


OBELISK

Part of Tibereum Group

# AUDITING REPORT

# Version Notes

| Version | No. Pages | Date | Revised By | Notes |
|---|---|---|---|---|
| 1.0 | Total: 25 | 2021-06-09 | Zapmore, Donut | |

# Audit Notes

| | |
|---|---|
| Audit Date | 2021-06-01 - 2021-06-09 |
| Auditor/Auditors | Donut, Plemonade, Hebilicious |
| Auditor/Auditors Contact Information | tibereum-obelisk@protonmail.com |
| Notes | Specified code and contracts are audited for security flaws.<br>UI/UX (website), logic, team, and tokenomics are not audited. |
| Audit Report Number | OB58875986 |

# 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.

# Table of Content

# Project Information

| | |
|---|---|
| Project Name | Kogefarm |
| Description | KogeFarm is a utility that auto-compounds high APR farms on Polygon/Matic. |
| Website | https://kogefarm.io/ |
| Contact | @kevinf1 |
| Contact information | @kevinf1 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 Kogefarm was conducted by three of Obelisks' security experts between the 1st of June 2021 and the 9th of June 2021.

**After finishing the full audit, Obelisk auditing can say that there was a medium issue found in the emergency withdrawal function which was mitigated. Also, low severity issues were found in the harvesting of rewards. Other Informational findings pose no risk to deposited funds while used correctly and not in malicious hands.**

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

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

## Summary Table

| Audited Part | Severity | Note |
|---|---|---|
| Emergency Withdraw Does Not Pause The Vault | Medium Risk | Mitigated |
| Only EOA Can Use Harvest Or Authorized Accounts | Low Risk | N/A |
| Harvest Authorization Can Be Changed | Informational | See Comment |
| Deposit Does Not Allow Contracts But Withdraw Does | Informational | N/A |
| Disabling Functionality In User Interface | Informational | N/A |
| Beware Of withdrawForSwap | Informational | Mitigated |
| Compiled Contract Does Not Match File Name | Informational | See Comment |
| Two Different Versions Of Contract StrategyIronBase | Informational | Mitigated |
| Possible Frontrunning On Harvesting Rewards | Low Risk | Mitigated |
| Minor Changes To Deployed Contracts | Informational | N/A |

# Introduction

Obelisk was commissioned by Kogefarm on the 1st of June 2021 to conduct a comprehensive audit of Kogefarm's vaults and pool on Polygon/Matic. The following audit was conducted between the 1st of June 2021 and the 9th of June 2021 and delivered on the 9th of June 2021. Three of Obelisk's security experts went through the related contracts using industry standards to find if any vulnerabilities could be exploited.

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

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.

The contracts were checked for vulnerabilities prior to deployment. The audit found a medium risk issue in the contracts related to emergency withdrawal and a low risk issue related to the harvest of rewards.  The medium risk issue was addressed prior to deployment.  Other informational findings are just for information and are not a security threat as long as the contracts are not used in a malicious way.

The contracts were checked once after they were deployed to ensure they are the same as the audited contracts. Several changes were made to the deployed contracts. One low risk issue, possible frontrunning on the harvested rewards, was identified in the changes. This issue can be considered mitigated by the project team's actions. The remaining changes were determined to be minor, with no security implications.

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

# Findings

## Manual Analysis

### Emergency Withdraw Does Not Pause The Vault

| | |
|---|---|
| SEVERITY | Medium Risk |
| RESOLVED | **YES** |
| LOCATION | strategyBase.sol -> 1371-1378<br>strategySingleAssets.sol -> 1373-1380<br>strategyTwoAssets.sol -> 1371-1378<br><br>Note: Code is identical at indicated locations. |

```
1         // If withdrawal penalty, subtract before transferring
2         // Otherwise, just comment out this section
3         address masterChefAddr = IStrategy(strategy).rewards();
4         MasterChef underlyingMC = MasterChef(masterChefAddr);
5         // Read pool info on deposit fee
6         uint256 withdrawPenalty = underlyingMC.withdrawPenalty();
7         if (withdrawPenalty>0) {
8             uint256 WithdrawalFee = r.mul(withdrawPenalty).div(keepMax);
9             r = r.sub(WithdrawalFee);
10        }
```

| | |
|---|---|
| DESCRIPTION | The emergency withdrawal function is expected to be used when there is a fault with the underlying vault. If this occurs, deposits should be paused. |
| RECOMMENDATION | Add a pause function which calls the emergency withdrawal and disables deposits. Add an unpause function to re-deposit the withdrawn tokens and re-enable deposits.<br><br>Ensure that the withdrawPenalty is defined as an immutable value since a high withdrawPenalty can still affect withdraws after an emergency withdraw. |
| MITIGATED/COMMENT | The Kogefarm team has mitigated this vulnerability by |

| | |
|---|---|
| | preventing deposits after a vault activates its emergency withdraw, and by disabling the impact of the withdraw fee during an emergency. As a result, this issue can be considered mitigated.<br><br>Note: There is currently no way to reset the emergency status and as a result the vaults would need to be redeployed. |

## Only EOA Can Use Harvest Or Authorized Accounts

| | |
|---|---|
| SEVERITY | Low Risk |
| RESOLVED | **NO** |
| LOCATION | strategyBase.sol -> 1436<br>strategySingleAssets.sol -> 1444<br>strategyTwoAssets.sol -> 1440<br><br>Note: Code is identical at indicated locations. |

```
1   function harvest() public override {
2         //prevent unauthorized smart contracts from calling harvest()
3         require(msg.sender == tx.origin || msg.sender == owner() || msg.sender == strategist ||
  msg.sender == multiHarvest, "not authorized");
```

| | |
|---|---|
| DESCRIPTION | Only an EOA(externally Owned address) or authorized address(owner, strategist, mulitHarvest) can use the harvest function. This prevents smart contracts from calling but also prevents multi-signature wallets like gnosis safe from calling it. Be aware that tx.origin might not stay useful. Vitalik has said: "Do NOT assume that tx.origin will continue to be usable or meaningful." |
| RECOMMENDATION | Be aware of the downsides of using tx.origin and watch out for changes concerning tx.origin. Use OpenZeppelin Address.isContract to check for contracts. |
| MITIGATED/COMMENT | Project Team Comment: “We understand and accept this behavior.” |

## Harvest Authorization Can Be Changed

| | |
|---|---|
| SEVERITY | Informational |
| RESOLVED | **NO** |
| LOCATION | strategyBase.sol -> 1426<br>strategySingleAssets.sol -> 1434<br>strategyTwoAssets.sol -> 1430<br><br>Note: Code is identical at indicated locations. |

```
1    function set_multiHarvest(address newHarvest) external onlyOwner() {
2        multiHarvest = newHarvest;
3    }
```

| | |
|---|---|
| DESCRIPTION | If a malicious user gains access to the owner account, they are able to set a new multiHarvest account and bypass the restrictions of harvest with a malicious contract. The owner and strategist can also bypass it directly when using a contract. |
| RECOMMENDATION | Consider limiting the permission of these users and placing set multiharvest under a timelock. This will ensure that no frontrunning of earnings can occur in the harvest function. |
| MITIGATED/COMMENT | Project Team Comment: “We understand and accept this behavior. Since the only risk is front-running of the harvested rewards, we feel that this issue can be mitigated by notifying our users should a breach occur."<br><br>Obelisk Comment: “Agreed.” |

## Deposit Does Not Allow Contracts But Withdraw Does

| | |
|---|---|
| SEVERITY | Informational |
| RESOLVED | **NO** |
| LOCATION | Vaultbase.sol -> 1127 |

```
1 function deposit(uint256 _amount) public {
2         require(msg.sender == tx.origin, "no contracts");
3         lastTimeStaked[msg.sender] = now;
```

| | |
|---|---|
| DESCRIPTION | Deposit checks for tx.origin in order to prevent contracts from calling but withdraw does not. This restriction will not prevent contracts from interacting with the vault as a user may deposit then transfer their share tokens to a contract afterward. |
| RECOMMENDATION | Consider whether the restriction to prevent contracts from calling is necessary. If so, ensure both withdrawal and deposit functions use the same check. The functionality of tx.origin is also liable to change in the future. Use OpenZeppelin Address.isContract to check for contracts. |
| MITIGATED/COMMENT | Project Team Comment: “We understand and accept this behavior.” |

## Disabling Functionality In User Interface

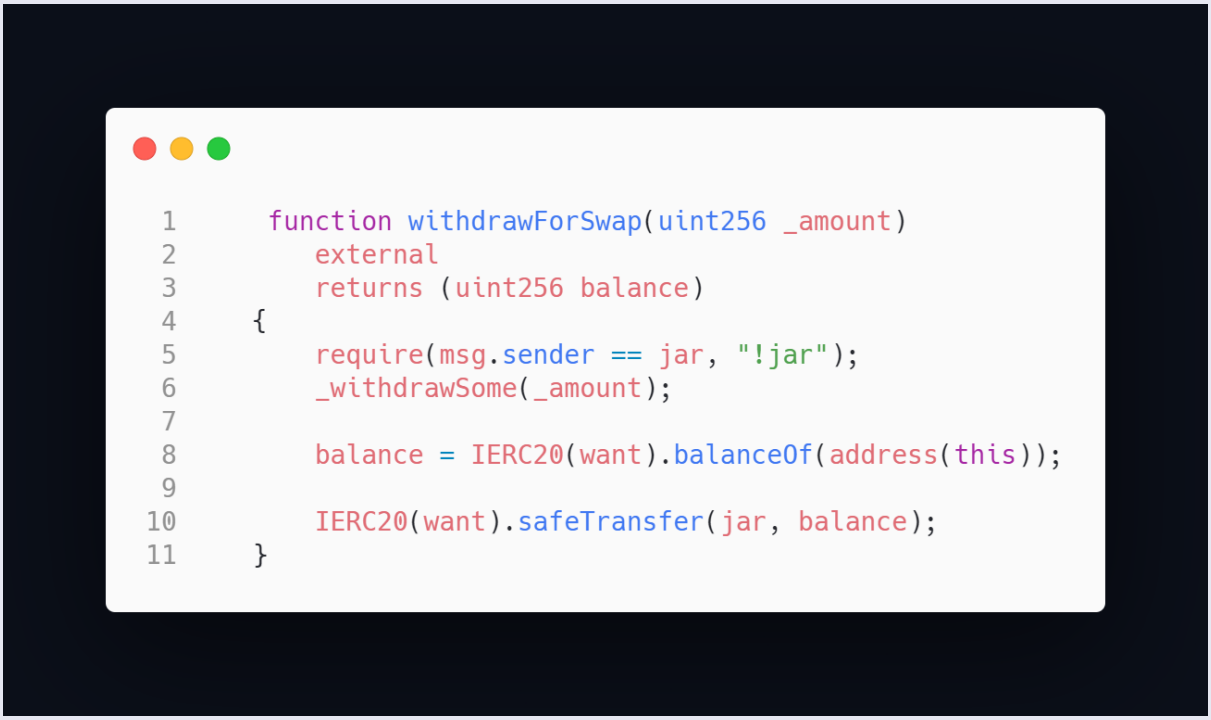
| | |
|---|---|
| SEVERITY | Informational |
| RESOLVED | **NO** |
| LOCATION | VaultBase.sol -> 1205 |

```
1    //Website UI will hide the restake button for a user if it's been less than 1 day since they
  last restaked
2    function getLastTimeRestaked(address _address) public view returns (uint256) {
3        return lastTimeRestaked[_address];
4    }
5
6    //Website UI will hide the restake button for a user if they staked after the migration
7    function getLastTimeStaked(address _address) public view returns (uint256) {
8        return lastTimeStaked[_address];
9    }
```

| | |
|---|---|
| DESCRIPTION | Comments suggest that functionality will be disabled via the user interface. This will not prevent interactions via direct calls or the explorer. |
| RECOMMENDATION | Consider implementing a lock such that users can't restake regardless of the user interface. |
| MITIGATED/COMMENT | Project Team Comment: “We understand and accept this behavior.” |

## Beware Of withdrawForSwap

| | |
|---|---|
| SEVERITY | Informational |
| RESOLVED | **YES** |
| LOCATION | StrategyBase.sol -> 1187<br>StrategySingleAsset.sol -> 1187<br>StrategyTwoAssets.sol -> 1185<br><br>Note: Code is identical at indicated locations. |

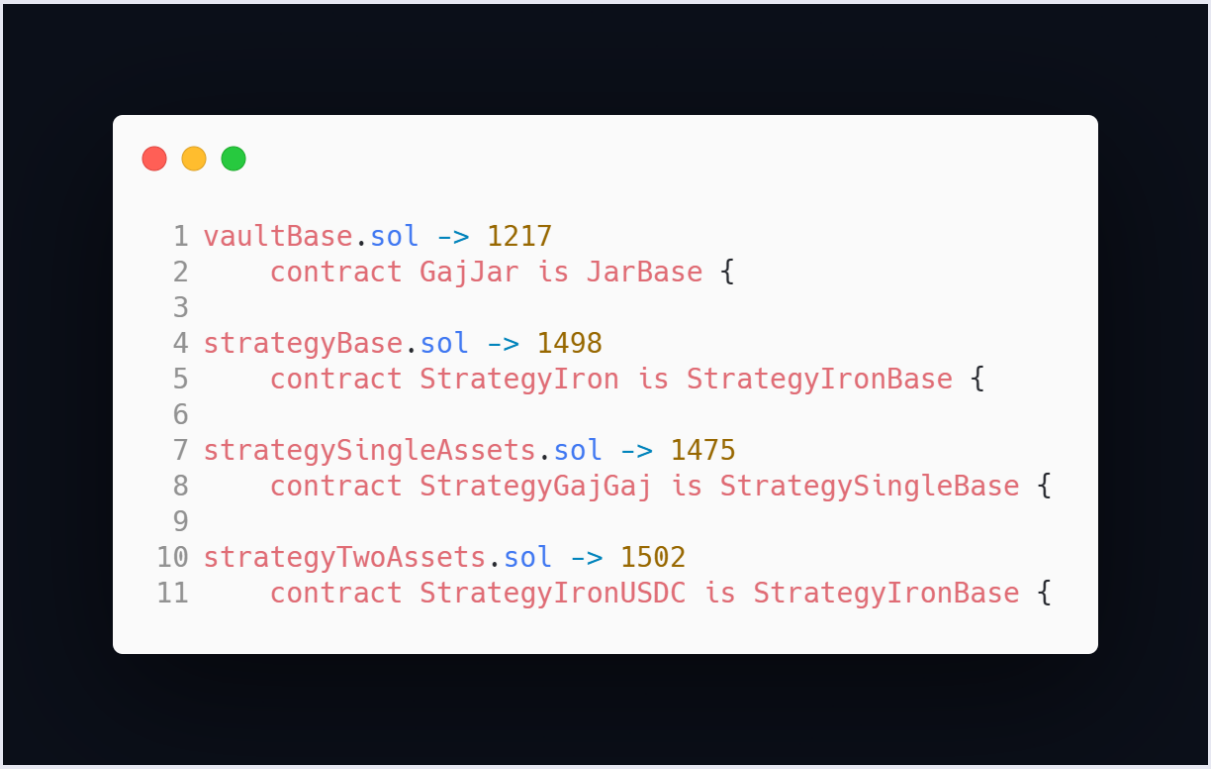


```
1     function withdrawForSwap(uint256 _amount)
2         external
3         returns (uint256 balance)
4     {
5         require(msg.sender == jar, "!jar");
6         _withdrawSome(_amount);
7
8         balance = IERC20(want).balanceOf(address(this));
9
10        IERC20(want).safeTransfer(jar, balance);
11    }
```

| | |
|---|---|
| DESCRIPTION | The function withdrawForSwap is no longer commonly used in similar jar-based contracts. The function has been the target of exploits in the past, such as the Evil Jar exploit. This function's vulnerability is that it may allow a malicious attacker to withdraw tokens from the vault. |
| RECOMMENDATION | Consider removing withdrawForSwap as it's not currently used. If retained, be very careful that withdrawForSwap cannot be called in an unintended way. |
| MITIGATED/COMMENT | Project Team Comment: “This function has been removed in an abundance of caution.” |

## Compiled Contract Does Not Match File Name

| | |
|---|---|
| SEVERITY | Informational |
| RESOLVED | **PARTIAL** |
| LOCATION | See code window |



```
1 vaultBase.sol -> 1217
2     contract GajJar is JarBase {
3
4 strategyBase.sol -> 1498
5     contract StrategyIron is StrategyIronBase {
6
7 strategySingleAssets.sol -> 1475
8     contract StrategyGajGaj is StrategySingleBase {
9
10 strategyTwoAssets.sol -> 1502
11     contract StrategyIronUSDC is StrategyIronBase {
```

| | |
|---|---|
| DESCRIPTION | The compiled contract in each file does not match the name of the file. |
| RECOMMENDATION | Rename contract files to match their compiled contract. |
| MITIGATED/COMMENT | Contracts have been renamed:<br><br>vaultBase.sol: contract vaultBase<br>strategyBase.sol: contract StrategyBase<br>strategySingleAsset.sol: contract StrategySingle<br>strategyTwoAssets.sol: contract StrategyFarmTwoAssets |

## Two Different Versions Of Contract StrategyIronBase

| | |
|---|---|
| SEVERITY | Informational |
| RESOLVED | **YES** |
| LOCATION | strategyBase.sol -> 1382-1496<br>strategyTwoAssets.sol -> 1382-1496 |

```
1 abstract contract StrategyIronBase is BaseStrategyMasterChef {
2     // [...]
3 }
```

| | |
|---|---|
| DESCRIPTION | Contract StrategyIronBase differs between the two noted files. Contracts should be consistent for clarity and to prevent compilation errors. |
| RECOMMENDATION | Separate the contracts into separate files to be imported and reused. |
| MITIGATED/COMMENT | Contracts have been renamed. |

## Static Analysis

No Findings

# On-Chain Analysis

## Possible Frontrunning On Harvesting Rewards

| | |
|---|---|
| SEVERITY | Low Risk |
| RESOLVED | **YES** |
| LOCATION | Various locations. See description. |
| DESCRIPTION | Contracts were changed prior to deployment:<br><br>*vaultBase.sol* compared with *IronUsdcSushiBase*:<br>At 0xBDdC52842add45eAe21FeaF06f5c348a157cd148<br>- Deposit and withdraw fees are disabled.<br><br>*vaultBase.sol* compared with *IronUsdcSushiBase*:<br>At 0x3fae5e941B7eb3A7BeE94399bF669224efa9432C<br>- Deposit and withdraw fees are disabled.<br><br>*vaultBase.sol* compared with *vaultBase (FishMaticSushi)*:<br>At 0xEa2F645691d114f0A7Fa7a759032F8c6f90D58d5<br>- Withdraw fees are disabled.<br><br>*vaultBase.sol* compared with *vaultBase (Fish)*:<br>At 0x05d83F3Ef95F921971763b035c00298BC42ff008<br>- Withdraw fees are disabled.<br><br>Deposit and withdraw fees are commonly used to prevent frontrunning on rewards. A malicious actor can detect when the reward distribution functions are called in order to deposit immediately before rewards are harvested, then withdraw immediately after. This can reduce the expected returns for users. |
| RECOMMENDATION | Ensure that the rewards are harvested frequently enough that front-running will not be viable. |
| MITIGATED/COMMENT | Project Team Comment: "we call harvest every minute so the front run risk is low." |

## Minor Changes To Deployed Contracts

| | |
|---|---|
| SEVERITY | Informational |
| RESOLVED | **NO** |
| LOCATION | Various locations. See description. |
| DESCRIPTION | *vaultBase.sol* compared with *IronUsdcSushiBase*:<br>At 0xBDdC52842add45eAe21FeaF06f5c348a157cd148<br>- View function getRatio checks for potential division by 0.<br>- vaultBase is renamed.<br><br>*vaultBase.sol* compared with *IronUsdcSushiBase*:<br>At 0x3fae5e941B7eb3A7BeE94399bF669224efa9432C<br>- View function getRatio checks for potential division by 0.<br>- vaultBase is renamed.<br><br>*vaultBase.sol* compared with *vaultBase (FishMaticSushi)*:<br>At 0xEa2F645691d114f0A7Fa7a759032F8c6f90D58d5<br>- View function getRatio checks for potential division by 0.<br><br>*vaultBase.sol* compared with *vaultBase (Fish)*:<br>At 0x05d83F3Ef95F921971763b035c00298BC42ff008<br>- View function getRatio checks for potential division by 0.<br><br>*strategyTwoAssets.sol* compared with *StrategyTwoAssets (Sushi)*:<br>At 0x99d41644b80a9c44715e263f61387a9351901647<br>- Router address is changed<br>- Underlying pool id is changed<br>- Trade paths are changed<br>- Reward and LP address are changed<br><br>*strategyBase.sol* compared with *StrategyBase (FishMaticSushi)*<br>At 0xc2386546A9710c00a0383d896A861154D45Dac94<br>- Trade paths are changed<br>- Reward and LP address are changed<br><br>*strategySingleAsset.sol* compared with *StrategySingle (Fish)*<br>At 0x78A714E456F6466530d0D7d882a8A3A922849fDd |

| | |
|---|---|
| | - Default fee rate is changed<br>- Underlying pool id is changed<br>- Trade paths are changed<br>- Reward and LP address are changed<br>- Strategist is used as a referral |
| RECOMMENDATION | These changes are minor and do not represent any security risk. No further action is necessary |
| MITIGATED/COMMENT | N/A |

# Appendix A - Reviewed Documents

| **Document** | **Address** |
|---|---|
| strategyBase.sol | *Fish-Matic (Sushi)*<br>0xc2386546A9710c00a0383d896A861154D45Dac94 |
| strategySingleAsset.sol | *Fish*<br>0x78a714e456f6466530d0d7d882a8a3a922849fdd |
| strategyTwoAssets.sol | *Iron-USDC (Sushi)*<br>0x99d41644b80a9c44715e263f61387a9351901647<br><br>*Iron-USDC (Quick)*<br>0xD40775038b8eD30AD8cE754F716e5C2701845A93 |
| vaultBase.sol | *Iron-USDC (Sushi)*<br>0xBDdC52842add45eAe21FeaF06f5c348a157cd148<br><br>*Iron-USDC (Quick)*<br>0x3fae5e941B7eb3A7BeE94399bF669224efa9432C<br><br>*Fish-Matic (Sushi)*<br>0xEa2F645691d114f0A7Fa7a759032F8c6f90D58d5<br><br>*Fish*<br>0x05d83F3Ef95F921971763b035c00298BC42ff008 |

# 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 - Icons

| Icon | Explanation |
| --- | --- |
|  | Solved by Project Team |
|  | Under Investigation of Project Team |
|  | Unsolved |

# Appendix D - 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.

# OBELISK

Part of Tibereum Group