OBELISK

OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 21	2021-06-01	Plemonade, Donut, Zapmore	Audit Draft

Audit Notes

Audit Date	2021-05-28 - 2021-06-02
Auditor/Auditors	Plemonade, Donut, Hebilicious
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 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	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 AAVE and Iron vaults was conducted by three of Obelisks' security experts between the 28th of May 2021 and the 2nd of June 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 Polycat's AAVE and Iron Vaults. Polycat mitigated most issues and commented on others 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
Frontrunning On Vault Swaps	Low Risk	Mitigated
Router Address Can Be Changed	Low Risk	See Comment
Risk Of Liquidation With Infrequent Interactions	Low Risk	Mitigated
Risk Of Liquidation Due To Aave Configuration Changing	Medium Risk	See Comment
No Timelock Contract	Medium Risk	See Comment
Contracts Not Verified	Low Risk	N/A

Introduction

Obelisk was commissioned by Polycat on the 27th of May 2021 to conduct a comprehensive audit of Polycat’s new Yield Optimizing AAVE and Iron Vaults. The following audit was conducted between the 28th of May 2021 and the 2nd of June 2021 and delivered on the 4th 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.

After our auditors went through the provided contracts, we found some issues especially with Risk Of Liquidation. These issues could cause some problems to the user or the project if in any circumstance these scenarios are played out as the worst case. There are also some other issues that could be mitigated with an added timelock, which the project team has stated they will add as soon as all vaults are in place.

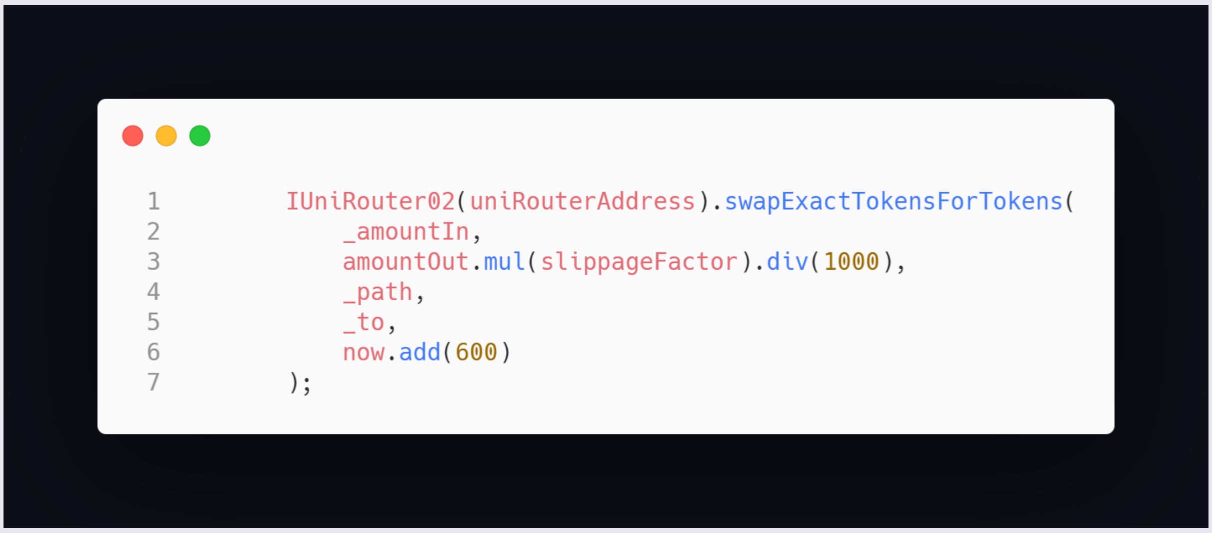
Please see each section of the audit to get a full understanding of the audit.

Findings

Manual Analysis

Frontrunning On Vault Swaps

SEVERITY	Mitigated (Low)
LOCATION	StrategyAave.sol -> 485-491 StrategyMasterchef.sol -> 426-432 StrategyMasterchef.sol -> 443-448



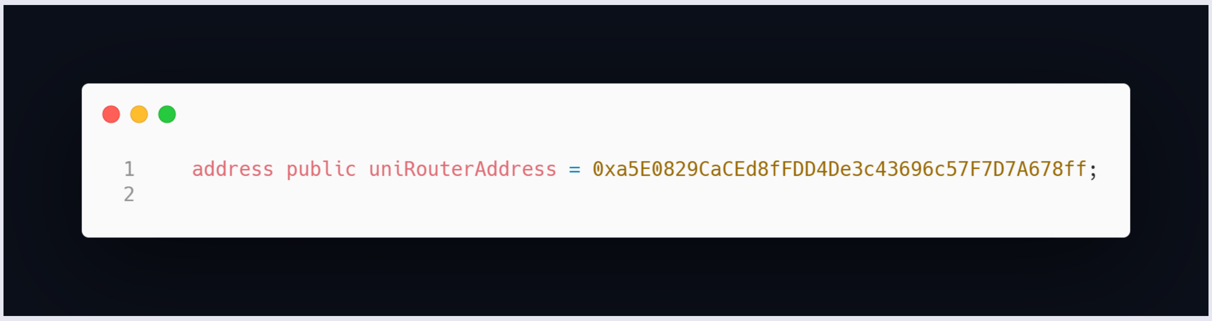
```
        IUniRouter02(uniRouterAddress).swapExactTokensForTokens(
            _amountIn,
            amountOut.mul(slippageFactor).div(1000),
            _path,
            _to,
            now.add(600)
        );
```

DESCRIPTION	The strategies use calls to a Uniswap type router in order to swap tokens for fees, buyback, and compounding of rewards. Calls to earn functions can be front-run as they effectively have unlimited slippage. By trading large amounts of the swapped tokens immediately before and after the earn function is called, a malicious actor can reduce the swap rate and reduce the anticipated rewards from a strategy. 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.
RECOMMENDATION	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

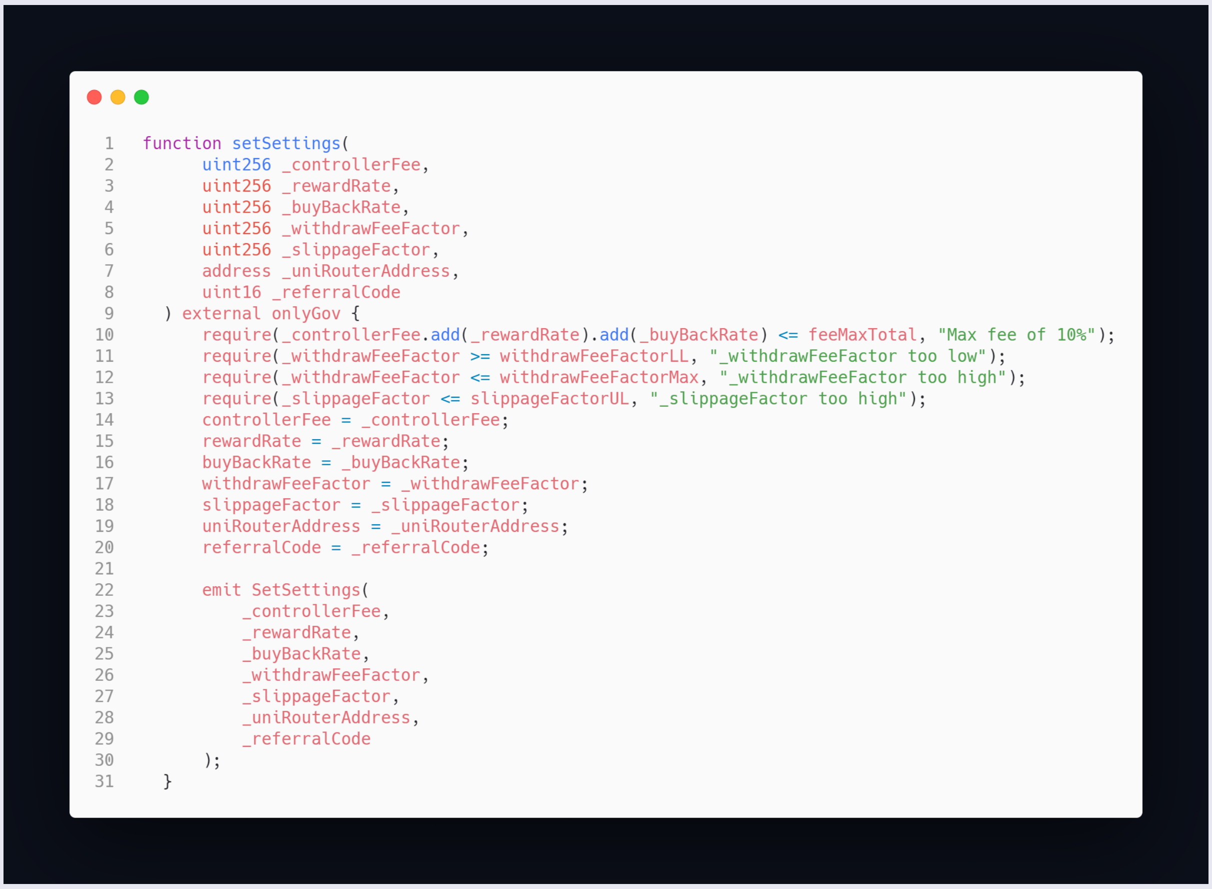
	contract.
MITIGATED/COMMENT	Project team comment: “We don't actually care about slippage that much tbh. The earn is called so often it doesn't matter (every minute)” Obelisk comment: “As long as swaps remain relatively small this should be ok.”

Router Address Can Be Changed

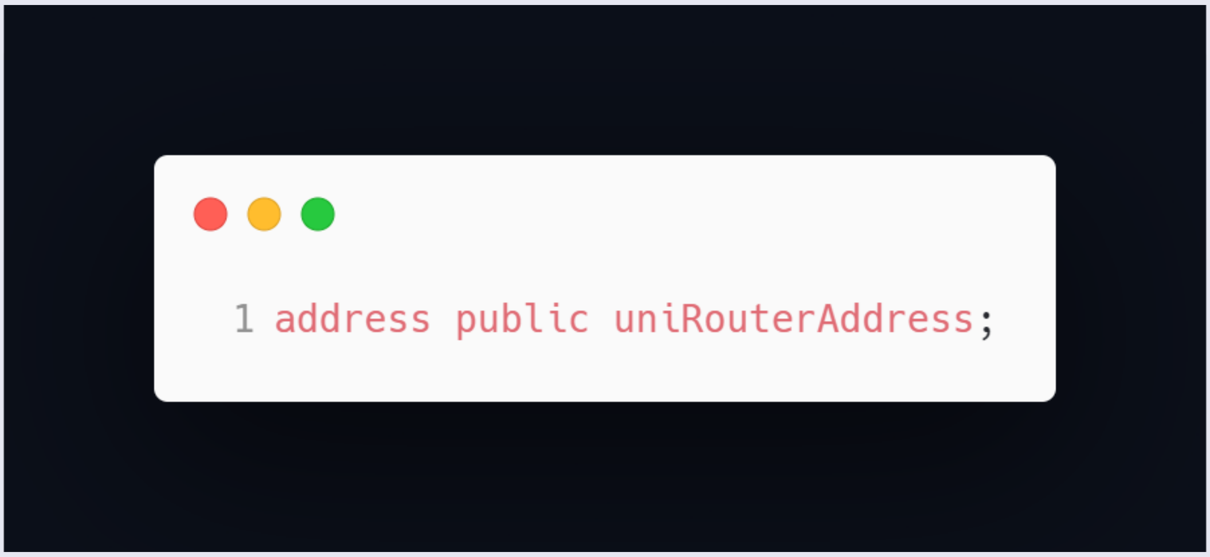
SEVERITY	Low
LOCATION	StrategyAave.sol -> 30 StrategyAave.sol -> 441-471 StrategyMasterchef.sol -> 27 StrategyMasterchef.sol -> 385-412



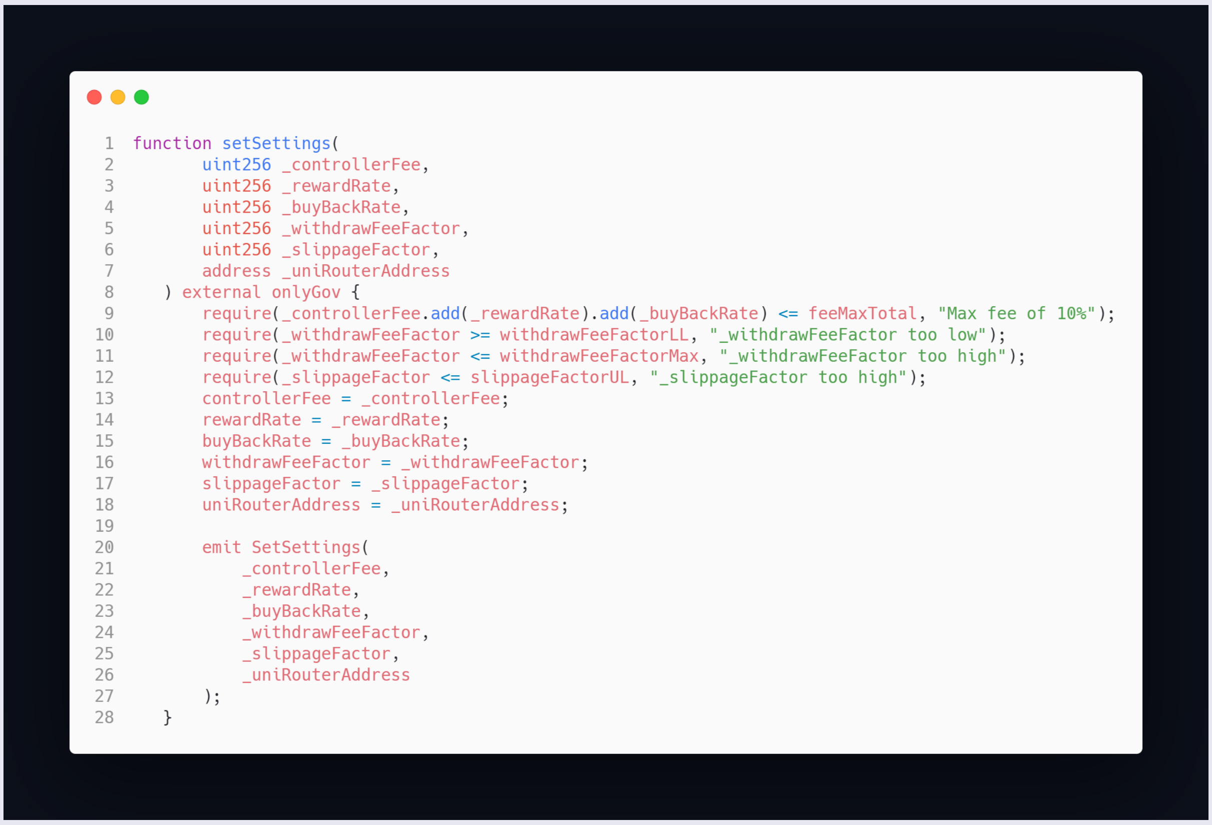
```
address public uniRouterAddress = 0xa5E0829CaCEd8fFDD4De3c43696c57F7D7A678ff;
```



```
function setSettings(
      uint256 _controllerFee,
      uint256 _rewardRate,
      uint256 _buyBackRate,
      uint256 _withdrawFeeFactor,
      uint256 _slippageFactor,
      address _uniRouterAddress,
      uint16 _referralCode
  ) external onlyGov {
      require(_controllerFee.add(_rewardRate).add(_buyBackRate) <= feeMaxTotal, "Max fee of 10%");
      require(_withdrawFeeFactor >= withdrawFeeFactorLL, "_withdrawFeeFactor too low");
      require(_withdrawFeeFactor <= withdrawFeeFactorMax, "_withdrawFeeFactor too high");
      require(_slippageFactor <= slippageFactorUL, "_slippageFactor too high");
      controllerFee = _controllerFee;
      rewardRate = _rewardRate;
      buyBackRate = _buyBackRate;
      withdrawFeeFactor = _withdrawFeeFactor;
      slippageFactor = _slippageFactor;
      uniRouterAddress = _uniRouterAddress;
      referralCode = _referralCode;

      emit SetSettings(
          _controllerFee,
          _rewardRate,
          _buyBackRate,
          _withdrawFeeFactor,
          _slippageFactor,
          _uniRouterAddress,
          _referralCode
      );
  }
```



```
address public uniRouterAddress;
```



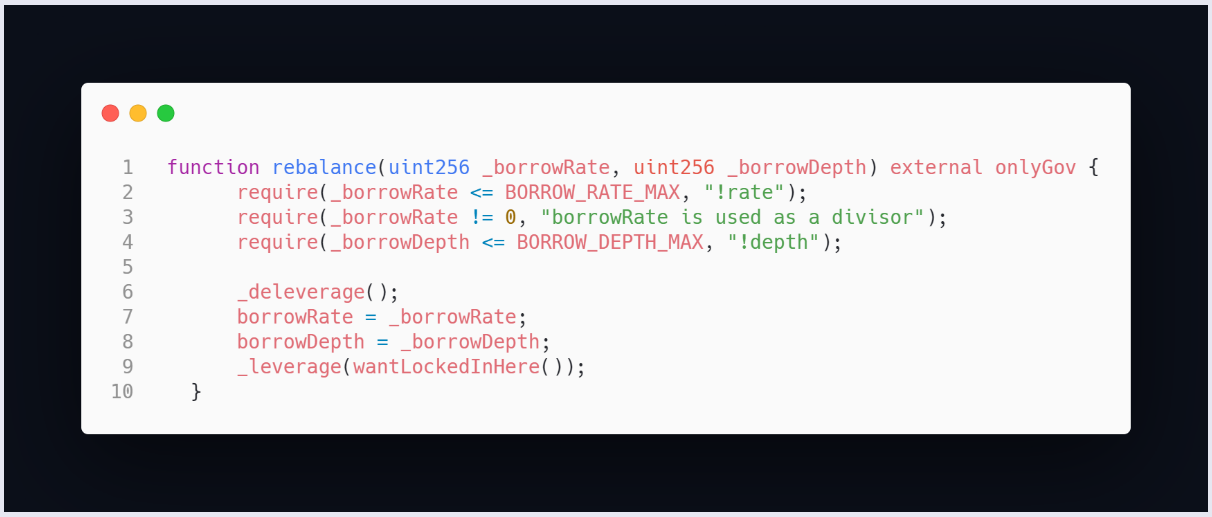
```
function setSettings(
        uint256 _controllerFee,
        uint256 _rewardRate,
        uint256 _buyBackRate,
        uint256 _withdrawFeeFactor,
        uint256 _slippageFactor,
        address _uniRouterAddress
    ) external onlyGov {
        require(_controllerFee.add(_rewardRate).add(_buyBackRate) <= feeMaxTotal, "Max fee of 10%");
        require(_withdrawFeeFactor >= withdrawFeeFactorLL, "_withdrawFeeFactor too low");
        require(_withdrawFeeFactor <= withdrawFeeFactorMax, "_withdrawFeeFactor too high");
        require(_slippageFactor <= slippageFactorUL, "_slippageFactor too high");
        controllerFee = _controllerFee;
        rewardRate = _rewardRate;
        buyBackRate = _buyBackRate;
        withdrawFeeFactor = _withdrawFeeFactor;
        slippageFactor = _slippageFactor;
        uniRouterAddress = _uniRouterAddress;

        emit SetSettings(
            _controllerFee,
            _rewardRate,
            _buyBackRate,
            _withdrawFeeFactor,
            _slippageFactor,
            _uniRouterAddress
        );
    }
```

DESCRIPTION	The uniRouterAddress is modifiable through the setSettings function. 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. Though this will not result in the loss of deposited funds, this vulnerability can be used to take any future rewards.
RECOMMENDATION	Return the uniRouterAddress to be a constant. Additionally,

	add a timelock to allow users to react to changes.
MITIGATED/COMMENT	Project team comment: “Added as we might require this change for a potential upcoming amm…” Obelisk comment: “Once timelock is added, this issue can be considered mitigated”

Risk Of Liquidation With Infrequent Interactions

SEVERITY	Low
LOCATION	*StrategyAave.sol -> 430-439*

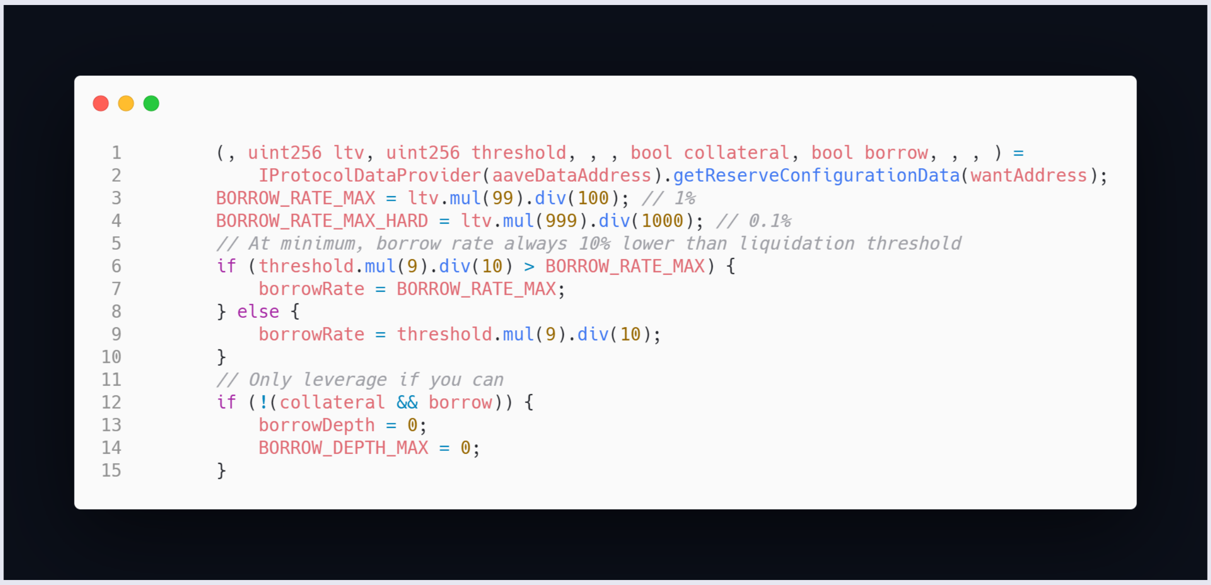


```
function rebalance(uint256 _borrowRate, uint256 _borrowDepth) external onlyGov {
      require(_borrowRate <= BORROW_RATE_MAX, "!rate");
      require(_borrowRate != 0, "borrowRate is used as a divisor");
      require(_borrowDepth <= BORROW_DEPTH_MAX, "!depth");

      _deleverage();
      borrowRate = _borrowRate;
      borrowDepth = _borrowDepth;
      _leverage(wantLockedInHere());
  }
```

DESCRIPTION	The rebalance function needs to be called at regular intervals in order to ensure that the borrowed amount remains below the liquidation threshold. Deleveraging and releveraging is performed during deposit and withdraw events. If no users interact with the contract for a sufficiently long time, the health of the loans will be at risk.
RECOMMENDATION	Provide a separate rebalancing function that does not update the borrowRate. Ensure that the vault is regularly re-balanced.
MITIGATED/COMMENT	Project team comment: "earn also called leverage(), which naturally will balance back to LTV 1.1. the risks are we don't call earn() for a week. This is extremely unlikely since we would also call our panic() function if this could potentially ever occur, which withdraws all funds from aave so no potential liquidation." Obelisk comment: “As long as earn is regularly called and the team monitors the health of the loans, then this issue can be considered mitigated.”

Risk Of Liquidation Due To Aave Configuration Changing

SEVERITY	Medium
LOCATION	*StrategyAave.sol -> 105-19*



```
(, uint256 ltv, uint256 threshold, , , bool collateral, bool borrow, , , ) =
    IProtocolDataProvider(aaveDataAddress).getReserveConfigurationData(wantAddress);
BORROW_RATE_MAX = ltv.mul(99).div(100); // 1%
BORROW_RATE_MAX_HARD = ltv.mul(999).div(1000); // 0.1%
// At minimum, borrow rate always 10% lower than liquidation threshold
if (threshold.mul(9).div(10) > BORROW_RATE_MAX) {
    borrowRate = BORROW_RATE_MAX;
} else {
    borrowRate = threshold.mul(9).div(10);
}
// Only leverage if you can
if (!(collateral && borrow)) {
    borrowDepth = 0;
    BORROW_DEPTH_MAX = 0;
}
```

DESCRIPTION	Aave configuration may change; however, they are only read during the initial setup of the vault. If the liquidation threshold changes, the vault may be liquidated with loss of deposited funds.
RECOMMENDATION	Move the reading of AAVE configuration to a separate function and regularly call to ensure the latest configuration values are available.
MITIGATED/COMMENT	Project team comment: “The code will be updated but not redeployed, we are always monitoring aave announcements for if issues occur” Obelisk comment: “Aave configuration changes are rare; however, the team should be prepare to re-deploy the contracts in the case of such an event”

Static Analysis

No Findings

On-Chain Analysis

No Timelock Contract

SEVERITY	Medium Risk
DESCRIPTION	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.
RECOMMENDATION	Provide a timelock contract.
MITIGATED/COMMENT	Project team comment: “Will be added after we finish auditing and deploying sushi and AAVE vaults.” Obelisk comment: “Once timelock is added, this issue can be considered mitigated”

Contracts Not Verified

SEVERITY	Low
DESCRIPTION	Not all contracts were not verified on the blockchain explorer. According to the Polycat team, the following vault ids should correspond to the reviewed contracts: **Contract** — **Vault Ids** StrategyAave.sol — 32, 40, 42, 43, 44, 45, 46 StrategyMasterchef.sol — 29 For StrategyAave, vault 32 was verified. Vaults 40, 42, 43, 44, 45, and 46 were not verified and their deployed bytecode differs substantially from the verified vault 32. Note that vaults 43, 44, 45, and 46 have identical bytecode. For StrategyMasterchef, 29 was verified. The Polycat team provided two versions of the AAVE vaults. The first version corresponded with vault 32. The second version could not be matched. See Appendix A.
RECOMMENDATION	Verify the remaining vault contracts.
MITIGATED/COMMENT	Note: The matic block explorer has persistent issues verifying contracts. The Polycat team has stated they will verify the deployed contracts as soon as possible.

Contract	Vault Ids
StrategyAave.sol	32, 40, 42, 43, 44, 45, 46
StrategyMasterchef.sol	29

Appendix A - Reviewed Documents

Document	**Address**
StrategyAave.sol - V1	0xd43d6fa7b5a435d38b2df5a84440b46bf3b8ddeb Note: Vault 32.
StrategyAave.sol - V2	0x638ff27406896f84477bbee4f04241a2e48e59d6 0x8a6d902c66cdda0e01545c019516ac59d7c86397 0xc5405ac6c36b3e1a1e7250519176b7613ada6d1f 0x678c356f019e305ecdc88c470042a0bfcb59db88 0x3d9d597f74f4a7d53e6b50f126a9f06b877fa086 0x8f4a9a607c40ce47aaf004bef237e3557275f83a Note: Vaults 40, 42, 43, 44, 45, 46.
StrategyMasterchef.sol	0x83e6250c35617869a1e91ede86702be21f1933e8 Note: Vault 29.
IAaveStake.sol	N/A
IProtocolDataProvider.sol	N/A
IStrategyFish.sol	N/A
IUniPair.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.

OBELISK

Part of Tibereum Group