OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 54	2021-07-05	Zapmore, Donut	Audit Draft

Audit Notes

Audit Date	2021-06-22 - 2021-07-05
Auditor/Auditors	Donut, Plemonade, 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	OB586589214

Disclaimer

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

Obelisk Auditing

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

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

Table of Content

Project Information

Project Name	Bundle
Description	Bundle is a community-governed project offering full exposure to crypto risk-management and the DeFi ecosystem through passively managed, non-custodial funds and indices.
Website	https://bundledao.org/
Contact	@Zenith#2054
Contact information	@Zenith#2054 on Discord
Token Name(s)	Bundle
Token Short	BDL
Contract(s)	See Appendix A
Code Language	Solidity
Chain	BSC

Executive Summary

The audit of Bundle was conducted by three of Obelisks' security experts between the 22nd of June 2021 and the 5th of July 2021.

After finishing the full audit, Obelisk can safely state that there are some vulnerabilities that could cause issues to the project. These issues include all risk levels.

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

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

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

Summary Table

Audited Part	ID	Severity	Note
Using Cached Variable When Storage Should Be Used	#0001	Medium Risk	Mitigated
Calculate Spot Price	#0002	Low Risk	Mitigated
Reindexing Bundle Allows Unbinding Of All Tokens	#0003	High Risk	Mitigated
Duplicate Bundle Tokens Are Not Checked	#0004	Medium Risk	Mitigated
Non-Standard ERC20 Tokens Issue	#0005	Medium Risk	See Comment
Unbound Tokens Can Be Traded On Arbitrary Paths	#0006	Medium Risk	Mitigated
Owner Can Mint Tokens	#0007	Medium Risk	Mitigated
Bound Tokens May Be Traded On Arbitrary Paths	#0008	Low Risk	Mitigated
Setting Target Denorm Does Not Take Previous Updates Into Account	#0009	Low Risk	Mitigated
Newly Bound Token May Have Denorm Above Limit	#0010	Low Risk	Mitigated
Use Of tx.origin	#0011	Low Risk	See Comment
Unbound Loop	#0012	Low Risk	See Comment
Controller Owner Permissions	#0013	Low Risk	See Comment
Potential Reentrancy	#0014	Low Risk	Mitigated
Timestamp Can Be Manipulated To An Extent	#0015	Informational	See Comment
Equation In Comment Does Not Match Math	#0016	Informational	N/A
Keep Symbols And Value Names Consistent In Comments And Code	#0017	Informational	N/A

Token Migrations Issue	#0018	Informational	See Comment
Oracle Information	#0019	Informational	Mitigated
Modified Pancake Contracts	#0020	Informational	See Comment
Struct With The Same Name As Existing Contract	#0021	Informational	Mitigated
Gas Improvement For Bundle Metadata	#0022	Informational	Mitigated
Burn Does Not Remove Delegates	#0023	Informational	Mitigated
No Events Emitted For Changes To Protocol Values	#0024	Informational	Mitigated
Contract Values Should Be Constant	#0025	Informational	Mitigated
No Time Delay On Minting	#0026	Informational	See Comment
Potential re-entrancy In Minter emergencyWithdraw	#0027	Low Risk	Mitigated
Missing Zero Checks	#0028	Informational	N/A

Introduction

Obelisk was commissioned by Bundle on the 21st of June 2021 to conduct a comprehensive audit of Bundle's contract. The following audit was conducted between the 22nd of June 2021 and the 5th of July 2021 and delivered on the 5th of July 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 audit was conducted on contracts that were not yet live in a production environment. Issue #0001 & #0002 were uncovered during the initial onboarding process and were quickly fixed by the project team. Please refer to commit 1d03a676b36af23e84404e8e0e101c319d670883 for code references to the initial findings. A full analysis of the contract code was conducted on commit 2a2f9a63f4e84facf788ecc03a49b681239c8f99, which all other findings refer to. A comprehensive on-chain analysis will be conducted as the contracts are live in order to match the audited contracts with the published contracts.

There were multiple findings of all severity degrees found in the contracts which could cause problems.

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

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

Findings

Manual Analysis

Using Cached Variable When Storage Should Be Used

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0001
LOCATION	Bundle.sol -> 821-854

```
Record memory inRecord = _records[tokenIn];
        uint256 inRecordBalance = _getBalance(tokenIn);
        Record memory outRecord = _records[tokenOut];

        require(tokenAmountOut <= bmul(outRecord.balance, MAX_OUT_RATIO), "ERR_MAX_OUT_RATIO");

        uint256 spotPriceBefore = calcSpotPrice(
                                    inRecordBalance,
                                    inRecord.denorm,
                                    outRecord.balance,
                                    outRecord.denorm,
                                    _swapFee
                                );
        require(spotPriceBefore <= maxPrice, "ERR_BAD_LIMIT_PRICE");

        tokenAmountIn = calcInGivenOut(
                            inRecordBalance,
                            inRecord.denorm,
                            outRecord.balance,
                            outRecord.denorm,
                            tokenAmountOut,
                            _swapFee
                        );
        require(tokenAmountIn <= maxAmountIn, "ERR_LIMIT_IN");

        _pullUnderlying(tokenIn, msg.sender, tokenAmountIn);
        _pushUnderlying(tokenOut, msg.sender, tokenAmountOut);

        // Update tokens
        _updateToken(tokenIn, badd(inRecord.balance, tokenAmountOut));
        _updateToken(tokenIn, bsub(inRecord.balance, tokenAmountOut));

        inRecord.balance = badd(inRecord.balance, tokenAmountIn);
        outRecord.balance = bsub(outRecord.balance, tokenAmountOut);
```

DESCRIPTION	*inRecord* is a memory variable and updates to it such as for *inRecord.balance* on line 844 will not be saved to the chain after the function returns. Therefore *inRecord.balance* is never updated.
RECOMMENDATION	Obelisk recommends looking over the memory variables and making sure that they're not supposed to be storage variables.

MITIGATED/COMMENT	The dev has implemented the recommended fix in commit1 and found another bug while working on the fix commit2. commit1: https://github.com/bundle-dao/bundle-contract/commit/42a4ef2c0a6efee1a4c90f80cb5b5bfb0e02174d#diff-b4af8b523236255ac4709c48e3fdb02522f1544fa1c9ad43a27b9ecd6abf0a33 commit2: https://github.com/bundle-dao/bundle-contract/commit/f102091983319f78eced1192789a2f2572209dc3

Calculate Spot Price

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0002
LOCATION	Bundle.sol -> 856-862

```
spotPriceAfter = calcSpotPrice(
                            inRecordBalance,
                            inRecord.denorm,
                            outRecord.balance,
                            outRecord.denorm,
                            _swapFee
                        );
```

DESCRIPTION	The calculation of *spotPriceAfter* uses an old variable that hasn't been updated with the received tokens. This could lead to incorrect spot price values.
RECOMMENDATION	The value of *inRecordBalance* should be changed to use the correct variable.
MITIGATED/COMMENT	The dev has implemented the recommended fix. Refer to commit: 2a2f9a63f4e84facf788ecc03a49b681239c8f99.

Reindexing Bundle Allows Unbinding Of All Tokens

SEVERITY	High Risk
RESOLVED	**YES**
FINDING ID	#0003
LOCATION	Bundle.sol -> 429-478

```
1     function reindexTokens(
2         address[] calldata tokens,
3         uint256[] calldata targetDenorms,
4         uint256[] calldata minBalances
5     )
6         external override
7         _lock_
8         _control_
9     {
10        // ...
11    }
```

DESCRIPTION	There is no check to ensure that all tokens are not being unbound by the reindexing function. Several interactions require there to be at least 1, if not 2 tokens within the bundle. The removal of all tokens from the bundle can have a variety of security implications. Potential adverse interactions include the ability for anyone to print any amount of shares and the potential irrecoverable loss of deposited funds. Additional care should be taken if a pool is re-indexing all tokens to a completely new set of tokens as this may result in a state of temporary vulnerability.

RECOMMENDATION	Track all the tokens which are to be unbound to ensure that the MIN_BOUND_TOKENS constraint is maintained.
MITIGATED/COMMENT	The project team has implemented the recommendation. Refer to commit: f586ae7bb8c95b09af48fd04ee44b6df59b81e54

Duplicate Bundle Tokens Are Not Checked

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0004
LOCATION	Bundle.sol -> 140-164

```
uint256 totalWeight = 0;
    for (uint256 i = 0; i < tokens.length; i++) {
        uint256 denorm = denorms[i];
        uint256 balance = balances[i];

        require(denorm >= MIN_WEIGHT && denorm <= MAX_WEIGHT, "ERR_BAD_WEIGHT");
        require(balance >= MIN_BALANCE, "ERR_MIN_BALANCE");

        address token = tokens[i];
        _records[token] = Record({
            bound: true,
            ready: true,
            denorm: denorm,
            targetDenorm: denorm,
            targetTime: 0,
            lastUpdateTime: 0,
            index: uint8(i),
            balance: balance
        });

        _tokens.push(token);
        totalWeight = badd(totalWeight, denorm);
        // Move underlying asset to pool
        _pullUnderlying(token, tokenProvider, balance);
    }
```

DESCRIPTION	While setting up a Bundle, there is no check for duplicate tokens. This results in an incorrect *totalWeight* and additional transfers from the tokenProvider. It also results in unexpected interactions in other function calls. For example, joining the pool will require multiple contributions of the same token. Exiting the pool can potentially drain the pool.
RECOMMENDATION	Add a check to make sure that a token cannot be bound twice in the setup.
MITIGATED/COMMENT	The project team has implemented the recommendation. Refer to commit: a6edbe9bd3e2364d2f31bc8ce0dd9f4040873363

Non-Standard ERC20 Tokens Issue

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0005
LOCATION	Bundle.sol
DESCRIPTION	Non-standard erc20 tokens such as deflationary tokens could cause issues with the swapping mechanism as the internal records are changed before the transfer of funds. This was the cause of Balancers STA exploit.
RECOMMENDATION	Make sure to inform users of the risk of using a pool with non-conforming erc20 tokens and or make sure that the non-standard tokens are safe enough to be approved for a pool.
MITIGATED/COMMENT	The project team has acknowledged the issue and they will not be supporting non standard erc20 tokens as underlying assets for Bundles(pools).

Unbound Tokens Can Be Traded On Arbitrary Paths

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0006
LOCATION	Unbinder.sol -> 159-176

```
1     function _handlePath(address[] calldata path, uint256 amountIn, uint256 amountOut, uint256
  deadline)
2         internal
3     {
4             require(amountOut > 0, "ERR_BAD_SWAP");
5
6             // Min amount out to be 99% of expectation
7             // unbinder used infrequently enough s.t. these don't need to be too strict
8             _router.swapExactTokensForTokens(
9                 amountIn,
10                amountOut.mul(9900).div(10000),
11                path,
12                address(_bundle),
13                deadline
14            );
15
16            // Add bToken back to balances
17            _bundle.gulp(path[path.length - 1]);
18    }
```

DESCRIPTION	Tokens recently unbound, as well as other unbound gulped tokens, can be directed towards arbitrary trade paths.
	
	Suppose a token *Gulped* has just been unbound. A malicious actor may deploy a token *Attack* that restricts transfers to themselves, the unbinder, the router, and trading pairs. They may then set up arbitrary exchange rates resulting in the majority of *Gulped* being caught in the trade path while a tiny amount of the value is returned to the bundle.
	
	For example, if the bundle consisted of 3 tokens (*A*, *B,* and *Gulped*), the trading pairs could be constructed with the ratios:
	
	`Gulped : Attack :: 1 : 1`
	`Attack : A :: 1000 : 1`
	`Attack : B :: 1000 : 1`
	
	If tokens in the bundle have weights close to the min

	weight, they may be unbound with very short notice. This can create a substantial loss of value for staked users.
RECOMMENDATION	Restrict the path to a set of highly liquid intermediate tokens. Most exchanges will have a set of base tokens that other tokens will typically pair with.
MITIGATED/COMMENT	The project team has implemented the recommendation. Refer to commit: 3169e8c043f4cfe683cf95fd31eb72c44cbb3cfc Event emits were added in this commit: d9ffe2f9f8667cd2163af5552f3bfd88389f2004

Owner Can Mint Tokens

SEVERITY	Medium Risk
RESOLVED	**Yes**
FINDING ID	#0007
LOCATION	BundleToken.sol -> 61-71

```
function manualMint(address _to, uint256 _amount) public onlyOwner {
    require(manualMinted.add(_amount) <= MANUAL_MINT_LIMIT, "mint limit exceeded");
    manualMinted = manualMinted.add(_amount);
    mint(_to, _amount);
}

function mint(address _to, uint256 _amount) public onlyOwner {
    require(totalSupply().add(_amount) <= cap(), "cap exceeded");
    _mint(_to, _amount);
    _moveDelegates(address(0), _delegates[_to], _amount);
}
```

LOCATION	Minter.sol -> 157-159

```
function manualMint(address _to, uint256 _amount) external onlyOwner {
  bundle.manualMint(_to, _amount);
}
```

LOCATION	MinterGuard.sol -> 48-54

```
function mintWarchest(address _to, uint256 _amount) external onlyOwner {
    require(mintCount.add(_amount) <= mintLimit, "MinterGuard::mintWarchest:: mint exceeded mintLimit");
    require(_amount <= individualMintLimit, "MinterGuard::mintWarchest:: mint exceeded individualMintLimit");
    minter.manualMint(_to, _amount);
    mintCount = mintCount.add(_amount);
    emit MintWarchest(_to, _amount);
}
```

DESCRIPTION	The contract owner is able to mint up to *MANUAL_MINT_LIMIT* tokens (10.5 million). As a result, a malicious actor as the contract owner may be able to inflate the supply at an opportunistic moment and remove a significant amount of value from any trading pair.
RECOMMENDATION	Specify when these functions will be used and incorporate that into the logic of the contract. Add a timelock to allow users to react to newly minted supply.
MITIGATED/COMMENT	The project team has already deployed a 2 day Timelock.

Bound Tokens May Be Traded On Arbitrary Paths

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0008
LOCATION	Rebalancer.sol -> 173-178

```
1   // Swap on Bundle w/ absolute min out and max price
2         IERC20Upgradeable(tokenIn).safeTransferFrom(msg.sender, address(this), amountIn);
3         (uint256 tokenAmountOut, ) = IBundle(pool).swapExactAmountIn(tokenIn, amountIn, tokenOut, 0,
  type(uint256).max);
4
5         // Swap tokenOut tokenIn on separate DEX
6         uint256[] memory amountsTokenOut = _router.swapExactTokensForTokens(tokenAmountOut,
  amountIn, path, address(this), deadline);
```

DESCRIPTION	Similar to the Unbinder, the Rebalancer allows tokens to be traded on the separate DEX using an arbitrary path. As a result, routing trades through tokens entirely under the control of a malicious actor will allow them to bypass contributing to the pool on re-balancing. This issue is not as severe in Unbinder as the Bundle contract ensures that staked value is retained.
RECOMMENDATION	Restrict the path to a set of highly liquid intermediate tokens. Most exchanges will have a set of base tokens that other tokens will typically pair with.
MITIGATED/COMMENT	Oracle ensures that the tokens from the external DEX are of similar value. Additionally, a swap whitelist was added as recommended. Refer to commit: d9ffe2f9f8667cd2163af5552f3bfd88389f2004

Setting Target Denorm Does Not Take Previous Updates Into Account

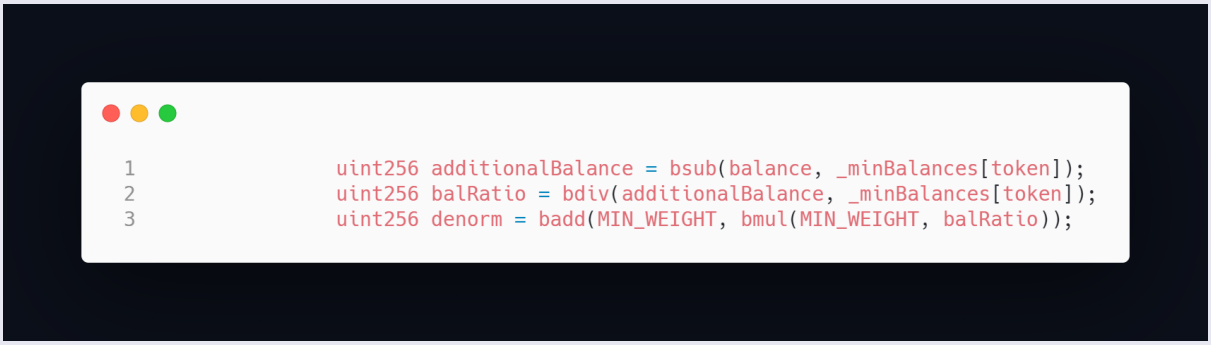
SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0009
LOCATION	Bundle.sol -> 555-564

```
1     function _setTargetDenorm(address token, uint256 denorm)
2         internal
3     {
4         require(_records[token].bound, "ERR_NOT_BOUND");
5         require(denorm >= MIN_WEIGHT || denorm == 0, "ERR_MIN_WEIGHT");
6         require(denorm <= MAX_WEIGHT, "ERR_MAX_WEIGHT");
7         _records[token].targetDenorm = denorm;
8         _records[token].targetTime = badd(block.timestamp, _targetDelta);
9         _records[token].lastUpdateTime = block.timestamp;
10    }
```

DESCRIPTION	The setting *targetDenorm* will negate any expected updates to denorm which are pending since the *lastUpdateTime*. For example, consider if the denorm is set to change from 1 to 3 over two days, and after 1 day, the target denorm is updated to be 6 over the next two days (ie. *_targetDelta* is two days). The expected behavior would be that the denorm would increase to 2 over the first day, then it would increase by 2 per day for the next two days. After the first day, when the new target is set, the denorm would instead be updated to 2.33.
RECOMMENDATION	Include an update to the denorm before updating the record. Alternatively. call *_updateDenorm* frequently such that the expected and actual values do not move out of sync.
MITIGATED/COMMENT	The project team has implemented the recommendation. Refer to commit: 646dbeb9faf31e3f8cb4b8bc6f8bfa747df5d0fa

Newly Bound Token May Have Denorm Above Limit

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0010
LOCATION	Bundle.sol -> 614-616



```
1    uint256 additionalBalance = bsub(balance, _minBalances[token]);
2    uint256 balRatio = bdiv(additionalBalance, _minBalances[token]);
3    uint256 denorm = badd(MIN_WEIGHT, bmul(MIN_WEIGHT, balRatio));
```

DESCRIPTION	After binding a new token, if a very large amount of tokens is added (eg. via *token.transfer* then *bundle.gulp*), the denorm may exceed *MAX_WEIGHT*. Note: After some time and contract interactions, the denorm will return to the expected range.
RECOMMENDATION	Add a limit to the denorm set here.
MITIGATED/COMMENT	The project team has implemented the recommendation. Refer to commit: 7733b7855320801ecb700532580ad04ad943934b

Use Of tx.origin

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0011
LOCATION	Rebalancer.sol -> 44-47 Unbinder.sol -> 45-48

```
1    modifier _eoa_() {
2        require(msg.sender == tx.origin, "ERR_NOT_EOA");
3        _;
4    }
```

DESCRIPTION	Only an EOA (externally Owned address) can use the *Rebalancer.swap* and *Unbinder.distributeUnboundToken* functions. This prevents smart contracts from calling but also prevents Multi-sig wallets like gnosis safe from calling it. Be aware that tx.origin might not stay useful, Vitalik: "Do NOT assume that tx.origin will continue to be usable or meaningful."
RECOMMENDATION	Be aware of the downsides of using *tx.origin* and watch out for changes concerning *tx.origin*. Use OpenZeppelin *Address.isContract* to check for contracts.
MITIGATED/COMMENT	After discussions around the risks the project team has decided to keep using tx.origin for the near-term and utilize the upgradability of their contracts to adapt to any changes made in the future around tx.origin.

Unbound Loop

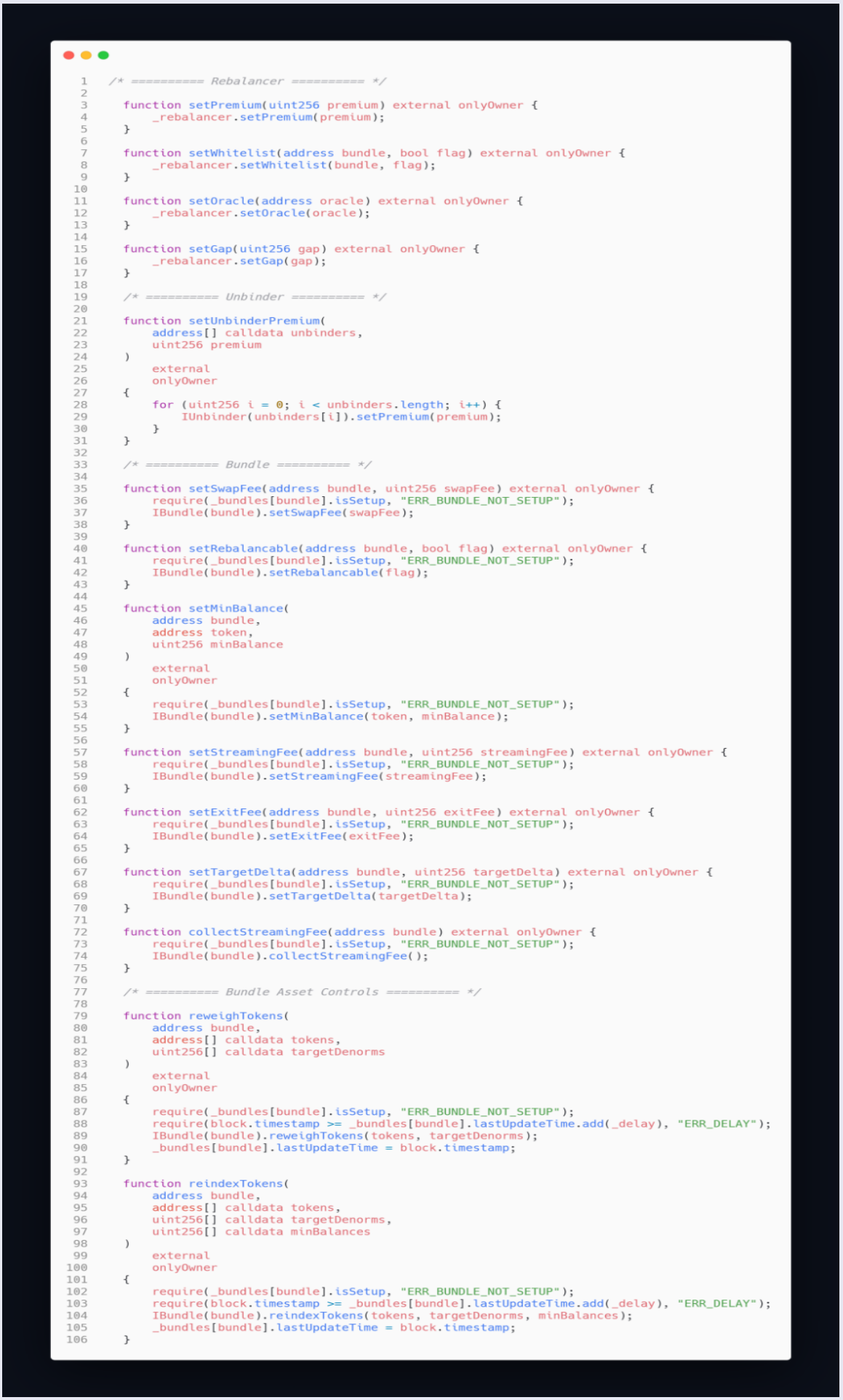
SEVERITY	Low Risk
RESOLVED	**PARTIALLY**
FINDING ID	#0012
LOCATION	BundleLock.sol -> 45-54

```
function getTier(address user)
    public view override
    returns (uint256)
{
    for (uint256 i = 0; i < _tiers.length; i++) {
        if (_bundleBalances[user] >= _tiers[_tiers.length - 1 - i]) {
            return _tiers.length - 1 - i;
        }
    }
}
```

DESCRIPTION	Transactions calling *getTier* may revert on maximum gas if the number of tiers is too large. As a view function, this will impact only calls from other contracts.
RECOMMENDATION	Do not set too many tiers. It is recommended to specify a maximum number of tiers.
MITIGATED/COMMENT	The project team:“BundleLock.sol was an experimental contract that has since been abandoned. It will not be deployed to mainnet.”

Controller Owner Permissions

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0013
LOCATION	Controller.sol -> 112-217



```
/* ========== Rebalancer ========== */

    function setPremium(uint256 premium) external onlyOwner {
        _rebalancer.setPremium(premium);
    }

    function setWhitelist(address bundle, bool flag) external onlyOwner {
        _rebalancer.setWhitelist(bundle, flag);
    }

    function setOracle(address oracle) external onlyOwner {
        _rebalancer.setOracle(oracle);
    }

    function setGap(uint256 gap) external onlyOwner {
        _rebalancer.setGap(gap);
    }

    /* ========== Unbinder ========== */

    function setUnbinderPremium(
        address[] calldata unbinders,
        uint256 premium
    )
        external
        onlyOwner
    {
        for (uint256 i = 0; i < unbinders.length; i++) {
            IUnbinder(unbinders[i]).setPremium(premium);
        }
    }

    /* ========== Bundle ========== */

    function setSwapFee(address bundle, uint256 swapFee) external onlyOwner {
        require(_bundles[bundle].isSetup, "ERR_BUNDLE_NOT_SETUP");
        IBundle(bundle).setSwapFee(swapFee);
    }

    function setRebalancable(address bundle, bool flag) external onlyOwner {
        require(_bundles[bundle].isSetup, "ERR_BUNDLE_NOT_SETUP");
        IBundle(bundle).setRebalancable(flag);
    }

    function setMinBalance(
        address bundle,
        address token,
        uint256 minBalance
    )
        external
        onlyOwner
    {
        require(_bundles[bundle].isSetup, "ERR_BUNDLE_NOT_SETUP");
        IBundle(bundle).setMinBalance(token, minBalance);
    }

    function setStreamingFee(address bundle, uint256 streamingFee) external onlyOwner {
        require(_bundles[bundle].isSetup, "ERR_BUNDLE_NOT_SETUP");
        IBundle(bundle).setStreamingFee(streamingFee);
    }

    function setExitFee(address bundle, uint256 exitFee) external onlyOwner {
        require(_bundles[bundle].isSetup, "ERR_BUNDLE_NOT_SETUP");
        IBundle(bundle).setExitFee(exitFee);
    }

    function setTargetDelta(address bundle, uint256 targetDelta) external onlyOwner {
        require(_bundles[bundle].isSetup, "ERR_BUNDLE_NOT_SETUP");
        IBundle(bundle).setTargetDelta(targetDelta);
    }

    function collectStreamingFee(address bundle) external onlyOwner {
        require(_bundles[bundle].isSetup, "ERR_BUNDLE_NOT_SETUP");
        IBundle(bundle).collectStreamingFee();
    }

    /* ========== Bundle Asset Controls ========== */

    function reweighTokens(
        address bundle,
        address[] calldata tokens,
        uint256[] calldata targetDenorms
    )
        external
        onlyOwner
    {
        require(_bundles[bundle].isSetup, "ERR_BUNDLE_NOT_SETUP");
        require(block.timestamp >= _bundles[bundle].lastUpdateTime.add(_delay), "ERR_DELAY");
        IBundle(bundle).reweighTokens(tokens, targetDenorms);
        _bundles[bundle].lastUpdateTime = block.timestamp;
    }

    function reindexTokens(
        address bundle,
        address[] calldata tokens,
        uint256[] calldata targetDenorms,
        uint256[] calldata minBalances
    )
        external
        onlyOwner
    {
        require(_bundles[bundle].isSetup, "ERR_BUNDLE_NOT_SETUP");
        require(block.timestamp >= _bundles[bundle].lastUpdateTime.add(_delay), "ERR_DELAY");
        IBundle(bundle).reindexTokens(tokens, targetDenorms, minBalances);
        _bundles[bundle].lastUpdateTime = block.timestamp;
    }
```

DESCRIPTION	The owner address of the controller has many permissions that could impact users like changing the fees or the denorms of tokens in bundles. Note that most functions have a hard limit on the range of allowable values such that the owner address can't maliciously manipulate the users.
RECOMMENDATION	Have the Owner address under a timelock/DAO vote and limit the scope of permissible functions to a minimum if possible.
MITIGATED/COMMENT	The project team will have the controller be behind a 2 day timelock and eventually transfer it to the DAO governance to control.

Potential Reentrancy

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0014
LOCATION	Minter.sol -> 286-295

```
1   // Withdraw without caring about rewards. EMERGENCY ONLY.
2   function emergencyWithdraw(uint256 _pid) external nonReentrant {
3     PoolInfo storage pool = poolInfo[_pid];
4     UserInfo storage user = userInfo[_pid][msg.sender];
5     IERC20(pool.stakeToken).safeTransfer(address(msg.sender), user.amount);
6     emit EmergencyWithdraw(msg.sender, _pid, user.amount);
7     user.amount = 0;
8     user.rewardDebt = 0;
9     user.bonusDebt = 0;
10  }
```

DESCRIPTION	If the underlying token is vulnerable to reentrancy then this function could be called over and over before setting the user balance to 0.
RECOMMENDATION	Apply Checks Effects Interactions to limit the attack vector of reentrancy.
MITIGATED/COMMENT	This issue can't happen as Openzeppelin Reentrancy guard is used but applying checks-effects-interactions pattern would be better(reentrancy would be allowed without causing issues). As the project team has already deployed they acknowledge the issue and deemed it ok to leave.

Timestamp Can Be Manipulated To An Extent

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0015
LOCATION	Bundle.sol -> 227-254

```
function collectStreamingFee()
        external override
        _lock_
        _control_
    {
        require(_tokens.length > 0, "ERR_SETUP");
        require(_lastStreamingTime < block.timestamp, "ERR_COLLECTION_TO_SOON");

        uint256 timeDelta = bsub(block.timestamp, _lastStreamingTime);

        for (uint256 i = 0; i < _tokens.length; i++) {
            address token = _tokens[i];

            // Shouldnt withdraw tokens if not ready
            if (_records[token].ready) {
                uint256 fee = bdiv(
                    bmul(bmul(_records[token].balance, _streamingFee), timeDelta),
                    BPY
                );

                _pushUnderlying(token, _controller, fee);
                _updateToken(token, bsub(_records[token].balance, fee));
            }
        }

        _lastStreamingTime = block.timestamp;
        emit LogStreamingFee(msg.sender);
    }
```

DESCRIPTION	The value of *block.timestamp* can be manipulated by the miner/validator to a certain extent. Ensure that the small manipulation of timedelta won't cause any issues.
RECOMMENDATION	Consider using *block.number* if possible as it's more consistent. Although it's possible that the length of a block might change in the future and it would require code changes on different blockchains and it might be off timewise if miners/validators are taking longer/shorter than expected.
MITIGATED/COMMENT	The project team acknowledged the risks and has decided not to change anything due to potential changes to block time.

Equation In Comment Does Not Match Math

SEVERITY	Informational
RESOLVED	**NO**
FINDING ID	#0016
LOCATION	BMath.sol -> 143-152

```
/*****************************************************************************************
// calcSingleInGivenPoolOut                                                             //
// tAi = tokenAmountIn             //(pS + pAo)\     /    1    \\                      //
// pS = poolSupply                || ---------  | ^ | --------- || * bI - bI           //
// pAo = poolAmountOut             \\    pS    /     \(wI / tW)//                      //
// bI = balanceIn          tAi =  ------------------------------------------           //
// wI = weightIn                            /      wI  \                               //
// tW = totalWeight                        |  1 - ----  |  * sF                         //
// sF = swapFee                             \      tW  /                               //
*****************************************************************************************/
```

DESCRIPTION	The denominator of the formula is described as (1 - wI/tW) * sF. This is incorrect and not a full inversion of the formula for *calcPoolOutGivenSingleIn*. The correct denominator is (1 - (1 - wI/tW) * sF). The calculation used in the code is correct.
RECOMMENDATION	Ensure the formulas listed are both correct and match the code.
MITIGATED/COMMENT	Acknowledged by project team but not fixed due to time constraints and to avoid changing the balancer core math contracts.

Keep Symbols And Value Names Consistent In Comments And Code

SEVERITY	Informational
RESOLVED	**NO**
FINDING ID	#0017
LOCATION	BMath.sol

DESCRIPTION	A variety of values are described in the math contract using slightly different symbols and names in varying locations.
	
	For example, *bI* is a common symbol for *tokenBalanceIn*. However, on line 110, the symbol used is *tBi*. Elsewhere on line 148, it is called *balanceIn*.
	
	Sometimes the inconsistencies are within the same comment, such as for *weightTokenIn* and *weightTi* on lines 127-128:
	
	`// That proportion is (1- weightTokenIn)`
	`// tokenAiAfterFee = tAi * (1 - (1-weightTi) * poolFee);`
	
	This makes it difficult to follow the reasoning in the math.

RECOMMENDATION	Ensure symbols and names are consistent.
MITIGATED/COMMENT	Acknowledged by project team but not fixed due to time constraints and to avoid changing the balancer core math contracts.

Token Migrations Issue

SEVERITY	Informational
RESOLVED	**PARTIALLY**
FINDING ID	#0018
LOCATION	Bundle.sol
DESCRIPTION	There is no function to handle new token migrations which could lead to loss of funds depending on how a token is migrated. For example, claiming new tokens would not be possible for the pool. While airdropping then calling gulp would transfer the tokens to the unbinder some value might still be lost for other tokens from swaps from it.
RECOMMENDATION	Make sure to inform users of the risks and or consider making functions to handle these issues. Make sure to rigorously test any potential migration-related functions.
MITIGATED/COMMENT	The project team acknowledges the risks of a token migration and will try to carefully monitor underlying projects and their roadmaps to mitigate the risks of such an event happening.

Oracle Information

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0019
LOCATION	PriceOracle.sol -> 25

```
1     uint256 public constant PERIOD = 1 hours;
```

DESCRIPTION	When using an on-chain single oracle two important factors are (Accuracy, Security). Increasing the time will raise the security but lower the accuracy. Decreasing time will lower security but raise accuracy.
RECOMMENDATION	Make sure that the loss of accuracy does not cause too much negative(in terms of value lost for the protocol) arbitrage and the security is enough to not have the pool manipulated.
MITIGATED/COMMENT	The project team agreed that 1 hour would probably cause a substantial amount of accuracy loss and as the oracle is only used to make sure funds aren't skimmed on the way through the trading paths lowering it would most likely make it more secure. Thus the project team changed it to 15 min. Refer to commit: 46443a8854a320c88a6f07390a4d5b92ff96462f

Modified Pancake Contracts

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0020
LOCATION	PriceOracle.sol -> 6-7

```
1 import "@bundle-dao/pancakeswap-peripheral/contracts/libraries/PancakeOracleLibrary.sol";
2 import "@bundle-dao/pancakeswap-peripheral/contracts/libraries/PancakeLibrary.sol";
```

LOCATION	PriceOracle.sol -> 127-147

```
1     function initializePair(address tokenA, address tokenB)
2         public override
3     {
4         require(address(tokenA) != address(0) && address(tokenB) != address(0), "ERR_BAD_ADDRESS");
5
6         IPancakePair pair = IPancakePair(PancakeLibrary.pairFor(_factory, tokenA, tokenB));
7
8         if (!_prices[address(pair)].initialized) {
9             (uint256 reserve0, uint256 reserve1, uint32 blockTimestampLast) = pair.getReserves();
10            require(reserve0 != 0 && reserve1 != 0, "ERR_NO_RESERVES"); // ensure that there's
  liquidity in the pair
11
12            _prices[address(pair)] = PriceData({
13                initialized: true,
14                lastUpdatedAt: blockTimestampLast,
15                price0CumulativeLast: pair.price0CumulativeLast(),
16                price1CumulativeLast: pair.price0CumulativeLast(),
17                price0Average: FixedPoint.uq112x112(0),
18                price1Average: FixedPoint.uq112x112(0)
19            });
20        }
21    }
```

DESCRIPTION	Bundle uses an internal copy of the pancake contracts where the solidity version was increased to match. The *pairFor* function was modified in this internal version to only return existing pair addresses. However, this will return the null address on uninitialized pairs. This will reverse an initialization transaction.
RECOMMENDATION	Please clarify the purpose behind changing the pancake library contract.
MITIGATED/COMMENT	Acknowledged by project team.

Project team comment: The issue was, as I understand, that the init code hash is ambiguous (difficult to find given these libraries are all seriously out of date / poorly maintained) so it was easier to just check from the factory.

Struct With The Same Name As Existing Contract

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0021
LOCATION	Controller.sol -> 26-31

```
1    struct Bundle {
2        address unbinder;
3        bool    isInitialized;
4        bool    isSetup;
5        uint256 lastUpdateTime;
6    }
```

LOCATION	Minter.sol -> 43

```
1   BundleToken public bundle;
```

DESCRIPTION	The *Bundle* struct used in Controller is used to keep track of information about deployed bundles. However, this name clashes with the actual Bundle contract which is deployed by the factory. Though there are no issues at present, it may present a challenge when updating the code in the future. In Minter.sol, the BundleToken is called *bundle*.
RECOMMENDATION	Rename the *Bundle* struct to clarify intent.
MITIGATED/COMMENT	The project team has implemented the recommendation. Refer to commit: 830bffa69af38890ff1b38d642993fa868ee5057

Gas Improvement For Bundle Metadata

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0022
LOCATION	Controller.sol -> 221-239

```
function getBundleMetadata(
    address bundle
)
    external view
    returns (
        address unbinder,
        bool isInitialized,
        bool isSetup,
        uint256 lastUpdateTime
    )
{
    Bundle memory metadata = _bundles[bundle];
    return (
        metadata.unbinder,
        metadata.isInitialized,
        metadata.isSetup,
        metadata.lastUpdateTime
    );
}
```

DESCRIPTION	Copying from storage to memory will require more gas than referencing(storage) when using a getter.
RECOMMENDATION	Change the metadata variable to storage.
MITIGATED/COMMENT	The project team skipped the variable directly, saving some more gas. Refer to commit: 94a4c0cded92b15f0e9842a123ca74dcb690ca74

Burn Does Not Remove Delegates

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0023
LOCATION	BundleToken.sol -> 73-75

```
1    function burn(address _account, uint256 _amount) external onlyOwner {
2        _burn(_account, _amount);
3    }
```

DESCRIPTION	Transferring and minting update the delegate counts to match. However, burning the token does not mean that the delegates will remain permanently.
RECOMMENDATION	Please confirm that this behavior is intended.
MITIGATED/COMMENT	The project team has acknowledged this is intended behavior.

No Events Emitted For Changes To Protocol Values

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0024
LOCATION	Bundle.sol -> 195-225

```
function setMinBalance(address token, uint256 minBalance)
        external override
        _control_
    {
        require(_records[token].bound && !_records[token].ready, "ERR_BAD_TOKEN");
        _minBalances[token] = minBalance;
    }

    function setStreamingFee(uint256 streamingFee)
        external override
        _control_
    {
        require(streamingFee < MAX_STREAMING_FEE, "ERR_MAX_STREAMING_FEE");
        _streamingFee = streamingFee;
    }

    function setExitFee(uint256 exitFee)
        external override
        _control_
    {
        require(exitFee < MAX_EXIT_FEE, "ERR_MAX_STREAMING_FEE");
        _exitFee = exitFee;
    }

    function setTargetDelta(uint256 targetDelta)
        external override
        _control_
    {
        require(targetDelta >= MIN_TARGET_DELTA && targetDelta <= MAX_TARGET_DELTA,
"ERR_TARGET_DELTA");
        _targetDelta = targetDelta;
    }
```

LOCATION	Rebalancer.sol -> 64-91

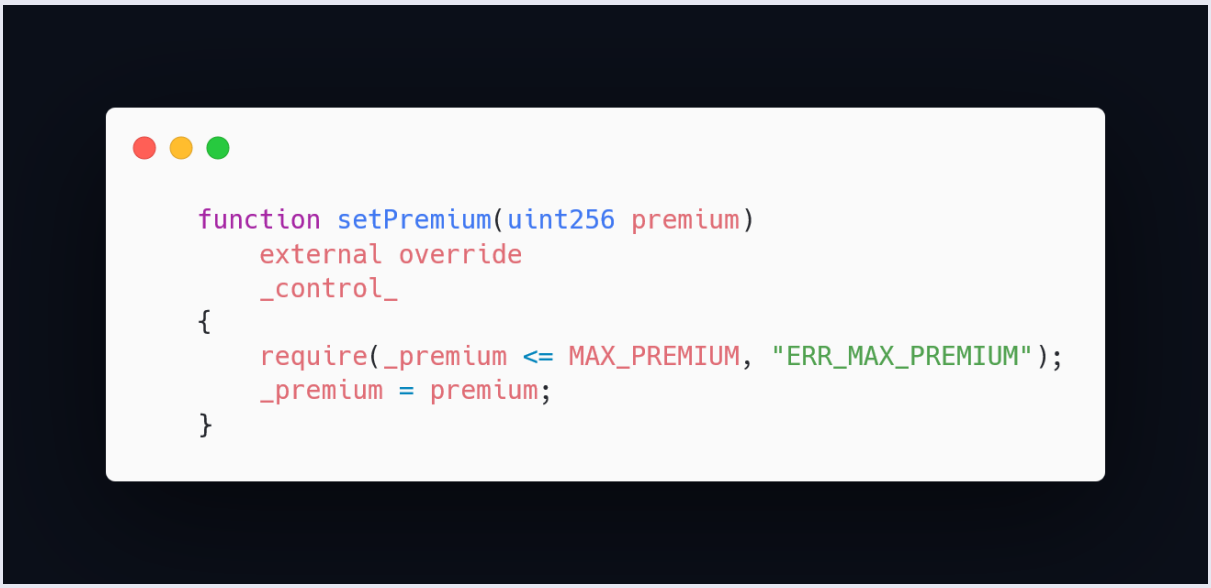
```
function setPremium(uint256 premium)
        external override
        _control_
    {
        require(_premium <= MAX_PREMIUM, "ERR_MAX_PREMIUM");
        _premium = premium;
    }

    function setWhitelist(address pool, bool flag)
        external override
        _control_
    {
        _poolAuth[pool] = flag;
    }

    function setOracle(address oracle)
        external override
        _control_
    {
        _oracle = IPriceOracle(oracle);
    }

    function setGap(uint256 gap)
        external override
        _control_
    {
        _gap = gap;
    }
```

LOCATION	Unbinder.sol -> 65-71



```
function setPremium(uint256 premium)
    external override
    _control_
{
    require(_premium <= MAX_PREMIUM, "ERR_MAX_PREMIUM");
    _premium = premium;
}
```

DESCRIPTION	Functions that change important variables should include emit logs such that users can more easily monitor the change.
RECOMMENDATION	Add emit logs to these functions.
MITIGATED/COMMENT	The project team has implemented the recommendation. Refer to commit for Bundle and Rebalancer: 5451e153dfb68ca57efc4051b43e00997af9d616 Refer to commit for Unbinder: caf66bff307764b4afb4ff0390d90a2cece01b97

Contract Values Should Be Constant

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0025
LOCATION	MinterGuard.sol -> 14-16

```
1   uint256 public mintLimit = 10500000e18;
2   uint256 public mintCount = 250000e18;
3   uint256 public individualMintLimit = 500000e18;
```

DESCRIPTION	Variables that are supposed to be *constant* or *immutable* should be marked as such to reduce gas cost and give readers a better understanding
RECOMMENDATION	Mark these variables with the constant keyword.
MITIGATED/COMMENT	The project team has implemented the recommendation. Refer to commit: ea4e5d525d9a6253f838c2272588c25a7abbad2d

No Time Delay On Minting

SEVERITY	Low Risk
RESOLVED	**PARTIALLY**
FINDING ID	#0026
LOCATION	MinterGuard.sol -> 48-54

```
1  function mintWarchest(address _to, uint256 _amount) external onlyOwner {
2    require(mintCount.add(_amount) <= mintLimit, "MinterGuard::mintWarchest:: mint exceeded
   mintLimit");
3    require(_amount <= individualMintLimit, "MinterGuard::mintWarchest:: mint exceeded
   individualMintLimit");
4    minter.manualMint(_to, _amount);
5    mintCount = mintCount.add(_amount);
6    emit MintWarchest(_to, _amount);
7  }
```

DESCRIPTION	Minter guard includes a limit on the amount mintable per manual mint call. However, this does not prevent multiple calls to mint in short succession.
RECOMMENDATION	Add a time delay to manual minting in minter guard, or use a time lock that prevents quick successive calls.
MITIGATED/COMMENT	The project team has decided not to change live deployed code. The minting is still behind a 2 days timelock even if successive calls aren't stopped so the extra logic might give users a heads up if multiple timelock calls are scheduled at once.

Static Analysis

Potential re-entrancy In Minter emergencyWithdraw

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0027
LOCATION	Minter.sol -> 287-295

```
function emergencyWithdraw(uint256 _pid) external nonReentrant {
  PoolInfo storage pool = poolInfo[_pid];
  UserInfo storage user = userInfo[_pid][msg.sender];
  IERC20(pool.stakeToken).safeTransfer(address(msg.sender), user.amount);
  emit EmergencyWithdraw(msg.sender, _pid, user.amount);
  user.amount = 0;
  user.rewardDebt = 0;
  user.bonusDebt = 0;
}
```

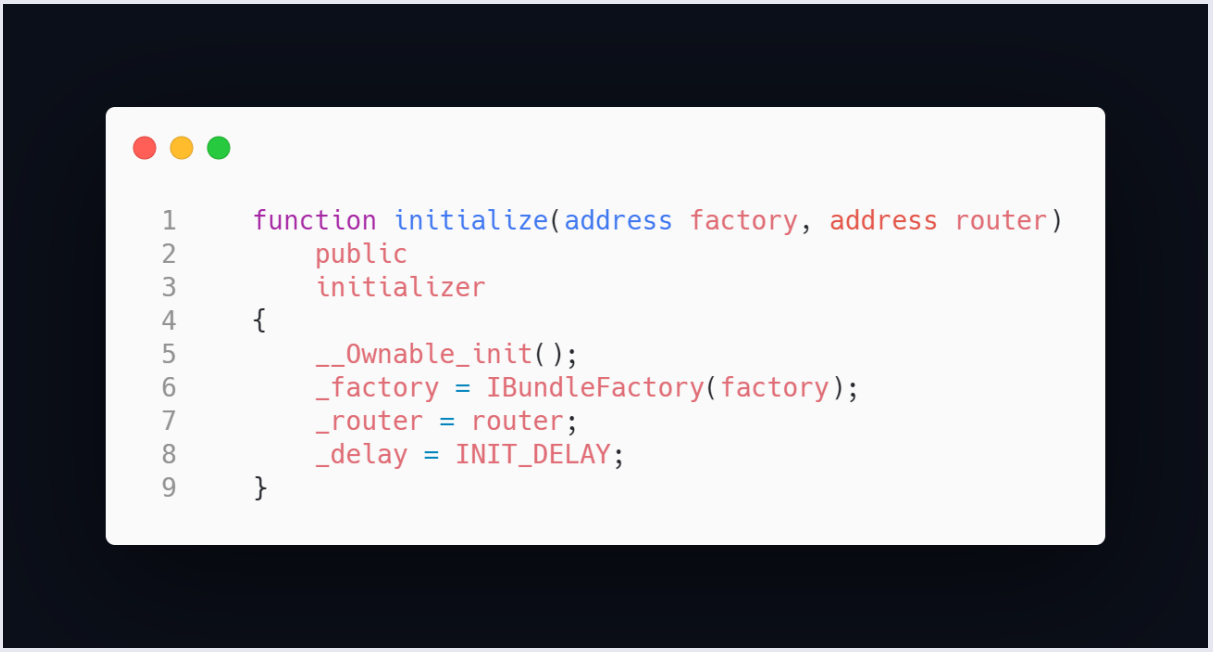
DESCRIPTION	Tokens susceptible to re-entrancy attacks may allow a malicious actor to drain tokens from the bundle by recursively calling emergency withdraw.
RECOMMENDATION	Use the checks-effects-interactions pattern.
MITIGATED/COMMENT	This issue can't happen as Openzeppelin Reentrancy guard is used but applying checks-effects-interactions pattern would be better(reentrancy would be allowed without causing issues).As the project team has already deployed they acknowledge the issue and deemed it ok to leave.

Missing Zero Checks

SEVERITY	Informational
RESOLVED	**NO**
FINDING ID	#0028
LOCATION	BundleFactory.sol -> 34-40

```
1     constructor(
2         address unbinderBeacon,
3         address bundleBeacon
4     ) public {
5         _unbinderBeacon = unbinderBeacon;
6         _bundleBeacon = bundleBeacon;
7     }
```

LOCATION	Controller.sol -> 43-51



```
1     function initialize(address factory, address router)
2         public
3         initializer
4     {
5         __Ownable_init();
6         _factory = IBundleFactory(factory);
7         _router = router;
8         _delay = INIT_DELAY;
9     }
```

LOCATION	Rebalancer.sol -> 51-60

```
function initialize(address router, address controller)
    public
    initializer
{
    __ReentrancyGuard_init();
    _controller = controller;
    _router = IPancakeRouter02(router);
    _premium = INIT_PREMIUM;
    _gap = INIT_ORACLE_GAP;
}
```

LOCATION	Unbinder.sol -> 52-61

```
function initialize(address bundle, address router, address controller)
    public override
    initializer
{
    __ReentrancyGuard_init();
    _bundle = IBundle(bundle);
    _router = IPancakeRouter02(router);
    _controller = controller;
    _premium = INIT_PREMIUM;
}
```

LOCATION	PriceOracle.sol -> 33-36

```
1    constructor(address factory, address peg) public {
2         _factory = factory;
3         _peg = peg;
4     }
```

LOCATION	Minter.sol -> 70-89

```
1   constructor(
2     BundleToken _bundle,
3     address _devaddr,
4     uint256 _blockRewards,
5     uint256 _startBlock
6   ) public {
7     bonusMultiplier = 0;
8     totalAllocPoint = 0;
9     bonusEndBlock = 0;
10    bundle = _bundle;
11    devaddr = _devaddr;
12    blockRewards = _blockRewards;
13    startBlock = _startBlock;
14  }
15
16  // Update dev address by the previous dev.
17  function setDev(address _devaddr) public {
18    require(msg.sender == devaddr, "setDev: caller not dev");
19    devaddr = _devaddr;
20  }
```

LOCATION	Timelock.sol -> 39-46

```
constructor(address factory, address peg) public {
    _factory = factory;
    _peg = peg;
}
```

DESCRIPTION	The contract address values can be set to zero address in various constructors, initializers, and setter functions. Zero addresses may cause incorrect contract behavior. In some cases, incorrect zero values may be corrected, such as in Minter. However, in others, such as the constructors and initializers, this may not be possible.
RECOMMENDATION	Add a check to ensure contract values are never set to invalid addresses.
MITIGATED/COMMENT	The project team has acknowledged the issue and deems it ok to stay as is.

On-Chain Analysis

Not Deployed Yet

Appendix A - Reviewed Documents

Document	Address
BConst.sol	N/A
BNum.sol	N/A
BMath.sol	N/A
BToken.sol	N/A
Bundle.sol	N/A
BundleFactory.sol	N/A
BundleLock.sol	N/A
Controller.sol	N/A
IBundle.sol	N/A
IBundleFactory.sol	N/A
IBundleLock.sol	N/A
BundleToken.sol	N/A
IERC20.sol	N/A
IMinter.sol	N/A
IMinterDetailed.sol	N/A
IPriceOracle.sol	N/A
IRebalancer.sol	N/A
IUnbinder.sol	N/A
IWETH.sol	N/A
Minter.sol	N/A
MinterGuard.sol	N/A
PriceOracle.sol	N/A
Rebalancer.sol	N/A

Timelock.sol	N/A
Unbinder.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