PALADIN
BLOCKCHAIN SECURITY

# Smart Contract Security Assessment

Final Report

## For Revault Network

07 August 2021

paladinsec.co

info@paladinsec.co

# Table of Contents

# Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

# 1 Overview

This report has been prepared for Revault Network on the Binance Smart Chain (BSC). Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

## 1.1 Paladin Round Table

This audit is a Paladin Round Table audit. As Paladin is a user-safety focused auditor, we may identify and mark issues as higher risk than what may be considered the norm in the industry since we tend to take the perspective of the user when approaching the audit.

Projects where the leadership requires more governance freedom than usual will fall under our Round Table audit. These projects are usually more complex than most others, and thus the project leadership requires the ability to navigate around and react appropriately to bugs and exploits, such as having the ability to correct mistakes and migrate funds to new contracts. This ability however could mean that the project leadership has full control over staked funds, assets, tokens and many of the key aspects of the protocol.

A Round Table audit has been chosen since the project runs behind proxies which allow the client to upgrade their contracts in case things go wrong. Since this freedom can also be used for less desirable reasons, including but not limited to withdrawing all staked funds, we have omitted all governance related issues from this audit. Investors should assess the honesty, reliability and community position of the client in their investment decision.

## 1.2 Summary

Paladin did not audit any of the vaults nor the implementations of how Revault interacts with these vaults.

The interesting feature of Revault is that the smart contracts are completely agnostic of the underlying protocols they will be used for. They have generic functions that allow for calling any contract in the name of a revault user proxy. While this function allows for a lot of freedom and interesting features, it does come with the disadvantage that 1) users might easily be tricked into approving malicious transactions, and 2) that Paladin is not able to audit any of the specific strategies.

Revault should thus be extra careful with testing each strategy and should also carefully protect their front-end to minimize the risk of it being compromised and tricking users into accepting malicious transactions.

| | |
|---|---|
| **Project Name** | Revault Network |
| **URL** | https://www.revault.network/ |
| **Platform** | Binance Smart Chain |
| **Language** | Solidity |

## 1.3 Contracts Assessed

The contracts were provided to Paladin in a zip archive. We will update this report after verifying that the deployed contracts match the ones we have audited.

| Name | Contract | Live Code Match |
|---|---|---|
| ReVault | ReVault.sol | |
| RevaUserProxy | RevaUserProxy.sol | |
| RevaUserProxyFactory | RevaUserProxyFactory.sol | |
| RevaChef | RevaChef.sol | |
| RevaStakingPool | RevaStakingPool.sol | |
| RevaLpStakingPool | RevaLpStakingPool.sol | |
| RevaFeeReceiver | RevaFeeReceiver.sol | |
| Zap | Zap.sol | |
| ZapAndDeposit | ZapAndDeposit.sol | |
| RevaToken | RevaToken.sol | |
| vRevaToken | vRevaToken.sol | |
| Timelock | Timelock.sol | |

## 1.4 Findings Summary

| Severity | Found | Resolved | Partially Resolved | Acknowledged (no change made) |
|---|---|---|---|---|
| High | 11 | 11 | - | - |
| Medium | 9 | 8 | 1 | - |
| Low | 14 | 11 | 1 | 2 |
| Informational | 26 | 13 | 2 | 11 |
| **Total** | **60** | **43** | **4** | **13** |

### Classification of Issues

| Severity | Description |
|---|---|
| High | Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency. |
| Medium | Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible. |
| Low | Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless. |
| Informational | Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any. |

### 1.4.1 General

| ID | Severity | Summary | Status |
|---|---|---|---|
| 01 | INFO | Usage of transfer instead of safeTransfer | RESOLVED |
| 02 | INFO | System is inefficient for tokens with a transfer tax | ACKNOWLEDGED |

### 1.4.2 ReVault

| ID | Severity | Summary | Status |
|---|---|---|---|
| 03 | HIGH | depositToVaultFor allows anyone to execute arbitrary code on any vault of another user | RESOLVED |
| 04 | HIGH | Vaults that can lose value will inflate the vaultchef balance of the user | RESOLVED |
| 05 | HIGH | `createAmountPayload` is insufficiently wide | RESOLVED |
| 06 | HIGH | `harvestCompoundVault` does not convert the harvested tokens to the deposit token due to a typo | RESOLVED |
| 07 | HIGH | Many functions do not update the `vaultchef` balance, allowing a user to inflate their rewards | RESOLVED |
| 08 | HIGH | `harvestCompoundVault` does not use the `callDepositVault` function to redeposit | RESOLVED |
| 09 | MEDIUM | `harvestCompoundVault` does not update the principal balance of the user | RESOLVED |
| 10 | MEDIUM | Harvested tokens from the deposits in the rebalance calls are stuck in the ReVault contract and not sent to the user | RESOLVED |
| 11 | MEDIUM | `harvestCompoundVault` and `rebalanceDepositVault` does not redeposit the exact amount earned | RESOLVED |
| 12 | MEDIUM | Lack of reentrancy guards could result in exploitation of before-after deposit value accounting | RESOLVED |
| 13 | LOW | Lack of non-duplicated check in the `addVault` function | RESOLVED |
| 14 | INFO | Codebase contains `console.log` which unnecessarily wastes gas | RESOLVED |
| 15 | INFO | `withdrawFromVault` sends the Reva to the `msg.sender` instead of `_user` | ACKNOWLEDGED |
| 16 | INFO | Smart contracts may not need to interact with ReVault | ACKNOWLEDGED |
| 17 | INFO | Vaults which deposit the native token of the underlying protocol will cause unintended side effects | ACKNOWLEDGED |
| 18 | INFO | The code contains several redundant logic which could make mistakes more likely to go unnoticed when changes are made | ACKNOWLEDGED |
| 19 | INFO | `createAmountPayload` could be optimized with encodePacked | RESOLVED |

### 1.4.3 RevaUserProxy

| ID | Severity | Summary | Status |
|---|---|---|---|
| 20 | LOW | Requiring the deposit token balance to be set to zero after a deposit allows for DoS by transferring a few tokens to the user proxy | RESOLVED |

### 1.4.4 RevaUserProxyFactory

No issues found.

### 1.4.5 RevaChef

| ID | Severity | Summary | Status |
|---|---|---|---|
| 21 | HIGH | `lastRewardBlock` is updated before the reward calculation resulting in zero rewards | RESOLVED |
| 22 | MEDIUM | Reward mechanism is vulnerable to price manipulation | PARTIAL |
| 23 | MEDIUM | TVL-based allocation mechanism does not guarantee that the `rewardRate` will be minted exactly | RESOLVED |
| 24 | INFO | Lack of parameter validation | PARTIAL |
| 25 | INFO | `tvlDiff` mechanism can be simplified | PARTIAL |
| 26 | INFO | Unused variables `_reVaultList`, `totalRevaAllocPoint`, `accReceivedRevaFromFees` and `accWithdrawnRevaFromFees` | RESOLVED |
| 27 | INFO | Lack of events for `setRevaPerBlock` and `setRevault` | RESOLVED |

### 1.4.6 RevaStakingPool and RevaLpStakingPool

| ID | Severity | Summary | Status |
|---|---|---|---|
| 28 | HIGH | `updatePool` will never update the Reva fees because the block number check is inverted | RESOLVED |
| 29 | HIGH | User amount is not adjusted on early withdrawal allowing any user to drain the pool of all tokens | RESOLVED |
| 30 | HIGH | Syrup bug present on `emergencyWithdraw` allows a user to mint huge amounts of vReva | RESOLVED |
| 31 | MEDIUM | Lack of re-entrancy guard could result in total loss of funds if the code is ever updated to include a way to reenter | RESOLVED |
| 32 | LOW | Unclear purpose of `withdrawalFee` rewards mechanism in LP staking pool | RESOLVED |
| 33 | LOW | Reva fee distribution mechanism could cause issues if `massUpdatePools` is not called on `allocPoint` modifications | RESOLVED |
| 34 | LOW | `revaFeeReward` will recalculate even though it was already up to date | RESOLVED |
| 35 | LOW | Withdrawal fee distribution mechanism will malfunction if the other wallets withdraw from the `feeReceiver` | RESOLVED |
| 36 | LOW | The `pendingReva` function will revert if `totalAllocPoint` is zero | ACKNOWLEDGED |
| 37 | INFO | `timeDeposited` not reset on emergency withdrawals | ACKNOWLEDGED |
| 38 | INFO | Lack of parameter validation | ACKNOWLEDGED |
| 39 | INFO | Lack of events for `add`, `set`, `setRevaPerBlock` and `setEarlyWithdrawalFee` | RESOLVED |

### 1.4.7 RevaFeeReceiver

No issues found.

### 1.4.8 Zap

| ID | Severity | Summary | Status |
|---|---|---|---|
| 40 | HIGH | `zapInTokenTokenTo` does not add liquidity in the correct token order causing the function to fail or severely malfunction for certain LP pairs | RESOLVED |
| 41 | MEDIUM | `getBUSDValue` returns only half the value of the LP pairs | RESOLVED |
| 42 | LOW | Token dust will accumulate when `_addLiquidity` is called | RESOLVED |
| 43 | LOW | Dust accumulated in the contract could be taken out by a malicious party by using a malicious token | RESOLVED |
| 44 | LOW | No parameter to indicate the minimal received amount can lead to higher loss of value through frontrunning for large orders | PARTIAL |
| 45 | LOW | Zap functions will malfunction for tokens with a transfer tax | RESOLVED |
| 46 | LOW | No validation that there is a pair in `getBUSDValue` | RESOLVED |
| 47 | LOW | `isFlip` will return `true` for tokens which are not yet added | ACKNOWLEDGED |
| 48 | INFO | Wrong usage of `require` | RESOLVED |
| 49 | INFO | Ambiguous variable name `isFlip` | ACKNOWLEDGED |
| 50 | INFO | Lack of events for `setRoutePairAddress`, `setNotFlip`, `removeToken`, `sweep` and `withdraw` | ACKNOWLEDGED |

### 1.4.9 ZapAndDeposit

| ID | Severity | Summary | Status |
|---|---|---|---|
| 51 | INFO | `createAmountPayload` could be optimized with encodePacked | RESOLVED |

### 1.4.10 RevaToken

| ID | Severity | Summary | Status |
|---|---|---|---|
| 52 | MEDIUM | Setting any fee address to the zero address will break most functionality | RESOLVED |
| 53 | LOW | Supply limit check can overflow in RevaToken | RESOLVED |
| 54 | INFO | Duplicate logic in `transfer` and `transferFrom` | ACKNOWLEDGED |
| 55 | INFO | Governance functionality is broken | RESOLVED |
| 56 | INFO | `delegateBySig` can be frontrun and cause denial of service | RESOLVED |
| 57 | INFO | `mint`, `setTreasury`, `setRevaStakeFeeReceiver`, `setRevaLpStakeFeeReceiver` and `setBurnFee` can be made `external` | RESOLVED |
| 58 | INFO | Lack of events for `setTreasury`, `setRevaStakeFeeReceiver`, `setRevaLpStakeFeeReceiver`, `setBurnFee` and `setMinter` | RESOLVED |

### 1.4.11 vRevaToken

| ID | Severity | Summary | Status |
|---|---|---|---|
| 59 | INFO | `delegateBySig` can be frontrun and cause denial of service | ACKNOWLEDGED |
| 60 | INFO | Wrong usage of `require` | RESOLVED |

### 1.4.12 Timelock

No issues found.

# 2 Findings

## 2.1 General Issues & Recommendations

| Issue #01 | Usage of `transfer` instead of `safeTransfer` |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Certain tokens may not use the `transfer` specification and may not revert on transfers (instead returning `false`). This has happened in a recent exploit where users could advertise they deposited a million tokens but in reality their wallet had none. The transfer function would return `false` but the contract would not handle this. |
| **Recommendation** | OpenZeppelin's SafeERC20 library will take care of this return value if any is returned, and it is also compatible with non-compliant tokens without any return value. Thus the OpenZeppelin library should be considered for all `transfer` and `transferFrom` calls. |
| **Resolution** | RESOLVED |

| Issue #02 | System is inefficient for tokens with a transfer tax |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | Deposits require at least two transfers while withdrawals require three transfers. A full round-loop thus will deduct 5 times the transfer tax instead of just 2 times for a normal user.<br><br>This is definitely not a critique on the system since it is inherent to any vault. |
| **Recommendation** | Consider acknowledging this shortcoming since it is inherent to all vaults and consider it whenever a token with a transfer tax is added. Ideally the project should cooperate with the underlying project to exclude the ReVault contract from the tax so that only 2 taxes will take place just as it would for an end user. |
| **Resolution** | ACKNOWLEDGED<br><br>The client has indicated that as their system is inefficient for said tokens it was never intended to be used for them. |

## 2.2 ReVault

The ReVault contract is the main management interface for all user vaults. It allows users to enter into vaults, to harvest vaults, to compound vaults and rebalance them.

When compounds are made, 29% of the principal increase is converted to Reva and given to the user, 1% of the principal increase is converted to Reva and sent to a `feeReceiver`, which we assume will be a RevaFeeReceiver distribution contract for these tokens to be distributed among stakers. The remaining 70% is sent to the user in the form of the principal token. When withdrawals are made, the full 30% is sent to the user in the form of Reva.

### 2.2.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `addVault`
- `withdrawToken`
- `setDepositMethod`
- `setWithdrawMethod`
- `setHarvestMethod`
- `setProfitToReva`
- `setProfitToRevaStakers`

### 2.2.2 Issues & Recommendations

| Issue #03 | **`depositToVaultFor` allows anyone to execute arbitrary code on any vault of another user** |
|---|---|
| **Severity** | HIGH SEVERITY |
| **Description** | The revault contract contains a method `depositToVaultFor` which allows a user to make a deposit for another user. The issue is that due to the way Revault works, deposits are just executing arbitrary code on a user's user proxy. Therefore, this deposit for function gives the user full freedom to execute code on any of the underlying protocols of another user. In case the underlying protocol for example contains a `withdrawTo` function, anyone can steal another user's stakes. |
| **Recommendation** | Consider removing the `depositToVaultFor` function or at best allowing it to be called solely by whitelisted contracts that are owned exclusively by Revault. |
| **Resolution** | RESOLVED<br>The method can now only be called if the receiver actually initiated the transaction, and in addition, there is now payload verification and whitelisting of the zap contracts that can call this function. |

| Issue #04 | **Vaults that can lose value will inflate the `vaultchef` balance of the user** |
|---|---|
| **Severity** | HIGH SEVERITY |
| **Description** | Many protocols have a `withdrawTo` function. A user can make a deposit that in fact withdraws their whole balance to their private wallet. In this case, the `vaultchef` balance will not be adjusted. Depositing again and repeating this over and over allows the user to inflate their `vaultchef` balance as much as they want.<br><br>In general, we see the `vaultchef` pattern as something ambitious but also highly difficult to secure. In our opinion it is simply a matter of when that a bad vault or token is added and the `vaultchef` is manipulated to inflate the rewards of an exploiter. |
| **Recommendation** | Although the `vaultchef` is a really interesting concept, given all the issues and risks it needs to be reconsidered.<br><br>Furthermore, there is no accounting for tokens with a transfer tax in the amount added to the `vaultchef`. This is however not extremely severe since in this instance it would just slightly inflate the user's rewards. |
| **Resolution** | RESOLVED<br><br>The project has implemented whitelisting of functions that can be called. That means that the project owner can explicitly say which functions can be used to withdraw from a vault. Since there is always a chance that a bad method is whitelisted, we have worked with the client to set up a risk management program for in case a bad payload is added. The client will actively monitor the BUSD values in each vault and has included extra code in the contract upon our recommendation that allows them to disable rewards on specific tokens. If a bad method is ever whitelisted by accident, the goal of the program is to mitigated this within 12 hours. This limits the impact of this issue significantly, since only at most the emissions over this period could be stolen.<br><br>It should be noted that the disabling is done on a token basis and not a vault basis, which means that there will likely be some user experience annoyance if this scenario presents itself. After the initial issue is mitigated through the risk management strategy, we recommend the client to do a careful upgrade to re-enable rewards for all other vaults that had the same token. |

| Issue #05 | `createAmountPayload` is insufficiently wide |
|---|---|
| **Severity** | HIGH SEVERITY |
| **Description** | The `createAmountPayload` method does not account for the length of the `_newdata` array. This will result in most calls that use `createAmountPayload` to fail. |
| **Recommendation** | Consider using the `abi.encodePacked` approach as is recommended in a later issue. |
| **Resolution** | RESOLVED<br>The function has been removed in favor of the recommended `abi.encodePacked` approach. |

| Issue #06 | `harvestCompoundVault` does not convert the harvested tokens to the deposit token due to a typo |
|---|---|
| **Severity** | HIGH SEVERITY |
| **Location** | Line 161<br>`zap.zapInToken(vault.nativeTokenAddress, IBEP20(vault.depositTokenAddress).balanceOf(address(this)), vault.depositTokenAddress);` |
| **Description** | The zap call in `harvestCompoundVault` tries to zap the `nativeToken` using the balance of the deposit token as input. This will almost always revert and will never give the desired behavior. |
| **Recommendation** | Consider changing the call to the correct token amount.<br>`zap.zapInToken(vault.nativeTokenAddress, IBEP20(vault.nativeTokenAddress).balanceOf(address(this)), vault.depositTokenAddress);`<br>⚠ It should also be noted that the deposit does not use the vault's balance but instead relies on the `_payloadDeposit` calldata so this might result in funds being stuck in the vault. `createAmountPayload` can be considered to ensure that all funds are taken out. |
| **Resolution** | RESOLVED<br>The function has been removed. |

| Issue #07 | **Many functions do not update the `vaultchef` balance, allowing a user to inflate their rewards** |
|---|---|
| **Severity** | HIGH SEVERITY |
| **Description** | Currently, many of the `deposit` and `withdraw` functions (for example `harvestWithdrawVault`) do not update the `vaultchef`. This allows a user to inflate their `vaultchef` balance. |
| **Recommendation** | Although the `vaultchef` is a really interesting concept, given all the issues and risks it needs to be reconsidered.<br><br>If the `vaultchef` feature should remain, consider carefully adding the balance adjustments everywhere. |
| **Resolution** | RESOLVED<br><br>The `deposit` and `withdraw` functions adjust the revachef balances properly, but the rebalance functions do not. However, since payloads are explicitly whitelisted and the client has set up a risk management strategy if they whitelist the wrong method, we have indicated this issue as resolved. In case a wrong method is whitelisted by the client, the impact can be managed as explained in a previous issue. The client should remain extremely careful with regards to which methods they whitelist and consider the possibility of abusing them through the rebalance function.<br><br>It should be noted that due to the current logic of the revault contract, any vault that has the same deposit token as the native token might misbehave. |

| Issue #08 | **`harvestCompoundVault` does not use the `callDepositVault` function to redeposit** |
|---|---|
| **Severity** | HIGH SEVERITY |
| **Location** | Line 163<br>`RevaUserProxy(userProxyAddress).callVault(vault.vaultAddress, vault.depositTokenAddress, vault.nativeTokenAddress, _payloadDeposit);` |
| **Description** | For deposits into a vault, the `callDepositVault` method should be used. |
| **Recommendation** | Consider using `callDepositVault` instead of `callVault` - transferring the tokens to the proxy should not be forgotten. |
| **Resolution** | RESOLVED<br>The function has been removed. |

| Issue #09 | **`harvestCompoundVault` does not update the principal balance of the user** |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Description** | Currently the principal of the user is not increased in the `harvestWithdrawVault` function deposit section. This is inconsistent with the other functions. |
| **Recommendation** | Consider increasing the user principal correctly in the `deposit` call. |
| **Resolution** | RESOLVED<br>The function has been removed. |

| Issue #10 | **Harvested tokens from the deposits in the rebalance calls are stuck in the ReVault contract and not sent to the user** |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Description** | On a deposit to a vault, the vault will send back any harvested tokens to ReVault. Within both `rebalanceDepositAll` and `rebalanceDepositAllDynamicAmount`, the functions do not actually forward these funds to the user, causing them to be stuck in the ReVault contract to be taken out through another party or by governance. |
| **Recommendation** | Consider carefully assessing the risk of diverging code as a result of all this redundancy – such risk has already clearly materialized in the many ways the principal and RevaChef balances are currently not properly adjusted. |
| **Resolution** | RESOLVED<br>The client has added functionality to withdraw the native token from the vault to the user after these rebalances. |

| Issue #11 | **`harvestCompoundVault` and `rebalanceDepositVault` does not redeposit the exact amount earned** |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Description** | When tokens are compounded in `harvestCompoundVault`, the subsequent deposit is predefined by the user who has to guess the deposit amount – this will likely either be too little or too large and result in tokens being wasted in the revault contract.<br><br>We expect `rebalanceDepositVault` to be less problematic since most withdraw payloads contain the exact amount. Thus here, the current behavior could be maintained. |
| **Recommendation** | Consider using the `createAmountPayload` method to specify the deposit amount on the go. |
| **Resolution** | RESOLVED<br><br>`HarvestCompoundVault` has been removed and the other function clearly indicates that it is a `depositAll` call now. |

| Issue #12 | **Lack of reentrancy guards could result in exploitation of before-after deposit value accounting** |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Description** | Currently, the code does not contain reentrancy guards – this allows an exploiter to inflate the before-after pattern balance if they can reenter between these two calls.<br><br>An example before-after pattern is provided in the `withdrawFromVault` function:<br><br>`uint prevRevaBalance = reva.balanceOf(msg.sender);`<br>`...`<br>`uint postRevaBalance = reva.balanceOf(msg.sender);`<br>`return (vaultDepositTokenAmount, postRevaBalance.sub(prevRevaBalance));`<br><br>In case a second call to this function is made at the location of the ..., the `postRevaBalance` of the outer call will be heavily inflated since it also captures the increase of the inner call. |
| **Recommendation** | Consider adding reentrancy guards to all user-facing functions.<br><br>Note: In the example provided above, even reentrancy guards would be insufficient as the user can transfer Reva to themselves from a separate wallet if they can inject code in the ... section and still inflate the balance. |
| **Resolution** | RESOLVED<br><br>All relevant functions now have reentrancy guards. |

| Issue #13 | Lack of non-duplicated check in the `addVault` function |
| --- | --- |
| **Severity** | LOW SEVERITY |
| **Description** | Two identical vaults may currently be added to the vault list. This will unnecessarily increase the vaults array size. |
| **Recommendation** | Consider adding a `nonDuplicated` modifier that keeps track of a vault signature hash.<br><br>`mapping(bytes32 => bool)vaultExists;`<br><br>`modifier nonDuplicated(address _vaultAddress, address _depositTokenAddress, address _nativeTokenAddress) {`<br>`  require(!vaultExists[keccak256(abi.encodePacked(_vaultAddress, _depositTokenAddress, _nativeTokenAddress))], “duplicate”);`<br>`  _;`<br>`}`<br><br>Note that within `addVaults`, this modifier should be included and afterward the `vaultExists` value must be set to true. |
| **Resolution** | RESOLVED |

| Issue #14 | Codebase contains `console.log` which unnecessarily wastes gas |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Currently the codebase still contains console logs which waste gas. |
| **Recommendation** | Consider removing the console dependency completely before going to production. |
| **Resolution** | RESOLVED |

| Issue #15 | `withdrawFromVault` sends the Reva to the `msg.sender` instead of `_user` |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | When tokens are withdrawn from a user's vault, they are sent to the `msg.sender` instead of the `_user`. This might not be desired. |
| **Recommendation** | Consider whether this is desired behavior. |
| **Resolution** | ACKNOWLEDGED |

| Issue #16 | Smart contracts may not need to interact with ReVault |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | Revault seems to be targeted at retail users primarily. A user that operates through management smart contracts or with timelock / multisig smart contracts might not be interested in using Revault. The attack surface of the protocol can thus be reduced by preventing smart contracts from interacting with the protocol.<br><br>This issue is marked as informational since it is only meant as something to consider and we do not see obvious abuse cases of smart contracts except for potential manipulations of the RevaChef, which have been pointed out there. |
| **Recommendation** | Consider adding an `onlyEOA` modifier to the user functions to ensure that smart contracts are unable to interact with ReVault. Since most of the vaults are isolated from each other, this is mainly a safeguard on the vault chef. |
| **Resolution** | ACKNOWLEDGED |

| Issue #17 | **Vaults which deposit the native token of the underlying protocol will cause unintended side effects** |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Many farms and vaults allow for users to deposit their native token; however this will cause unintended side effects since the Revault protocol distinguishes between the deposit and native tokens. |
| **Recommendation** | Consider not allowing native token vaults or adding an `isNative` boolean to the `VaultInfo` and adjusting all relevant logic like the payout mechanism during withdrawal and mechanisms within the proxy itself. |
| **Resolution** | ACKNOWLEDGED |

| Issue #18 | **The code contains several redundant logic which could make mistakes more likely to go unnoticed when changes are made** |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | The code contains several redundant logic – for example, `rebalanceDepositAll` and `rebalanceDepositAlDynamicAmount` are nearly identical. |
| **Recommendation** | Consider carefully assessing the risk of diverging code as a result of all this redundancy – such risk has already clearly materialized in the many ways the principal and RevaChef balances are currently not properly adjusted. |
| **Resolution** | ACKNOWLEDGED |

| Issue #19 | `createAmountPayload` could be optimized with encodePacked |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | Currently the `createAmountPayload` function is highly inefficient. We are unsure why the Solidity method `abi.encodePacked` is not used to cheaply concatenate the byte arrays. |
| **Recommendation** | Consider simply using `abi.encodePacked(_leftCallData, _newData, _rightCallData)` to concatenate the three arrays. |
| **Resolution** | RESOLVED |

## 2.3 RevaUserProxy

The RevaUserProxy represents a virtual user which the revault system can use to execute deposits and withdrawals in almost any underlying project. Each user has their own user proxy.

### 2.3.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `callVault`
- `callDepositVault`

### 2.3.2 Issues & Recommendations

| Issue #20 | **Requiring the deposit token balance to be set to zero after a deposit allows for DoS by transferring a few tokens to the user proxy** |
|---|---|
| **Severity** | LOW SEVERITY |
| **Location** | Line 59<br>`require(IBEP20(_depositTokenAddress).balanceOf(address(this)) == 0, "Proxy didn't deposit all");` |
| **Description** | A malicious party could deposit a few tokens into the RevaUserProxy of a user they do not like. This user can then no longer deposit since each deposit fails as these tokens do not remain (few underlying protocols take the whole balance of the sender; instead they take a predetermined amount). |
| **Recommendation** | Consider carefully redesigning this check to, for example, require the balance before to be equal to the balance afterwards.<br><br>It should also be noted that this function might not allow depositing in a native pool in a Masterchef, whereas native tokens are sent to the user proxy while the user proxy also deposits tokens. This is an extra reason to carefully redesign this requirement.<br><br>There is also no check on the contract BNB balance in case a BNB deposit is done. We assume the deposit token is still set to WBNB in this case to allow the price oracle to function, however, of course no BNB will be transferred in. |
| **Resolution** | RESOLVED<br><br>A before-after pattern is implemented that checks that the complete deposited amount has left the proxy. However, this pattern still does not work for `depositAll` payloads since in this case a front-runner could send some extra tokens to the proxy that would be deposited as well. In this case the equality check would still fail since more is deposited to the vault then was sent to the proxy during the deposit.<br><br>After discussing with the client, we believe this is an acceptable compromise since this check is an important component for the brittle RevaChef mechanism which can easily fail if a deposit payload can be manipulated into not actually making a deposit. Furthermore they have stated that they will not use `depositAll` payloads in these cases. |

## 2.4 RevaUserProxyFactory

The RevaUserProxyFactory is a utility contract that can create user proxies for a user through the `createUserProxy` function. The value of having such a factory is that the validity of a proxy can easily be verified by checking that the proxy was created through interaction with the RevaUserProxyFactory address and that it separates the concern from the main ReVault contract. This might have been necessary as to not hit the maximum byte limit on the ReVault contract.

When the ReVault contract creates `userProxies`, the ownership will be transferred to ReVault.

### 2.4.1 Issues & Recommendations

No issues found.

## 2.5 RevaChef

The RevaChef is a contract which is called by the ReVault contract whenever a deposit or withdrawal is made. It keeps track of the user balances and serves as an extra reward contract for users that have stakes in ReVault.

The RevaChef is similar to a Masterchef with the difference that the ReVault contract is in charge of managing deposits and withdrawals. The only function users are allowed to call is `harvest`. More importantly, the main difference between RevaChef and a Masterchef is the fact that the allocation of pools is based on the USD value of these pools, which is indicated by the PancakeSwap spot price provided by the Zap function.

### 2.5.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `setRevaPerBlock`
- `setRevault`
- `notifyDeposited`
- `notifyWithdrawn`

## 2.5.2 Issues & Recommendations

| Issue #21 | `lastRewardBlock` is updated before the reward calculation resulting in zero rewards |
|---|---|
| **Severity** | HIGH SEVERITY |
| **Location** | Lines 121–124<br>`tokenInfo.lastRewardBlock = block.number;`<br><br>`if (tokenInfo.totalPrincipal > 0 && tokenInfo.tvlBusd > 0) {`<br>`uint256 multiplier =`<br>`(block.number).sub(tokenInfo.lastRewardBlock);` |
| **Description** | The multiplier for the rewards will always be zero since the `lastRewardBlock` is set before the calculation of it. |
| **Recommendation** | Consider setting the `lastRewardBlock` after the multiplier calculation. |
| **Resolution** | RESOLVED<br>The `lastRewardBlock` is now moved downward. |

| Issue #22 | Reward mechanism is vulnerable to price manipulation |
| --- | --- |
| **Severity** | MEDIUM SEVERITY |
| **Description** | Due to the fact that the BUSD value of a vault is calculated on the spot price of the token in Uniswap, a malicious user might manipulate this value heavily by, for example, taking a flash loan from the pair to adjust the price to any rate.<br><br>This issue is currently marked as medium severity because they will only be able to take 100% of the allocation points since the last update of that specific token. However, we believe that it is only going to be a matter of time before a token with malicious functions gets whitelisted and allows an exploiter to seriously abuse the mechanism. This will become a genuine risk especially when many new vaults start being added. |
| **Recommendation** | In the short term, consider adding a `massUpdatePools` method which is regularly called by the owner to minimize the potential impact of price manipulation.<br><br>In the long term, consider using a TWAP oracle – Uniswap provides useful components to easily set one up. |
| **Resolution** | PARTIALLY RESOLVED<br><br>The client has added a `massUpdatePools` method which will be called regularly. Furthermore, the client has added protection against flash-loan based price manipulation which makes the cost and difficulty of manipulation higher. |

| Issue #23 | **TVL-based allocation mechanism does not guarantee that the `rewardRate` will be minted exactly** |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Description** | Due to the TVL-based allocation mechanism, there is no guarantee that the `rewardRate` will be exactly minted. This means that there could be more or less tokens rewarded per block then the `rewardRate`. |
| **Recommendation** | Consider carefully the magnitude of this issue. Consider carefully redesigning the mechanism in case it is deemed too large. The main way to solve this would be to update all reward variables on every `deposit` and `withdraw` but this would be highly costly. |
| **Resolution** | RESOLVED<br>The client has moved to an allocation point mechanism where the allocation points are periodically updated based on the TVL in the pools. Then the individual user rewards are based upon their nominal weight in the pools. This mechanism should ensure an exact reward rate. |

| Issue #24 | **Lack of parameter validation** |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Both in the initializer and governance functions, important variables are not validated and could be set wrongly by accident. |
| **Recommendation** | Consider validating that the `startBlock` is in the future and that `revaPerBlock` is not excessive in both the constructor and governance functions. |
| **Resolution** | PARTIALLY RESOLVED<br>The `startBlock` is now validated. |

| Issue #25 | `tvlDiff` mechanism can be simplified |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | The `tvlDiff` mechanism to update the `totalRevaultTvlBusd` is unnecessary complex. |
| **Recommendation** | Consider validating the following proposal and then simplifying the `totalRevaultTvlBusd` adjustment to<br>`totalRevaultTvlBusd = totalRevaultTvlBusd.add(currTokenTvlBusd).sub(prevTokenTvlBusd);` |
| **Resolution** | PARTIALLY RESOLVED<br>The `tvlDiff` mechanism is removed. |

| Issue #26 | Unused variables `_reVaultList`, `totalRevaAllocPoint`, `accReceivedRevaFromFees` and `accWithdrawnRevaFromFees` |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | The contract contains variables that are unused. These variables could confuse third-party reviewers in their assessment of the contract. |
| **Recommendation** | Consider removing these unused variables. |
| **Resolution** | RESOLVED |

| Issue #27 | Lack of events for `setRevaPerBlock` and `setRevault` |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | Functions that affect the status of sensitive variables should emit events as notifications. |
| **Recommendation** | Add events for the above functions. |
| **Resolution** | RESOLVED |

## 2.6 RevaStakingPool and RevaLpStakingPool

The RevaStakingPool is a staking contract that allows users to stake and lock-in Reva tokens for a period. The staking pool can have various configurable staking plans and over the duration of the stake, users will earn Reva tokens from the Reva emissions and early withdrawal fees of other users. While staking, users receive the non-transferrable vReva governance token to participate in voting. Governance can create pools with an infinitely high minimum staking period and an early withdrawal penalty of up to 100%. Staking fees should go into the RevaFeeReceiver contract where they can later be used to distribute to stakers. Please see the RevaFeeReceiver section for more information.

The RevaLpStakingPool is a simplified contract based on the RevaStakingPool. It allows the staking of LP tokens similar to a Masterchef but does not have the lockup mechanism of the RevaStakingPool.

### 2.6.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `add`
- `set`
- `setRevaPerBlock`

## 2.6.2 Issues & Recommendations

| Issue #28 | **`updatePool` will never update the Reva fees because the block number check is inverted** |
|---|---|
| **Severity** | HIGH SEVERITY |
| **Contract** | RevaStakingPool, RevaLPStakingPool |
| **Location** | `if (block.number <= lastUpdatedRevaFeesBlock) {` |
| **Description** | Currently, the variable to account for the rewards from withdrawal fees is only updated if the current block is smaller than the last time this was done, which is obviously impossible. Withdrawal fees will thus not be distributed to stakers. |
| **Recommendation** | Consider inverting the check.<br>`if (block.number > lastUpdatedRevaFeesBlock) {` |
| **Resolution** | RESOLVED<br>The check has been inverted. [RevaLPStakingPool] |

| Issue #29 | **User amount is not adjusted on early withdrawal allowing any user to drain the pool of all tokens** |
|---|---|
| **Severity** | HIGH SEVERITY |
| **Contract** | RevaStakingPool |
| **Description** | Currently when a user calls `withdrawEarly`, their balance is not decreased and they can simply call it as many times as they want to receive all tokens of the pool (minus the withdrawal fee). |
| **Recommendation** | Consider consolidating the withdrawal function to reduce the risk of redundancy errors and to follow the DRY pattern. Otherwise, simply consider adding `user.amount = user.amount.sub (_amount);` before the `rewardDebt` updates in the `withdrawEarly` function. |
| **Resolution** | RESOLVED |

| Issue #30 | **Syrup bug present on `emergencyWithdraw` allows a user to mint huge amounts of vReva** |
| --- | --- |
| **Severity** | HIGH SEVERITY |
| **Contract** | RevaStakingPool |
| **Description** | Currently, the vReva token is not burned. This is similar to the Syrup bug that PancakeSwap had and is the reason why PancakeSwap no longer uses the Syrup token.<br><br>Users can deposit and `emergencyWithdraw` to retain their vReva tokens, and if this is done many times, a user can amass huge amounts of vReva. |
| **Recommendation** | Consider consolidating the two `emergencyWithdraw` functions and adding the proper vReva burn logic in them. |
| **Resolution** | RESOLVED<br><br>vReva is now burned again on emergency withdrawals. |

| Issue #31 | **Lack of re-entrancy guard could result in total loss of funds if the code is ever updated to include a way to reenter** |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Contract** | RevaStakingPool, RevaLPStakingPool |
| **Description** | Currently we could not find any way to re-enter; however, due to various sections of the code (such as `rewardDebt` update not following checks-effects-interactions, before-after pattern on `deposit`) and the possibility that `lpTokens` may have re-entrancy vectors as we do not know what token type they are, we recommend adding re-entrancy guards.<br><br>If ever a re-entrancy vector is introduced, be it in a fork of the contract or through a bad `lpToken`, the contract would be completely drained of tokens. |
| **Recommendation** | Consider adding re-entrancy guards to all `deposit` and `withdraw` functions so that future forks will not accidentally implement dangerous code if they use different tokens rather than safe `lpTokens` without hooks. |
| **Resolution** | RESOLVED<br><br>All relevant functions now have re-entrancy guards. |

| Issue #32 | **Unclear purpose of `withdrawalFee` rewards mechanism in LP staking pool** |
|---|---|
| **Severity** | LOW SEVERITY |
| **Contract** | RevaLPStakingPool |
| **Description** | The LP staking pool contains the same reward mechanism as the Reva staking pool; however since there are no withdrawal fees here we are unsure of the purpose. |
| **Recommendation** | Consider whether this mechanism was added by accident, and if not, consider explaining the purpose of it. |
| **Resolution** | RESOLVED<br>The client has indicated that this pool will also receive rewards but through a separate mechanism then the withdrawal fees. |

| Issue #33 | **Reva fee distribution mechanism could cause issues if `massUpdatePools` is not called on `allocPoint` modifications** |
|---|---|
| **Severity** | LOW SEVERITY |
| **Contract** | RevaStakingPool, RevaLPStakingPool |
| **Description** | If `massUpdatePools` is not called before `add` or `set` is called before the `allocPoint` modifications, pools could accumulate excessive `revaPerShareFromFees` if the `updatePools` are called in a bad order. |
| **Recommendation** | Consider making the transfer from the `feeReceiver` not revert even if there are insufficient tokens. This has been recommended with more detail in another issue. Furthermore, consider making `massUpdatePools` a requirement in any function that changes `allocPoint`. At some point, this will of course set a hard limit on the amount of pools since the function reverts from running out of gas. |
| **Resolution** | RESOLVED<br>`add` and `set` now update all `allocPoints`. The transfer from the `feeReceiver` now also does a balance check to ensure it does not revert. |

| Issue #34 | `revaFeeReward` will recalculate even though it was already up to date |
|---|---|
| **Severity** | LOW SEVERITY |
| **Contract** | RevaStakingPool, RevaLPStakingPool |
| **Location** | Line 148<br>`if (pool.lastAccRevaFromFees <= accRevaFromFees) {` |
| **Description** | Currently, even though the `lastAccRevaFromFees` variable is already up to date, `accRevaFromFees` will be recalculated. |
| **Recommendation** | Consider making the check stricter.<br>`if (pool.lastAccRevaFromFees < accRevaFromFees) {` |
| **Resolution** | RESOLVED |

| Issue #35 | Withdrawal fee distribution mechanism will malfunction if the other wallets withdraw from the `feeReceiver` |
|---|---|
| **Severity** | LOW SEVERITY |
| **Contract** | RevaStakingPool, RevaLPStakingPool |
| **Description** | Within the `feeReceiver`, multiple addresses could withdraw currency from the `feeReceiver`. In case another address that is not the staking pool does this, the staking pool will revert because `accWithdrawnRevaFromFees` will be set incorrectly.<br><br>If both contracts use the same `feeReceiver`, this will very likely cause issues. |
| **Recommendation** | Consider creating a function `transferFromFeeReceiver` that contains the logic to withdraw from the `feeReceiver`. This function should not revert if the `feeReceiver` has insufficient funds (eg. through try-catch). |
| **Resolution** | RESOLVED<br><br>The client has indicated that separate `feeReceivers` will be used, and the client has added a balance check to ensure this withdrawal will not fail. |

| Issue #36 | The `pendingReva` function will revert if `totalAllocPoint` is zero |
|---|---|
| **Severity** | LOW SEVERITY |
| **Contract** | RevaStakingPool, RevaLPStakingPool |
| **Description** | In the `pendingReva` function, at some point a division is made by the `totalAllocPoint` variable. If all pools have their rewards set to zero, this variable will be zero as well. The requests will then revert with a division by zero error. |
| **Recommendation** | Consider only calculating the accumulated rewards since the `lastRewardBlock` if the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the `block.number` and `lpSupply`, like so:<br>`if (block.number > pool.lastRewardBlock && poo.lpSupply != 0 && totalAllocPoint > 0) {`<br>It should be noted that `pendingReva` also currently does not account for the Reva received from withdrawal fees. The client can consider fixing this as well. |
| **Resolution** | ACKNOWLEDGED |

| Issue #37 | `timeDeposited` not reset on emergency withdrawals |
|---|---|
| **Severity** | INFORMATIONAL |
| **Contract** | RevaStakingPool |
| **Description** | The `timeDeposited` variable is not reset on emergency withdrawals. Although this is not a problem as far as we can see, it is inconsistent with the other variables. |
| **Recommendation** | Consider whether this is desired behavior. |
| **Resolution** | ACKNOWLEDGED |

| Issue #38 | Lack of parameter validation |
|---|---|
| **Severity** | INFORMATIONAL |
| **Contract** | RevaStakingPool, RevaLPStakingPool |
| **Description** | Both in the initializer and governance functions, important variables are not validated and could be set wrongly by accident. |
| **Recommendation** | Consider validating that the `revaFeeReceiver` is not the zero address, the `startBlock` is greater then the current block number, `revaPerBlock` is not set to an excessive value and that `earlyWithdrawalFee` is not set to an excessive value.<br><br>This validation should be done in all relevant functions (initializer and setters).<br><br>Furthermore, on the `lpStakingPool`, a validation should be done in the add function that the token added is in fact a token, otherwise `massUpdatePools` might permanently break:<br><br>`IBEP20(_lpToken).balanceOf(address(this));` |
| **Resolution** | ACKNOWLEDGED |

| Issue #39 | Lack of events for `add`, `set`, `setRevaPerBlock` and `setEarlyWithdrawalFee` |
|---|---|
| **Severity** | INFORMATIONAL |
| **Contract** | RevaStakingPool, RevaLPStakingPool |
| **Description** | Functions that affect the status of sensitive variables should emit events as notifications. |
| **Recommendation** | Add events for the above functions. |
| **Resolution** | RESOLVED |

## 2.7 RevaFeeReceiver

The RevaFeeReceiver is a simple holder contract for Reva tokens. The contract owner can approve recipients to withdraw tokens from it at any time.

To have this contract function properly, the RevaStakingPool should be the only approved contract. It should be noted that if other contracts or wallets are approved and take tokens from the receiver, the staking pool will malfunction.

### 2.7.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `addRecipient`

### 2.7.2 Issues & Recommendations

No issues found.

## 2.8 Zap

The zap contract is a utility contract which allows users to easily zap into PancakeSwap v2 pairs and tokens.

When there is a remainder of one token when liquidity is added, this remainder is sent to the `feeReceiver` and not the user.

### 2.8.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `setRoutePairAddress`
- `setNotFlip`
- `removeToken`
- `sweep`
- `withdraw`

### 2.8.2 Issues & Recommendations

| Issue #40 | **zapInTokenTokenTo does not add liquidity in the correct token order causing the function to fail or severely malfunction for certain LP pairs** |
|---|---|
| **Severity** | HIGH SEVERITY |
| **Location** | `address other = _from == token0 ? token1 : token0;`<br>`uint sellAmount = amount.div(2);`<br>`uint otherAmount = _swap(_from, sellAmount, other, address(this));`<br>`_addLiquidity(token0, token1, amount.sub(sellAmount), otherAmount, receiver);` |
| **Description** | As seen in the code sample, the sale happens from `_from` to `other`, but liquidity is added with `token0` and `token1`. This could cause the liquidity amounts to be inverted and the user to receive significantly less LP tokens then they expected. The remaining tokens will furthermore be stuck in the contract. |
| **Recommendation** | Consider using the correct order.<br>`_addLiquidity(from, other, amount.sub(sellAmount), otherAmount, receiver);` |
| **Resolution** | RESOLVED<br>The recommendation has been implemented. |

| Issue #41 | `getBUSDValue` returns only half the value of the LP pairs |
| --- | --- |
| **Severity** | MEDIUM SEVERITY |
| **Location** | Line 73<br>`return IBEP20(BUSD).balanceOf(_token).mul(_amount).div(IBEP20(_token).totalSupply());`<br><br>Line 77<br>`return IBEP20(BUSD).balanceOf(busdWbnbPair).mul(wbnbAmount).div(IBEP20(WBNB).balanceOf(busdWbnbPair));` |
| **Description** | Currently the `getBUSDValue` function checks if either BUSD or WBNB is present in the LP pair, and if so, it derives the value of the provided amount. However, the code currently neglects to consider the fact that the pair consists of two tokens of equal value. The return variables should thus be multiplied by two. |
| **Recommendation** | Consider verifying that this behavior is indeed only returning half the value and if so, consider multiplying the return value by two in these statements. |
| **Resolution** | RESOLVED<br>The variables are now multiplied by two. |

| Issue #42 | **Token dust will accumulate when `_addLiquidity` is called** |
|---|---|
| **Severity** | LOW SEVERITY |
| **Description** | The Uniswap router will only add liquidity in the exact proportion of the pair, and any amount remaining simply remains stuck in the contract and is currently not being refunded. |
| **Recommendation** | Consider refunding dust; however, if gas cost is a consideration, this issue can also be marked as resolved through the explanation that these tokens will be taken out through sweep calls. It should be noted that a malicious party will be able to manipulate the zapper contract to take out these tokens as well, as explained in the next issue. |
| **Resolution** | RESOLVED<br>Dust is sent to the owner (`feeReceiver`). |

| Issue #43 | **Dust accumulated in the contract could be taken out by a malicious party by using a malicious token** |
|---|---|
| **Severity** | LOW SEVERITY |
| **Description** | The dust tokens accumulated in the contract through unbalanced `addLiquidity` calls can likely still be taken out by a malicious party using a malicious token. This is because the `amountsOut` parameter can be manipulated in a [maliciousToken, WBNB] pair.<br><br>A malicious user could for example call `zapInTokenTo` strategically to take out dust BNB. |
| **Recommendation** | Consider simply refunding dust tokens or acknowledging that third parties can take it out. Alternatively before-after checks combined with rentrancy guards can be used instead of relying on the `amountsOut` variable. |
| **Resolution** | RESOLVED<br>Dust is sent to the owner (`feeReceiver`). |

| Issue #44 | **No parameter to indicate the minimal received amount can lead to higher loss of value through frontrunning for large orders** |
|---|---|
| **Severity** | LOW SEVERITY |
| **Description** | The zap functions currently do not define a parameter for the user to indicate the minimum amount of tokens they need to receive. On less liquid pairs, this might result in unnecessary value being lost through frontrunning (so-called "sandwich" attacks). If the minimum amount to receive is automatically calculated on the front-end and injected as a requirement, this will prevent bots from extracting more value then is explicitly allowed by the user.<br><br>This issue is marked as low severity since it mainly prevents itself on less liquid pairs and mainly for higher value orders. It also does not lead to complete loss but only some leakage of value. |
| **Recommendation** | Consider adding a minimum received parameter to `zapInTokenTo`, `zapInToken`, `zapIn` and `zapOut`. |
| **Resolution** | PARTIALLY RESOLVED<br><br>The client has indicated that zapping should only be done for minimal amounts and thus frontrunning should be less of a concern. |

| Issue #45 | Zap functions will malfunction for tokens with a transfer tax |
| --- | --- |
| **Severity** | LOW SEVERITY |
| **Description** | Currently the zap functions transfer in the token first and then forward it to the router – this would not only be uneconomical but it also will not work as the zap functions ask the router to transfer the complete amount, and not the received amount. |
| **Recommendation** | If tokens with a transfer tax are ever desired to be zapped in, a fundamental redesign needs to be considered to minimize the amount of transfers. This would be possible by including the Uniswap router logic in the contract itself to directly transfer the token from the user to the pair.<br><br>In the short term, consider simply not using the zapper for tokens with a transfer tax. In case the non-economical route is desired, one could still use the before-after deposit pattern to figure out how many tokens are actually deposited. |
| **Resolution** | RESOLVED<br><br>The client has indicated that the zap function is inherently inefficient for these sorts of tokens so it should not be used for them. |

| Issue #46 | No validation that there is a pair in `getBUSDValue` |
|---|---|
| **Severity** | LOW SEVERITY |
| **Location** | `address pair = FACTORY.getPair(_token, WBNB);` |
| **Description** | The `getBUSDValue` method currently does not revert if the pair is non-existent whenever the pair is fetched. This will likely still be caught by a division by zero exception if there are no tokens in the zero address but might confuse users. |
| **Recommendation** | Consider checking the existence of the pair.<br>`address pair = FACTORY.getPair(_token, WBNB);`<br>`require(pair != address(0), "No pair");` |
| **Resolution** | RESOLVED |

| Issue #47 | `isFlip` will return `true` for tokens which are not yet added |
|---|---|
| **Severity** | LOW SEVERITY |
| **Description** | The `isFlip` function returns `true` for any address that is not explicitly initialized - this might confuse derivative contracts into thinking that certain tokens are an `lpPair` when they are just not added yet.<br><br>Furthermore, this might cause functionality to break for tokens that are not explicitly marked as `isNotFlip`. |
| **Recommendation** | Consider rethinking the `isFlip` logic in case the contract should be used for tokens which are not explicitly marked as `isNotFlip`. |
| **Resolution** | ACKNOWLEDGED<br><br>The client has indicated that doing the opposite would result in similar behavior. We have indicated that it might be smart to have a boolean for tokens that is always true for any token with an explicit `isFlip` status. This final recommendation was not implemented. |

| Issue #48 | Wrong usage of `require` |
|---|---|
| **Severity** | INFORMATIONAL |
| **Location** | `require(false, "throw");` |
| **Description** | The codebase currently uses `require` to revert; however, the best-practice is to use the `revert` keyword directly. |
| **Recommendation** | Consider using revert directly.<br>`revert("revert reason");` |
| **Resolution** | RESOLVED |

| Issue #49 | Ambiguous variable name `isFlip` |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | The zap contract uses the terminology `flip` to indicate an LP token. This might be confusing to third-party reviewers. |
| **Recommendation** | Consider renaming the flip terminologies to something like `lpPair`, `lpToken`, or `pair`. |
| **Resolution** | ACKNOWLEDGED |

| Issue #50 | Lack of events for `setRoutePairAddress`, `setNotFlip`, `removeToken`, `sweep` and `withdraw` |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Functions that affect the status of sensitive variables should emit events as notifications. |
| **Recommendation** | Add events for the above functions. |
| **Resolution** | ACKNOWLEDGED |

## 2.9 ZapAndDeposit

The ZapAndDeposit contract allows users to easily zap into Revault using other tokens than the one the vault requires.

### 2.9.1 Issues & Recommendations

| Issue #51 | `createAmountPayload` could be optimized with `encodePacked` |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Currently the `createAmountPayload` function is highly inefficient. We are unsure why the Solidity method `abi.encodePacked` is not used to cheaply concatenate the byte arrays. |
| **Recommendation** | Consider simply using `abi.encodePacked(_leftCallData, _newData, _rightCallData)` to concatenate the three arrays. |
| **Resolution** | RESOLVED<br>The recommendation has been implemented. |

## 2.10 RevaToken

The RevaToken is a simple ERC20 token with an array of transfer fees:

- treasury fee
- Reva stake fee
- Reva LP stake fee
- burn fee

Apart from the burn fee, the contract owner can freely set the fee destination for the fees. Fees can be adjusted by the owner to a maximum of 1% of the transfer value each. Any wallet in the minter set can mint the tokens.

### 2.10.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `mint`
- `setTreasury`
- `setRevaStakeFeeReceiver`
- `setRevaLpStakeFeeReceiver`
- `setBurnFee`
- `setMinter`

## 2.10.2 Issues & Recommendations

| Issue #52 | Setting any fee address to the zero address will break most functionality |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Description** | In the `transfer` function, the fees are paid to three different configurable fee addresses. Should this be set to the zero address, then transfers will fail and thus break all token transfers. |
| **Recommendation** | To prevent this from ever happening by accident and to limit governance risks, consider adding requirements like<br>`require(_treasury != address(0), "!nonzero");`<br>to the configuration functions. |
| **Resolution** | RESOLVED<br>The recommended non-zero checks have been implemented to all three fee recipients. |

| Issue #53 | Supply limit check can overflow in RevaToken |
|---|---|
| **Severity** | LOW SEVERITY |
| **Location** | Line 28<br>`require(totalSupply() + _amount <= _maxSupply, "max supply");` |
| **Description** | The supply limit check in the contract can overflow since SafeMath is not used.<br>Note that this issue is marked as Low Severity as we expect the `totalSupply` incrementation to still revert correctly. |
| **Recommendation** | Consider using the SafeMath `add` function for the addition. |
| **Resolution** | RESOLVED<br>SafeMath is now used for addition. |

| Issue #54 | Duplicate logic in `transfer` and `transferFrom` |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | The fee logic is added twice in the overridden `transfer` and `transferFrom` method. This is unnecessary and could cause divergences if the code is amended in the future. |
| **Recommendation** | Consider moving the fee logic to the `_transfer` override function and using `super._transfer` to call the raw transfer function. |
| **Resolution** | ACKNOWLEDGED |

| Issue #55 | Governance functionality is broken |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Although there is YAM related delegation code in the token contract which is usually used for governance and voting, the delegation code can be abused as the delegates are not moved during transfers and burns. This allows for double spending attacks on the voting mechanism.<br><br>It should be noted that this mistake is present in pretty much every single farm out there including PancakeSwap and even SushiSwap. |
| **Recommendation** | The broken delegation related code can be removed to reduce the size of the contract. If voting is ever desired, it can still be done through snapshot.org, which is used by many of the larger projects. |
| **Resolution** | RESOLVED<br><br>The governance functionality is removed. |

| Issue #56 | `delegateBySig` can be frontrun and cause denial of service |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Currently if `delegateBySig` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `delegateBySig` transactions in the mempool and execute them before a contract can. The issue with this is that the rest of said contract functionality would be lost as well. This could be a problem in case it would have been executed by a contract that would have rewarded you for your delegation for example. |
| **Recommendation** | Consider taking this behavior into consideration in derivative contracts that use this function. These derivative contracts could for example only execute `delegateBySig` if the current delegation is not yet set correctly. This way the frontrunning will have no effect. |
| **Resolution** | RESOLVED<br>The governance functionality is removed. |

| Issue #57 | `mint`, `setTreasury`, `setRevaStakeFeeReceiver`, `setRevaLpStakeFeeReceiver` and `setBurnFee` can be made `external` |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Functions not used internally can be changed from `public` to `external`. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases. |
| **Recommendation** | Consider making these functions `external`. |
| **Resolution** | RESOLVED |

| Issue #58 | **Lack of events for `setTreasury`, `setRevaStakeFeeReceiver`, `setRevaLpStakeFeeReceiver`, `setBurnFee` and `setMinter`** |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | Functions that affect the status of sensitive variables should emit events as notifications. |
| **Recommendation** | Add events for the aforementioned functions. |
| **Resolution** | RESOLVED |

## 2.11 vRevaToken

The vRevaToken is a non-transferrable governance token used for voting on proposals. The owner can mint and burn this token to and from wallets.

### 2.11.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `mint`
- `burn`

### 2.11.2 Issues & Recommendations

| Issue #59 | `delegateBySig` can be frontrun and cause denial of service |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Currently if `delegateBySig` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `delegateBySig` transactions in the mempool and execute them before a contract can. The issue with this is that the rest of said contract functionality would be lost as well. This could be a problem in case it would have been executed by a contract that would have rewarded you for your delegation for example. |
| **Recommendation** | Consider taking this behavior into consideration in derivative contracts that use this function. These derivative contracts could  for example only execute `delegateBySig` if the current delegation is not yet set correctly. This way the frontrunning will have no effect. |
| **Resolution** | ACKNOWLEDGED |

| Issue #60 | Wrong usage of `require` |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Location** | `require(false, "Can't transfer VREVA");` |
| **Description** | The codebase currently uses require to revert, however, the best-practice is to use the revert keyword directly. |
| **Recommendation** | Consider using revert directly.<br>`revert("revert reason");` |
| **Resolution** | RESOLVED |


PALADIN
BLOCKCHAIN SECURITY

| | | |
|---|---|---|
| **Grace Period** | 14 days | After the delay has expired after queueing a transaction, the administrator can only execute it within the grace period. This is to prevent them from hiding a malicious transaction among much earlier transactions, hoping that it goes unnoticed or buried, which can be executed in the future. |

### 2.12.1 Issues & Recommendations

No issues found.