OBELISK

OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 27	2021-09-27	Zapmore, Donut	Audit Final

Audit Notes

Audit Date	2021-08-25 - 2021-09-23
Auditor/Auditors	Donut, Zenith
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	OB581242157

Disclaimer

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

Obelisk Auditing

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

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

Audit Information

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

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

Table of Contents

Project Information

Project Name	Kukafe
Description	KuKafe is one of the first cross-DEX yield farm on the Kucoin Community Chain (KCC) focused on offering the highest sustainable yield farming APRs.
Website	https://kukafe.finance/#/
Contact	@kukafeBarista
Contact information	@kukafeBarista on TG
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	KCC

Executive Summary

The audit of Kukafe was conducted by two of Obelisks' security experts between the 25th of August 2021 and the 23rd of September 2021.

After auditing Kukafe, there were multiple security findings found. Obelisk updated Kukafe with these findings in which Kukafe worked to solve all of the outstanding risk rating issues besides #5 (please see project comment).

Other Informational findings are there for informational purposes and don't impact the project on a larger scale on audited implementation.

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

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

Summary Table

Audited Part	ID	Severity	Resolved
Contract Owner Can Modify Vaults	#0001	Medium Risk	Mitigated
Contract Owner Can Panic And Pause Personal Vault	#0002	Low Risk	Mitigated
Vaults Can Be Created By Anyone	#0003	Low Risk	Mitigated
Buyback Fees Can Be Unlimited	#0004	Low Risk	Mitigated
Swap To Compound Vault Can Be Frontrun	#0005	Low Risk	See Comment
EOA Only Functions May Still Be Called By A Contract	#0006	Low Risk	Mitigated
Local Copy of OpenZeppelin Contract	#0007	Informational	Mitigated
Strategy Harvests Rewards Via Deposit	#0008	Informational	Mitigated
Missing Zero Checks	#0009	Informational	Partial
No Events Emitted For Changes To Protocol Values	#0010	Informational	Mitigated

Introduction

Obelisk was commissioned by Kukafe on the 29th of July 2021 to conduct a comprehensive audit of some of Kukafes’ contracts (see appendix for list). The following audit was conducted between the 25th of August 2021 and the 23rd of September 2021 and delivered on the 27th of September 2021. Two of Obelisk's security experts went through the related contracts using industry standards to find if any vulnerabilities could be exploited.

The audit was conducted on contracts that were not live in a production environment. A comprehensive on-chain analysis was conducted on the live contracts in order to match the audited contracts with the published contracts, which was a correct match.

There were multiple findings of Low and Medium severity that could cause a problem while using the contracts. The project solved all of these outstanding issues besides #5 which is a Low-Risk issue regarding Oracle for rewards where rewards from the swap can be front-run, but which don’t risk users' personal funds (please see project comment).

The informational findings are good to know while interacting with the project but don’t directly damage the project in its current state.

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

Findings

Manual Analysis

Contract Owner Can Modify Vaults

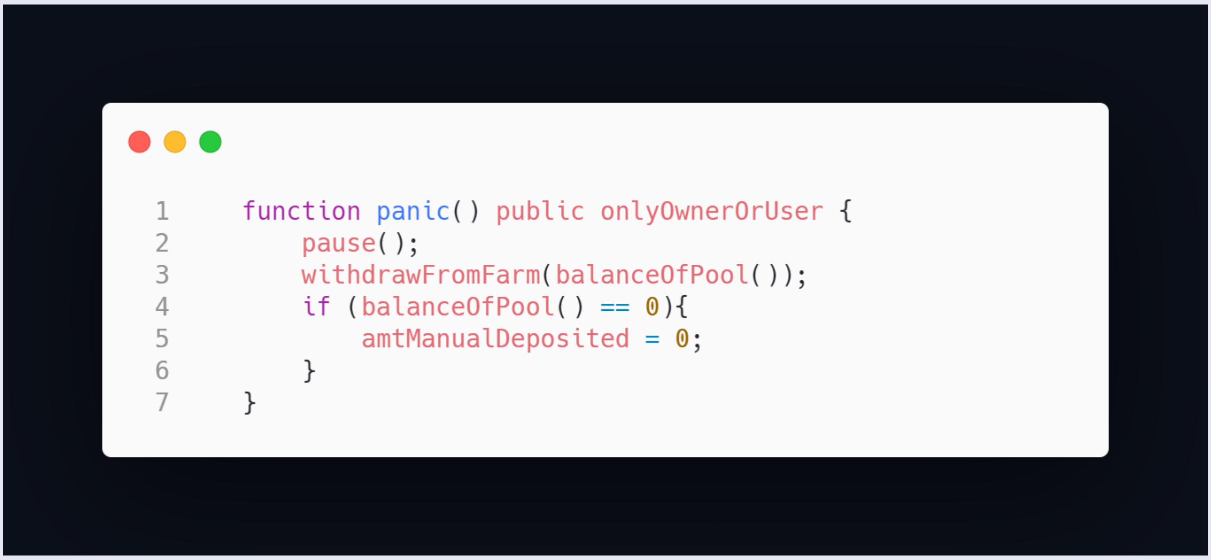
SEVERITY	Medium Risk
RESOLVED	**Mitigated**
FINDING ID	#0001
LOCATION	StrategyLP2.sol -> 88-90 : function setBuybackStrat(address _address) external onlyOwner StrategyLP2.sol -> 91-93 : function setStakingMode(bool _b) external onlyOwner StrategyLP2.sol -> 94-96 : function setReferralMode(bool _b) external onlyOwner StrategyLP2.sol -> 97-99 : function setSwapPathRegistry(address _a) external onlyOwner StrategyLP2.sol -> 100-102 : function setMinToLiquify(uint256 n) external onlyOwner StrategyLPPersonalVault.sol -> 94-96 : function setBuybackStrat(address _address) external onlyOwner StrategyLPPersonalVault.sol -> 97-99 : function setStakingMode(bool _b) external onlyOwner StrategyLPPersonalVault.sol -> 100-102 : function setReferralMode(bool _b) external onlyOwner StrategyLPPersonalVault.sol -> 103-105 : function setSwapPathRegistry(address _a) external onlyOwner StrategyLPPersonalVault.sol -> 106-108 : function setMinToLiquify(uint256 n) external onlyOwner

DESCRIPTION	The contract owner of the personal vault can modify protocol values. Changes to these parameters can break the interaction of the vault with the underlying farm contract.
RECOMMENDATION	Restrict the contract owner's ability to affect the operation of the vault.
MITIGATED/COMMENT	Ownership of the GrowthVaultLaunch, PrivateVaultFactory_LPReferralHarvestLock, and personal

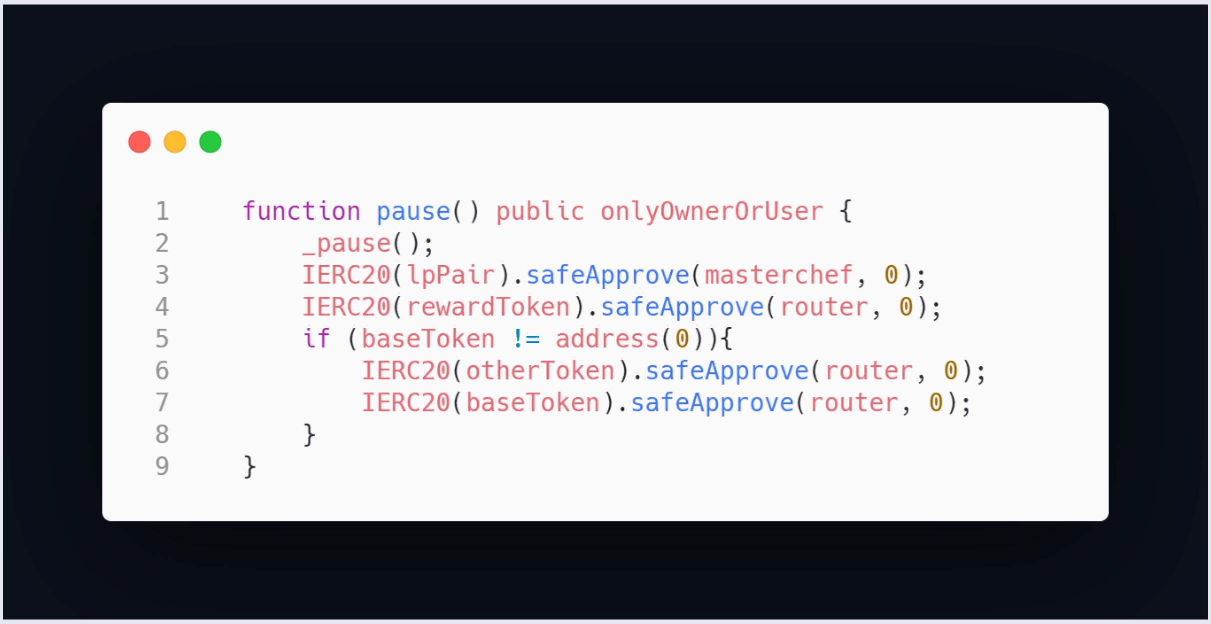
	vaults were transferred to a 72 hours (3 days) timelock. GrowthVaultLaunch 0x79aC1133C9CF22598744C95F8611142Ff23C884a PrivateVaultFactory_LPReferralHarvestLock 0xB29495301434d53BE6d49598DC4D8C7F37a158a5 Timelock 0xa5D65C1215dBa10EdB16B7b40B2a8981ef7f783C

Contract Owner Can Panic And Pause Personal Vault

SEVERITY	Low Risk
RESOLVED	**Mitigated**
FINDING ID	#0002
LOCATION	StrategyLPPersonalVault.sol -> 377-383 StrategyLPPersonalVault.sol -> 395-403



```
function panic() public onlyOwnerOrUser {
    pause();
    withdrawFromFarm(balanceOfPool());
    if (balanceOfPool() == 0){
        amtManualDeposited = 0;
    }
}
```



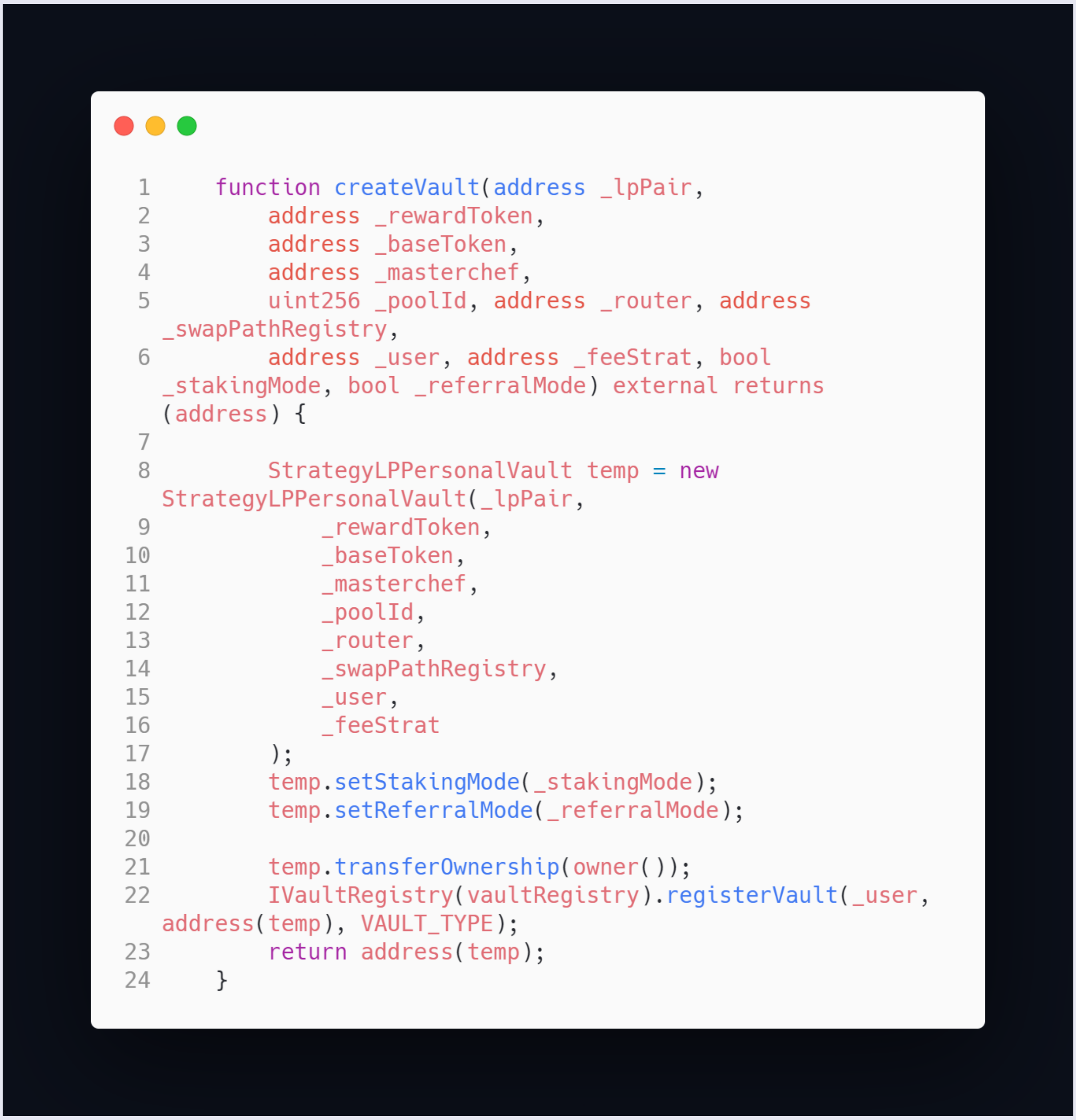
```
function pause() public onlyOwnerOrUser {
    _pause();
    IERC20(lpPair).safeApprove(masterchef, 0);
    IERC20(rewardToken).safeApprove(router, 0);
    if (baseToken != address(0)){
        IERC20(otherToken).safeApprove(router, 0);
        IERC20(baseToken).safeApprove(router, 0);
    }
}
```

DESCRIPTION	The contract owner of the personal vault can call panic. This causes the vault to withdraw tokens from the underlying farm and delete the amount the user manually deposited. This effectively will delete the record of how long they were staked. These tokens can be recovered using *saveToken*(); however, this is unconventional. The

	contract owner may also pause the vaults, potentially breaking the vault's interactions with the underlying farm.
RECOMMENDATION	Restrict the contract owner's ability to affect the operation of the vault.
MITIGATED/COMMENT	Ownership of personal vaults was transferred to the timelock. A new operator address was added to allow the project team to panic vaults in case of an emergency. Timelock 0xa5D65C1215dBa10EdB16B7b40B2a8981ef7f783C

Vaults Can Be Created By Anyone

SEVERITY	Low Risk
RESOLVED	**Mitigated**
FINDING ID	#0003
LOCATION	PrivateVaultFactory_LPReferralHarvestLock.sol -> 34



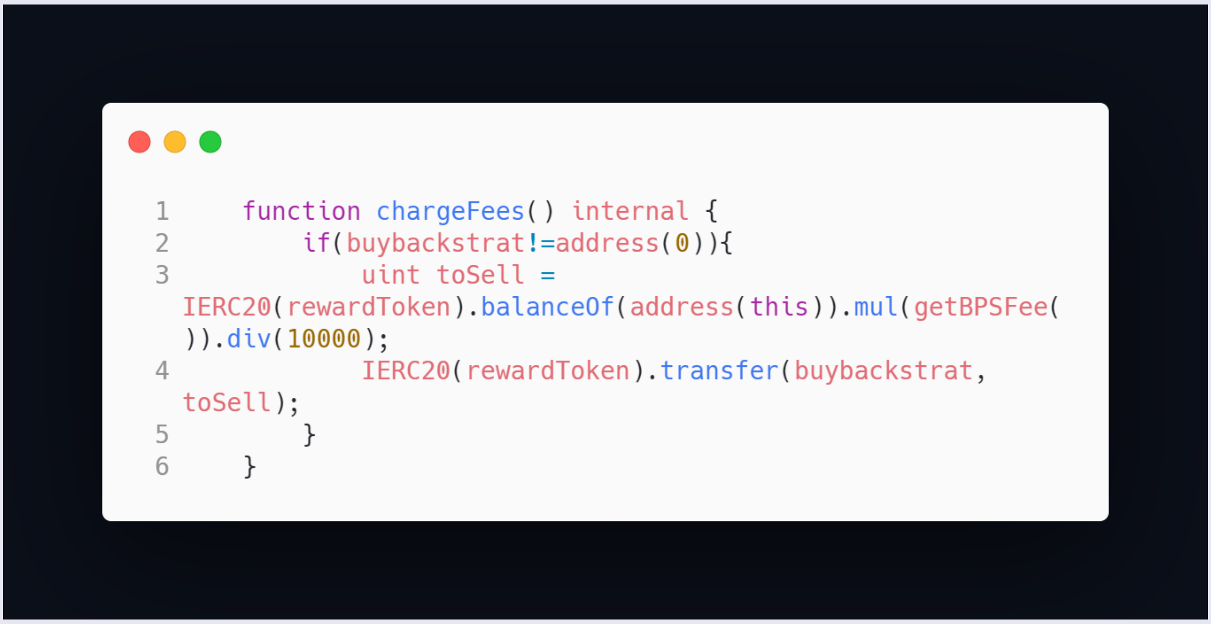
```
    function createVault(address _lpPair,
        address _rewardToken,
        address _baseToken,
        address _masterchef,
        uint256 _poolId, address _router, address _swapPathRegistry,
        address _user, address _feeStrat, bool _stakingMode, bool _referralMode) external returns (address) {

        StrategyLPPersonalVault temp = new StrategyLPPersonalVault(_lpPair,
            _rewardToken,
            _baseToken,
            _masterchef,
            _poolId,
            _router,
            _swapPathRegistry,
            _user,
            _feeStrat
        );
        temp.setStakingMode(_stakingMode);
        temp.setReferralMode(_referralMode);

        temp.transferOwnership(owner());
        IVaultRegistry(vaultRegistry).registerVault(_user, address(temp), VAULT_TYPE);
        return address(temp);
    }
```

DESCRIPTION	Vaults may be created by any caller for any given user. This could result in unexpected behavior, depending on how the vaultRegistry functions.
RECOMMENDATION	Restrict the addresses which can create a vault for a user. One option could be to restrict the msg.sender to *_user* or

	owner().
MITIGATED/COMMENT	Vaults can now only be created by the user who will use it.

Buyback Fees Can Be Unlimited

SEVERITY	Low Risk
RESOLVED	**Mitigated**
FINDING ID	#0004
LOCATION	StrategyLP2.sol -> 269-274 StrategyLPPersonalVault.sol -> 344-349

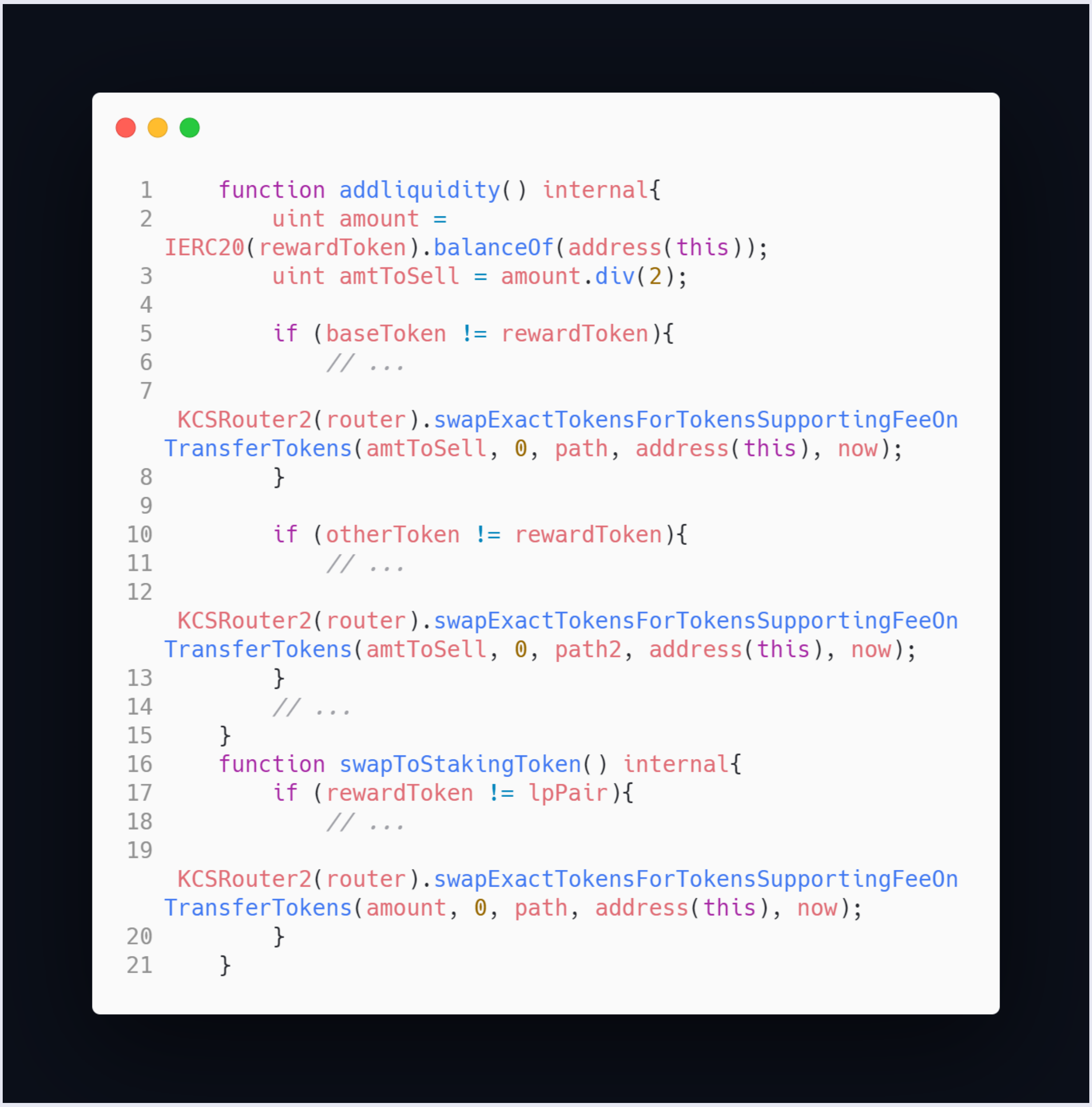


```
    function chargeFees() internal {
        if(buybackstrat!=address(0)){
            uint toSell = IERC20(rewardToken).balanceOf(address(this)).mul(getBPSFee()).div(10000);
            IERC20(rewardToken).transfer(buybackstrat, toSell);
        }
    }
```

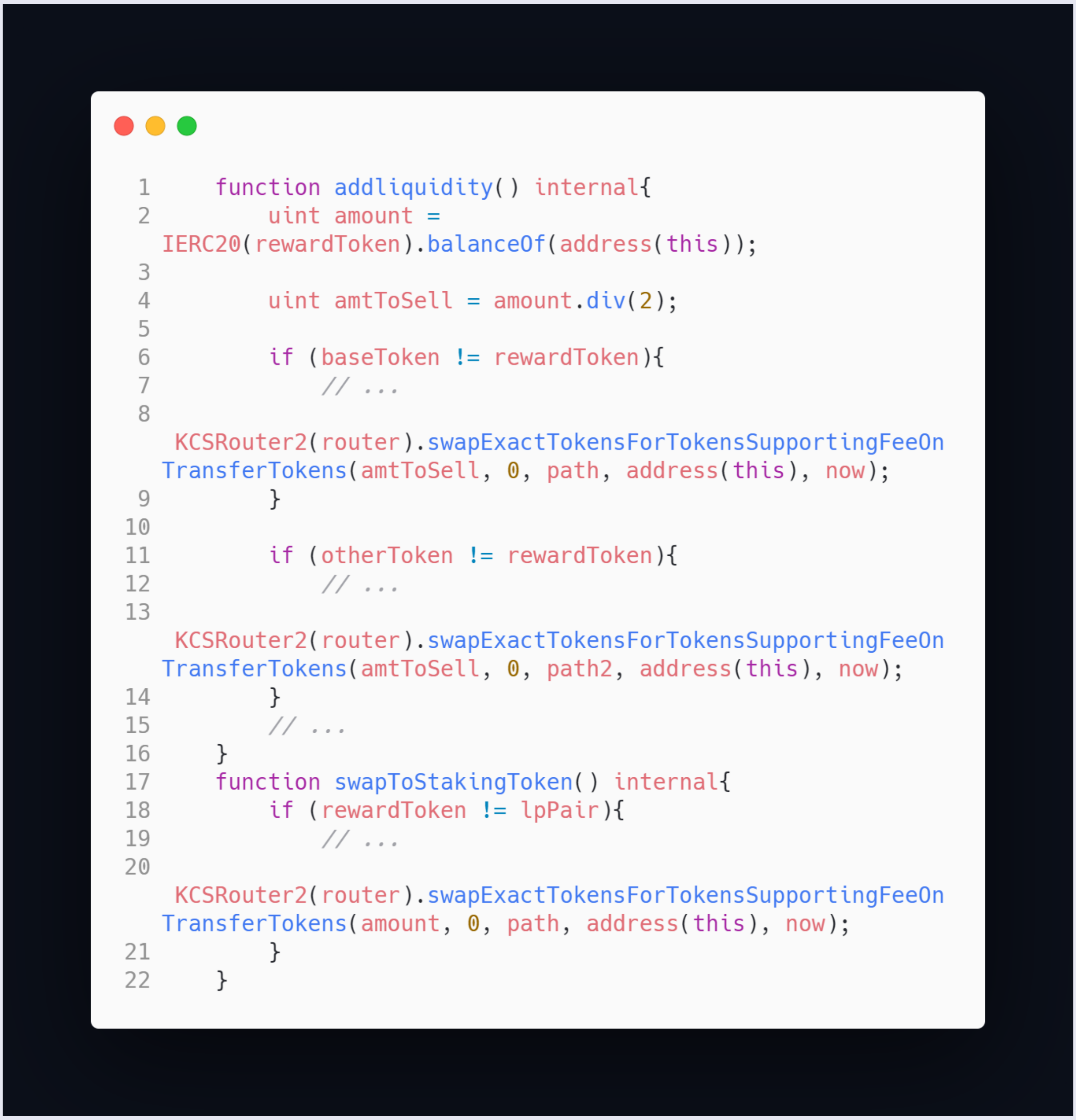
DESCRIPTION	The buyback strat determines the fee for the strategy. This may allow for large fee collection, or non-transparent fee collection to occur.
RECOMMENDATION	Set a limit on the fee share from within the strategy contract.
MITIGATED/COMMENT	A hard limit of 30% was added to any fees for the buyback strategy.

Swap To Compound Vault Can Be Frontrun

SEVERITY	Low Risk
RESOLVED	**See Comment**
FINDING ID	#0005
LOCATION	StrategyLPPersonalVault.sol -> 261-286 StrategyLP2.sol -> 261-286



```
    function addliquidity() internal{
        uint amount = IERC20(rewardToken).balanceOf(address(this));
        uint amtToSell = amount.div(2);

        if (baseToken != rewardToken){
            // ...
            KCSRouter2(router).swapExactTokensForTokensSupportingFeeOnTransferTokens(amtToSell, 0, path, address(this), now);
        }

        if (otherToken != rewardToken){
            // ...
            KCSRouter2(router).swapExactTokensForTokensSupportingFeeOnTransferTokens(amtToSell, 0, path2, address(this), now);
        }
        // ...
    }
    function swapToStakingToken() internal{
        if (rewardToken != lpPair){
            // ...
            KCSRouter2(router).swapExactTokensForTokensSupportingFeeOnTransferTokens(amount, 0, path, address(this), now);
        }
    }
```

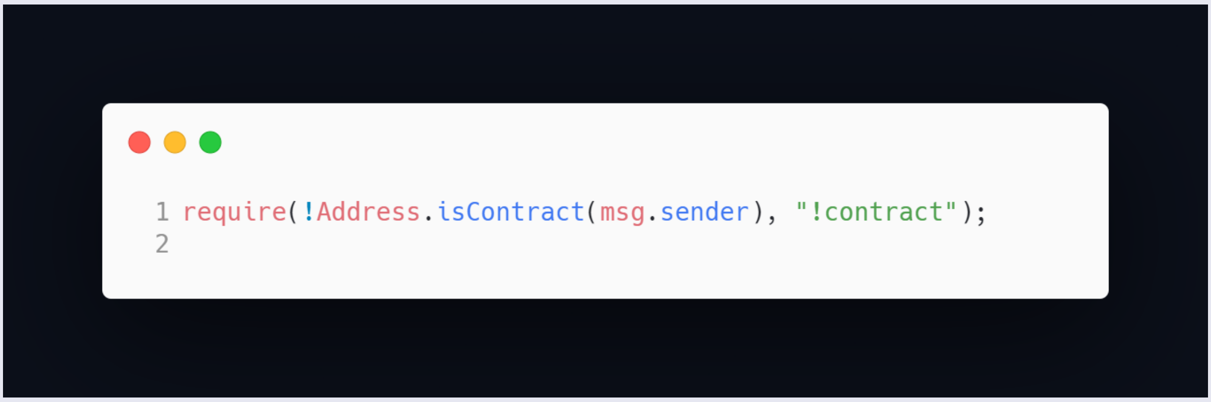


```
function addliquidity() internal{
    uint amount = IERC20(rewardToken).balanceOf(address(this));

    uint amtToSell = amount.div(2);

    if (baseToken != rewardToken){
        // ...
        KCSRouter2(router).swapExactTokensForTokensSupportingFeeOnTransferTokens(amtToSell, 0, path, address(this), now);
    }

    if (otherToken != rewardToken){
        // ...
        KCSRouter2(router).swapExactTokensForTokensSupportingFeeOnTransferTokens(amtToSell, 0, path2, address(this), now);
    }
    // ...
}
function swapToStakingToken() internal{
    if (rewardToken != lpPair){
        // ...
        KCSRouter2(router).swapExactTokensForTokensSupportingFeeOnTransferTokens(amount, 0, path, address(this), now);
    }
}
```

DESCRIPTION	The *StrategyLP2* and *StrategyLPPersonalVault* contracts trade for the staked token on the router but do not provide a slippage limit. The rewards from the swap can be front-run.
RECOMMENDATION	Provide a slippage limit for the token as a parameter or use an oracle's twap in order to prevent frontrunning.
MITIGATED/COMMENT	Project team comment: "will consider implementing a price oracle in future. for now, we will keep with frequent compounds to limit impact if exploited." Obelisk comment: "Frequent compounds will limit the risk. However, there is no way to guarantee that fees will be compounded at a given frequency."

EOA Only Functions May Still Be Called By A Contract

SEVERITY	Low Risk
RESOLVED	**Mitigated**
FINDING ID	#0006
LOCATION	StrategyLP2.sol -> 234 StrategyLPPersonalVault.sol -> 297



```
require(!Address.isContract(msg.sender), "!contract");

```

DESCRIPTION	The OpenZeppelin Address utility restricts harvesting to externally owned accounts (EOAs). However, Address will not detect contracts under certain conditions. Refer to: https://docs.openzeppelin.com/contracts/2.x/api/utils#Address-isContract-address-
RECOMMENDATION	Use tx.origin with the understanding that it may be non-functional or deprecated in the future, or modify this function such that contract calls are supported.
MITIGATED/COMMENT	A check was added to check tx.origin, and a toggle to switch to OpenZeppelin Address if necessary.

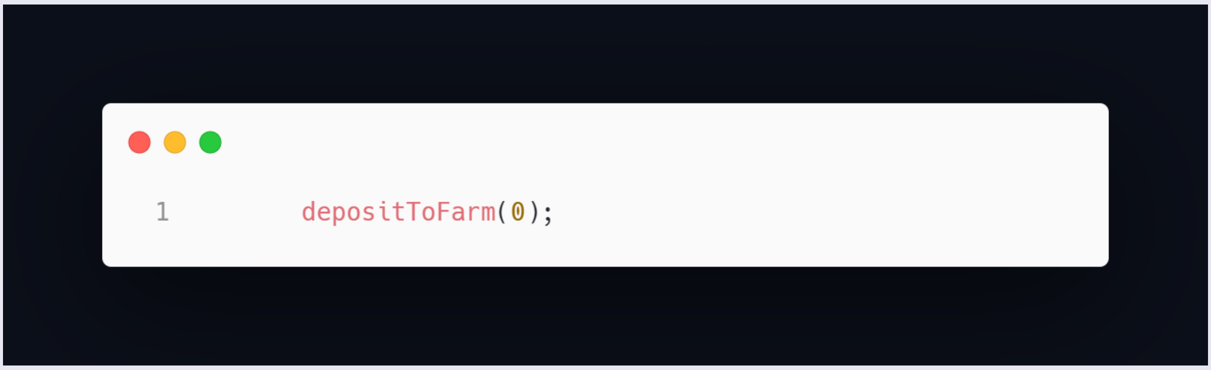
Local Copy of OpenZeppelin Contract

SEVERITY	Informational
RESOLVED	**Mitigated**
FINDING ID	#0007
LOCATION	Pausable.sol

DESCRIPTION	The contract includes local copies of OpenZeppelin contracts. The contract is identical to Openzeppelin 3.1.0.
RECOMMENDATION	Import OpenZeppelin instead of using local copies.
MITIGATED/COMMENT	OpenZeppelin contract was directly imported.

Strategy Harvests Rewards Via Deposit

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0008
LOCATION	StrategyLP2.sol -> 244 StrategyLPPersonalVault.sol -> 322



```
        depositToFarm(0);
```

DESCRIPTION	Harvesting of rewards is done by depositing 0 to the underlying farming contract. This may not work if the underlying contract does not automatically harvest on deposit or if it prevents deposits of 0 amount.
RECOMMENDATION	Ensure that deployed strategies use underlying farms which harvest rewards upon deposit.
MITIGATED/COMMENT	N/A

Static Analysis

Missing Zero Checks

SEVERITY	Informational
RESOLVED	**Partial**
FINDING ID	#0009
LOCATION	GrowthVaultLaunch.sol -> 62-64 : function setSnapshotter(address _a) external onlyOwner GrowthVaultLaunch.sol -> 67-71 : function setStrategy(address _strategy) external onlyOwner GrowthVaultLaunch.sol -> 230-233 : setBuybackStrat(address _address) public onlyOwner StrategyLP2.sol -> 88-90 : function setBuybackStrat(address _address) external onlyOwner StrategyLP2.sol -> 97-99 : function setSwapPathRegistry(address _a) external onlyOwner StrategyLPPersonalVault.sol -> 94-96 : function setBuybackStrat(address _address) external onlyOwner StrategyLPPersonalVault.sol -> 103-105 : function setSwapPathRegistry(address _a) external onlyOwner

DESCRIPTION	The contract address values can be set to zero address in various constructors, initializers, and setter functions. Zero addresses may cause incorrect contract behavior.
RECOMMENDATION	Add a check to ensure contract values are never set to an invalid zero address.
MITIGATED/COMMENT	Still unresolved: ● GrowthVaultLaunch.sol -> 62-64 : function setSnapshotter(address _a) external onlyOwner ● GrowthVaultLaunch.sol -> 230-233 : setBuybackStrat(address _address) public onlyOwner ● StrategyLP2.sol -> 88-90 : function setBuybackStrat(address _address) external onlyOwner ● StrategyLPPersonalVault.sol -> 94-96 : function setBuybackStrat(address _address) external onlyOwner

No Events Emitted For Changes To Protocol Values

SEVERITY	Informational
RESOLVED	**Mitigated**
FINDING ID	#0010
LOCATION	GrowthVaultLaunch.sol -> 58-60 : function setHarvestBeforeDeposit(bool _b) external onlyOwner GrowthVaultLaunch.sol -> 62-64 : function setSnapshotter(address _a) external onlyOwner GrowthVaultLaunch.sol -> 67-71 : function setStrategy(address _strategy) external onlyOwner GrowthVaultLaunch.sol -> 73-74 : function changeApprovalDelay(uint _time) public onlyOwner GrowthVaultLaunch.sol -> 211-228 : function upgradeStrat() public onlyOwner GrowthVaultLaunch.sol -> 230-233 : setBuybackStrat(address _address) public onlyOwner PrivateVaultFactory_LPReferralHarvestLock.sol -> 16-18 : function setVaultRegistry(address _r) external onlyOwner StrategyLP2.sol -> 88-90 : function setBuybackStrat(address _address) external onlyOwner StrategyLP2.sol -> 91-93 : function setStakingMode(bool _b) external onlyOwner StrategyLP2.sol -> 94-96 : function setReferralMode(bool _b) external onlyOwner StrategyLP2.sol -> 97-99 : function setSwapPathRegistry(address _a) external onlyOwner StrategyLP2.sol -> 100-102 : function setMinToLiquify(uint256 n) external onlyOwner StrategyLPPersonalVault.sol -> 94-96 : function setBuybackStrat(address _address) external onlyOwner StrategyLPPersonalVault.sol -> 97-99 : function setStakingMode(bool _b) external onlyOwner StrategyLPPersonalVault.sol -> 100-102 : function setReferralMode(bool _b) external onlyOwner StrategyLPPersonalVault.sol -> 103-105 : function setSwapPathRegistry(address _a) external onlyOwner StrategyLPPersonalVault.sol -> 106-108 : function setMinToLiquify(uint256 n) external onlyOwner StrategyLPPersonalVault.sol -> 166-168 : function setExitMode(bool _exit) public onlyUser

DESCRIPTION	Functions that change important variables should emit events such that users can more easily monitor the change.

RECOMMENDATION	Emit events from these functions.
MITIGATED/COMMENT	Events were added.

On-Chain Analysis

No Findings

Appendix A - Reviewed Documents

Document	**Address**
Buybackstrat.sol	N/A
GrowthVaultLaunch.sol	0x79aC1133C9CF22598744C95F8611142Ff23C884a
IPanwexPair.sol	N/A
IStrategy.sol	N/A
ISwapPathRegistry.sol	N/A
KCSRouter2.sol	N/A
Pausable.sol	N/A
PrivateVaultFactory_LPReferralHarvestLock.sol	0x1E6B84530E4c3cF67e79fc0F4DC87793F7174135
StrategyLP2.sol	0x780a2f48F7566b3306737432ed4A271F6925BDCF
StrategyLPPersonalVault.sol	N/A

rev-1: Commit b9e5cd956e98d0648d0d8dbff612a132df1f5cf6
rev-2: Commit a5983375c66fed2b6f784df290ed7e08bbe57de9
rev-3: Commit ba7804aa01de0a4e75de18d42b5c148196df41de
rev-4: Commit 2534e6ad85c4687dd0e361c1bb3457a8cc1da1bf
rev-5: Commit 88bc138103eb75daab30cd9d855542d7ef20f60c

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.

Follow Obelisk Auditing for the Latest Information

ObeliskOrg ObeliskOrg

OBELISK

Part of Tibereum Group