



Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 22	2021-05-20	Plemonade, Donut, Zapmore,	Final Audit
1.1	Total: 25	2022-01-12	Plemonade, Donut, Zapmore	Added new finding "Vault Shares Is Calculated As Deposited Tokens"

Audit Notes

Audit Date	2021-05-07 - 2021-05-19
Auditor/Auditors	Plemonade, Donut, MrTeaThyme
Auditor/Auditors Contact Information	tibereum-obelisk@protonmail.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB58157682

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 was 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 phased 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

Version Notes	2
Audit Notes	2
Disclaimer	2
Obelisk Auditing	3
Project Information	5
Executive Summary	6
Summary Table	7
Introduction	8
Findings	9
Manual Analysis	9
VaultChef can Deposit and Withdraw on Behalf of Users	9
Frontrunning on Vault Swaps	9
Frontrunning on Reward Distribution	12
StrategyVaultBurn Earn Behaviour Dependent on Timing	13
Router Address Can Be Changed	14
Vault Shares Are Calculated As Deposited Tokens	16
Static Analysis	19
No Findings	19
On-Chain Analysis	20
Vault Contracts not Verified	20
No Timelock Contract	20
Appendix A - Reviewed Documents	21
Appendix B - Risk Ratings	23
Appendix C - Icons	23
Appendix D - Testing Standard	24

Project Information

Project Name	Polycat
Description	Polycat is a decentralized yield farm with a referral system running on Polygon.
Website	https://polycat.finance
Contact	@FinnTheAdventurer
Contact information	@FinnTheAdventurer 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 Polycats Yield Optimizer Vaults was conducted by three of Obelisks' security experts between the 7th of May 2021 and the 19th of May 2021. The contracts were audited and then compared to their deployed counterparts.

After finishing the full audit, Obelisk auditing can say that there were some security issues during the initial audit of the audited contracts from Polycats Vaults. Polycat mitigated most issues and commented on others creating in order to create a safer project.

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

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

Summary Table

Audited Part	Severity	Note
VaultChef can Deposit and Withdraw on Behalf of Users	Medium Risk	Mitigated
Frontrunning on Vault Swaps	Low Risk	Mitigated
Frontrunning on Reward Distribution	Informational	Mitigated
StrategyVaultBurn Earn Behaviour Dependent on Timing	Informational	Mitigated
Router Address Can Be Changed	Low Risk	See Comment
Vault Contracts not Verified	Informational	Mitigated
No Timelock Contract	Medium Risk	See Comment
Vault Shares Is Calculated As Deposited Tokens	High Risk	Open

Introduction

Obelisk was commissioned by Polycat on the 7th of May 2021 to conduct a comprehensive audit of Polycats new Yield Optimizing Vaults. The following audit was conducted between the 7th of May 2021 and the 19th of May 2021 and delivered on the 20th of May 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 initial audit found an issue in the VaultChef contract which was mitigated by updating and re-deploying the contracts. The on-chain analysis determined that the initial issue was fixed. However, during the on-chain analysis, it was found that the quick swap vaults were changed from the audited contracts, introducing a new vulnerability. After the project team has fixed vulnerabilities, there is only a missing timelock to consider. A timelock is an important aspect for users to have time to react to new information.

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

Findings

Manual Analysis

VaultChef can Deposit and Withdraw on Behalf of Users

SEVERITY	Mitigated (Medium)
LOCATION	<i>VaultChef.sol -> 79-87</i>

```
// For unique contract calls
function deposit(address _sender, uint256 _pid, uint256 _wantAmt) public nonReentrant onlyOperator
{
    _deposit(_sender, _pid, _wantAmt);
}

// For unique contract calls
function withdraw(address _sender, uint256 _pid, uint256 _wantAmt) public nonReentrant onlyOperator
{
    _withdraw(_sender, _pid, _wantAmt);
}
```

DESCRIPTION

The VaultChef allows an operator to withdraw and deposit tokens on behalf of users.

A malicious actor in control of the owner's address will be able to drain all of a user's approved funds. This can be done by creating a malicious strategy that allows the withdrawal of deposited funds, adding it as a new pool, then calling the operator only deposit function.

Additionally, a malicious actor as an operator (but not the contract owner) can remove a user's approved tokens over several transactions using the fees of a legitimate strategy. This can be done by repeatedly depositing and withdrawing all of a user's tokens.

RECOMMENDATION

Remove the operator-only variants of the *deposit* and *withdrawal* functions. Additionally, provide a time lock so that users can be notified of any new pools which may be added.

MITIGATED/COMMENT

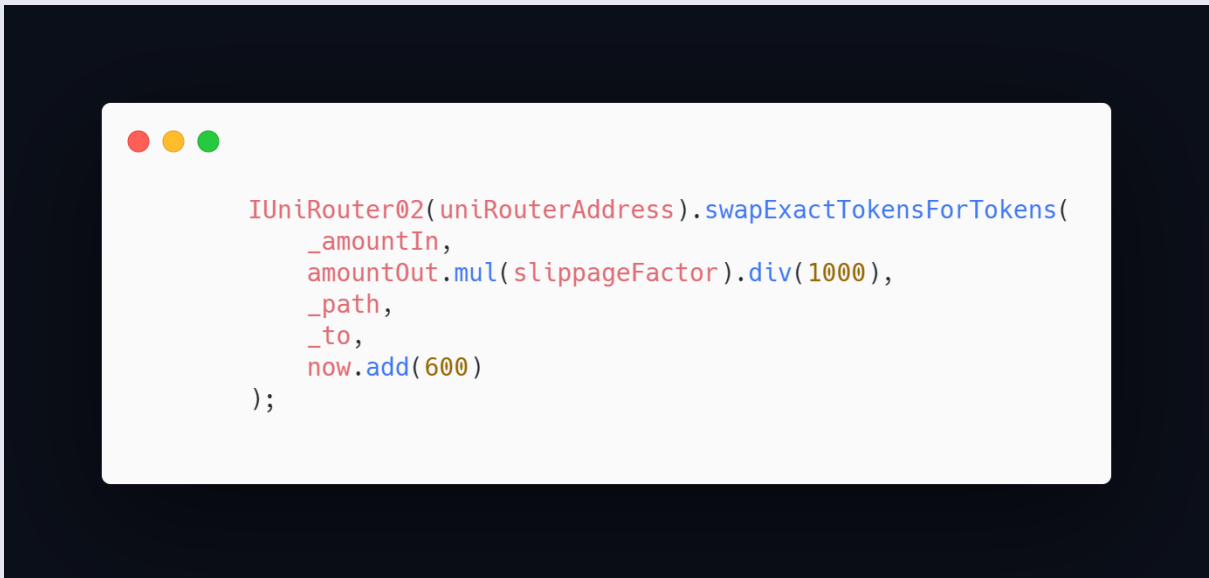
The Polycat team is deploying an updated [VaultChef contract](#). This contract was reviewed and the above issue is resolved.

Frontrunning on Vault Swaps

SEVERITY	Mitigated (Low)
LOCATION	<i>StrategyQuickSwap.sol -> 416-422</i> <i>StrategyQuickSwap.sol -> 433-439</i> <i>StrategyVaultBurn.sol -> 236-242</i>

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

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



DESCRIPTION	<p>The strategies use calls to a Uniswap type router in order to swap tokens for fees, buyback, and compounding of rewards. Calls to <i>earn</i> functions can be front-run as they effectively have unlimited slippage. By trading large amounts of the swapped tokens immediately before and after the <i>earn</i> function is called, a malicious actor can reduce the swap rate and reduce the anticipated rewards from a strategy.</p> <p>The strategies attempt to mitigate this using internal checks for slippage; however, these checks will be run within the same block as the swap itself and will therefore always pass.</p>
RECOMMENDATION	<p>Calculate the slippage outside the function call and pass it as a parameter. Alternatively, use the time-weighted average price in order to calculate slippage without being affected by short-term price manipulation. The time-weighted average price should be gathered for several blocks beforehand or use a separate oracle contract.</p>
MITIGATED/COMMENT	<p>Project team comment: "We don't actually care about slippage that much tbh. The earn is called so often it doesn't matter (every minute)"</p> <p>Obelisk comment: "As long as swaps remain relatively small this should be ok."</p>

Frontrunning on Reward Distribution

SEVERITY	Mitigated (Informational)
----------	---------------------------

LOCATION	<i>StrategyFish.sol</i> -> 60-83
----------	----------------------------------

```
function deposit(address _userAddress, uint256 _wantAmt) external onlyOwner nonReentrant
whenNotPaused returns (uint256) {
    UserInfo storage user = userInfo[_userAddress];

    uint256 pending = user.shares.mul(accUsdPerShare).div(1e18).sub(user.rewardDebt);
    if (pending > 0) {
        if (pending > 0) {
            IERC20(usdcAddress).safeTransfer(_userAddress, pending);
        }
    }

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

    sharesTotal = sharesTotal.add(_wantAmt);
    wantLockedTotal = sharesTotal;
    user.shares = user.shares.add(_wantAmt);

    user.rewardDebt = user.shares.mul(accUsdPerShare).div(1e18);

    return _wantAmt;
}
```

DESCRIPTION

The strategy for the FISH does not use withdrawal or deposit fees. A malicious actor could front calls to *depositReward* to purchase FISH and immediately deposit it, then withdraw and sell the received tokens in the next block. This allows the actor to receive the entire harvest without staking in the long term.

RECOMMENDATION

Add a deposit fee/withdrawal fee or a minimum time to withdraw.

MITIGATED/COMMENT

Project team comment: "We don't actually care about slippage that much tbh. The earn is called so often it doesn't matter (every minute)"

Obelisk comment: "As long as swaps remain relatively small this should be ok."

StrategyVaultBurn Earn Behaviour Dependent on Timing

SEVERITY	Mitigated (Informational)
LOCATION	<i>StrategyVaultBurn.sol</i> -> 127-133

```
function earn() external nonReentrant whenNotPaused onlyGov {  
    if (block.timestamp > lastEarnBlock.add(burnCycle)) {  
        burn();  
    } else {  
        lair();  
    }  
}
```

DESCRIPTION	The behavior of the <i>earn</i> function for <i>StrategyVaultBurn</i> is dependent on the timing between calls. If it is called less than <i>burnCycle</i> from the last time it was called, it will only ever call <i>burn</i> . This will result in rewards never being collected.
RECOMMENDATION	Get rewards during the <i>burn</i> function as well as the <i>lair</i> function.
MITIGATED/COMMENT	Project team comment: "Same here with the issue as we call <i>earn()</i> every minute"

Router Address Can Be Changed

SEVERITY	Low
LOCATION	<i>StrategySushiSwap.sol</i> -> 27 <i>StrategyQuickSwap.sol</i> -> 25 (Note: code is identical in both files)

```
address public uniRouterAddress = 0xa5E0829CaEd8fFDD4De3c43696c57F7D7A678ff;
```

LOCATION	<i>StrategySushiSwap.sol</i> -> 438-465 <i>StrategyQuickSwap.sol</i> -> 379-406 (Note: code is identical in both files)
----------	---

```
1 function setSettings(  
2     uint256 _controllerFee,  
3     uint256 _rewardRate,  
4     uint256 _buyBackRate,  
5     uint256 _withdrawFeeFactor,  
6     uint256 _slippageFactor,  
7     address _uniRouterAddress  
8 ) external onlyGov {  
9     require(_controllerFee.add(_rewardRate).add(_buyBackRate) <= feeMaxTotal, "Max fee of 10%");  
10    require(_withdrawFeeFactor >= withdrawFeeFactorLL, "_withdrawFeeFactor too low");  
11    require(_withdrawFeeFactor <= withdrawFeeFactorMax, "_withdrawFeeFactor too high");  
12    require(_slippageFactor <= slippageFactorUL, "_slippageFactor too high");  
13    controllerFee = _controllerFee;  
14    rewardRate = _rewardRate;  
15    buyBackRate = _buyBackRate;  
16    withdrawFeeFactor = _withdrawFeeFactor;  
17    slippageFactor = _slippageFactor;  
18    uniRouterAddress = _uniRouterAddress;  
19  
20    emit SetSettings(  
21        _controllerFee,  
22        _rewardRate,  
23        _buyBackRate,  
24        _withdrawFeeFactor,  
25        _slippageFactor,  
26        _uniRouterAddress  
27    );  
28 }  
29
```

DESCRIPTION	<p>The <i>uniRouterAddress</i> is modifiable through the <i>setSettings</i> function.</p> <p>A malicious actor in control of a quickswap or sushiswap vault can change the uniswap router address to a malicious router which drains any swapped tokens or added liquidity.</p> <p>Though this will not result in the loss of deposited funds, this vulnerability can be used to take any future rewards.</p>
RECOMMENDATION	<p>Return the <i>uniRouterAddress</i> to be a constant. Additionally, add a timelock to allow users to react to changes.</p>
MITIGATED/COMMENT	<p>Project team comment: "Added as we might require this change for a potential upcoming amm..."</p> <p>Obelisk comment: "Once timelock is added, this issue can be considered mitigated"</p>

Vault Shares Are Calculated As Deposited Tokens

SEVERITY

High

LOCATION

StrategyQuickSwap.sol -> 130-139

StrategyVaultBurn.sol -> 86-95

```
1  function _farm() internal returns (uint256) {  
2      uint256 wantAmt = IERC20(wantAddress).balanceOf(address(this));  
3      if (wantAmt == 0) return 0;  
4  
5      uint256 sharesBefore = vaultSharesTotal();  
6      IStakingRewards(quickSwapAddress).stake(wantAmt);  
7      uint256 sharesAfter = vaultSharesTotal();  
8  
9      return sharesAfter.sub(sharesBefore);  
10 }
```

LOCATION

StrategyQuickSwap.sol -> 324-326

StrategyVaultBurn.sol -> 173-175

```
1  function vaultSharesTotal() public view returns (uint256) {  
2      return IStakingRewards(quickSwapAddress).balanceOf(address(this));  
3  }
```


LOCATION

StrategySushiSwap.sol -> 145-154

```
1 function _farm() internal returns (uint256) {
2     uint256 wantAmt = IERC20(wantAddress).balanceOf(address(this));
3     if (wantAmt == 0) return 0;
4
5     uint256 sharesBefore = vaultSharesTotal();
6     ISushiStake(sushiYieldAddress).deposit(pid, wantAmt, address(this));
7     uint256 sharesAfter = vaultSharesTotal();
8
9     return sharesAfter.sub(sharesBefore);
10 }
```

LOCATION

StrategySushiSwap.sol -> 371-373

```
1 function vaultSharesTotal() public view returns (uint256) {
2     (uint256 balance,) = ISushiStake(sushiYieldAddress).userInfo(pid, address(this));
3     return balance;
4 }
```

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.

MITIGATED/COMMENT

N/A

Static Analysis

No Findings

On-Chain Analysis

Vault Contracts not Verified

SEVERITY	Mitigated (Informational)
DESCRIPTION	<p>Vault contracts are not verified on-chain.</p> <p>Note, 17 vaults were found on-chain:</p> <ul style="list-style-type: none">- Vault 0 matches StrategyFish.sol- Vault 1 matches StrategyVaultBurn.sol- Vaults 2 - 15 matches StrategyQuickswap.sol- Vault 16 is not audited. <p>Deployed contract bytecode is essentially identical to the source code, but should be verified nonetheless.</p>
RECOMMENDATION	Verify each vault contract.
MITIGATED/COMMENT	<p>Contracts were redeployed and verified.</p> <p>In the redeployed contracts, 28 vaults were found on-chain:</p> <ul style="list-style-type: none">- Vault 0 matches StrategyFish.sol- Vault 1 matches StrategyVaultBurn.sol- Vaults 2 - 15 match StrategyQuickSwap.sol- Vault 16 - 27 match StrategySushiSwap.sol

No Timelock Contract

SEVERITY	Medium
DESCRIPTION	<p>Vault settings should be set behind a timelock to allow users to react to changes in withdrawal fees and other fees. A timelock contract may exist, but could not be verified at the time of report as vault contracts were not verified on-chain.</p>
RECOMMENDATION	Provide a timelock contract.
MITIGATED/COMMENT	<p>Project team comment: "Will be added after we finish auditing and deploying sushi and aave vaults."</p> <p>Obelisk comment: "Once timelock is added, this issue can be considered mitigated"</p>

Appendix A - Reviewed Documents




Document	Address
Operators.sol	N/A
StrategyFish.sol	0x917fb15e8aaa12264dcbdc15afef7cd3ce76ba39
StrategySushiSwap.sol	0xd1f73333f5725fbfc47f682c7517cbba9a0231b6 0xb830f1c82f207920c2590594c004e56d7789b664 0x804db5b714cef63f20189599232a68372b43fccf 0x3ea0b00b5af4aec7b6a6c08df9a534a9904f283e 0x516b26b00700d96586bd3cbf08fa12f8e23e36fc 0xbafb2c8fd1a3d279fedd5f8c2680f87252498407 0xa81d797b8d618b3f2270d463a4f4e894a06117ca 0x803cf39b1c46ac6ad6607f5ebcf0ce18f94567f 0x164a64174f1bc4468a0711b6e5491adb27c27f00 0x79fe6e3b4324b1a269fe95aad3b4e2cc5130548e 0x4f1f7499e96908671f8061409476edfbbacbd743 0x52cf543e09bb87f0ccb5a36433c9ef0053bcd1ac
StrategyQuickSwap.sol	0x8d283cb9c75f7ef26c874b08a79e8545a958fe5b 0x0820c216d412b54167c3a9661e64fe4fb17581a2 0x4d9775066c0d9cede14ea304a5101c95c5b4cf00 0x12ff345cfad2b1dca5d385ccdbd0c2032fe3de70 0x232a9964b0d9bd8486d285d11aadbec158fa3c18 0x7ca775709e10e688a58f38b57b9a7a65138a5b05 0x00a817719d07722b07db09ebe3e9dab949c1dd60 0x3d65b6d07a1ae8cd067fe61ee05b917f36fabd66 0x6768d4784375c2e018418de5ffd7f4a7b43d899d 0xabfab1305022660a42ea120daf998f54ab78a84d 0xf94b36e7d98aed08e9474f919b94b2c843d8037b 0xc1874607d0545f7e9e24e289dfd1e1f678be260b 0x5a80f6fc727f7d0c060aeaaab1640bb92fcfb30b 0x62ef27818848cb1b50727fd6d65b1c60ac48fe58
StrategyVaultBurn.sol	0x897c0d19c4a64e45480acb79cea2f608d2d226bd
VaultChef.sol	0xBdA1f897E851c7EF22CD490D2Cf2DAce4645A904
IDragonLair.sol	N/A
IStakingRewards.sol	N/A
IStrategy.sol	N/A
IStrategyFish.sol	N/A
ISushiStake.sol	N/A

IUniPair.sol	N/A
IUniRouter01.sol	N/A
IUniRouter02.sol	N/A
IWETH.sol	N/A

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



Part of Tibereum Group