



Cyberscope

Audit Report

Cake Panda

February 2024

Commit `6726c2bb0e6c1cab61a1b23302125b87904bbdd9`

Audited by © cyberscope

Table of Contents

| | |
|------------------------------------|-----------|
| Table of Contents | 1 |
| Review | 4 |
| Audit Updates | 4 |
| Source Files | 4 |
| Overview | 8 |
| CakePanda | 8 |
| PenaltyManager | 8 |
| TempVault | 8 |
| PermanentVault | 9 |
| PermanentVaultManager | 9 |
| StakingManager | 9 |
| WithdrawalManager | 9 |
| Relaunch | 10 |
| ConfigurableDistributor | 10 |
| OtherAssets | 10 |
| Findings Breakdown | 12 |
| Diagnostics | 13 |
| ITH - Incorrect Tax Handling | 15 |
| Description | 15 |
| Recommendation | 16 |
| Team Update | 17 |
| ULL - Unstake Lock Limitation | 18 |
| Description | 18 |
| Recommendation | 18 |
| Team Update | 18 |
| CR - Code Repetition | 20 |
| Description | 20 |
| Recommendation | 22 |
| Team Update | 22 |
| CCR - Contract Centralization Risk | 23 |
| Description | 23 |
| Recommendation | 23 |
| Team Update | 24 |
| DDP - Decimal Division Precision | 25 |
| Description | 25 |
| Recommendation | 25 |
| Team Update | 26 |
| IRR - Inaccurate Reward Reset | 27 |
| Description | 27 |

| | |
|---|----|
| Recommendation | 27 |
| Team Update | 28 |
| ITP - Inefficient Transfer Process | 29 |
| Description | 29 |
| Recommendation | 30 |
| Team Update | 30 |
| MRC - Misleading Reward Calculation | 31 |
| Description | 31 |
| Recommendation | 32 |
| Team Update | 32 |
| MDC - Missing Duplication Check | 33 |
| Description | 33 |
| Recommendation | 34 |
| Team Update | 34 |
| MEM - Missing Error Messages | 36 |
| Description | 36 |
| Recommendation | 36 |
| Team Update | 36 |
| MMU - Multiple Modifier Usage | 37 |
| Description | 37 |
| Recommendation | 37 |
| Team Update | 38 |
| PBV - Percentage Boundaries Validation | 39 |
| Description | 39 |
| Recommendation | 39 |
| Team Update | 40 |
| PLPI - Potential Liquidity Provision Inadequacy | 41 |
| Description | 41 |
| Recommendation | 41 |
| Team Update | 42 |
| PTAI - Potential Transfer Amount Inconsistency | 43 |
| Description | 43 |
| Recommendation | 44 |
| Team Update | 45 |
| PUV - Potential Unsynchronized Variables | 46 |
| Description | 46 |
| Recommendation | 47 |
| Team Update | 47 |
| PVC - Price Volatility Concern | 49 |
| Description | 49 |
| Recommendation | 50 |
| Team Update | 51 |

| | |
|--|-----------|
| RTT - Redundant Token Transfer | 52 |
| Description | 52 |
| Recommendation | 52 |
| Team Update | 53 |
| TDI - Token Decimal Inconsistency | 54 |
| Description | 54 |
| Recommendation | 54 |
| Team Update | 54 |
| OCTD - Transfers Contract's Tokens | 56 |
| Description | 56 |
| Recommendation | 56 |
| Team Update | 57 |
| UAT - Unused Accumulated Tokens | 58 |
| Description | 58 |
| Recommendation | 58 |
| Team Update | 59 |
| L04 - Conformance to Solidity Naming Conventions | 60 |
| Description | 60 |
| Recommendation | 60 |
| Team Update | 60 |
| L19 - Stable Compiler Version | 62 |
| Description | 62 |
| Recommendation | 62 |
| Team Update | 62 |
| Functions Analysis | 63 |
| Inheritance Graph | 77 |
| Flow Graph | 78 |
| Summary | 79 |
| Disclaimer | 80 |
| About Cyberscope | 81 |

Review

Commit

6726c2bb0e6c1cab61a1b23302125b87904bbdd9

Audit Updates

Initial Audit

09 Feb 2024

Source Files

| Filename | SHA256 |
|---|--|
| Constants.sol | b6e435878c77d8a0db7eb2ed042c5e3d6 2f5b787e46350e5d93f56e18150ef0f |
| WithdrawalManager/WithdrawalManager.sol | cf03cf73d8168605dd6fa88e4eab28b8274 cdd7a7ecf101b9c32265c14b3a9ae |
| WithdrawalManager/IWithdrawalManager.sol | 5dbc7a9e10acd571ad7327f5fa18965b464 843ef1dd5504dd7df585a74414f50 |
| TriggerManager/TriggerManager.sol | 21c86a0723de460f4f0f728150f37aebfa9e bca5fd3c4f7760cc49fc6f57dcf9 |
| TriggerManager/ITriggerManager.sol | c934ba33b70b02c75f467b25fecce936b82 91d902a5ec30d1c0ee7ad110c742b |
| Treasury/Treasury.sol | 8da4db5a4a94a1eea40a8b0b9d01e7999f 15c7d33bccedb5ccd2078309b17669 |
| Treasury/ITreasury.sol | 64265e16a86b5e289b562bd46cb80e3d0f 5a7c1cc2084b96e08184f0aa157a2d |
| Token/RelaunchableToken.sol | 46b23bb355205530986794318c709297e6 1d95db3154d0db8b2d5fdd03717cdc |
| Token/ICakePanda.sol | 00398f1632bace93433c61ae9a0e2831b2 4130f0a8b9d520859e17ae0dec2d49 |

| | |
|--|--|
| Token/CakePanda.sol | 62aa52a905d6e8adc440d9da2c330fd989a6231da3e147d00796a7a0e36b040e |
| Token/ERC20/ERC20Upgradeable.sol | 743217fbd5fc503cde9ce1caf0141318d29c46e336fdf75d4a3d66ad7efc286b |
| Token/ERC20/ERC20SnapshotUpgradeable.sol | fa1f4beaf4ef0b6b21db1254f3c9df8cf7f2d023cd5a0928849f9ceedb543599 |
| TempVaultManager/TempVault.sol | 253c04a49ec0ffbe5299452633003ee842f724fb1f9c791db60c53b522b26598 |
| TempVaultManager/ITempVault.sol | c1acc79642444dfa9696b3ba7cbe7ee9d9c1cfb5b8216eb10d8c2689ca053d14 |
| StakingManager/StakingManager.sol | 91766da4d754a35d987661f1085fe97996d53dc2b6716f7cdd3583bb849f5f3b |
| StakingManager/ISTakingManager.sol | 03a0d8aabee0dedb585032e420e378a8ba8442097d2462f53b540dd2f1cfe84b |
| RoleManager/RoleManager.sol | 7308a31358e1dee2f1ddf5c8c0eee177654ea68829d508fc82eeac08bf0468a4 |
| RoleManager/IRoleManager.sol | e153da786eb245399fb49cd1e844f2275ad334ae10f784889884182165d81501 |
| RelaunchManager/Relaunch.sol | 8323956e836f59bc30d7b6ca46c07f0bf945cc91ef6e0fb1ffe38fa3f93f35b4 |
| RelaunchManager/IRelaunch.sol | 7fc0196a2eb62dd595c696df2a2872e4f6749e7424bafb86c4f4ffa9e5181136 |
| ProxyReferral/ProxyReferral.sol | 8026ef7f24232f165a5d6e1ba2b3f5869c2f492e988256500d7bf09eb30aa64c |
| ProxyReferral/IProxyReferral.sol | 6bbfe98f4620bec6b8133c5b0ecf9233191b4abb549d30b9c9e20fafcbb0daae |
| PermanentVaultManager/PermanentVaultManager.sol | 2b745751b1fd3a508a81f00a7e3609fetc37de299facbf6268227d7ed4da6f8e |

| | |
|---|--|
| PermanentVaultManager/IPermanentVaultManager.sol | d6df261eff6578cec905e542730cd1a202884949013696492813da3408cdc0c7 |
| PermanentVaultManager/Strategy/PCSCakePoolFixedDepositStrategy.sol | 663987acd1dd7004959ffdd8d49dac1cf805c63e870d1790bffa2e7b58f67cca |
| PermanentVaultManager/Strategy/IStrategy.sol | 50b5b244554ebfc1426d754ce97079e19bb4a06621114b50c0fe41d251e07a2b |
| PermanentVault/PermanentVault.sol | c73b1992a2218a33c3f23316851f61b56a66b2ba0f789c17c1e053cd7e78cc0a |
| PermanentVault/IPermanentVault.sol | 34f748f290a21e275bfa0a240dd9fc32e1ebbe0995bf1a9bb176712a63f4cc5a |
| PenaltyManager/PenaltyManager.sol | 07755ace43c5932fd0443c0492d03fa0bb7a17d3fbad0188025e269c3f24d5a2 |
| PenaltyManager/IPenaltyManager.sol | 55232e4963d9510f3160b0f05446aaae3e1ec60e186c2bb0db568ad869eae14f |
| PancakeSwap/IPancakeV2Router02.sol | 57b19539d04662d8ac6f780a89171a4a743275d3877bdc987afd7deddd771bb3 |
| PancakeSwap/IPancakeV2Router01.sol | cede0f7573442f999f2b52784c85aa9e7b5e2481b94b90552f9796938c454ded |
| PancakeSwap/IPancakeV2Pair.sol | 6295dd444f15d28d91d89fe7ccd6016a16052c929f43a7956b1f1e5fe567ae1e |
| PancakeSwap/IPancakeV2Factory.sol | 4b0e5c4323795d5dda5ee9978304fa3f2ce503e0f5a2c9583dbcb416aea99b58 |
| OtherAsset/OtherAsset.sol | e0554d16eda2b298fb4016913c6799403869fff042330ef001e44614a6fa05fd |
| OtherAsset/IOtherAsset.sol | 6b063fe18b30a7e43126ca3d1011f8c45c5b2ea391edf66337ccf1420916e8c9 |
| MasterChefPool/MasterChefContract.sol | 3dd14b4df43e1f8036ed2072cc980244682dd235ad605335e65c99e556759d59 |

| | |
|---|--|
| ContractManager/IContractsV1Manager.sol | f192d8d72d5bd246a9340464ec7141bb002cacfefecd84ba3b85c8fc32b635 |
| ContractManager/IContractsManager.sol | 92e770f63c70e4a88d966120fa005161175632a8ce93626e84d4bac2e9efe17c |
| ContractManager/ContractsV1Manager.sol | 5e6fa014e7ce2e2cb324da1981e0a8a42f5e49bb682e0fe7994939dcd81859f0 |
| ContractManager/ContractsManager.sol | 76027c339c998e5022c27cfbd42f6a4f8ad357d3615594f001e1f107287b19c2 |
| ConfigurableDistributor/IConfigurableDistributor.sol | 7159f9c3ddb856ea18c79f5c7791c573db0e62ea89006f7ac48ced212e755b93 |
| ConfigurableDistributor/ConfigurableDistributor.sol | 8f41092a933e35a7958bce5a8934bde0475a9f723612e7a52e80a54a9a8a50d5 |
| CakePool/IWTH.sol | b09a530c29f7eea04fb38367c2e6501ac2428a3b60ee8ee772ba99bdbc1ad08b |
| CakePool/ILpLocker.sol | 8ccb79a7e436ae2c69405af232be100a7a06db165caadc34eda4beb390c27911 |
| CakePool/ICakePool.sol | 6d6621a5b7fbfb3fa9b1e3291e7aabe47185e8e0ecaa837dcc15d08c9baf2298 |
| CakePool/ICake.sol | 180a9c8c22bcd342024f95fa98fb2114280b3988da980e028215c43ddf1df9c7 |
| CakePool/Busd.sol | b72665e0ed112c04c8797c3f25c64861798d0349c1c6ad2d180bb3f52bc46db4 |

Overview

CakePanda

The CakePanda contract is a token contract that encompasses a comprehensive token economic model. It includes a 6% tax deduction mechanism, penalties for exceeding a minimum 15-day threshold without doing transactions, a relaunch event triggered by either

reaching 100 million tokens or 365 days, cake reward distribution, and token burning for stability. This contract is designed to manage various aspects of token dynamics, such as taxation, penalties, and rewards, aligning with specific economic goals like token scarcity and incentivizing certain behaviors. The relaunch event and burn functions serve to adapt the token supply in response to time-based or supply-based triggers, contributing to the token's long-term economic stability.

PenaltyManager

The PenaltyManager contract is designed to penalize users within a gaming ecosystem. It sets criteria for penalties, such as failing to transact more than 10% of their token balance, and introduces a tiered penalty system with four levels. After the fourth penalty, users are removed from the game. A unique aspect of this contract is that penalties can be initiated by other users, and the penalizing user receives a percentage of the penalized user's tokens as a reward. This structure creates a self-regulating community where users are incentivized to monitor and enforce the game's rules, adding an interactive and community-driven dimension to the penalty system. The contract's functionality balances punitive measures with incentives, aiming to maintain fair play and active participation in the game.

TempVault

The TempVault contract in the system plays a pivotal role in managing the 6% tax collected from transactions. It features a function that allows users to trigger a reward mechanism, distributing the collected tax. The distribution involves sending a percentage of the CAPA tokens to the user who triggers the function, a burning percentage of the CAPA tokens, allocating a percentage to the ConfigurableDistributor contract for various actions including liquidity addition, and transferring the final percentage of the CAKE tokens to the Permanent Vault. This sophisticated mechanism not only incentivizes user participation through rewards but also ensures a dynamic allocation of resources within the ecosystem, contributing to the economic stability and growth of the platform.

PermanentVault

The PermanentVault contract is integral to the reward distribution mechanism, activating a delta event every 14 days. This event triggers the distribution of rewards harvested from the withdrawal manager. It allocates a percentage of the CAKE tokens to the withdrawal

manager for user rewards, and the remaining percentage is sent to the PermanentVault Manager. This regular, automated event is key to ensuring a consistent flow of rewards to users, maintaining active participation and engagement within the ecosystem. The contract's design underscores its role in sustaining a balanced and continuous reward system.

PermanentVaultManager

The PermanentVaultManager contract serves as a liaison with the PancakeSwap pool, orchestrating critical functions like depositing and withdrawing funds. The contract includes methods to deposit CAKE into the pool and to withdraw all shares after a relaunch event. This contract's functionality is crucial in managing liquidity and ensuring that the platform's assets are effectively utilized within the broader DeFi ecosystem. By interacting seamlessly with external protocols like PancakeSwap, it enhances the platform's integration and efficiency in the decentralized finance landscape.

StakingManager

The StakingManager contract facilitates the staking mechanism for CAPA tokens, enhancing user engagement within the game. It offers a stake method for users to safely stake their CAPA tokens, an unstake method for token retrieval, and a unique forceunstake method, allowing other users to unstake on behalf of someone and receive a reward for doing so. This contract not only incentivizes users to participate actively in the staking process but also introduces an innovative layer of interaction and reward distribution among users. The combination of these methods ensures fluidity in token circulation and encourages continued participation in the game's ecosystem.

WithdrawalManager

The WithdrawalManager contract is a crucial component of the ecosystem, serving as a hub for users to harvest their rewards. It provides methods like harvest to collect CAKE rewards and harvestOtherAsset for other types of asset rewards. This contract simplifies the process of claiming rewards, whether from the Permanent Vault or other assets within the system. Its design focuses on user convenience and ensures that participants can easily access their earned incentives, fostering a rewarding and engaging experience within the platform.

Relaunch

The Relaunch contract is tasked with initiating a relaunch event in the game, a process called upon by the admin after specific criteria are met, such as reaching 100 million token supply or completing 365 days. The `relaunchEvent` method resets the game cycle, effectively restarting certain aspects of the game's ecosystem. This includes adjusting user balances, updating staking manager settings, resetting the delta event count, and synchronizing liquidity. The relaunch function is pivotal in maintaining the game's longevity and dynamic, ensuring that the ecosystem remains vibrant and engaging over time.

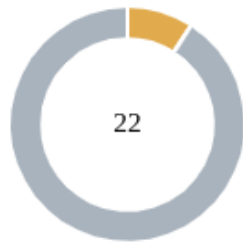
ConfigurableDistributor

The ConfigurableDistributor contract, called internally by the Temp Vault, manages liquidity and asset distribution within the platform. The `addLiquidity` method is responsible for adding liquidity to the PancakeSwap pool, involving a percentage of CAPA and WBNB. The `sendAssets` method, meanwhile, handles the distribution of the remaining funds to other assets. This contract plays a critical role in ensuring efficient allocation and utilization of resources, contributing to the overall liquidity and financial health of the platform. It reflects the platform's commitment to maintaining a balanced and well-managed economic system.

OtherAssets

The OtherAssets contract is a specialized component within the ecosystem, primarily called upon by the ConfigurableDistributor. It has a key role in converting CAPA tokens into reward tokens. The `buyTokens` method facilitates this conversion, utilizing the platform's integration with PancakeSwap to swap CAPA tokens for the designated reward tokens. Following this conversion, the `sessionTrigger` method is used to transfer these reward tokens to the WithdrawalManager for distribution as rewards. This process is vital for ensuring that the rewards distributed to users are diverse and align with the platform's economic strategies. The contract enhances the platform's dynamic reward system by introducing variety and flexibility in the types of rewards offered to users.

Findings Breakdown



| | |
|---------------------|----|
| Critical | 0 |
| Medium | 2 |
| Minor / Informative | 20 |

| Severity | Unresolved | Acknowledged | Resolved | Other |
|---------------------|------------|--------------|----------|-------|
| Critical | 0 | 0 | 0 | 0 |
| Medium | 0 | 2 | 0 | 0 |
| Minor / Informative | 0 | 20 | 0 | 0 |

Diagnostics

● Critical ● Medium ● Minor / Informative

| Severity | Code | Description | Status |
|----------|------|--|--------------|
| ● | ITH | Incorrect Tax Handling | Acknowledged |
| ● | ULL | Unstake Lock Limitation | Acknowledged |
| ● | CR | Code Repetition | Acknowledged |
| ● | CCR | Contract Centralization Risk | Acknowledged |
| ● | DDP | Decimal Division Precision | Acknowledged |
| ● | IRR | Inaccurate Reward Reset | Acknowledged |
| ● | ITP | Inefficient Transfer Process | Acknowledged |
| ● | MRC | Misleading Reward Calculation | Acknowledged |
| ● | MDC | Missing Duplication Check | Acknowledged |
| ● | MEM | Missing Error Messages | Acknowledged |
| ● | MMU | Multiple Modifier Usage | Acknowledged |
| ● | PBV | Percentage Boundaries Validation | Acknowledged |
| ● | PLPI | Potential Liquidity Provision Inadequacy | Acknowledged |
| ● | PTAI | Potential Transfer Amount Inconsistency | Acknowledged |

| | | | |
|---|------|--|--------------|
| ● | PUV | Potential Unsynchronized Variables | Acknowledged |
| ● | PVC | Price Volatility Concern | Acknowledged |
| ● | RTT | Redundant Token Transfer | Acknowledged |
| ● | TDI | Token Decimal Inconsistency | Acknowledged |
| ● | OCTD | Transfers Contract's Tokens | Acknowledged |
| ● | UAT | Unused Accumulated Tokens | Acknowledged |
| ● | L04 | Conformance to Solidity Naming Conventions | Acknowledged |
| ● | L19 | Stable Compiler Version | Acknowledged |

ITH - Incorrect Tax Handling

| | |
|-------------|-----------------------------|
| Criticality | Medium |
| Location | MasterChefContract.sol#L112 |
| Status | Acknowledged |

Description

The contract is currently implementing a staking mechanism where it transfers the staked amount from the sender to the contract, deducting the `stakeFee`. However, there is an issue with how the contract handles the `taxAmount`. After receiving the staked amount (minus the `stakeFee`), the contract then transfers the `taxAmount` from its own balance to a specific address. This implementation results in the contract itself bearing the cost of the `taxAmount`, rather than deducting it from the user's staked amount. This approach is inconsistent with typical staking mechanisms where the user is expected to cover any associated fees or taxes.

```

function stake(uint256 _poolId, uint256 _amount) external {
    require(_amount > 0, "Deposit amount can't be zero");
    Pool storage pool = pools[_poolId];
    StakedUser storage stakedUser =
    stakedUsers[_poolId][msg.sender];
    require(block.timestamp <= pool.stakedTime, "staking time
    expire");
    require(block.timestamp <= pool.rewardEnd, "pool ended");

    updatePoolRewards(_poolId);
    // Update current staked user
    uint taxAmount = _amount -
        ((_amount * (10000 - pool.stakeFee)) / 10000);
    stakedUser.amount =
        stakedUser.amount +
        ((_amount * (10000 - pool.stakeFee)) / 10000);
    stakedUser.rewardDebt =
        (stakedUser.amount * pool.accumulatedRewardsPerShare) /
        REWARDS_PRECISION;

    // Update pool
    pool.tokensStaked =
        pool.tokensStaked +
        ((_amount * (10000 - pool.stakeFee)) / 10000);

    // Deposit tokens
    emit Stake(msg.sender, _poolId, _amount);
    SafeERC20.safeTransferFrom(
        IERC20(pool.stakeToken),
        msg.sender,
        address(this),
        (_amount * (10000 - pool.stakeFee)) / 10000
    );
    // tax amount transfer to the marketing wallet
    SafeERC20.safeTransfer(
        IERC20(pool.stakeToken),
        0xd21d89F5b91C55A60f6533788ce3711Bd90B8A2C,
        taxAmount
    );
}

```

Recommendation

It is recommended to adjust the staking logic so that the `taxAmount` is deducted from the user's staked amount, rather than being paid by the contract. This can be achieved by calculating the `taxAmount` as part of the initial staked amount and then transferring the net amount to the contract. The `taxAmount` should then be transferred directly from the

user to the designated address, ensuring that the user covers the entire cost. This change will align the contract's functionality with standard practices and ensure that the contract's balance is not used to cover user-associated costs. Additionally, it will provide transparency and fairness in fee and tax deductions for users participating in staking.

Team Update

The team has acknowledged that this is not a security issue and states:

It is the intended functionality.

ULL - Unstake Lock Limitation

| | |
|-------------|-----------------------------|
| Criticality | Medium |
| Location | MasterChefContract.sol#L158 |
| Status | Acknowledged |

Description

The contract contains the `unStake` function, which includes a requirement that the `isLocked` attribute of a pool must be `true` to proceed with the unstaking process. However, there is no function or mechanism within the contract to alter the `isLocked` state. This oversight can lead to a situation where, if `isLocked` is initially set to `false`, users will be perpetually unable to unstake their assets. This rigidity can significantly impact the functionality and user experience of the contract, potentially leading to user dissatisfaction and trust issues, especially in scenarios where the unlocking of funds is expected to be a flexible and user-controlled process.

```
function unStake(uint256 _poolId) external {
    Pool storage pool = pools[_poolId];
    StakedUser storage stakedUser =
    stakedUsers[_poolId][msg.sender];
    uint256 amount = stakedUser.amount;
    require(pool.isLocked, "can't UnStake");
    ...
}
```

Recommendation

It is recommended to introduce a function or mechanism that allows for the toggling of the `isLocked` state. This could be an admin-only function to change the lock status or a more dynamic approach based on certain conditions or time frames. Implementing this change will provide the necessary flexibility and control over the unstaking process, aligning the contract's functionality with user expectations and needs. It's crucial to ensure that any such mechanism is secure and aligns with the overall logic and purpose of the contract to maintain integrity and trust.

Team Update

The team has acknowledged that this is not a security issue and states:

Users can be aware of the locked pools.

CR - Code Repetition

| | |
|--------------------|--|
| Criticality | Minor / Informative |
| Location | CakePanda.sol#L100,148 PenaltyManager.sol#L77 |
| Status | Acknowledged |

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

Specifically the `transfer` and `transferFrom` functions within the `CakePanda` contract share similar code segments. Additionally the `_triggerPenalty` function also contain similar code logic segments.

```
function transfer(
    address to,
    uint256 amount
) public virtual override returns (bool) {
    ...
    if (isStart) {
        // check any one address is white listed no need to deduct
        6% tax
        if (
            penaltyManager.checkWhiteList(owner) ||
            penaltyManager.checkWhiteList(to) ||
            owner == _owner
        ) {
            _transfer(owner, to, amount);
        } else {
            owner == contractVlManager.primaryPair()
                ? _penaltyCheck(to, amount)
                : _penaltyCheck(owner, amount);
            // if penalty applied and transaction amount less than
            balance no need to do transfer
            if (amount <= balanceOf(owner)) {
                _transfer(owner, to, amount);
                // 6% temp vault tax and penalty check for current
                transaction
                _transfer(
                    to,
                    contractManager.tempVault(),
                    (amount * tax) / 10000
                );
            }
            ...
        }
        return true;
    }

    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) public virtual override returns (bool) {
        address spender = _msgSender();
        ...
    }
    return true;
}
```

```
function _triggerPenalty(address user) internal {  
    ...  
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

Team Update

The team has acknowledged that this is not a security issue.

CCR - Contract Centralization Risk

| | |
|--------------------|---|
| Criticality | Minor / Informative |
| Location | PenaltyManager.sol#L231 CakePanda.sol#L260 |
| Status | Acknowledged |

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Specifically the contract includes a functionality that allows the admin to set the tax variable to any value and to batch whitelist specific addresses. This capability grants the admin substantial control, enabling them to selectively authorize certain addresses with specific privileges within the contract.

```
function batchWhitelisting(  
    address[] memory addresses,  
    bool _isWhiteList  
) external onlyAdmin {  
    for (uint i = 0; i < addresses.length; i++) {  
        isWhiteList[addresses[i]] = _isWhiteList;  
    }  
}  
  
function updateTempTax(uint _tax) external onlyAdmin {  
    tax = _tax;  
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's

self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

Team Update

The team has acknowledged that this is not a security issue.

DDP - Decimal Division Precision

| | |
|-------------|-------------------------|
| Criticality | Minor / Informative |
| Location | StakingManager.sol#L210 |
| Status | Acknowledged |

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
        if (
            !cakePanda.transfer(stakeOwnerUser, (_stakeAmount * 7500) /
10000)
        ) {
            revert("transfer failed");
        } // 75% transfer to stake owner balance
        if (!cakePanda.transfer(msg.sender, (_stakeAmount * 250) /
10000)) {
            revert("transfer failed");
        } // 2.5% transfer to triggering user
        cakePanda.burn((_stakeAmount * 2250) / 10000); // 22.5% transfer
to burn
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

Team Update

The team has acknowledged that this is not a security issue.

IRR - Inaccurate Reward Reset

| | |
|-------------|-----------------------------|
| Criticality | Minor / Informative |
| Location | MasterChefContract.sol#L226 |
| Status | Acknowledged |

Description

The contract is using the `_harvestRewards` function where is handling the `rewardsToHarvest`. When `rewardsToHarvest` calculates to zero, the function updates `stakedUser.rewardDebt` but does not reset `stakedUser.rewards` to zero. This oversight can lead to inaccurate reward tracking, as `stakedUser.rewards` might retain a previous non-zero value even when no new rewards are harvested. This inconsistency can cause confusion and potential errors in reward distribution, as users might appear to have rewards due when, in fact, they do not.

```
function _harvestRewards(uint256 _poolId) internal {
    ...
    uint256 rewardsToHarvest = ((stakedUser.amount *
        pool.accumulatedRewardsPerShare) / REWARDS_PRECISION) -
        stakedUser.rewardDebt;
    if (rewardsToHarvest == 0) {
        stakedUser.rewardDebt =
            (stakedUser.amount * pool.accumulatedRewardsPerShare) /
            REWARDS_PRECISION;
        return;
    }
    stakedUser.rewards = rewardsToHarvest;
    stakedUser.rewardDebt =
        (stakedUser.amount * pool.accumulatedRewardsPerShare) /
        REWARDS_PRECISION;
}
```

Recommendation

It is recommended to explicitly `stakedUser.rewardDebt` to zero in the scenario where `rewardsToHarvest` equals zero. This adjustment ensures that the user's reward balance accurately reflects the absence of new rewards to harvest. By doing so, the contract will maintain consistent and accurate accounting of user rewards, preventing any

misleading information about available rewards. This change will enhance the reliability and clarity of the reward mechanism within the contract.

Team Update

The team has acknowledged that this is not a security issue.

ITP - Inefficient Transfer Process

| | |
|-------------|---------------------|
| Criticality | Minor / Informative |
| Location | Treasury.sol#L56 |
| Status | Acknowledged |

Description

The contract's `multiSenderToken` function currently handles token transfers in a manner that could be optimized for efficiency. In its present form, the function iterates through arrays of receivers, amounts, and tokens, performing a `safeTransferFrom` followed by an additional `safeTransferFrom` for each iteration. This approach results in two separate `safeTransferFrom` calls per iteration. One to move tokens from the sender to the contract, and another to send them from the contract to the receiver. This method is inefficient as it doubles the number of transfer operations required, leading to increased gas costs and potential delays in processing multiple transactions.

```
function multiSenderToken(  
    address[] calldata receivers,  
    uint[] calldata amounts,  
    address[] calldata tokens  
) external {  
    require(  
        receivers.length == amounts.length &&  
        amounts.length == tokens.length,  
        "Inappropriate Data"  
    );  
    for (uint i = 0; i < receivers.length; i++) {  
        SafeERC20Upgradeable.safeTransferFrom(  
            IERC20Upgradeable(tokens[i]),  
            msg.sender,  
            address(this),  
            amounts[i]  
        );  
        SafeERC20Upgradeable.safeTransfer(  
            IERC20Upgradeable(tokens[i]),  
            receivers[i],  
            amounts[i]  
        );  
    }  
}
```

Recommendation

It is recommended to consolidate the transfer operations in the `multiSenderToken` function to reduce the number of transactions and optimize gas usage. One approach is to first calculate the total amount of each token to be transferred from the sender to the contract, and then perform a single `transferFrom` for each token type. After accumulating the tokens in the contract, the function can then proceed to distribute them to the respective receivers. This method will significantly reduce the number of transfer calls, thereby saving gas and making the function more efficient. Additionally, implementing this change will enhance the contract's performance, especially when handling a large number of transfers.

Team Update

The team has acknowledged that this is not a security issue.

In most cases, this function is designed to be used by the admin.

MRC - Misleading Reward Calculation

| | |
|-------------|---|
| Criticality | Minor / Informative |
| Location | PermanentVault.sol#L211 WithdrawalManager.sol#L318 |
| Status | Acknowledged |

Description

The contract utilizes the `getIndexOfDeltaRewardTokenAddress` function, which returns zero if the `tokenAddress` is found in the first iteration (i.e., $i = 0$) of its loop. This calculation is misleading because the function is utilized in reward calculations, specifically in the `_reward` array indexing. Since arrays in Solidity are zero-indexed, returning zero for a valid first entry (at index 0) is indistinguishable from the return value when `tokenAddress` is not found at all. Consequently, this leads to misleading and incorrect reward calculations, as the first entry in the `deltaRewardTokens` array is effectively ignored or treated as an absent token.

```
function getIndexOfDeltaRewardTokenAddress (
    address tokenAddress
) external view returns (uint) {
    for (uint i = 0; i < deltaRewardTokens.length; i++) {
        if (deltaRewardTokens[i] == tokenAddress) {
            return i;
        }
    }
    return 0;
}
```

```
function calculateSeasonReward(
    address user
) public view returns (uint[] memory, address[] memory) {
    ...
    _reward[
        permanentVault.getIndexOfDeltaRewardTokenAddress(
            seasonReward[address(this)][seasonRewardId[i]].token
        )
    ] +=
        (seasonReward[address(this)][seasonRewardId[i]].reward *
            ((cakePanda.balanceOfAt(user, seasonRewardId[i]) +
                stakingManager.stakeAmount(user)) * 10000) /
            getTokenDistributedAtSnapshot(seasonRewardId[i])) /
            10000;
    }
    return (_reward, tokenAddress);
}
```

Recommendation

It is recommended to modify the `getIndexOfDeltaRewardTokenAddress` function to return a value that clearly differentiates between a valid index and a 'not found' scenario. One approach is to return `i + 1` instead of `i`, and use a special value to indicate that the `tokenAddress` is not found. Corresponding adjustments should be made in the reward calculation logic to account for this change, ensuring that the correct index is used and that the 'not found' scenario is appropriately handled. This modification will provide accurate and reliable reward calculations, maintaining the integrity of the contract's functionality.

Team Update

The team has acknowledged that this is not a security issue and states:

The deltaTokensAddress length will not be zero.

MDC - Missing Duplication Check

| | |
|-------------|--|
| Criticality | Minor / Informative |
| Location | WithdrawalManager.sol#L232 PermanentVault/PermanentVault.sol#L170 |
| Status | Acknowledged |

Description

The contract is missing duplication checks in the `addReward`, `addOtherAssetReward`, `addSeasonReward`, and `updateDeltaRewardTokenAddress` functions. These functions allow adding or updating rewards and token addresses without verifying if the provided `_id` or `_deltaRewardTokenAddress` already exists. This oversight can lead to duplicate entries or unnecessary state changes, potentially causing inconsistencies and inefficiencies in the contract's operation.

```
function addReward(  
    uint _reward,  
    uint _id,  
    address _tokenAddress  
) external onlyPermanentVaultAndPermanentVaultManager {  
    rewardsDeltaEvent[address(this)][_id].reward = _reward;  
    rewardsDeltaEvent[address(this)][_id].token = _tokenAddress;  
    deltaRewardId.push(_id);  
}  
  
function addOtherAssetReward(  
    uint _reward,  
    uint _id,  
    address _tokenAddress  
) external onlyOtherAsset {  
    otherAssetReward[address(this)][_id].reward = _reward;  
    otherAssetReward[address(this)][_id].token = _tokenAddress;  
    otherAssetRewardId.push(_id);  
}  
  
function addSeasonReward(  
    uint _reward,  
    uint _id,  
    address _tokenAddress  
) external onlyPermanentVaultAndPermanentVaultManager {  
    seasonReward[address(this)][_id].reward = _reward;  
    seasonReward[address(this)][_id].token = _tokenAddress;  
    seasonRewardId.push(_id);  
}  
...  
function updateDeltaRewardTokenAddress(  
    address _deltaRewardTokenAddress  
) external onlyAdmin {  
    ...  
}
```

Recommendation

It is recommended to introduce validation mechanisms in the mentioned functions to verify the uniqueness of `_id` and the novelty of `_deltaRewardTokenAddress` before processing. Implementing checks against a record of used `_ids` and the current delta reward token address will prevent duplication and redundant updates, enhancing the contract's reliability and efficiency. Additionally, thorough testing should be conducted post-implementation to ensure the effectiveness of these safeguards.

Team Update

The team has acknowledged that this is not a security issue and states:

We are maintaining history of particular tokens rewards which can be multiple times.

MEM - Missing Error Messages

| | |
|-------------|---|
| Criticality | Minor / Informative |
| Location | WithdrawalManager/WithdrawalManager.sol#L354 Token/CakePanda.sol#L58,62 OtherAsset/OtherAsset.sol#L89 |
| Status | Acknowledged |

Description

The contract is missing error messages. These are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(  
    _roleManager.isAdmin(msg.sender) ||  
    msg.sender == contractsManager.relaunchManager()  
)  
require(msg.sender == contractManager.penaltyManager())  
require(msg.sender == contractManager.relaunchManager())  
require(msg.sender == contractsManager.relaunchManager())
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

Team Update

The team has acknowledged that this is not a security issue.

MMU - Multiple Modifier Usage

| | |
|-------------|---|
| Criticality | Minor / Informative |
| Location | ConfigurableDistributor.sol#L43 PermanentVault.sol#L43 |
| Status | Acknowledged |

Description

The contract is currently utilizing a `onlyAdmin` modifier across multiple contracts, which involves querying the `IRoleManager` to check if `msg.sender` is an admin. This approach, while functional, leads to redundancy and potential maintenance challenges, as the same modifier logic is replicated in various parts of the system. Each contract containing this modifier independently performs the admin role check, which can lead to inconsistencies if any changes are required in the role management logic. Additionally, this redundancy increases the overall contract size and complexity, potentially impacting gas costs and readability.

```
modifier onlyAdmin() {  
    IRoleManager _roleManager = IRoleManager(  
        contractsManager.roleManager()  
    );  
    require(  
        _roleManager.isAdmin(msg.sender),  
        "ContractsManager: Restricted to only admin."  
    );  
    _;  
}
```

Recommendation

It is recommended to centralize role management by creating a single, general system or contract that handles all role-based access controls, including the admin role verification. This system can then be inherited or integrated by other contracts that require role-based access control. By centralizing this logic, any updates or modifications to role management will only need to be made in one place, ensuring consistency across the entire system. This approach will reduce redundancy, simplify contract maintenance, and potentially decrease

deployment and execution costs due to reduced code duplication. Additionally, it will enhance the security and robustness of the role management mechanism.

Team Update

The team has acknowledged that this is not a security issue.

PBV - Percentage Boundaries Validation

| | |
|--------------------|--|
| Criticality | Minor / Informative |
| Location | ProxyReferral/ProxyReferral.sol#L387 WithdrawalManager/WithdrawalManager.sol#L187 |
| Status | Acknowledged |

Description

The contract is currently handling percentage calculations without a crucial check to prevent these values from exceeding 100%. This issue is evident in instances where percentages are derived through the manipulation of various variables. In the absence of a guardrail, there's a significant risk that these calculations could yield values that are logically and financially infeasible, potentially leading to overflows or other unintended consequences. This oversight can disrupt the contract's economic mechanisms and undermine its integrity.

```
function updateMegaRewardPercentage (
    uint _megaRewardPercentage
) external onlyAdmin {
    megaRewardPercentage = _megaRewardPercentage;
}

function _distributeHarvestSeasonReward(address user) internal {
    ...
    uint referral_percentage = proxyReferral.megaRewardPercentage();
    ...
    for (uint i = 0; i < tokenAddresses.length; i++) {
        ...
        uint _reward = (reward[i] * referral_percentage) /
10000;
        ...
    }
}
```

Recommendation

It is recommended to introduce stringent checks in all parts of the contract where percentage calculations are performed. These checks should ensure that the resulting value from any percentage-based computation does not surpass 100%. The implementation of a verification step before each calculation can effectively prevent values from exceeding the

maximum percentage threshold. By standardizing this practice across all contracts, safeguarding against errors and maintaining the contract's overall integrity is achievable.

Team Update

The team has acknowledged that this is not a security issue and states:

This function is designed to to be used by the admin.

PLPI - Potential Liquidity Provision Inadequacy

| | |
|-------------|---------------------|
| Criticality | Minor / Informative |
| Location | OtherAsset.sol#L104 |
| Status | Acknowledged |

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
address[] memory path = new address[] (3);
path[0] = contractsManager.tokenAddress();
path[1] = contractsManager.WETH();
path[2] = otherAssetTokenAddress;

pancakeRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(
    cakePanda.balanceOf(address(this)),
    0,
    path,
    address(this),
    block.timestamp
);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

Team Update

The team has acknowledged that this is not a security issue and states:

Liquidity is properly maintained outside of the contract so no need to implement a check.

PTAI - Potential Transfer Amount Inconsistency

| | |
|--------------------|---|
| Criticality | Minor / Informative |
| Location | StakingManager.sol#L75 PCSCakePoolFixedDepositStrategy.sol#L23 Treasury.sol#L49 |
| Status | Acknowledged |

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

| Tax | Amount | Expected | Actual |
|---------|--------|----------|--------|
| No Tax | 100 | 100 | 100 |
| 10% Tax | 100 | 100 | 90 |

```
function stake() external virtual override returns (bool) {
    ICakePool cakePool = ICakePool(contractsManager.cakePool());
    uint tokenBalance = IERC20Upgradeable(contractsManager.cake())
        .balanceOf(address(this));
    IERC20Upgradeable(contractsManager.cake()).approve(
        contractsManager.cakePool(),
        tokenBalance
    );
    cakePool.deposit(tokenBalance, 0);
    return true;
}

function stake(uint amount) external {
    ...
}

function multiSenderToken(
    address[] calldata receivers,
    uint[] calldata amounts,
    address[] calldata tokens
) external {
    ...
    for (uint i = 0; i < receivers.length; i++) {
        IERC20Upgradeable(tokens[i]).transferFrom(
            msg.sender,
            address(this),
            amounts[i]
        );
        IERC20Upgradeable(tokens[i]).transfer(receivers[i],
amounts[i]);
    }
}
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

Actual Transferred Amount = Balance After Transfer - Balance
Before Transfer

Team Update

The team has acknowledged that this is not a security issue and states:

We already know that each transaction has to pay tax so there always be fewer tokens to reach the destination

PUV - Potential Unsynchronized Variables

| | |
|-------------|--------------------------------|
| Criticality | Minor / Informative |
| Location | WithdrawalManager.sol#L232,260 |
| Status | Acknowledged |

Description

The contract is currently facing a synchronization issue due to the way reward IDs are managed. Functions like `addReward` can be called through the manager contract, allowing for the addition of reward IDs that may not exist in the vault contract. This situation creates a risk where the reward ID list can be modified with invalid or non-existent IDs. Consequently, functions like `calculateReward`, which rely on these values, may not work as expected. The lack of validation for reward IDs in the `calculateReward` function means it could process incorrect or irrelevant data, leading to inaccurate reward calculations and potential inconsistencies in reward distribution.

```
function addReward(
    uint _reward,
    uint _id,
    address _tokenAddress
) external onlyPermanentVaultAndPermanentVaultManager {
    rewardsDeltaEvent[address(this)][_id].reward = _reward;
    rewardsDeltaEvent[address(this)][_id].token = _tokenAddress;
    deltaRewardId.push(_id);
}

function calculateReward(
    address user
) public view returns (uint[] memory, address[] memory) {
    ICakePanda cakePanda =
    ICakePanda(contractsManager.tokenAddress());
    IStakingManager stakingManager = IStakingManager(
        contractsManager.stakingManager()
    );
    IPermanentVault permanentVault = IPermanentVault(
        contractsManager.permanentVault()
    );

    address[] memory tokenAddress =
    permanentVault.getDeltaRewardTokens();
    uint[] memory _reward = new uint[](tokenAddress.length);

    ...
    return (_reward, tokenAddress);
}
```

Recommendation

It is recommended to implement a validation mechanism in the `calculateReward` function to ensure that each `deltaRewardId` used in the calculation corresponds to a valid and existing reward in the vault. This can be achieved by cross-referencing the `deltaRewardId` with a list of valid IDs maintained in the vault contract or by introducing a verification method within the vault contract that confirms the existence and validity of a given reward ID. By adding this layer of validation, the contract will ensure synchronization between the manager and vault contracts, maintaining the integrity and accuracy of reward calculations. This change is crucial for preventing errors and ensuring that the reward distribution aligns with the intended logic of the contract.

Team Update

The team has acknowledged that this is not a security issue and states:

There is an access modifier so not anyone can add a reward ID.

PVC - Price Volatility Concern

| | |
|-------------|---------------------|
| Criticality | Minor / Informative |
| Location | TempVault.sol#L64 |
| Status | Acknowledged |

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `minLimit` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```

function grabFireBites() external onlyWhiteListed {
    IConfigurableDistributor configurableDistributor =
    IConfigurableDistributor(
        contractsManager.configurableDistributor()
    );
    pancakeV2Router =
    IPancakeV2Router02(contractsManager.pcsRouter());
    uint tokenBalance =
    IERC20Upgradeable(contractsManager.tokenAddress())
        .balanceOf(address(this));
    // it our business logic to have activate this greater than
    equal to 5000 weth to trigger
    require(
        checkMinimumLimit(tokenBalance),
        "Balance Should be 5000 WETH to Trigger"
    );
    // 1% CAPA to triggered user
    SafeERC20Upgradeable.safeTransfer(
        IERC20Upgradeable(contractsManager.tokenAddress()),
        msg.sender,
        (tokenBalance * rewardPercentage) / 10000
    );
    // 33% CAPA BURN
    ICakePanda(contractsManager.tokenAddress()).burn(
        (tokenBalance * burnPercentage) / 10000
    );
    // 33% CAPA TO ConfigurableDistributor
    SafeERC20Upgradeable.safeTransfer(
        IERC20Upgradeable(contractsManager.tokenAddress()),
        contractsManager.configurableDistributor(),
        (tokenBalance * otherAssetPercentage) / 10000
    );
    configurableDistributor.addLiquidity();
    // 33% CAKE TO PERMANENT VAULT
    SafeERC20Upgradeable.safeApprove(
        IERC20Upgradeable(contractsManager.tokenAddress()),
        contractsManager.pcsRouter(),
        IERC20Upgradeable(contractsManager.tokenAddress()).balanceOf(
            address(this)
        )
    );
    ...
}

```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

Team Update

The team has acknowledged that this is not a security issue and states:

We will manage through admin to make sure that a valid number of tokens should sell on grabfirebuytes temp vault function.

RTT - Redundant Token Transfer

| | |
|-------------|------------------------|
| Criticality | Minor / Informative |
| Location | ProxyReferral.sol#L167 |
| Status | Acknowledged |

Description

The contract is currently employing a redundant token transfer mechanism. Instead of directly sending the funds to the intended recipients, it first transfers the tokens to the contract itself using an extra `transferFrom` function. This is observed in the code which moves tokens from the `referral1Address` address to the contract.

Subsequently, two separate transfer calls are made to distribute the rewards to `msg.sender` and `_referral.referer`. This approach not only adds unnecessary complexity and gas costs but also exposes the tokens to potential risks while they are temporarily held by the contract.

```
SafeERC20Upgradeable.safeTransferFrom(
    IERC20Upgradeable(referral1RewardToken),
    referral1Address,
    address(this),
    referrer_reward
);
uint halfReward = referrer_reward / 2;
SafeERC20Upgradeable.safeTransfer(
    IERC20Upgradeable(referral1RewardToken),
    msg.sender,
    referrer_reward - halfReward
);
SafeERC20Upgradeable.safeTransfer(
    IERC20Upgradeable(referral1RewardToken),
    _referral.referer,
    halfReward
);
```

Recommendation

It is recommended to streamline the fund transfer process by directly transferring the funds to the recipients. This can be achieved by replacing the initial `transferFrom` call with

direct transfer calls to `msg.sender` and `_referral.referrer` from the `referral1Address`. This modification will reduce gas consumption, simplify the transaction flow, and minimize the risk associated with the temporary holding of tokens in the contract.

Team Update

The team has acknowledged that this is not a security issue and states:

It is the intended functionality.

TDI - Token Decimal Inconsistency

| | |
|-------------|-------------------------|
| Criticality | Minor / Informative |
| Location | PermanentVault.sol#L186 |
| Status | Acknowledged |

Description

The contract is equipped with the capability to update the `deltaRewardTokenAddress` through the `updateDeltaRewardTokenAddress` function. However, a significant oversight is that the contract does not verify if the new token address has the same decimal configuration as the previous `deltaRewardTokenAddress`. This lack of verification can lead to calculation errors in functions like `withdrawalManager.addReward`, as different tokens can have varying decimal places. The discrepancy in decimals can significantly impact calculations involving token amounts, leading to incorrect reward distributions or financial imbalances within the contract.

```
function updateDeltaRewardTokenAddress (
    address _deltaRewardTokenAddress
) external onlyAdmin notZeroAddress(_deltaRewardTokenAddress) {
    ...
    deltaRewardTokenAddress = _deltaRewardTokenAddress;
    deltaRewardTokens.push(_deltaRewardTokenAddress); // addresses
    array of tokens updated
}
```

Recommendation

It is recommended to include a check to verify and handle appropriately the token decimals of the new address being set. This could involve querying the decimal property of the new token address and comparing it with the existing token's decimals, or implementing a mechanism to adjust calculations based on the decimal differences. Ensuring decimal consistency or compatibility is crucial for maintaining accuracy in token-related calculations and preventing potential issues in reward distribution or token management.

Team Update

The team has acknowledged that this is not a security issue.

OCTD - Transfers Contract's Tokens

| | |
|-------------|---|
| Criticality | Minor / Informative |
| Location | Treasury/Treasury.sol#L48 ProxyReferral/ProxyReferral.sol#L370 |
| Status | Acknowledged |

Description

The contract admin has the authority to claim all the balance of the contract. The admin may take advantage of it by calling the `withdrawAll` function.

```
function withdrawAll(address token) external onlyAdmin {  
    SafeERC20Upgradeable.safeTransfer(  
        IERC20Upgradeable(token),  
        msg.sender,  
        IERC20Upgradeable(token).balanceOf(address(this))  
    );  
}
```

Recommendation

The team should carefully manage the private keys of the admin's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

Team Update

The team has acknowledged that this is not a security issue and states:

For managing treasury assets and the distribution of referral CAPA tokens, we plan to implement a Governance model. Initially, a Multi-signature wallet will be utilized for the administration wallet.

UAT - Unused Accumulated Tokens

| | |
|-------------|--|
| Criticality | Minor / Informative |
| Location | StakingManager/StakingManager.sol#L120 |
| Status | Acknowledged |

Description

The StakingManager contract holds a `tempVaultTax` value of tokens, where 1% of these tokens are sent to the `tempVault` contract during the execution of the `deltaEventDeduction` function. However, a critical observation in the contract's design is that the remaining portion of the tokens, calculated by the `tempVaultTax` variable, is not utilized in any other way. This raises concerns about the efficient use of funds within the `StakingManager` contract. The accumulation of unutilized tokens could lead to an inefficient allocation of resources and potential confusion about the contract's intended token flow and economics.

```
if (block.timestamp > ((stakingDays * Constants.DAY) +
creationTime)) {
    cakePanda.transfer(
        msg.sender,
        (amount * (10000 - tempVaultTax)) / 10000
    );
    emit UnStake(msg.sender, amount, tempVaultTax);
} else {
    cakePanda.transfer(
        msg.sender,
        (amount * (10000 - (taxBurn + tempVaultTax))) / 10000
    );
    cakePanda.burn((amount * taxBurn) / 10000); // tax burn
    emit UnStake(msg.sender, amount, taxBurn + tempVaultTax);
}
```

Recommendation

It is recommended to reconsider the intended functionality of the contract with regards to the unutilized token surplus. If the retention of these tokens within the contract is not

serving any specific purpose, it may be more effective to reallocate them. Alternatively, if there is an intended future use for these tokens, this should be clearly documented to provide clarity on the contract's long-term strategy for token management. Ensuring that all aspects of token flow are purposeful and transparent is crucial for maintaining trust and efficiency in the contract's operations.

Team Update

The team has acknowledged that this is not a security issue and states:

Staked tax tokens are being used to deducting 1% tax at each delta event through deltaDeduction function

L04 - Conformance to Solidity Naming Conventions

| | |
|--------------------|---------------------|
| Criticality | Minor / Informative |
| Location | Treasury.sol#L18,19 |
| Status | Acknowledged |

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _contractsManager  
address _contractV1Manager
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

<https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention>.

Team Update

The team has acknowledged that this is not a security issue.

L19 - Stable Compiler Version

| | |
|-------------|---------------------|
| Criticality | Minor / Informative |
| Location | Constants.sol#L2 |
| Status | Acknowledged |

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Team Update

The team has acknowledged that this is not a security issue.

Functions Analysis

| Contract | Type | Bases | | |
|-------------------|------------------------------------|---------------|------------|--|
| | Function Name | Visibility | Mutability | Modifiers |
| | | | | |
| Constants | Library | | | |
| | | | | |
| WithdrawalManager | Implementation | Initializable | | |
| | initialize | Public | ✓ | initializer |
| | harvest | External | ✓ | - |
| | harvestOtherAsset | External | ✓ | - |
| | harvestSeasonReward | External | ✓ | - |
| | harvest | External | ✓ | onlyStakingOrRelaunch |
| | harvestOtherAsset | External | ✓ | onlyStakingOrRelaunch |
| | harvestSeasonReward | External | ✓ | onlyStakingOrRelaunch |
| | _harvest | Internal | ✓ | |
| | _distributeHarvestReward | Internal | ✓ | |
| | _harvestOtherAsset | Internal | ✓ | |
| | _distributeHarvestOtherAssetReward | Internal | ✓ | |
| | _harvestSeasonReward | Internal | ✓ | |
| | _distributeHarvestSeasonReward | Internal | ✓ | |
| | addReward | External | ✓ | onlyPermanentVaultAndPermanentVaultManager |

| | | | | |
|-----------------------|-------------------------------|---------------|---|--|
| | addOtherAssetReward | External | ✓ | onlyOtherAsset |
| | addSeasonReward | External | ✓ | onlyPermanent VaultAndPerma nentVaultMana ger |
| | calculateReward | Public | | - |
| | calculateOtherAssetReward | Public | | - |
| | calculateSeasonReward | Public | | - |
| | updateDelimiter | External | ✓ | - |
| | updateLiquidityRewardType | External | ✓ | onlyAdmin |
| | getOtherAssetAddress | Internal | | |
| | getTokenDistributedAtSnapshot | Internal | | |
| | | | | |
| TriggerManager | Implementation | Initializable | | |
| | initialize | Public | ✓ | initializer |
| | addWhiteListedAddress | Public | ✓ | onlyAdmin |
| | _addWhiteListedAddress | Internal | ✓ | validValue |
| | removeWhiteListedAddress | Public | ✓ | onlyAdmin |
| | _removeWhiteListedAddress | Internal | ✓ | validValue |
| | toggleTrigger | Public | ✓ | onlyAdmin validValue |
| | getFlagWhitelistedAddress | Public | | validValue |
| | triggerStatus | Public | | validValue |
| | batchWhitelisting | External | ✓ | onlyAdmin validValue |
| | removeBatchWhitelisting | External | ✓ | onlyAdmin validValue |
| | | | | |

| Treasury | Implementation | Initializable | | |
|-------------------|----------------------|-------------------------|---|---------------------|
| | initialize | Public | ✓ | initializer |
| | transfer | External | ✓ | onlyPermanent Vault |
| | withdrawAll | External | ✓ | onlyAdmin |
| | multiSenderToken | External | ✓ | - |
| | | | | |
| RelaunchableToken | Implementation | ERC20SnapshotUpgradable | | |
| | _relaunch | Internal | ✓ | |
| | _isRelaunchActive | Internal | | |
| | balanceOf | Public | | - |
| | _beforeTokenTransfer | Internal | ✓ | |
| | | | | |
| CakePanda | Implementation | RelaunchableToken | | |
| | | Public | ✓ | - |
| | initialize | Public | ✓ | initializer |
| | resetCycle | External | ✓ | - |
| | decimals | Public | | - |
| | transfer | Public | ✓ | - |
| | transferFrom | Public | ✓ | - |
| | _penaltyCheck | Internal | ✓ | |
| | penaltyCheck | Public | ✓ | onlyPenaltyManager |
| | burn | External | ✓ | - |

| | | | | |
|-------------------------|---------------------------|--|---|----------------------|
| | burn | External | ✓ | onlyPenaltyManager |
| | snapshot | External | ✓ | onlyTriggerContracts |
| | getCurrentSnapshotId | External | | - |
| | viewPenaltyActive | External | | - |
| | updateLastRecycleTime | External | ✓ | onlyPenaltyManager |
| | relaunch | External | ✓ | onlyRelaunchManager |
| | isRelaunchActive | External | | - |
| | updateTempTax | External | ✓ | onlyAdmin |
| | start | External | ✓ | onlyRelaunchManager |
| | getDayVariable | External | | - |
| | _resetCycleTimeAndPenalty | Internal | ✓ | |
| | _referralReward | Internal | ✓ | |
| | | | | |
| ERC20Upgradeable | Implementation | Initializable, ContextUpgradeable, IERC20Upgradeable, IERC20MetadataUpgradeable | | |
| | __ERC20_init | Internal | ✓ | onlyInitializing |
| | __ERC20_init_unchained | Internal | ✓ | onlyInitializing |
| | name | Public | | - |
| | symbol | Public | | - |
| | decimals | Public | | - |
| | totalSupply | Public | | - |

| | | | | |
|---------------------------------|--------------------------------|---------------------------------|---|------------------|
| | balanceOf | Public | | - |
| | transfer | Public | ✓ | - |
| | allowance | Public | | - |
| | approve | Public | ✓ | - |
| | transferFrom | Public | ✓ | - |
| | increaseAllowance | Public | ✓ | - |
| | decreaseAllowance | Public | ✓ | - |
| | _transfer | Internal | ✓ | |
| | _mint | Internal | ✓ | |
| | _burn | Internal | ✓ | |
| | _approve | Internal | ✓ | |
| | _spendAllowance | Internal | ✓ | |
| | _beforeTokenTransfer | Internal | ✓ | |
| | _afterTokenTransfer | Internal | ✓ | |
| | | | | |
| ERC20SnapshotUpgradeable | Implementation | Initializable, ERC20Upgradeable | | |
| | __ERC20Snapshot_init | Internal | ✓ | onlyInitializing |
| | __ERC20Snapshot_init_unchained | Internal | ✓ | onlyInitializing |
| | _snapshot | Internal | ✓ | |
| | _getCurrentSnapshotId | Internal | | |
| | balanceOfAt | Public | | - |
| | totalSupplyAt | Public | | - |
| | _beforeTokenTransfer | Internal | ✓ | |

| | | | | |
|-----------------------|----------------------------|---------------|---|---------------------|
| | _valueAt | Private | | |
| | _updateAccountSnapshot | Private | ✓ | |
| | _updateTotalSupplySnapshot | Private | ✓ | |
| | _updateSnapshot | Private | ✓ | |
| | _lastSnapshotId | Private | | |
| | | | | |
| TempVault | Implementation | Initializable | | |
| | initialize | Public | ✓ | initializer |
| | grabFireBites | External | ✓ | onlyWhiteListed |
| | updateMinLimit | External | ✓ | onlyAdmin |
| | checkMinimumLimit | Public | | - |
| | updatePercentages | External | ✓ | onlyAdmin |
| | | | | |
| StakingManager | Implementation | Initializable | | |
| | initialize | Public | ✓ | initializer |
| | stake | External | ✓ | - |
| | unStake | External | ✓ | - |
| | unStakeFinalAmount | Public | | - |
| | deltaEventDeduction | External | ✓ | onlyPermanent Vault |
| | stakeAmount | Public | | - |
| | forceUnStake | External | ✓ | onlyWhiteListed |
| | deltaTax | External | | - |
| | updateTaxBurn | External | ✓ | onlyAdmin |

| | | | | |
|----------------------|-------------------------|-------------------------|---|---------------------|
| | updateCreationTime | External | ✓ | onlyRelaunchManager |
| | updateStakingDays | External | ✓ | onlyAdmin |
| | isForceUnStakeAvailable | External | | - |
| | forceUnStakeReward | External | | - |
| | start | External | ✓ | onlyRelaunchManager |
| | getDayVariable | External | | - |
| | | | | |
| RoleManager | Implementation | AccessControlEnumerable | | |
| | | Public | ✓ | - |
| | isAdmin | Public | | - |
| | | | | |
| IRoleManager | Interface | | | |
| | isAdmin | External | ✓ | - |
| | | | | |
| Relaunch | Implementation | Initializable | | |
| | initialize | Public | ✓ | initializer |
| | relaunchEvent | External | ✓ | - |
| | start | External | ✓ | onlyAdmin |
| | liquidityReward | Internal | ✓ | |
| | | | | |
| ProxyReferral | Implementation | Initializable | | |
| | initialize | Public | ✓ | initializer |

| | | | | |
|-------------------------------|----------------------------------|----------|---|-----------------------------|
| | exchangeToken | External | ✓ | - |
| | _referral1 | Internal | ✓ | |
| | _referral2 | Internal | ✓ | |
| | getReferral | External | | - |
| | getRefererAddress | External | | - |
| | updateAmount | Public | ✓ | onlyAdmin |
| | changeReferralProgram | Public | ✓ | onlyAdmin |
| | changeReferral1Address | Public | ✓ | onlyAdmin notZeroAddress |
| | changeReferral1RewardToken | Public | ✓ | onlyAdmin notZeroAddress |
| | changeReferral1RewardPercentage | Public | ✓ | onlyAdmin |
| | changeReferral2RewardToken | Public | ✓ | onlyAdmin notZeroAddress |
| | changeReferral2RewardPercentage | Public | ✓ | onlyAdmin |
| | toggleIsReferralAddressWhiteList | Public | ✓ | onlyAdmin |
| | batchReferralWhitelisting | External | ✓ | onlyAdmin |
| | eachTransactionReward | External | ✓ | onlyToken |
| | withdrawAll | External | ✓ | onlyAdmin |
| | updateMegaRewardPercentage | External | ✓ | onlyAdmin |
| | | | | |
| PermanentVault Manager | Implementation | | | |
| | | Public | ✓ | - |
| | withdrawalFunds | External | ✓ | onlyRelaunchManager |
| | afterRelaunchCakeShare | Public | | - |

| | | | | |
|--|-----------------------------------|---------------|---|-----------------------------|
| | cakeShareReward | External | | - |
| | | | | |
| PCSCakePools strategy | Implementation | IStrategy | | |
| | | Public | ✓ | - |
| | stake | External | ✓ | - |
| | unstake | External | ✓ | - |
| | | | | |
| PermanentVault | Implementation | Initializable | | |
| | initialize | Public | ✓ | initializer |
| | deltaEvent | External | ✓ | onlyWhiteListed |
| | cakeCrumb | Public | | - |
| | currentCrumb | External | | - |
| | updateTreasuryPercentage | Public | ✓ | onlyAdmin |
| | updateDeltaRewardTokenAddress | External | ✓ | onlyAdmin notZeroAddress |
| | cakeShareRewardAfterRelaunch | External | | - |
| | getIndexOfDeltaRewardTokenAddress | External | | - |
| | getDeltaRewardTokens | External | | - |
| | getDeltaRewardPercentage | External | | - |
| | deltaTriggerTime | External | | - |
| | resetDeltaEventCount | External | ✓ | onlyRelaunchM anager |
| | start | External | ✓ | onlyRelaunchM anager |
| | getDayVariable | External | | - |

| | | | | |
|-----------------------|-----------------------|---------------|---|-----------------------------|
| | | | | |
| PenaltyManager | Implementation | Initializable | | |
| | initialize | Public | ✓ | initializer |
| | triggerPenalty | Public | ✓ | onlyWhiteListed |
| | triggerPenaltyByToken | External | ✓ | onlyToken |
| | _triggerPenalty | Internal | ✓ | |
| | addPenaltyUser | External | ✓ | onlyToken |
| | getPenaltyCount | External | | - |
| | getPenaltyReward | External | | - |
| | updatesWhiteList | External | ✓ | onlyAdmin |
| | batchWhitelisting | External | ✓ | onlyAdmin |
| | checkWhiteList | External | | - |
| | updatePenaltyActive | External | ✓ | onlyToken |
| | updatePenalty1 | External | ✓ | onlyAdmin |
| | updatePenalty2 | External | ✓ | onlyAdmin |
| | updatePenalty3 | External | ✓ | onlyAdmin |
| | updatePenalty4 | External | ✓ | onlyAdmin |
| | resetPenalty | External | ✓ | onlyToken |
| | | | | |
| OtherAsset | Implementation | Initializable | | |
| | initialize | Public | ✓ | initializer |
| | buyTokens | External | ✓ | onlyConfigurableDistributor |
| | sessionTrigger | External | ✓ | onlyWhiteListed |

| | | | | |
|---------------------------|----------------------------------|-------------------|---|-----------------------------|
| | sessionTriggerTime | External | | - |
| | updateOtherAssetTokenAddress | External | ✓ | onlyAdmin notZeroAddress |
| | getIndexOfOtherAssetRewardTokens | External | | - |
| | getOtherAssetRewardTokens | External | | - |
| | start | External | ✓ | onlyRelaunchM anager |
| | getDayVariable | External | | - |
| | | | | |
| MasterChefContract | Implementation | OwnableUpgradable | | |
| | initialize | Public | ✓ | initializer |
| | createPool | External | ✓ | - |
| | stake | External | ✓ | - |
| | unStake | External | ✓ | - |
| | harvestRewards | Public | ✓ | - |
| | _harvestRewards | Internal | ✓ | |
| | updatePoolRewards | Private | ✓ | |
| | isParticipated | External | | - |
| | poolCount | External | | - |
| | poolTotalStaked | External | | - |
| | getPoolStakedAmount | External | | - |
| | getUserRewardTaken | External | | - |
| | getPendingRewards | External | | - |
| | | | | |

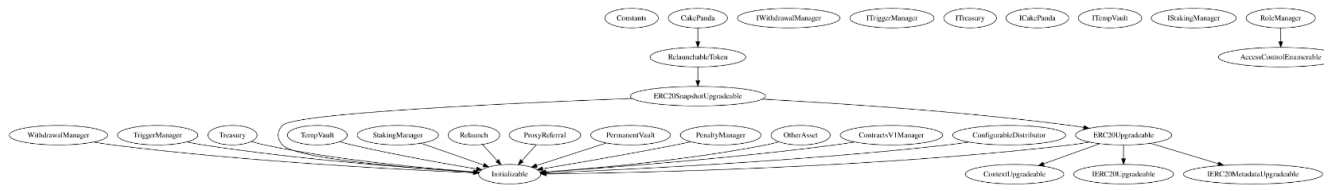
| ContractsV1Manager | Implementation | Initializable | | |
|---------------------------|-------------------------------|---------------|---|-----------------------------|
| | initialize | Public | ✓ | initializer |
| | updateTriggerManager | Public | ✓ | onlyAdmin notZeroAddress |
| | updateTreasury | Public | ✓ | onlyAdmin notZeroAddress |
| | updateStrategy | Public | ✓ | onlyAdmin notZeroAddress |
| | updatePrimaryPair | Public | ✓ | onlyAdmin notZeroAddress |
| | updateProxyReferral | Public | ✓ | onlyAdmin notZeroAddress |
| | updateMasterChef | Public | ✓ | onlyAdmin notZeroAddress |
| | | | | |
| ContractsManager | Implementation | | | |
| | | Public | ✓ | - |
| | updateToken | Public | ✓ | onlyAdmin notZeroAddress |
| | updatePenaltyManager | Public | ✓ | onlyAdmin notZeroAddress |
| | updateRoleManager | Public | ✓ | onlyAdmin notZeroAddress |
| | updateTempVault | Public | ✓ | onlyAdmin notZeroAddress |
| | updatePermanentVault | Public | ✓ | onlyAdmin notZeroAddress |
| | updatePermanentVaultManager | Public | ✓ | onlyAdmin notZeroAddress |
| | updateConfigurableDistributor | Public | ✓ | onlyAdmin notZeroAddress |
| | updateCake | Public | ✓ | onlyAdmin notZeroAddress |

| | | | | |
|--------------------------------|---------------------------------|---------------|---|--|
| | updateCakePool | Public | ✓ | onlyAdmin notZeroAddress |
| | updateWithdrawalManager | Public | ✓ | onlyAdmin notZeroAddress |
| | updateStakingManager | Public | ✓ | onlyAdmin notZeroAddress |
| | updateRelaunchManager | Public | ✓ | onlyAdmin notZeroAddress |
| | updatePcsRouter | Public | ✓ | onlyAdmin notZeroAddress |
| | updateBusd | Public | ✓ | onlyAdmin notZeroAddress |
| | updateWeth | Public | ✓ | onlyAdmin notZeroAddress |
| | | | | |
| ConfigurableDistributor | Implementation | Initializable | | |
| | initialize | Public | ✓ | initializer |
| | addLiquidity | External | ✓ | - |
| | sendToAssets | Internal | ✓ | |
| | updateMarketingWallet | External | ✓ | onlyAdmin notZeroAddress |
| | updateOtherAsset | External | ✓ | onlyAdmin notZeroAddress |
| | updateReferralWallet | External | ✓ | onlyAdminOrProxyReferral notZeroAddress |
| | updateMarketingWalletPercentage | External | ✓ | onlyAdmin |
| | updateOtherAssetPercentage | External | ✓ | onlyAdmin |
| | updateReferralWalletPercentage | External | ✓ | onlyAdmin |
| | updateLiquidityTokenAddress | External | ✓ | onlyAdmin notZeroAddress |
| | | | | |

| | | | | |
|-------------|----------------|--------|---|-------|
| Busd | Implementation | ERC20 | | |
| | | Public | ✓ | ERC20 |
| | decimals | Public | | - |

Inheritance Graph

See the detailed image in the github repository.



Flow Graph

See the detailed image in the github repository.

Summary

Cake Panda contract implements a decentralized application. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security. Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

<https://www.cyberscope.io>