OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 32	2022-04-27	Plemonade, Donut	Audit Final

Audit Notes

Audit Date	2022-04-22 - 2022-04-27
Auditor/Auditors	Plemonade, Mechwar
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	OB532568759

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	Redemption
Description	“The first true Profit Optimizer powered by magic and fantasy, protected by Riskia the Redeemed.”
Website	https://www.redemption.fi/
Contact	https://twitter.com/RedemptionPO
Contact information	@ThisRisky#1111 on Discord
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	Fantom

Audit of Redemption

Obelisk was commissioned by Redemption on the 18th of April 2022 to conduct a comprehensive audit of Redemptions' contracts. The following audit was conducted between the 22nd of April 2022 and the 27th of April 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.

Obelisk was commissioned to audit v2 of the Redemption project before deployment in order to check for possible security issues. The auditors found a possible high-risk issue #1, that the project team solved. Solving issue #1 created low-risk issue #2 which is partially solved as tokens can still be paused, but now it's all or nothing which cannot be abused as it could in issue #1.

Issue #3 is a low-risk issue that refers to a move away from the standard Uniswap structure but is partially closed by the project team's own implementation. Low-risk issue #4 was fully closed.

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, however, please take a note of them. Please see Appendix B and C for further explanation on issue ratings.

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

Code Analysis

Finding	ID	Severity	Status
Centralization On Liquidity Pairs (A)	#0001	High Risk	Closed
Centralization on liquidity pairs (B)	#0002	Low Risk	Partially Closed
Missing Fees	#0003	Low Risk	Partially Closed
Migrator Interaction	#0004	Low Risk	Closed
Gas Optimization 1	#0005	Informational	Open
Fee Interaction With Flashswaps	#0006	Informational	Open
Missing Zero Checks	#0007	Informational	Closed
Gas Optimization/Unnecessary Code	#0008	Informational	Closed

On-Chain Analysis

No Findings

Findings

Code Analysis

Centralization On Liquidity Pairs (A)

FINDING ID	#0001
SEVERITY	High Risk
STATUS	Closed
LOCATION	RedemptionPair.sol -> 91-123

```
1     /**
2      * @dev Modifier to make a function callable only when the contract
  is not paused.
3      */
4     modifier whenNotPaused() {
5         require(!paused);
6         _;
7     }
8
9     /**
10     * @dev Modifier to make a function callable only when the contract
  is paused.
11     */
12    modifier whenPaused() {
13        require(paused);
14        _;
15    }
16
17    /**
18     * @dev called by the owner to pause, triggers stopped state
19     */
20    function pause() public whenNotPaused {
21        require(msg.sender ==
  IRedemptionFactory(factory).feeToSetter(), 'RedemptionV2: FORBIDDEN');
22        paused = true;
23        emit Pause();
24    }
25
26    /**
27     * @dev called by the owner to unpause, returns to normal state
28     */
29    function unpause() public whenPaused {
30        require(msg.sender ==
  IRedemptionFactory(factory).feeToSetter(), 'RedemptionV2: FORBIDDEN');
31        paused = false;
32        emit Unpause();
33    }
```

LOCATION	RedemptionPair.sol -> 257-279

```
1     function burn(address to) external lock whenNotPaused returns
  (uint256 amount0, uint256 amount1) {
2         // ...
3     }
```

DESCRIPTION	The *burn()* function can be paused by the *feeToSetter* address. A pause would lock the liquidity tokens inside the contract until the *feeSetter* unpauses the contract. Generally, a DEX (decentralized exchange) liquidity pair should not be centralized such that it can be frozen as it brings centralization risks to users.
RECOMMENDATION	Obelisk recommends not having a pause and unpause function on the *burn()* function. If this feature is deemed important, make sure the *feeToSetter* address is under a timelock and educate users that their asset can be frozen inside the liquidity pair.
RESOLUTION	The project team removed the pause functionality from the *burn()* function.

Centralization on liquidity pairs (B)

FINDING ID	#0002
SEVERITY	Low Risk
STATUS	Partially Closed
LOCATION	RedemptionPair.sol -> 91-123

```
1     /**
2      * @dev Modifier to make a function callable only when the contract
  is not paused.
3      */
4     modifier whenNotPaused() {
5         require(!paused);
6         _;
7     }
8
9     /**
10     * @dev Modifier to make a function callable only when the contract
  is paused.
11     */
12    modifier whenPaused() {
13        require(paused);
14        _;
15    }
16
17    /**
18     * @dev called by the owner to pause, triggers stopped state
19     */
20    function pause() public whenNotPaused {
21        require(msg.sender ==
  IRedemptionFactory(factory).feeToSetter(), 'RedemptionV2: FORBIDDEN');
22        paused = true;
23        emit Pause();
24    }
25
26    /**
27     * @dev called by the owner to unpause, returns to normal state
28     */
29    function unpause() public whenPaused {
30        require(msg.sender ==
  IRedemptionFactory(factory).feeToSetter(), 'RedemptionV2: FORBIDDEN');
31        paused = false;
32        emit Unpause();
33    }
```

LOCATION	RedemptionPair.sol -> 226-254

```
1     function mint(address to) external lock whenNotPaused returns
  (uint256 liquidity) {
2         // ...
3     }
```

LOCATION	RedemptionPair.sol -> 282-336

```
1     function swap(
2         uint256 amount0Out,
3         uint256 amount1Out,
4         address to,
5         bytes calldata data
6     ) external lock whenNotPaused {
7         // ...
8     }
```

DESCRIPTION	The *swap()* and *mint()* functions can be paused by the *feeToSetter* address. A pause would lock the trading and deposit functionality until the *feeSetter* unpauses the contract. Generally, a DEX (decentralized exchanges) liquidity pair should not be centralized such that it can be frozen as it brings centralization risks to users.
RECOMMENDATION	Obelisk recommends not having a pause and unpause for the *swap()* and *mint()* functions in a liquidity pair. But if this feature is deemed important: • Make sure the *feeToSetter* address is under a timelock (users should have time to react). • Move the pause functionality to the factory so that specific pair tokens can not be unilaterally paused (specific tokens can be banned from the smart contract).
RESOLUTION	Issue is partially closed. The project team implemented the recommendation to move the pause into the factory contract. Pausing can still occur but every token would be affected which is more in line with the intended security feature. The current *feeToSetter* is a multisig.

	Multisig: (1 out of 3 signers) 0x18F06E4c059A7D610802DdE64291938DD7308069 0xd5ad7b4B3a5478f390254E26f70A3D8B169F9075 0xf7afF8e3FFeE3dfa001A85af07c197f60dFb3e7d

Missing Fees

FINDING ID	#0003
SEVERITY	Low Risk
STATUS	Partially Closed
LOCATION	RedemptionPair.sol -> 282-336

```
1   function swap(
2         uint256 amount0Out,
3         uint256 amount1Out,
4         address to,
5         bytes calldata data
6     ) external lock whenNotPaused {
7         require(amount0Out > 0 || amount1Out > 0, 'RedemptionV2:
  INSUFFICIENT_OUTPUT_AMOUNT');
8         (uint112 _reserve0, uint112 _reserve1, ) = getReserves(); //
  gas savings
9         require(amount0Out < _reserve0 && amount1Out < _reserve1,
  'RedemptionV2: INSUFFICIENT_LIQUIDITY');
10
11        uint256 balance0;
12        uint256 balance1;
13        FeesAndAmounts memory feesAndAmounts = _getFees(amount0Out,
  amount1Out);
14        {
15            // scope for _token{0,1}, avoids stack too deep errors
16            address _token0 = token0;
17            address _token1 = token1;
18            require(to != _token0 && to != _token1, 'RedemptionV2:
  INVALID_TO');
19            if (amount0Out > 0) {
20                _safeTransfer(_token0, controllerFeeAddress,
  feesAndAmounts.fee0);
21                _safeTransfer(_token0, to,
  feesAndAmounts.amount0OutAfterFee); // optimistically transfer tokens
22            }
23            if (amount1Out > 0) {
24                _safeTransfer(_token1, controllerFeeAddress,
  feesAndAmounts.fee1);
25                _safeTransfer(_token1, to,
  feesAndAmounts.amount1OutAfterFee); // optimistically transfer tokens
26            }
27            if (data.length > 0) {
28                IRedemptionCallee(to).RedemptionCall(
29                    msg.sender,
30                    feesAndAmounts.amount0OutAfterFee,
31                    feesAndAmounts.amount1OutAfterFee,
32                    data
33                );
34            }
```

```
35              balance0 =
IERC20Redemption(_token0).balanceOf(address(this));
36              balance1 =
IERC20Redemption(_token1).balanceOf(address(this));
37          }
38          AmountsIn memory amountsIn = AmountsIn(
39              balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 -
amount0Out) : 0,
40              balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 -
amount1Out) : 0
41          );
42          require(amountsIn.amount0In > 0 || amountsIn.amount1In > 0,
'RedemptionV2: INSUFFICIENT_INPUT_AMOUNT');
43          {
44              // scope for reserve{0,1}Adjusted, avoids stack too deep
errors
45              uint256 balance0Adjusted =
balance0.mul(10000).sub(amountsIn.amount0In.mul(feeAmount));
46              uint256 balance1Adjusted =
balance1.mul(10000).sub(amountsIn.amount1In.mul(feeAmount));
47              require(
48                  balance0Adjusted.mul(balance1Adjusted) >=
uint256(_reserve0).mul(_reserve1).mul(10000**2),
49                  'RedemptionV2: K'
50              );
51          }
52
53          _update(balance0, balance1, _reserve0, _reserve1);
54          emit Swap(msg.sender, amountsIn.amount0In, amountsIn.amount1In,
amount0Out, amount1Out, to);
55      }
```

LOCATION	RedemptionPair.sol -> 338-342

```
1      function _getFees(uint256 amount0Out, uint256 amount1Out) private
view returns (FeesAndAmounts memory) {
2          uint256 fee0 = amount0Out.mul(controllerFeeAddress !=
address(0) ? controllerFeeShare : 0) / 10000;
3          uint256 fee1 = amount1Out.mul(controllerFeeAddress !=
address(0) ? controllerFeeShare : 0) / 10000;
4          return FeesAndAmounts(fee0, fee1, amount0Out - fee0, amount1Out
- fee1);
5      }
```

DESCRIPTION	In some cases, a transaction will pay fewer fees than it should due to a precision loss. The *_getFees()* calculation uses *((amountOut*fee)/10000)*. This will mean in some cases, a transaction could pay 0% controller fees (i.e when: *amountOut*fee < 10000*). If gas

	fees are low enough and token value is high enough, it would be possible to do lots of small swaps to pay less fees.
RECOMMENDATION	Implement Uniswap's way of calculating fees to prevent this from happening. Note that Uniswap instead would charge more (not possible to pay a fraction off 1 wei) which makes it impossible to exploit (although you could end up paying 1 more wei than necessary).
RESOLUTION	The project team partially solved the issue by rounding up if fee is 0 and amountOut is not 0. The issue still persists but it cannot be exploited to a significant extent (no free trades but fee reduction up to 1/2 when 1 wei is more than the gas fee). Losses of 1 wei (to the controller) will occur in certain trades but the project team deemed it not to be worth addressing.

Migrator Interaction

FINDING ID	#0004
SEVERITY	Low Risk
STATUS	Closed
LOCATION	RedemptionPair.sol -> 235-247

```
1         if (_totalSupply == 0) {
2             address migrator = IRedemptionFactory(factory).migrator();
3             if (msg.sender == migrator) {
4                 liquidity = IMigrator(migrator).desiredLiquidity();
5                 require(liquidity > 0 && liquidity != uint256(-1), 'Bad
  desired liquidity');
6             } else {
7                 require(migrator == address(0), 'Must not have
  migrator');
8                 liquidity =
  Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
9                 _mint(address(0), MINIMUM_LIQUIDITY); // permanently
  lock the first MINIMUM_LIQUIDITY tokens
10            }
11        } else {
12            liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0,
  amount1.mul(_totalSupply) / _reserve1);
13        }
```

DESCRIPTION	Obelisk has not received the source code for the migrator contract. For example, the migrator can optimistically choose liquidity tokens to mint. However, nothing enforces any transfer of tokens in such a case (which causes reserve values to be 0. Division by 0 issues would arise). Also, a malicious actor could also front-run the setting of a migrator address in the factory. It could then create the intended pairs, which would cause the migrator to fail if the migrator does not account for it.
RECOMMENDATION	Make sure the migrator contract accounts for all issues that could arise. If the migrator function is not used, Obelisk recommends removing the migrator code from the factory and pair contracts.
RESOLUTION	The project team implemented the recommended fix.

Gas Optimization 1

FINDING ID	#0005
SEVERITY	Informational
STATUS	Open
LOCATION	RedemptionPair.sol -> 226-254

```
1     function mint(address to) external lock whenNotPaused returns
  (uint256 liquidity) {
2         // ...
3     }
```

DESCRIPTION	Controller fees are sent at the start of a swap. This could have been done with uniswaps *mintFee()*. From Uniswap documentation: "Collecting this 0.05% fee at the time of the trade would impose an additional gas cost on every trade. To avoid this, accumulated fees are collected only when liquidity is deposited or withdrawn". This would save 1-2 transfers per swap and gas fees for the calculations(in the swap function)
RECOMMENDATION	Obelisk recommends making use of *mintfee* to reduce gas costs for a swap.
RESOLUTION	Project team comment: "we need the swap fees to be transferred as soon as the swaps happen as we have other business logic based on this in other contracts"

Fee Interaction With Flashswaps

FINDING ID	#0006
SEVERITY	Informational
STATUS	Open
LOCATION	RedemptionPair.sol -> 282-336

```
  function swap(
        uint256 amount0Out,
        uint256 amount1Out,
        address to,
        bytes calldata data
    ) external lock whenNotPaused {
        require(amount0Out > 0 || amount1Out > 0, 'RedemptionV2:
INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1, ) = getReserves(); //
gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1,
'RedemptionV2: INSUFFICIENT_LIQUIDITY');

        uint256 balance0;
        uint256 balance1;
        FeesAndAmounts memory feesAndAmounts = _getFees(amount0Out,
amount1Out);
        {
            // scope for _token{0,1}, avoids stack too deep errors
            address _token0 = token0;
            address _token1 = token1;
            require(to != _token0 && to != _token1, 'RedemptionV2:
INVALID_TO');
            if (amount0Out > 0) {
                _safeTransfer(_token0, controllerFeeAddress,
feesAndAmounts.fee0);
                _safeTransfer(_token0, to,
feesAndAmounts.amount0OutAfterFee); // optimistically transfer tokens
            }
            if (amount1Out > 0) {
                _safeTransfer(_token1, controllerFeeAddress,
feesAndAmounts.fee1);
                _safeTransfer(_token1, to,
feesAndAmounts.amount1OutAfterFee); // optimistically transfer tokens
            }
            if (data.length > 0) {
                IRedemptionCallee(to).RedemptionCall(
                    msg.sender,
                    feesAndAmounts.amount0OutAfterFee,
                    feesAndAmounts.amount1OutAfterFee,
                    data
                );
            }
```

```
35              balance0 =
IERC20Redemption(_token0).balanceOf(address(this));
36              balance1 =
IERC20Redemption(_token1).balanceOf(address(this));
37          }
38          AmountsIn memory amountsIn = AmountsIn(
39              balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 -
amount0Out) : 0,
40              balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 -
amount1Out) : 0
41          );
42          require(amountsIn.amount0In > 0 || amountsIn.amount1In > 0,
'RedemptionV2: INSUFFICIENT_INPUT_AMOUNT');
43          {
44              // scope for reserve{0,1}Adjusted, avoids stack too deep
errors
45              uint256 balance0Adjusted =
balance0.mul(10000).sub(amountsIn.amount0In.mul(feeAmount));
46              uint256 balance1Adjusted =
balance1.mul(10000).sub(amountsIn.amount1In.mul(feeAmount));
47              require(
48                  balance0Adjusted.mul(balance1Adjusted) >=
uint256(_reserve0).mul(_reserve1).mul(10000**2),
49                  'RedemptionV2: K'
50              );
51          }
52
53          _update(balance0, balance1, _reserve0, _reserve1);
54          emit Swap(msg.sender, amountsIn.amount0In, amountsIn.amount1In,
amount0Out, amount1Out, to);
55      }
```

LOCATION	RedemptionPair.sol -> 338-342

```
1       function _getFees(uint256 amount0Out, uint256 amount1Out) private
view returns (FeesAndAmounts memory) {
2           uint256 fee0 = amount0Out.mul(controllerFeeAddress !=
address(0) ? controllerFeeShare : 0) / 10000;
3           uint256 fee1 = amount1Out.mul(controllerFeeAddress !=
address(0) ? controllerFeeShare : 0) / 10000;
4           return FeesAndAmounts(fee0, fee1, amount0Out - fee0, amount1Out
- fee1);
5       }
```

DESCRIPTION	When a flashswap is executed more will need to be borrowed than a Uniswap equivalent, as the controller fee is taken before the optimistic transfer. This will need to be accounted for in the external contracts to utilize the flashswap feature of the exchange pair efficiently. Flashswap transactions will end up paying more in many

	cases than a uniswap equivalent if swap fees are turned on. However, for a normal user, there is no difference.
RECOMMENDATION	Implement the Uniswap design of taking fees or make sure to document the different designs such that arbitrage bots can easily understand how to use the pair.
RESOLUTION	Project team comment: "We have implemented the fees as we have other business logic based on this. the amounts transferred to the caller are being sent to the callback function"

Missing Zero Checks

FINDING ID	#0007
SEVERITY	Informational
STATUS	Closed
LOCATION	• RedemptionFactory.sol -> 18-20: *constructor(address _feeToSetter) public {* • RedemptionFactory.sol -> 57-60: *function setFeeToSetter(address _feeToSetter) external override {*

DESCRIPTION	The *feeToSetter* can be set to zero address in the aforementioned constructor and setter functions. A zero address *feeToSetter* will disable some contract functionality.
RECOMMENDATION	Add a check to ensure that the *feeToSetter* address is never set to an invalid zero address.
RESOLUTION	The project team implemented the recommended fix.

Gas Optimization/Unnecessary Code

FINDING ID	#0008
SEVERITY	Informational
STATUS	Closed
LOCATION	RedemptionPair.sol -> 338-342

```
1     function _getFees(uint256 amount0Out, uint256 amount1Out) private
  view returns (FeesAndAmounts memory) {
2         uint256 fee0 = amount0Out.mul(controllerFeeAddress !=
  address(0) ? controllerFeeShare : 0) / 10000;
3         uint256 fee1 = amount1Out.mul(controllerFeeAddress !=
  address(0) ? controllerFeeShare : 0) / 10000;
4         return FeesAndAmounts(fee0, fee1, amount0Out - fee0, amount1Out
  - fee1);
5     }
```

DESCRIPTION	The *controllerFeeAddress* cannot be zero, therefore the conditional (ternary) check is unnecessary.
RECOMMENDATION	Remove the conditional (ternary) check and multiply by *controllerFeeShare* for both *fee0* and *fee1*.
RESOLUTION	The project team has removed the conditional (ternary) check.

On-Chain Analysis

No Findings

External Addresses

Externally Owned Accounts

FeeToSetter Multisig Signers

ACCOUNT	0x18F06E4c059A7D610802DdE64291938DD7308069 0xd5ad7b4B3a5478f390254E26f70A3D8B169F9075 0xf7afF8e3FFeE3dfa001A85af07c197f60dFb3e7d
USAGE	0xDEac7Afb519B7bb2865725f5d13d1097bC5aAf32 *Signers*
IMPACT	• receives elevated permissions as owner, operator, or other

External Contracts

These contracts are not part of the audit scope.

GnosisSafeProxy

ADDRESS	0xDEac7Afb519B7bb2865725f5d13d1097bC5aAf32
USAGE	0xa2dF50d1401afF182D19Bb41d76cf35953942c51 *RedemptionFactory.feeTo* - Variable *RedemptionFactory.feeToSetter* - Variable
IMPACT	• receives elevated permissions as owner, operator, or other

External Tokens

These contracts are not part of the audit scope.

No external tokens

Appendix A - Reviewed Documents

Deployed Contracts

Document	Address
RedemptionFactory	0xa2dF50d1401afF182D19Bb41d76cf35953942c51
RedemptionRouter02	0x6726a3c9aFD89Ce0beaC9A7F6D3cc6870dD26331

Libraries And Interfaces

IERC20.sol IRedemptionCallee.sol IRedemptionERC20.sol IRedemptionFactory.sol IRedemptionPair.sol IRedemptionRouter01.sol IRedemptionRouter02.sol IWETH.sol Math.sol RedemptionLibrary.sol SafeMath.sol TransferHelper.sol UQ112x112.sol RedemptionERC20.sol RedemptionPair.sol

Revisions

Revision 1	c7331d4adbb4a95a7f00a50dd8dec7b1b7817b83
Revision 2	03fc8a9321a0d1ac07a86d24202ce4fb37593ad5

Imported Contracts

N/A	N/A

Appendix B - Risk Ratings

Risk	Description
High Risk	Security risks that are ***almost certain*** to lead to ***impairment or loss of funds***. Projects are advised to fix as soon as possible.
Medium Risk	Security risks that are ***very likely*** to lead to ***impairment or loss of funds*** with ***limited impact***. Projects are advised to fix as soon as possible.
Low Risk	Security risks that can lead to ***damage to the protocol***. Projects are advised to fix. Issues with this rating might be used in an exploit with other issues to cause significant damage.
Informational	Noteworthy information. Issues may include code conventions, missing or conflicting information, gas optimizations, and other advisories.

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 - Glossary

Contract Structure

Contract: An address with which provides functionality to users and other contracts. They are implemented in code and deployed to the blockchain.
Protocol: A system of contracts which work together.
Stakeholders: The users, operators, owners, and other participants of a contract.

Security Concepts

Bug**:** A defect in the contract code.
Exploit**:** A chain of events involving bugs, vulnerabilities, or other security risks which damages a protocol.
Funds: Tokens deposited by users or other stakeholders into a protocol.
Impairment: The loss of functionality in a contract or protocol.
Security risk: A circumstance that may result in harm to the stakeholders of a protocol. Examples include vulnerabilities in the code, bugs, excessive permissions, missing timelock, etc.
Vulnerability: A vulnerability is a flaw that allows an attacker to potentially cause harm to the stakeholders of a contract. They may occur in a contract's code, design, or deployed state on the blockchain.

Appendix E - 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