OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 41	2022-04-04	Plemonade, Donut	Audit Final

Audit Notes

Audit Date	2022-01-28 - 2022-03-29
Auditor/Auditors	Plemonade, thing_theory
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	OB96865924

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	Waterfall DeFi
Description	Waterfall DeFi is a platform that offers true risk diversification through tranching of yield generating DeFi assets.
Website	https://app.waterfalldefi.org/
Contact	@Waterfalldefi on Twitter
Contact information	@nachi0x on TG
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	BSC

Audit of Waterfall

Obelisk was commissioned by Waterfall on the 24th of January 2022 to conduct a comprehensive audit of Waterfalls' new strategy contracts. The following audit was conducted between the 28th of January 2022 and the 29th of March 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.

Most code issues found were either closed or mitigated with only high-risk issue #1 being still fully open. Also, there are 2 low-risk issues, #3 and #11 that could be handled in safer ways. Other High- and Medium Risk issues found on-chain adhere mostly to the non-standard unaudited timelock that is implemented. Issue #3 and #16 also include a comment on a previous audit done by Slowmist, Obelisk hasn't audited the contract for issue #16, the issues remain open in this audit.

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.

The team has not reviewed the UI/UX, logic, team, or tokenomics of the Waterfall 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

Manual Analysis

Finding	ID	Severity	Status
Price Can Be Set Arbitrarily	#0001	High Risk	Open
Deposit Can Silently Fail	#0002	High Risk	Closed
Oracle Is Trusted Implicitly	#0003	Low Risk	Open
Warmup Will Break Contract	#0004	High Risk	Mitigated
Bond Price Decimals May Not Match Token Decimals	#0005	Medium Risk	Closed
Only Reward Type 0 Is Claimed	#0006	Low Risk	Closed
Gas Optimization	#0007	Informational	Closed
Unnecessary Override	#0008	Informational	Closed
Dangerous Modifier	#0009	Informational	Open
Protocol Values Should Be Public	#0010	Informational	Open

Static Analysis

Finding	ID	Severity	Status
Division Before Multiplication	#0011	Low Risk	Partially Closed
No Events Emitted For Changes To Protocol Values	#0012	Informational	Closed

On-Chain Analysis

Finding	ID	Severity	Status
Timelock Can Be Bypassed	#0013	High Risk	Open
Withdraw Function Is Not Initialized	#0014	High Risk	Mitigated
Oracle Can Be Changed	#0015	High Risk	Open
Contracts Owners Are Unaudited	#0016	High Risk	Open

Router and Trade Paths Can Be Changed	#0017	High Risk	Partially Closed
Timelock Minimum Delay Is Short	#0018	Medium Risk	Open

Findings

Manual Analysis

Price Can Be Set Arbitrarily

FINDING ID	#0001
SEVERITY	High Risk
STATUS	Open
LOCATION	strategy/StrategyWonderland.sol -> 127-134

```
1    function setIsMint(bool _isMint) public onlyGovernor {
2        isMint = _isMint;
3    }
4
5    function setTimePrice(uint256 _timePrice) public onlyGovernor {
6        // 6 digit number
7        timePrice = _timePrice;
8    }
```

DESCRIPTION	Arbitrarily setting the price is dangerous. A code comment also mentions it needs to be six digits, but this is never checked to be correct. The time price is then used for a token swap.
RECOMMENDATION	Do not Arbitrarily set *timePrice*. Using an old price to swap could result in a significant sandwich attack. It will depend on how the *StrategyWonderland* contract is used as the strategy's vault is not being audited. Consider taking user UI inputs in conjunction with a trusted oracle; this also will depend on how the vault interacts with this contract.
RESOLUTION	Project comment: "Since no well-established oracle provides pricing data for $TIME / $MAXI, we need to manually set the market price of the coin according to CoinGecko right before the start and the end of the cycle. We will update the smart contract when there is an appropriate pricing data feed from a well-established oracle." Obelisk comment: "Governor role is able to set the price thus users have to trust the multisig that has access to the

	governor role.”

Deposit Can Silently Fail

FINDING ID	#0002
SEVERITY	High Risk
STATUS	Closed
LOCATION	strategy/StrategyTraderJoe.sol -> 59-67

```
1     function deposit(uint256 _depositAmt) public nonReentrant
  whenNotPaused {
2
   IERC20(inputTokenAddress).safeTransferFrom(address(msg.sender),
  address(this), _depositAmt);
3         _deposit(_depositAmt);
4     }
5
6     function _deposit(uint256 _depositAmt) internal {
7         IERC20(inputTokenAddress).safeApprove(jTokenAddress,
  _depositAmt);
8         IJToken(jTokenAddress).mint(_depositAmt);
9     }
```

DESCRIPTION	The deposit function takes MIM tokens (assumed from variable names) and tries to mint JTokens (same as Compound ctokens). However, if the Jtoken mint function call fails, it will silently fail as Compound requires the caller to check the error code being returned. Depending on how the vault contract uses this deposit function, this could be dangerous.
RECOMMENDATION	Check the return code or make sure that the interacting contract checks the change in balance and that it has error handling to cover all scenarios possible that could cause problems.
RESOLUTION	The project team added a check that the mint was successful on deposit. Note: The return value is safe to ignore assuming *mintNative()* is https://github.com/traderjoe-xyz/joe-lending/blob/50883a2da822546a159f51d398fcff8396731543/contracts/JWrappedNative.sol#L72 However, it may be advisable to check the return value as

	future implementations may expect callers to check the return value.

Oracle Is Trusted Implicitly

FINDING ID	#0003
SEVERITY	Low Risk
STATUS	Open
LOCATION	strategy/StrategyTraderJoe.sol -> 70-85

```
1     function withdraw() public onlyMultistrategy nonReentrant {
2
   IXJoe(xJoeAddress).leave(IXJoe(xJoeAddress).balanceOf(address(this))
   );
3
   IJToken(jTokenAddress).redeem(IJToken(jTokenAddress).balanceOf(addre
   ss(this)));
4         uint256 swapAmt =
   IERC20(joeAddress).balanceOf(address(this));
5         IERC20(joeAddress).safeApprove(dexRouterAddress, swapAmt);
6         uint256 minReturnWant = _calculateMinReturn(swapAmt);
7         IJoeRouter(dexRouterAddress).swapExactTokensForTokens(
8             swapAmt,
9             minReturnWant,
10            earnedToInputTokenPath,
11            address(this),
12            block.timestamp.add(600)
13        );
14
15        IERC20(inputTokenAddress).safeTransfer(msg.sender,
   IERC20(inputTokenAddress).balanceOf(address(this)));
16    }
```

LOCATION	strategy/StrategyTraderJoe.sol -> 97-103

```
1     function _calculateMinReturn(uint256 _amount) internal view
   returns (uint256 minReturn) {
2         uint256 oraclePriceUsdPerJoe =
   IOracle(oracle).getLatestPrice(joeAddress);
3         uint256 oraclePriceUsdPerMim =
   IOracle(oracle).getLatestPrice(inputTokenAddress);
4         uint256 priceMimPerJoe =
   oraclePriceUsdPerJoe.mul(1e18).div(oraclePriceUsdPerMim);
5         uint256 total = _amount.mul(priceMimPerJoe).div(1e18);
6         minReturn = total.mul(100 - swapSlippage).div(100);
7     }
```

DESCRIPTION	Contract gathers the price from an oracle and trusts it to

	return a correct value. If the oracle can be manipulated or malfunctions, then tokens could be lost when withdrawing tokens. Depending on what oracle and how this *withdraw()* function is used it could be dangerous to trust the oracle.
RECOMMENDATION	Ensure that the oracle is reliable and provides mitigation against price manipulations.
RESOLUTION	The contract uses Chainlink oracles. This still relies on the accuracy of the Chainlink oracles but mitigates the risk. Onchain note: Contract uses an unaudited oracle implementation, which uses Chainlink Price Feeds. The Price Feeds used by the oracle can be updated by the owner. Currently price feed corresponds to a real chainlink price feed. Oracle contract 0xF34aA0C3c87e0AED8d7c061fd331f9C872BFa226 The project team has noted that there is an audit of the contracts by Slowmist for a previous deployment on Avax (link). The project team also hosts a copy of a Slowmist audit for a deployment to BSC (link).

Warmup Will Break Contract

FINDING ID	#0004
SEVERITY	High Risk
STATUS	Mitigated
LOCATION	strategy/StrategyWonderland.sol -> 74-87

```
1     function _buyAndStake(uint256 _depositAmt) internal {
2         uint256 minReturnWant = _calculateMinTimeReturn(_depositAmt);
3         IERC20(inputTokenAddress).safeApprove(dexRouterAddress,
  _depositAmt);
4         IJoeRouter(dexRouterAddress).swapExactTokensForTokens(
5             _depositAmt,
6             minReturnWant,
7             inputTokenToTimePath,
8             address(this),
9             block.timestamp.add(600)
10        );
11        uint256 timeBalance =
  IERC20(timeAddress).balanceOf(address(this));
12        IERC20(timeAddress).safeApprove(timeStakingHelperAddress,
  timeBalance);
13        IStakingHelper(timeStakingHelperAddress).stake(timeBalance,
  address(this));
14    }
```

DESCRIPTION	When depositing into time staking there is the possibility of a warmup period. If this is the case *StrategyWonderland* would lose access to those funds as it doesn't implement *claim()* or *forfeit()* to receive those funds.
RECOMMENDATION	Add logic to handle a potential warmup period.
RESOLUTION	The project team partially added logic to handle a warmup period. For example if the warmup has not expired, the IStaking.claim() will transfer 0 staked tokens to the strategy. No funds will be lost, but they will be temporarily unavailable until the warmup period finishes. If the strategy contract has a claimed balance of staked tokens and also has tokens which are not claimable yet; withdraw() will result in a partial withdrawal. Specifically only the claimed tokens will be withdrawn from the strategy, care should be taken to ensure this is handled correctly in the base contract.

Bond Price Decimals May Not Match Token Decimals

FINDING ID	#0005
SEVERITY	Medium Risk
STATUS	Closed
LOCATION	strategy/StrategyWonderland.sol -> 60

```
1     function deposit(uint256 _depositAmt) public nonReentrant
  whenNotPaused {
2
   IERC20(inputTokenAddress).safeTransferFrom(address(msg.sender),
  address(this), _depositAmt);
3         if(isMint == true &&
  ITimeBondDepository(timeBondDepositoryAddress).bondPrice() <
  timePrice) {
4             _mint(_depositAmt);
5         } else {
6             _buyAndStake(_depositAmt);
7         }
8     }
```

DESCRIPTION	In the majority of Wonderland style BondDepository contracts typically use 2 decimals for bondPrice, EG a MIM bond whose bondPrice is 1200 has a dollar price of $12. LP bonds can have a variety of scaling factors and two different LP bonds might have a different dollar price for the same bondPrice. Different bonds may produce different or incorrect swap rates depending on how many decimals are used or what the scaling factor is.
RECOMMENDATION	Add logic to account for any potential difference between the number of decimals.
RESOLUTION	The project team has stated the decimals value for bond price will be confirmed and updated (if needed) for any future contracts. (Decimals could differ between bonding contracts)

Only Reward Type 0 Is Claimed

FINDING ID	#0006
SEVERITY	Low Risk
STATUS	Closed
LOCATION	strategy/StrategyTraderJoe.sol -> 51-57

```
1     function stakeJoe() public nonReentrant whenNotPaused {
2         IJoetroller(joetrollerAddress).claimReward(0, address(this));
3         uint256 stakeAmt =
  IERC20(joeAddress).balanceOf(address(this));
4
5         IERC20(joeAddress).safeApprove(xJoeAddress, stakeAmt);
6         IXJoe(xJoeAddress).enter(stakeAmt);
7     }
```

DESCRIPTION	The *claimReward()* is only called for type 0. *claimReward()* is never called for type 1. If type 0 rewards exist, any rewards would not be claimed (type 1 is native tokens i.e the token for the chain the project is on which is avax in this case). If type one is used, then the contract requires a payable function. Refer to TraderJoe contracts: https://github.com/traderjoe-xyz/joe-lending/blob/main/contracts/Joetroller.sol#L1378 https://github.com/traderjoe-xyz/joe-lending/blob/main/contracts/RewardDistributor.sol#L410
RECOMMENDATION	Add *claimReward()* for type 1 tokens if type 1 tokens give out rewards for the used Jtoken. This requires the contract to be payable.
RESOLUTION	The team has added a fix for the issue. Note: It may be worth moving the mechanism for claiming the reward token to its own function outside of withdrawal or include it in the stakejoe function instead. A withdrawal currently will cost a lot of gas; however, such a change is optional. Note: Also the *stakeJoe* function is never called in a

	withdrawal which could mean some unclaimed funds are still left in the contract. (It might be called in the *Multistrategy* contract which has not been examined)

Gas Optimization

FINDING ID	#0007
SEVERITY	Informational
STATUS	Closed
LOCATION	strategy/StrategyTraderJoe.sol -> 76-82

```
1         IJoeRouter(dexRouterAddress).swapExactTokensForTokens(
2             swapAmt,
3             minReturnWant,
4             earnedToInputTokenPath,
5             address(this),
6             block.timestamp.add(600)
7         );
```

DESCRIPTION	In addition to the deadline parameter in *swapExactTokensForTokens()* when *block.timestamp* is determined within the current transaction will not make a difference. The value of *block.timestamp* is fixed within a transaction and will not change during its execution.
RECOMMENDATION	Remove the addition of 600 to *block.timestamp*.
RESOLUTION	The project team implemented the recommended fix.

Unnecessary Override

FINDING ID	#0008
SEVERITY	Informational
STATUS	Closed
LOCATION	strategy/StrategyTraderJoe.sol -> 87-95

```
1    function _pause() internal override {
2         super._pause();
3
4     }
5
6     function _unpause() internal override {
7         super._unpause();
8
9     }
```

DESCRIPTION	Currently the override implementation does nothing.
RECOMMENDATION	Remove the override implementation.
RESOLUTION	The project team implemented the recommended fix.

Dangerous Modifier

FINDING ID	#0009
SEVERITY	Informational
STATUS	Open
LOCATION	refs/CoreRef.sol -> 53-59

```
1     modifier onlyRoleOrOpenRole(bytes32 role) {
2         require(
3             _core.hasRole(role, address(0)) || _core.hasRole(role,
  msg.sender),
4             "CoreRef::onlyRoleOrOpenRole: Not permit"
5         );
6         _;
7     }
```

DESCRIPTION	Letting a modifier work when a role is unassigned could be dangerous if it is misused.
RECOMMENDATION	Remove the modifier or be cautious when using it. (currently not used in audited contract's)
RESOLUTION	N/A

Protocol Values Should Be Public

FINDING ID	#0010
SEVERITY	Informational
STATUS	Open
LOCATION	@OpenZeppelin - AccessControl.sol -> 48-53

```
1    struct RoleData {
2        EnumerableSet.AddressSet members;
3        bytes32 adminRole;
4    }
5
6    mapping (bytes32 => RoleData) private _roles;
```

DESCRIPTION	Variables critical to the operation of the protocol should be public or have an associated view function.
RECOMMENDATION	Add getter functions or change the values to be public. For *AccessControl*, the *AccessControlEnumerable* contract provides this functionality.
RESOLUTION	N/A

Static Analysis

Division Before Multiplication

FINDING ID	#0011
SEVERITY	Low Risk
STATUS	Partially Closed
LOCATION	strategy/StrategyTraderJoe.sol -> 97-103

```
1     function _calculateMinReturn(uint256 _amount) internal view
  returns (uint256 minReturn) {
2         uint256 oraclePriceUsdPerJoe =
  IOracle(oracle).getLatestPrice(joeAddress);
3         uint256 oraclePriceUsdPerMim =
  IOracle(oracle).getLatestPrice(inputTokenAddress);
4         uint256 priceMimPerJoe =
  oraclePriceUsdPerJoe.mul(1e18).div(oraclePriceUsdPerMim);
5         uint256 total = _amount.mul(priceMimPerJoe).div(1e18);
6         minReturn = total.mul(100 - swapSlippage).div(100);
7     }
```

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. (*mul(1e18)* and *div(1e18)* cancel each other out).
RESOLUTION	The project team has implemented the recommended fix for *strategy/StrategyTraderJoe.sol* but not *strategy/StrategyWonderland.sol*. The risk of rounding error is minimal, but present.

No Events Emitted For Changes To Protocol Values

FINDING ID	#0012
SEVERITY	Informational
STATUS	Closed
LOCATION	• strategy/StrategyWonderland.sol -> 127-129: *function setIsMint(bool _isMint) public onlyGovernor {* • strategy/StrategyWonderland.sol -> 131-134: *function setTimePrice(uint256 _timePrice) public onlyGovernor {* • strategy/StrategyWonderland.sol -> 136-138: *function setDexRouter(address _dexRouterAddress) public onlyGovernor {* • strategy/StrategyWonderland.sol -> 140-142: *function setInputTokenToTimePath(address[] calldata _inputTokenToTimePath) public onlyGovernor {* • strategy/StrategyWonderland.sol -> 144-146: *function setEarnedToInputTokenPath(address[] calldata _earnedToInputTokenPath) public onlyGovernor {* • strategy/StrategyWonderland.sol -> 148-151: *function setSlippage(uint256 _swapSlippage) public onlyGovernor {*

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 fix.

On-Chain Analysis

Timelock Can Be Bypassed

FINDING ID	#0013
SEVERITY	High Risk
STATUS	Open
LOCATION	0x881a0E89A3F010635E8967BB90F8E7132eF0f09F *Core.GOVERN_ROLE*

DESCRIPTION	A malicious actor with the *GOVERN_ROLE* can add new addresses with the *TIMELOCK_ROLE*. For the audited contracts, the timelock role is only able to withdraw “stuck” tokens. However, the govern role is able to change which tokens cannot be withdrawn via the timelock. Refer to finding #17. Furthermore, this may adversely impact other contracts in the project.
RECOMMENDATION	Change the role admin of the timelock to the timelock. Obelisk recommends a timelock delay of at least 72 hours.
RESOLUTION	Fixed in the *Core* contract. However, the timelock is non-standard and is not audited.

Withdraw Function Is Not Initialized

FINDING ID	#0014
SEVERITY	High Risk
STATUS	Mitigated
LOCATION	StrategyTraderJoe 0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b

DESCRIPTION	Currently the values of *isJoeReward* and *isAvaxReward* are set to false. This will prevent the *withdraw()* function from completing the withdrawal process for the rewards and may leave unretrieved funds behind.
RECOMMENDATION	Ensure that the contracts are correctly initialized.
RESOLUTION	The project team stated the pool used had no rewards.

Oracle Can Be Changed

FINDING ID	#0015
SEVERITY	High Risk
STATUS	Open
LOCATION	StrategyTraderJoe 0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b

DESCRIPTION	The *GOVERN_ROLE* (currently a multisig) is able to change the oracle without a timelock. Changes to this address may happen without notice and can cause the loss of user funds
RECOMMENDATION	Set the role which can change the oracle to the Timelock. Obelisk recommends a timelock delay of at least 72 hours.
RESOLUTION	The contract still uses the governor's role to change the price feed. This role is currently under a timelock, but the timelock contract is nonstandard and is not audited.

Contracts Owners Are Unaudited

FINDING ID	#0016
SEVERITY	High Risk
STATUS	Open
LOCATION	StrategyTraderJoe 0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b *StrategyTraderJoe.owner* StrategyWonderland 0x513c123F62A23e2d65595a33a9c39aCe4d1315FF *StrategyWonderland.owner* Core 0x881a0E89A3F010635E8967BB90F8E7132eF0f09F *Core.MULTISTRATEGY_ROLE*

DESCRIPTION	The *owner* of StrategyWonderland and StrategyTraderJoe contracts, as well as the *MULTISTRATEGY_ROLE*, are an unaudited contract. This contract is essential to the withdrawal of user deposits via the *MULTISTRATEGY_ROLE*. MultiStrategy 0x1c71C3c32F47D9490647f711026a08240e7e90aE
RECOMMENDATION	Ensure that the contracts' implementation and deployment is correct.
RESOLUTION	The project team has noted that there is an audit of the contracts by Slowmist for a previous deployment on Avax (link). The project team also hosts a copy of a Slowmist audit for a deployment to BSC (link).

Router and Trade Paths Can Be Changed

FINDING ID	#0017
SEVERITY	High Risk
STATUS	Partially Closed
LOCATION	StrategyTraderJoe 0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b StrategyWonderland 0x513c123F62A23e2d65595a33a9c39aCe4d1315FF

DESCRIPTION	The timelock is able to change the path and the dex router. A malicious actor controlling the timelock can lock user funds, or change the router to redirect swapped tokens. In combination with the timelock role, a malicious actor can change tokens held in the contract to bypass the check for *inCaseTokensGetStuck()*. This will allow them to drain all the tokens in the StrategyWonderland and some tokens in StrategyTraderJoe.
RECOMMENDATION	Set the govern role to the timelock. Obelisk recommends a timelock delay of at least 72 hours.
RESOLUTION	The contracts were updated in revision 6: 5a9581699e670b81f98fec75c89216dd32965835 The router and trade paths are now only changeable by the timelock. Note that the Timelock contract is non-standard timelock, and is not audited.

Timelock Minimum Delay Is Short

FINDING ID	#0018
SEVERITY	Medium Risk
STATUS	Open
LOCATION	TimelockController 0x4D63799b2c446e7ea0Acf1DAe055B97F8741e4C9

DESCRIPTION	The timelock delay is set to 2 hours. Obelisk recommends a timelock delay for all functionality of at least 72 hours.
RECOMMENDATION	Set the minimum delay of timelock to 72 hours.
RESOLUTION	The timelock's *minDelayCritical* is 72 hours. However the *minDelayNormal* is still 3 hours. Note that the timelock is not standard, and is not audited.

External Addresses

Externally Owned Accounts

Multisig Owners

ACCOUNT	0x351273EbF1790A1e144afC4f436d459ccF90cafA 0x0258A120fb8a81d48da01B0D013f890323B7F8Ec 0x8aA57F27Bac2816d6b757E1521f532165f8b23c8 0x93beb9A49bd95AF96A1B1068e90761B45545af0E
USAGE	0x881a0E89A3F010635E8967BB90F8E7132eF0f09F *Core.EXECUTOR_ROLE* *Core.PROPOSER_ROLE* Multisig address (2 of 4 signatures required): 0x1ACC14EAf24F87835A9cc2F1F2DBcEEd1C7Fc324
IMPACT	• receives elevated permissions as owner, operator, or other

External Contracts

These contracts are not part of the audit scope.

Bond Depository

ADDRESS	0x103F6bd55C192b86aD576C0c36Be7AB0945Ebe48
USAGE	0x513c123F62A23e2d65595a33a9c39aCe4d1315FF *StrategyWonderland_old.bondDepositoryAddress*
IMPACT	• impacts ability to deposit or withdraw tokens

Joetroller

ADDRESS	0xdc13687554205E5b89Ac783db14bb5bba4A1eDaC
USAGE	0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b *StrategyTraderJoe.joetrollerAddress*
IMPACT	• impacts ability to deposit or withdraw tokens

MultiStrategy

ADDRESS	0x1c71C3c32F47D9490647f711026a08240e7e90aE
USAGE	0x881a0E89A3F010635E8967BB90F8E7132eF0f09F *Core.MULTISTRATEGY_ROLE* 0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b *StrategyTraderJoe.owner* 0x513c123F62A23e2d65595a33a9c39aCe4d1315FF *StrategyWonderland_old.owner*
IMPACT	• receives elevated permissions as owner, operator, or other

Oracle

ADDRESS	0xF34aA0C3c87e0AED8d7c061fd331f9C872BFa226
USAGE	0x881a0E89A3F010635E8967BB90F8E7132eF0f09F *Core.TIMELOCK_ROLE* 0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b *StrategyTraderJoe.oracle*
IMPACT	• impacts ability to deposit or withdraw tokens

Staking Contract

ADDRESS	0x6d7AD602Ec2EFdF4B7d34A9A53f92F06d27b82B1
USAGE	0x513c123F62A23e2d65595a33a9c39aCe4d1315FF *StrategyWonderland_old.stakingAddress*
IMPACT	• impacts ability to deposit or withdraw tokens

Staking Helper

ADDRESS	0x93c375fDA3158b18889437D30049F2ABeFA34275
USAGE	0x513c123F62A23e2d65595a33a9c39aCe4d1315FF *StrategyWonderland_old.stakingHelperAddress*
IMPACT	• impacts ability to deposit or withdraw tokens

TimelockController

ADDRESS	0x4D63799b2c446e7ea0Acf1DAe055B97F8741e4C9
USAGE	0x8d75996F928AbD0DA2b202de5a464B8C716F043a *Core.GOVERN_ROLE* *Core.TIMELOCK_ROLE*
IMPACT	• receives elevated permissions as owner, operator, or other

TraderJoeRouter

ADDRESS	0x60aE616a2155Ee3d9A68541Ba4544862310933d4

USAGE	0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b *StrategyTraderJoe.dexRouterAddress* 0x513c123F62A23e2d65595a33a9c39aCe4d1315FF *StrategyWonderland_old.dexRouterAddress*
IMPACT	• receives transfer of tokens deposited by users

TrancheMaster

ADDRESS	0xABb14DDE57E78e64d9a645aA537b408bA3f643D4
USAGE	0x881a0E89A3F010635E8967BB90F8E7132eF0f09F *Core.MASTER_ROLE*
IMPACT	• does not impact audited contracts

External Tokens

These contracts are not part of the audit scope.

Avalanche Bridge Dai.e

ADDRESS	0xd586E7F844cEa2F87f50152665BCbc2C279D8d70
USAGE	0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b *StrategyTraderJoe.inputTokenAddress* 0x513c123F62A23e2d65595a33a9c39aCe4d1315FF *StrategyWonderland_old.inputTokenAddress*
IMPACT	• ERC20 Token

Banker Joe Dai

ADDRESS	0xc988c170d0E38197DC634A45bF00169C7Aa7CA19
USAGE	0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b *StrategyTraderJoe.jTokenAddress*
IMPACT	• ERC20 Token

Joe Token

ADDRESS	0x6e84a6216eA6dACC71eE8E6b0a5B7322EEbC0fDd
USAGE	0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b *StrategyTraderJoe.joeAddress*
IMPACT	• ERC20 Token

Maxi Token

ADDRESS	0x7C08413cbf02202a1c13643dB173f2694e0F73f0
USAGE	0x513c123F62A23e2d65595a33a9c39aCe4d1315FF *StrategyWonderland_old.baseTokenAddress*
IMPACT	• ERC20 Token

Staked Maxi Token

ADDRESS	0xEcE4D1b3C2020A312Ec41A7271608326894076b4
USAGE	0x513c123F62A23e2d65595a33a9c39aCe4d1315FF *StrategyWonderland_old.stakedTokenAddress*
IMPACT	• ERC20 Token

Wrapped AVAX

ADDRESS	0xB31f66AA3C1e785363F0875A1B74E27b85FD66c7
USAGE	0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b *StrategyTraderJoe.wavaxAddress*
IMPACT	• ERC20 Token

Appendix A - Reviewed Documents

Document	Address
core/Core.sol	0x881a0E89A3F010635E8967BB90F8E7132eF0f09F
core/Permissions.sol	N/A
interfaces/ICore.sol	N/A
interfaces/IOracle.sol	N/A
interfaces/ITraderJoe.sol	N/A
interfaces/IWonderland.sol	N/A
refs/CoreRef.sol	N/A
strategy/StrategyTraderJoe.sol	0x7764Be06A17c842FFaec8d8363A21Ed0Fb829e3b
strategy/StrategyWonderland_old.sol	0x513c123F62A23e2d65595a33a9c39aCe4d1315FF

Revisions

Revision 1	19a1b63e4348d84593534ffdcdbf7f0e764f90d0
Revision 2	6be62600684dab5adfef7250c50fee6a5f4fb839
Revision 3	321fd174d438759169d3dc9343914a7665ef15f4
Revision 4	a578a955a2b18b746d6ae95f9cac79b64bbcc167
Revision 5	0a3f7a50019c8e25ce135de955927bf397396a09
Revision 6	5a9581699e670b81f98fec75c89216dd32965835

Imported Contracts

OpenZeppelin	3.4.2

Appendix B - Risk Ratings

Risk	Description
High Risk	A fatal vulnerability that can cause the loss of all Tokens / Funds.
Medium Risk	A vulnerability that can cause the loss of some Tokens / Funds.
Low Risk	A vulnerability which can cause the loss of protocol functionality.
Informational	Non-security issues such as functionality, style, and convention.

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 - 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