OBELISK

OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total:	YYYY-MM-DD	Zapmore, Auditor1	Audit Draft

Audit Notes

Audit Date	YYYY-MM-DD - YYYY-MM-DD
Auditor/Auditors	Auditor1, Auditor2
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	OB5XXXXXXXX

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. In instances where an auditor or team member has a personal connection with the audited project, that auditor or team member will be excluded from viewing or impacting any internal communication regarding the specific audit.

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	
Description	
Website	
Contact	
Contact information	@XXXX on TG
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	Polygon / BSC

Audit of T-Node 2

The main takeaway will be added here after the audit is completed and the final draft is created.

Obelisk was commissioned by XXXX on the XXXX th of XXXX 2022 to conduct a comprehensive audit of XXXX' contracts. The following audit was conducted between the XXXXth of XXXX 2022 and the XXXXth of XXXX 2022. 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.

Findings and other relevant info will be updated at audit completion and added here.

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, however, please take note of them.

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

This document is a summary of the findings that the auditors found. Please read the full document for a complete understanding of the audit.

Summary Table

Code Analysis

Finding	ID	Severity	Status
Owner Can Withdraw Deposited Assets And Rewards	#0001	High Risk	Closed
Depositing For Another Account Spends Their Token	#0002	Medium Risk	Closed
No Limit For Protocol Values	#0003	Low Risk	Closed
External Dependency	#0004	Low Risk	Closed
Use Safe Transfer	#0005	Low Risk	Closed
Rewards Not Allocated If Last Reward Was Over 20 Weeks Ago	#0006	Low Risk	Partially Closed
365 Days Will Cause Offset	#0007	Informational	Open
Division Before Multiplication	#0008	Informational	Closed
Overall And User Checkpoint Updated Simultaneously	#0009	Informational	Open
Addresses Are Hard Coded	#0010	Informational	Closed
Constants Using Testnet Addresses	#0011	Informational	Closed
Missing Zero Checks	#0012	Informational	Closed
No Events Emitted For Changes To Protocol Values	#0013	Informational	Closed
Block Rate Assumed Constant	#0014	Informational	Open

On-Chain Analysis

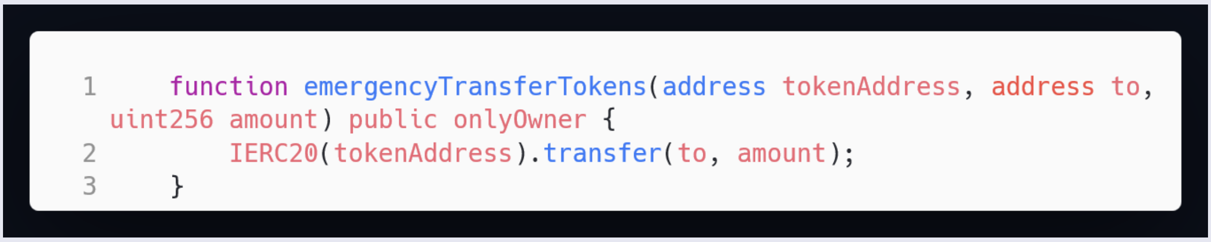
Finding	ID	Severity	Status
-	-	-	Open

Findings

Code Analysis

Owner Can Withdraw Deposited Assets And Rewards

FINDING ID	#0001
SEVERITY	High Risk
STATUS	Open
LOCATION	Vault.sol -> 217-219

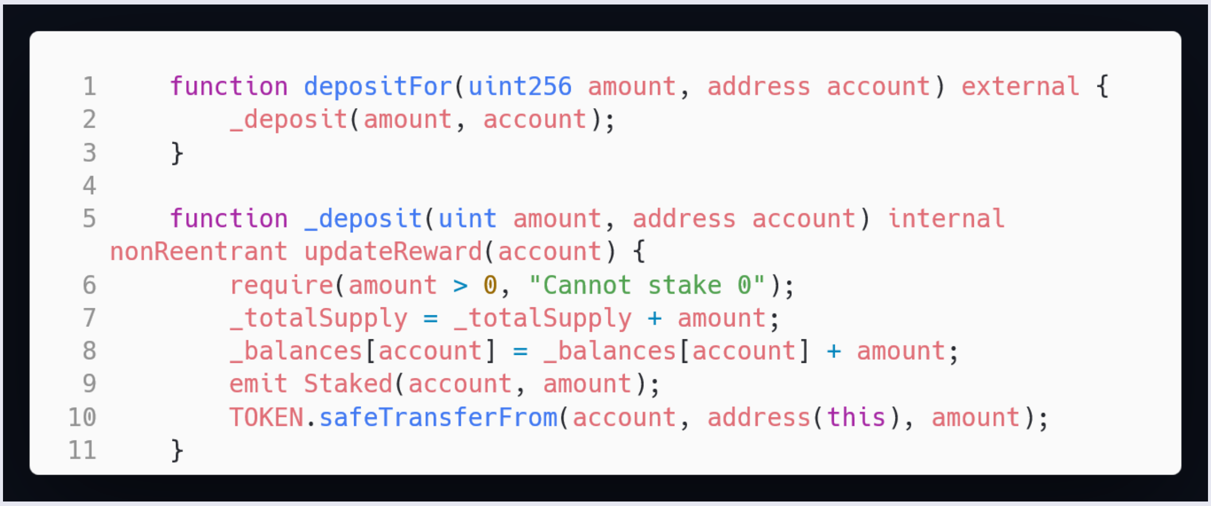


```
    function emergencyTransferTokens(address tokenAddress, address to, uint256 amount) public onlyOwner {
        IERC20(tokenAddress).transfer(to, amount);
    }
```

DESCRIPTION	The *emergencyTransferTokens()* function allows the Owner to withdraw any token from the contract, including the deposited token and the reward token.
RECOMMENDATION	Add a require statement that prohibits the withdrawal of the deposited and reward token.
RESOLUTION	The project team implemented the recommended changes.

Depositing For Another Account Spends Their Token

FINDING ID	#0002
SEVERITY	Medium Risk
STATUS	Closed
LOCATION	Vault.sol -> 131-141



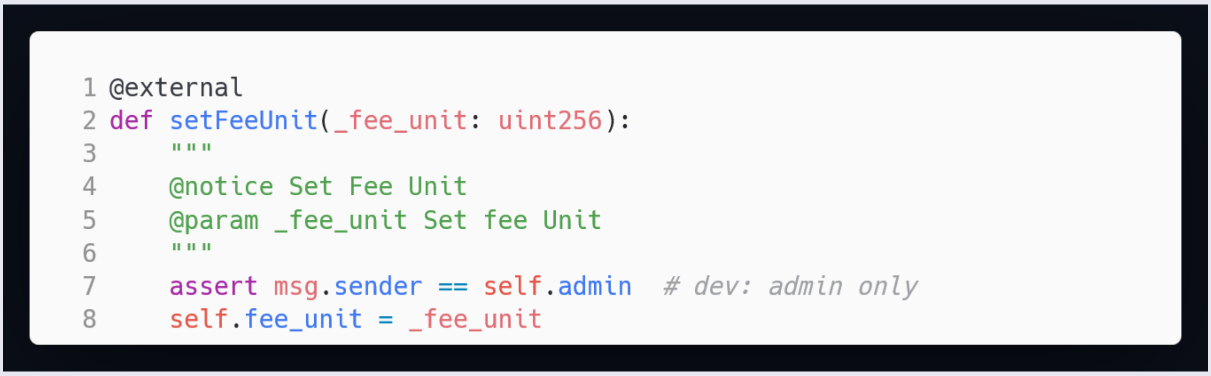
```
    function depositFor(uint256 amount, address account) external {
        _deposit(amount, account);
    }

    function _deposit(uint amount, address account) internal nonReentrant updateReward(account) {
        require(amount > 0, "Cannot stake 0");
        _totalSupply = _totalSupply + amount;
        _balances[account] = _balances[account] + amount;
        emit Staked(account, amount);
        TOKEN.safeTransferFrom(account, address(this), amount);
    }
```

DESCRIPTION	Function *depositFor()* is used to deposit for another account. The intuitive way to implement this functionality, is to transfer token from the caller's address to the contract, and increase the balance of the target account. In this implementation though, the caller is using the target's address tokens to deposit for them. That could be abused in order to move someone's tokens without their approval (In case they have approved the *Vault.sol* contract to spend their tokens).
RECOMMENDATION	Instead of transfering *TOKEN* from the account address, transfer it from the caller's address.
RESOLUTION	The function has been removed.

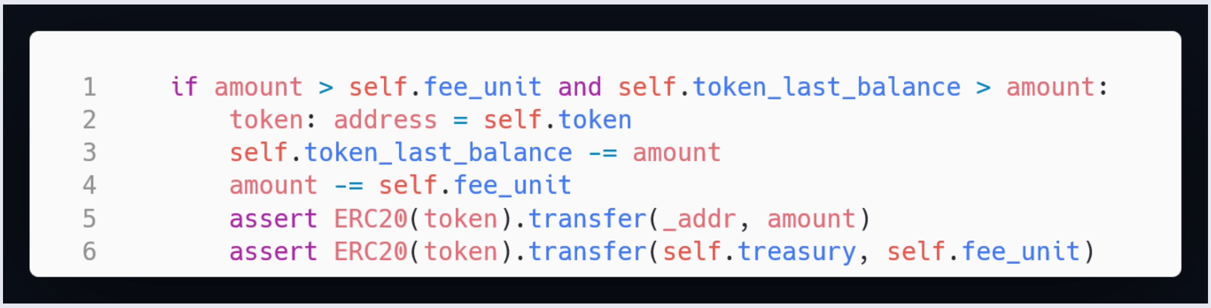
No Limit For Protocol Values

FINDING ID	#0003
SEVERITY	Low Risk
STATUS	Open
LOCATION	fee-distributor.vy -> 103-109



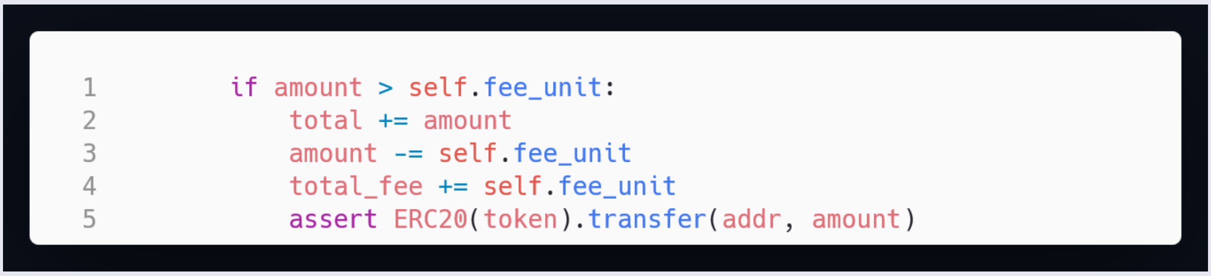
```
@external
def setFeeUnit(_fee_unit: uint256):
    """
    @notice Set Fee Unit
    @param _fee_unit Set fee Unit
    """
    assert msg.sender == self.admin  # dev: admin only
    self.fee_unit = _fee_unit
```

LOCATION	fee-distributor.vy -> 346-351



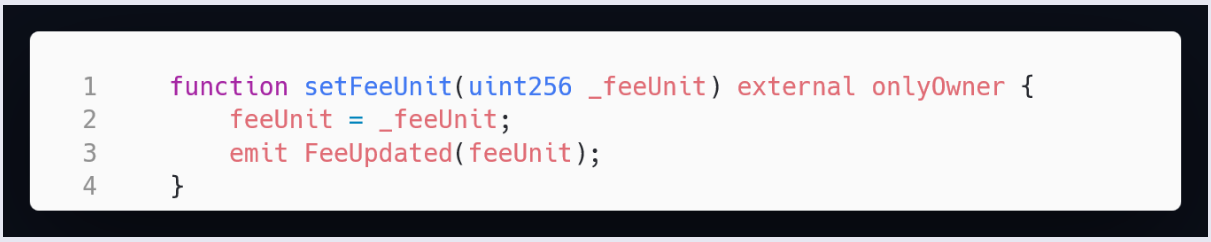
```
    if amount > self.fee_unit and self.token_last_balance > amount:
        token: address = self.token
        self.token_last_balance -= amount
        amount -= self.fee_unit
        assert ERC20(token).transfer(_addr, amount)
        assert ERC20(token).transfer(self.treasury, self.fee_unit)
```

LOCATION	fee-distributor.vy -> 393-397



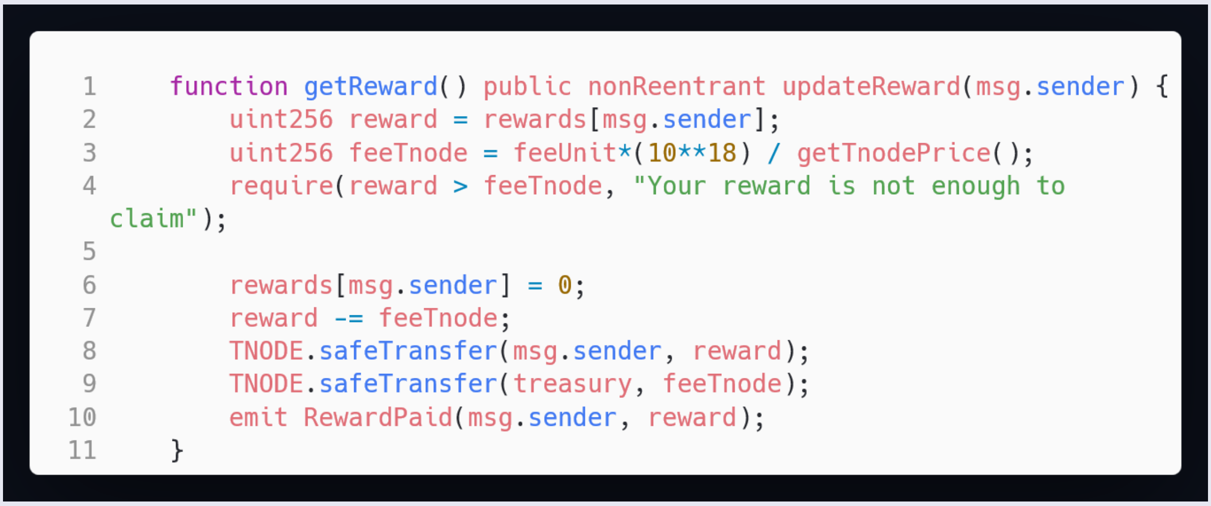
```
        if amount > self.fee_unit:
            total += amount
            amount -= self.fee_unit
            total_fee += self.fee_unit
            assert ERC20(token).transfer(addr, amount)
```

LOCATION	Vault.sol -> 66-69



```
    function setFeeUnit(uint256 _feeUnit) external onlyOwner {
        feeUnit = _feeUnit;
        emit FeeUpdated(feeUnit);
    }
```

LOCATION	Vault.sol -> 165-175



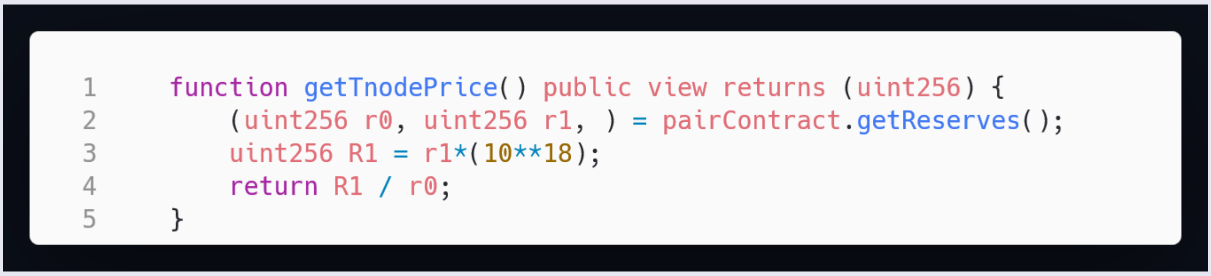
```solidity
    function getReward() public nonReentrant updateReward(msg.sender) {
        uint256 reward = rewards[msg.sender];
        uint256 feeTnode = feeUnit*(10**18) / getTnodePrice();
        require(reward > feeTnode, "Your reward is not enough to claim");

        rewards[msg.sender] = 0;
        reward -= feeTnode;
        TNODE.safeTransfer(msg.sender, reward);
        TNODE.safeTransfer(treasury, feeTnode);
        emit RewardPaid(msg.sender, reward);
    }
```

DESCRIPTION	*fee_unit* and *feeUnit* can be set arbitrarily high, potentially leading to users claiming 0 rewards.
RECOMMENDATION	Add an upper limit to the value.
RESOLUTION	The project team implemented the recommended changes.

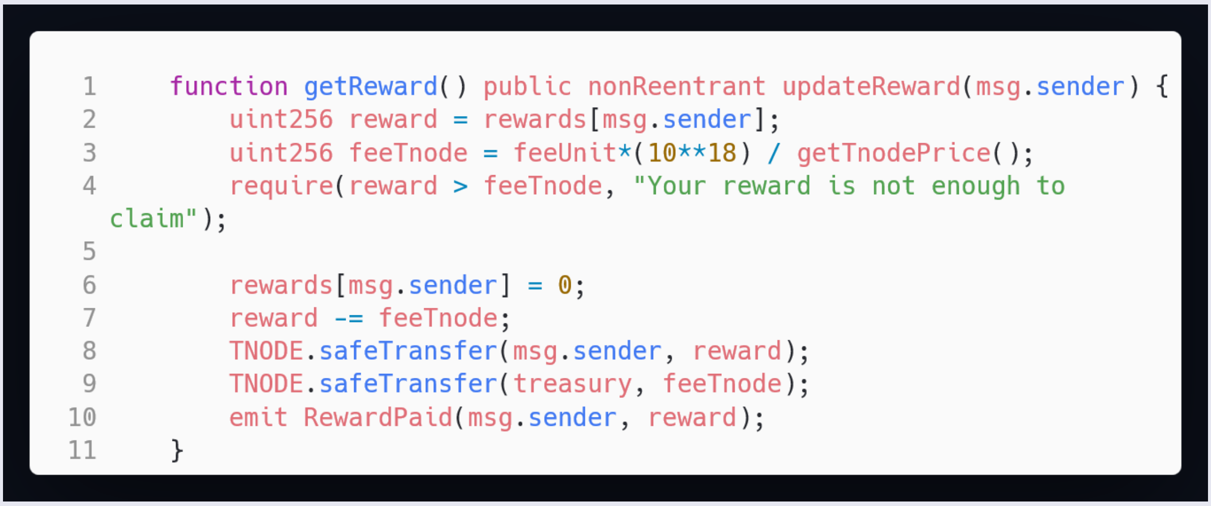
External Dependency

FINDING ID	#0004
SEVERITY	Low Risk
STATUS	Closed
LOCATION	Vault.sol -> 159-163



```
function getTnodePrice() public view returns (uint256) {
    (uint256 r0, uint256 r1, ) = pairContract.getReserves();
    uint256 R1 = r1*(10**18);
    return R1 / r0;
}
```

LOCATION	Vault.sol -> 165-175



```
function getReward() public nonReentrant updateReward(msg.sender) {
    uint256 reward = rewards[msg.sender];
    uint256 feeTnode = feeUnit*(10**18) / getTnodePrice();
    require(reward > feeTnode, "Your reward is not enough to claim");

    rewards[msg.sender] = 0;
    reward -= feeTnode;
    TNODE.safeTransfer(msg.sender, reward);
    TNODE.safeTransfer(treasury, feeTnode);
    emit RewardPaid(msg.sender, reward);
}
```

DESCRIPTION	The *.getReserves()* external call might fail and return 0, resulting in division by zero. This will result in users being unable to claim any rewards.
RECOMMENDATION	Use a default fee value to safeguard against this.
RESOLUTION	The project team implemented the recommended changes.

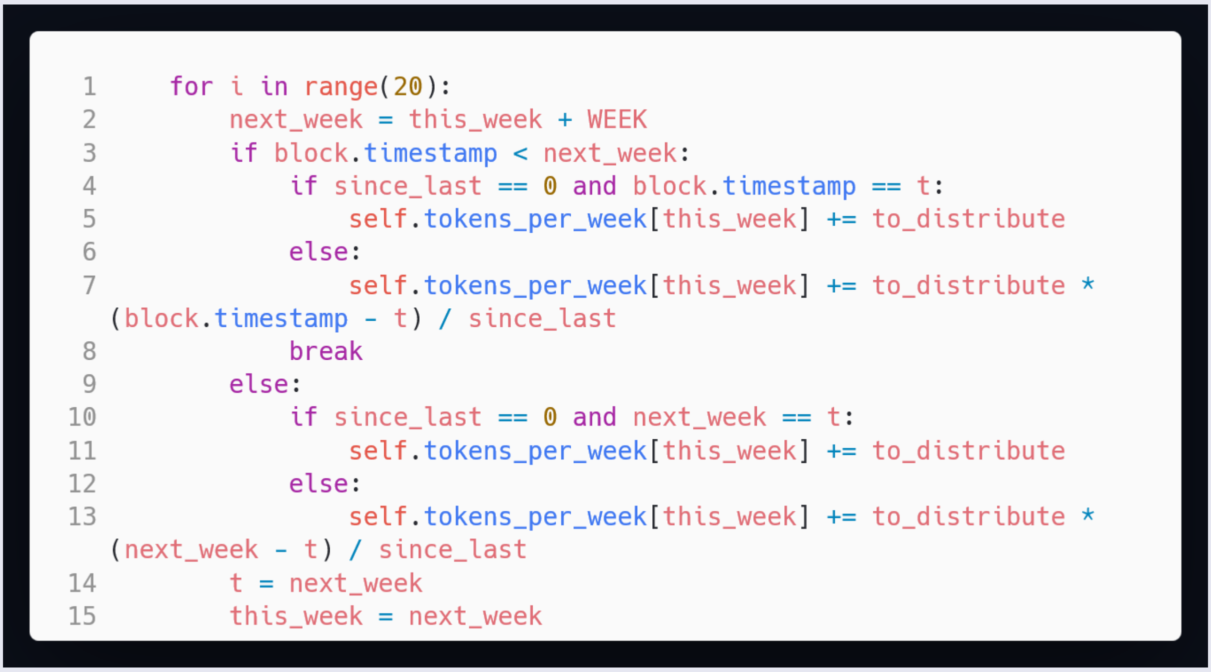
Use Safe Transfer

FINDING ID	#0005
SEVERITY	Low Risk
STATUS	Closed
LOCATION	• fee-distributor.vy -> 350: *assert ERC20(token).transfer(_addr, amount)* • fee-distributor.vy -> 351: *assert ERC20(token).transfer(self.treasury, self.fee_unit)* • fee-distributor.vy -> 397: *assert ERC20(token).transfer(addr, amount)* • fee-distributor.vy -> 402: *assert ERC20(token).transfer(self.treasury, total_fee)* • fee-distributor.vy -> 418: *ERC20(_coin).transferFrom(msg.sender, self, amount)* • fee-distributor.vy -> 471: *assert ERC20(token).transfer(self.emergency_return, ERC20(token).balanceOf(self))* • veTnode.vy -> 378: *assert ERC20(self.token).transferFrom(_addr, self, _value)* • veTnode.vy -> 513: *assert ERC20(self.token).transfer(msg.sender, value)*

DESCRIPTION	Direct transfer functions are called.
RECOMMENDATION	Use safe transfer functions. These safe transfer function are used to catch when a transfer fails as well as unusual token behaviour.
RESOLUTION	The project team implemented a low level safe transfer call that ensures a successful transfer. The low level calls could be moved to new functions instead of using duplicated code.

Rewards Not Allocated If Last Reward Was Over 20 Weeks Ago

FINDING ID	#0006
SEVERITY	Low Risk
STATUS	Partially Closed
LOCATION	fee-distributor.vy -> 132-146

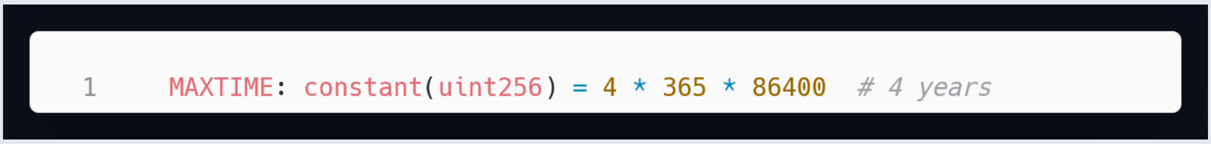


```
for i in range(20):
    next_week = this_week + WEEK
    if block.timestamp < next_week:
        if since_last == 0 and block.timestamp == t:
            self.tokens_per_week[this_week] += to_distribute
        else:
            self.tokens_per_week[this_week] += to_distribute * (block.timestamp - t) / since_last
        break
    else:
        if since_last == 0 and next_week == t:
            self.tokens_per_week[this_week] += to_distribute
        else:
            self.tokens_per_week[this_week] += to_distribute * (next_week - t) / since_last
    t = next_week
    this_week = next_week
```

DESCRIPTION	The tokens per week are not allocated correctly if the number of weeks to reward will exceed 20 weeks. Note that the calculation logic of the *tokens_per_week* is highly inconsistent. For example, some of the branches will never be executed.
RECOMMENDATION	Consolidate the reward distribution logic and increase the bounds or make sure function is called every 20 weeks.
RESOLUTION	The range has been updated to 30 weeks. Vyper does not allow flexible ranged loops which might cause contracts to run into gas limits. The project team has to balance between lower gas estimation vs loss of rewards when the contract is highly inactive. Project team comment: “This function must be called more than once per week based on its logic. If it’s not called over 30 weeks, this means the project has been stopped for over half a year”

365 Days Will Cause Offset

FINDING ID	#0007
SEVERITY	Informational
STATUS	Open
LOCATION	veTnode.vy -> 86

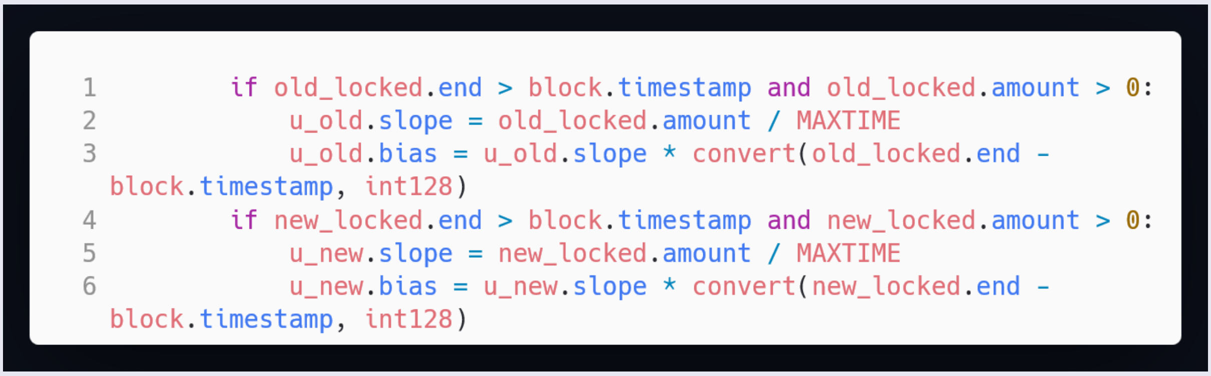


```
MAXTIME: constant(uint256) = 4 * 365 * 86400  # 4 years
```

DESCRIPTION	Since there is a different number of days in leap years, this will cause an offset.
RECOMMENDATION	Keep this in mind when designing the front-end.
RESOLUTION	Project team comment: “Will consider it in frontend.”

Division Before Multiplication

FINDING ID	#0008
SEVERITY	Informational
STATUS	Closed
LOCATION	veTnode.vy -> 251-256

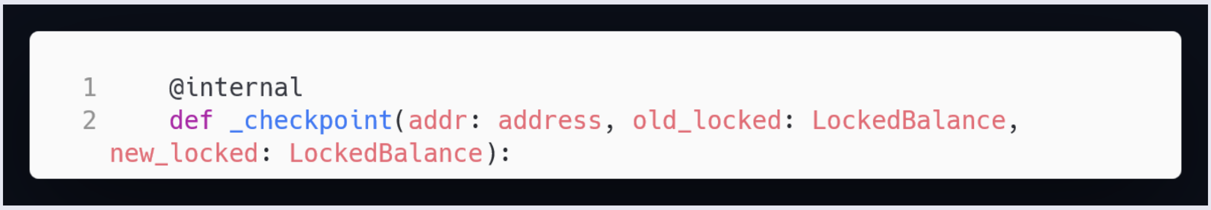


```
        if old_locked.end > block.timestamp and old_locked.amount > 0:
            u_old.slope = old_locked.amount / MAXTIME
            u_old.bias = u_old.slope * convert(old_locked.end - block.timestamp, int128)
        if new_locked.end > block.timestamp and new_locked.amount > 0:
            u_new.slope = new_locked.amount / MAXTIME
            u_new.bias = u_new.slope * convert(new_locked.end - block.timestamp, int128)
```

DESCRIPTION	The calculations noted use mixed orders of multiplication and division. This may cause rounding errors, resulting in reverted transactions or miscalculations in general.
RECOMMENDATION	Change the calculations to first multiply, then divide.
RESOLUTION	The multiplication is now done in a correct order to avoid rounding errors.

Overall And User Checkpoint Updated Simultaneously

FINDING ID	#0009
SEVERITY	Informational
STATUS	Open
LOCATION	veTnode.vy -> 235

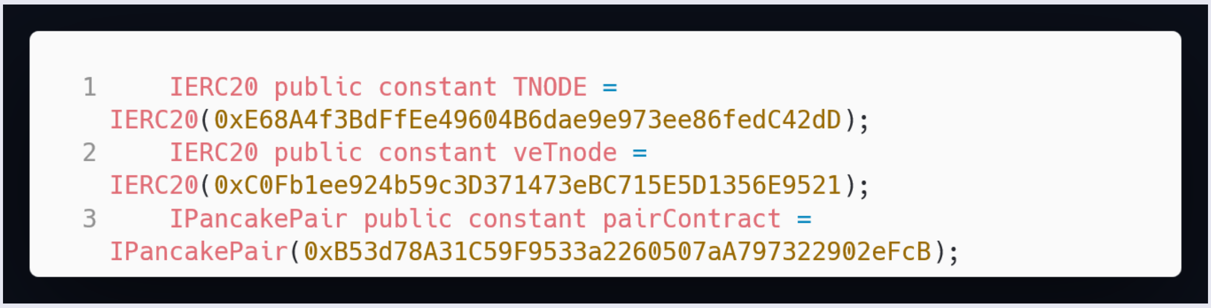


```
@internal
def _checkpoint(addr: address, old_locked: LockedBalance, new_locked: LockedBalance):
```

DESCRIPTION	The *veTnode.vy* contract updates the overall and user checkpoints at the same time. This leads to complex logic which may cause unpredictable behaviour.
RECOMMENDATION	Separate and simplify for these systems.
RESOLUTION	Project team comment: “Will consider it.”

Addresses Are Hard Coded

FINDING ID	#0010
SEVERITY	Informational
STATUS	Closed
LOCATION	Vault.sol -> 21-23

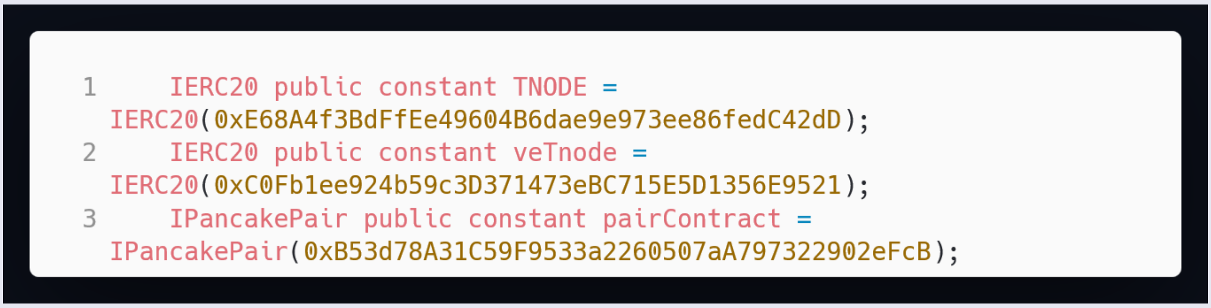


```
    IERC20 public constant TNODE = IERC20(0xE68A4f3BdFfEe49604B6dae9e973ee86fedC42dD);
    IERC20 public constant veTnode = IERC20(0xC0Fb1ee924b59c3D371473eBC715E5D1356E9521);
    IPancakePair public constant pairContract = IPancakePair(0xB53d78A31C59F9533a2260507aA797322902eFcB);
```

DESCRIPTION	The noted addresses are hard coded.
RECOMMENDATION	Add parameters to the constructor to allow for more flexible deployment.
RESOLUTION	The project team implemented the recommended changes.

Constants Using Testnet Addresses

FINDING ID	#0011
SEVERITY	Informational
STATUS	Closed
LOCATION	Vault.sol -> 21-23



```
    IERC20 public constant TNODE = IERC20(0xE68A4f3BdFfEe49604B6dae9e973ee86fedC42dD);
    IERC20 public constant veTnode = IERC20(0xC0Fb1ee924b59c3D371473eBC715E5D1356E9521);
    IPancakePair public constant pairContract = IPancakePair(0xB53d78A31C59F9533a2260507aA797322902eFcB);
```

DESCRIPTION	The addresses above are from Binance Smart Chain Public Testnet.
RECOMMENDATION	Replace them with the mainnet addresses.
RESOLUTION	Hard coded addresses are removed.

Missing Zero Checks

FINDING ID	#0012
SEVERITY	Informational
STATUS	Closed
LOCATION	● fee-distributor.vy -> 74-81: *def __init__(_voting_escrow: address,_start_time: uint256,_token: address,_admin: address,_emergency_return: address,_treasury: address):* ● fee-distributor.vy -> 112: *def setTreasury(_treasury: address):`* ● Vault.sol -> 52-56: *`constructor(address _token, address _treasury) {* ● Vault.sol -> 58: *function setDistribution(address _distribution) external onlyOwner {* ● Vault.sol -> 62 *function setTreasury(address _treasury) external onlyOwner {* ● veTnode.vy -> 119 *def __init__(token_addr: address, _name: String[64], _symbol: String[32], _version: String[32]):*

DESCRIPTION	The aforementioned functions can set addresses to zero address. Zero addresses may cause incorrect contract behavior.
RECOMMENDATION	Add a check to ensure contract values are never set to an invalid zero address.
RESOLUTION	The project team implemented the recommended changes.

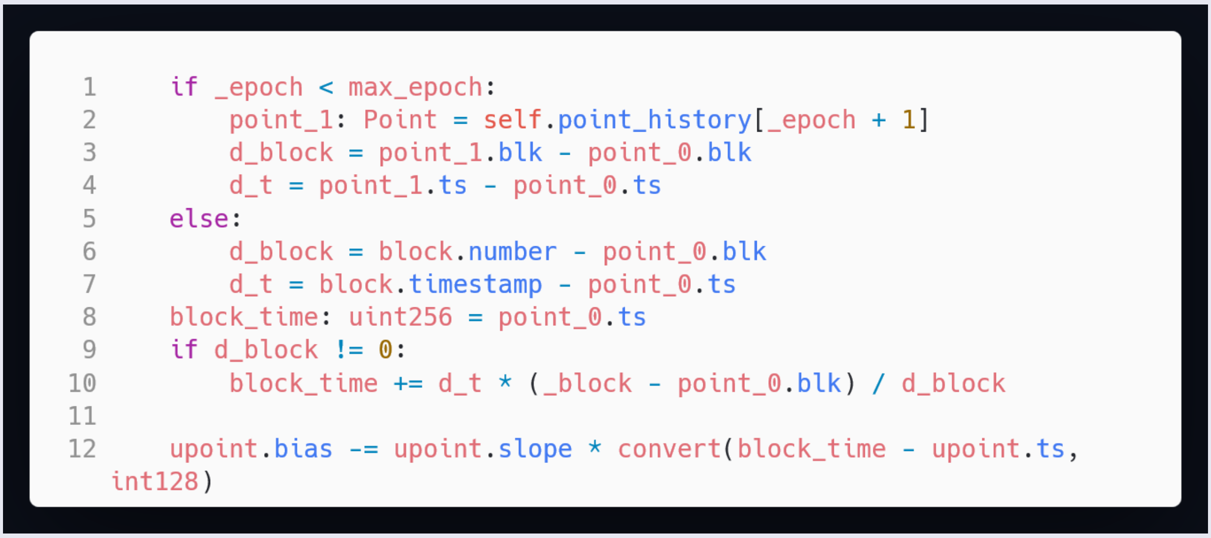
No Events Emitted For Changes To Protocol Values

FINDING ID	#0013
SEVERITY	Informational
STATUS	Closed
LOCATION	● Vault.sol -> 58-60: *function setDistribution(address _distribution) external onlyOwner* ● Vault.sol -> 62-64: *function setTreasury(address _treasury) external onlyOwner* ● fee-distributor.vy -> 103-109: *def setFeeUnit(_fee_unit: uint256)* ● fee-distributor.vy -> 111-118: *def setTreasury(_treasury: address)*

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 implemented the recommended changes.

Block Rate Assumed Constant

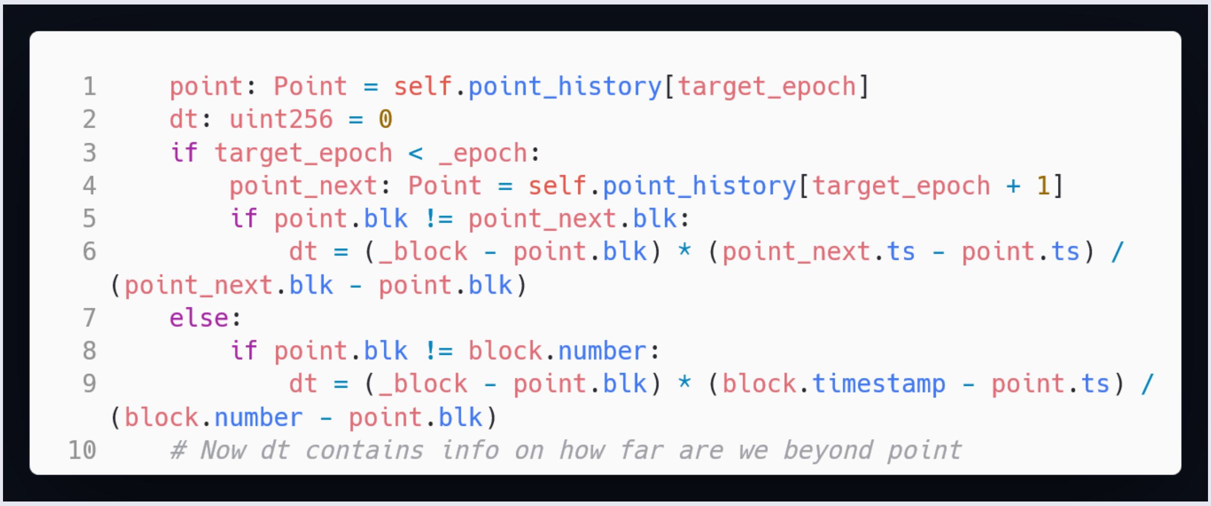
FINDING ID	#0014
SEVERITY	Informational
STATUS	Open
LOCATION	veTnode.vy -> 600-611



```
    if _epoch < max_epoch:
        point_1: Point = self.point_history[_epoch + 1]
        d_block = point_1.blk - point_0.blk
        d_t = point_1.ts - point_0.ts
    else:
        d_block = block.number - point_0.blk
        d_t = block.timestamp - point_0.ts
    block_time: uint256 = point_0.ts
    if d_block != 0:
        block_time += d_t * (_block - point_0.blk) / d_block

    upoint.bias -= upoint.slope * convert(block_time - upoint.ts, int128)
```

LOCATION	veTnode.vy -> 672-681



```
    point: Point = self.point_history[target_epoch]
    dt: uint256 = 0
    if target_epoch < _epoch:
        point_next: Point = self.point_history[target_epoch + 1]
        if point.blk != point_next.blk:
            dt = (_block - point.blk) * (point_next.ts - point.ts) / (point_next.blk - point.blk)
    else:
        if point.blk != block.number:
            dt = (_block - point.blk) * (block.timestamp - point.ts) / (block.number - point.blk)
    # Now dt contains info on how far are we beyond point
```

DESCRIPTION	The calculation of a timestamp for a given block number is done by interpolating between the values in saved *Point* objects. This assumes that the rate of blocks is constant.
RECOMMENDATION	Be aware that the block timestamp may differ when designing the front-end or associated contracts.
RESOLUTION	Project team comment: “Will consider it in frontend.”

On-Chain Analysis

Not Analyzed Yet

External Addresses

Externally Owned Accounts

Owner

ACCOUNT	Address
USAGE	0x123456… *Contract.owner* - Variable
IMPACT	● receives elevated permissions as owner, operator, or other

External Contracts

These contracts are not part of the audit scope.

Some Vault

ADDRESS	[ETH - 0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2]
USAGE	0x123456... *SomeContract.Vault* - Constant
IMPACT	● ERC20 Token

External Tokens

These contracts are not part of the audit scope.

Wrapped Ether

ADDRESS	ETH - 0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2
USAGE	0x123456... *SomeContract.WETH* - Constant
IMPACT	● ERC20 Token

Appendix A - Reviewed Documents

Deployed Contracts

Document	Address
	N/A
	N/A
	N/A
	N/A
	N/A
	N/A
	N/A
	N/A
	N/A
	N/A
	N/A
	N/A

Libraries And Interfaces

Revisions

Revision 1	Hash

Imported Contracts

Contracts	Version

Appendix B - Risk Ratings

Risk	Description
High Risk	Security risks that are ***almost certain*** to lead to ***impairment or loss of funds***. Projects are advised to fix as soon as possible.
Medium Risk	Security risks that are ***very likely*** to lead to ***impairment or loss of funds*** with ***limited impact***. Projects are advised to fix as soon as possible.
Low Risk	Security risks that can lead to ***damage to the protocol***. Projects are advised to fix. Issues with this rating might be used in an exploit with other issues to cause significant damage.
Informational	Noteworthy information. Issues may include code conventions, missing or conflicting information, gas optimizations, and other advisories.

Appendix C - Finding Statuses

Closed	Contracts were modified to permanently resolve the finding.
Mitigated	The finding was resolved on-chain. The issue may require monitoring, for example in the case of a time lock.
Partially Closed	Contracts were modified to partially fix the issue
Partially Mitigated	The finding was resolved by project specific methods which cannot be verified on chain. Examples include compounding at a given frequency, or the use of a multisig wallet.
Open	The finding was not addressed.

Appendix D - Glossary

Contract Structure

Contract: An address with which provides functionality to users and other contracts. They are implemented in code and deployed to the blockchain.
Protocol: A system of contracts which work together.
Stakeholders: The users, operators, owners, and other participants of a contract.

Security Concepts

Bug**:** A defect in the contract code.
Exploit**:** A chain of events involving bugs, vulnerabilities, or other security risks which damages a protocol.
Funds: Tokens deposited by users or other stakeholders into a protocol.
Impairment: The loss of functionality in a contract or protocol.
Security risk: A circumstance that may result in harm to the stakeholders of a protocol. Examples include vulnerabilities in the code, bugs, excessive permissions, missing timelock, etc.
Vulnerability: A vulnerability is a flaw that allows an attacker to potentially cause harm to the stakeholders of a contract. They may occur in a contract's code, design, or deployed state on the blockchain.

Appendix E - 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