used If the deposited token is the wrapped native token, this can be used to drain all of the wantAddress tokens from the contract.
RECOMMENDATION	Set the stakingPoolAddress a single time during the constructor. Alternatively, add a timelock to allow users to react to changes to the protocol. Obelisk recommends a timelock of at least 72 hours.
RESOLUTION	The project team implemented the recommendation by

	removing the *updateStakingPoolAddress()* function. Fixed in commit e45c0669a98dc728471d6c71db5f3aca6085fc23.

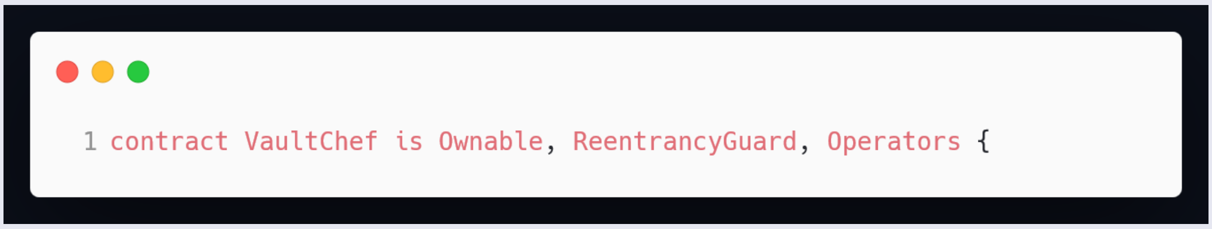
Operator Addresses Cannot Easily Be Identified

FINDING ID	#0005
SEVERITY	High Risk
STATUS	Closed
LOCATION	Farm/Operators.sol -> 11 Farm/Referral.sol -> 13



```
    mapping(address => bool) public operators;
```

LOCATION	Vault2/VaultChef.sol -> 13

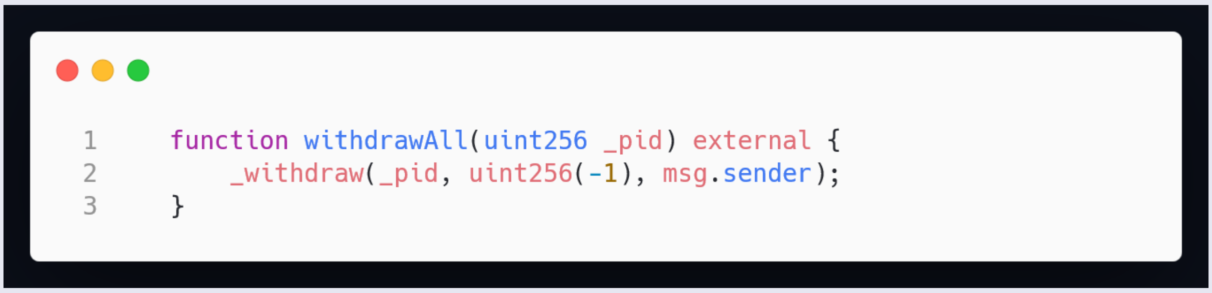


```
contract VaultChef is Ownable, ReentrancyGuard, Operators {
```

DESCRIPTION	The operator addresses noted are not easily identified by users. It is important to ensure that all addresses are accounted for.
RECOMMENDATION	Add an enumerated list of all operator addresses. Add events that emit when an address is added or removed.
RESOLUTION	The project team has implemented the recommendation however the list also shows old operators. This is marked as a comment over the variable and thus assumed to be intended as current operators can still be checked with the mapping after. Fixed in commit 5920b6024287f3c0ba7b22bc49f1e76ecb420079.

Potential Reentrancy

FINDING ID	#0006
SEVERITY	Medium Risk
STATUS	Closed
LOCATION	[VaultChef.sol -> 136-138]()



```
function withdrawAll(uint256 _pid) external {
    _withdraw(_pid, uint256(-1), msg.sender);
}
```

DESCRIPTION	The withdraw all function of the vault chef needs to be non-reentrant. If the underlying token is vulnerable to reentrancy then this function could be re-entered and drain a user's balance
RECOMMENDATION	Add the *nonReentrant* modifier to the *withdrawAll* function.
RESOLUTION	The project team has implemented the recommendation Fixed in commit 5920b6024287f3c0ba7b22bc49f1e76ecb420079.

Swapping Tokens Can Be Frontrun

FINDING ID	#0007
SEVERITY	Medium Risk
STATUS	Partially Closed
LOCATION	[Vault2/BaseStrategy.sol -> 243]()

```
        require(_slippageFactor <= slippageFactorUL, "_slippageFactor too high");
```

LOCATION	[Vault2/BaseStrategy.sol -> 276-282]()

```
        IUniRouter02(uniRouterAddress).swapExactTokensForTokens(
            _amountIn,
            amountOut.mul(slippageFactor).div(1000),
            _path,
            _to,
            now.add(600)
        );
```

LOCATION	[Vault2/BaseStrategy.sol -> 293-299]()

```
        IUniRouter02(uniRouterAddress).swapExactTokensForETH(
            _amountIn,
            amountOut.mul(slippageFactor).div(1000),
            _path,
            _to,
            now.add(600)
        );
```

DESCRIPTION	Tokens are exchanged via a DEX router using a slippage calculated using the immediate swap rate. This method cannot prevent sandwich attacks as it uses the immediate

	spot price. Therefore the rewards from the swap can be front-run. Additionally, even if using a time-weighted price, slippage can be set to an arbitrarily low value, which can lead to a higher risk of front running.
RECOMMENDATION	Calculate the slippage limit for the token using an oracle's time-weighted average price (TWAP) in order to prevent frontrunning. Add a lower limit to the slippage factor.
RESOLUTION	The project team has partially implemented the recommended fix. However, the project team has provided the following comment in regards to oracle TWAP: "I understand the spot price issue but we're not going to implement an oracle for this". This means the pool has to at least enough to cover the slippage factor in a buy situation(low liquidity pools will fail). However, the front-running/sandwich attack is still not fixed as the price is fetched in the same transaction. Partially fixed in commit 1d99588c5f6dc12b7c38642a5fcde66d2f3c07d5.

Centralization Risk

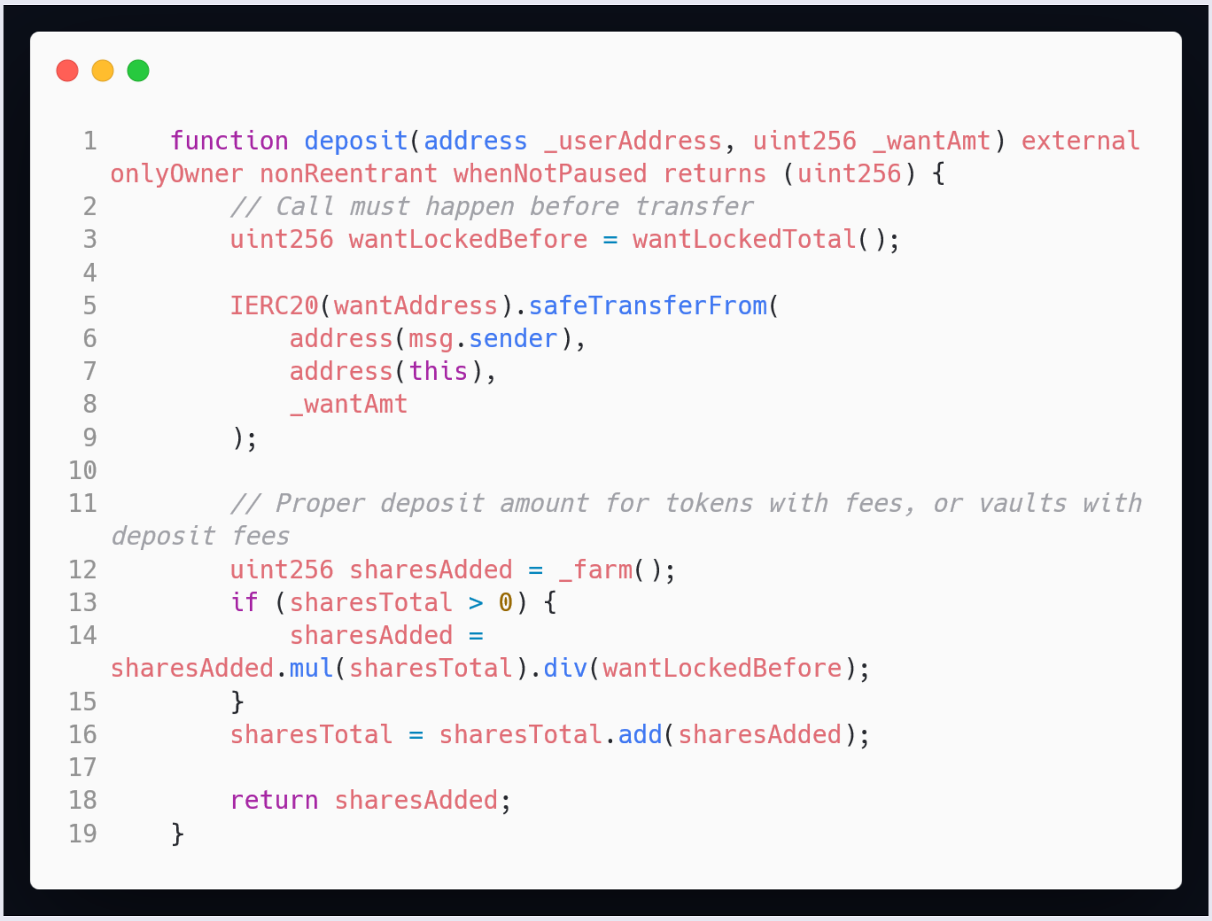
FINDING ID	#0008
SEVERITY	Medium Risk
STATUS	Open
LOCATION	• Vault2/BaseStrategy.sol -> 202-204: *function resetAllowances() external onlyGov* • Vault2/BaseStrategy.sol -> 206-208: *function pause() external onlyGov* • Vault2/BaseStrategy.sol -> 210-213: *function unpause() external onlyGov* • Vault2/BaseStrategy.sol -> 215-218: *function panic() external onlyGov* • Vault2/BaseStrategy.sol -> 220-223: *function unpanic() external onlyGov* • Vault2/BaseStrategy.sol -> 225-227: *function setGov() external onlyGov* • Vault2/BaseStrategy.sol -> 229-266: *function setSettings(uint256 _controllerFee, uint256 _operatorFee, uint256 _rewardRate, uint256 _withdrawFeeFactor, uint256 _slippageFactor, address _uniRouterAddress, address _rewardAddress, address _withdrawFeeAddress, address _controllerFeeAddress) external onlyGov* • Vault2/StrategyArtemis.sol -> 231-234: *function emergencyPanic() external onlyGov* • Vault2/StrategyElk.sol -> 227-230: *function emergencyPanic() external onlyGov* • Vault2/StrategyElk.sol -> 232-235: *function emergencyRewardWithdraw(uint amt) external onlyGov* • *Vault2/StrategyElk.sol -> 237-239*: *function updateStakingPoolAddress(address _stakingPoolAddress) external onlyGov* • Vault2/StrategyFarmersOnly.sol -> 231-234: *function emergencyPanic() external onlyGov* • Vault2/StrategyFuzz.sol -> 231-234: *function emergencyPanic() external onlyGov* • Vault2/StrategySushiBurn.sol -> 244-247: *function emergencyPanic() external onlyGov*

	• Vault2/StrategySushiSwap.sol -> 264-267: *function emergencyPanic() external onlyGov*

DESCRIPTION	A strategy's governor has a wide range of permissions which can dramatically affect the contract operation. In particular, *resetAllowances()*, *pause()*, *unpause()*, *panic()*, *unpanic()*, and *emergencyPanic()* will likely need to be activated in real time. In contrast, the *setGov()* and *setSettings()* change important protocol values and should be restricted by a timelock. Furthermore, *setSettings()* changes too many protocol values at the same time.
RECOMMENDATION	Separate the permissions of *setGov()* and *setSettings()* to a new operator address which is then set to a timelock. If these functionalities correspond to the operator and controller addresses, it may be worth renaming the variables for clarity. It may be helpful to replace the *Owner* base contract with an explicit *vaultAddress* variable and *onlyVault* modifier. Split the *setSettings()* function into multiple setters to remove the need to change every single setting at the same time.
RESOLUTION	The project team has implemented a check to make sure the *_controllerFeeAddress* is the same as the gov. This also adds a potential bug that *setsettings()* -> *setGov()* has to be called in that order as *setGov()* -> *setSettings()* then the controller will be on the last address and fail.

Users Can Earn Rewards From Before Deposit

FINDING ID	#0009
SEVERITY	Medium Risk
STATUS	Open
LOCATION	[Vault2/BaseStrategy.sol -> 79-97]



```
    function deposit(address _userAddress, uint256 _wantAmt) external onlyOwner nonReentrant whenNotPaused returns (uint256) {
        // Call must happen before transfer
        uint256 wantLockedBefore = wantLockedTotal();

        IERC20(wantAddress).safeTransferFrom(
            address(msg.sender),
            address(this),
            _wantAmt
        );

        // Proper deposit amount for tokens with fees, or vaults with deposit fees
        uint256 sharesAdded = _farm();
        if (sharesTotal > 0) {
            sharesAdded = sharesAdded.mul(sharesTotal).div(wantLockedBefore);
        }
        sharesTotal = sharesTotal.add(sharesAdded);

        return sharesAdded;
    }
```

DESCRIPTION	Deposited tokens receive shares based only on the strategy's want tokens. However some of the value of the strategy will be held as earned tokens instead. These earned tokens are converted to the want token when *earn()* is called. Because only want tokens are considered, users will receive rewards as if they had deposited since the last compound. This can be front-run by depositing a large sum of want tokens immediately before compounding.
RECOMMENDATION	Include compounding rewards as part of the deposit process.

RESOLUTION	The project team has indicated that they have no plans in solving this issue.

Non-Standard ERC20 Tokens Not Supported

FINDING ID	#0010
SEVERITY	Low Risk
STATUS	Mitigated
LOCATION	[BaseStrategy.sol -> 139-145]()

```
        if (withdrawFee > 0) {
            IERC20(wantAddress).safeTransfer(withdrawFeeAddress, withdrawFee);
        }

        _wantAmt = _wantAmt.sub(withdrawFee);

        IERC20(wantAddress).safeTransfer(vaultChefAddress, _wantAmt);
```

LOCATION	[VaultChef.sol -> 84-87]()

```
            pool.want.safeTransferFrom(msg.sender, address(this), _wantAmt);

            uint256 sharesAdded = IStrategy(poolInfo[_pid].strat).deposit(_to, _wantAmt);
            user.shares = user.shares.add(sharesAdded);
```

DESCRIPTION	1) If the want tokens have any transfer fees, the incorrect amount of withdrawal fee will be subtracted from the _*wantAmt*. The subtraction of the *withdrawFee* from _*wantAmt* should account for any transfer fees. 2) If the want tokens have any transfer fees, the incorrect amount of tokens would be transferred to the strategy from the vault. This incorrect amount would be more than what the vault would have available due to the transfer fees taking a portion; thus this would cause the transaction to revert.
RECOMMENDATION	Add a *balanceOf()* before and after the transferring of the want token, to take into

	account any applicable transfer fees.
RESOLUTION	The project team has provided the following commentary: "We're aware of this and won't fix it unless we want to implement support for a transfer fee token." Since the application for these contracts is not for transfer fee tokens, thus this will be a non-issue with this assumption. However, the contracts should be monitored on-chain if a transfer fee token is used.

Incorrect Fee/Reward Calculation

FINDING ID	#0011
SEVERITY	Low Risk
STATUS	Open
LOCATION	[StrategyArtemis.sol -> 76-96]()

```
        earnedAmt = distributeFees(earnedAmt, earnedAddress);
        earnedAmt = distributeOperatorFees(earnedAmt, earnedAddress);
        earnedAmt = distributeRewards(earnedAmt, earnedAddress);

        if (earnedAddress != token0Address) {
            // Swap half earned to token0
            _safeSwap(
                earnedAmt.div(2),
                earnedToToken0Path,
                address(this)
            );
        }

        if (earnedAddress != token1Address) {
            // Swap half earned to token1
            _safeSwap(
                earnedAmt.div(2),
                earnedToToken1Path,
                address(this)
            );
        }
```

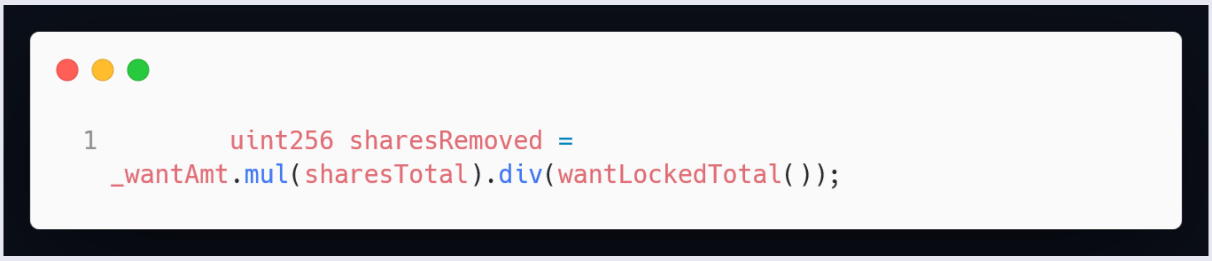
LOCATION	Other similar locations: • StrategyElk.sol -> 73-93 • StrategyFarmersOnly.sol -> 76-96 • StrategyFuzz.sol -> 76-96 • StrategySushiBurn.sol -> 75-94 • StrategySushiBurn.sol -> 141-153 • StrategySushiSwap.sol -> 84-129

DESCRIPTION	The fee and reward percentage is removed from the earned amount at each step of the calculation. This reduces the overall fees or rewards at later steps.

RECOMMENDATION	Use the original amount when calculating fees and rewards.
RESOLUTION	The project team has provided the following commentary: "We're aware of this but we compound every ~7mins right now and have a small withdraw fee so it can't be exploited right now." The *BaseStrategy* contract defines the max fee to be 10% spread between the various fees and rewards. Thus if a larger fee was taken prior to calculating the remaining fees and rewards, this could result in a large discrepancy for the subsequent fees and/or rewards.

Divide By Zero

FINDING ID	#0012
SEVERITY	Informational
STATUS	Closed
LOCATION	[BaseStrategy.sol -> 129]()

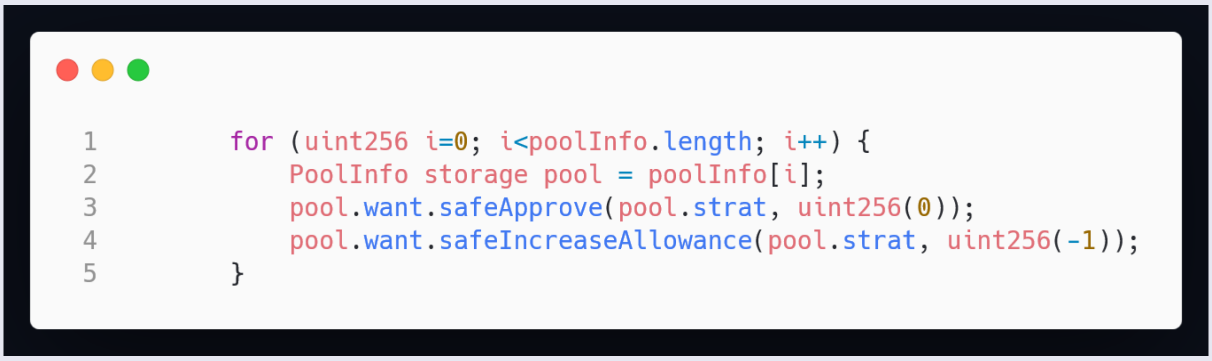


```
        uint256 sharesRemoved =
_wantAmt.mul(sharesTotal).div(wantLockedTotal());
```

DESCRIPTION	If there is no want token locked within the strategy then calling the *withdraw()* function results in the transaction reverting due to divide by zero.
RECOMMENDATION	Add a check for the amount in *sharesTotal* and *wantLockedTotal()* needing to be greater than zero before calculating the withdrawal fees and performing the transfer to the vault chef.
RESOLUTION	The project team has implemented the recommended fix. Fixed in commit 81515bb5f79e59ac9ccb40b3f313b3ec5dab4ef4.

Unbound Loop

FINDING ID	#0013
SEVERITY	Informational
STATUS	Open
LOCATION	Looping over *poolInfo.length*: • VaultChef.sol -> 141-145



```
        for (uint256 i=0; i<poolInfo.length; i++) {
            PoolInfo storage pool = poolInfo[i];
            pool.want.safeApprove(pool.strat, uint256(0));
            pool.want.safeIncreaseAllowance(pool.strat, uint256(-1));
        }
```

DESCRIPTION	Unbound loops may revert due to the gas fee limit.
RECOMMENDATION	Add an upper/lower bound parameter to the function to loop over a specific range.
RESOLUTION	The project team has indicated that they have no plans in solving this issue.

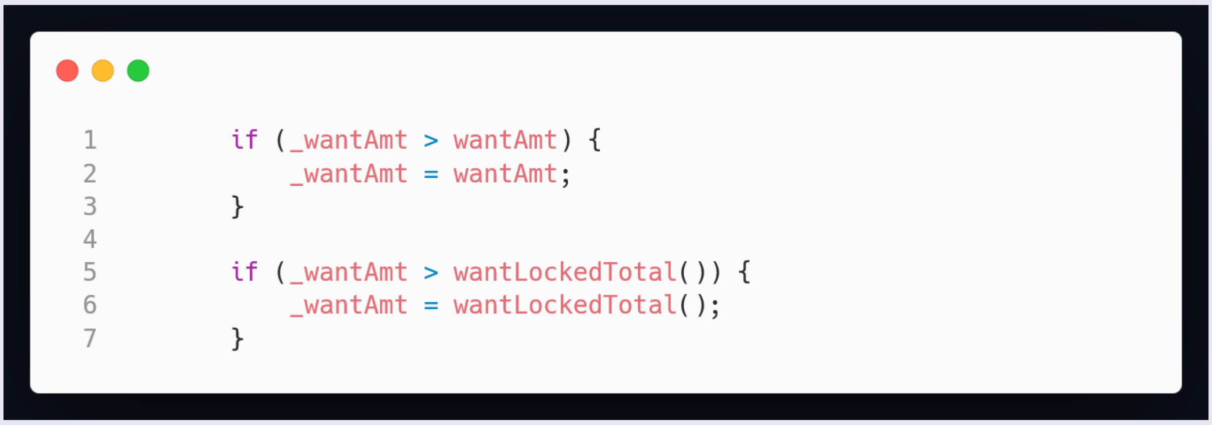
Unused Base Contract Functions

FINDING ID	#0014
SEVERITY	Informational
STATUS	Open
LOCATION	• Vault2/BaseStrategy.sol -> 151-165: *function distributeFees(uint256 _earnedAmt) internal returns (uint256)* • Vault2/BaseStrategy.sol -> 167-181: *function distributeOperatorFees(uint256 _earnedAmt) internal returns (uint256)* • Vault2/BaseStrategy.sol -> 183-200: *function distributeRewards(uint256 _earnedAmt) internal returns (uint256)*

DESCRIPTION	The implementation of these functions in the base contract are overridden and never used.
RECOMMENDATION	The override implementations appear to be similar. Replace the logic of the base contract with the derived contracts' logic.
RESOLUTION	The project team has indicated that they have no plans in solving this issue.

Redundant Want Amount Check

FINDING ID	#0015
SEVERITY	Informational
STATUS	Open
LOCATION	BaseStrategy.sol -> 121-127



```
        if (_wantAmt > wantAmt) {
            _wantAmt = wantAmt;
        }

        if (_wantAmt > wantLockedTotal()) {
            _wantAmt = wantLockedTotal();
        }
```

DESCRIPTION	The purpose of the checks on *_wantAmt* is to prevent users from withdrawing more tokens than the amount in the strategy and underlying contract. Since the first check statement already sets the lower bound, the subsequent check is redundant.
RECOMMENDATION	Remove the check of *_wantAmt* with the return value of *wantLockedTotal()*.
RESOLUTION	The project team has indicated that they have no plans in solving this issue.

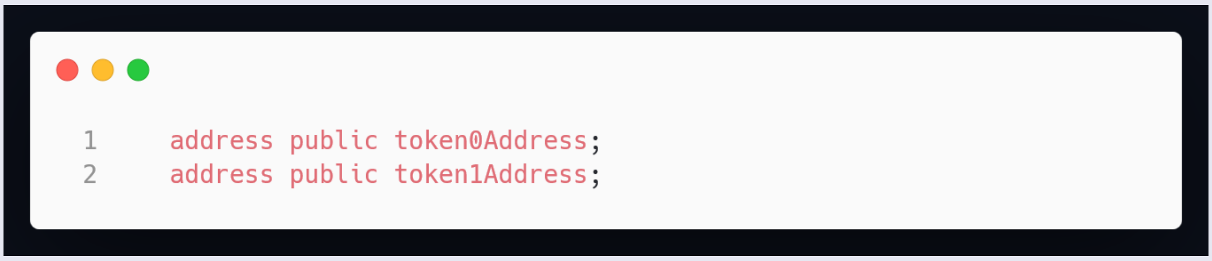
Timelock Minimum Delay Is Too Short

FINDING ID	#0016
SEVERITY	Informational
STATUS	Closed
LOCATION	• Timelock.sol -> 28: *uint public constant MINIMUM_DELAY = 3 hours;*

DESCRIPTION	A minimum delay of 3 hours is too short for users to be able to respond to changes.
RECOMMENDATION	Change the minimum delay. A minimum delay of 72 hours is recommended.
RESOLUTION	The project team has implemented the recommended fix. Fixed in commit 9608ca01a9b5264878cdfcd38b2369b7d5b878a8.

LP Specific Variables

FINDING ID	#0017
SEVERITY	Informational
STATUS	Open
LOCATION	Vault2/BaseStrategy.sol -> 20-21



```
address public token0Address;
address public token1Address;
```

DESCRIPTION	The noted variables are specific to LP-based strategies.
RECOMMENDATION	These variables are used by the *BaseStrategyLP* and should be located in the derived contract.
RESOLUTION	The project team has indicated that they have no plans in solving this issue.

Static Analysis

No Events Emitted For Changes To Protocol Values

FINDING ID	#0018
SEVERITY	Informational
STATUS	Open
LOCATION	• BaseStrategy.sol -> 225-227: *function setGov(address _govAddress) external onlyGov*

DESCRIPTION	Functions that change important variables should emit events such that users can more easily monitor the change.
RECOMMENDATION	Emit events from these functions.
RESOLUTION	The project team has indicated that they have no plans in solving this issue.

Unused Parameters

FINDING ID	#0019
SEVERITY	Informational
STATUS	Open
LOCATION	• BaseStrategy.sol -> 79: *function deposit(address _userAddress, uint256 _wantAmt) external onlyOwner nonReentrant whenNotPaused returns (uint256) {* • BaseStrategy.sol -> 110: *function withdraw(address _userAddress, uint256 _wantAmt) external onlyOwner nonReentrant returns (uint256) {* • StrategyElk.sol -> 232: *function emergencyRewardWithdraw(uint amt) external onlyGov {*

DESCRIPTION	For *BaseStrategy.sol*, the *_userAddress* variable is unused in the *deposit()* and *withdraw()* functions. For *StrategyElk.sol*, the *amt* variable is unused in the *emergencyRewardWithdraw()* function.
RECOMMENDATION	Remove the unused input variable.
RESOLUTION	The project team has indicated that they have no plans in solving this issue.

Unused Variable

FINDING ID	#0020
SEVERITY	Informational
STATUS	Open
LOCATION	Vault2/StrategySushiBurn.sol -> 15

```
address public foxAddress;
```

DESCRIPTION	The noted variable is never used.
RECOMMENDATION	Remove the variable or incorporate it into the contract functionality.
RESOLUTION	The project team has indicated that they have no plans in solving this issue.

Missing Zero Checks

FINDING ID	#0021
SEVERITY	Informational
STATUS	Open
LOCATION	• BaseStrategy.sol -> 225-227: *function setGov(address _govAddress) external onlyGov* • BaseStrategy.sol -> 229-266: *function setSettings(uint256 _controllerFee, uint256 _operatorFee, uint256 _rewardRate, uint256 _withdrawFeeFactor, uint256 _slippageFactor, address _uniRouterAddress, address _rewardAddress, address _withdrawFeeAddress, address _controllerFeeAddress) external onlyGov*

DESCRIPTION	The contract address values can be set to zero address in various setter functions. Zero addresses may cause incorrect contract behavior.
RECOMMENDATION	Add a check for zero address.
RESOLUTION	The project team has indicated that they have no plans in solving this issue.

On-Chain Analysis

Unverified Unknown Strategy Contracts

FINDING ID	#0022
SEVERITY	High Risk
STATUS	Open
LOCATION	0xE3fDA94D30cF11b7D5B535d14c6A3c6a4336ba87 0x579854e7c17B248F3AcA5680494e9FdB00bC8004 0x8aecA1F52BC03A6C1250c7e37E526805bACFd852 0x253C7D72AFC927Aed43E31FCA082E0119bC26871 0xAaA84d3fA77586E9679C8f4793F7D4867599757e 0x726a580F4ddB64cE55041d55b04be511AFD1993F 0x31dbA8a969B59A53269b661Ce131C676cF7e2B56 0xbAD0a431801ECD35986629c01CaD18ECd48F7A13 0xeBa469c3dfaeb1e39224C5FBF6a5ba2a1736cD16 0xae64fF1d33168684fAf0Cc1927907962600db6b6 0xdE3dC0e0081DC569354aB9433283E1C6803Bd1Ce 0xd0b0fCD56352611c59d884445AbDDDA989e2AA4f 0x11EBd48c564D8A9235260EA20133E53A8dCF811E 0x56eE9963eA5058dF4c99Ac4f396128c39E1CEa31 0x04e34e10D123b9551437e9Fa7341e6fFF168413a 0x2E4F9a05d1D13804B0725232aa956C263dA8901c 0xeF51e9931c7D6F6e329c06CfEA31574B0a6CFf28 0x8e4d781289351a0A598eA62a4E7fDb74B6c852B6 0xc9A64B99500505A922f0E8F3EeC9031F8e330e1F 0x536C93bef400fb912DB53954B5D0A49ef8BecCe5 0x6fF43936E4C28FD16CFB03936281566D231cD55C

DESCRIPTION	The aforementioned strategies are unverified contracts on-chain.
RECOMMENDATION	Verify these contracts.
RESOLUTION	N/A

Revised Contracts Not Deployed

FINDING ID	#0023
SEVERITY	High Risk
STATUS	Open
LOCATION	For all contracts, refer to Appendix A

DESCRIPTION	All deployed contracts match the code in the initial revision.
RECOMMENDATION	Deploy the revised contracts.
RESOLUTION	N/A

No Timelock

FINDING ID	#0024
SEVERITY	High Risk
STATUS	Open
LOCATION	All contracts, refer to Appendix A

DESCRIPTION	The contracts do not have a timelock for important variable changes. currently, the permission is handled by a Externally Owned Account (EOA). Contract owner: 0x54efdae67807cf4394020e48c7262bdbbdebd9f2
RECOMMENDATION	Transfer the ownership to a timelocked contract.
RESOLUTION	N/A

Hard To Verify List Of Operators

FINDING ID	#0025
SEVERITY	Low Risk
STATUS	Open
LOCATION	VaultChef 0x2914646e782cc36297c6639734892927b3b6fe56

DESCRIPTION	Since the list of operators are in a mapping. It is hard to retrieve the list of operators.
RECOMMENDATION	Migrate to using the rev 3 Operators.sol source code where operators are tracked in an array.
RESOLUTION	N/A

Changes To Deployed Contract - StrategyElk

FINDING ID	#0026
SEVERITY	Informational
STATUS	Closed
LOCATION	StrategyElk 0xbff66412b430c85db7de153a56004ff7550f0836 0xEc560037A8aD182169AA29737fd9A33c54F2Bb02 Rev-1 - StrategyElk.sol -> 237 - 239

```
    function updateStakingPoolAddress(address _stakingPoolAddress)
external onlyGov {
        stakingPoolAddress = _stakingPoolAddress;
    }
```

DESCRIPTION	Minor adjustments to the contracts were made prior to deployment. The noted function was removed.
RECOMMENDATION	No changes necessary as rev-1 had an issue with this and subsequent rev's removed it.
RESOLUTION	N/A

Changes To Deployed Contract - StrategyTranquil

FINDING ID	#0027
SEVERITY	Informational
STATUS	Closed
LOCATION	StrategyTranquil 0xb3c9A185bC39335787Df6f71498F8FDee46C2eB0 0xd0b0fCD56352611c59d884445AbDDDA989e2AA4f 0xA80EE2810Edeb3B6AdFDF7805e4864D2c6e522c0 0x77c9F0b20Fb3Ea9b3a2E81859F10E870F18ad6b6 0x9686d4f80c096f8e0383818A497cb5781b6d2Fef Rev 1 - IWETH.sol -> 9 - 15

```
    function approve(address guy, uint256 wad) external returns (bool);
    function transferFrom(
        address src,
        address dst,
        uint256 wad
    ) external returns (bool);
    function balanceOf(address account) external view returns (uint256);
```

DESCRIPTION	Minor adjustments to the contracts were made prior to deployment. The noted interface functions were removed.
RECOMMENDATION	No changes necessary.
RESOLUTION	N/A

Appendix A - Reviewed Documents

Document	**Address**
Farm/Operators.sol	N/A
Farm/Referral.sol	N/A
Farm/Timelock.sol	N/A
interfaces/IArtemisChef.sol	N/A
interfaces/IElkStake.sol	N/A
interfaces/IFarmersOnlyChef.sol	N/A
interfaces/IFuzzChef.sol	N/A
interfaces/IReferral.sol	N/A
interfaces/IStrategy.sol	N/A
interfaces/IStrategyBurnVault.sol	N/A
interfaces/ISushiStake.sol	N/A
interfaces/IUniPair.sol	N/A
interfaces/IUniRouter01.sol	N/A
interfaces/IUniRouter02.sol	N/A
interfaces/IWETH.sol	N/A
Vault2/BaseStrategy.sol	N/A
Vault2/BaseStrategyLP.sol	N/A
Vault2/StrategyArtemis.sol	N/A
Rev 1 Vault2/StrategyElk.sol	0xbFf66412b430c85Db7De153a56004ff7550f0836 0xEc560037A8aD182169AA29737fd9A33c54F2Bb02
Rev 1 Vault2/StrategyFarmersOnly.sol	0x4Bb7ADf64eC5e88dE79A0311D02610698e5451B2 0x4195Ffd17b99F3456984A7A277EBFe4de43a2549 0xa9236A025BBADa283B3E713B73f29187fCDd8Eb4

	0x7D5Fe069E1737d6982202aF02ea40b90FCC1B067 0xfc490ad8e505A1E30E48CDdaB5c56dE231a9ee60 0x2283402719CF75843c9c3e8eEEc66da41786B3ea 0x0C8ECa0B4cAB21e1A4bc66172D920D58f1040bc0
Rev 1 Vault2/StrategyFuzz.sol	0x4432df432858F95A424685fDE89C8d07A8dF1AB5 0x63dA44E0BC05D7515b7387Db850c838E64aEF0f9 0xC9ba0cEaD2483428d22676E196d0Da129fA5e5A7 0x2d15Fce8b738867b44CcA6E4C95a9fA0e92397e3 0x38d8bD3365bbE925b9bd7261a55968193e591b21 0x98730e6972a0Edc3919AA964AAe32c7B5F51036B 0xFE41F5782c96047B8F9A190325258C2df5E0b130 0xb48383Ad14Bc4965c36c3CEd03f6084AD7E58A45 0xb98681Eac487739A085B44F30a7Ce7B8F14B457f 0x0B378DB8903958F080CC2293AAB1dF91a883A4db 0x2e1483f123Fd19614f531f524427a2F89d2238a7 0x4Da1Da9aBC25E239614ae053E17349d426238588 0x7429a097ae99f91a5e195b9DB0600837bCDDa8CE
Rev 1 Vault2/StrategySushiBurn.sol	0x8491F1c41AB7a4c5785Ae05AA03FC9D5d799004c 0x99317cA257097c59B5044e451D04E88C095B5d37
Rev 1 Vault2/StrategySushiSwap.sol	0x1838a9976393539f68d412f625AAF6964ffF4779 0xD325970AB2C4eFabDaf1b4dEA350506F7D38433c 0x13D32982c5a72F9acF82fFC299E755af968b50B7
Rev 1 Vault2/VaultChef.sol	0x2914646e782cc36297c6639734892927b3b6fe56

Revisions

Revision 1	191b38390ed3ca37f249de41a9db17769ceb727f
Revision 2	5920b6024287f3c0ba7b22bc49f1e76ecb420079
Revision 3	9608ca01a9b5264878cdfcd38b2369b7d5b878a8

Imported Contracts

OpenZeppelin	3.1.0

Externally Owned Accounts

VaultChef Owner Strategy Governor	0x54EfdaE67807cf4394020e48c7262bdbbdEbd9F2

VaultChef Operators	Could not be determined

External Contracts

These contracts are not part of the audit scope.

N/A	N/A

Appendix B - Risk Ratings

Risk	Description
High Risk	A fatal vulnerability that can cause the loss of all Tokens / Funds.
Medium Risk	A vulnerability that can cause the loss of some Tokens / Funds.
Low Risk	A vulnerability which can cause the loss of protocol functionality.
Informational	Non-security issues such as functionality, style, and convention.

Appendix C - Finding Statuses

Closed	Contracts were modified to permanently resolve the finding.
Mitigated	The finding was resolved by other methods such as revoking contract ownership. The issue may require monitoring, for example in the case of a time lock.
Partially Closed	Contracts were updated to fix the issue in some parts of the code.
Partially Mitigated	Fixed by project specific methods which cannot be verified on chain. Examples include compounding at a given frequency.
Open	The finding was not addressed.

Appendix D - Audit Procedure

A typical Obelisk audit uses a combination of the three following methods:

Manual analysis consists of a direct inspection of the contracts to identify any security issues. Obelisk auditors use their experience in software development to spot vulnerabilities. Their familiarity with common contracts allows them to identify a wide range of issues in both forked contracts as well as original code.

Static analysis is software analysis of the contracts. Such analysis is called “static” as it examines the code outside of a runtime environment. Static analysis is a powerful tool used by auditors to identify subtle issues and to verify the results of manual analysis.

On-chain analysis is the audit of the contracts as they are deployed on the block-chain. This procedure verifies that:

- deployed contracts match those which were audited in manual/static analysis;
- contract values are set to reasonable values;
- contracts are connected so that interdependent contract function correctly;
- and the ability to modify contract values is restricted via a timelock or DAO mechanism. (We recommend a timelock value of at least 72 hours)

Each obelisk audit is performed by at least two independent auditors who perform their analysis separately.

After the analysis is complete, the auditors will make recommendations for each issue based on best practice and industry standards. The project team can then resolve the issues, and the auditors will verify that the issues have been resolved with no new issues introduced.

Our auditing method lays a particular focus on the following important concepts:

- Quality code and the use of best practices, industry standards, and thoroughly tested libraries.
- Testing the contract from different angles to ensure that it works under a multitude of circumstances.
- Referencing the contracts through databases of common security flaws.

Follow Obelisk Auditing for the Latest Information

ObeliskOrg ObeliskOrg

OBELISK

Part of Tibereum Group